UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Proyecto *Apis y Funciones Jarro_Valle*

Curso: *Tópicos de Base de Datos Avanzados*

Docente: *Mag. Patrick Cuadros*

Integrantes:

Jose Luis Jarro Cachi (2020067148)
Gustavo Alonso Valle Bustamante (2020066916)

Tacna – Perú
2024

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	MPV	ELV	ARV	10/10/2020	Versión Original

Sistema *Proyecto Apis y Funciones Jarro_Valle*
Documento de Visión

Versión *1.0*

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	MPV	ELV	ARV	10/10/2020	Versión Original

INDICE GENERAL

1. Introducción

El presente Documento de Visión describe el proyecto Apis y Funciones Jarro_Valle, el cual tiene como objetivo principal el desarrollo de una API para gestionar eventos dentro de un entorno académico. Este proyecto utiliza tecnologías modernas, como .NET Core, MongoDB, Docker, y GitHub Actions, con el fin de proporcionar una solución robusta y eficiente para el manejo de eventos mediante operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

La necesidad de este proyecto surge del deseo de ofrecer a los usuarios una herramienta sencilla y escalable para gestionar eventos de manera efectiva. Dado que los eventos son fundamentales en muchas actividades académicas, el sistema permite una gestión centralizada y de fácil acceso a los registros de eventos. La API se integra con MongoDB, una base de datos NoSQL, lo que permite una estructura flexible para almacenar la información de los eventos, y se empaqueta en contenedores Docker para facilitar su despliegue y portabilidad.

A través de GitHub Actions, el proyecto incorpora un sistema de integración continua, que asegura que las versiones del proyecto sean desplegadas automáticamente y que los desarrolladores puedan trabajar de manera más eficiente sin preocuparse por la infraestructura de implementación.

Este documento de visión tiene como propósito proporcionar una visión clara de los objetivos, funcionalidades y el alcance de la API Apis y Funciones Jarro_Valle, sirviendo como guía para su desarrollo, pruebas y despliegue. Además, define los términos técnicos, el flujo de trabajo del proyecto y las herramientas utilizadas para garantizar que todos los involucrados en el proyecto, desde los desarrolladores hasta los usuarios finales, tengan una comprensión clara del propósito y las características del sistema.

Con el fin de facilitar la adopción y el uso de esta API, el sistema incluye documentación interactiva a través de Swagger UI, permitiendo a los usuarios y desarrolladores explorar y probar la API de manera intuitiva.

Este documento es un punto de partida para la comprensión del proyecto, sus objetivos y las tecnologías involucradas, y servirá como referencia durante todo el ciclo de vida del proyecto

1.1 Propósito

El propósito del proyecto **Apis y Funciones Jarro_Valle** es desarrollar una API robusta y escalable para la gestión de eventos dentro de un entorno académico. Esta API permite a los usuarios crear, leer, actualizar y eliminar (CRUD) eventos de manera eficiente, utilizando tecnologías modernas como **.NET Core** para el desarrollo de la API, **MongoDB** como base de datos NoSQL para el almacenamiento de información, y **Docker** para facilitar el despliegue y la portabilidad del sistema.

La principal meta es proporcionar una solución sencilla y accesible para gestionar eventos académicos, tales como conferencias, seminarios, actividades extracurriculares, entre otros. La API está diseñada para ser utilizada por administradores, personal académico o sistemas automáticos que necesiten interactuar con los datos de los eventos. Además, la implementación de **GitHub Actions** facilita la integración continua y el despliegue automatizado, lo que asegura que siempre se utilice la versión más actual del sistema en un entorno de producción.

En resumen, el propósito de este proyecto es ofrecer una plataforma confiable y fácil de usar para la gestión de eventos académicos, con una infraestructura que garantice escalabilidad, automatización, y facilidad de integración en diferentes entornos de uso.

1.1. Alcance

El proyecto Apis y Funciones Jarro_Valle tiene como alcance la creación de una API para la gestión de eventos en un entorno académico, utilizando .NET Core como framework de desarrollo, MongoDB como base de datos NoSQL, y Docker para la implementación y portabilidad del sistema. La API estará disponible para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los registros de eventos, que son almacenados en la base de datos MongoDB.

Funcionalidades Clave:

1. Gestión de Eventos:
    - Crear nuevos eventos académicos.
    - Leer la información de eventos existentes.
    - Actualizar los detalles de eventos previamente creados.
    - Eliminar eventos de la base de datos.
2. Interacción con la Base de Datos (MongoDB):
    - La API se conecta a MongoDB, donde se almacenan los eventos. MongoDB se utiliza por ser una base de datos NoSQL, que ofrece flexibilidad y escalabilidad para manejar los datos de eventos.
3. Despliegue con Docker:
    - El proyecto está empaquetado en contenedores Docker, lo que permite un despliegue sencillo y consistente en diferentes entornos. Docker facilita la portabilidad del sistema, permitiendo que la API pueda ejecutarse en cualquier máquina que soporte Docker, sin importar el entorno o la configuración local.
4. Automatización y CI/CD:
    - GitHub Actions se emplea para la integración continua (CI) y despliegue continuo (CD) del proyecto. Esto asegura que las versiones más recientes del código se construyan, prueben y desplieguen automáticamente en el entorno de producción, garantizando la calidad y disponibilidad del sistema.
5. Documentación Interactiva con Swagger:
    - Se incluye Swagger UI para la documentación interactiva de la API, permitiendo a los desarrolladores explorar y probar las funcionalidades de la API de manera fácil y visual. Esto facilita la integración de la API en otros sistemas y aplicaciones.

Límites del Proyecto:

- Autenticación y Seguridad: Este proyecto no incluye una solución avanzada de autenticación o gestión de usuarios. Aunque se pueden implementar medidas básicas, la seguridad detallada de la API (por ejemplo, JWT, OAuth, etc.) no está contemplada en esta versión.
- Funcionalidades Adicionales: El alcance del proyecto se limita exclusivamente a la gestión de eventos académicos. Funcionalidades adicionales, como la gestión de usuarios o la integración con otros sistemas académicos, no forman parte de este proyecto.
- Escalabilidad en el Futuro: Si bien el sistema está diseñado para ser escalable, el proyecto no incluye planes específicos para la gestión de un volumen masivo de eventos más allá de la infraestructura básica de MongoDB y Docker.

1.3 Definiciones, Siglas y Abreviaturas

A continuación se presentan las definiciones, siglas y abreviaturas utilizadas en este documento, que son esenciales para la comprensión del proyecto Apis y Funciones Jarro_Valle:

Definiciones

- API (Interfaz de Programación de Aplicaciones): Es un conjunto de reglas y especificaciones que permiten que una aplicación interactúe con otras. En este proyecto, la API permite gestionar los eventos mediante operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- CRUD (Crear, Leer, Actualizar, Eliminar): Son las cuatro operaciones básicas que se pueden realizar sobre los datos en una base de datos. Este proyecto permite realizar estas operaciones en los registros de eventos almacenados en MongoDB.
- MongoDB: Es una base de datos NoSQL orientada a documentos. En este proyecto, se utiliza para almacenar los registros de eventos. Su modelo flexible de datos permite una fácil escalabilidad y manejo de grandes volúmenes de datos.

- Docker: Es una plataforma de contenedores que permite empaquetar aplicaciones y sus dependencias en unidades ligeras y portátiles llamadas contenedores. Docker se usa en este proyecto para empaquetar y desplegar la API de manera eficiente y escalable.
- GitHub Actions: Es una herramienta de integración continua (CI) y entrega continua (CD) proporcionada por GitHub, que permite automatizar la construcción, prueba y despliegue del código. En este proyecto, se utiliza para automatizar el flujo de trabajo del proyecto, incluyendo la construcción de contenedores Docker y su despliegue.
- Swagger UI: Es una herramienta de documentación interactiva que facilita la visualización y prueba de una API. A través de Swagger, los desarrolladores pueden interactuar con los puntos finales de la API sin necesidad de escribir código.
- Integración Continua (CI): Es una práctica de desarrollo de software que consiste en integrar cambios de código de manera frecuente en un repositorio compartido. Cada integración es verificada mediante pruebas automáticas para detectar errores rápidamente.
- Despliegue Continuo (CD): Es la práctica de automatizar el proceso de entrega del código desde el desarrollo hasta la producción, asegurando que el software se despliegue de manera segura y sin problemas.

Siglas y Abreviaturas

- API: Application Programming Interface (Interfaz de Programación de Aplicaciones).
- CRUD: Create, Read, Update, Delete (Crear, Leer, Actualizar, Eliminar).
- MongoDB: Database Management System (Sistema de Gestión de Bases de Datos NoSQL).
- Docker: Plataforma para empaquetar y ejecutar aplicaciones.
- GitHub Actions: Herramienta de CI/CD para automatización de flujos de trabajo.

- Swagger UI: Interfaz de documentación interactiva de APIs.
- CI: Continuous Integration (Integración Continua).
- CD: Continuous Deployment (Despliegue Continuo).
- .NET Core: Framework de desarrollo de aplicaciones basado en .NET, utilizado para crear la API.

Abreviaturas Específicas del Proyecto

- VJ: Iniciales de los integrantes del proyecto (Jarro y Valle).
- SI-8811: Código del curso académico relacionado con el proyecto (Tópicos de Bases de Datos Avanzados).
- Docker Hub: Repositorio de imágenes Docker, donde se almacena la imagen del contenedor de la API.

Estas definiciones y abreviaturas permiten una mejor comprensión del proyecto y de las tecnologías utilizadas en su desarrollo, asegurando que los términos clave sean entendidos por todos los participantes del proyecto y futuros usuarios de la API.

1.4 Referencias

A continuación se presentan las referencias clave utilizadas para el desarrollo del proyecto Apis y Funciones Jarro_Valle. Estas fuentes incluyen documentación técnica, recursos de desarrollo y tecnologías empleadas en la creación de la API y su infraestructura.

a) Documentación Oficial de .NET Core

   a. URL: https://learn.microsoft.com/en-us/dotnet/core/
      Descripción: Guía oficial de Microsoft para el desarrollo de aplicaciones con .NET Core, el framework utilizado para la creación de la API. Proporciona información sobre cómo trabajar con la arquitectura, las librerías y las herramientas asociadas.

b) MongoDB Documentation

a. URL: https://www.mongodb.com/docs/
Descripción: Documentación completa de MongoDB, la base de datos NoSQL utilizada en el proyecto para almacenar información de los eventos. Esta fuente ofrece detalles sobre su instalación, configuración y operaciones básicas.

c) Docker Documentation

a. URL: https://docs.docker.com/
Descripción: Documentación oficial de Docker, la plataforma utilizada para la creación y despliegue de contenedores. Explica cómo empaquetar aplicaciones y ejecutar contenedores de manera eficiente.

d) GitHub Actions Documentation

a. URL: https://docs.github.com/en/actions
Descripción: Guía oficial sobre GitHub Actions, utilizada para automatizar los flujos de trabajo de integración continua (CI) y despliegue continuo (CD). Específicamente relevante para la automatización del proceso de construcción y despliegue del proyecto.

e) Swagger UI Documentation

a. URL: https://swagger.io/tools/swagger-ui/
Descripción: Documentación de Swagger UI, la herramienta utilizada para generar una interfaz interactiva de documentación y prueba de la API. Es útil para explorar los puntos finales de la API de forma visual y simplificada.

f) Docker Hub

a. URL: https://hub.docker.com/
Descripción: Plataforma de distribución de imágenes Docker, utilizada para almacenar y distribuir las imágenes de contenedores creadas en el proyecto.

g) GitFlow Workflow

a. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Descripción: Recurso que explica el flujo de trabajo GitFlow, el cual es utilizado en el proyecto para la gestión de ramas en el control de versiones con Git.

h) .NET Core Swagger Integration

a. URL: https://swashbuckle.github.io/
Descripción: Información sobre la integración de Swagger con aplicaciones .NET Core. Se utiliza para la documentación y prueba interactiva de la API desarrollada en este proyecto.

Estas referencias proporcionan las bases teóricas y prácticas necesarias para el desarrollo, implementación, y despliegue de la API, asegurando que se sigan buenas prácticas y se utilicen herramientas estándar en la industria.

1.5 Visión General

El proyecto Apis y Funciones Jarro_Valle tiene como objetivo principal el desarrollo de una API para la gestión de eventos académicos, proporcionando una solución robusta, escalable y fácil de usar para administrar eventos en un entorno académico. Esta API está diseñada para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre registros de eventos almacenados en MongoDB, una base de datos NoSQL, con el fin de permitir un manejo eficiente de los datos en entornos educativos.

La API se construye utilizando .NET Core, un framework de desarrollo de aplicaciones, y se implementa utilizando Docker para asegurar que el sistema sea fácilmente portable y desplegable en diversos entornos. Para facilitar la gestión del proyecto, se utiliza GitHub Actions para automatizar la construcción, las pruebas y el despliegue del código mediante un proceso de integración y entrega continua (CI/CD).

Además, la API se documenta utilizando Swagger UI, lo que proporciona una interfaz interactiva que permite a los desarrolladores y usuarios explorar, probar y comprender los puntos finales de la API de forma visual e intuitiva. Esto facilita su integración con otras aplicaciones y servicios, además de mejorar la experiencia de los desarrolladores al interactuar con el sistema.

Alcance del Proyecto: La API permite a los usuarios gestionar eventos mediante operaciones CRUD. Los eventos pueden ser de diversa índole, como conferencias,

seminarios o actividades académicas. Sin embargo, el alcance del proyecto se limita a la gestión de estos eventos, sin incluir funcionalidades adicionales como la gestión de usuarios o la integración con otros sistemas académicos más allá de la base de datos de eventos.

Objetivos Técnicos:

- Desarrollo de la API: Usando .NET Core, se desarrollan los puntos finales para interactuar con los datos de eventos.
- Base de Datos: MongoDB se emplea para almacenar y gestionar los eventos de manera eficiente.
- Contenerización: El proyecto utiliza Docker para asegurar la portabilidad y facilidad de despliegue de la API.
- Automatización: A través de GitHub Actions, se implementa la integración continua y el despliegue automático de la API.
- Documentación Interactiva: Swagger UI se utiliza para generar documentación de la API y facilitar su exploración por los desarrolladores.

2. Posicionamiento

2.1 Oportunidad de negocio

La gestión de eventos es una actividad esencial dentro de las instituciones educativas, especialmente en el ámbito académico, donde se realizan conferencias, seminarios, talleres y actividades extracurriculares. Sin embargo, muchas veces los procesos involucrados en la gestión de estos eventos son manuales, fragmentados o dependen de sistemas obsoletos que no ofrecen la eficiencia ni la escalabilidad necesarias. La oportunidad de negocio que ofrece el proyecto Apis y Funciones Jarro_Valle radica en la automatización y centralización de la gestión de eventos académicos a través de una API moderna, escalable y fácil de integrar.

Al proporcionar una plataforma que facilite la creación, consulta, actualización y eliminación de eventos a través de una API, el proyecto satisface la necesidad de optimizar la administración de eventos dentro de las instituciones académicas. Las principales oportunidades de negocio incluyen:

- Reducción de la Carga Administrativa: Muchas instituciones académicas todavía gestionan eventos de manera manual, lo que implica un esfuerzo considerable en términos de tiempo y recursos. La API automatiza estos procesos, reduciendo la carga administrativa y mejorando la eficiencia operativa.
- Escalabilidad y Flexibilidad: Al utilizar MongoDB y Docker, el sistema puede adaptarse a diversas escalas, desde pequeñas instituciones hasta grandes universidades que gestionan una gran cantidad de eventos al mismo tiempo. La flexibilidad de la base de datos NoSQL permite manejar diversos tipos de datos de eventos, mientras que el uso de contenedores Docker facilita el despliegue y escalabilidad horizontal.
- Acceso Remoto y en Tiempo Real: La API proporciona una solución accesible desde cualquier lugar a través de la web o aplicaciones móviles, lo que facilita la gestión de eventos en tiempo real sin importar la ubicación.
- Integración con Otros Sistemas: La API está diseñada para ser fácilmente integrable con otros sistemas y plataformas académicas, lo que amplía sus posibilidades de aplicación más allá de la gestión de eventos, como por ejemplo, integrarse con sistemas de inscripción o plataformas de difusión.
- Facilidad de Uso: Gracias a la documentación interactiva proporcionada por Swagger UI, tanto los desarrolladores como los administradores pueden aprender rápidamente a usar la API, simplificando la adopción de la tecnología dentro de la institución.

2.2 Definición del problema

Las instituciones académicas, como universidades y centros educativos, enfrentan varios desafíos al gestionar eventos de forma manual o mediante sistemas tradicionales. Estos problemas incluyen la falta de automatización, la ineficiencia en el manejo de datos y la dificultad para acceder a la información en tiempo real. Algunos de los problemas clave que este proyecto busca resolver incluyen:

a) Gestión Manual de Eventos: Muchos centros educativos siguen gestionando eventos académicos mediante procesos manuales o mediante

hojas de cálculo, lo que hace que las tareas como el registro de eventos, la asignación de fechas y la actualización de información sean propensas a errores humanos. Esto resulta en una gestión ineficaz y difícil de escalar.

b) Falta de Centralización: Los sistemas existentes para gestionar eventos suelen estar fragmentados, lo que significa que los datos sobre los eventos (fechas, lugares, participantes, resultados) se distribuyen en diferentes sistemas o bases de datos. Esta falta de centralización genera confusión, pérdida de información y hace que la actualización y consulta de datos sea más difícil.
c) Escalabilidad Limitada: Los sistemas tradicionales no están diseñados para manejar grandes cantidades de datos o crecer de manera eficiente a medida que aumentan el número de eventos o participantes. Esto genera cuellos de botella en el proceso de gestión y dificulta el uso del sistema en instituciones con un alto volumen de eventos académicos.
d) Dificultad en la Integración con Otros Sistemas: Muchas veces, las soluciones para la gestión de eventos no están diseñadas para integrarse fácilmente con otras plataformas, como sistemas de gestión académica, sistemas de inscripción o plataformas de difusión de eventos. Esto provoca la necesidad de duplicar esfuerzos y realizar tareas redundantes, como la actualización de la misma información en distintos sistemas.
e) Falta de Accesibilidad y Visibilidad en Tiempo Real: Las instituciones académicas pueden enfrentar problemas con la accesibilidad de los datos de eventos, lo que impide que los usuarios, ya sean administradores, docentes o estudiantes, tengan acceso a la información en tiempo real o desde cualquier lugar. Esto limita la capacidad de la institución para organizar eventos de manera eficiente y responde de manera rápida ante cualquier cambio o necesidad.

El proyecto Apis y Funciones Jarro_Valle resuelve estos problemas al proporcionar una API que centraliza la información de eventos en una base de datos flexible como MongoDB. La automatización de las operaciones CRUD permite a las instituciones gestionar eventos de manera más eficiente, mientras que la escalabilidad y flexibilidad de la solución garantizan que pueda adaptarse al crecimiento y a las necesidades cambiantes de la institución. Además, la facilidad de integración con

otros sistemas académicos y la documentación interactiva proporcionada por Swagger UI mejoran la accesibilidad y la adopción del sistema, ofreciendo una solución moderna y accesible para la gestión de eventos.

3. Descripción de los interesados y usuarios

3.1 Resumen de los interesados

Los interesados son todas aquellas personas, grupos o entidades que tienen un interés en el desarrollo y éxito del proyecto, pero que no necesariamente interactúan directamente con la solución tecnológica. En el caso del proyecto Apis y Funciones Jarro_Valle, los principales interesados incluyen:

a) Institución Educativa (Universidades/Facultades): Los administradores de las universidades y facultades son los principales interesados. Ellos necesitan una solución eficiente para gestionar los eventos académicos, ya que tienen la responsabilidad de garantizar que los recursos y la programación de los eventos estén bien organizados.
b) Desarrolladores del Proyecto: El equipo de desarrollo, compuesto por programadores y arquitectos de software, son responsables de la creación, implementación y mantenimiento de la API. Su interés radica en garantizar la calidad y funcionalidad de la solución.
c) Equipo de Infraestructura y Soporte Técnico: Estos son los encargados de implementar, gestionar y mantener la infraestructura del sistema, como los servidores, bases de datos y contenedores Docker. Su interés es asegurar que la API sea escalable, esté disponible y funcione correctamente.
d) Usuarios Administrativos: Personal encargado de supervisar el funcionamiento de la API y gestionar los eventos en la plataforma. Este grupo puede incluir coordinadores de eventos y personal administrativo.
e) Proveedores de Tecnología: Proveedores de servicios en la nube o plataformas como Docker Hub, MongoDB, y GitHub Actions, que facilitan la infraestructura necesaria para el desarrollo, despliegue y mantenimiento de la solución.

3.2 Resumen de los usuarios

Los usuarios son aquellos que interactúan directamente con la API, ya sea para gestionar eventos, obtener información o realizar tareas específicas. En el caso de este proyecto, los principales usuarios incluyen:

a) Administradores de Eventos: Son los usuarios responsables de crear, actualizar, y eliminar eventos. También gestionan las fechas, descripciones y resultados de los eventos académicos.
b) Docentes y Profesores: En algunas instituciones, los docentes pueden tener acceso a la plataforma para gestionar eventos relacionados con sus actividades académicas, como conferencias, seminarios, y otros eventos docentes.
c) Estudiantes: Aunque no tienen acceso a la gestión de eventos, los estudiantes pueden interactuar con la API para consultar eventos futuros o pasados, conocer los resultados de los eventos y obtener detalles relacionados.
d) Desarrolladores e Integradores: Usuarios técnicos que se encargarán de integrar la API con otras plataformas académicas o servicios adicionales, como sistemas de inscripción o plataformas de gestión de cursos.

3.3 Entorno de usuario

El entorno de usuario se refiere al contexto en el cual los usuarios interactúan con la API, considerando las herramientas y tecnologías disponibles. En este proyecto, el entorno de usuario incluye:

a) Interfaz de Usuario Web/Móvil: Los usuarios acceden a la API a través de una interfaz web o móvil que consume los puntos finales de la API para visualizar y gestionar eventos. La interfaz debe ser intuitiva y fácil de usar para que los usuarios puedan acceder a las funcionalidades de la API sin dificultad.
b) API RESTful: Los usuarios interactúan con la API a través de solicitudes HTTP. Los puntos finales de la API permiten realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los eventos almacenados en la base de datos MongoDB.

c) Swagger UI: La API está documentada usando Swagger UI, lo que facilita a los usuarios y desarrolladores la visualización y prueba de los diferentes puntos finales de la API. Esto también permite a los usuarios explorar las funcionalidades y probar la API de manera interactiva.
d) Entorno de Desarrollo y Despliegue: La API se desarrolla en un entorno .NET Core, y se despliega en contenedores Docker, lo que asegura que los desarrolladores puedan trabajar en un entorno controlado y escalable. Los administradores pueden gestionar la infraestructura a través de Docker y la integración de GitHub Actions para CI/CD.

3.4 Perfiles de los interesados

Institución Educativa:

- Responsabilidades: Supervisar el correcto uso y la implementación del sistema.
- Intereses: Garantizar la eficiencia en la gestión de eventos académicos y la satisfacción de los usuarios.
- Expectativas: Obtener una solución escalable, segura y fácil de integrar con otros sistemas existentes.

Desarrolladores del Proyecto:

- Responsabilidades: Desarrollar y mantener la API, implementando nuevas funcionalidades y resolviendo problemas técnicos.
- Intereses: Asegurar la calidad, rendimiento y escalabilidad de la API.
- Expectativas: Desarrollar una API eficiente y documentada, con un código limpio y fácil de mantener.

Equipo de Infraestructura y Soporte Técnico:

- Responsabilidades: Gestionar la infraestructura tecnológica, servidores y bases de datos.
- Intereses: Asegurar que la infraestructura sea confiable, escalable y esté bien configurada para el rendimiento de la API.

- Expectativas: Que el sistema sea fácil de implementar, administrar y escalar.

Proveedores de Tecnología:

- Responsabilidades: Proveer las plataformas y servicios necesarios para el funcionamiento de la API.
- Intereses: Asegurar el correcto uso de las tecnologías que ofrecen (por ejemplo, Docker, MongoDB, GitHub Actions).
- Expectativas: Que el proyecto utilice sus servicios de manera eficiente y que no surjan problemas técnicos relacionados con su infraestructura.

3.5 Perfiles de los Usuarios

Administrador de Eventos:

- Responsabilidades: Gestionar y organizar eventos académicos, actualizar detalles de los eventos, y realizar tareas CRUD a través de la API.
- Intereses: Tener una herramienta eficiente para gestionar eventos de manera centralizada y fácil de usar.
- Expectativas: Contar con una interfaz simple y funcional que permita realizar tareas administrativas de manera rápida y sin errores.

Docentes y Profesores:

- Responsabilidades: Consultar información sobre eventos, actualizar detalles si es necesario.
- Intereses: Acceder fácilmente a los eventos relevantes para su área de trabajo.
- Expectativas: Contar con acceso a eventos relacionados con su docencia y ser capaces de realizar consultas de manera sencilla.

Estudiantes:

- Responsabilidades: Consultar eventos académicos, ver detalles de los eventos a los que pueden asistir y revisar resultados.
- Intereses: Estar informados sobre los eventos académicos y sus resultados.

- Expectativas: Tener acceso a una lista clara de eventos disponibles y resultados actualizados.

3.6 Necesidades de los interesados y usuarios

Interesados:

1. Institución Educativa: Necesita una solución que centralice la gestión de eventos, ahorre tiempo y recursos, y se integre fácilmente con otros sistemas.
2. Desarrolladores del Proyecto: Necesitan una arquitectura escalable, un código bien estructurado y herramientas que faciliten el despliegue y mantenimiento de la API.
3. Equipo de Infraestructura y Soporte Técnico: Necesita una infraestructura confiable, fácil de gestionar y con capacidad para escalar según sea necesario.
4. Proveedores de Tecnología: Necesitan que sus servicios sean utilizados de manera eficiente y sin inconvenientes técnicos.

Usuarios:

1. Administrador de Eventos: Necesita una herramienta sencilla para gestionar eventos sin problemas técnicos ni complejidades.
2. Docentes y Profesores: Necesitan consultar y gestionar eventos relacionados con su actividad académica de forma rápida y clara.
3. Estudiantes: Necesitan acceso fácil a la información sobre eventos y resultados de manera rápida y sin complicaciones.

4. Vista General del Producto

La API de Gestión de Eventos Jarro_Valle es una plataforma diseñada para gestionar de manera eficiente y centralizada los eventos académicos en instituciones educativas. A continuación, se detallan las secciones clave que proporcionan una visión integral del producto.

4.1 Perspectiva del producto

El producto es una API RESTful que permite a las instituciones académicas gestionar eventos académicos a través de operaciones CRUD (Crear, Leer, Actualizar, Eliminar). El sistema está basado en tecnologías modernas como .NET Core para el backend, MongoDB para la base de datos NoSQL y Docker para el despliegue y escalabilidad de la solución.

La API está diseñada para integrarse fácilmente con otras plataformas existentes en el entorno académico, como sistemas de inscripción o plataformas de gestión de cursos. Además, es accesible tanto a través de interfaces web y móviles, brindando flexibilidad en su uso.

La arquitectura de la API es escalable y flexible, permitiendo su implementación en instituciones de cualquier tamaño, desde universidades pequeñas hasta grandes universidades con múltiples facultades y eventos. El uso de Docker y GitHub Actions facilita la automatización del proceso de construcción y despliegue, lo que mejora la eficiencia operativa y garantiza un sistema robusto.

En términos de infraestructura, la API se gestiona de manera eficiente en plataformas de contenedores, lo que asegura una fácil implementación, escalabilidad horizontal y alta disponibilidad. Se espera que este producto sea utilizado tanto por los administradores de eventos como por estudiantes, docentes y desarrolladores encargados de la integración.

4.2 Resumen de capacidades

Las principales capacidades del producto son las siguientes:

a) Gestión de Eventos:
    a. Crear, leer, actualizar y eliminar eventos a través de solicitudes HTTP.
    b. Permitir la gestión de atributos de eventos como nombre, fecha, facultad, descripción, entre otros.
    c. Asociar resultados y detalles adicionales a los eventos académicos.

b) Acceso Remoto:
    a. Acceso a la API a través de Swagger UI, que permite visualizar y probar los diferentes puntos finales de la API.
    b. Accesibilidad desde aplicaciones móviles, web y otros sistemas que consumen la API.

c) Integración con Otros Sistemas:

   a. Posibilidad de integración con otros sistemas académicos, como sistemas de inscripción o plataformas de gestión de estudiantes.

   b. Facilidad para que los desarrolladores integren la API con servicios de terceros a través de su arquitectura RESTful.

d) Escalabilidad:

   a. Despliegue en contenedores Docker, permitiendo una escalabilidad eficiente tanto vertical como horizontal.

   b. Almacenamiento en MongoDB, que permite gestionar grandes volúmenes de datos de manera flexible y escalable.

e) Autenticación y Seguridad:

   a. Implementación de mecanismos de autenticación para proteger el acceso a la API y garantizar que solo los usuarios autorizados puedan realizar operaciones sensibles.

f) Interfaz de Usuario Interactiva:

   a. Uso de Swagger UI para interactuar con la API, facilitando su aprendizaje y prueba.

   b. Interfaces móviles y web para los usuarios finales (administradores, docentes, estudiantes).

4.3 Suposiciones y dependencias

Suposiciones

1. Requisitos de Infraestructura: Se asume que la infraestructura de la institución educativa será capaz de manejar contenedores Docker y que el acceso a internet será adecuado para el funcionamiento de la API.

2. Conocimiento Técnico de los Usuarios: Se asume que los administradores y desarrolladores tendrán el conocimiento técnico necesario para integrar y utilizar la API, especialmente en lo que respecta a las operaciones CRUD.

3. Consistencia en los Datos: Se supone que la base de datos MongoDB se mantendrá consistente y sin conflictos, garantizando que las operaciones de la API sean fiables y consistentes.

4. Acceso Web y Móvil: Se supone que los usuarios tendrán acceso a dispositivos móviles o equipos con navegador web para interactuar con la API de manera remota.

Dependencias

1. Docker: La API depende de Docker para su despliegue y gestión en contenedores. La infraestructura debe ser capaz de soportar la ejecución de contenedores Docker.
2. MongoDB: La API depende de MongoDB como su base de datos principal para almacenar la información de eventos. Cualquier problema con MongoDB podría afectar el funcionamiento de la API.
3. GitHub Actions: Para el proceso de integración continua y despliegue (CI/CD), la API depende de GitHub Actions, lo que asegura la automatización de las actualizaciones y despliegues de la solución.
4. Swagger UI: La documentación interactiva de la API se genera a través de Swagger UI, lo que permite a los usuarios probar y entender fácilmente los puntos finales de la API.

4.4 Costos y precios

El modelo de costos y precios dependerá principalmente de los siguientes factores:

Infraestructura de Servidores:

Si la API se aloja en servidores propios de la institución educativa, se tendrán costos asociados al mantenimiento de la infraestructura.

Si se opta por usar un servicio en la nube (AWS, Azure, etc.), habrá costos relacionados con la provisión de instancias de servidor y almacenamiento.

Licencias y Herramientas de Desarrollo:

MongoDB: La versión básica de MongoDB es de código abierto, pero si se necesita una versión con más características empresariales (como soporte o mayor capacidad), pueden aplicarse costos adicionales.

Docker: Docker es gratuito para la mayoría de los usos, pero las empresas pueden optar por suscripciones premium para obtener características avanzadas.

GitHub Actions: Dependiendo del uso, GitHub Actions puede tener costos adicionales si se superan los límites gratuitos proporcionados por GitHub.

Desarrollo y Mantenimiento:

Los costos asociados al desarrollo, mantenimiento y actualizaciones periódicas de la API también deben ser considerados. Esto incluye el tiempo y los recursos de los desarrolladores, así como la capacitación de usuarios y administradores.

Soporte y Consultoría:

En caso de que se necesite soporte adicional o consultoría técnica para la implementación, integraciones o personalización de la API, se pueden incurrir en costos por servicios externos.

Modelo de Precios:

El modelo de precios podría ser basado en la suscripción, donde las instituciones educativas pagan una tarifa mensual o anual por el uso de la API. Los precios podrían variar dependiendo del número de usuarios, eventos gestionados o volumen de datos. Alternativamente, la institución podría elegir una opción de pago único, donde se paga una tarifa fija por el acceso y uso de la API.

4.5 Licenciamiento e instalación

Licenciamiento

El producto Apis y Funciones Jarro_Valle puede estar licenciado bajo una licencia de código abierto (por ejemplo, MIT License o GPLv3), lo que permite a las instituciones educativas modificar el código, realizar personalizaciones y contribuir al proyecto. Sin embargo, si se opta por utilizar versiones comerciales de servicios de infraestructura (por ejemplo, MongoDB Atlas o Docker Enterprise), esos servicios estarán sujetos a sus propios términos y condiciones de licencia.

Instalación

1. Requisitos Previos:
    - Docker: Para ejecutar la API en un contenedor, se requiere tener instalado Docker en el sistema.
    - MongoDB: Debe configurarse una instancia de MongoDB para almacenar los datos de los eventos.
    - .NET Core: La API está desarrollada utilizando .NET Core, por lo que debe estar instalado en el sistema para desarrolladores que deseen realizar cambios en el código fuente.
2. Instrucciones de Instalación:
    - Clonar el repositorio del proyecto desde GitHub.
    - Construir la imagen Docker del proyecto utilizando el comando docker build -t eventos_vj ..
    - Ejecutar el contenedor Docker utilizando docker run -d -p 9091:80 eventos_vj.
    - Acceder a la API a través de la URL proporcionada en Swagger UI: http://localhost:9091/swagger.

La instalación es sencilla y está diseñada para ser ejecutada en entornos locales o de producción con configuraciones mínimas.

5. Características del producto

La API de Gestión de Eventos Jarro_Valle está diseñada para proporcionar una solución eficiente y flexible para la gestión de eventos académicos. A continuación, se describen las características principales que hacen de esta API una herramienta robusta y fácil de usar.

5.1 Gestión de Eventos

Operaciones CRUD:

La API permite realizar operaciones de Crear, Leer, Actualizar y Eliminar eventos académicos mediante solicitudes HTTP estándar (GET, POST, PUT, DELETE).

Cada evento tiene atributos como: nombre, fecha de inicio, fecha de término, facultad, resultado y descripción.

Creación de Eventos:

Los usuarios pueden crear eventos académicos proporcionando información detallada (nombre del evento, fechas, facultad involucrada, descripción, etc.).

La API asegura que los datos sean validados antes de ser almacenados en la base de datos.

Actualización de Eventos:

Los eventos existentes pueden ser actualizados, permitiendo cambios en cualquier campo relacionado con el evento.

Eliminación de Eventos:

Los usuarios pueden eliminar eventos previamente registrados, garantizando la eliminación de todos los datos asociados al evento.

5.2 Acceso a la API

Acceso Restful:

La API sigue el estilo de arquitectura REST, lo que garantiza una comunicación eficiente entre el cliente (móvil, web o backend) y el servidor.

Swagger UI:

Se ha integrado Swagger UI para proporcionar una interfaz interactiva donde los desarrolladores y usuarios pueden probar los puntos finales de la API.

Swagger proporciona una visualización clara de todos los endpoints disponibles, sus parámetros y respuestas esperadas.

Autenticación:

La API soporta mecanismos de autenticación para garantizar que solo los usuarios autorizados puedan acceder a recursos sensibles.

Se puede utilizar autenticación basada en tokens (JWT) para proteger los puntos finales.

5.3 Integración con MongoDB

Base de Datos NoSQL:

Los eventos se almacenan en una base de datos MongoDB, una base de datos NoSQL, que es flexible y adecuada para manejar grandes volúmenes de datos no estructurados.

MongoDB facilita el almacenamiento y la consulta rápida de eventos, incluso en instituciones con una gran cantidad de eventos y usuarios.

Modelo de Datos Flexible:

La base de datos se adapta a cambios en el modelo de datos sin necesidad de realizar migraciones complejas, lo que permite la evolución del sistema conforme cambian los requisitos.

5.4 Despliegue y Escalabilidad

Docker:

La API está empaquetada en un contenedor Docker, lo que facilita su despliegue en diferentes entornos (locales, servidores de producción, plataformas en la nube).

Docker permite la escalabilidad horizontal, lo que significa que se pueden agregar más contenedores según sea necesario para manejar el aumento en el tráfico o la carga.

Integración con GitHub Actions:

La integración de GitHub Actions permite la automatización del proceso de construcción, pruebas y despliegue de la API.

Esto garantiza que los cambios en el código sean implementados rápidamente y sin errores, lo que facilita la integración continua y el despliegue continuo (CI/CD).

5.5 Interfaz de Usuario

Interfaces Web y Móviles:

Aunque la API está diseñada para ser consumida por otros sistemas, también ofrece interfaces de usuario web y móviles para los administradores de eventos, docentes y otros usuarios que deseen interactuar directamente con la API.

Estas interfaces permiten gestionar eventos de manera sencilla a través de formularios y dashboards fáciles de usar.

Documentación Interactiva:

La API está completamente documentada utilizando Swagger, lo que permite a los desarrolladores y usuarios acceder a documentación interactiva, realizar pruebas de los puntos finales y entender cómo interactuar con la API sin necesidad de leer documentación extensa.

5.6 Validación y Manejo de Errores

Validaciones de Entrada:

La API realiza validaciones de entrada para asegurar que los datos recibidos sean correctos y completos antes de ser procesados o almacenados en la base de datos.

Las validaciones incluyen la comprobación de campos obligatorios, formatos de fecha, y coherencia entre los datos proporcionados.

Manejo de Errores:

La API proporciona respuestas claras y detalladas en caso de errores, facilitando la resolución de problemas.

Se manejan diferentes tipos de errores, como errores de validación, errores de autenticación y errores de conexión a la base de datos.

5.7 Seguridad

Autenticación y Autorización:

La API implementa un sistema de autenticación utilizando JSON Web Tokens (JWT) para validar y autorizar a los usuarios.

Esto garantiza que solo los usuarios autenticados y autorizados puedan realizar ciertas operaciones, como crear, actualizar o eliminar eventos.

Protección de Datos Sensibles:

Los datos sensibles, como contraseñas o información personal, se manejan de forma segura utilizando técnicas de cifrado.

Se sigue un enfoque de seguridad en capas para proteger tanto la comunicación como el almacenamiento de datos.

5.8 Soporte Multiplataforma

Compatibilidad con Diferentes Plataformas:

La API es compatible con aplicaciones móviles y web, lo que permite a los usuarios interactuar con el sistema desde diferentes dispositivos y entornos.

Los desarrolladores pueden integrar la API en diferentes sistemas académicos, independientemente de la plataforma que utilicen.

Manejo de Diferentes Versiones:

Se mantiene un control de versiones en la API para permitir a los desarrolladores adaptar sus sistemas a las versiones más recientes o utilizar versiones anteriores si lo requieren.

5.9 Desempeño y Optimización

Desempeño Rápido:

Se implementan técnicas de optimización en las consultas a la base de datos y en la comunicación entre los componentes del sistema para garantizar tiempos de respuesta rápidos.

El uso de MongoDB como base de datos NoSQL ayuda a gestionar grandes volúmenes de datos sin comprometer el rendimiento.

Escalabilidad:

La arquitectura de la API permite escalar el sistema fácilmente para satisfacer la demanda en entornos de producción con alto tráfico o grandes cantidades de eventos.

6. Restricciones

Las restricciones del proyecto son los límites o condiciones que afectan el desarrollo, funcionamiento y uso de la API de gestión de eventos. A continuación, se describen las principales restricciones que deben tenerse en cuenta durante el diseño y operación del sistema.

Restricciones Tecnológicas

- Dependencia de MongoDB:
  La API depende de MongoDB como base de datos NoSQL. Esto implica que el modelo de datos debe adaptarse a las características y limitaciones de MongoDB (por ejemplo, consultas no tan complejas como en bases de datos SQL, manejo de documentos en lugar de tablas). Esto puede limitar la flexibilidad en algunos casos cuando se requiere realizar consultas extremadamente complejas o transacciones más avanzadas.
- Compatibilidad con .NET Core:
  El proyecto está desarrollado utilizando .NET Core. Esto significa que cualquier servidor que implemente la API debe ser compatible con .NET Core, lo cual puede restringir el uso de algunas plataformas que no soporten esta tecnología de manera eficiente.
- Dockerización y Contenedores:
  La API está contenida en un contenedor Docker, lo que puede generar algunas restricciones en cuanto a la configuración de servidores o el uso de infraestructura que no sea compatible con contenedores o que requiera una infraestructura muy específica. Además, la solución debe estar desplegada en

una plataforma que soporte contenedores Docker (por ejemplo, AWS Elastic Beanstalk o servidores locales con Docker).

Restricciones de Rendimiento

- Tiempos de Respuesta en Consultas:
  Aunque MongoDB es eficiente para manejar grandes volúmenes de datos, el rendimiento de la API puede verse afectado por un alto volumen de solicitudes simultáneas, especialmente si las consultas no están optimizadas o si el número de eventos gestionados crece significativamente. Es necesario implementar mecanismos de caching o indexación para mejorar el rendimiento.
- Escalabilidad Limitada en el Ambiente Local:
  La escalabilidad de la aplicación puede estar limitada en un entorno de desarrollo o pruebas. En producción, la API está diseñada para escalar horizontalmente mediante la creación de más instancias del contenedor Docker, pero este proceso debe ser monitoreado y gestionado adecuadamente, ya que no todos los entornos de implementación pueden soportar escalabilidad sin restricciones.

Restricciones de Seguridad

- Autenticación y Autorización Básica:
  El sistema de autenticación basado en JWT proporciona un buen nivel de seguridad, pero podría no ser suficiente para ciertos requisitos avanzados, como autenticación multifactor o control de acceso granular. En algunos casos, será necesario ampliar las funcionalidades de seguridad, lo que podría aumentar la complejidad del sistema.
- Protección de Datos Sensibles:
  Si bien los datos sensibles (como contraseñas y detalles personales de los usuarios) se manejan con medidas de seguridad como cifrado y HTTPS, el sistema no está diseñado para cumplir con todos los estándares de seguridad más estrictos, como los establecidos en GDPR (Reglamento General de Protección de Datos) o HIPAA (Ley de Portabilidad y Responsabilidad del Seguro de Salud), lo que podría ser una restricción si la API se utiliza en entornos con requisitos de seguridad específicos.

Restricciones de Infraestructura

- Dependencia de AWS o Plataforma Similar:
  Para un despliegue adecuado, la API está optimizada para AWS Elastic Beanstalk o plataformas similares que soporten contenedores Docker. Si se requiere desplegar la API en una infraestructura diferente o privada sin estas características, pueden surgir complicaciones adicionales en la configuración y mantenimiento.
- Uso de Docker Hub:
  La solución depende de Docker Hub para almacenar las imágenes Docker del proyecto. Esto introduce una restricción en cuanto a la disponibilidad del servicio de Docker Hub, que debe estar operativo para que los contenedores puedan descargarse y ejecutarse correctamente. Además, si se requieren características específicas de Docker Hub (como repositorios privados), puede haber costos adicionales asociados.

Restricciones Funcionales

- Acceso y Control de Usuarios Limitado:
  Aunque la API permite crear, actualizar y eliminar eventos, el control de usuarios se limita al mecanismo de autenticación básico (JWT) sin una gestión avanzada de roles o permisos. Esto podría ser una limitación si se requieren funcionalidades como la asignación de permisos específicos (por ejemplo, administradores, usuarios con acceso solo de lectura, etc.).
- Soporte para Solo un Tipo de Evento:
  En su versión actual, la API está diseñada para gestionar solo un tipo de evento (por ejemplo, eventos académicos), lo que limita su uso para otros tipos de eventos o dominios que no estén relacionados con este ámbito académico. Adaptar la API para gestionar múltiples tipos de eventos podría requerir una reestructuración significativa del modelo de datos y las operaciones.

Restricciones de Usuario

- Interfaz Limitada:
  Aunque la API cuenta con documentación de Swagger para facilitar la interacción, no proporciona una interfaz de usuario avanzada o altamente

personalizada para los administradores o usuarios. La solución depende principalmente de la integración con otros sistemas, lo que significa que los usuarios finales podrían necesitar conocimientos técnicos para interactuar de manera efectiva con la API.

- Dependencia del Entorno Local para Pruebas:
  Durante las pruebas de desarrollo, la solución requiere un entorno local con Docker y MongoDB configurado, lo cual puede ser un desafío para aquellos que no estén familiarizados con estas tecnologías. Esto podría dificultar la adopción de la API por parte de usuarios sin experiencia técnica o sin la infraestructura adecuada.

Restricciones Legales y de Licenciamiento

- Licencia de Uso:
  El proyecto está licenciado bajo una licencia de código abierto (por ejemplo, MIT o GPL), lo que significa que los usuarios pueden modificar y distribuir el código según los términos de la licencia. Sin embargo, la API no debe utilizarse para fines comerciales sin el cumplimiento de los términos de la licencia, lo que podría ser una restricción si se pretende comercializar la API.
- Cumplimiento Normativo:
  El sistema debe cumplir con las leyes locales relacionadas con la protección de datos personales (como la Ley de Protección de Datos Personales en Perú o el GDPR en Europa). Aunque la API toma medidas para asegurar la privacidad de los datos, puede que no cumpla completamente con todos los estándares regulatorios si no se toman medidas adicionales.

7. Rangos de calidad

Los rangos de calidad describen los criterios y estándares que el sistema debe cumplir para garantizar su efectividad, fiabilidad, usabilidad, seguridad y rendimiento. Estos rangos definen los niveles de calidad que se esperan del sistema, y son esenciales para evaluar el éxito del producto y para asegurar que se cumplan las expectativas de los usuarios finales. A continuación, se detallan los rangos de calidad para la API de gestión de eventos.

Fiabilidad

- Disponibilidad:
    - La API debe estar disponible 99.9% del tiempo en un entorno de producción. Esto significa que la API puede estar fuera de servicio por un máximo de 8.76 horas al año.
    - Para garantizar esta disponibilidad, se debe implementar un sistema de monitoreo continuo y notificación de fallos.
- Tolerancia a Fallos:
    - La API debe ser capaz de recuperarse de fallos con un tiempo de recuperación de no más de 10 minutos en caso de caídas del sistema, y debe incluir mecanismos como réplicas de base de datos y escalabilidad automática.
- Pruebas de Confiabilidad:
    - Las pruebas de estrés y carga deben garantizar que la API pueda manejar al menos 500 solicitudes por segundo de manera estable, con un tiempo de respuesta inferior a 1 segundo para operaciones comunes.

Rendimiento

- Tiempo de Respuesta:
    - El tiempo de respuesta promedio para cualquier solicitud GET o POST debe ser inferior a 500 milisegundos bajo condiciones normales de carga.
    - Las consultas a la base de datos deben ser optimizadas, de modo que el tiempo de respuesta de las consultas a MongoDB no exceda 1 segundo para los datos más frecuentes.
- Escalabilidad:
    - La solución debe ser escalable para manejar un aumento en la carga sin afectar significativamente el rendimiento. Esto implica que el

sistema pueda manejar un aumento del 100% en el volumen de usuarios o solicitudes sin degradar el servicio.

- Capacidad de Manejo de Carga:
    - El sistema debe soportar al menos 1000 usuarios concurrentes realizando solicitudes al mismo tiempo sin experimentar tiempos de respuesta excesivos o caídas del servicio.

Usabilidad

- Documentación:
    - La API debe incluir documentación clara y detallada, accesible a través de Swagger UI, para facilitar la comprensión y uso por parte de los desarrolladores. La documentación debe cubrir:
        - Descripción de cada endpoint.
        - Ejemplos de solicitudes y respuestas.
        - Instrucciones claras sobre cómo utilizar los parámetros de la API.
- Interfaz de Usuario:
    - Si se proporciona una interfaz de usuario web o móvil, esta debe ser intuitiva y fácil de usar, permitiendo a los usuarios gestionar eventos sin dificultad. El diseño debe ser coherente con las mejores prácticas de diseño de interfaces y debe funcionar sin problemas en dispositivos móviles y de escritorio.
- Accesibilidad:
    - El sistema debe ser accesible para usuarios con discapacidad visual o auditiva. Esto incluye el uso de etiquetas adecuadas para campos de formulario, compatibilidad con lectores de pantalla, y la implementación de colores accesibles para garantizar la legibilidad.

Seguridad

- Autenticación y Autorización:
    - La API debe implementar un mecanismo de autenticación JWT (JSON Web Token) robusto para asegurar que solo los usuarios autenticados puedan acceder a la información.
    - Debe incluir un control adecuado de roles y permisos, permitiendo que los usuarios con privilegios diferentes (por ejemplo, administradores y usuarios regulares) tengan acceso solo a los recursos necesarios.
- Protección de Datos Sensibles:
    - Todos los datos sensibles deben ser almacenados y transmitidos de forma cifrada. Los datos en reposo deben ser cifrados utilizando estándares de seguridad como AES-256.
    - Las contraseñas deben ser cifradas utilizando un algoritmo seguro como bcrypt.
- Protección contra Ataques Comunes:
    - La API debe estar protegida contra ataques comunes como inyección SQL, XSS (Cross-Site Scripting) y CSRF (Cross-Site Request Forgery).
    - El sistema debe ser capaz de detectar y bloquear patrones de tráfico sospechosos que puedan indicar ataques de denegación de servicio (DoS).

Mantenibilidad

- Código Modular y Legible:
    - El código fuente debe estar organizado de manera modular, facilitando la comprensión, actualización y extensión. Cada componente debe tener una única responsabilidad clara.
    - El código debe ser comentado adecuadamente para asegurar que los desarrolladores puedan entender las decisiones de diseño y la lógica detrás del sistema.
- Pruebas Unitarias:

- El sistema debe tener un conjunto de pruebas unitarias que cubran al menos el 90% del código. Esto asegura que cualquier cambio en el código no rompa funcionalidades existentes y ayuda a detectar errores en fases tempranas.

- Actualización y Soporte:
  - El sistema debe permitir actualizaciones sin interrupciones del servicio. Las actualizaciones deben realizarse de manera segura, con un proceso de integración continua (CI) que garantice que las nuevas versiones de la API no introduzcan regresiones o errores.

Compatibilidad

- Compatibilidad con Navegadores y Dispositivos:
  - Si la API se integra con aplicaciones web, estas deben ser compatibles con los principales navegadores modernos como Chrome, Firefox, Safari, y Edge.
  - Si la API es consumida por aplicaciones móviles, debe funcionar correctamente en los sistemas operativos Android e iOS.
- Compatibilidad con Versiones de API:
  - La API debe ser compatible hacia atrás con versiones anteriores, asegurando que los cambios en la API no rompan el funcionamiento de las aplicaciones existentes.
  - Debe permitir el versionado de la API para facilitar la coexistencia de múltiples versiones y evitar problemas de compatibilidad a medida que el sistema evoluciona.

Costos

- Eficiencia en Costos:
  - La implementación de la solución debe ser eficiente en términos de recursos, para minimizar el uso de infraestructura y costos operativos.

- El uso de tecnologías como Docker y MongoDB debe garantizar que el sistema sea escalable sin incurrir en costos excesivos de infraestructura, especialmente en escenarios de alta demanda.

- Licencias de Software:
  - El uso de tecnologías de código abierto como .NET Core y MongoDB debe asegurarse de que no haya costos de licencias imprevistos. Además, cualquier dependencia de terceros debe ser bien gestionada para evitar costos adicionales.

8. Precedencia y Prioridad

La precedencia y prioridad son conceptos fundamentales en la planificación y gestión de proyectos, especialmente al desarrollar un producto complejo como una API para la gestión de eventos. Definir claramente las prioridades y la relación de precedencia entre las tareas y funcionalidades del sistema es esencial para la correcta ejecución del proyecto y para garantizar que los recursos se asignen de manera efectiva.

Precedencia

La precedencia se refiere a la relación temporal y dependiente entre las actividades del proyecto, en la que ciertas tareas deben completarse antes de que otras puedan iniciarse. Es crucial entender qué elementos dependen de otros para garantizar que las fases del desarrollo se sigan de forma lógica y eficiente.

Relación de Precedencia para el Proyecto:

1. Análisis y Diseño Inicial:
   - El análisis de requerimientos y el diseño de la arquitectura deben completarse antes de la codificación de la API. Esto incluye la definición de los endpoints y la estructura de la base de datos.
2. Desarrollo de la API:
   - El desarrollo de la API debe seguir el diseño previamente establecido. Las funcionalidades críticas de la API, como las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los eventos, deben desarrollarse antes de implementar características adicionales como la autenticación y el manejo de errores.
3. Integración de Base de Datos (MongoDB):
   - La configuración de MongoDB debe completarse antes de poder comenzar las operaciones de interacción con la base de datos.

Esto incluye la creación de la base de datos "juegos_florales" y la colección "Evento".

4. Desarrollo de la Interfaz de Usuario:
    - La interfaz de usuario (web o móvil) debe ser desarrollada después de que la API esté funcional. Esto se debe a que la interfaz interactúa directamente con la API, por lo que debe depender de la disponibilidad de los endpoints definidos previamente.
5. Pruebas Unitarias y de Integración:
    - Las pruebas deben comenzar después de que se haya completado la funcionalidad básica del sistema. Las pruebas unitarias deben realizarse de manera continua para garantizar la fiabilidad del código, mientras que las pruebas de integración aseguran que la API y la base de datos interactúan correctamente.
6. Despliegue y Publicación en DockerHub:
    - El despliegue de la API debe realizarse después de completar las pruebas y antes de que la API sea accesible para los usuarios finales. La imagen Docker debe estar lista para ser subida a DockerHub una vez que la aplicación esté configurada y las pruebas hayan sido satisfactorias.

Prioridad

Prioridades en el Proyecto:

1. Alta Prioridad:
    - Funcionalidad CRUD (Crear, Leer, Actualizar, Eliminar):
        - La capacidad de realizar operaciones CRUD sobre los eventos es la funcionalidad principal de la API, y debe tener la mayor prioridad en su desarrollo. Esto incluye la capacidad de manejar los eventos en la base de datos MongoDB de manera eficiente.
    - Conexión de la API a la base de datos MongoDB:
        - Establecer una conexión estable entre la API y la base de datos es esencial para que las operaciones CRUD funcionen correctamente. Esta tarea debe completarse antes que cualquier otra funcionalidad adicional.
    - Autenticación y Autorización:
        - La seguridad de la API, a través de la implementación de JWT y mecanismos de autorización, es crítica para garantizar que solo los usuarios autorizados puedan

interactuar con los datos. Esta funcionalidad debe implementarse lo más rápido posible.

2. Media Prioridad:
    - Manejo de Errores y Respuestas Adecuadas:
        - Si bien no es tan crítico como las funcionalidades CRUD, el manejo de errores es esencial para mejorar la experiencia del usuario. Esto incluye devolver respuestas claras y concisas en caso de errores en las solicitudes de la API.
    - Documentación de la API (Swagger):
        - La documentación clara es esencial para el uso eficiente de la API. Aunque no afecta directamente la funcionalidad del sistema, debe ser priorizada después de que las funcionalidades básicas estén implementadas, para que los desarrolladores puedan comprender cómo interactuar con la API.
3. Baja Prioridad:
    - Interfaz de Usuario (Web/Móvil):
        - La interfaz de usuario tiene una prioridad más baja en comparación con las funcionalidades de la API, ya que no es crítica para el funcionamiento básico del sistema. Sin embargo, debe desarrollarse para mejorar la accesibilidad y usabilidad del sistema.
    - Optimización y Escalabilidad:
        - Aunque la optimización y escalabilidad son importantes, en las primeras fases del proyecto puede haber cierta flexibilidad. Estas tareas deben llevarse a cabo una vez que el sistema funcione correctamente bajo cargas normales de trabajo.

Gestión de Cambios en Precedencia y Prioridad

A medida que el proyecto avanza, puede haber necesidad de revisar la precedencia y la prioridad de ciertas tareas, debido a:

1. Nuevos requerimientos del cliente o usuario.
2. Imprevistos técnicos o problemas de integración.
3. Feedback de pruebas que puede indicar áreas de mejora o nuevos riesgos.

Es fundamental tener un proceso ágil de gestión de cambios, utilizando herramientas como Jira o Trello para priorizar las tareas y gestionar los cambios de manera efectiva.

9. Otros requerimientos del producto

En esta sección se describen los **otros requerimientos del producto** que no están directamente relacionados con la funcionalidad básica del sistema, pero que son esenciales para asegurar su correcta operación, su alineación con las normativas legales, su calidad y seguridad, así como su capacidad de comunicación con otros sistemas o plataformas.

b) Estandares legales

El sistema debe cumplir con todas las normativas legales y regulaciones locales e internacionales aplicables a la gestión de datos, la privacidad de los usuarios, y la operatividad de la API en el entorno académico y empresarial. Los requisitos legales pueden variar según la ubicación de los usuarios y la naturaleza de los datos gestionados. A continuación, se detallan los estándares legales a considerar:

- Protección de Datos Personales (GDPR):
    - Si el sistema maneja datos personales de usuarios, debe cumplir con el Reglamento General de Protección de Datos (GDPR, por sus siglas en inglés) de la Unión Europea, que regula el almacenamiento, tratamiento y uso de datos personales de los ciudadanos de la UE. Esto incluye:
        - Consentimiento explícito para recolectar datos.
        - Derecho a la eliminación de datos (derecho al olvido).
        - Transparencia sobre cómo se utilizan los datos personales.
- Ley de Protección de Datos Personales en Perú (Ley N° 29733):
    - En el contexto peruano, el sistema también debe alinearse con la Ley N° 29733 que regula la protección de los datos personales. Esto implica:
        - Consentimiento informado para la recolección de datos personales.
        - Implementación de medidas de seguridad adecuadas para proteger los datos almacenados.
- Licencias de Software:
    - El sistema debe cumplir con las licencias de software de las tecnologías utilizadas (por ejemplo, .NET Core, MongoDB, Docker), garantizando que el uso de estas herramientas no infrinja

las leyes de propiedad intelectual ni los derechos de los creadores del software.

c) Estandares de comunicación

Los estándares de comunicación son fundamentales para garantizar la interoperabilidad entre los componentes del sistema y la correcta integración con otros servicios y plataformas. A continuación, se mencionan algunos de los estándares de comunicación clave para el proyecto:

- Protocolo HTTP/HTTPS:
    - La API debe comunicarse utilizando HTTP o, preferiblemente, HTTPS para garantizar la seguridad de las transacciones. HTTPS debe estar habilitado para garantizar que todas las solicitudes y respuestas se cifren durante la transmisión.
- JSON como Formato de Intercambio de Datos:
    - La API debe utilizar JSON (JavaScript Object Notation) como formato estándar para la entrada y salida de datos. JSON es ampliamente utilizado por su facilidad de lectura y compatibilidad con la mayoría de los lenguajes de programación.
- RESTful API:
    - La API debe seguir los principios de un diseño RESTful (Representational State Transfer), lo que implica:
        - Uso de métodos HTTP estándar (GET, POST, PUT, DELETE).
        - URLs claras y descriptivas que representen recursos (por ejemplo, /api/eventos).
        - Respuestas coherentes con códigos de estado HTTP apropiados (200 OK, 201 Created, 400 Bad Request, etc.).
- Autenticación mediante JWT (JSON Web Tokens):

- La API debe implementar JWT para la autenticación de usuarios, garantizando que las solicitudes solo sean aceptadas si están firmadas con un token válido y seguro.

- CORS (Cross-Origin Resource Sharing):
  - Se deben establecer políticas de CORS para permitir que la API sea consumida desde diferentes dominios, asegurando que las solicitudes externas sean adecuadamente gestionadas y autorizadas.

d) Estandaraes de cumplimiento de la plataforma

El producto debe estar alineado con los estándares de cumplimiento de la plataforma para garantizar su correcta operación y conformidad en los entornos en los que se despliegue. Esto incluye:

- Cumplimiento con Docker y Kubernetes:
  - El sistema debe ser compatible con contenedores Docker para su fácil despliegue y escalabilidad. Además, debe ser capaz de integrarse con Kubernetes si se utiliza para la orquestación de contenedores en el entorno de producción.

- Despliegue en la Nube (AWS, Azure, Google Cloud):
  - El producto debe ser compatible con los principales proveedores de servicios en la nube (como Amazon Web Services (AWS), Microsoft Azure, o Google Cloud Platform (GCP)). La solución debe ser escalable y permitir un despliegue automático a través de herramientas como DockerHub o GitHub Actions.

- Cumplimiento con las mejores prácticas de CI/CD:
  - La integración continua (CI) y el despliegue continuo (CD) deben ser gestionados mediante GitHub Actions o herramientas similares, garantizando que las actualizaciones del sistema se realicen de manera segura y eficiente, sin tiempos de inactividad importantes.

- Compatibilidad con Sistemas Operativos y Navegadores:
    - El sistema debe ser compatible con los principales sistemas operativos como Windows, Linux y macOS. Además, la interfaz web debe ser accesible a través de los navegadores más utilizados como Chrome, Firefox, Safari y Edge.

e) Estandaraes de calidad y seguridad

Los estándares de calidad y seguridad son fundamentales para asegurar que el sistema funcione correctamente y esté protegido frente a vulnerabilidades. Estos incluyen:

- Normas de Calidad del Código:
    - El código fuente debe seguir las mejores prácticas de programación y estándares de codificación (por ejemplo, PSR-12 para PHP, PEP-8 para Python). Además, debe contar con un sistema de control de versiones mediante Git para facilitar el seguimiento de cambios y las colaboraciones entre los desarrolladores.
- Pruebas Automatizadas:
    - Deben implementarse pruebas unitarias, de integración y de carga para asegurar la estabilidad y el rendimiento del sistema. Las pruebas deben ser ejecutadas de manera automática durante el proceso de desarrollo mediante herramientas de integración continua (CI) como GitHub Actions.
- Seguridad de la API:
    - La API debe ser segura frente a vulnerabilidades comunes como inyección SQL, Cross-Site Scripting (XSS), y Cross-Site Request Forgery (CSRF). Además, debe utilizar HTTPS para cifrar la comunicación y OAuth2 o JWT para asegurar la autenticación de los usuarios.
- Protección de Datos:

  - La API debe cumplir con las normativas de protección de datos, como la encriptación de datos sensibles en reposo y en tránsito. Además, debe implementar políticas de privacidad que permitan a los usuarios gestionar su información personal y solicitar su eliminación.

- Manejo de Errores y Respuestas Seguras:
  - El sistema debe garantizar que se manejen de manera segura todos los errores y excepciones. No debe proporcionar información sensible en los mensajes de error que pueda ser utilizada por atacantes para comprometer el sistema.

CONCLUSIONES

- El sistema está diseñado para cumplir tanto con los requerimientos funcionales (gestionar eventos académicos) como con los no funcionales (seguridad, escalabilidad, usabilidad y rendimiento). El uso de tecnologías como .NET Core y MongoDB permite crear un sistema ágil, escalable y robusto.
- Se utilizarán principios de diseño RESTful para la API, asegurando que las interacciones sean intuitivas y eficientes. La implementación de GitFlow y GitHub Actions asegura un flujo de trabajo eficiente en el desarrollo y despliegue continuo del proyecto.
- El proyecto enfatiza la importancia de seguridad mediante el uso de mecanismos de autenticación robustos como JWT, así como el cumplimiento de normativas de protección de datos como GDPR y la Ley de Protección de Datos Personales en Perú.
- La integración de Docker y DockerHub facilita la automatización del despliegue, lo que asegura que el producto sea fácilmente distribuible y escalable en entornos de producción, alineado con los estándares de la plataforma.
- La solución proporciona una interfaz accesible a través de Swagger UI, permitiendo una fácil integración para los desarrolladores y usuarios interesados en interactuar con la API. La solución se despliega en plataformas en la nube, permitiendo una alta disponibilidad.

RECOMENDACIONES

- Se recomienda realizar pruebas unitarias e integradas para validar cada funcionalidad de la API, garantizando que todas las operaciones CRUD funcionen correctamente antes de poner el sistema en producción.
- Se debe establecer un sistema de monitoreo para observar el desempeño de la API y detectar posibles problemas en tiempo real. Las actualizaciones periódicas y el mantenimiento de las dependencias de software son cruciales para mantener el sistema seguro y funcional.
- Aunque la arquitectura está diseñada para ser escalable, es importante realizar pruebas de carga para verificar el rendimiento del sistema bajo condiciones de alta demanda. La capacidad de escalar horizontalmente utilizando contenedores Docker debe ser evaluada y ajustada según las necesidades.
- La documentación del proyecto debe ser continua y mejorada, tanto para desarrolladores como para usuarios finales, asegurando que todos los aspectos del sistema estén claramente explicados y sean fácilmente accesibles.
- Es crucial que el equipo de desarrollo se mantenga actualizado con las normativas de protección de datos y las leyes locales o internacionales aplicables. Esto incluye revisar los procesos de recolección y almacenamiento de datos personales, y realizar auditorías de seguridad periódicas.

BIBLIOGRAFIA

- Microsoft. .NET Documentation. https://learn.microsoft.com/en-us/dotnet/
- MongoDB, Inc. MongoDB Documentation. https://www.mongodb.com/docs/
- European Commission. General Data Protection Regulation (GDPR). https://gdpr.eu/
- PERÚ. Ley N° 29733 - Ley de Protección de Datos Personales. https://www.gob.pe/institucion/jne/normas-legales/31699-ley-29733

- Docker, Inc. Docker Documentation. https://docs.docker.com/
- GitHub, Inc. GitHub Actions Documentation. https://docs.github.com/en/actions

WEBGRAFIA

- Docker Hub. Repositorio de Docker Images. https://hub.docker.com/
- Swagger. Swagger API Documentation. https://swagger.io/
- GitHub. Repositorio del Proyecto SI-8811. https://github.com/tuusuario/proyecto-si-8811.git
- MongoDB Atlas. MongoDB Atlas Cloud Database. https://www.mongodb.com/cloud/atlas
- Elastic Beanstalk Documentation. https://aws.amazon.com/es/elasticbeanstalk/