Presentación 1

Arquitectura de Software

Ing. Andrés Sánchez

Adrián Eduardo Ruiz Cerquera

Miguel Ángel Márquez Posso

Juan Felipe Galvis Vargas

Pontificia Universidad Javeriana

Bogotá, Colombia

Octubre de 2025

Tabla de contenidos

Estilo: MVC + Capas

Definición

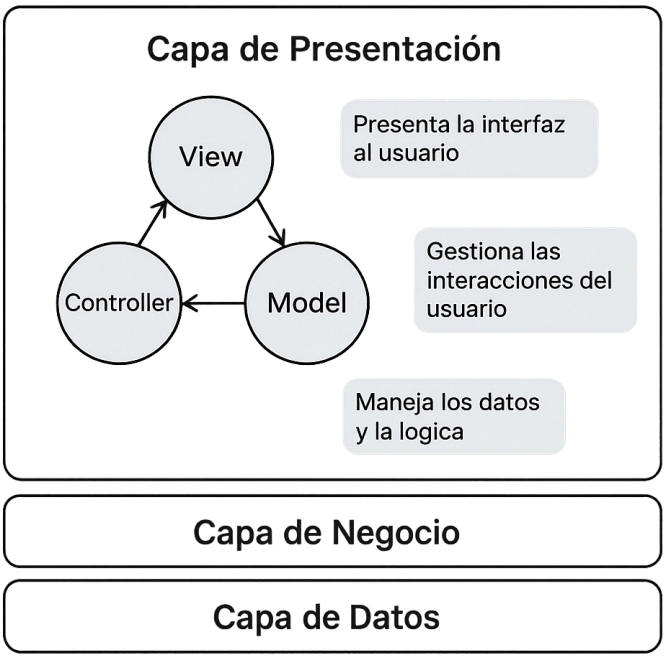
El patrón MVC (Modelo–Vista–Controlador) combinado con la arquitectura en capas es una de las estructuras más utilizadas en el desarrollo de software moderno.

En esta arquitectura, el sistema se organiza en capas horizontales que agrupan responsabilidades similares (por ejemplo: presentación, lógica de negocio y acceso a datos) y, dentro de la capa de presentación, se aplica el patrón MVC para gestionar cómo la interfaz de usuario se comunica con la lógica interna del sistema.

- Modelo (Model): representa los datos y las reglas de negocio. Gestiona la información que la aplicación manipula.
- Vista (View): es la interfaz con la que interactúa el usuario (pantallas, formularios, reportes).
- Controlador (Controller): actúa como intermediario; recibe las acciones del usuario, las interpreta y coordina la comunicación entre la vista y el modelo.

Al combinar el modelo MVC con capas, la estructura se puede ver:

Capa	Función principal	Ejemplo
Capa de Presentación (UI)	Contiene la interfaz y la lógica de interacción (**implementa MVC**).	Vistas web, formularios, controladores JSF.
Capa de Aplicación o Negocio	Define las reglas y procesos de negocio.	Servicios Java, clases de dominio.
Capa de Persistencia o Datos	Administra la comunicación con la base de datos.	DAO, Repositorios, ORM (Hibernate, JPA).
Capa de Infraestructura	Proporciona servicios técnicos transversales.	Configuraciones, seguridad, logging.



Características

Característica	Descripción
Separación de responsabilidades	Cada capa y componente cumple una función específica, evitando mezclar lógica de negocio con presentación.
Modularidad	Permite dividir el desarrollo entre distintos equipos y mantener independencia entre partes del sistema.
Escalabilidad	Facilita el crecimiento de la aplicación agregando nuevas capas, módulos o funciones sin alterar las existentes.
Reutilización	Las capas inferiores (como servicios o persistencia) pueden ser reutilizadas en otras aplicaciones.
Facilidad de prueba	Cada parte (modelo, controlador, servicio, repositorio) puede probarse de forma independiente.
Mantenibilidad	Cambios en una capa no afectan las demás, siempre que se respeten las interfaces y contratos definidos.

Historia y evolución

Años -1970 Origen de MVC

El patrón MVC fue creado en 1979 por Trygve Reenskaug, ingeniero en Xerox PARC, durante el desarrollo del lenguaje Smalltalk-76. Su objetivo era separar la lógica de presentación de la lógica de negocio en las interfaces gráficas, algo innovador para la época. Años -1970 Origen de MVC

Años 1990 - Popularización de MVC

Con el auge de las aplicaciones gráficas en entornos Windows y MacOS, MVC empezó a adoptarse en frameworks de interfaz como Java Swing, Microsoft Foundation Classes (MFC) y Smalltalk-80, donde la interacción usuario–sistema debía organizarse mejor. Años 1990 - Popularización de MVC

Años 2005–2010 – Surgen las variantes de MVC

Patrón	Año	Características
MVP (Model–View–Presenter)	2005.	Separa la lógica de presentación en un *presenter* que actúa entre vista y modelo. Usado en .NET, Android y WinForms.
MVT (Model–View–Template)	2005	Adaptación usada en Django. La "vista" gestiona la lógica y el "template" muestra los datos.
MVA (Model–View–Adapter)	2007	Introduce un adaptador que traduce datos entre vista y modelo (útil en sistemas complejos o multi-plataforma).
MVVM (Model–View–ViewModel)	2005	Propuesto por Microsoft para WPF y Silverlight. Permite "data binding" entre vista y lógica mediante ViewModels. Inspiró frameworks modernos como Angular, React y Vue.
HMVC (Hierarchical MVC)	2008	Divide el sistema en sub-módulos MVC independientes para mejorar la modularidad (usado en PHP y Android).

Años 2010–Actualidad – Integración con arquitecturas modernas

El patrón MVC y sus variantes se combinan hoy con arquitecturas en capas, DDD (Domain-Driven Design) y microservicios. Frameworks como Spring Boot, ASP.NET Core, Angular, React (con MVVM o Flux) y Django mantienen principios del MVC original, adaptados a arquitecturas distribuidas y escalables en la nube.

Ventajas y desventajas

Ventaja	Explicación
Separación clara de responsabilidades	Cada componente cumple una función específica, evitando mezclar lógica y presentación. Esto mejora la organización y reduce errores.
Mantenibilidad	Los cambios en una capa no afectan las demás, lo que facilita actualizaciones y reduce costos de mantenimiento.
Reutilización del código	Servicios y módulos pueden emplearse en otros proyectos, aprovechando mejor el trabajo desarrollado.
Escalabilidad estructurada	La arquitectura permite añadir nuevas funciones o capas sin alterar el sistema existente.
Facilidad para realizar pruebas	Las capas y componentes pueden probarse de forma independiente, lo que simplifica la detección de errores.
Adaptabilidad tecnológica	Permite sustituir tecnologías (por ejemplo, bases de datos o frameworks) sin reescribir toda la aplicación.
Buenas prácticas de diseño	Fomenta principios como *Single Responsibility* y *Separation of Concerns*, generando código más limpio y mantenible.
Comprensión global del sistema	La estructura jerárquica facilita entender el flujo general del software y localizar fallos con rapidez.

Desventaja	Explicación
Complejidad inicial	Requiere un diseño previo detallado, lo que aumenta el tiempo de desarrollo en fases iniciales.
Más código y archivos	Genera estructuras amplias y repetitivas que pueden ser innecesarias en sistemas pequeños.
Posible pérdida de rendimiento	Las múltiples capas introducen pasos adicionales que pueden ralentizar el procesamiento si no se optimiza.
Curva de aprendizaje alta	Los nuevos desarrolladores pueden confundirse al ubicar la lógica en la capa adecuada.
Riesgo de sobrediseño	Aplicar demasiadas capas o abstracciones puede volver el sistema más pesado y difícil de mantener.
Depuración más compleja	Un error puede atravesar varias capas, dificultando su rastreo sin herramientas adecuadas.
Dependencia de frameworks	Cambiar de entorno o versión puede ser costoso si la arquitectura depende fuertemente del framework.
Poco útil en proyectos cortos	En sistemas simples o de vida corta, la inversión en estructura puede no justificarse.

Casos de Uso (Situaciones y/o problemas donde se pueden aplicar)

Situación	Explicación
Aplicaciones empresariales de gran tamaño	Ideal para sistemas con muchas funcionalidades, múltiples usuarios y operaciones complejas (por ejemplo, plataformas bancarias, educativas o ERP).
Aplicaciones web modernas	Permite separar claramente la lógica de presentación (frontend) y la lógica del negocio (backend). Usado en portales, e-commerce, redes sociales, etc.
Sistemas con múltiples interfaces	Cuando un mismo backend debe atender una app web, una app móvil y una API, las capas ayudan a reutilizar lógica y mantener consistencia.
Proyectos de larga duración	Facilita el mantenimiento, actualizaciones y reemplazo de tecnologías sin reescribir todo el sistema.
Entornos académicos o de formación	Es el patrón más usado para enseñar principios de diseño, modularidad y buenas prácticas de desarrollo.
Sistemas con equipos distribuidos	Al dividir el software en capas, varios grupos (frontend, backend, base de datos) pueden trabajar en paralelo sin interferirse.
Aplicaciones con necesidad de pruebas automáticas	Las capas bien definidas permiten crear pruebas unitarias e integradas para cada componente.

Casos de aplicación (Ejemplos y casos de éxito en la industria)

Aplicación / Framework	Descripción	Implementación de MVC + Capas
Spring Boot (Java)	Framework empresarial basado en Spring MVC. Organiza el sistema en controladores, servicios y repositorios.	Usa MVC en la capa de presentación y una arquitectura de 3 capas para backend.
Django (Python)	Framework web con patrón MVT (similar a MVC).	Vista controla la lógica, Template maneja presentación y Modelo gestiona datos.
ASP.NET Core (C#)	Framework de Microsoft para web y APIs REST.	Implementa MVC con separación por capas y soporta inyección de dependencias.
Angular (TypeScript)	Framework frontend basado en MVVM.	Usa componentes (vista), servicios (modelo) y binding bidireccional entre ambos.

React (JavaScript)	Librería frontend declarativa inspirada en MVVM.	La vista se actualiza automáticamente cuando cambian los estados del modelo.
Laravel (PHP)	Framework MVC modular con arquitectura en capas.	Divide el código en controladores, modelos, vistas y servicios reutilizables.
Android (Kotlin/Java)	Usa MVVM y arquitectura en capas (UI, dominio, datos).	Ideal para separar la lógica de presentación del acceso a APIs o BD local.

Backend: Symfony - PHP

Definición

Symfony es un framework de desarrollo web gratuito y de código abierto creado con PHP, que sirve para construir aplicaciones modernas, rápidas y fáciles de mantener. Su objetivo principal es ayudar a los desarrolladores a organizar el código de forma ordenada, evitando repeticiones y errores, y permitiendo que los proyectos crezcan sin volverse complicados.

Symfony está formado por módulos llamados componentes, que pueden usarse juntos o por separado según las necesidades del proyecto. Por ejemplo, un desarrollador puede usar solo el sistema de rutas o el motor de plantillas sin instalar todo el framework. Esto hace que Symfony sea muy flexible y adaptable a diferentes tipos de aplicaciones.

Además, Symfony utiliza el patrón Modelo–Vista–Controlador (MVC), que separa la parte visual (interfaz del usuario), la lógica del programa y los datos. Gracias a esto, el código es más limpio, fácil de entender y de modificar. También incluye herramientas avanzadas como inyección de dependencias, motor de plantillas Twig, sistema de seguridad, gestión de caché y validación de formularios, que agilizan el trabajo del programador.

Características

Característica	Descripción
Modularidad por componentes	Symfony está formado por más de 50 componentes reutilizables (como Routing, Validator, HTTP Foundation o Form). Estos módulos pueden usarse de forma individual o en conjunto, lo que permite construir desde pequeñas funciones hasta aplicaciones completas.
Estructura basada en MVC	Sigue el patrón Modelo–Vista–Controlador, separando los datos, la lógica y la interfaz. Esto mejora la organización del código y facilita el trabajo en equipo (por ejemplo, desarrolladores de backend y frontend pueden trabajar en paralelo).

Motor de plantillas Twig	Usa Twig para crear vistas dinámicas. Este motor permite insertar variables PHP en HTML de forma segura y ordenada, evitando mezclar código con presentación y reduciendo errores de sintaxis o seguridad.
Inyección de dependencias (DI)	Symfony incluye un contenedor de servicios que administra las dependencias de los objetos automáticamente. Esto mejora la modularidad y permite cambiar o sustituir clases sin afectar el resto del sistema.
ORM con Doctrine	Integra Doctrine, una capa de persistencia que facilita trabajar con bases de datos sin escribir SQL directamente. Los datos se manejan como objetos, simplificando el mantenimiento y la portabilidad.
Sistema de Rutas (Routing)	Administra la forma en que las URLs se asocian con los controladores. Permite definir rutas limpias, amigables y seguras, facilitando el SEO y la claridad del flujo de la aplicación.
Seguridad y autenticación integradas	Proporciona módulos de seguridad, control de acceso, cifrado y gestión de usuarios listos para usar. Esto garantiza que las aplicaciones cumplan con buenas prácticas de protección de datos.
Sistema de caché y rendimiento	Implementa mecanismos de caché integrados (HTTP Cache, ESI, Redis, Memcached) para mejorar la velocidad de carga y optimizar el consumo de recursos del servidor.
Flexibilidad y extensibilidad	Los proyectos pueden ampliarse mediante bundles (módulos adicionales). Symfony también permite crear tus propios bundles o integrar otros frameworks como API Platform o EasyAdmin.
Versiones con soporte a largo plazo (LTS)	Symfony publica versiones con soporte extendido (3 años para correcciones y 4 para seguridad), ideal para aplicaciones empresariales que requieren estabilidad prolongada.
Herramienta Symfony Flex	Es un asistente que automatiza la instalación y configuración de paquetes, permitiendo crear proyectos más rápido y mantenerlos actualizados con menos esfuerzo.
Testing integrado	Permite realizar pruebas unitarias y funcionales con PHPUnit y herramientas propias, facilitando la verificación continua de la calidad del software.
Amplia comunidad y documentación	Symfony cuenta con una comunidad activa y documentación muy completa, lo que facilita el aprendizaje, la resolución de problemas y la colaboración entre desarrolladores.

Historia y evolución

Año	Versión / Evento	Descripción y relevancia
2004	**Inicio del proyecto**	Fabien Potencier, fundador de SensioLabs (Francia), comienza a desarrollar Symfony con el objetivo de crear un framework PHP estructurado, profesional y reutilizable.
2005	**Lanzamiento de Symfony 1.0**	Se publica la primera versión oficial como software libre bajo licencia MIT. Adopta el patrón MVC y gana popularidad por su enfoque en buenas prácticas y automatización del desarrollo web.
2007–2010	**Consolidación de Symfony 1.x**	Se fortalece la arquitectura MVC y se integran herramientas como Propel ORM y Doctrine, además del sistema de "plugins" precursor de los actuales bundles. Symfony se adopta ampliamente en Europa.
2011	**Symfony 2.0 (Reescritura completa)**	Marca un antes y un después: introduce el sistema de Bundles, el contenedor de servicios (DI), namespaces, componentes reutilizables y compatibilidad con PHP 5.3+. Sienta las bases de la versión moderna del framework.
2013	**Symfony Components**	Los componentes de Symfony se vuelven independientes, permitiendo su uso fuera del framework. Frameworks como Laravel, Drupal 8+ y Magento 2 adoptan partes de Symfony.
2015	**Symfony 3.0**	Se simplifica la estructura de proyectos, mejora el rendimiento y la configuración mediante autowiring. Se centra en optimizar la experiencia del desarrollador.
2017	**Symfony 4.0 y Symfony Flex**	Introduce un enfoque más ligero y flexible (microkernel). Symfony Flex automatiza la instalación de paquetes y reduce la configuración manual.
2019	**Symfony 5.0**	Aumenta el uso de tipado estricto en PHP, mejora el rendimiento y la seguridad. Se consolida la integración con APIs y microservicios.
2021	**Symfony 6.0**	Moderniza el núcleo con soporte para PHP 8+, mayor eficiencia en la inyección de dependencias y compatibilidad con arquitecturas modernas (Docker, Kubernetes).
2022–2024	**Ecosistema actual**	Symfony se mantiene como un pilar del desarrollo PHP profesional. Grandes proyectos como Drupal, Shopware, OroCRM y Upply utilizan Symfony o sus componentes.

Presente	**Symfony como estándar empresarial**	Reconocido mundialmente por su estabilidad, soporte LTS y comunidad activa. Continúa siendo uno de los frameworks más usados en Europa y América Latina para aplicaciones escalables y seguras.

Ventajas y desventajas

Ventaja	Explicación
Alta modularidad	Symfony está dividido en muchos componentes que pueden usarse por separado o juntos. Esto permite crear desde pequeñas funciones hasta aplicaciones completas sin depender de todo el framework.
Código limpio y mantenible	Su estructura basada en MVC y la inyección de dependencias permite escribir código ordenado y fácil de mantener, lo que reduce errores y facilita el trabajo en equipo.
Reutilización de componentes	Los mismos módulos (como formularios, validaciones o seguridad) pueden reutilizarse en varios proyectos, ahorrando tiempo y esfuerzo.
Comunidad activa y ecosistema sólido	Symfony cuenta con una gran comunidad y miles de extensiones (bundles), lo que asegura soporte, actualizaciones constantes y amplia documentación.
Integración con estándares PHP modernos	Cumple con las normas del PHP-FIG (PSR), garantizando compatibilidad con otras librerías y frameworks del ecosistema PHP.
Seguridad avanzada	Incluye sistemas integrados de autenticación, autorización, encriptación y protección frente a ataques comunes como CSRF o XSS.
Escalabilidad y rendimiento	Gracias a su arquitectura modular, Symfony puede adaptarse a proyectos de cualquier tamaño, desde pequeños portales hasta plataformas empresariales.
Versiones con soporte a largo plazo (LTS)	Ofrece actualizaciones de seguridad y estabilidad por varios años, ideal para empresas que requieren proyectos duraderos y confiables.
Flexibilidad para integraciones	Symfony se integra fácilmente con APIs, microservicios o sistemas externos (como CRMs, ERPs o servicios en la nube).
Herramientas de desarrollo profesional	Ofrece utilidades como la consola de comandos, depuradores, perfiles de rendimiento y Symfony Flex, que facilitan la gestión y configuración de proyectos.

Desventaja	Explicación
Curva de aprendizaje pronunciada	Symfony requiere conocimientos previos de PHP orientado a objetos y conceptos como dependencias, servicios y bundles. Esto puede ser un reto para principiantes.
Complejidad inicial	Configurar un nuevo proyecto puede ser más lento al principio por la cantidad de archivos y estructuras necesarias, aunque esto mejora con Symfony Flex.
Sobrecarga en proyectos pequeños	Para aplicaciones simples o prototipos rápidos, Symfony puede ser más pesado que frameworks ligeros como Laravel o Slim.
Migraciones entre versiones mayores	Las actualizaciones de versión (por ejemplo, de Symfony 4 a 5 o 6) pueden requerir cambios en la estructura del código o bundles utilizados.
Mayor consumo de recursos	Comparado con microframeworks, Symfony puede consumir más memoria y tiempo de carga si no se optimiza correctamente.
Dependencia de bundles externos	Algunos módulos dependen de extensiones creadas por terceros. Si un bundle deja de mantenerse, puede generar incompatibilidades o fallos.
Requiere planificación arquitectónica	Para aprovechar su potencia, es necesario diseñar correctamente las capas y dependencias. Un mal diseño puede volver el sistema rígido o difícil de escalar.
Tiempo de desarrollo inicial más largo	La estructura robusta y modular requiere definir servicios, entidades y configuraciones antes de ver resultados funcionales, lo que puede retrasar las primeras entregas.

Casos de Uso (Situaciones y/o problemas donde se pueden aplicar)

Situación / Escenario	Descripción
Aplicaciones empresariales complejas	Symfony es ideal para sistemas con múltiples módulos (usuarios, roles, facturación, reportes, etc.) porque su arquitectura modular permite mantener el código organizado y fácilmente ampliable.
Plataformas web de larga duración	Gracias a sus versiones con soporte a largo plazo (LTS) y su estabilidad, Symfony es perfecto para proyectos que necesitan mantenimiento continuo durante años.
APIs REST y microservicios	Symfony permite crear APIs estables y seguras usando componentes como API Platform. Su soporte para JSON, controladores y

	autenticación lo hace ideal para conectar aplicaciones móviles o frontends modernos.
Portales institucionales o educativos	Su sistema de roles, formularios y seguridad permite desarrollar portales de universidades, gobiernos o empresas que gestionen usuarios y contenidos con control de acceso.
E-commerce personalizados	En tiendas en línea con lógica de negocio compleja (descuentos, inventarios, reportes, pasarelas de pago), Symfony ofrece control total sobre la arquitectura y permite integrar sistemas externos como ERP o CRM.
Sistemas internos (Intranets)	Muchas empresas utilizan Symfony para crear intranets corporativas seguras, con autenticación de empleados, carga de archivos, gestión documental y reportes.
Plataformas SaaS (Software como servicio)	Symfony soporta multi-tenant (varios clientes en una sola aplicación) y facilita la administración de suscripciones, autenticación y escalabilidad en la nube.
Aplicaciones con altos requerimientos de seguridad	Sus mecanismos integrados de cifrado, validación y control de acceso lo hacen ideal para proyectos donde los datos deben protegerse, como bancos, hospitales o sistemas legales.
Proyectos modulares o distribuidos	Symfony permite desarrollar módulos independientes (bundles) que pueden integrarse fácilmente o compartirse entre varios proyectos. Esto acelera el desarrollo de sistemas grandes y en equipo.
Integración con otros frameworks o CMS	Muchas plataformas como Drupal, Magento o Laravel utilizan componentes de Symfony, lo que permite integrar nuevas funciones o migrar proyectos sin perder compatibilidad.

Casos de aplicación (Ejemplos y casos de éxito en la industria)

Nombre / Empresa	Uso / contexto	Comentarios relevantes
Upply	Plataforma de transporte / logística digital	Decidieron mantenerse con PHP + Symfony a pesar de considerar migrar a lenguajes “más modernos”, debido al ecosistema y experiencia del equipo.
Drupal (versión 8 en adelante)	CMS muy difundido	Drupal integró los componentes de Symfony para mejorar la modularidad y modernidad de su arquitectura.

Grandes portales y sitios	Sitios de contenido, plataformas institucionales	Usan Symfony para manejar tráfico elevado, módulos personalizados, integración con otros sistemas.
E-commerce sofisticados	Tiendas en línea que requieren lógica compleja, personalización, escalabilidad	Symfony es usado cuando el e-commerce debe ser extensible, integrado con otros sistemas (ERP, CRM).
Aplicaciones internas / intranet corporativa	Portales internos, herramientas de gestión empresarial	Symfony es adecuado porque permite organizar funciones complejas con control de seguridad, roles, flujos de negocio.

Frontend: SvelteJS - JavaScript

Definición

SvelteJS es un framework compilador frontend de código abierto creado por Rich Harris, diseñado para construir interfaces de usuario modernas mediante JavaScript. A diferencia de otros frameworks que corren mayor parte del código en el navegador al ejecutarse, Svelte traslada gran parte de ese trabajo a tiempo de compilación, generando código JavaScript optimizado que manipula el DOM directamente.

Cada componente en Svelte se escribe en archivos que mezclan HTML, CSS y lógica JS reactiva de forma concisa. Durante la compilación, Svelte convierte esos componentes en funciones eficientes que actualizan el DOM sin la sobrecarga de un Virtual DOM.

Svelte es altamente liviano y produce paquetes finales muy pequeños, lo cual mejora los tiempos de carga y el rendimiento en el cliente, especialmente en dispositivos más limitados o conexiones lentas.

Características

Característica	Descripción
Compilador en vez de runtime	El trabajo se hace antes de que el usuario vea la página: Svelte compila los componentes a JavaScript puro, eliminando la necesidad de un runtime pesado.
Reactividad declarativa	Variables marcadas como reactivas se actualizan automáticamente cuando cambian, y el DOM se ajusta en consecuencia.

Bundlest pequeño / rendimiento	El tamaño del paquete entregado al cliente es reducido, lo que mejora los tiempos de carga y reduce uso de ancho de banda.
Componentes con estilo integrado	En el mismo archivo .svelte mezclas HTML, CSS y lógica de componente, lo que facilita el desarrollo modular.
Sistema de Stores / estado compartido	Proporciona mecanismos ligeros (writable, readable, derived) para compartir estado entre componentes.
Transiciones y animaciones incorporadas	Svelte tiene soporte directo para animaciones y efectos de transición sin librerías externas.
Extensible con SvelteKit	Para construir aplicaciones completas, páginas múltiples, carga de datos, rutas y SSR, se usa SvelteKit (framework meta).
Mejoras recientes en versión 5	Svelte 5 fue una reescritura interna que introduce "runes" para declarar estados proactivamente, mejora la consistencia y optimiza la arquitectura.

Historia y evolución

Año	Evento clave	Importancia
2016	Primer lanzamiento de Svelte	Rich Harris desarrolla una versión mínima (inspirada en Ractive.js) como compilador para aplicaciones ligeras.
2018	Svelte 2	Ajustes en sintaxis (por ejemplo, usar llaves simples), refinamiento de diseño del framework.
2019	Svelte 3	Rediseño de reactividad: el compilador "instrumenta" las asignaciones para detectar cambios y actualizar UI.
2020-2022	Lanzamiento de SvelteKit	Surge como solución para routing, SSR, generación de páginas, carga de datos y estructura de app completa.
2023	Svelte 4	Mejora del framework, optimización en tamaño, pequeños cambios internos para estabilidad.
2024	Lanzamiento de Svelte 5	Reescritura interna significativa; introducción de "runes" como mecanismo explícito para reactividad; apps más rápidas y consistentes.

Ventajas y desventajas

Ventajas	Descripción
Carga más rápida / mejor rendimiento	Al compilar y eliminar runtime, la aplicación que llega al navegador es más ligera y responde más rápido.
Menor tamaño de bundle	Al no incluir el peso del framework en runtime, el paquete final es más pequeño y eficiente.
Menos complejidad en el navegador	Al delegar trabajo al compilador, el navegador ejecuta código optimizado, no lógica pesada de framework.
Fácil de aprender	Su sintaxis es cercana a HTML/JS, sin mucha abstracción extra, lo que acelera su curva de adopción.
Reactivo de forma natural	Los cambios en variables disparan actualizaciones automáticas de UI sin librerías adicionales.
Animaciones y transiciones integradas	Incluye soporte interno para efectos visuales sin necesidad de librerías externas.
Buen soporte para apps completas con SvelteKit	Facilita rutas, SSR, carga de datos, generación de páginas y aplicaciones robustas.

Desventajas	Explicación
Menor ecosistema que frameworks maduros	Aunque crece rápido, sigue teniendo menos librerías y componentes listos que React o Vue.
Complejidad al escalar	En apps muy grandes puede ser más difícil organizar múltiples componentes y flujos de datos complejos.
Compatibilidad con librerías externas	Algunas librerías hechas para React/Vue pueden no integrarse directamente sin adaptaciones.
Cambio en versiones mayores	Las reescrituras (como el salto a Svelte 5) pueden generar ajustes significativos en código.
Depuración más difícil en algunos casos	Si surgen errores generados por transformaciones del compilador, rastrear la causa puede ser más complejo.
Menos soporte empresarial establecido	Comparado con frameworks veteranos, tiene menor “historial institucional” en algunas grandes empresas.

Casos de Uso (Situaciones y/o problemas donde se pueden aplicar)

Escenario	Uso
Prototipos rápidos / MVPs	La simplicidad y velocidad de desarrollo permiten validar ideas con poco esfuerzo.
Aplicaciones web con alta performance	Ideal cuando se necesita que la UI responda rápido, con poca latencia y animaciones fluidas.
Dispositivos con recursos limitados	En clientes con poca CPU o conexiones lentas, un bundle ligero marca la diferencia.
Dashboards y visualización de datos	La reactividad y animaciones nativas ayudan a actualizar gráficos e interfaces dinámicas eficientemente.
Aplicaciones multipágina o híbridas	Cuando necesitas rutas y carga de datos, SvelteKit complementa perfectamente el framework.
Interfaces interactivas aisladas en apps grandes	Puedes usar Svelte para ciertos módulos o widgets dentro de una aplicación más grande.

Casos de aplicación (Ejemplos y casos de éxito en la industria)

Empresa / Proyecto	Uso / aplicación
Square	Utiliza Svelte para interfaces de productos móviles o páginas interactivas.
Grammarly	Integró Svelte para partes de su editor web, mejorando interactividad y desempeño.
The New York Times	Ha usado Svelte o SvelteKit para algunas secciones interactivas en su sitio web. (Wikipedia)
IKEA / Yahoo / Spotify / Apple	Estas grandes compañías han adoptado Svelte o componentes basados en Svelte para partes de sus productos web. (Wikipedia)
Proyectos de la comunidad	Muchos desarrolladores usan Svelte para portales, dashboards, aplicaciones internas ligeras.

Base de Datos: SQLite

Definición

SQLite es un sistema de gestión de bases de datos relacional (RDBMS) ligero, autónomo y de código abierto. A diferencia de otros motores como MySQL o PostgreSQL, SQLite no requiere un servidor independiente, ya que almacena toda la base de datos en un único archivo físico dentro del sistema. Esto lo convierte en una opción ideal para aplicaciones locales, móviles o embebidas que necesitan almacenamiento estructurado sin depender de un servidor externo.

SQLite implementa casi por completo el estándar SQL-92, permitiendo realizar consultas, uniones, índices y transacciones con alta confiabilidad. A pesar de su simplicidad, ofrece integridad referencial, soporte para triggers y vistas, y manejo de tipos de datos básicos, siendo lo suficientemente potente para la mayoría de los casos de uso cotidianos.

Gracias a su diseño compacto y eficiente, SQLite es ampliamente utilizado en navegadores, dispositivos móviles, IoT y software de escritorio. Es una tecnología clave en entornos donde se necesita persistencia de datos rápida, segura y sin mantenimiento complejo, como en Android, iOS, Firefox, Chrome, o incluso en sistemas operativos y aplicaciones integradas en automóviles.

Características

Característica	Descripción
Base de datos sin servidor	No necesita instalar ni configurar un servidor. La base de datos se almacena en un solo archivo .sqlite o .db.
Librería embebida	SQLite se ejecuta dentro de la misma aplicación como una librería C, eliminando la comunicación cliente-servidor.
Ligera y portátil	Su tamaño es inferior a 1 MB y funciona en casi cualquier sistema operativo sin dependencias adicionales.
Cumple con SQL estándar	Implementa la mayoría de las funciones del estándar SQL (SELECT, JOIN, INDEX, TRANSACTION, etc.).
Transaccional y segura	Soporta transacciones ACID (Atómica, Consistente, Aislada y Duradera), garantizando integridad de datos.
Multiplataforma	Funciona en Windows, Linux, macOS, Android, iOS y sistemas embebidos sin modificaciones.
Alta confiabilidad	Es estable, probado y ampliamente utilizado por millones de aplicaciones en producción.

Modo de concurrencia segura	Maneja múltiples lecturas simultáneas y escritura exclusiva, evitando corrupción del archivo.
Soporte de tipos dinámicos	Usa tipado dinámico (no rígido), lo que otorga flexibilidad en la definición de tablas y columnas.
Mantenimiento casi nulo	No requiere administración compleja ni procesos de respaldo automáticos, simplificando su uso.

Historia y evolución

Año	Evento importante	Descripción y relevancia
2000	Creación de SQLite	Desarrollado por D. Richard Hipp con el objetivo de crear una base de datos pequeña y fácil de integrar.
2001–2004	Primeras versiones estables	Se añade soporte para transacciones y sentencias complejas SQL. Gana popularidad en entornos UNIX y sistemas embebidos.
2005	Adopción por proyectos grandes	Firefox y otros navegadores comienzan a usar SQLite como motor de almacenamiento local.
2008	Integración en Android e iOS	Se convierte en la base de datos por defecto para dispositivos móviles, impulsando su expansión global.
2010–2016	Mejoras en rendimiento	Se optimiza el manejo de concurrencia y la compatibilidad con Unicode y funciones matemáticas.
2019	Versión 3.28+	Implementa mejoras de seguridad y rendimiento en transacciones y consultas.
2024	Uso generalizado	SQLite es hoy el RDBMS más desplegado del mundo, integrado en millones de dispositivos, navegadores y aplicaciones.

Ventajas y desventajas

Ventajas	Explicación
Sin necesidad de servidor	Todo funciona desde un único archivo, eliminando la configuración y los procesos de administración.

Liviana y rápida	Su pequeño tamaño permite una ejecución eficiente, ideal para dispositivos con pocos recursos.
Fácil de usar e integrar	Puede incorporarse directamente en cualquier lenguaje (Python, Java, C, C#, etc.) sin configuraciones extra.
Cumple con estándares SQL	Permite usar consultas complejas y estructuradas, igual que otros RDBMS tradicionales.
Alta estabilidad y confiabilidad	Lleva más de 20 años en producción y es probada por grandes empresas y millones de usuarios.
Portabilidad	Su base de datos puede copiarse o moverse entre sistemas sin pérdida de datos.
Ideal para pruebas y desarrollo	Perfecta para entornos locales o de desarrollo sin necesidad de infraestructura pesada.

Desventajas	Explicación sencilla
No es multiusuario concurrente	Solo permite una escritura a la vez, por lo que no es ideal para sistemas con muchos usuarios simultáneos.
Falta de funciones avanzadas	No incluye herramientas de administración o monitoreo complejas como MySQL o PostgreSQL.
Limitaciones en escalabilidad	Aunque eficiente, no está diseñada para manejar millones de transacciones simultáneas.
Seguridad básica	La protección depende del sistema operativo; no tiene control de usuarios ni cifrado avanzado por defecto.
No apta para entornos distribuidos	No ofrece replicación ni clustering nativo, lo que limita su uso en arquitecturas de gran escala.
Soporte limitado de tipos de datos	Usa tipado dinámico y no maneja tipos personalizados o avanzados como otros motores SQL.

Casos de Uso (Situaciones y/o problemas donde se pueden aplicar)

Escenario	Descripción
Aplicaciones móviles	Android e iOS utilizan SQLite como base de datos local por su ligereza y bajo consumo.
Software de escritorio	Ideal para aplicaciones pequeñas que necesitan guardar información local (por ejemplo, historial, configuración o sesiones).
Proyectos educativos o de prueba	Perfecta para aprender SQL o probar prototipos sin instalar servidores externos.
Aplicaciones embebidas / IoT	Muy usada en dispositivos inteligentes, autos y electrodomésticos, donde se requiere almacenamiento pequeño y confiable.
Navegadores web	Firefox, Chrome y Safari usan SQLite para gestionar historial, cookies y marcadores.
Aplicaciones empresariales ligeras	Usada en sistemas internos que no requieren múltiples usuarios concurrentes.
Copias locales o caché de sistemas grandes	Puede servir como almacenamiento temporal o caché local de una base de datos más grande.

Casos de aplicación (Ejemplos y casos de éxito en la industria)

Proyecto / Empresa	Uso de SQLite	Relevancia
Android OS (Google)	Base de datos predeterminada para todas las aplicaciones móviles.	Millones de apps en Android usan SQLite internamente.
Apple iOS / macOS	Utilizada en Core Data y apps del sistema (Mensajes, Notas, Contactos).	Garantiza almacenamiento local eficiente en todos los dispositivos Apple.
Mozilla Firefox	Almacena historial, contraseñas y cookies.	Proporciona un rendimiento rápido y estable para millones de usuarios diarios.
Google Chrome	Usa múltiples bases SQLite para historial, pestañas y marcadores.	Permite persistencia rápida sin ralentizar el navegador.

Dropbox	Emplea SQLite para manejar sincronización local de archivos antes de subirlos a la nube.	Aumenta la confiabilidad y evita pérdida de datos.
Skype y WhatsApp Desktop	Guardan mensajes y configuraciones localmente usando SQLite.	Permite funcionamiento offline y sincronización posterior.
Tesla / Automotriz	Usan SQLite para registrar datos de sensores e información del sistema a bordo.	Fiabilidad y bajo consumo en entornos embebidos.

Integración: Websocket

Definición

WebSocket es un protocolo de comunicación bidireccional que permite establecer una conexión continua entre un cliente y un servidor. A diferencia del modelo tradicional HTTP, donde el cliente debe enviar una solicitud para obtener una respuesta, WebSocket mantiene un canal abierto de comunicación que permite enviar y recibir datos en tiempo real sin necesidad de recargar la página.

Este protocolo fue diseñado para superar las limitaciones del modelo “request-response” de HTTP, ofreciendo una comunicación más rápida, eficiente y persistente. Una vez que la conexión se establece mediante un “handshake” inicial (negociación a través de HTTP), el canal se actualiza a WebSocket y ambas partes pueden intercambiar información libremente usando un flujo continuo de mensajes.

Gracias a su naturaleza en tiempo real, WebSocket es ideal para aplicaciones que requieren actualizaciones instantáneas, como chats, juegos en línea, paneles de control, sistemas financieros o notificaciones en vivo. Es una de las tecnologías clave detrás del desarrollo de la Web moderna interactiva, junto con APIs como WebRTC y EventSource.

Características

Característica	Descripción sencilla y técnica
Comunicación bidireccional	Permite que tanto el servidor como el cliente envíen datos en cualquier momento, sin necesidad de solicitudes repetidas.
Conexión persistente	Una vez establecida, la conexión permanece abierta hasta que alguna de las partes la cierra, evitando el gasto de crear nuevas conexiones.

Bajo consumo de ancho de banda	Reduce la sobrecarga del protocolo HTTP, ya que los encabezados se envían solo una vez durante el "handshake".
Basado en TCP	Utiliza el protocolo TCP, lo que garantiza confiabilidad, orden y entrega completa de los mensajes.
Soporte para binarios y texto	Puede transmitir tanto datos en formato texto (como JSON) como binario (por ejemplo, imágenes o audio).
Seguridad integrada (wss://)	La versión segura de WebSocket (WSS) cifra la comunicación mediante TLS, igual que HTTPS.
Compatibilidad con navegadores modernos	Está soportado por todos los navegadores principales: Chrome, Firefox, Edge, Safari, Opera, etc.
Interfaz JavaScript simple	El API nativo de WebSocket en JavaScript permite crear conexiones fácilmente con solo unas líneas de código.
Uso en servidores y backend	Compatible con Node.js, Java, Python, C#, Go, PHP, entre otros lenguajes, mediante librerías específicas.
Ideal para datos en tiempo real	Se utiliza ampliamente en chats, notificaciones push, videojuegos y aplicaciones colaborativas.

Historia y evolución

Año	Descripción	Importancia
2008	Creación del concepto por Ian Hickson	Se propuso dentro del proyecto WHATWG como solución a la falta de comunicación bidireccional en la web.
2009	Primer prototipo implementado en navegadores	Se realizaron las primeras pruebas de conexión persistente en Chrome y Firefox.
2011	Publicación del estándar oficial RFC 6455	El IETF publica la especificación formal del protocolo WebSocket, marcando su adopción generalizada.
2013–2015	Soporte en servidores y frameworks	Node.js, Java EE, .NET y otros incorporan librerías oficiales para WebSocket.
2016–2020	Expansión en aplicaciones modernas	Se vuelve esencial para chats, notificaciones, streaming y dashboards en tiempo real.
2021–2024	Integración con nuevas tecnologías	Se combina con WebRTC, GraphQL Subscriptions y microservicios, fortaleciendo su rol en la web moderna.

Ventajas y desventajas

Ventajas	Explicación
Comunicación en tiempo real	Ideal para enviar y recibir datos instantáneamente sin recargar la página.
Reducción de latencia	La conexión persistente elimina el retardo de establecer nuevas solicitudes HTTP.
Ahorro de recursos	Menor consumo de ancho de banda y CPU gracias a la reutilización del canal TCP.
Flexibilidad de datos	Permite transmitir texto o binarios, adaptándose a distintos tipos de aplicaciones.
Integración fácil con JavaScript	Su API es nativa en navegadores, lo que facilita la implementación.
Compatibilidad multiplataforma	Funciona con casi cualquier lenguaje y sistema operativo.
Ideal para sistemas interactivos	Permite experiencias fluidas en juegos, chats o aplicaciones colaborativas.

Desventajas	Explicación
Complejidad en la escalabilidad	Las conexiones persistentes pueden consumir muchos recursos del servidor si hay miles de usuarios conectados.
No apto para todas las aplicaciones	En sistemas simples o con poca interacción en tiempo real, puede ser innecesario.
Mantenimiento de conexión	Requiere manejar reconexiones y detección de errores manualmente en caso de fallos de red.
Compatibilidad con proxies y firewalls	Algunos entornos bloquean conexiones WebSocket, lo que puede requerir configuraciones adicionales.
Falta de herramientas de depuración avanzadas	Aunque existen, son menos maduras que las de HTTP estándar.
Posibles riesgos de seguridad	Si no se usa WSS o validaciones, puede ser vulnerable a ataques de inyección o denegación de servicio.

Casos de Uso (Situaciones y/o problemas donde se pueden aplicar)

Escenario	Descripción
Chats en tiempo real	Los mensajes se envían y reciben instantáneamente sin recargar la página.
Juegos en línea	Permite la comunicación continua entre jugadores y servidores sin interrupciones.
Notificaciones en vivo	Ideal para alertas, actualizaciones de estado o mensajes emergentes en aplicaciones web.
Paneles de monitoreo (dashboards)	Muestra datos actualizados constantemente, como sensores o métricas de sistemas.
Aplicaciones financieras	Facilita la actualización de cotizaciones o movimientos en tiempo real.
Aplicaciones colaborativas	Permite a varios usuarios editar o interactuar simultáneamente (como Google Docs).
Transmisiones en vivo / streaming	Optimiza la entrega continua de datos de audio o video.
IoT	Conecta dispositivos inteligentes para enviar datos y recibir comandos de manera instantánea.

Casos de aplicación (Ejemplos y casos de éxito en la industria)

Proyecto / Empresa	Uso de WebSocket	Relevancia o beneficio
Slack	Comunicación en tiempo real entre usuarios y canales.	Permite mensajería instantánea con sincronización en todos los dispositivos.
Trello	Actualiza tableros de tareas sin recargar la página.	Ofrece colaboración fluida entre equipos.
Binance / Coinbase	Transmisión en vivo de precios y órdenes de mercado.	Reduce la latencia en datos financieros.
Facebook Messenger / WhatsApp Web	Mantiene las conversaciones actualizadas en tiempo real.	Brinda experiencia continua sin recargas.

Google Docs	Permite edición colaborativa en simultáneo.	Varios usuarios pueden trabajar sobre el mismo documento sin conflictos.
Spotify Web Player	Usa WebSocket para sincronizar canciones y estados de reproducción.	Garantiza actualización instantánea entre cliente y servidor.
TradingView	Envía gráficos y valores financieros en vivo.	Imprescindible para análisis bursátil en tiempo real.

Relación entre los temas asignados.

¿Qué tan común es el stack designado?

El conjunto de tecnologías analizadas MVC + Capas, Symfony (PHP), SvelteJS (JavaScript), SQLite y WebSocket forma un stack de desarrollo web completo, que combina herramientas tradicionales con tecnologías modernas orientadas a la eficiencia y la interactividad. Aunque no es el stack más popular a nivel mundial, su composición es sólida, equilibrada y se usa ampliamente en proyectos empresariales, académicos y de software a medida.

- Symfony (PHP) y la arquitectura MVC + Capas representan una base estable, estructurada y probada por años en entornos empresariales y educativos.
- SvelteJS, en cambio, introduce una capa moderna y reactiva en la interfaz, ofreciendo mayor rendimiento, tiempos de carga mínimos y menor complejidad que frameworks tradicionales como React o Angular.
- Este equilibrio permite desarrollar aplicaciones actuales y mantenibles, sin sacrificar robustez ni simplicidad.
- Symfony y SQLite son tecnologías altamente comunes en entornos reales, especialmente en Europa y Latinoamérica, donde PHP continúa siendo uno de los lenguajes más usados para backend.
- WebSocket se ha convertido en un estándar para aplicaciones en tiempo real, adoptado por plataformas de mensajería, finanzas y colaboración.
- SvelteJS, aunque menos masivo que React o Vue, crece rápidamente gracias a su eficiencia y facilidad de aprendizaje.
- En conjunto, este stack no busca popularidad, sino eficiencia y aplicabilidad práctica en contextos donde la estabilidad y el rendimiento son más valiosos que la tendencia.

El stack compuesto por MVC + Symfony + SvelteJS + SQLite + WebSocket es poco común en grandes corporaciones, pero muy utilizado en entornos académicos, startups y proyectos personalizados. Su fuerza radica en la eficiencia, modularidad y facilidad de integración, más que en la tendencia.

Matriz de análisis de Principios SOLID vs Temas

Principio	MVC + Capas	Symfony – PHP	SvelteJS – JS	SQLite	WebSocket
S – Single Responsibility	Cada capa asume un único rol (presentación, aplicación, datos) y dentro de presentación, MVC separa Vista/Controlador/Modelo; evita clases 'Dios'.	Controladores delgados y Servicios/Repositorios/Forms con tareas acotadas; DI permite aislar lógica y pruebas.	SRP depende de dividir componentes y extraer lógica a stores/servicios; los componentes tienden a crecer si no se segmentan.	Capa DAO/Repositorio encapsula exclusivamente persistencia; el dominio no conoce SQL ni detalles de archivo.	Conviene separar conexión, protocolo de mensajes y handlers por tópico; si se centraliza todo en un único 'socket service' se viola SRP.
O – Open/Closed	Se agregan casos de uso o capas sin modificar las existentes mediante interfaces/puertos; extensible por composición.	Extensión por eventos/listeners y decoradores de servicios; nuevos bundles añaden capacidades sin tocar código base.	Extensión vía props/slots/stores; si el contrato del componente no está claro, futuras extensiones fuerzan cambios internos.	Esquemas estables se amplían con vistas/queries/DAOs; cambios de modelo pueden requerir migraciones que modifican estructuras.	Nuevos tipos de mensajes/eventos se agregan sin romper los existentes si el protocolo está versionado.
L – Liskov Substitution	Servicios/Repositorios con interfaces permiten intercambiar implementaciones (p.ej., mock vs real) sin alterar clientes.	Contratos definidos por interfaces y tipos; DI inyecta sustitutos (mocks/stubs) manteniendo el comportamiento esperado.	Sustitución depende de respetar contratos de props/eventos; supuestos implícitos de estado rompen LSP.	Backends de almacenamiento pueden cambiarse si el repositorio mantiene el mismo contrato.	Se puede sustituir el transporte (WS↔SSE/HTTP polling) si la interfaz de mensajería se abstrae correctamente.
I – Interface Segregation	Capas exponen interfaces específicas (puertos de aplicación); los controladores no dependen de detalles de persistencia.	Servicios y Repositorios con interfaces pequeñas y enfocadas; validadores/forms desacoplados.	Segmentar props/eventos mínimos por componente y stores especializados; evitar componentes 'omnívoros'.	Repositorios con métodos focalizados por agregado; evitar 'mega DAO' con demasiadas operaciones.	Separar canales/rooms y handlers por tópico; una interfaz gigante de eventos viola ISP.
D – Dependency Inversion	Capas altas dependen de abstracciones (puertos/servicios); infraestructura se inyecta como implementación concreta.	Contenedor de DI primero: controladores dependen de interfaces; infraestructura se resuelve por configuración.	Invertir dependencias moviendo fetch/WS a servicios y proveyéndolos a componentes; sin esto, la UI queda acoplada.	El dominio depende de repositorios abstractos; la implementación SQLite vive en infraestructura.	UI/dominio dependen de una abstracción de mensajería; el adaptador concreto (WS) se inyecta.

Matriz de análisis de Atributos de Calidad vs Temas

Atributo de Calidad	MVC + Capas	Symfony – PHP	SvelteJS – JavaScript	SQLite	WebSocket
Adecuación funcional	Estructura bien definida que asegura que cada capa cumpla su función, evitando redundancias.	Cumple ampliamente por su modularidad y componentes reutilizables.	Proporciona una interfaz reactiva que mejora la precisión funcional de la UI.	Base de datos estable y confiable con soporte completo de SQL.	Permite funcionalidades en tiempo real (mensajería, notificaciones).
Rendimiento / Eficiencia de desempeño	Reduce complejidad y optimiza llamadas, depende de implementación.	Alto rendimiento con caché y optimización, requiere configuración avanzada.	Genera código JS optimizado en compilación, excelente rendimiento.	Acceso directo a disco, muy rápido en entornos locales.	Transmisión continua y eficiente, mínima latencia.
Mantenibilidad	Estructura clara que facilita lectura y modificación del código.	Alta mantenibilidad con DI, bundles y documentación extensa.	Componentes aislados facilitan mantenimiento, aunque comunidad es pequeña.	Liviano y de bajo mantenimiento, pero limitado para proyectos grandes.	Requiere modularizar eventos para evitar código complejo.
Escalabilidad	Escalable dividiendo capas, aunque requiere refactorizaciones.	Altamente escalable por su arquitectura modular.	Escalable en frontend si se segmenta correctamente.	Limitada a pocos usuarios, no apta para grandes volúmenes.	Escalable con clusters, pero complejo de gestionar.
Seguridad	Permite filtros y validaciones en cada capa; adaptable a políticas de seguridad.	Sistema robusto de autenticación, CSRF y encriptación.	Depende del backend, pero protege frente a XSS.	Seguridad básica; depende del sistema operativo.	Cifrado TLS (WSS) y validación de origen necesarias para proteger datos.
Compatibilidad / Interoperabilidad	Fácil de integrar con otros sistemas mediante APIs.	Cumple PSR y estándares PHP-FIG; alta interoperabilidad.	Compatible con cualquier backend REST o WebSocket.	Compatible con múltiples lenguajes mediante SQL estándar.	Intercambia datos entre distintos lenguajes y frameworks.
Usabilidad	Fácil de entender y mantener; estructura clara para nuevos desarrolladores.	Herramientas CLI y profiler mejoran la experiencia del desarrollador.	Excelente experiencia de usuario final con UI fluida y reactiva.	No afecta la interfaz, pero mejora confiabilidad de datos.	Permite interacciones instantáneas y fluidas sin recarga.
Portabilidad	Diseño modular facilita migrar módulos a otros entornos.	Altamente portable en múltiples servidores y sistemas.	Compila a JS estándar compatible con cualquier navegador.	Extremadamente portable; un solo archivo ejecutable en cualquier SO.	Amplio soporte en navegadores y servidores modernos.

Matriz de análisis de Tácticas vs Temas

Táctica / Tema	MVC + Capas	Symfony – PHP	SvelteJS – JavaScript	SQLite	WebSocket
Separación de responsabilidades (mantenibilidad)	Divide claramente la aplicación en capas con roles específicos, evitando dependencias cruzadas.	Refuerza esta separación mediante controladores, servicios y entidades.	Logra separación si los componentes son pequeños y la lógica se extrae a stores.	Permite aislar la capa de persistencia mediante DAOs o repositorios.	Requiere modularizar eventos y listeners para mantener claridad en la lógica.
Uso de caché (rendimiento)	Puede implementar caché a nivel de capa o controlador para reducir tiempos de respuesta.	Ofrece sistemas de caché HTTP y resultados de consultas integrados.	Usa el renderizado reactivo, lo que reduce el trabajo del DOM.	Caché implícito en consultas frecuentes, aunque no nativo.	Reutiliza conexiones persistentes, minimizando overhead en transmisión.
Control de concurrencia (consistencia / rendimiento)	Facilita sincronización por capas; cada capa gestiona su dominio.	Gestiona concurrencia mediante bloqueos de base de datos y control transaccional.	Controla estados compartidos con stores o reactive variables.	Soporta lectura concurrente, escritura exclusiva; ideal para baja concurrencia.	Gestiona múltiples conexiones simultáneas mediante canales o tópicos.
Autenticación y autorización (seguridad)	Se ubica en capa de aplicación para separar lógica de acceso.	Posee componentes nativos de autenticación, roles y cifrado.	Depende del backend, pero puede integrar validación JWT o tokens.	No incluye seguridad interna; depende del sistema que la use.	Usa verificación de origen y tokens para conexiones seguras.
Validación de entrada (seguridad / confiabilidad)	Cada capa puede validar datos antes de pasarlos a la siguiente.	Incluye validadores automáticos y constraints en formularios.	Validación en tiempo real en formularios reactivos.	No valida por sí mismo; validación ocurre en el cliente o aplicación.	Validación de mensajes en el servidor para evitar inyecciones.
Gestión de errores y excepciones (confiabilidad)	Aísla errores por capa para evitar propagación no controlada.	Manejo centralizado con try-catch y controladores de error HTTP.	Sistema de “boundaries” para manejar errores sin afectar el render.	Proporciona códigos de error claros en operaciones SQL.	Usa códigos y eventos para detectar y manejar desconexiones.
Monitorización y logging (disponibilidad / mantenimiento)	Puede implementar logging por capa, facilitando trazabilidad.	Framework de logging integrado y compatible con Monolog.	Logs en consola y herramientas dev integradas.	Registra errores y transacciones en archivos locales.	Permite registrar eventos y estados de conexión en tiempo real.

Escalado modular (escalabilidad)	Cada capa puede escalarse independientemente según la carga.	Alta modularidad con bundles y microservicios.	Escala por componentes y renderizado selectivo.	Limitado: solo puede escalar verticalmente.	Escala mediante balanceo de conexiones y clústeres de servidores.
Transacciones atómicas (consistencia)	Posible a través de la capa de negocio con gestión unificada.	Implementa transacciones ACID nativas con ORM Doctrine.	No aplica directamente; depende del backend.	Soporta transacciones ACID completas, incluso embebido.	Las operaciones no son transaccionales; requieren control en el backend.
Comunicación asincrónica (rendimiento / disponibilidad)	Se logra entre capas mediante colas o servicios desacoplados.	Puede integrarse con colas (RabbitMQ, Kafka) para tareas asíncronas.	Usa Promises y eventos para tareas no bloqueantes.	No tiene mecanismos nativos; opera de forma síncrona.	Nativo: comunicación bidireccional en tiempo real asincrónica.

Matriz de análisis de Patrones vs Temas

Patrón	**MVC + Capas**	**Symfony – PHP**	**SvelteJS – JavaScript**	**SQLite**	**WebSocket**
MVC (Model–View–Controller)	Patrón base: separa la lógica, la vista y los datos, logrando modularidad y mantenibilidad.	Implementado nativamente con controladores, vistas Twig y entidades Doctrine.	No sigue MVC puro, pero usa un modelo reactivo (MVU/MVVM) para separar UI y estado.	No aplica directamente, pero puede participar como modelo de datos.	Se integra como backend MVC que usa WS como canal de comunicación.
Arquitectura en Capas (N-Tier)	Núcleo del estilo: organiza el sistema en presentación, aplicación, dominio y datos.	Adoptada totalmente: Controller → Service → Repository → DB.	Adaptada en frontend con capas de componentes, servicios y red.	Representa la capa de persistencia.	Se implementa como capa de transporte dentro del backend.
Microkernel (Plug-in Architecture)	Facilita ampliaciones por módulos o servicios sin alterar el núcleo.	Soportado por bundles que funcionan como plugins independientes.	Soporta modularidad por componentes y librerías externas.	No aplica; SQLite es un motor compacto.	Puede emplearse para módulos de protocolo o handlers.
Event-Driven Architecture (EDA)	Se usa entre capas para emitir y manejar eventos internos.	Symfony posee EventDispatcher y suscriptores para procesos asíncronos.	Nativo: los componentes reaccionan a eventos del DOM o stores.	Se integra fácilmente en un sistema EDA mayor.	Es el núcleo: los mensajes WS funcionan como eventos distribuidos.
CQRS (Command	Aplica separando controladores o	Doctrine y controladores	Stores para lectura y acciones	Puede actuar como modelo de	WS separa comandos

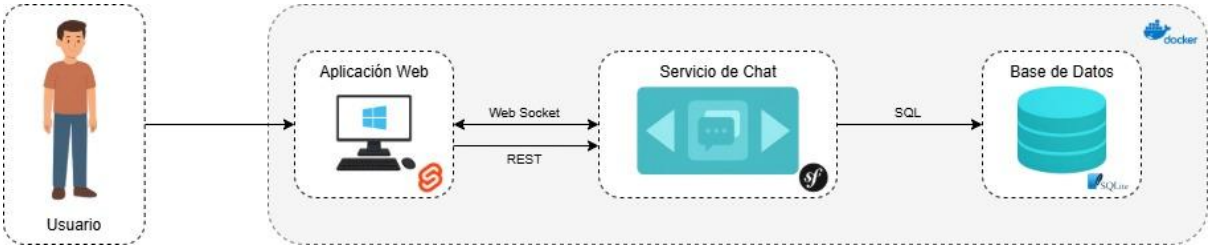
Query Responsibility Segregation)	servicios de lectura y escritura.	permiten aplicar CQRS.	para escritura implementan este patrón naturalmente.	lectura local sincronizado.	(escritura) y suscripciones (lectura).
Strategy	Define algoritmos intercambiables por capa (validación, seguridad, lógica de negocio).	Usado para cacheo, serialización, y políticas de autenticación.	Estrategias para renderización o gestión de datos.	Estrategias de consultas según carga o índice.	Estrategias para reconexión o retransmisión de mensajes.
Factory / Abstract Factory	Crea instancias de servicios y controladores de forma flexible.	Utilizado en creación de servicios y entidades Doctrine.	Factories para componentes, stores o servicios reutilizables.	Fábricas de conexiones o consultas parametrizadas.	Fábricas de sockets o handlers por canal.
Adapter	Conecta capas de aplicación con sistemas externos.	Adaptadores para correo, APIs o servicios externos.	Adaptadores de API/WS para desacoplar la lógica de red.	Adaptadores que unifican acceso a SQLite.	Adaptador WS o SSE bajo interfaz común de mensajería.
Facade	Simplifica el uso de subsistemas complejos desde capas superiores.	Services o managers centralizan acceso a módulos del framework.	Servicios que agrupan lógica y llamadas a APIs.	Fachada de persistencia para manejo de conexiones y consultas.	Fachada de tiempo real para enviar, recibir y cerrar conexiones.
Observer / Publish–Subscribe	Capas o módulos se comunican mediante eventos.	EventDispatcher implementa el patrón; listeners manejan eventos.	Patrón base en Svelte: stores y eventos reactivos.	Limitado a nivel de aplicación.	Nativo: WS usa subscripciones y emisión de eventos.
Mediator	Coordina interacción entre capas sin acoplamiento directo.	Mediadores entre controladores, servicios y eventos.	Coordinadores que gestionan comunicación entre componentes.	Mediador entre consultas y módulos.	Mediador entre múltiples conexiones WS.
Command	Encapsula acciones y facilita auditoría o ejecución diferida.	Comandos del CLI y patrón CommandBus.	Acciones de usuario o eventos disparan comandos.	Transacciones SQL funcionan como comandos.	Mensajes WS actúan como comandos distribuidos.
Event Sourcing	Guarda eventos que reconstruyen el estado del sistema.	Compatible con Event Store externo y sistema de eventos Symfony.	Permite depuración y trazabilidad a partir de eventos.	Puede almacenar registros append-only.	WS distribuye eventos en tiempo real como fuente de verdad.

Matriz de análisis de Mercado Laboral vs Temas

Tema / Tecnología	Demanda laboral actual	Fortalezas en el mercado	Desafíos / Debilidades	Comentario contextual
MVC + Capas (arquitectura tradicional)	Alta demanda de desarrolladores backend estructurados (PHP, .NET, Java)	Las empresas buscan sistemas bien organizados y mantenibles	A veces se les exige conocimiento de frameworks modernos, microservicios o arquitecturas más flexibles	MVC + Capas como fundamento sigue siendo valorado en empresas tradicionales
Symfony – PHP	Muy alta demanda para desarrolladores PHP con Symfony en Colombia. Por ejemplo, "Desarrollador Symfony PHP" tiene decenas de ofertas en Computrabajo.	Buen salario para senior, opciones remotas, empresas de software nacionales buscan experiencia Symfony	Competencia fuerte; para niveles junior exige buenos proyectos previos o portafolio	Symfony es uno de los frameworks PHP más requeridos en Colombia
SvelteJS – JavaScript	Demanda moderada, especialmente en proyectos frontend modernos y startups	Ventaja competitiva: pocos desarrolladores lo dominan, lo que permite destacarse	Menor cantidad de ofertas comparado con React/Vue; colectivos más pequeños	En roles frontend podrían pedir "JavaScript avanzado / moderno / frameworks modernos" donde Svelte es una opción
SQLite	Poca demanda específica — es visto más como parte de stack técnico que como habilidad principal	Usado en apps móviles, pequeños proyectos o prototipos; útil conocimiento complementario	Rara vez se menciona explícitamente en ofertas; se da por sentado como conocimiento de BD	Es una herramienta de soporte más que rol principal en vacantes
WebSocket / tiempo real	Demanda creciente en soluciones en tiempo real (chat, notificaciones, dashboards)	Las empresas que usan WebSocket valoran experiencia real con sockets, transmisiones en vivo	Se requiere conocimiento especializado (persistencia, reconexión, manejo eficiente)	WebSocket es un plus diferencial en muchas ofertas de frontend/backend moderno

Ejemplo práctico y funcional relacionando los temas

Alto nivel



C4Model

Diagrama de contexto:

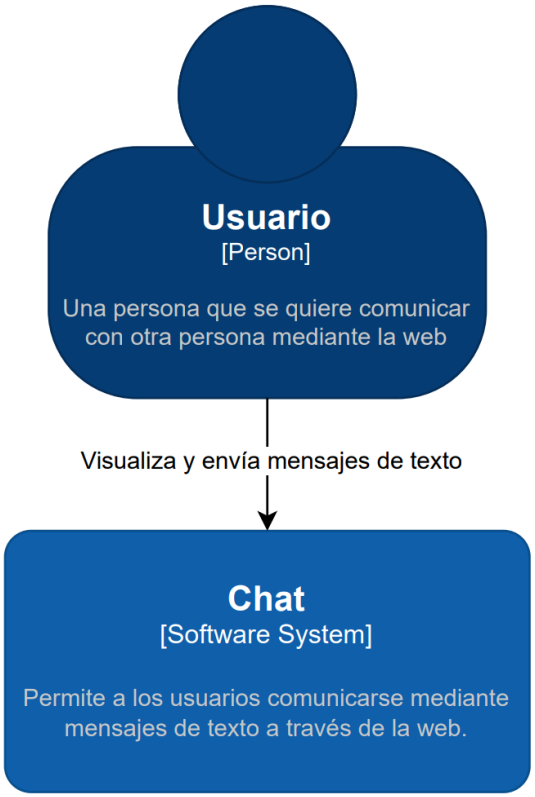


Diagrama de contenedores:

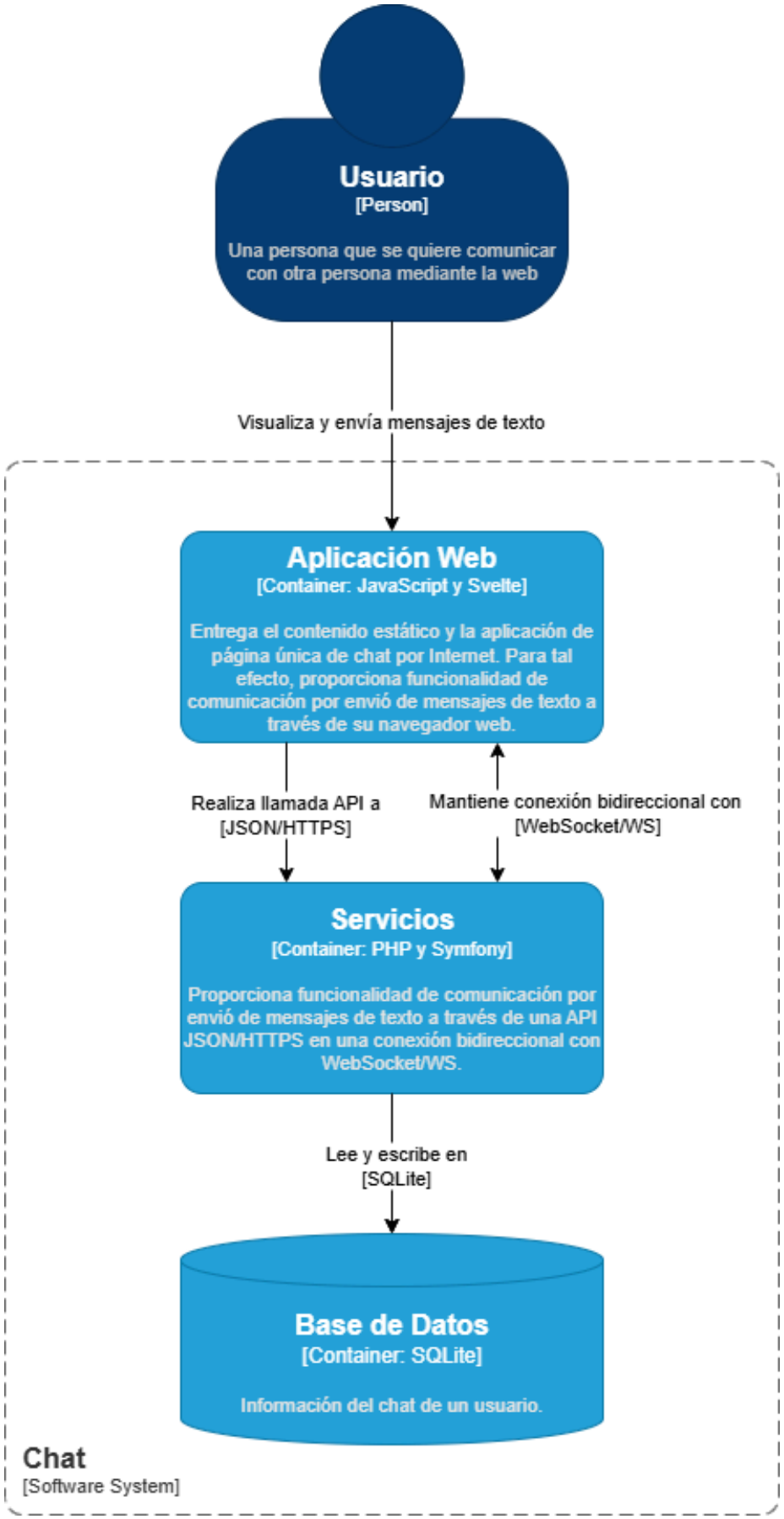


Diagrama de componentes:

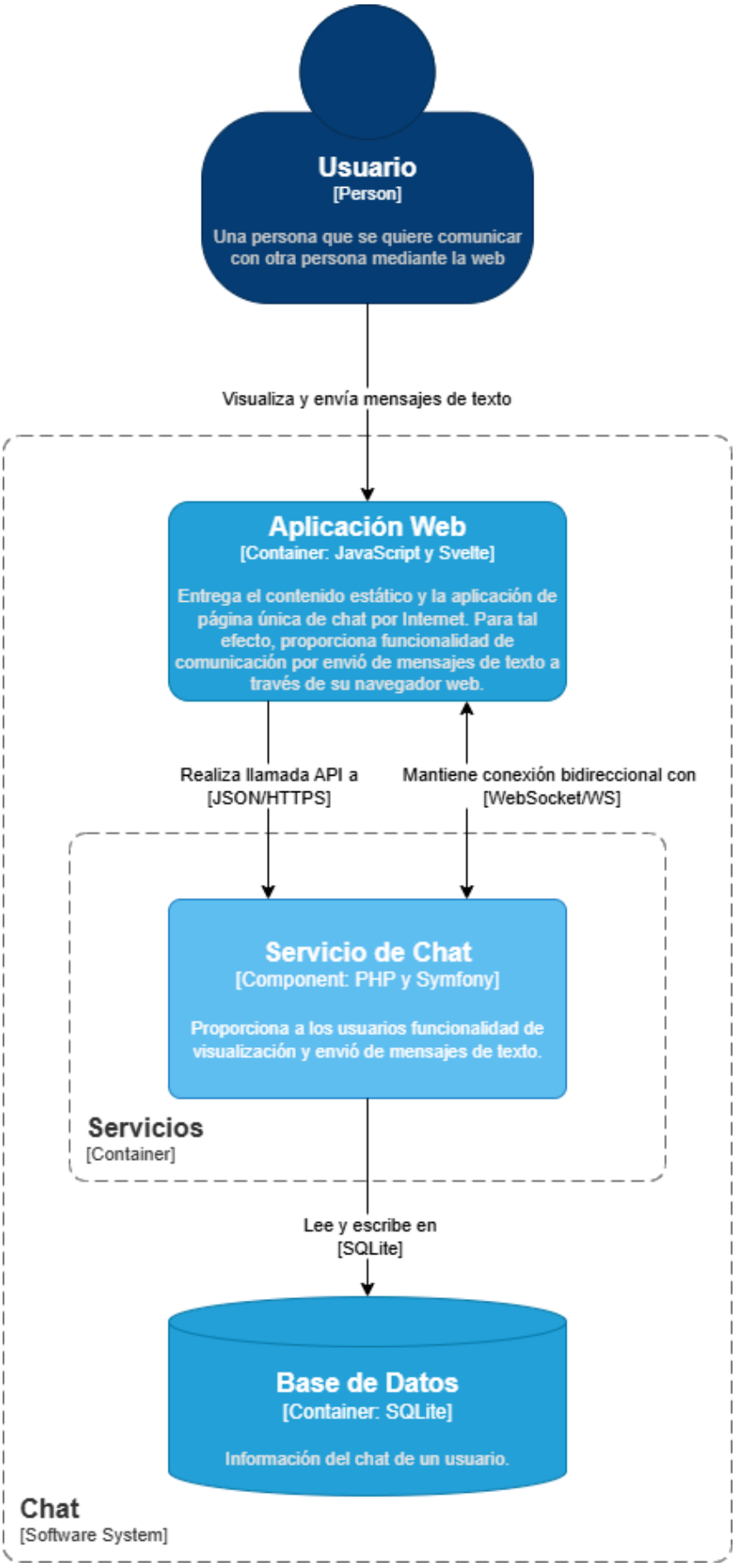


Diagrama Dynamic

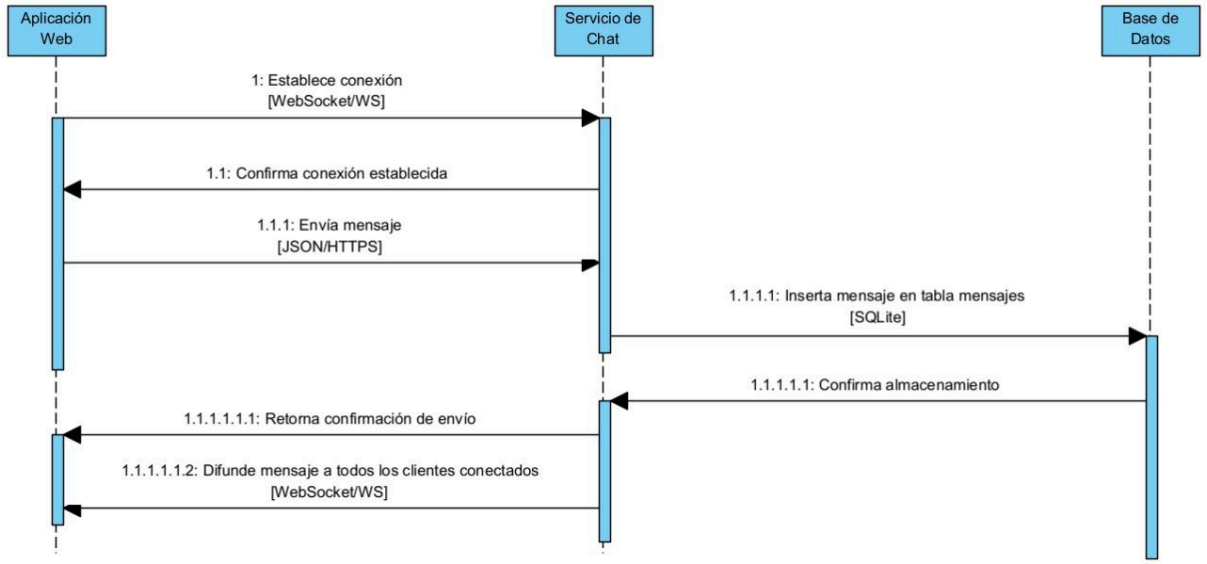


Diagrama Despliegue C4

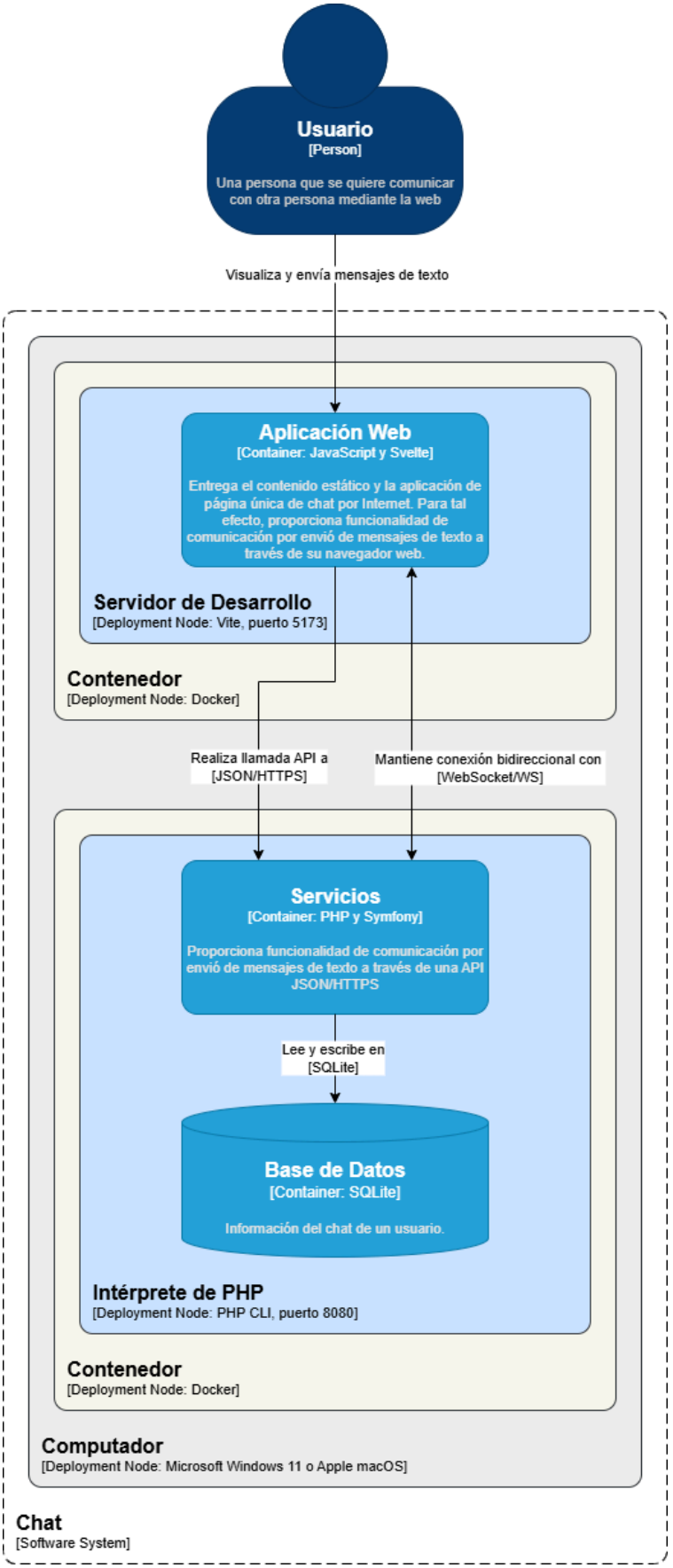


Diagrama de paquetes UML de cada componente

Diagrama de paquetes UML del componente Servicio de Chat:

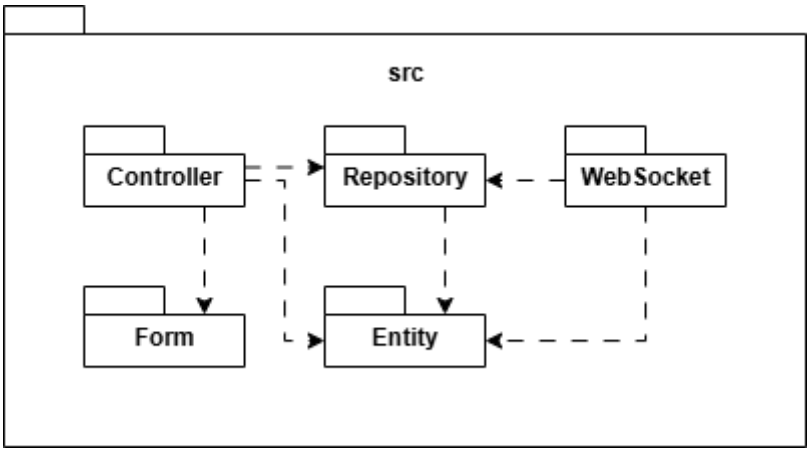
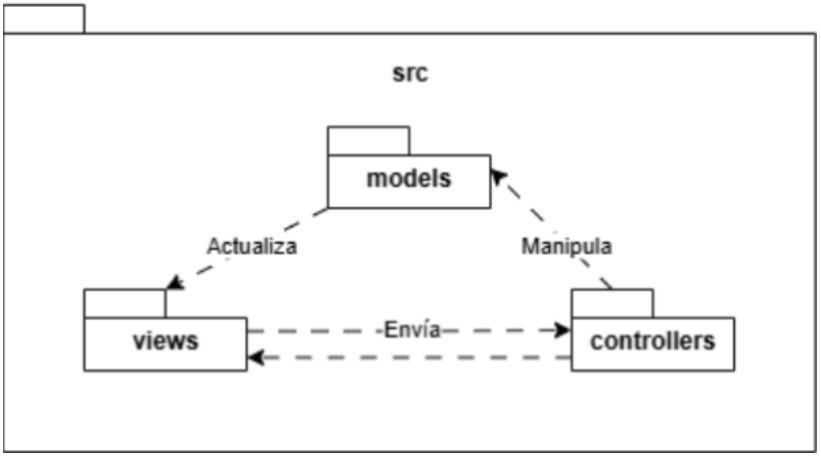


Diagrama de paquetes UML del componente que se encuentra dentro de la Aplicación Web:



Código Fuente en repositorio/s públicos git

https://github.com/adrianrrruiz/chat-app

TAG: https://github.com/adrianrrruiz/chat-app/releases/tag/FINISHED

Referencias

Cardacci, D. G. (2015). *Arquitectura de software académica para la comprensión* [PDF]. Econstor. Recuperado de https://www.econstor.eu/bitstream/10419/130825/1/837816424.pdf EconStor

González, L. F. S. (2021). *Aplicación Web Basada en el Patrón de Arquitectura Modelo-Vista-Controlador (MVC)*. TERC. Recuperado de https://terc.mx/index.php/terc/article/view/187 Tercero

Desarrolloweb. (2023, 20 de septiembre). ¿Qué es MVC? Recuperado de https://desarrolloweb.com/articulos/que-es-mvc.html DesarrolloWeb

FreeCodeCamp (español). (2021, 28 de junio). El patrón modelo-vista-controlador: Arquitectura y patrón. Recuperado de https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/ FreeCodeCamp

Catacora Murillo, L. A. (2017). *Impacto de un Sistema Web Empleando la Arquitectura MVC en los Procesos de Gestión y Administración Académica* [Tesis]. Repositorio UPT. Recuperado de https://repositorio.upt.edu.pe/handle/20.500.12969/162 Repositorio UPT

Lab Wallarm. (s. f.). ¿Qué es WebSocket y cómo funciona? Recuperado de https://lab.wallarm.com/what/protocolo-websocket/?lang=es Wallarm

Sendbird. (s. f.). Protocolos de comunicación WebSocket vs. HTTP. Recuperado de https://sendbird.com/es/developer/tutorials/websocket-vs-http-communication-protocols Sendbird

Microsoft Learn. (s. f.). API de componentes del protocolo WebSocket. Recuperado de https://learn.microsoft.com/es-es/windows/win32/websock/web-socket-protocol-component-api-portal