Pontificia Universidad Javeriana

Arquitectura de Software

Presentación 1

GRUPO 8

Samuel Ramírez Álvarez
Juan Diego Pérez Duarte
Juan Miguel Zuluaga Suárez

21 de abril de 2025

Facultad de Ingeniería Sistemas

Bogotá D.C. 2025

1. Introducción

En este trabajo del curso de Arquitectura de Software se presenta el desarrollo completo de un proyecto práctico basado en el stack: monolito con ASP.NET, C#, MVC y SQL Server. A partir de los temas asignados, se realiza una investigación teórica, su análisis y una implementación funcional, acompañada de los respectivos diagramas, matrices comparativas y documentación técnica.

El objetivo principal es comprender el funcionamiento y la interacción entre los distintos elementos del stack: la arquitectura monolítica como enfoque de desarrollo centralizado; el patrón MVC (Modelo-Vista-Controlador) como organizador lógico de la aplicación; el patrón DAO (Data Access Object) como intermediario entre la lógica de negocio y la base de datos; y ASP.NET como framework de desarrollo web. Todo esto se conecta con SQL Server, el sistema de gestión de bases de datos relacional que da soporte a la persistencia de datos en la aplicación.

El informe está estructurado en secciones que desarrollan las definiciones, características, historia, ventajas y desventajas de cada tema, casos de uso reales, análisis de su relación e integración, así como una serie de matrices comparativas que permiten evaluar el stack asignado frente a principios de diseño como SOLID, atributos de calidad, tácticas de arquitectura, patrones de diseño y su demanda en el mercado laboral.

Finalmente, se presenta un ejemplo funcional desarrollado por el grupo, que demuestra de forma práctica el uso e integración de todos los conceptos abordados. Este ejemplo incluye diagramas UML y C4, una arquitectura funcional en ASP.NET MVC con DAO, pruebas con Postman y despliegue mediante Docker. Todo esto, documentado en un repositorio público, garantiza la trazabilidad y reproducibilidad del sistema propuesto.

2. Temas Asignados

2.1 Monolito

- **Definición**
  Una arquitectura monolítica es un enfoque de diseño de software en el cual todos los componentes funcionales de una aplicación están integrados y ejecutados como una única unidad o proceso. Esto incluye la interfaz de usuario, la lógica de negocio, el acceso a datos y otros servicios internos. En este modelo, todas las funcionalidades están interconectadas y desplegadas juntas, lo que significa que cualquier cambio en una parte del sistema puede afectar a todo el conjunto.
- **Características**
  - Un solo artefacto desplegable.
  - Comunicación interna mediante llamadas directas entre componentes.
  - Acoplamiento estrecho entre módulos.
  - Escalabilidad vertical.
  - Despliegue y desarrollo simplificados.
- **Historia y evolución**
  La arquitectura monolítica, donde todos los componentes de una aplicación están integrados en una sola unidad, ha sido el enfoque tradicional desde los inicios de la programación estructurada. Se estima que su origen se remonta a los años 60 o 70, cuando las aplicaciones eran más simples y se desarrollaban principalmente en mainframes. Durante las décadas de 1980 y 1990, este modelo era común en sistemas empresariales, especialmente en entornos donde la escalabilidad no era un problema crítico. Sin embargo, con el auge de internet y aplicaciones web en la década de 2000, las limitaciones de la arquitectura monolítica, como la dificultad para escalar componentes específicos y la interdependencia de módulos, se hicieron evidentes. Esto llevó al surgimiento de arquitecturas distribuidas, como los microservicios, que permiten una mayor flexibilidad y mantenibilidad. Actualmente, aunque menos común para aplicaciones grandes, sigue siendo útil para proyectos pequeños o prototipos, como MVPs, debido a su simplicidad en desarrollo y despliegue.
- **Ventajas y desventajas**
  - **Ventajas**
    - Desarrollo y despliegue rápidos.
    - Fácil de probar en su conjunto.
    - Menor complejidad inicial.
    - Rendimiento eficiente en aplicaciones pequeñas.
  - **Desventajas**

        - Dificultad para escalar componentes específicos.
        - Cambios en una parte pueden afectar todo el sistema.
        - Complicaciones en equipos grandes debido al acoplamiento estrecho.
        - Menor flexibilidad para adoptar nuevas tecnologías.
- **Casos de uso**
    - Aplicaciones internas de pequeñas empresas.
    - MVPs o prototipos que requieren desarrollo rápido.
    - Sistemas con pocas dependencias externas y bajo requerimiento de escalabilidad.
- **Casos de aplicación (industria)**
    - ERPs locales para pequeñas y medianas empresas.
    - Aplicaciones de gestión académica, contable o de inventarios pequeñas.
    - Herramientas administrativas para PYMEs.

2.2 MVC (Model-View-Controller)

- **Definición**
  El patrón de diseño MVC (Modelo-Vista-Controlador) es una arquitectura de software que separa una aplicación en tres componentes principales: el Modelo (gestión de datos y lógica de negocio), la Vista (interfaz de usuario) y el Controlador (gestión de las peticiones del usuario). Esta separación permite una mayor modularidad y facilita el mantenimiento y escalabilidad de las aplicaciones.
- **Características**
    - Separación clara de responsabilidades.
    - Mejora el mantenimiento y la reutilización del código.
    - Facilita las pruebas unitarias y el desarrollo paralelo.
    - Promueve una estructura de código más organizada y modular.
- **Historia y evolución**
  El patrón MVC, que separa una aplicación en Modelo (datos y lógica de negocio), Vista (interfaz de usuario) y Controlador (manejo de interacciones), fue creado por Trygve Reenskaug en la década de 1970 mientras trabajaba en Smalltalk-79 en Xerox Palo Alto Research Center (PARC). Inicialmente, incluía cuatro partes: Modelo, Vista, Cosa y Editor, pero tras discusiones con desarrolladores de Smalltalk, se simplificó a tres componentes en Smalltalk-80, donde se implementó por primera vez con clases abstractas

como view y controller. En 1988, un artículo en The Journal of Object Technology (JOT) por ex-empleados de PARC consolidó MVC como un paradigma general para desarrollo de GUIs. Con la popularización de las aplicaciones web en los 90, frameworks como NeXT's WebObjects (1996, escrito en Objective-C) adoptaron MVC, extendiéndolo a entornos web. Variantes como HMVC, MVP y MVVM han surgido para abordar necesidades específicas, y frameworks modernos como Spring (2002), Ruby on Rails (2004) y Django (2005) han consolidado su relevancia. Hoy, MVC sigue siendo fundamental en desarrollo web y móvil, adaptándose a tecnologías como React y Angular.

- **Ventajas y desventajas**
    - **Ventajas**
        - Mejora la organización del código y la separación de preocupaciones.
        - Facilita la colaboración entre equipos de desarrollo y diseño.
        - Permite el desarrollo y mantenimiento más eficientes de aplicaciones complejas.
        - Facilita la reutilización de componentes y la escalabilidad.
    - **Desventajas**
        - Curva de aprendizaje inicial para desarrolladores nuevos en el patrón.
        - Puede sobreestructurar aplicaciones pequeñas, añadiendo complejidad innecesaria.
        - Requiere una planificación cuidadosa para evitar la dispersión de lógica entre componentes.
- **Casos de uso**
    - Aplicaciones web estructuradas con múltiples vistas y lógica de negocio compleja.
    - Sistemas que requieren una separación clara entre la interfaz de usuario y la lógica de negocio.
    - Aplicaciones que se benefician de un desarrollo y mantenimiento modular.
- **Casos de aplicación (industria)**
    - ASP.NET MVC en portales académicos y sistemas de gestión empresarial.
    - Spring MVC en aplicaciones bancarias y de seguros.
    - Ruby on Rails en startups y plataformas SaaS.

2.3 DAO (Data Access Object)

- **Definición**
  El patrón Data Access Object (DAO) es una estrategia de diseño estructural que proporciona una interfaz abstracta para acceder a una base de datos u otro mecanismo de almacenamiento persistente. Al mapear las llamadas de la aplicación a la capa de persistencia, el DAO permite operaciones de datos sin exponer los detalles de la base de datos, promoviendo así el principio de responsabilidad única y facilitando la separación entre la lógica de negocio y la lógica de acceso a datos.
- **Características**
  - Encapsula las operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
  - Proporciona una interfaz abstracta para la persistencia de datos.
  - Facilita la reutilización y mantenimiento del código.
  - Permite cambiar el mecanismo de almacenamiento sin afectar la lógica de negocio.
- **Historia y evolución**
  El patrón DAO, que proporciona una interfaz abstracta para acceder a bases de datos u otros mecanismos de persistencia, se popularizó en el contexto de Java EE, especialmente con las "Core J2EE Patterns" de Sun Microsystems, publicadas en los años 2000 como mejores prácticas para aplicaciones empresariales. Aunque su origen exacto no está documentado con precisión, parece haber surgido en los 90 como una forma de separar la lógica de acceso a datos de la lógica de negocio, alineándose con principios como la responsabilidad única. DAO se asoció tradicionalmente con bases de datos relacionales accesadas vía JDBC, y su adopción se intensificó con frameworks como Hibernate, Spring JPA y JPA, que facilitaron la implementación de la persistencia. Su evolución incluye su uso en otros lenguajes y tecnologías, aunque en algunos casos ha sido reemplazado por soluciones modernas como ORM. Actualmente, DAO sigue siendo relevante para mantener la modularidad y facilitar pruebas unitarias, especialmente en proyectos con cambios frecuentes en la capa de persistencia.
- **Ventajas y desventajas**
  - **Ventajas**
    - Promueve la separación de preocupaciones, mejorando la mantenibilidad del código.
    - Facilita las pruebas unitarias al permitir la simulación de la capa de datos.

        - Permite cambiar el sistema de almacenamiento sin modificar la lógica de negocio.
        - Mejora la escalabilidad y flexibilidad de la aplicación.
    - **Desventajas**
        - Puede introducir complejidad adicional en aplicaciones simples.
        - Requiere una planificación cuidadosa para evitar la duplicación de código.
        - Puede aumentar el número de clases y archivos en el proyecto.
- **Casos de uso**
    - Aplicaciones empresariales con múltiples fuentes de datos.
    - Sistemas que requieren una capa de persistencia modular y reutilizable.
    - Proyectos que necesitan facilitar las pruebas unitarias de la lógica de negocio.
- **Casos de aplicación (industria)**
    - Sistemas bancarios que manejan múltiples bases de datos.
    - Aplicaciones de comercio electrónico con lógica de negocio compleja.
    - Plataformas de gestión de contenido que requieren flexibilidad en la persistencia de datos.

2.4 ASP.NET

- **Definición**
  ASP.NET es un framework de desarrollo web del lado del servidor desarrollado por Microsoft, diseñado para construir aplicaciones web dinámicas, sitios web y servicios web. Permite a los desarrolladores crear aplicaciones utilizando lenguajes compatibles con .NET, como C# y VB.NET, y proporciona una amplia gama de herramientas y bibliotecas para facilitar el desarrollo web.
- **Características**
    - Soporte para múltiples modelos de programación, incluyendo Web Forms, MVC y Web API.
    - Integración con el Common Language Runtime (CLR) de .NET.
    - Compatibilidad con herramientas de desarrollo como Visual Studio.
    - Capacidades de seguridad integradas, como autenticación y autorización.
- **Historia y evolución**
  ASP.NET, un framework de desarrollo web de Microsoft, fue lanzado en

2002 como parte del .NET Framework, sucediendo a Active Server Pages (ASP), lanzado en 1996. ASP.NET introdujo un enfoque compilado basado en .NET, ofreciendo mejoras en rendimiento, seguridad y escalabilidad comparado con ASP, que era interpretado. Su evolución incluye:

- ***ASP.NET 1.0 (2002):*** Introdujo Web Forms, un modelo basado en eventos para desarrollo web.
- ***ASP.NET 2.0 (2005):*** Añadió Master Pages, Themes y mejoras en seguridad.
- ***ASP.NET 3.5 (2007):*** Incluyó soporte para AJAX y LINQ, facilitando el desarrollo de aplicaciones dinámicas.
- ***ASP.NET 4.0 (2010):*** Mejoró el rendimiento y las características de implementación web.
- ***ASP.NET MVC (2009):*** Introdujo el patrón MVC para aplicaciones web, separando Modelo, Vista y Controlador.
- ***ASP.NET Web API (2012):*** Para crear servicios HTTP, complementando las capacidades web.
- ***ASP.NET Core (2016):*** Un rediseño cross-platform, de código abierto y de alto rendimiento, unificando ASP.NET MVC y Web API. Soporta Windows, Linux y macOS, con soporte para Blazor desde versiones

Esta evolución refleja la adaptación de ASP.NET a las necesidades modernas, como la portabilidad y la integración con tecnologías emergentes.recientes, permitiendo desarrollo web con C# en el lado del cliente.

- **Ventajas y desventajas**
  - **Ventajas**
    - Permite el desarrollo rápido de aplicaciones web robustas y escalables.
    - Ofrece una amplia gama de herramientas y bibliotecas para facilitar el desarrollo.
    - Proporciona un alto nivel de seguridad y manejo de errores.
    - Facilita la integración con otros servicios y tecnologías de Microsoft.
  - **Desventajas**
    - Puede tener una curva de aprendizaje empinada para desarrolladores nuevos en el ecosistema .NET.

        - Las aplicaciones ASP.NET tradicionales están limitadas a plataformas Windows, aunque ASP.NET Core es multiplataforma.
        - El rendimiento puede verse afectado si no se optimiza adecuadamente.
- **Casos de uso**
    - Desarrollo de aplicaciones web empresariales.
    - Creación de servicios web y APIs RESTful.
    - Construcción de sitios web dinámicos y portales de contenido.
- **Casos de aplicación (industria)**
    - Portales gubernamentales y sistemas de gestión pública.
    - Aplicaciones de gestión empresarial (ERP, CRM).
    - Sistemas de comercio electrónico y plataformas de servicios en línea.

2.5 SQL Server

- **Definición**
  Microsoft SQL Server es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado por Microsoft. Está diseñado para almacenar, recuperar y administrar datos de manera eficiente, permitiendo a las aplicaciones acceder y manipular información de forma estructurada. Utiliza Transact-SQL (T-SQL), una extensión del estándar SQL, para interactuar con los datos.
- **Características**
    - Un solo artefacto desplegable.
    - Comunicación interna mediante llamadas directas entre componentes.
    - Acoplamiento estrecho entre módulos.
    - Escalabilidad vertical.
    - Despliegue y desarrollo simplificados.
- **Historia y evolución**
  Microsoft SQL Server, un sistema de gestión de bases de datos relacionales (RDBMS), comenzó con la versión 1.0 en 1989, desarrollada en colaboración con Sybase para OS/2. La partnership terminó en los años 90, y Microsoft continuó su desarrollo de forma independiente. Su evolución incluye:
    - ***SQL Server 1.0 (1989):*** Lanzamiento inicial, diseñado como un sistema de 16 bits para OS/2, desarrollado en colaboración con Sybase.

    - ***SQL Server 4.2 (1992):*** Compatible con OS/2 y Windows NT, marcando una transición hacia plataformas Microsoft.
    - ***SQL Server 6.0 (1993):*** Primera versión diseñada específicamente para Windows NT, independiente de Sybase, con mejoras en rendimiento.
    - ***SQL Server 7.0 (1998):*** Introdujo soporte para vistas indexadas y particionadas, mejorando la gestión de datos complejos.
    - ***SQL Server 2000 (2000):*** Añadió soporte para XML y funciones definidas por el usuario, facilitando integración con aplicaciones modernas.
    - ***SQL Server 2005 (2005):*** Incorporó Service Broker para mensajería y soporte para integración con CLR (Common Language Runtime).
    - ***SQL Server 2008 (2008):*** Introdujo soporte para datos espaciales, Filestream para manejar datos no estructurados y cifrado transparente.
    - ***SQL Server 2012 (2012):*** Añadió AlwaysOn Availability Groups para alta disponibilidad e índices columnstore para análisis de datos.
    - ***SQL Server 2014 (2014):*** Implementó OLTP en memoria para mejorar el rendimiento y ajuste automático de bases de datos.
    - ***SQL Server 2016 (2016):*** Introdujo Query Store para monitoreo de consultas y soporte para Linux, expandiendo su compatibilidad.
    - ***SQL Server 2017 (2017):*** Añadió soporte para contenedores Docker y servicios de aprendizaje automático integrados.
    - ***SQL Server 2019 (2019):*** Incorporó Big Data Clusters para análisis de datos masivos y soporte para Ledger, mejorando la trazabilidad.
    - ***SQL Server 2022 (2022):*** Introdujo integración con Purview para gobernanza de datos y Vector Engine para soporte de aplicaciones de inteligencia artificial.
- **Ventajas y desventajas**
    - **Ventajas**
        - Alta escalabilidad y rendimiento en entornos empresariales.
        - Amplia documentación y soporte por parte de Microsoft.
        - Integración fluida con otros productos y servicios de Microsoft.
    - **Desventajas**
        - Costo de licenciamiento elevado para algunas ediciones.
        - Requiere recursos de hardware considerables para un rendimiento óptimo.

        - Curva de aprendizaje pronunciada para usuarios sin experiencia previa.
- **Casos de uso**
    - Gestión de bases de datos en aplicaciones empresariales.
    - Implementación de soluciones de inteligencia de negocios (BI).
    - Desarrollo de aplicaciones web y móviles con backend robusto.
- **Casos de aplicación (industria)**
    - Instituciones financieras para manejo de transacciones y análisis de datos.
    - Hospitales y clínicas para gestión de historiales médicos electrónicos.
    - Empresas minoristas para control de inventarios y análisis de ventas.

3. Relación entre los temas asignados

Los temas asignados —Monolito, MVC, DAO, ASP.NET y SQL Server— están estrechamente interconectados y forman un stack tecnológico coherente para el desarrollo de aplicaciones empresariales en el ecosistema .NET.

- **Monolito**: Arquitectura donde todos los componentes de la aplicación se integran en una única unidad desplegable.
- **MVC (Model-View-Controller)**: Patrón de diseño que separa la lógica de presentación, la lógica de negocio y el acceso a datos, promoviendo una estructura modular y mantenible.
- **DAO (Data Access Object)**: Patrón que encapsula el acceso a la base de datos, proporcionando una interfaz abstracta para realizar operaciones CRUD, lo que facilita la separación de preocupaciones y mejora la mantenibilidad.
- **ASP.NET**: Framework de desarrollo web de Microsoft que permite construir aplicaciones web dinámicas, integrando de forma nativa el patrón MVC y facilitando la implementación de DAO.
- **SQL Server**: Sistema de gestión de bases de datos relacional que se integra perfectamente con ASP.NET, proporcionando un backend robusto para el almacenamiento y recuperación de datos.

En conjunto, estos componentes permiten desarrollar aplicaciones monolíticas estructuradas, donde ASP.NET implementa MVC y DAO para interactuar eficientemente con SQL Server.

4. Análisis del Stack Asignado

4.1 ¿Qué tan común es este stack?

- **Popularidad actual:**

  El stack compuesto por **ASP.NET Core**, **MVC**, **DAO** y **SQL Server** sigue siendo ampliamente utilizado en el desarrollo de aplicaciones empresariales en 2025. ASP.NET Core ha evolucionado significativamente, ofreciendo mejoras en rendimiento, seguridad y compatibilidad multiplataforma. La integración con herramientas modernas y su enfoque en el desarrollo de aplicaciones web escalables lo mantienen como una opción preferida para muchas organizaciones.

- **En qué sectores se utiliza:**
  Este stack es comúnmente adoptado en diversos sectores, incluyendo:
  - ***Finanzas:*** Para desarrollar sistemas bancarios, plataformas de inversión y aplicaciones de gestión financiera.
  - ***Salud:*** En la creación de sistemas de gestión hospitalaria, historiales médicos electrónicos y aplicaciones de telemedicina.
  - ***Educación:*** Para plataformas de aprendizaje en línea, sistemas de gestión estudiantil y portales educativos.
  - ***Comercio electrónico:*** En el desarrollo de tiendas en línea, sistemas de gestión de inventario y plataformas de pago.
  - ***Administración pública:*** Para portales gubernamentales, sistemas de gestión de trámites y aplicaciones de servicio al ciudadano.
- **Ventajas para proyectos de tipo empresarial:**

  El uso de este stack en proyectos empresariales ofrece múltiples ventajas:

  - ***Escalabilidad:*** ASP.NET Core permite desarrollar aplicaciones que pueden escalar horizontal y verticalmente, adaptándose al crecimiento del negocio.
  - ***Mantenibilidad:*** La implementación del patrón MVC y DAO facilita la separación de responsabilidades, lo que simplifica el mantenimiento y la evolución del software.
  - ***Seguridad:*** ASP.NET Core ofrece características de seguridad integradas, como autenticación y autorización, protegiendo las aplicaciones contra amenazas comunes.

- ***Integración:*** La compatibilidad con SQL Server y otras herramientas del ecosistema Microsoft permite una integración fluida con sistemas existentes.
- ***Productividad:*** El uso de herramientas como Visual Studio y Entity Framework acelera el desarrollo, reduciendo el tiempo de comercialización.

---

5. Matrices de Análisis

5.1 Principios SOLID vs Temas

Principio SOLID	Monolito	MVC	DAO	ASP.NET
S (Responsabilidad Única)	Fácil de mantener si se separan responsabilidades internamente.	Cada componente (modelo, vista, controlador) tiene su función clara.	Cada clase DAO se enfoca solo en acceso a datos.	Permite separar capas si se estructura bien el proyecto.
O (Abierto/Cerrado)	Limitado, requiere refactor para extender funcionalidad.	Se pueden extender controladores o vistas sin modificar los existentes.	Nuevas entidades pueden tener sus propios DAOs.	Flexible para agregar nuevos controladores.
L (Sustitución de Liskov)	No muy usado si no se abstraen clases.	Aplica si hay interfaces para controladores base.	Implementable con interfaces para DAOs.	Compatible con abstracción de servicios.
I (Segregación de Interfaces)	No se implementa por defecto.	Posible con interfaces personalizadas.	Implementable si DAOs son desacoplados.	Apoya desacoplamiento mediante servicios.
D (Inversión de Dependencias)	En su forma básica no aplica.	Posible si se inyectan dependencias en	Mejora con inyección de interfaces DAO.	Compatible con Dependency Injection.

		controladores.		

5.2 Atributos de Calidad vs Temas

Atributo de Calidad	Monolito	MVC	DAO	ASP.NET	SQL Server
Mantenibilidad	Media, difícil si el proyecto crece.	Alta, separa responsabilidades.	Alta, código reutilizable.	Buena documentación y tooling.	Alta si se diseña bien.
Escalabilidad	Baja, solo vertical.	Depende del diseño.	Escalable en capas de datos.	Puede integrarse con soluciones escalables.	Escalable a nivel de datos.
Rendimiento	Rápido en proyectos pequeños.	Eficiente por separación de capas.	Ejecuta consultas optimizadas.	Permite caching, optimización.	Alto rendimiento nativo.
Seguridad	Limitada si no se implementan buenas prácticas.	Protege contra inyecciones con validaciones.	Control de consultas parametrizadas.	Apoya autenticación, validaciones.	Soporta cifrado, roles, etc.
Usabilidad	Rápido de implementar.	Interfaces claras entre capas.	Sencillo de mantener.	Soporta vistas HTML completas.	No aplica directamente

5.3 Tácticas vs Temas

Tema	Táctica de Rendimiento	Táctica de Disponibilidad	Táctica de Seguridad	Táctica de Mantenibilidad

Monolito	Ejecución local sin sobrecarga de red	Despliegue único, menor complejidad	Autenticación básica	Código centralizado fácil de rastrear
ASP.NET	Caching de vistas y recursos	Gestión de errores personalizada	Validación de entrada y roles	Separación por capas y controladores reutilizables
DAO	Consultas optimizadas por entidad	Dependencia única a SQL Server	Uso de parámetros para evitar inyección	Abstracción del acceso a datos
SQL Server	Índices y procedimientos almacenados	Clústeres y backups automáticos	Cifrado de conexión y autenticación	Administración mediante SQL Server Management Studio

5.4 Patrones vs Temas

Tema	Patrón Aplicado	Descripción
Monolito	Patrón Arquitectónico	Organiza toda la aplicación como una única unidad de despliegue.
ASP.NET	Patrón MVC	Separa la lógica de presentación, control y datos de forma estructurada.
DAO	Patrón DAO	Aísla la lógica de acceso a datos del resto de la aplicación.
SQL Server	Patrones de Persistencia	Ejecuta operaciones CRUD mediante comandos SQL a través del patrón DAO.

5.5 Mercado Laboral vs Temas

Tecnología o Patrón	Demanda Actual	Uso en la Industria	Dificultad de Aprendizaje
Monolito	Media	Sistemas internos, PYMEs	Baja
MVC (C#/ASP.NET)	Alta	Aplicaciones empresariales	Media
DAO	Media	Persistencia clásica sin ORM	Baja
ASP.NET (Framework/Core)	Muy Alta	Gobierno, banca, salud	Media
SQL Server	Muy Alta	Financiero, retail, ERP	Media

---

6. Ejemplo práctico

6.1 Descripción funcional

El código desarrollado corresponde a un Sistema de Gestión Académica, una plataforma web diseñada para facilitar el registro, consulta y administración de información académica básica. El sistema permite a los usuarios gestionar estudiantes, asignaturas y notas de forma eficiente mediante un flujo CRUD completo (crear, consultar, actualizar y eliminar). La aplicación está construida como un monolito clásico utilizando ASP.NET MVC para la estructura del frontend y backend, junto con el patrón DAO (Data Access Object) para el acceso y manejo directo de datos.

El sistema almacena toda su información en una base de datos SQL Server, donde se registran las entidades Estudiante, Asignatura y Nota, permitiendo mantener relaciones integradas y consistentes entre ellas. El modelo MVC asegura una clara separación entre lógica de presentación, controladores y lógica de acceso a datos, mientras que DAO facilita la reutilización del código y el mantenimiento de las operaciones con SQL de manera directa y desacoplada del controlador.

Este proyecto se enfoca en proveer una solución estable, mantenible y fácilmente desplegable para entornos educativos como colegios o universidades que necesiten

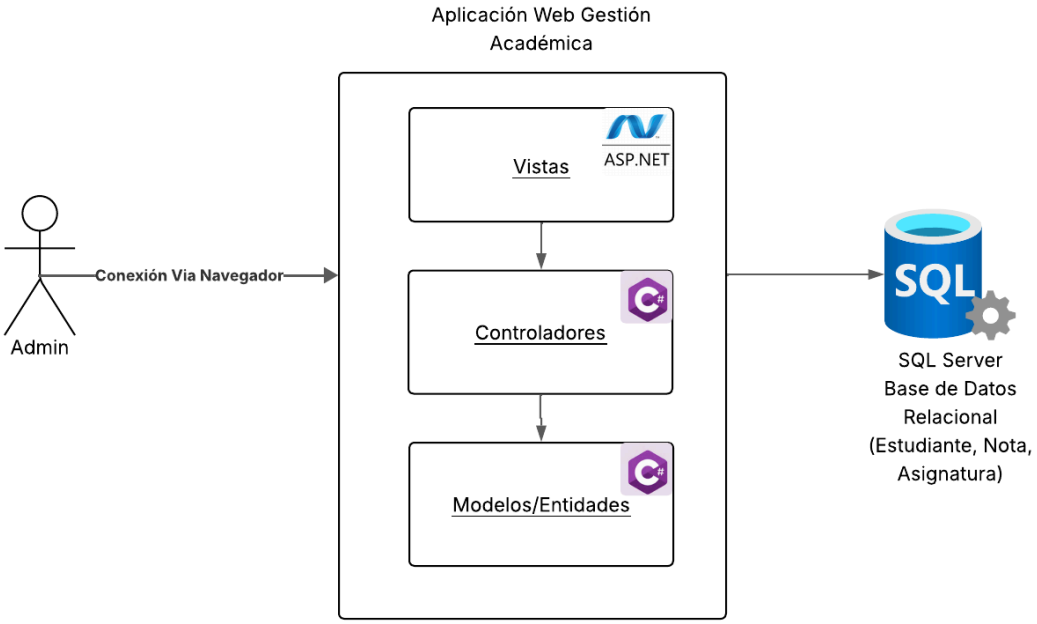
gestionar calificaciones de estudiantes. Además, el sistema fue dockerizado para permitir su ejecución completa a través de docker compose, integrando tanto la aplicación como el servicio de base de datos, y cuenta con una colección de pruebas Postman para validar su funcionalidad de forma automatizada.

- ***Entidades utilizadas***
    - **Estudiante:** Representa a cada alumno, con atributos como nombre, correo y documento de identidad.
    - **Asignatura:** Contiene información sobre las materias que puede cursar un estudiante.
    - **Nota:** Actúa como entidad puente, vinculando a un estudiante con una asignatura, y registrando la calificación y fecha de registro.
- ***Flujo general de uso***
    - El usuario ingresa al sistema desde la pantalla principal.
    - Desde la barra de navegación puede acceder a las secciones:
        - Gestión de Estudiantes
        - Gestión de Asignaturas
        - Gestión de Notas
    - En cada módulo puede:
        - Listar los registros existentes.
        - Crear, editar o eliminar registros.
        - Consultar detalles de cada entidad.
    - Al crear una nota, el sistema muestra menús desplegables para seleccionar el estudiante y la asignatura correspondientes.

---

7. Diagramas

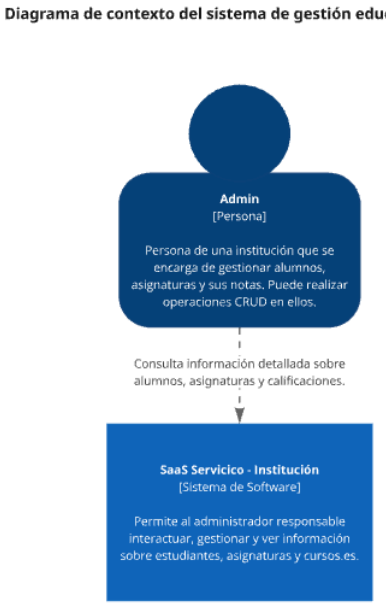
7.1 Diagrama de Alto Nivel



Ofrece una perspectiva general del sistema, donde se destacan los componentes principales y cómo se relacionan entre sí, abarcando tanto el frontend como el backend y la base de datos.

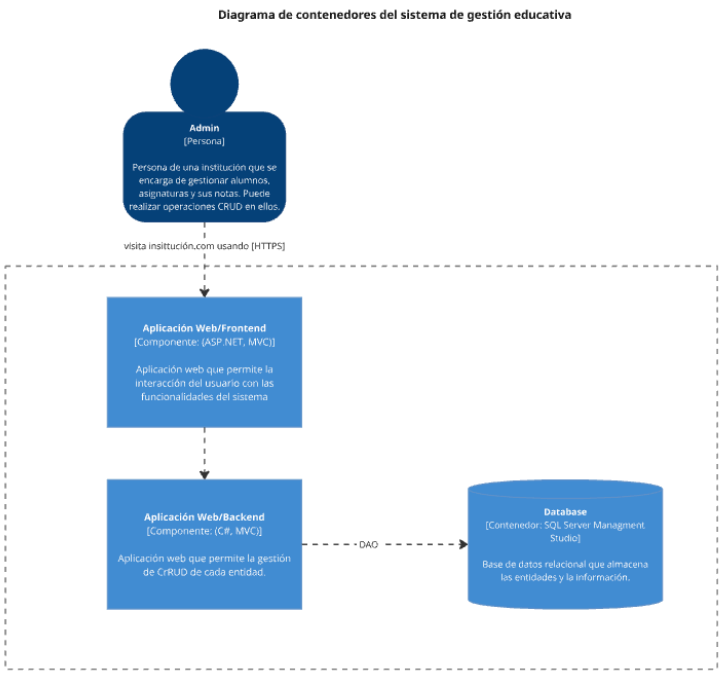
7.2 C4 Model

- **Nivel de Contexto**



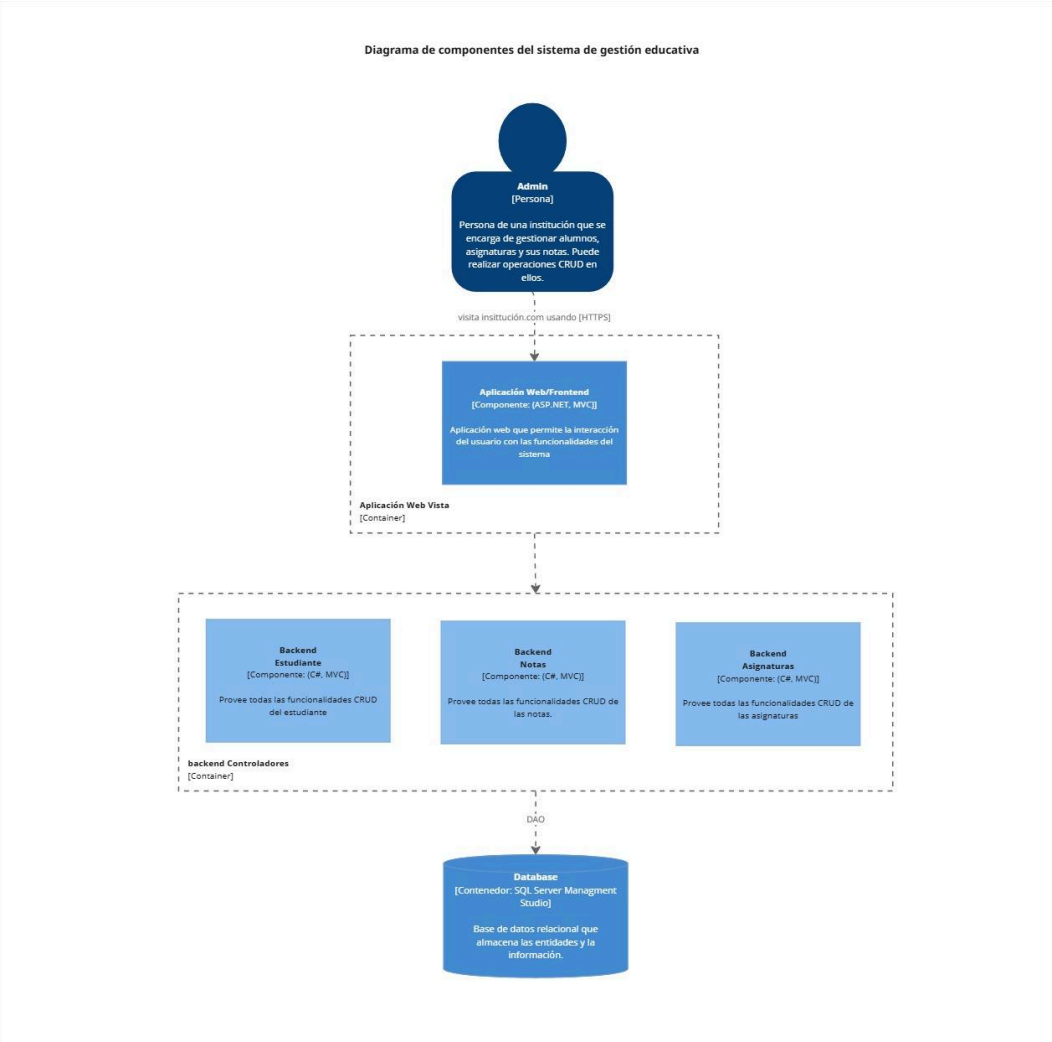
Proporciona una visión general del sistema dentro de su entorno. Tiene como propósito mostrar de forma clara el actor principal y cómo interactúan a alto nivel con el sistema. Permite entender el propósito general de la aplicación y cómo se relaciona con los elementos externos que la rodean.

- **Nivel de Contenedor**



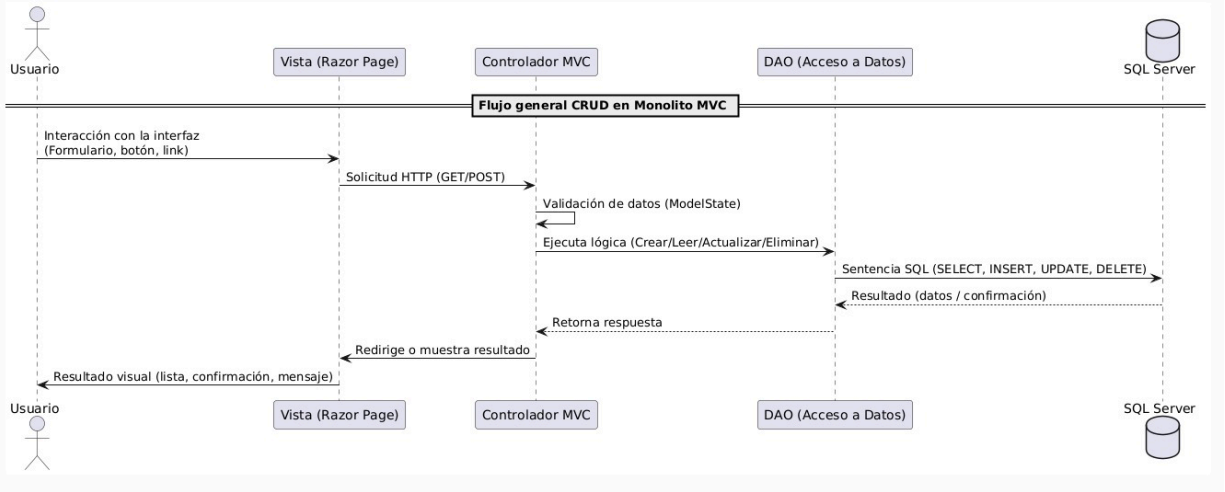
Muestra los principales bloques del sistema, como el frontend, backend y la base de datos, destacando cómo se comunican entre sí y las tecnologías que utilizan. Representa la arquitectura lógica de alto nivel de la aplicación

- **Nivel de Componente**



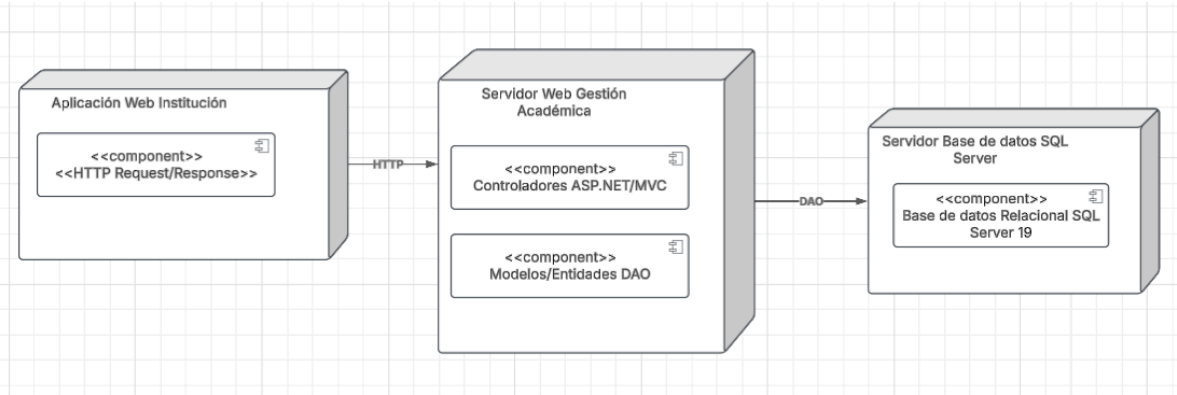
Detalla la estructura interna de cada contenedor, mostrando los componentes que lo conforman. Muestra cómo interactúa el sistema a un nivel de detalle mayor.

7.3 Diagrama Dinámico C4



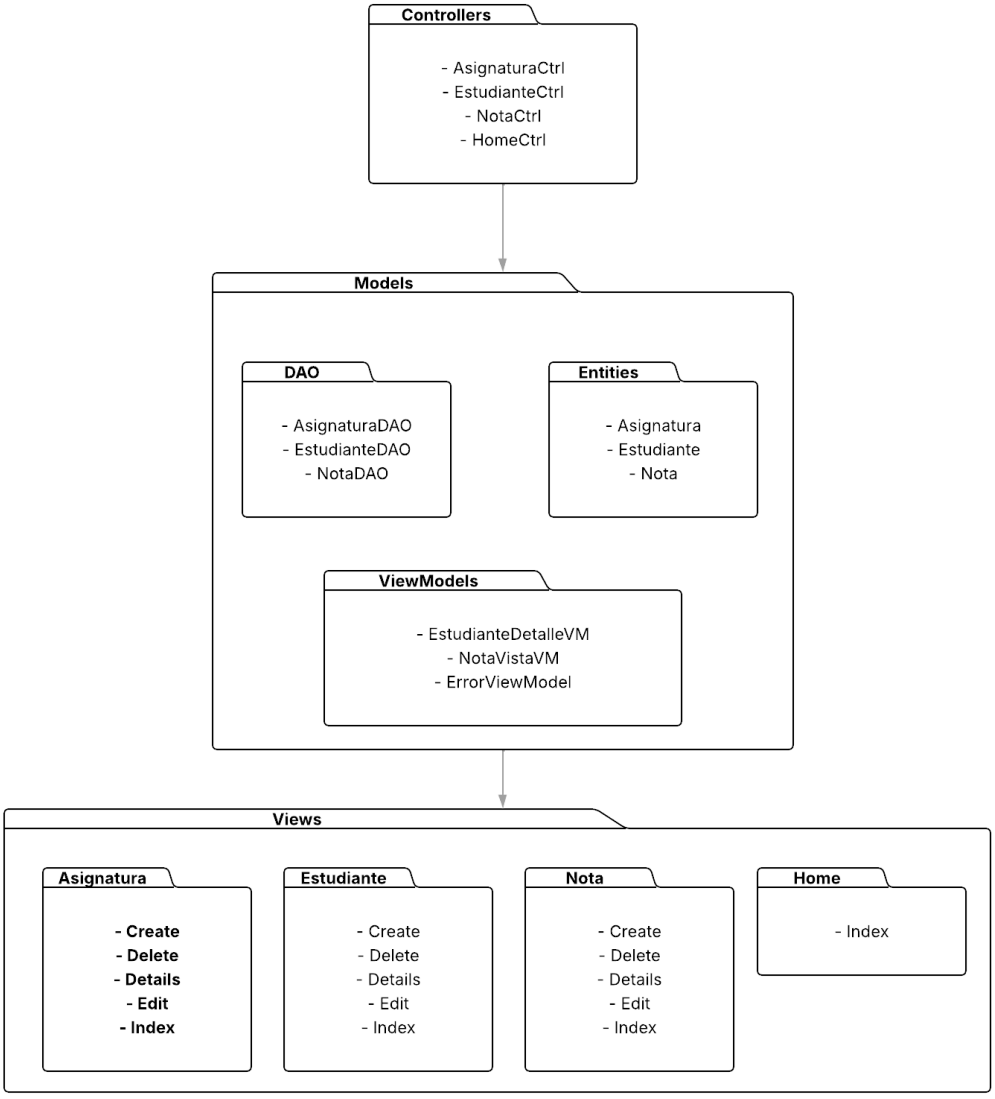
Muestra cómo los componentes del sistema colaboran en tiempo de ejecución para cumplir una funcionalidad, representando el flujo y orden de las interacciones entre ellos.

7.4 Diagrama de Despliegue C4



Muestra cómo los componentes del software se distribuyen en el hardware de un sistema, proporcionando una visión clara de la infraestructura física que soporta la aplicación y de cómo interactúan entre sí.

7.5 Diagrama UML de paquetes



El diagrama de paquetes UML muestra la organización modular del sistema en distintos componentes lógicos, agrupando clases relacionadas en paquetes como Controladores, Modelos (Entities), DAO y Vistas. Esta estructura refleja la arquitectura basada en MVC, destacando las dependencias entre capas y la separación clara de responsabilidades dentro del monolito.

---

8. Repositorio y ejecución

8.1 Código fuente

- https://github.com/juanzulu/MonolitoArquiTaller2

8.2 Postman

Puedes acceder a la colección de pruebas para la API del sistema MonolitoTaller (CRUD de Estudiantes, Asignaturas y Notas) a través del siguiente enlace:

- 🔗 Colección en Postman – MonolitoTaller CRUD API

---

9. Muestra funcional

- https://youtu.be/xd3Isod1ev0

---

10. Referencias

- https://learn.microsoft.com/en-us/dotnet/?WT.mc_id=dotnet-35129-website
- https://learn.microsoft.com/es-es/office/client-developer/access/desktop-database-reference/document-object-dao
- https://developer.mozilla.org/en-US/docs/Glossary/MVC
- https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022
- https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2019?view=sql-server-ver16
- https://www.ibm.com/think/topics/monolithic-architecture
- https://www.linkedin.com/pulse/identifying-monolithic-architectures-guide-harivola-loic-rabehaja/