ADR

ADR-001: Selección del Lenguaje de Programación para el Backend

<table>
<tr><td>Contexto</td><td>Se necesita definir el lenguaje de programación para el backend con el fin de iniciar el desarrollo del servidor y los servicios de negocio. El proyecto tiene una duración total de cinco meses y es desarrollado por un equipo de siete estudiantes de Ingeniería en Informática, por lo que se prioriza un lenguaje que facilite la productividad y aproveche los conocimientos previos del grupo.</td></tr>
<tr><td>Decisión</td><td>Se utiliza Python como lenguaje principal para el desarrollo del backend.</td></tr>
<tr><td>Consecuencias</td><td><table>
<tr><td>Positivas</td><td>- La mayoría del equipo posee experiencia previa en Python, lo que reduce la curva de aprendizaje y los tiempos iniciales de configuración.
- Python cuenta con una gran cantidad de librerías y frameworks, lo que permite avanzar rápidamente en el desarrollo de funcionalidades.
- La sintaxis simple y la comunidad activa facilitan el trabajo colaborativo y la resolución de problemas.</td></tr>
<tr><td>Negativas</td><td>- Python presenta un rendimiento menor que lenguajes compilados como Java, lo que podría afectar la eficiencia si el proyecto creciera significativamente.
- Su tipado dinámico puede generar errores en tiempo de ejecución, lo que requiere mantener buenas prácticas de testing para prevenirlos.</td></tr>
</table></td></tr>
<tr><td>Alternativas consideradas</td><td>Se evalúa únicamente Java, dado que junto con Python son los dos lenguajes dominados por todos los integrantes del equipo. Se descarta por requerir más tiempo de configuración y una mayor complejidad para</td></tr>
</table>

	un proyecto de corta duración.

ADR-002: Selección del Framework Backend

Contexto	Se necesita elegir un framework para desarrollar el backend que permita construir una API REST de forma ágil, flexible y compatible con Python. El proyecto tiene una duración de cinco meses y se busca minimizar la complejidad de configuración.
Decisión	Se utiliza Flask como framework principal para el desarrollo del backend.
Consecuencias	<table><tr><td>Positivas</td><td>- Permite desarrollar de forma rápida gracias a su simplicidad y flexibilidad. - Facilita la integración con bibliotecas como SQLAlchemy. - Su estructura liviana se adapta bien al tamaño y duración del proyecto.</td></tr><tr><td>Negativas</td><td>- Flask requiere definir manualmente parte de la estructura y las dependencias, a diferencia de frameworks más completos como Django. - No ofrece tantas herramientas integradas por defecto, lo que puede aumentar el código necesario para ciertas funcionalidades.</td></tr></table>
Alternativas consideradas	Se evalúa Django, pero se descarta por su mayor complejidad y porque ofrece una estructura más rígida que no se ajusta al alcance del proyecto.

ADR-003: Uso de ORM

Contexto	Se necesita una herramienta para mapear las entidades del dominio con la base de datos relacional, evitando escribir SQL manual y facilitando la mantenibilidad del código.
Decisión	Se utiliza SQLAlchemy como ORM para el manejo de la persistencia de datos.

<table>
<tr><td>Consecuencias</td><td>
<table>
<tr><td>Positivas</td><td>- Ofrece gran flexibilidad y control sobre las consultas y relaciones.
- Se integra fácilmente con Flask, simplificando el flujo de desarrollo.
- Tiene una comunidad amplia y buena documentación.</td></tr>
<tr><td>Negativas</td><td>- Requiere más configuración inicial que ORMs como Django ORM o Peewee.
- Su curva de aprendizaje es ligeramente más alta, lo que puede consumir tiempo en las etapas iniciales del desarrollo.</td></tr>
</table>
</td></tr>
<tr><td>Alternativas consideradas</td><td>Se consideran Django ORM y Peewee, pero se descartan por no integrarse tan bien con Flask o por ofrecer menor control sobre las consultas.</td></tr>
</table>

ADR-004: Gestión de Migraciones de Base de Datos

<table>
<tr><td>Contexto</td><td>Se requiere una herramienta para controlar los cambios en el esquema de la base de datos y mantener consistencia entre entornos de desarrollo.</td></tr>
<tr><td>Decisión</td><td>Se utiliza Alembic como sistema de migraciones de base de datos.</td></tr>
<tr><td>Consecuencias</td><td>
<table>
<tr><td>Positivas</td><td>- Permite aplicar y revertir cambios en el esquema fácilmente.
- Se integra nativamente con SQLAlchemy, reduciendo errores de compatibilidad.
- Facilita el trabajo colaborativo entre los desarrolladores.</td></tr>
<tr><td>Negativas</td><td>- Añade una capa adicional de configuración.
- Puede generar conflictos si no se coordinan correctamente los archivos de migración.</td></tr>
</table>
</td></tr>
</table>

Alternativas consideradas	?

ADR-005: Autenticación y Manejo de Tokens

Contexto	Se necesita implementar un sistema de autenticación que permita registrar, autenticar y manejar usuarios de forma segura sin desarrollar una solución desde cero.
Decisión	Se utiliza Amazon Cognito para la autenticación y el manejo de tokens JWT.
Consecuencias	<table><tr><td>Positivas</td><td>- Simplifica la gestión de usuarios, contraseñas y sesiones. - Aporta seguridad y cumplimiento de buenas prácticas. - Reduce significativamente el tiempo de desarrollo y mantenimiento.</td></tr><tr><td>Negativas</td><td>- Depende de un servicio externo, lo que puede dificultar la depuración en entornos locales. - Su integración inicial requiere configurar correctamente roles y permisos en AWS.</td></tr></table>
Alternativas consideradas	Se evalúa implementar un sistema propio de autenticación, pero se descarta por el costo de tiempo y el riesgo de errores de seguridad.

ADR-006: Selección de Base de Datos Relacional

Contexto	Se necesita un sistema de gestión de bases de datos relacional confiable, gratuito y compatible con SQLAlchemy.
Decisión	Se utiliza PostgreSQL como base de datos principal del sistema.
Consecuencias	<table><tr><td>Positivas</td><td>- Ofrece gran estabilidad y soporte para integridad referencial.</td></tr></table>

	<table><tr><td></td><td>- Es compatible con SQLAlchemy y ampliamente usada. - Permite escalar en caso de ser necesario.</td></tr><tr><td>Negativas</td><td>- Puede requerir configuraciones adicionales para optimizar el rendimiento en entornos locales. - Su instalación puede ser más compleja que la de alternativas más ligeras como SQLite.</td></tr></table>
Alternativas consideradas	Se considera SQLite por su simplicidad, pero se descarta por no soportar adecuadamente múltiples conexiones concurrentes.

ADR-007: Framework Móvil

Contexto	Se necesita desarrollar una aplicación móvil multiplataforma para que los usuarios puedan interactuar desde Android e iOS sin duplicar el código.
Decisión	Se utiliza React Native para el desarrollo de la aplicación móvil.
Consecuencias	<table><tr><td>Positivas</td><td>- Permite reutilizar código entre plataformas Android e iOS. - Facilita el trabajo en equipo por su similitud con React. - Dispone de una amplia comunidad y soporte de librerías.</td></tr><tr><td>Negativas</td><td>- Puede presentar limitaciones en el acceso a componentes nativos complejos. - Requiere optimización para mantener un rendimiento fluido.</td></tr></table>
Alternativas consideradas	Se considera Flutter, pero se descarta porque el equipo tiene más experiencia con JavaScript y React.

ADR-008: Uso de Expo para React Native

<table>
<tr><td>Contexto</td><td>Se necesita simplificar la configuración del entorno de desarrollo móvil y facilitar la ejecución de la aplicación en distintos dispositivos sin complicadas configuraciones nativas.</td></tr>
<tr><td>Decisión</td><td>Se utiliza Expo como herramienta de desarrollo para React Native.</td></tr>
<tr><td>Consecuencias</td><td><table>
<tr><td>Positivas</td><td>- Facilita la ejecución y prueba de la app sin configurar entornos nativos.
- Permite desplegar y testear rápidamente en distintos dispositivos.
- Simplifica la gestión de dependencias y librerías comunes.</td></tr>
<tr><td>Negativas</td><td>- Puede limitar el uso de módulos nativos avanzados no soportados por Expo.
- Dependencia de la plataforma Expo para actualizaciones.</td></tr>
</table></td></tr>
<tr><td>Alternativas consideradas</td><td>Se considera usar React Native CLI directamente, pero se descarta por requerir más tiempo de configuración y mantenimiento.</td></tr>
</table>

ADR-009: Framework Web Administrativo

<table>
<tr><td>Contexto</td><td>Se necesita una interfaz web para que los administradores gestionen los contenidos y usuarios del sistema.</td></tr>
<tr><td>Decisión</td><td>Se utiliza React para desarrollar la interfaz web del administrador.</td></tr>
<tr><td>Consecuencias</td><td><table>
<tr><td>Positivas</td><td>- Permite crear interfaces dinámicas y reutilizables.
- Es compatible con React Native, lo que facilita compartir lógica entre proyectos.
- Cuenta con una gran</td></tr>
</table></td></tr>
</table>

	<table><tr><td></td><td>comunidad y soporte.</td></tr><tr><td>Negativas</td><td>- Requiere configurar herramientas adicionales como enrutamiento o manejo de estado. - Tiene una curva de aprendizaje inicial si se desean optimizar buenas prácticas.</td></tr></table>
Alternativas consideradas	Se considera Vue.js, pero se descarta por falta de experiencia del equipo.

ADR-010: Patrón de Arquitectura

Contexto	Se necesita una estructura clara que separe la lógica de negocio, la interfaz y el manejo de datos, facilitando el trabajo colaborativo y la mantenibilidad del código.
Decisión	Se adopta el patrón Model View Controller (MVC) como base de la organización del proyecto.
Consecuencias	<table><tr><td>Positivas</td><td>- Favorece la separación de responsabilidades y el trabajo en paralelo entre backend y frontend. - Mejora la mantenibilidad y la claridad del código.</td></tr><tr><td>Negativas</td><td>- Puede requerir más estructura para un proyecto pequeño. - La división de capas introduce más archivos y coordinación entre módulos.</td></tr></table>
Alternativas consideradas	Se consideran arquitecturas orientadas a servicios (SOA) y microservicios, que ofrecen mayor escalabilidad y despliegue independiente de módulos. Sin embargo, se descartan porque su complejidad y necesidad de infraestructura adicional no se justifican en un proyecto de corta duración y con un solo equipo de desarrollo.

ADR-011: Hosting del Proyecto

<table>
<tr><td>Contexto</td><td>Se necesita definir la infraestructura de despliegue considerando los recursos disponibles y la naturaleza académica del proyecto.</td></tr>
<tr><td>Decisión</td><td>Se aloja la base de datos en AWS y se ejecuta el servidor de la aplicación en local.</td></tr>
<tr><td>Consecuencias</td><td>
<table>
<tr><td>Positivas</td><td>- AWS ofrece una base de datos accesible y gratuita con buena disponibilidad.
- Ejecutar el servidor en local facilita el control del sistema durante el desarrollo.</td></tr>
<tr><td>Negativas</td><td>- La comunicación entre el servidor local y la base remota puede generar latencia.
- No es una solución escalable a producción, ya que depende del entorno local del equipo.</td></tr>
</table>
</td></tr>
<tr><td>Alternativas consideradas</td><td>Se considera desplegar todo el sistema en AWS, pero se descarta por las limitaciones del plan gratuito y la simplicidad requerida para el entorno académico.</td></tr>
</table>