UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Informe Final

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	GVB	JJC	JJC	24/11/2024	Versión Original

INDICE GENERAL

1. ANTECEDENTES

El proyecto "Apis y Funciones Jarro_Valle" surge de la necesidad de optimizar la gestión de eventos académicos en instituciones educativas. Muchas de estas instituciones dependen de métodos manuales o herramientas fragmentadas que resultan en ineficiencias, errores humanos y dificultades para la centralización de la información. La implementación de tecnologías modernas como .NET Core, MongoDB y Docker busca resolver estas problemáticas proporcionando una solución automatizada, escalable y segura.

El contexto académico, caracterizado por un flujo constante de actividades y eventos, demanda soluciones tecnológicas que aseguren accesibilidad, fiabilidad y una integración sencilla con plataformas existentes. Este proyecto fue diseñado para abordar estas necesidades mediante la creación de una API robusta que facilite la gestión de eventos académicos, desde su creación hasta la evaluación de resultados.

2. Planteamiento del Problema

2.1 Problema

Las instituciones educativas enfrentan múltiples desafíos en la gestión de eventos académicos:

- Uso de herramientas manuales o fragmentadas.
- Ausencia de un sistema centralizado para el manejo de datos.
- Dificultad para acceder a información en tiempo real.
- Procesos manuales propensos a errores y que consumen recursos.

2.2 Justificación

Este proyecto es esencial para modernizar la gestión de eventos académicos, reducir la carga administrativa y asegurar la escalabilidad del sistema. Además, fomenta una toma de decisiones más eficiente mediante el acceso a datos centralizados y en tiempo real.

2.3 Alcance

- Implementación de una API RESTful para operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- Uso de MongoDB para el almacenamiento de datos.
- Contenerización con Docker para despliegue en múltiples entornos.
- Documentación interactiva con Swagger UI.
- Automatización del flujo de trabajo con GitHub Actions.

3. Objetivos

3.1 Objetivo General

- Desarrollar una API escalable, eficiente y segura para la gestión de eventos académicos.

3.2 Objetivos Específicos

- Diseñar e implementar un modelo de base de datos en MongoDB.
- Crear endpoints funcionales para la gestión de eventos.
- Automatizar pruebas y despliegues con GitHub Actions.
- Proveer documentación clara y accesible con Swagger UI.
- Garantizar la portabilidad mediante Docker.

4. Marco Teórico

4.1 API RESTful

Una API RESTful (Representational State Transfer) es una interfaz que permite la comunicación entre diferentes sistemas mediante el uso de solicitudes HTTP estándar como GET, POST, PUT y DELETE. Las APIs RESTful se han convertido en un estándar en el desarrollo moderno debido a:

4.1.1 Flexibilidad: Permite el consumo desde aplicaciones web, móviles y de escritorio.

4.1.2 Independencia de plataforma: La comunicación está basada en protocolos HTTP, ampliamente soportados.

4.1.3 Escalabilidad: Diseñadas para manejar grandes volúmenes de tráfico.

4.1.4 En este proyecto, la API permite gestionar eventos académicos mediante operaciones CRUD, integrándose de manera eficiente con otras plataformas existentes.

4.2 MongoDB

MongoDB es una base de datos NoSQL orientada a documentos. Sus características la hacen ideal para aplicaciones que requieren flexibilidad y escalabilidad:

4.2.1 Modelo de datos flexible: Utiliza documentos JSON para representar datos, lo que facilita adaptaciones futuras.

4.2.2 Rendimiento: Optimizada para manejar grandes cantidades de datos con acceso rápido.

4.2.3 Escalabilidad horizontal: Diseñada para distribuir datos en múltiples servidores.

4.2.4 MongoDB se utiliza en este proyecto para almacenar datos relacionados con los eventos, permitiendo consultas rápidas y estructuración adaptable.

4.3 Contenerización con Docker

Docker es una tecnología de virtualización ligera que permite empaquetar aplicaciones junto con todas sus dependencias en contenedores. Esto garantiza que el sistema sea:

4.3.1 Portátil: Los contenedores pueden ejecutarse en cualquier sistema compatible con Docker.

4.3.2 Consistente: Evita problemas de configuración entre entornos de desarrollo y producción.

4.3.3 Escalable: Permite la ejecución de múltiples instancias de la aplicación en paralelo.

4.4 CI/CD con GitHub Actions

La integración continua (CI) y el despliegue continuo (CD) son prácticas que aseguran un flujo de desarrollo eficiente. GitHub Actions automatiza tareas como:

4.4.1 Compilación del código.

4.4.2 Ejecución de pruebas unitarias.

4.4.3 Generación y despliegue de contenedores Docker.

4.4.4 Esto minimiza errores, reduce tiempos de despliegue y garantiza la calidad del software.

4.5 Documentación interactiva con Swagger UI

Swagger UI es una herramienta para documentar y probar APIs. Ofrece:

4.5.1 Visualización interactiva: Los usuarios pueden explorar y probar los endpoints de la API.

4.5.2 Estándar de la industria: Facilita la comprensión de la API por parte de desarrolladores externos.

4.5.3 Pruebas rápidas: Permite probar funcionalidades sin necesidad de herramientas externas.

5. Desarrollo de la Solución

5.1 Análisis de Factibilidad

5.1.1 Factibilidad Técnica

El proyecto es viable desde el punto de vista técnico por los siguientes motivos:

- Tecnologías seleccionadas: Se emplean herramientas modernas y ampliamente utilizadas como .NET Core, MongoDB y Docker, las cuales tienen documentación extensa y comunidades activas.
- Requerimientos de hardware: Los servidores requeridos son accesibles, con especificaciones básicas (CPU de 4 núcleos, 8 GB de RAM y 50 GB de almacenamiento SSD), y pueden escalarse fácilmente mediante servicios en la nube.
- Compatibilidad: Las tecnologías seleccionadas funcionan bien juntas; Docker garantiza que la API se implemente de forma consistente en diferentes entornos.

- Flexibilidad y escalabilidad: MongoDB y Docker permiten manejar un creciente volumen de datos y usuarios sin complicaciones técnicas.

5.1.2 Factibilidad Económica

El costo del proyecto se mantiene bajo gracias al uso de herramientas de código abierto y estrategias de optimización:

- Licencias: Se evita el uso de software comercial costoso mediante la adopción de tecnologías gratuitas o de bajo costo como MongoDB Community Edition y Docker.
- Desglose de costos:
- Infraestructura: S/. 1,500.
- Desarrollo y personal: S/. 6,000.
- Capacitación y documentación: S/. 1,000.
- Total estimado: S/. 8,500.
- Relación costo-beneficio: Se espera que los beneficios superen significativamente los costos iniciales.

5.1.3 Factibilidad Operativa

- Usuarios finales: Administradores, docentes y estudiantes podrán adaptarse rápidamente gracias a la interfaz amigable y la documentación interactiva con Swagger UI.
- Automatización: La implementación de CI/CD con GitHub Actions asegura procesos automatizados y simplifica las actualizaciones futuras.
- Soporte técnico: El equipo de TI cuenta con la capacidad para mantener el sistema operando, apoyado por manuales y guías desarrolladas durante el proyecto.

5.1.4 Factibilidad Social

- Mejora la comunicación y organización de eventos académicos.
- Incrementa la confianza de los usuarios al centralizar y digitalizar la información.
- Contribuye a la alfabetización tecnológica dentro de las instituciones educativas.

5.1.5 Factibilidad Legal

- Protección de datos: Se implementan medidas de seguridad básica, como autenticación para proteger datos sensibles.
- Licencias: Las herramientas empleadas tienen licencias compatibles con el propósito del proyecto (por ejemplo, licencia MIT de .NET Core).
- No se identificaron restricciones legales relacionadas con su implementación o uso.

5.1.6 Factibilidad Ambiental

- Reducción del uso de papel: Al digitalizar los procesos, disminuye la dependencia de documentos impresos.
- Uso responsable de recursos: El diseño escalable minimiza el consumo de energía al optimizar los recursos tecnológicos.

5.2 Tecnología de Desarrollo

Herramientas seleccionadas:

5.2.1 Backend: .NET Core (versión 6.0 o superior), por su robustez, velocidad y soporte para aplicaciones modernas.

5.2.2 Base de datos: MongoDB, por su flexibilidad para almacenar datos no estructurados y escalabilidad horizontal.

5.2.3 Contenerización: Docker, para asegurar que la API sea portable y consistente en cualquier entorno.

5.2.4 Automatización CI/CD: GitHub Actions, para integrar y desplegar automáticamente el sistema con pruebas y validaciones.

5.2.5 Documentación interactiva: Swagger UI, que permite explorar y probar los endpoints de la API de manera visual e intuitiva.

5.2.6 Compatibilidad y dependencias:

- Sistemas operativos: El proyecto es compatible con Windows Server y distribuciones Linux como Ubuntu.
- Navegadores web: Funciona en Chrome, Edge y Firefox, asegurando acceso universal a los puntos finales de la API.

5.3 Metodología de Implementación

5.3.1 Enfoque metodológico

Se sigue un enfoque iterativo basado en metodologías ágiles, combinando fases de diseño, desarrollo y pruebas de manera continua para garantizar la calidad del producto.

5.3.2 Fases del proyecto:

5.3.2.1 Análisis y levantamiento de requerimientos:

- Identificar necesidades clave de usuarios y documentarlas en el SRS (Especificación de Requerimientos de Software).
- Establecer prioridades entre los requerimientos funcionales y no funcionales.

5.3.2.2 Diseño:

- Crear un modelo de datos en MongoDB alineado con los objetivos del proyecto.
- Documentar la arquitectura del sistema en el SAD (Documento de Arquitectura de Software), incluyendo vistas lógicas, físicas y de despliegue.

5.3.2.3 Desarrollo:

- Implementar endpoints CRUD en .NET Core para gestionar eventos.
- Integrar autenticación básica para proteger las rutas críticas.
- Configurar Docker para contenerizar la API y MongoDB.

5.3.2.4 Pruebas:

- Realizar pruebas unitarias y de integración automatizadas utilizando GitHub Actions.
- Validar la funcionalidad y rendimiento en escenarios simulados.

5.3.2.5 Despliegue:

- Implementar la API en un entorno de producción utilizando Docker y Elastic Beanstalk (opcional para nube).
- Publicar documentación interactiva con Swagger UI.

5.3.3 Capacitación y soporte:

- Crear manuales de usuario y guías técnicas.
- Capacitar a los usuarios finales en el uso del sistema y sus funcionalidades.

6. Cronograma

Semana	Actividad
1	Levantamiento de requerimientos.
2-3	Diseño del modelo de datos en MongoDB.
4-5	Desarrollo de endpoints CRUD en .NET Core.
6-7	Configuración de Docker y pruebas iniciales.
8	Implementación de CI/CD con GitHub Actions.
9	Documentación de la API con Swagger UI.
10	Pruebas unitarias y validación final.
11	Despliegue y presentación del sistema.

7. Presupuesto

Item	Costo (PEN)
Costos Generales	3876.00
Costos Operativos	950.00
Costos del Ambiente	437.00
Costos de Personal	3800.00
Total	9063.00

8. Conclusiones

- El sistema propuesto resuelve las limitaciones actuales en la gestión de eventos.
- El uso de tecnologías modernas garantiza escalabilidad y eficiencia.
- La integración de Docker y GitHub Actions asegura despliegues consistentes y automáticos.

Recomendaciones

- Implementar medidas de seguridad avanzadas en futuras versiones.
- Evaluar el uso de servicios en la nube para una mayor escalabilidad.
- Realizar sesiones de capacitación para maximizar el uso del sistema.

Bibliografía

- MongoDB Documentation: https://www.mongodb.com/docs/
- .NET Core Documentation: https://learn.microsoft.com/en-us/dotnet/core/

- Docker Documentation: https://docs.docker.com/
- Swagger UI Documentation: https://swagger.io/tools/swagger-ui/

Anexos

Informe de Factibilidad

- Descripción del Proyecto: Desarrollar una API para la gestión eficiente de eventos académicos con CRUD usando .NET Core, MongoDB y Docker.
- Estudio de Factibilidad:
    - Técnica: Viable por el uso de tecnologías modernas y ampliamente soportadas.
    - Económica: Uso de herramientas de código abierto, con un costo total estimado de S/. 9,063.
    - Operativa: Garantiza automatización, escalabilidad y mantenimiento sencillo.
    - Legal, Social y Ambiental: Cumple con normativas locales y promueve la digitalización.
- Riesgos Identificados: Seguridad, cronograma, infraestructura y aceptación del usuario final.
- Análisis Financiero: Relación beneficio/costo favorable (1.54).

Anex0 02 Documento de Visión

- Propósito y Alcance: Desarrollar una API escalable y accesible para gestionar eventos académicos mediante operaciones CRUD y contenedores Docker.
- Posicionamiento: Resolver problemas de gestión manual de eventos mediante una solución automatizada, centralizada y fácilmente integrable.
- Usuarios e Interesados:
    - Usuarios: Administradores de eventos, docentes, estudiantes y desarrolladores.
    - Interesados: Instituciones educativas, equipos técnicos y proveedores de tecnología.
- Capacidades del Producto:
    - Gestión de eventos, despliegue con Docker, documentación con Swagger y soporte para integración continua.
    - Suposiciones y Dependencias: Requiere infraestructura para Docker y MongoDB, así como conexión a internet adecuada.

Anexo 03 Documento SRS

- Objetivos: Detallar requisitos para desarrollar un sistema eficiente y centralizado de gestión de eventos.
- Problemas Detectados: Procesos manuales, comunicación dispersa y falta de integración tecnológica.
- Requisitos del Sistema:

- Funcionales: CRUD para eventos, generación de reportes automáticos y seguimiento en tiempo real.
- No funcionales: Escalabilidad, accesibilidad y seguridad.
- Reglas de Negocio: Definir políticas para participación y evaluación, asegurando inscripción y retroalimentación en tiempos establecidos.
- Análisis de Procesos: Propone automatizar la creación, seguimiento y evaluación de eventos.

Anexo 04 Documento SAD

- Propósito y Alcance: Describir la arquitectura de la API de gestión de eventos académicos desarrollada en C# .NET Framework con MongoDB. Incluye vistas lógicas y físicas del sistema, modelos de datos y componentes principales.
- Objetivos y Restricciones:
  - Permitir operaciones CRUD para eventos.
  - Cumplir con atributos de calidad como tiempo de respuesta (<500 ms) y soporte para 10,000 registros.
  - Usar Docker para despliegue y autenticación básica en rutas críticas.
- Representación del Sistema:
  - Diagramas de casos de uso, clases, bases de datos, secuencia, colaboración y despliegue.
- Atributos de Calidad:
  - Escenarios enfocados en funcionalidad, usabilidad (Swagger para documentación), confiabilidad (registros de errores) y rendimiento.