ADR Actualizados

TÍtulo	001-v2: Uso del sistema de autenticación de Supabase para el módulo de Autenticación
Estado	Adoptado
Contexto	El sistema universitario requiere mecanismos modernos y seguros de autenticación y autorización para alumnos y administrativos. Es importante simplificar la experiencia de login, minimizar complejidad operativa y habilitar acceso con cuentas institucionales de Google. El sistema debe soportar flujos de autenticación social, autorización robusta para APIs y gestión eficiente de sesiones y roles. Este ADR modifica la decisión propuesta en el ADR-001, donde inicialmente se planeó usar Better Auth.
Referencia histórica	Este ADR modifica y reemplaza la decisión propuesta en el ADR-001 (“Uso de Better Auth para el módulo de autenticación”), el cual queda obsoleto tras esta decisión
Decisión	Se adopta el sistema de autenticación de Supabase, aprovechando su soporte nativo para OAuth2 (incluyendo cuentas institucionales de Google) y JWT (para acceso seguro a las APIs). Supabase permite que los alumnos ingresen fácilmente con su cuenta de Google y cumple con los requisitos modernos de seguridad sin necesidad de integrar servicios externos complejos. Además, ofrece una gestión integrada de usuarios, sesiones y roles que se alinea con el stack JS/TS del proyecto.
Consecuencias	● Se habilita el login directo con cuentas de Google Workspace, mejorando la experiencia de usuario. ● Las sesiones y permisos en APIs se gestionan mediante JWT, lo que permite que el backend opere de forma “serverless”. ● Mejora el rendimiento y facilita la escalabilidad. ● Se reduce el esfuerzo de integración y mantenimiento en comparación con una implementación manual de OAuth2 y JWT. ● Supabase ofrece flexibilidad para incorporar futuras formas de autenticación (correo, redes sociales, 2FA). ● La solución está optimizada para entornos JS/TS; otros lenguajes pueden requerir adaptaciones adicionales. ● Se modifica la propuesta original (ADR-001) debido a la mayor integración y optimización encontradas con Supabase respecto a Better Auth.
Alternativas Consideradas	● OAuth2 sin Supabase. ● Autenticación solo con JWT. ● Auth.js / Auth0. ● Soluciones nativas PHP/Laravel

Título	002-v2: Elección de PostgreSQL con Supabase como sistema de almacenamiento de datos
Estado	Adoptado
Contexto	El backend se desarrollará en NestJS y GraphQL. Se requiere un motor de almacenamiento robusto, escalable y con soporte para APIs. Se busca optimizar la gestión de datos relacionales, reducir la complejidad operativa y facilitar la integración con servicios gestionados.
Referencia histórica	Este ADR reemplaza y modifica la propuesta documentada en el ADR-002 (“Elección de PostgreSQL y Redis como sistemas de almacenamiento de datos”).
Decisión	Se selecciona PostgreSQL como base de datos principal, utilizando Supabase como plataforma BaaS (Backend-as-a-Service) para manejar información relacional, usuarios, inscripciones, pagos, incidencias y administración del sistema. Supabase ofrece integración directa con PostgreSQL, soporte para GraphQL/REST, autenticación nativa, gestión de roles y funciones de seguridad alineadas con los requisitos del proyecto. No se utiliza Redis. Aunque en ADR-002 se consideró este motor para sesiones y cacheo, se optó por centralizar control y persistencia en Supabase/PostgreSQL, aprovechando funciones nativas, vistas materializadas y políticas de acceso.
Consecuencias	● Facilita el desarrollo con NestJS y GraphQL gracias a la compatibilidad nativa con PostgreSQL. ● Supabase reduce la carga operativa ofreciendo panel de administración, backups automáticos y reglas de seguridad. ● El rendimiento es optimizado mediante SQL avanzado y control granular, sin integrar Redis. ● Mejora la escalabilidad y mantenibilidad, eliminando la necesidad de múltiples motores de almacenamiento. ● La arquitectura depende de los límites y capacidades de Supabase, requiriendo monitoreo activo para futuras necesidades de rendimiento.
Alternativas Consideradas	● PostgreSQL autogestionado + Redis: Mayor control, más complejidad operativa. ● Bases de datos solo relacionales (MySQL, SQL Server): Menor flexibilidad con GraphQL. ● NoSQL (MongoDB): Menos adecuado para datos estructurados y relacionales. ● Firebase o BaaS similares: Menor compatibilidad con SQL y menos control sobre el modelo de datos.

ADR Anteriores

Título	001: Uso de Better-Auth para el módulo de Autenticación
Estado	Rechazado
Contexto	El sistema universitario requiere mecanismos modernos y seguros de autenticación y autorización para alumnos y administrativos. Es importante simplificar la experiencia de login, minimizar complejidad operativa y habilitar acceso con cuentas institucionales de Google. El sistema debe soportar flujos de autenticación social, autorización robusta para APIs y gestión eficiente de sesiones/roles.
Decisión	Se adopta Better Auth como solución de autenticación, aprovechando su integración directa con OAuth2 (cuentas institucionales de Google) y JWT (para acceso seguro a las APIs). Better Auth permite que los alumnos ingresen fácilmente con su cuenta de Google y cumple requisitos modernos de seguridad sin depender de servicios externos costosos. Además, facilita la gestión de roles y tipos de acceso.
Consecuencias	● Permite login directo con cuentas de Google Workspace, mejorando la experiencia de usuario. ● Las sesiones y permisos en APIs se gestionan con JWT internos, lo que permite que el backend funcione de forma “sin estado” (es decir, el servidor no necesita recordar información de cada usuario entre peticiones, ya que el token JWT contiene toda la información necesaria para identificar y autorizar cada solicitud). Esto hace que el sistema sea más rápido y fácil de escalar. ● Se reduce el esfuerzo de integración y mantenimiento versus construir desde cero solo con OAuth2. ● Mayor flexibilidad para soportar nuevas formas de login (correo, social, 2FA) según futuras necesidades. ● Depende del stack JS/TS para la integración directa; otros lenguajes requieren soluciones alternas o adaptaciones.
Alternativas Consideradas	● OAuth2 sin Better Auth: Requiere implementar y mantener manualmente la lógica de tokens, login social y gestión de sesiones, aumentando el esfuerzo y los riesgos operativos. ● Autenticación sólo con JWT: No permite login con Google ni integración social, ni autorización delegada granular. ● Auth.js/Auth0: Soluciones similares, pero con mayores restricciones de costo/licenciamiento o menor flexibilidad en “bring-your-own-backend”. ● Soluciones nativas del stack PHP/Laravel (Passport, Socialite): No cumplen los requisitos del stack JS/TS principal del rediseño del sistema universitario.

Título	002: Elección de PostgreSQL y Redis como sistemas de almacenamiento de datos
Estado	Rechazado
Contexto	El backend del sistema se desarrollará usando NestJS y GraphQL, lo que requiere motores de almacenamiento robustos, escalables y con buen soporte para APIs. Se busca optimizar la gestión de datos relacionales, acelerar transacciones y mejorar el rendimiento de consultas frecuentes mediante caché.
Decisión	Se selecciona PostgreSQL como base de datos principal para manejar información relacional (usuarios, inscripciones, pagos, incidencias) y Redis como sistema auxiliar para almacenamiento en caché y manejo rápido de sesiones y datos temporales. GraphQL se integra eficientemente con PostgreSQL, permitiendo resolver consultas complejas y relaciones entre datos de manera rápida y flexible. Redis se usará para acelerar el acceso a información que se consulta repetidamente (ej. expedientes, resultados de pagos, estados de inscripción), evitar sobrecarga en la base de datos, y gestionar sesiones o tokens.
Consecuencias	● Simplifica y agiliza el desarrollo con NestJS y GraphQL, ya que ambos se integran perfectamente con PostgreSQL para modelos relacionales. ● Redis ayuda a mantener información en memoria, haciendo que ciertas operaciones sean mucho más rápidas (por ejemplo, consultar si un pago fue confirmado o el estado de una inscripción). ● Mejora el rendimiento y escalabilidad, ya que Redis reduce presión sobre la base de datos principal y permite escalar mejor el backend. ● Depende de una correcta configuración y sincronización entre los sistemas para evitar inconsistencias.
Alternativas Consideradas	● Bases de datos solo relacionales (ej. MySQL, SQL Server): menor flexibilidad con GraphQL y menos rendimiento en caché. ● Bases de datos NoSQL como MongoDB: menos adecuadas para datos altamente estructurados y relaciones complejas entre entidades académicas. ● Usar solo PostgreSQL sin Redis: mayor presión sobre la base de datos principal y menor rendimiento en consultas intensivas.