UISEK
UNIVERSIDAD INTERNACIONAL SEK

Ingeniería en Informática

Desarrollo de Aplicaciones Web

Ruth Sánchez

APE Nº17 Proyecto Final

Jhelan Basantes, Sophia Chuquillangui,

Esteban Guaña, Arely Pazmiño

22/07/2025

Introducción

Esta propuesta presenta el diseño e implementación de una plataforma web para turismo comunitario en Ecuador. Este sistema busca vincular turistas con comunidades rurales y nativas que ofrecen experiencias culturales y naturales únicas. A través de una interfaz intuitiva y segura, las comunidades podrán autogestionar sus ofertas, mientras los visitantes podrán buscar, reservar y valorar experiencias.

El sistema está diseñado para ser escalable, modular y mantenible, haciendo uso de tecnologías modernas tanto en el frontend como en el backend. Los beneficios esperados incluyen mayor visibilidad de los destinos rurales, fortalecimiento de la economía local, inclusión digital y fomento de un turismo sostenible y consciente.

Desarrollo

1. Links de Interés

- **Contextualización**
- **Mapa de Empatía**
- **Diagramas de Casos de Uso**
- **Diagrama Diseño de Arquitectura**

2. Objetivos del Proyecto

- Diseñar e implementar una solución web escalable y segura para el turismo comunitario.
- Brindar a las comunidades una herramienta para gestionar experiencias, reservas y retroalimentación.
- Facilitar a los turistas una navegación sencilla, filtros de búsqueda y un proceso de reserva optimizado.
- Garantizar integridad, autenticidad y disponibilidad de los datos.

3. Alcance del Proyecto

3.1 Funcionalidades que se desarrollarán:

- Registro e inicio de sesión para turistas y comunidades anfitrionas.
- Publicación y edición de experiencias turísticas.
- Exploración de experiencias mediante búsqueda y filtros.
- Reservas de experiencias con confirmación por parte de la comunidad.
- Valoración y retroalimentación de experiencias.
- Paneles de control diferenciados por rol.

3.2 Funcionalidades fuera de alcance:

- Integración de pagos en línea.
- Aplicaciones móviles nativas.
- Traducción automática multilingüe.

4. Arquitectura y tecnologías propuestas

El sistema se hizo inicialmente con una arquitectura monolítica basada en MVC, lo que implicaba que tanto la lógica de negocio como las vistas y la manipulación de datos estaban acopladas dentro de un solo proyecto. Este modelo resultaba limitado para equipos de trabajo colaborativo, dificultaba la escalabilidad del sistema y generaba redundancias al momento de actualizar componentes.

Para solventar estas limitaciones, el sistema fue migrado a una arquitectura desacoplada, implementando un patrón cliente-servidor moderno. Ahora el sistema se compone de una API RESTful desarrollada en ASP.NET Core que se comunica con una SPA (Single Page Application) construida en React. Esta separación no solo mejora el rendimiento y la escalabilidad, sino que permite una mejor gestión de versiones y un mantenimiento más eficiente del código fuente.

4.1 Estructura actual:

- **Frontend:** React como SPA + MUI
- **Backend:** ASP.NET Core con API REST.
- **Base de datos:** SQL Server utilizando Entity Framework Core.

Esta arquitectura permite separar completamente la lógica de presentación del backend, facilitando la escalabilidad y el trabajo colaborativo. La comunicación entre capas se realiza mediante JSON por HTTP.

4.2. Tecnologías Propuestas

4.2.1. Lenguajes y frameworks:

- **Frontend:** React, Vite, HTML5, CSS3, MUI.
- **Backend:** ASP.NET Core 8.0

4.2.2. Infraestructura:

- **Base de datos:** SQL Server
- **ORM:** Entity Framework Core

4.2.3. Librerías y Herramientas adicionales:

- **Pruebas:** Postman, Swagger.
- **Control de versiones:** GitHub
- **Estilos**: Librerías MUI
    - @mui/material
    - @emotion/react
    - @emotion/styled
    - @mui/icons-material
    - @mui/lab

Actualmente, el sistema se encuentra desplegado y ejecutándose en un entorno local, tanto para el frontend como para el backend. Esta configuración ha permitido realizar pruebas funcionales, validaciones de flujo y ajustes de diseño durante la etapa de desarrollo.

Se ha contemplado la migración futura a un servidor web o plataforma en la nube, lo que permitirá ofrecer disponibilidad pública y un acceso más eficiente para los usuarios finales. Gracias a su arquitectura desacoplada, el sistema puede escalar fácilmente y manejar frontend y backend en entornos separados.

4.2.4. Seguridad

- **JWT:** para autenticación de usuarios en sesiones seguras.
- **Bcrypt:** para hashing de contraseñas, evitando que sean visibles.
- **HTTPS:** toda comunicación se realiza mediante protocolo seguro.
- **Roles:** control de acceso a rutas protegidas.
- **Validación cruzada:** datos validados en frontend y backend.

Se mejoró el manejo de contraseñas respecto a versiones anteriores, cumpliendo con estándares de seguridad. Se utiliza el algoritmo BCrypt para aplicar hashing a las contraseñas, impidiendo su exposición directa. Además, la autenticación de los usuarios se maneja mediante tokens seguros JWT y todas las comunicaciones se realizan bajo el protocolo HTTPS, cumpliendo con las buenas prácticas de protección de datos personales.

	Id	Username	PasswordHash	Email	Role	FechaRegistro	Wishlist
1	1	admin	HASHED_PASSWORD_ADMIN	admin@turismo.com	Administrador	2025-07-06 23:29:21.717	NULL
2	2	guia1	HASHED_PASSWORD_GUIA	guia1@turismo.com	Guia	2025-07-06 23:29:21.717	NULL
3	3	turista1	HASHED_PASSWORD_TURISTA	turista1@turismo.com	Turista	2025-07-06 23:29:21.717	NULL
4	4	Prueba1	$2a$11$XE0mQKFgwRlWHxCbW2dZsOil9GlbF2Xs9QGj84LOre...	prueba1@example.com	Turista	2025-07-07 09:42:05.017	NULL
5	5	Sophie	$2a$11$Mr6eEciLToKVVViBGM6Q8ewY1T98Bd5IpMhX5YeMw...	s@hotmail.com	Guia	2025-07-10 11:18:08.490	[4,8,9]
6	6	ArelyPaz	$2a$11$5KYNGv1ZiNDmV2lcLT6dn./02FiytETUepZUpghtgOe7G...	a.p.m@hotmail.com	Guia	2025-07-14 02:08:49.090	NULL

***Imagen**: DB con usuarios y sus contraseñas encriptadas mediante BCrypt.*

5. Metodología de desarrollo

Se usó Scrum como marco ágil de desarrollo para este proyecto, dividiendo el trabajo por roles y en ciclos cortos: *Sprints*, con entregas funcionales incrementales, cada uno con duración de cuatro días. También se usaron herramientas como GitHub para control de versiones, Postman y Swagger para documentación y pruebas de APIs, y reuniones periódicas para control de

avances y resolución de obstáculos, además de Notion como workspace compartido para gestionar tareas y plazos.

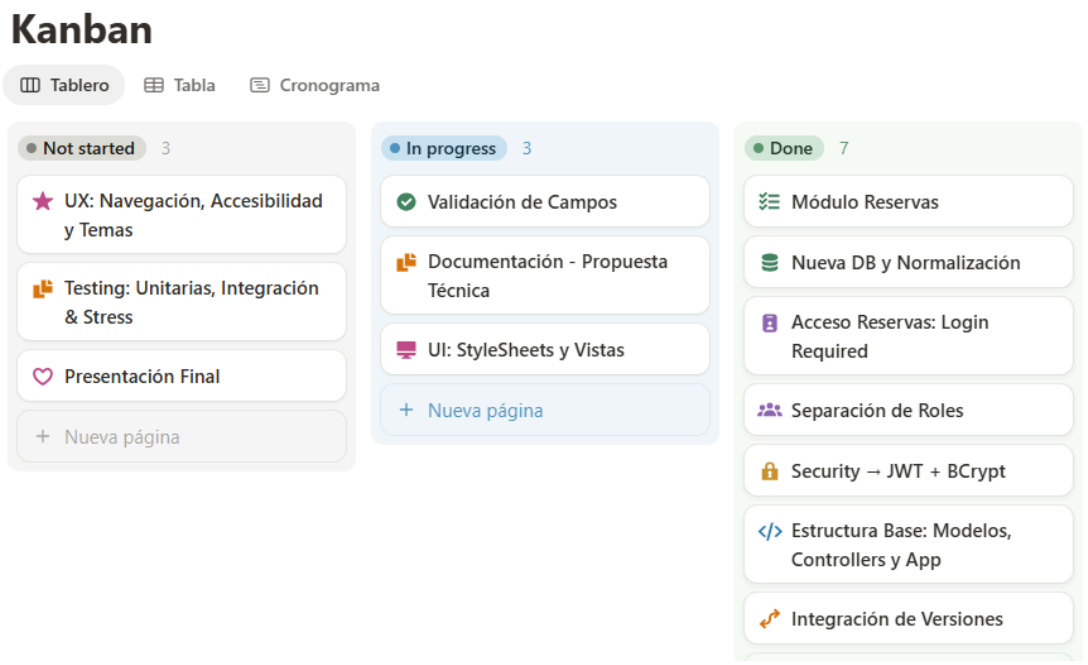
5.1 Roles asignados:

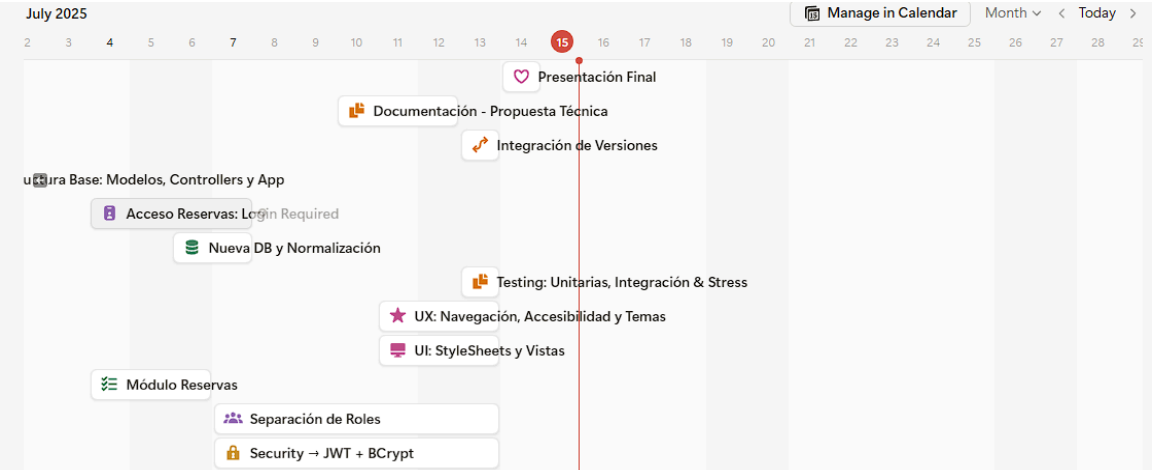
- **Product Owner:** define requerimientos funcionales.
- **Scrum Master:** facilita el proceso y elimina obstáculos.
- **Equipo de desarrollo:** dividido en frontend y backend.
- **QA Tester:** diseña y ejecuta las pruebas.

ROLES & RESPONSABILIDADES:

Member	Tasks	
Jhelan Basantes	Backend dev	Master
Sophie Chuquillangui	Frontend dev	DEV
Esteban Guaña	Backend dev	DEV
Arely Pazmiño	Backend dev	DEV
DAW - DC - SQA	Revisión - Feedback & Seguimiento	PO

5.2. Workspace & Herramientas: Tablero Kanban y Timeline





6. Cronograma con hitos

Semana	Actividad
1	Análisis y diseño del proyecto: investigación de proyectos similares, levantamiento de requerimientos, elaboración de bocetos, definición de estructura de base de datos y selección de tecnologías (React, ASP.NET Core, SQL Server).
2	Desarrollo inicial del frontend: implementación de las primeras interfaces en React, estructura de navegación, integración de formularios y funcionalidad de búsqueda de experiencias en el catálogo.
3	Creación del backend (API RESTful): desarrollo de la base de datos, construcción de la API con ASP.NET Core Web API, conexión con Entity Framework Core y visualización de datos en el frontend. Se establecieron roles de usuario (turista y administrador).
4	Desarrollo de nuevas secciones: integración del módulo de reservas, vista de historial de reservas, detalles ampliados en el catálogo de experiencias y lógica básica de confirmación de reservas.
5	Integración y validación de flujos: conexión total entre frontend y backend, pruebas con Postman y desde la interfaz, validación de los flujos de inicio de sesión, publicación y reserva.

6	Agregación de nuevas funcionalidades: se añadieron nuevos formularios, mejoras en el control de roles, validaciones adicionales en backend, e implementación del sistema de valoraciones por parte del usuario.
7	Ajustes visuales y de estilo: se mejoró el diseño de la interfaz, estructura de los componentes, uso de colores, alineación, espaciado, y se refinaron los mensajes de error y confirmación para el usuario.
8	Documentación técnica: elaboración del informe del sistema, catálogo de endpoints, flujos de navegación por rol, validaciones aplicadas y pruebas realizadas.Y preparación de la exposición final del proyecto

7. Equipo de trabajo

Nombre	Rol	Responsabilidades principales
Sophia Nicole Chuquillangui Villareal	Desarrolladora Frontend Principal	Encargada del desarrollo del Frontend en React
Arely Carolina Pazmiño Moreta	Documentación y apoyo en Frontend	Encargada de la documentación técnica y responsable del diseño e implementación de interfaces con React
Esteban Fernando Guaña Toapanta	Documentación y apoyo en Backend	Encargado de documentación técnica y colaboración en validaciones del backend
Jhelan Israel Basantes Jacome	Desarrollador Backend Principal	Encargado del desarrollo completo del backend en ASP.NET Core Web API

8. Riesgos cuantitativos y cualitativos

La gestión de riesgos es fundamental en proyectos de software. Según los principios de SQA (Software Quality Assurance), se han identificado riesgos cuantitativos como cualitativos que podrían comprometer el éxito del sistema.

Riesgos cualitativos	Riesgos cuantitativos
Falta de adaptación cultural y lingüística	Sobrecarga de la Base de Datos (Escalabilidad)
Interacción limitada con tecnologías inteligentes.	Retraso en la Implementación de Seguridad (JWT / BCrypt)
Promoción basada principalmente en imgenes y videos sin textos alternativos	Costo Elevado en Mantenimiento Futuro
Poca información sobre destinos, eventos o servicios, daña la credibilidad y utilidad del sitio	Desvío de Tiempos por Pruebas Insuficientes
Desajuste con necesidades de usuarios diversos	Costo de Actualización de Tecnología

9. Catálogo de endpoints

Módulo	Método	Endpoint	Descripción	Usuario/Rol
Auth	POST	/api/Auth/register	Registra un nuevo usuario	Público
	POST	/api/Auth/login	Inicia sesión y devuelve un token JWT	Público
Usuarios	GET	/api/Usuarios	Lista todos los usuarios registrados	Admin
	GET	/api/Usuarios/{id}	Consulta un usuario específico por ID	Usuario autenticado
	POST	/api/Usuarios	Crea un nuevo usuario manualmente	Público

	PUT	/api/Usuarios/{id}	Edita la información de un usuario	Usuario autenticado
	DELETE	/api/Usuarios/{id}	Elimina un usuario registrado	Admin
	PUT	/api/Usuarios/{id}/wishlist	Actualiza la lista de deseos (wishlist) del usuario	Usuario autenticado
Lugares	GET	/api/Lugares	Lista todas las experiencias disponibles	Público
	GET	/api/Lugares/{id}	Consulta los detalles de una experiencia específica	Público
	POST	/api/Lugares	Publica una nueva experiencia turística	Comunidad anfitriona
	PUT	/api/Lugares/{id}	Edita los datos de una experiencia publicada	Comunidad anfitriona
	DELETE	/api/Lugares/{id}	Elimina una experiencia turística	Comunidad anfitriona
Reservas	GET	/api/Reservas	Lista todas las reservas registradas	Admin
	GET	/api/Reservas/{id}	Consulta una reserva por ID	Usuario autenticado

	POST	/api/Reservas	Crea una nueva reserva para una experiencia	Turista
	PUT	/api/Reservas/{id}	Edita una reserva existente	Turista
	DELETE	/api/Reservas/{id}	Elimina una reserva y sus pagos asociados	Usuario autorizado
Pagos	GET	/api/Pagos	Lista todos los pagos realizados	Admin
	GET	/api/Pagos/{id}	Consulta un pago específico por ID	Usuario autenticado
	POST	/api/Pagos	Registra un nuevo pago asociado a una reserva	Sistema
	PUT	/api/Pagos/{id}	Edita un pago existente	Admin / Sistema
	DELETE	/api/Pagos/{id}	Elimina un pago registrado	Admin
Opiniones	GET	/api/Opiniones	Lista todas las opiniones de los usuarios	Público
	GET	/api/Opiniones/{id}	Consulta una opinión específica	Usuario autenticado
	POST	/api/Opiniones	Crea una nueva opinión sobre un lugar visitado	Turista
	PUT	/api/Opiniones/{id}	Edita una opinión ya registrada	Usuario

	DELETE	/api/Opiniones/{id}	Elimina una opinión específica	Usuario
	GET	/api/Opiniones/promedios	Devuelve el promedio de calificaciones por lugar	Público

10. Validaciones

El sistema implementa validaciones tanto en el frontend como en el backend para garantizar la integridad de los datos, la experiencia del usuario y la seguridad de la información.

10.1 Validaciones del sistema

El sistema implementa validaciones tanto en el frontend como en el backend para garantizar la integridad de los datos, la experiencia del usuario y la seguridad de la información.

Se han definido límites para ciertos campos del sistema. Por ejemplo, las descripciones de experiencias están limitadas a un máximo de 200 caracteres y el sistema restringe la cantidad de imágenes por publicación para optimizar el tiempo de carga. Estas medidas aseguran un rendimiento adecuado y previenen errores de visualización o almacenamiento innecesario.

10.2 Validaciones en el Frontend (React):

- Validaciones básicas con HTML5: campos requeridos, tipo de dato, límites mínimos/máximos.

- Validaciones con expresiones regulares para campos como correo electrónico, número de teléfono y cédula.
- Validación de campos en tiempo real mediante React Hooks.
- Evita el envío de formularios si existen errores visibles.

10.3 Validaciones en el Backend (ASP.NET Core Web API):

- Se utilizan atributos como **[Required], [StringLength], [EmailAddress], [Range]**, entre otros.
- Validaciones lógicas, tales como:
    - Control de acceso según el rol del usuario autenticado.
    - Verificar que los cupos disponibles sean mayores que cero.
    - Asegurar que solo el usuario autorizado pueda editar sus experiencias.
    - Asegurar que los cupos disponibles no sean negativos o cero.

10.4 Tipos de testing aplicados

Durante el desarrollo se aplicaron diferentes tipos de pruebas para garantizar el funcionamiento correcto del sistema. Las tres principales fases de testing seleccionadas fueron:

Tipo de testing	Aplicación en el proyecto
Caja negra (Black Box)	Se verificó que los módulos funcionen correctamente sin revisar el código fuente. Por ejemplo, validar formularios, enviar reservas y registrar usuarios desde la vista del turista.
Caja blanca (White Box)	Se revisaron estructuras internas del código, condiciones lógicas y controladores en ASP.NET Core, asegurando el flujo correcto de datos.

Pruebas de integración	Se probó la conexión entre frontend y backend mediante herramientas como Postman y pruebas directas desde la interfaz, garantizando el intercambio correcto de datos.

Adicionalmente, se consideraron:

- Pruebas funcionales: Validación de que cada módulo cumpla su propósito (reserva, exploración, edición, etc.).
- Pruebas de estrés : Se discutió la posibilidad de realizar muchas reservas simultáneas para evaluar el rendimiento.
- Pruebas de seguridad básicas: Simulación de intentos de acceso no autorizado, modificación de roles o envío de datos inválidos.

10.5 Requisitos Funcionales

- RF01: Registro y gestión de cuentas para comunidades y turistas
- RF02: Publicación de experiencias turísticas
- RF03: Búsqueda inteligente de experiencias
- RF04: Sistema de reservas y confirmaciones
- RF05: Sistema de valoración y retroalimentación

10.6 Requisitos no Funcionales

- RNF1: Coherencia visual (UX/UI)
- RNF2: Rendimiento y velocidad de carga
- RNF3: Seguridad de datos personales
- RNF4: Usabilidad multiplataforma
- RNF5: Escalabilidad del sistema

11. Flujos de navegación

11.1 Flujo de Navegación – Turista

Este flujo representa cómo un turista interactúa con la plataforma web desde que entra hasta que califica una experiencia. Todo el recorrido está pensado para que sea intuitivo y rápido.

1. **Ingreso a la plataforma**

   El turista accede al sitio web desde cualquier dispositivo para empezar a explorar las experiencias disponibles.

2. **Exploración de experiencias**

   Puede ver todas las experiencias publicadas por las comunidades anfitrionas. Desde esta parte también puede aplicar filtros por tipo de actividad, precio o ubicación.

3. **Registro o inicio de sesión**

   Si no tiene cuenta, el sistema le permite registrarse como turista. Si ya está registrado, solo debe iniciar sesión.

4. **Reserva de experiencia**

   Una vez dentro, puede seleccionar una experiencia y reservarla. La reserva queda guardada y pendiente de ser aceptada por la comunidad anfitriona.

5. **Esperar confirmación**

   El turista debe esperar a que la comunidad anfitriona confirme la reserva. Este paso es importante para evitar sobrecupo o problemas de disponibilidad.

6. **Pago**

   Cuando la reserva es aceptada, el sistema le da acceso al módulo de pagos, donde puede realizar la transacción.

7. **Valoración**

   Después de haber asistido a la experiencia, el turista puede calificarla y dejar una opinión. Esto ayuda a otros usuarios y también a mejorar el servicio.

8. **Fin del proceso**

   El flujo del turista termina una vez que deja su calificación. A partir de ahí puede volver a explorar o realizar nuevas reservas.

11.2 Flujo de Navegación – Comunidad Anfitriona / Administrador

Este flujo muestra cómo una comunidad anfitriona (o administrador) maneja su cuenta, pública experiencias y gestiona reservas dentro del sistema.

1. **Inicio de sesión**

   La comunidad inicia sesión con sus credenciales. Esto da acceso a todas las funciones que tiene disponibles según su rol.

2. **Panel de control**

   Desde el panel, puede ver su perfil, las experiencias que ha creado, y también las reservas pendientes que ha recibido.

3. **Gestión de experiencias**

   Puede publicar nuevas experiencias, editarlas si necesita hacer cambios, o eliminarlas si ya no están disponibles.

4. **Revisión de reservas**

   En esta parte revisa las reservas que los turistas han hecho. Cada una

muestra la información básica del turista y los detalles de la experiencia solicitada.

5. **Confirmar o rechazar reservas**

   Si la comunidad acepta la reserva, se activa el proceso de pago. Si no puede atenderla, puede rechazarla directamente desde el sistema.

6. **Ver pagos realizados**

   Una vez aceptada una reserva, la comunidad puede ver si el turista ya hizo el pago correspondiente.

7. **Consultar opiniones**

   También puede revisar las valoraciones y comentarios que han dejado los turistas sobre sus experiencias. Esto sirve como retroalimentación para mejorar el servicio ofrecido.

8. **Fin del proceso**

   Cuando la comunidad ha gestionado sus reservas, confirmado pagos y revisado valoraciones, el flujo termina. Desde ahí puede seguir ofreciendo más experiencias.

12. Aseguramiento de calidad

- Plan de Pruebas

13. Código y enlaces a repositorios

- Repositorio - GitHub
- Repositorio - Folder Drive

Conclusiones

- La arquitectura en capas aplicada en el proyecto facilitó la modularidad y mantenibilidad del sistema, permitiendo separar claramente las responsabilidades entre la capa de presentación, lógica de negocio y acceso a datos. Esto hizo posible trabajar de forma paralela en frontend y backend, así como facilitar futuras mejoras o integraciones.
- Las funcionalidades implementadas, como registro, gestión de experiencias, reservas, pagos y valoraciones, cubren adecuadamente las necesidades básicas de una plataforma de turismo comunitario, brindando una experiencia fluida y segura para turistas y comunidades anfitrionas, promoviendo la interacción y el turismo responsable.
- La definición y documentación clara de los endpoints REST, junto con las validaciones y el uso de tokens JWT para autenticación y autorización, fortalecen la seguridad y el correcto funcionamiento del sistema, evitando accesos no autorizados y garantizando la integridad de los datos.

Recomendaciones

- Mantener y actualizar regularmente la arquitectura en capas, asegurando que cada componente (frontend, backend, base de datos) siga funcionando de manera eficiente y permitiendo la incorporación de nuevas funcionalidades sin afectar la estabilidad.
- Documentar continuamente los endpoints y flujos de navegación conforme se realicen modificaciones o ampliaciones, para facilitar el trabajo de nuevos desarrolladores o equipos que puedan tomar el proyecto en el futuro.
- Establecer un plan de mantenimiento y monitoreo del sistema en producción, incluyendo revisión periódica de logs, rendimiento y seguridad, para detectar y corregir oportunamente cualquier falla o vulnerabilidad.

Referencias

- Microsoft. (2025). ASP.NET Core, an open-source web development framework | .NET. Microsoft. https://dotnet.microsoft.com/en-us/apps/aspnet
- OpenAI. (2023). ChatGPT (versión del 15 de julio) [Modelo de lenguaje de gran tamaño]. https://chat.openai.com/chat