

# **Arquitectura de Software**

1. Miranda Eugenia Colorado Arróniz A01737027
2. Estefania Antonio Villaseca A01736897
3. Alejandro Kong Montoya A01734271
4. Sofia Zugasti Delgado A00837478
5. 7 de junio de 2025
7. **Desarrollo e implantación de sistemas de software**
8. Profa. Juan Manuel González Calleros
9. **Arquitectura de software**

*1. 1 Objetivos*

**G1. Proporcionar una plataforma centralizada para la gestión de programas de servicio social.**El sistema debe unificar los procesos de búsqueda de programas, postulación, aprobación, seguimiento de actividades y generación de reportes en un solo lugar accesible tanto para estudiantes como para administradores.

**G2. Garantizar escalabilidad e integración futura con sistemas institucionales.**La arquitectura debe permitir la integración futura con sistemas de autenticación, plataformas de seguimiento académico y repositorios de documentos.

**G3. Priorizar la usabilidad para usuarios no técnicos.**La interfaz debe ser intuitiva y fácil de usar para estudiantes y administradores, requiriendo una capacitación mínima y fomentando su adopción.

**G4. Minimizar los costos operativos y de mantenimiento.**El sistema debe basarse en tecnologías open-source bien soportadas y servicios de hospedaje que sean rentables y de fácil mantenimiento.

**G5. Garantizar alta confiabilidad y consistencia de datos.**Es fundamental evitar la pérdida de datos o conflictos, especialmente en lo relacionado con el estado de las postulaciones, reportes y documentación legal.

*1.2 Restricciones*

**C1. El stack tecnológico está limitado a herramientas de código abierto o licencias educativas.**Debido a restricciones presupuestas, sólo se permite el uso de tecnologías gratuitas, de código abierto o disponibles en versiones académicas (como PostgreSQL).

**C2. El sistema debe estar desplegado y en funcionamiento al final del periodo académico.**Esto limita el tiempo de desarrollo y establece hitos claros para las fases de entrega y pruebas.

**C3. Debe cumplir con las regulaciones de protección de datos.**El sistema debe proteger la información sensible de los estudiantes conforme a los estándares legales e institucionales de privacidad y seguridad.

*1.3 Principios de la arquitectura*

| ***Referencia del Principio*** | ***AP1*** |
| --- | --- |
| ***Enunciado del Principio*** | *Usar estándares abiertos y frameworks comprobados.* |
| ***Justificación*** | *Para garantizar la mantenibilidad y el soporte de la comunidad.* |
| ***Implicaciones*** | *Elegir frameworks ampliamente adoptados (como Next.js, Tailwind) con respaldo comunitario a largo plazo.* |
| ***Información Adicional*** | *Alineado con las guías de desarrollo existentes de la institución.* |

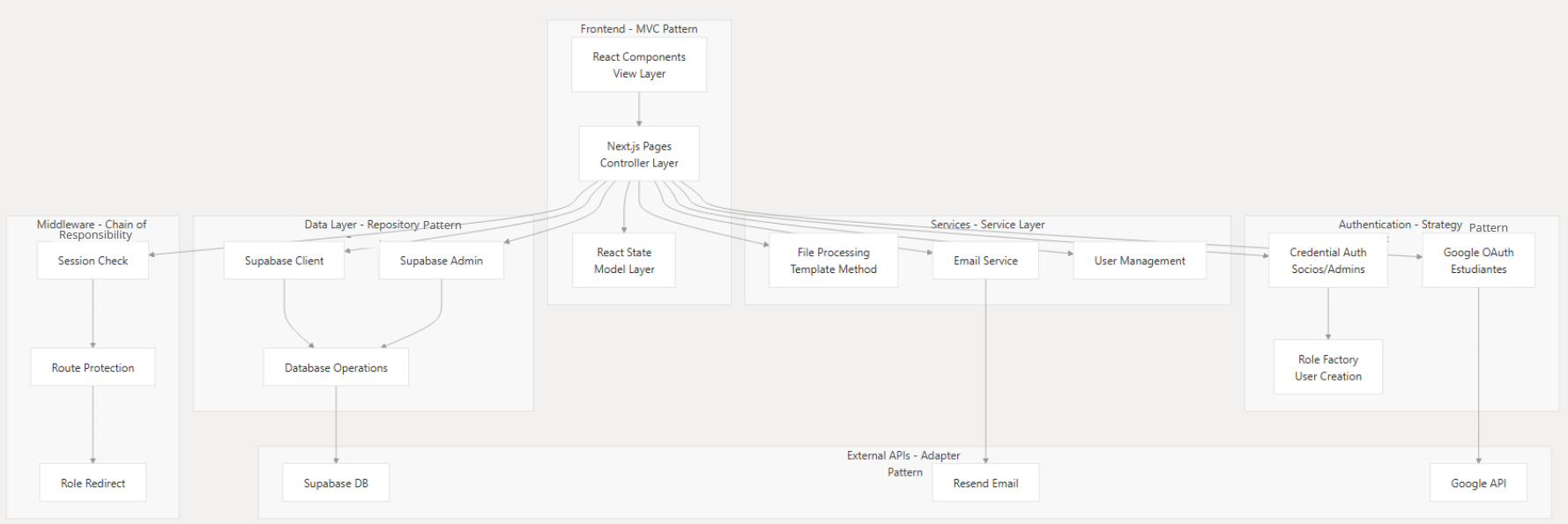
| ***Referencia del Principio*** | ***AP2*** |
| --- | --- |
| ***Enunciado del Principio*** | *Optimizar la accesibilidad y diseño responsivo.* |
| ***Justificación*** | *La plataforma será utilizada en distintos dispositivos y por diversos tipos de usuarios.* |
| ***Implicaciones*** | *El diseño debe seguir principios de diseño responsive y accesible.* |
| ***Información Adicional*** | *Considerar el uso de TailwindCSS o Material UI.* |

| ***Referencia del Principio*** | ***AP3*** |
| --- | --- |
| ***Enunciado del Principio*** | *Priorizar la seguridad y la privacidad del usuario.* |
| ***Justificación*** | *Es necesario proteger datos personales y académicos sensibles.* |
| ***Implicaciones*** | *Incluir autenticación, cifrado y capas de control de acceso.* |
| ***Información Adicional*** | *Seguir las buenas prácticas de OWASP.* |

| ***Referencia del Principio*** | ***AP4*** |
| --- | --- |
| ***Enunciado del Principio*** | *Habilitar la posibilidad de pruebas en todos los componentes.* |
| ***Justificación*** | *Para asegurar una plataforma confiable y facilitar las pruebas de regresión durante actualizaciones.* |
| ***Implicaciones*** | *Adoptar una arquitectura por capas y usar inyección de dependencias para pruebas unitarias e integradas.* |
| ***Información Adicional*** | *Utilizar herramientas que lo faciliten* |

| ***Referencia del Principio*** | ***AP5*** |
| --- | --- |
| ***Enunciado del Principio*** | *Documentar la arquitectura y las APIs de forma continua.* |
| ***Justificación*** | *Facilita la incorporación de nuevos desarrolladores, soporte técnico e integración con otros sistemas.* |
| ***Implicaciones*** | *Todos los endpoints de API y componentes deben documentarse en paralelo al desarrollo.* |
| ***Información Adicional*** | *Mantener vínculo actualizado con la documentación interna.* |

*1.4 Diagrama de sistema de arquitectura con cada patrón de diseño utilizado*

**

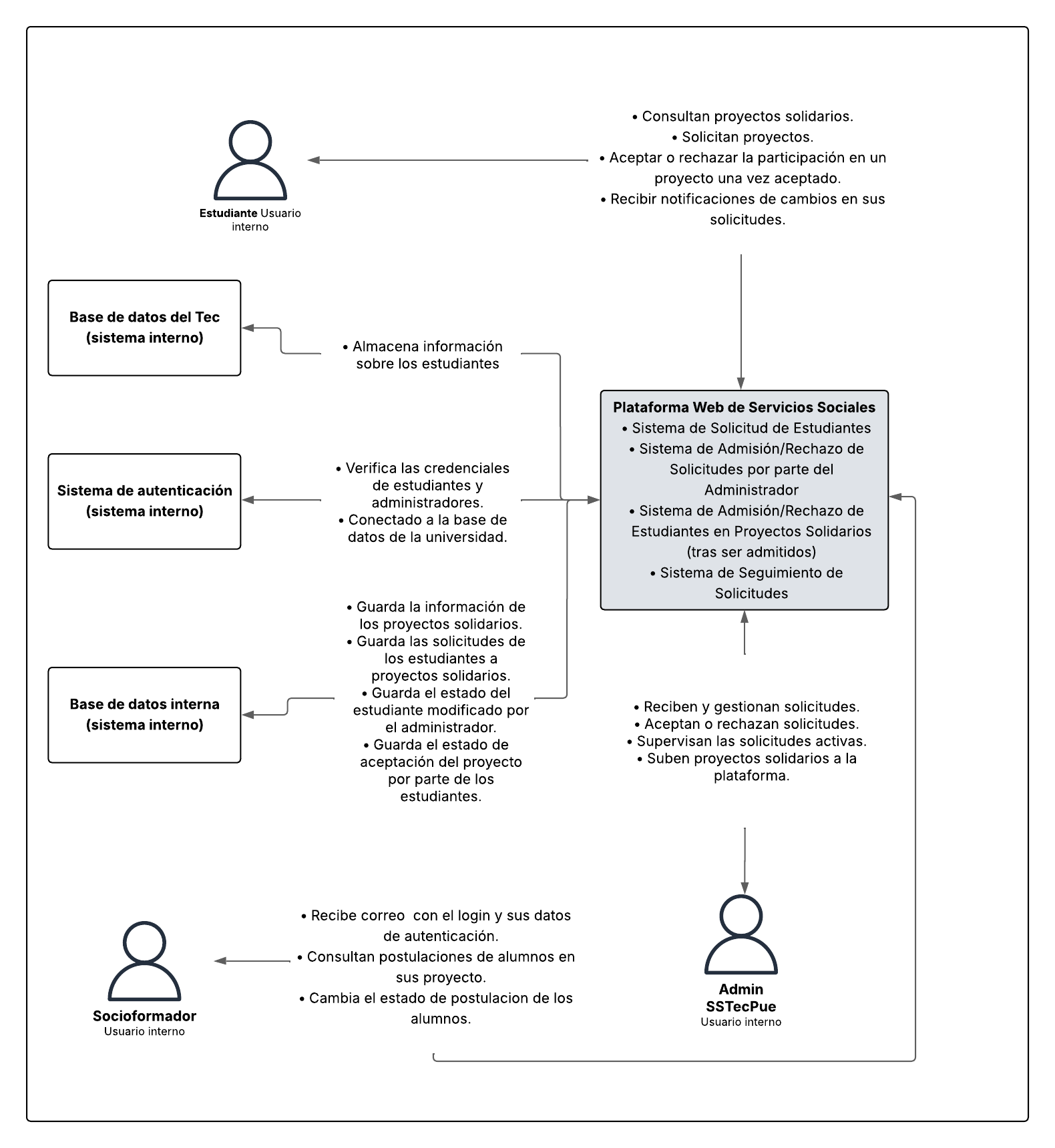
*Figura 1. Sistema de arquitectura con patrones de diseño.*

El sistema se organiza siguiendo patrones de diseño como MVC, Repository, Service y Strategy. El frontend está compuesto por componentes React que presentan la interfaz, controladores Next.js que gestionan la lógica y un modelo de estado que sincroniza los datos. La capa de servicios maneja operaciones clave como envío de correos, gestión de usuarios y procesamiento de archivos. La autenticación se divide entre Google OAuth para estudiantes y credenciales para socios y admins, usando una factoría de roles. La capa de datos abstrae el acceso a Supabase mediante clientes y funciones administrativas. Finalmente, el middleware protege rutas y redirige según roles.

Cuando un usuario accede, el middleware valida su sesión y rol. Luego, la autenticación determina si se trata de un estudiante (OAuth) o un socio/admin (credencial). El usuario interactúa con la interfaz, que consulta o modifica datos a través de servicios, que a su vez se comunican con Supabase. Los servicios también interactúan con APIs externas como Resend o Google.

En esta versión, se agregó una capa clara de servicios para desacoplar la lógica del frontend, se introdujo el patrón de Strategy en la autenticación y se integró el procesamiento de archivos como plantilla reutilizable. Además, se mejoró la organización con middleware en cadena para mayor seguridad.

*1.5 Diagrama de Contexto*

**

*Figura 2. Diagrama de contexto.*

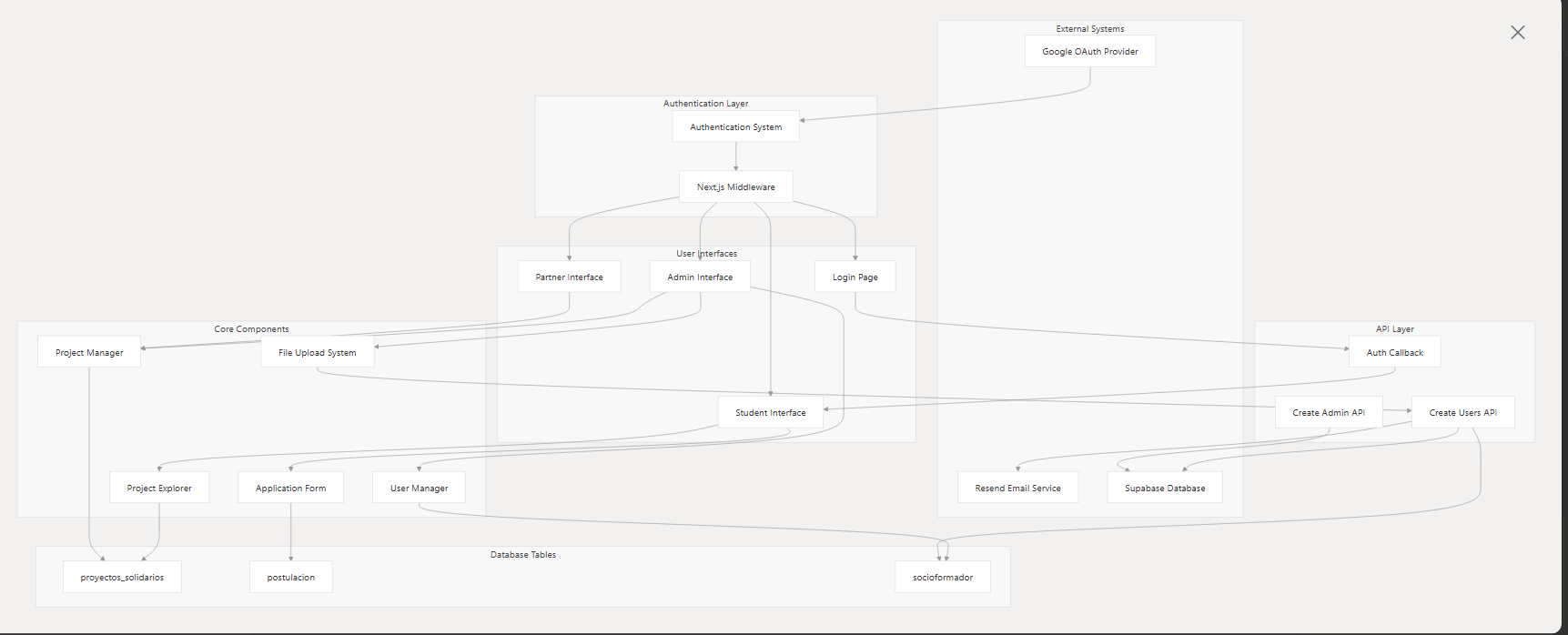
El diagrama de contexto representa la estructura principal y los flujos de interacción de la Plataforma Web de Servicios Sociales del Tecnológico de Monterrey, Campus Puebla. En este sistema intervienen tres actores clave: el estudiante, quien accede al sistema para consultar proyectos solidarios, postularse y dar seguimiento a sus solicitudes; el administrador, encargado de gestionar los proyectos, revisar postulaciones y mantener actualizada la disponibilidad; y el socio formador, actor externo vinculado a las organizaciones responsables de los proyectos, quien accede a la plataforma mediante una clave para revisar y actualizar el estado de los estudiantes postulantes a sus iniciativas.

La plataforma centraliza procesos mediante módulos funcionales: el sistema de solicitud, el sistema de admisión/rechazo por parte del administrador, el sistema de admisión/rechazo por parte del socio formador y el sistema de seguimiento de solicitudes. Estos módulos se apoyan en tres sistemas adicionales: una base de datos del Tec para validar la información institucional del alumno, un sistema de autenticación que permite el acceso con cuenta institucional, y una base de datos interna, que almacena los datos operativos como proyectos, postulaciones e historial.

Entre los principales flujos se encuentran: el acceso del estudiante a la plataforma, la autenticación mediante cuenta del Tec, la consulta del catálogo de proyectos, la postulación y la posterior revisión tanto por el administrador como por el socio formador. La base de datos interna registra toda la interacción y evolución del estado de las solicitudes.

En esta versión, como cambio importante respecto a versiones anteriores, se destaca la incorporación del rol del socio formador, ampliando el modelo de interacción y permitiendo una participación más activa de las organizaciones en el proceso de validación de estudiantes.

*1.6 Vista funcional*

**

*Figura 3. Vista funcional.*

Este diagrama muestra la vista funcional del sistema, donde se organizan las interfaces de usuario (alumno, administrador, socio formador) conectadas a componentes centrales como el explorador de proyectos, formulario de postulación y carga de archivos.

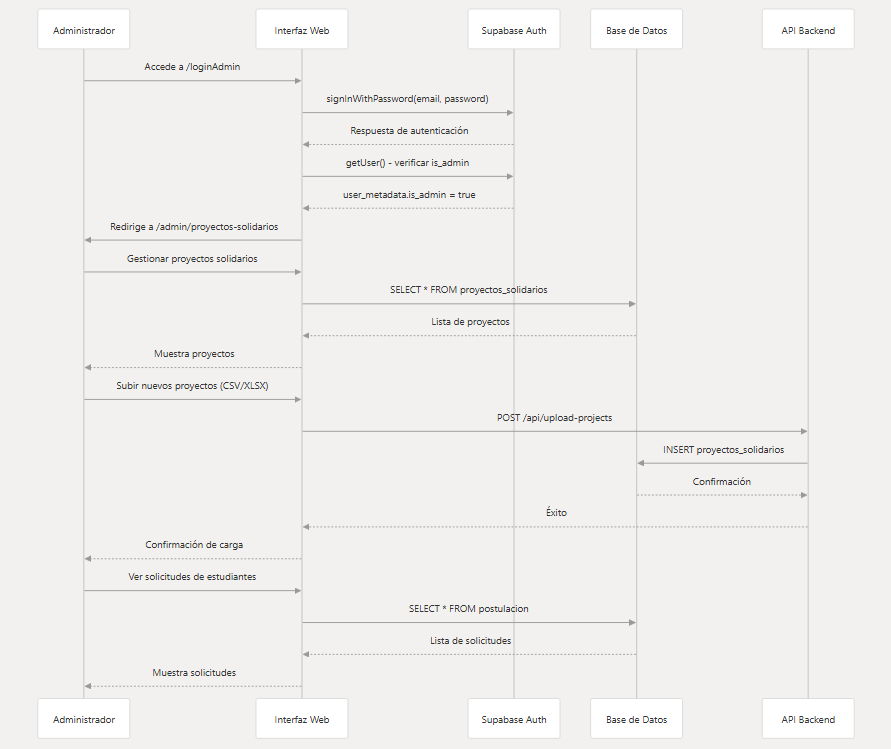
La autenticación se gestiona con Google OAuth y middleware de Next.js, que determina el acceso según el rol. Las interfaces se comunican con la base de datos (proyectos, postulaciones, socio formadores) y con una capa de API para creación de usuarios y callbacks de inicio de sesión.

En esta versión, también se integra un servicio de correos automáticos (Resend) para notificaciones. Esta versión resalta la estructura modular y el control de accesos por rol.

*1.7 Escenarios funcionales*

Esta sección proporciona diagramas de interacción para ilustrar cómo los elementos funcionales del sistema interactúan a través de sus interfaces para lograr escenarios funcionales clave. Al modelar estas interacciones, podemos comprender mejor la secuencia de eventos e intercambios de datos que permiten que el sistema funcione según lo previsto. Los diagramas representan el flujo de información entre los componentes, lo que garantiza la claridad en la colaboración entre las diferentes partes del sistema para satisfacer las solicitudes de los usuarios

1.7.1 Diagrama de Secuencia - Administrador

**

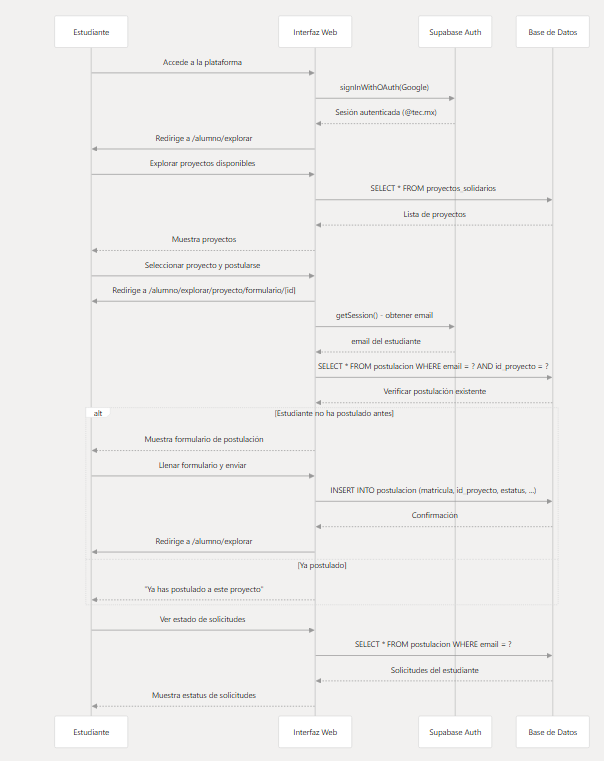
*Figura 4. Diagrama de secuencia de administrador.*

Este diagrama de secuencia representa el flujo de acciones que realiza un administrador en la plataforma de proyectos solidarios. El proceso inicia cuando accede a la ruta /loginAdmin, donde la interfaz web solicita la autenticación a Supabase usando signInWithPassword. Si la autenticación es válida y se verifica que el usuario tiene el rol de administrador (user\_metadata.is\_admin = true), se redirige a la vista de gestión de proyectos.

A continuación, el sistema consulta los proyectos almacenados mediante SELECT \* FROM proyectos\_solidarios y muestra la lista al administrador. Si este decide subir nuevos proyectos mediante archivos CSV o XLSX, la interfaz envía los datos a través de un POST a /api/upload-projects, lo cual desencadena una inserción en la base de datos. Una vez completado, se devuelve una confirmación de carga exitosa.

En esta versión, el administrador puede revisar las solicitudes de estudiantes, para lo cual la interfaz ejecuta una consulta SELECT \* FROM postulacion y muestra los resultados obtenidos de la base de datos. Este flujo garantiza que los administradores puedan autenticar su rol, gestionar proyectos, cargar nuevos registros y monitorear postulaciones de manera integrada.

1.7.2 Diagrama de Secuencia - Alumno



*Figura 5. Diagrama de secuencia de alumno.*

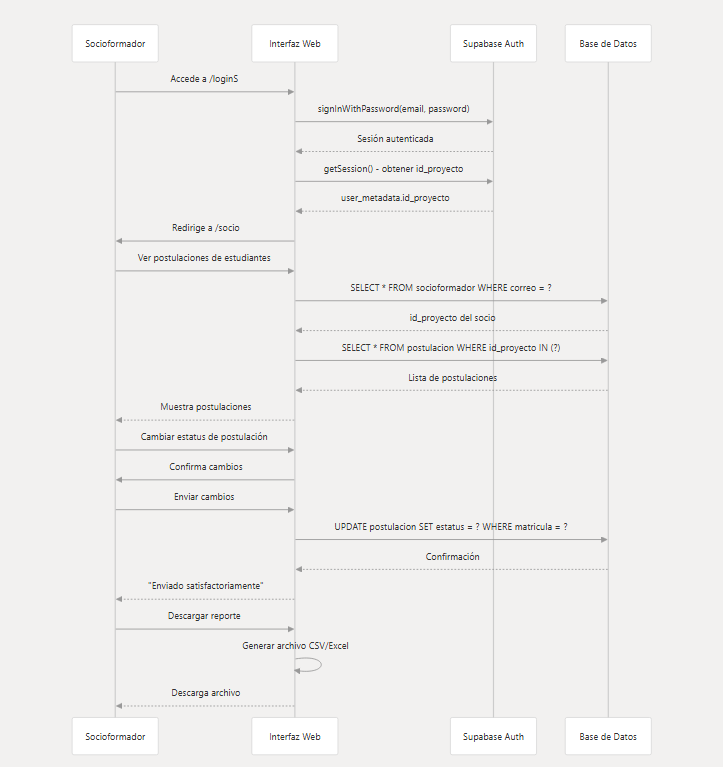
Este diagrama de secuencia muestra el flujo del estudiante dentro de la plataforma de servicio social. Primero, el estudiante accede a la plataforma e inicia sesión con su cuenta institucional mediante Google OAuth. Una vez autenticado, es redirigido a la vista de exploración de proyectos (/alumno/explorar).

En esta vista, el sistema obtiene la lista de proyectos solidarios disponibles desde la base de datos y los presenta en la interfaz. Cuando el estudiante selecciona un proyecto y decide postularse, es redirigido a un formulario específico del proyecto (/alumno/explorar/proyecto/formulario/[id]). Aquí se recupera su sesión y correo, y se consulta si ya ha postulado anteriormente a ese proyecto.

Si no hay una postulación previa, se muestra el formulario. El estudiante lo completa y lo envía, y los datos se insertan en la base de datos. Si ya ha postulado, se le notifica que su solicitud ya existe.

En esta versión, el estudiante puede consultar el estado de sus postulaciones. La plataforma recupera esa información desde la base de datos y la despliega en la interfaz. Este flujo garantiza que el proceso de postulación sea seguro, personalizado y controlado.

1.7.3 Diagrama de Secuencia - Socio Formador



*Figura 6. Diagrama de secuencia de socio formador.*

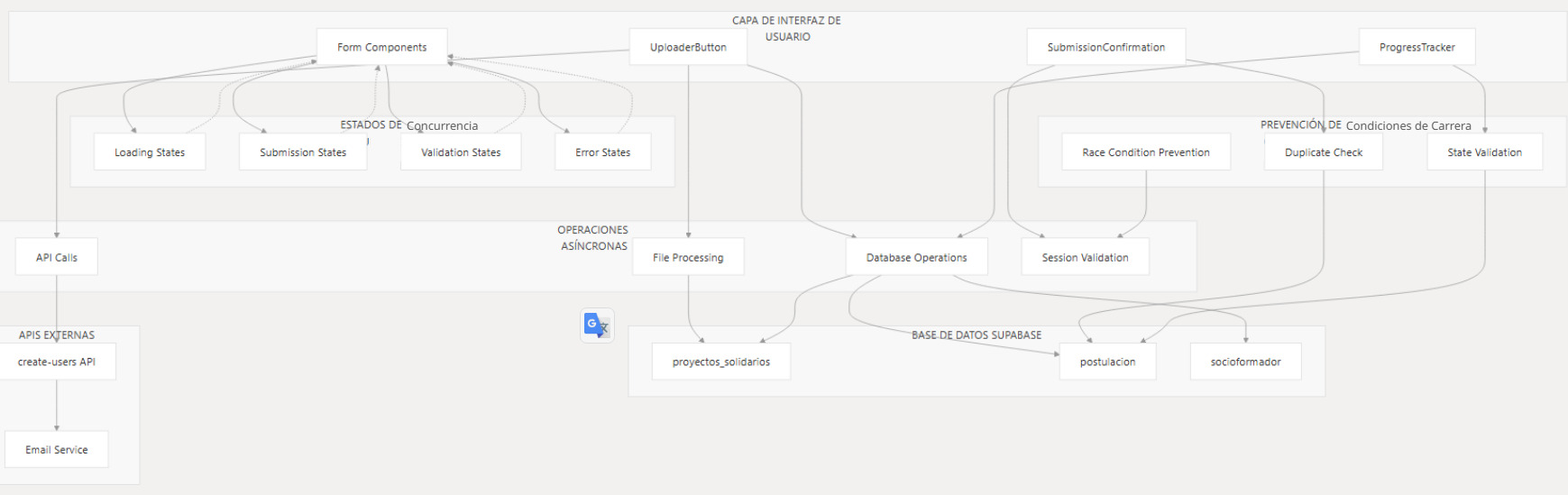
Este diagrama de secuencia muestra el flujo de interacción del socio formador dentro de la plataforma. El proceso inicia cuando accede a la ruta /loginS e ingresa sus credenciales. La autenticación se realiza mediante signInWithPassword de Supabase, y se obtiene la sesión, incluyendo el id\_proyecto asociado al socio.

Una vez autenticado, el socio formador es redirigido a la vista /socio, donde puede ver las postulaciones de estudiantes a su proyecto. La plataforma consulta primero el id\_proyecto del socio formador en la base de datos y luego obtiene las postulaciones asociadas a ese proyecto. Esta información se muestra en la interfaz.

El socio formador puede modificar el estatus de las postulaciones (por ejemplo, aceptar o rechazar), confirmar los cambios y enviarlos. La plataforma actualiza los registros mediante una sentencia UPDATE en la base de datos, y al recibir la confirmación, muestra un mensaje de éxito.

En esta versión, el socio formador tiene la opción de descargar un reporte con la información de las postulaciones. La interfaz genera un archivo CSV o Excel que puede ser descargado directamente desde el navegador.

*1.8 Diagrama de Concurrencia*

**

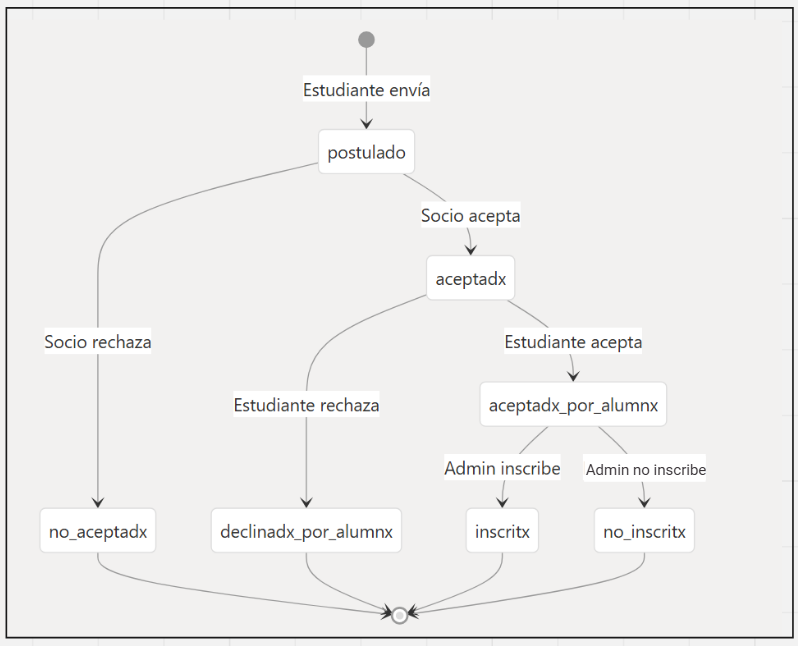
*Figura 7. Diagrama de concurrencia.*

Este diagrama muestra la interacción entre los componentes de la interfaz de usuario (como Form Components, UploaderButton, ProgressTracker) y las operaciones asincrónicas (File Processing, Database Operations). También se incluye validación de sesión, prevención de condiciones de carrera y estados de concurrencia (loading, error, etc.).

El usuario envía datos mediante formularios que desencadenan llamadas asincrónicas a la base de datos y servicios externos como create-users API y Email Service. Los datos pasan por validaciones y se procesan en Supabase, actualizando entidades como postulacion, proyectos\_solidarios y socioformador.

En esta versión, se integró control de estados de concurrencia, prevención de duplicados y validación de carrera, lo que mejora la consistencia y la experiencia del usuario durante los envíos.

*1.9 Modelo de estados*

**

*Figura 8. Diagrama de estados de postulación de alumno.*

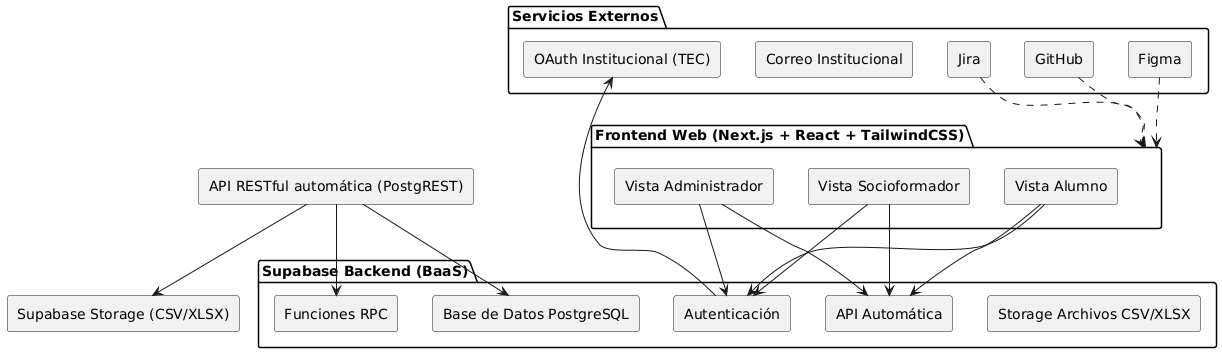
Este diagrama muestra los posibles estados de una postulación en la plataforma. El proceso comienza cuando el estudiante envía su solicitud y pasa a estado postulado. Luego, el socio formador puede aceptarla (aceptadx) o rechazarla (no\_aceptadx). Si acepta, el estudiante debe confirmar su interés (aceptadx\_por\_alumnx), tras lo cual el administrador decide si inscribirlo (inscritx) o no (no\_inscritx).

**Flujos principales de interacción:**

1. Estudiante envía la solicitud → postulado.
2. El socio acepta o rechaza → si rechaza, termina en no\_aceptadx.
3. Si acepta, el estudiante puede aceptar o rechazar → si rechaza, termina en declinadx\_por\_alumnx.
4. Si acepta, el administrador inscribe o no → finaliza en inscritx o no\_inscritx.

En esta versión, se incorporó un estado adicional para distinguir si la no inscripción fue decisión del administrador (no\_inscritx), y se representa explícitamente la posibilidad de rechazo tanto por parte del socio como del alumno. Esto mejora la trazabilidad del flujo de decisiones.

*1.10 Diagrama de componentes*



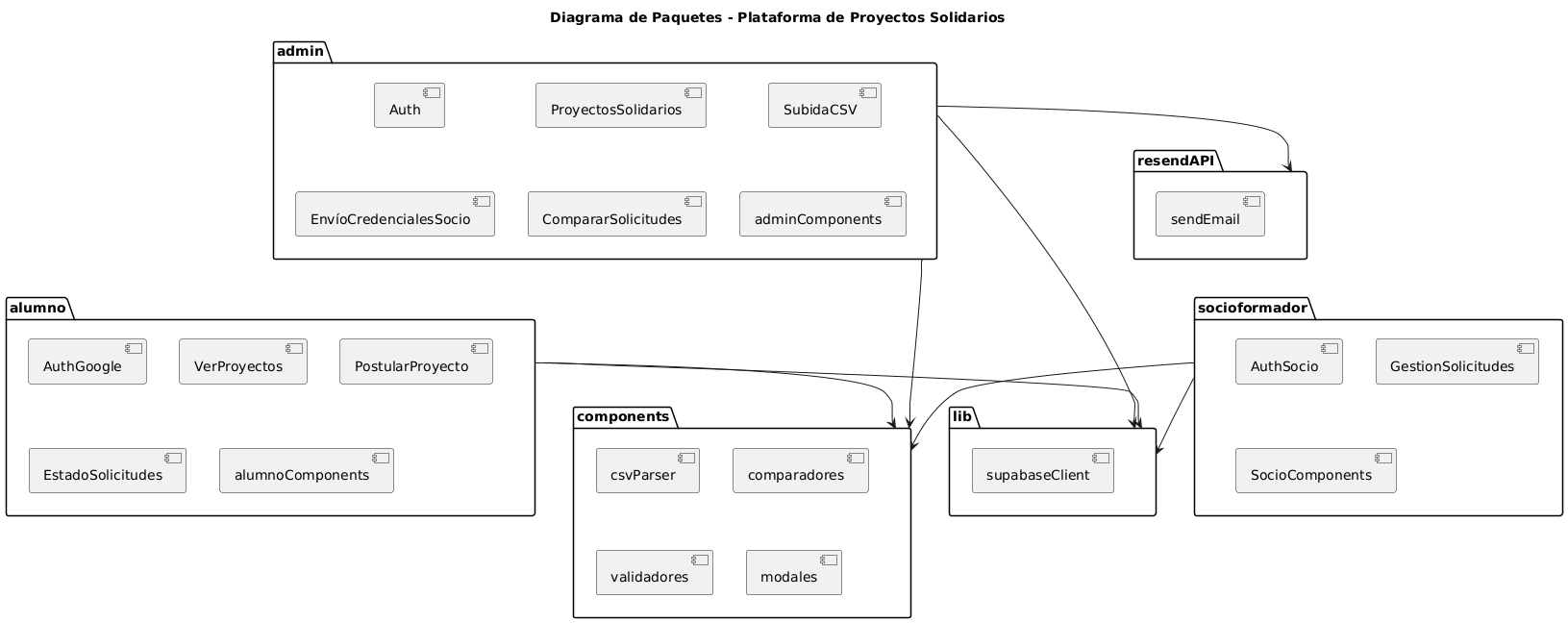
*Figura 9. Diagrama de componentes.*

Este diagrama muestra la arquitectura general del sistema. El Frontend Web (Next.js + React + TailwindCSS) tiene vistas para administradores, alumnos y socioformadores. Cada rol accede a funciones específicas mediante una interfaz personalizada.

El Backend usa Supabase, que proporciona autenticación, base de datos PostgreSQL, funciones RPC y una API automática (PostgREST). Además, permite cargar y descargar archivos CSV/XLSX. Se integran servicios externos como OAuth institucional, correo del Tec, Figma, GitHub y Jira. Los usuarios acceden con su cuenta institucional, interactúan con su vista y las acciones se gestionan por la API.

En esta versión, los cambios clave incluyen la nueva vista para socioformadores y la gestión de archivos para postulaciones.

*1.11 Diagrama de paquetes*



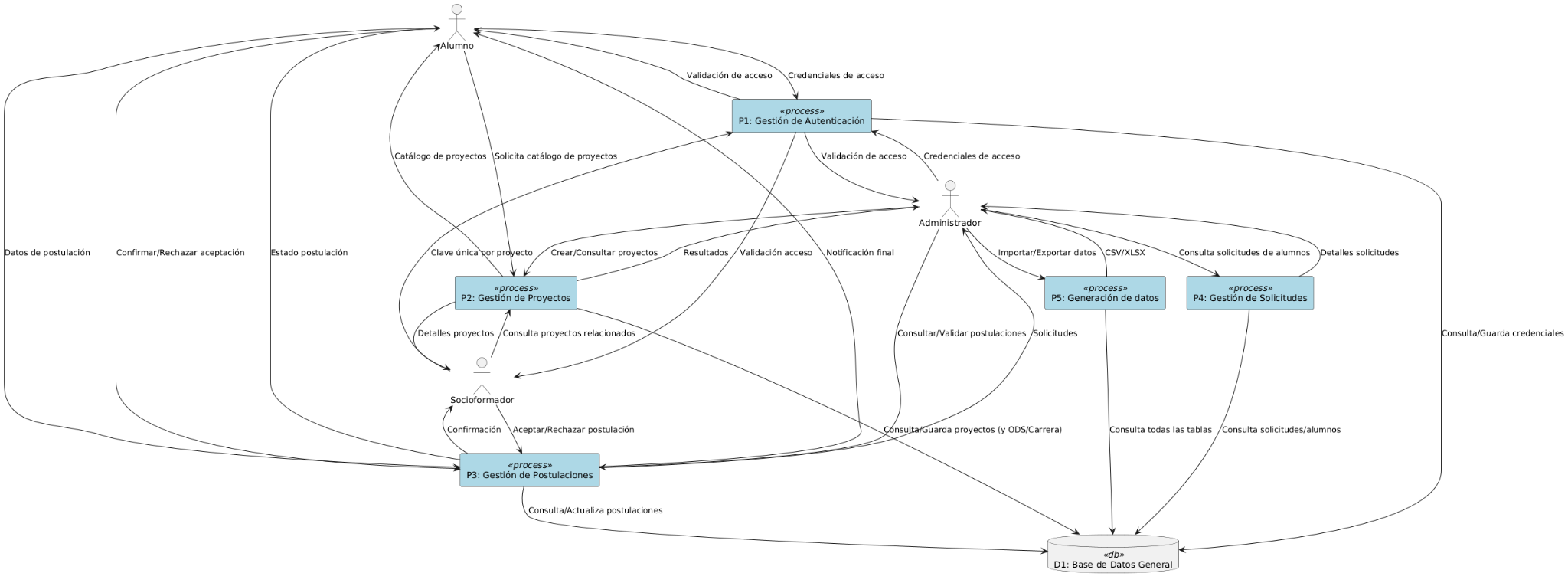
*Figura 10. Diagrama de paquetes.*

El diagrama muestra la organización modular del sistema. Los paquetes admin, alumno y socioformador contienen las vistas y lógicas específicas de cada rol. El paquete lib centraliza el cliente de Supabase, y components incluye utilidades compartidas como validadores, comparadores y modales. resendAPI maneja el envío de correos.

Cada vista accede a componentes comunes y al cliente de Supabase. Admin gestiona proyectos y postulaciones, alumno visualiza y postula, y el socio revisa solicitudes. Los datos viajan entre vistas, lib, y utilidades en components.

En esta versión, se consolidaron funciones comunes en components y lib para reducir redundancia, y se agregó resendAPI para mejorar la notificación por correo. También se separó claramente la lógica por tipo de usuario.

*1.12 Diagrama de flujo de datos*

**

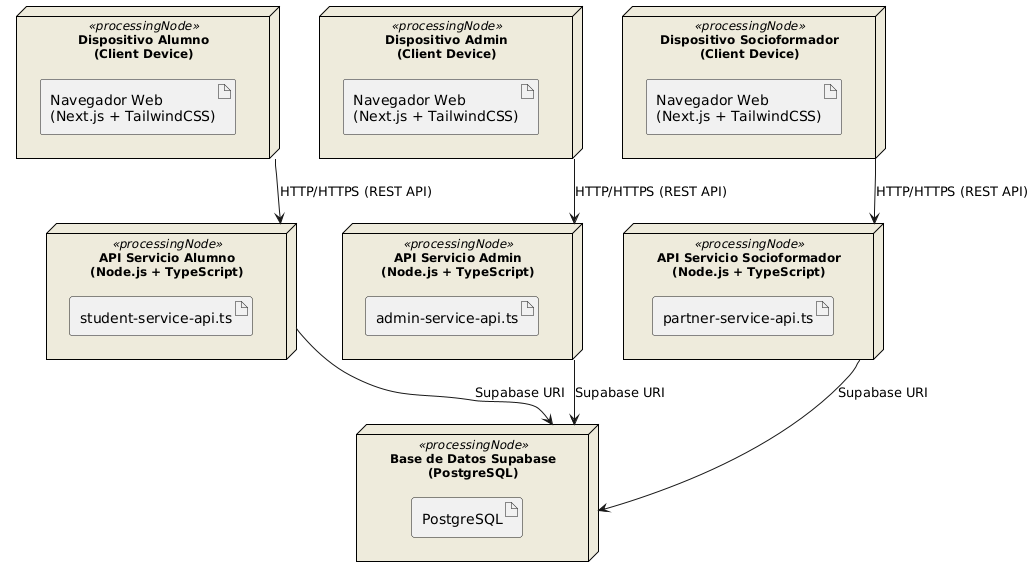
*Figura 11. Diagrama de flujo de datos.*

El diagrama representa a los tres usuarios principales: Alumno, Socioformador y Administrador. Cada uno interactúa con procesos clave: autenticación (P1), gestión de proyectos (P2), gestión de postulaciones (P3), gestión de solicitudes (P4) y generación de datos (P5). Todos los procesos están conectados a una base de datos central (D1).

El alumno se autentica, consulta proyectos, postula y revisa el estado. El socio formador consulta y acepta/rechaza postulaciones. El administrador valida credenciales, gestiona proyectos, postulaciones, exporta datos y controla toda la plataforma. Todos los procesos consultan o actualizan la base de datos.

En esta versión, se integró el proceso P5 para manejar la exportación/importación de datos. Además, ahora el socio formador tiene un rol más activo en la aceptación de postulaciones, y la validación de acceso se aplica a los tres tipos de usuarios.

*1.13 Diagrama de despliegue*

**

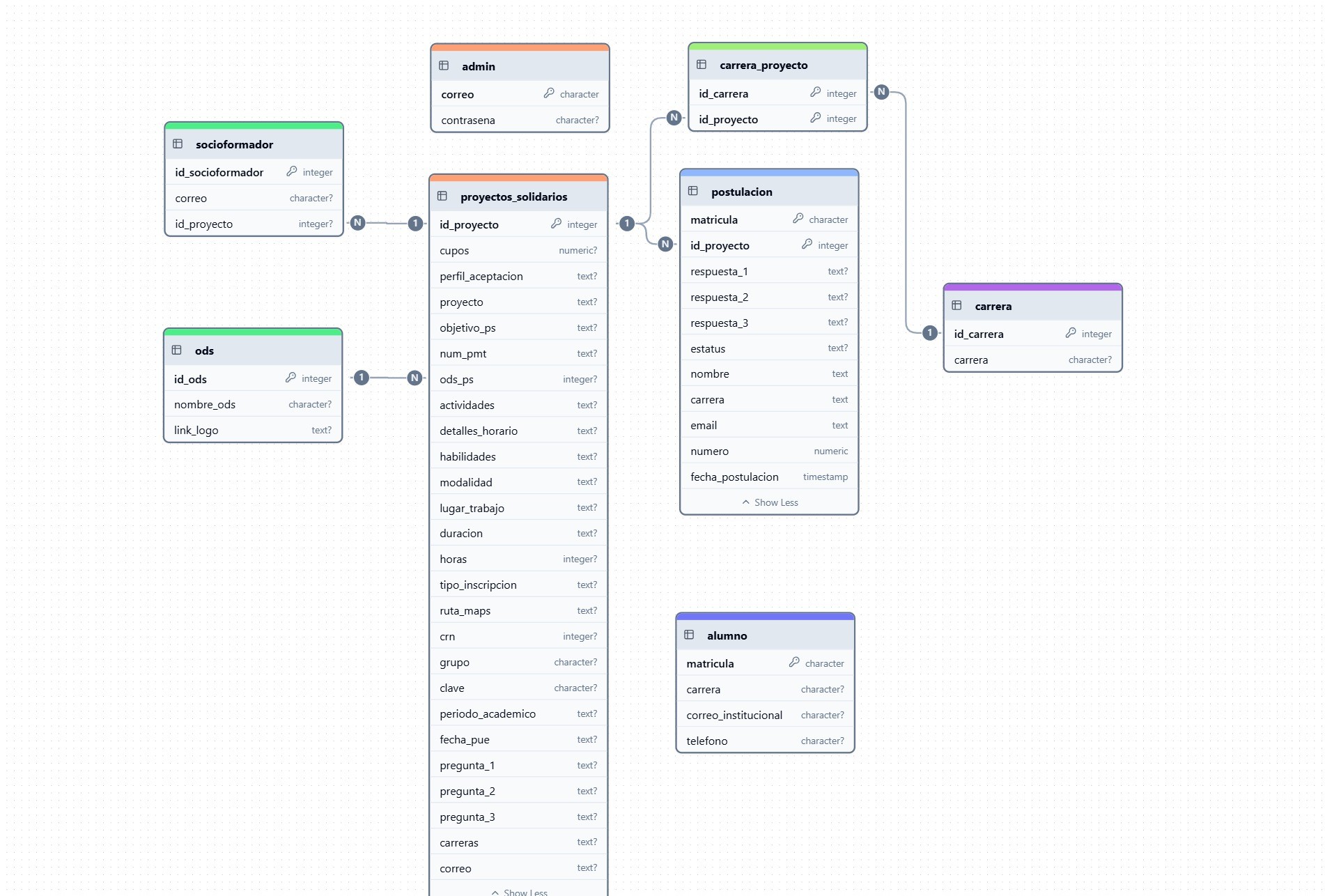
*Figura 12. Diagrama de despliegue.*

Cada tipo de usuario (Alumno, Admin y Socio Formador) accede a la plataforma desde su dispositivo mediante un navegador web desarrollado con Next.js y TailwindCSS. Estos clientes interactúan con APIs específicas (student-service-api.ts, admin-service-api.ts y partner-service-api.ts) construidas en Node.js + TypeScript. Todas las APIs se conectan a una base de datos central gestionada en Supabase (PostgreSQL).

Los dispositivos cliente envían solicitudes vía HTTP/HTTPS a sus respectivas APIs. Estas, a su vez, realizan operaciones sobre la base de datos a través de URIs de Supabase para leer o modificar datos relevantes (postulaciones, proyectos, usuarios, etc.).

En esta versión, se modularizó la arquitectura dividiendo las responsabilidades por tipo de usuario en servicios independientes, lo que facilita el mantenimiento, escalabilidad y pruebas específicas por rol. Además, se unificó el acceso a la base de datos mediante Supabase para todas las APIs.

*1.14 Vista de información*

**

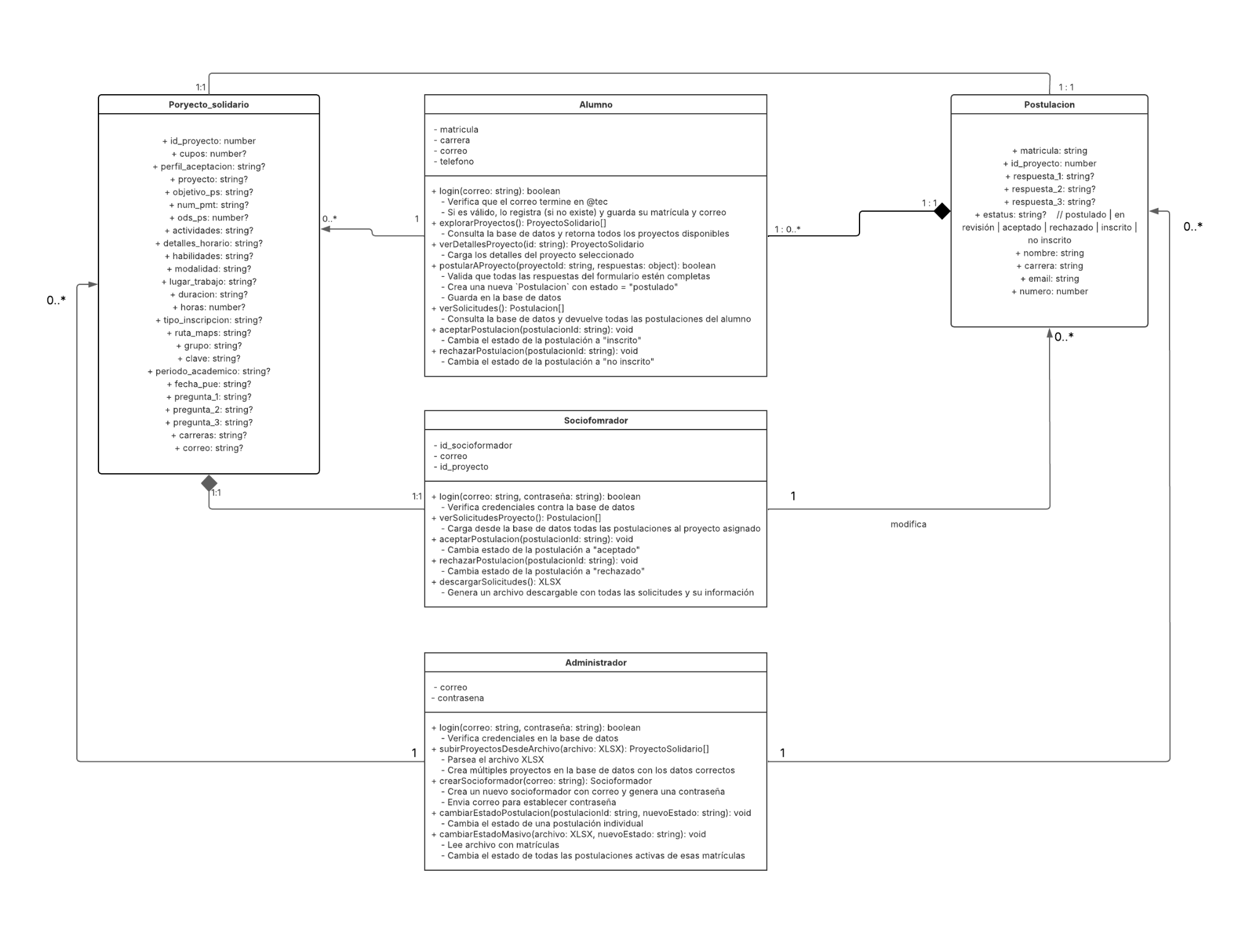
*Figura 13. Diagrama de base de datos.*

El diagrama representa la estructura de datos de la plataforma. La tabla alumno almacena la información básica de los estudiantes, mientras que admin y socioformador gestionan el acceso de administradores y socios respectivamente. Los socios están vinculados a un proyecto específico.

Los proyectos están definidos en proyectos\_solidarios y se relacionan con postulaciones de alumnos a través de la tabla postulacion. Cada postulación guarda respuestas, estado y datos del alumno. Además, los proyectos pueden asociarse a una carrera (carrera\_proyecto) y a un objetivo ODS (ods).

En esta versión, entre los cambios recientes está la incorporación de relaciones más claras entre proyectos, carreras y socios, lo que permite un mejor control del acceso y la trazabilidad de postulaciones.

*1.15 Diagrama de clases*

**

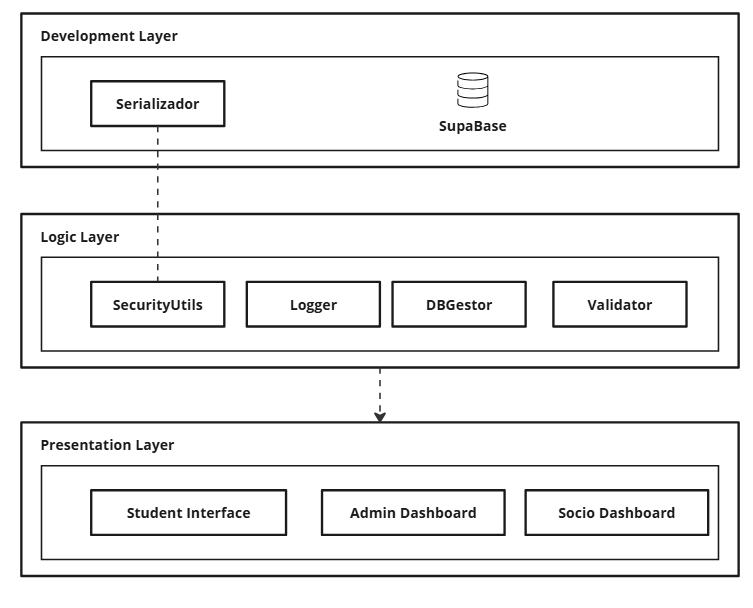
*Figura 14. Diagrama de clases.*

Este diagrama muestra las clases principales del sistema: Alumno, Administrador, Socioformador, Postulacion y Proyecto\_solidario. Cada clase representa una entidad clave con sus atributos y métodos. Por ejemplo, el Alumno puede explorar y postularse a proyectos, mientras que el Administrador gestiona proyectos, socios y postulaciones. El Socioformador valida las solicitudes asignadas a su proyecto.

El Alumno inicia sesión, consulta proyectos y realiza postulaciones. Estas son almacenadas y gestionadas por el Administrador, quien puede también subir proyectos vía archivo. El Socioformador revisa y aprueba o rechaza postulaciones. La clase Postulacion actúa como vínculo entre Alumno y Proyecto\_solidario.

En esta versión, se agregaron métodos detallados en cada clase para reflejar acciones específicas (e.g., aceptarPostulacion, descargarSolicitudes, subirProyectosDesdeArchivo) y se fortaleció el control sobre el estado de cada postulación. También se definieron mejor las relaciones entre entidades.

1.16 Diagrama de estructura de módulos



*Figura 15. Diagrama de estructura de módulos.*

La capa de presentación incluye las vistas específicas para estudiantes, administradores y socios formadores. La capa lógica gestiona validaciones, seguridad, logs y operaciones de base de datos. La capa de desarrollo se encarga de la serialización y conexión con Supabase.

Las interfaces de usuario interactúan con la capa lógica para validar y procesar datos. Esta capa, a su vez, utiliza serializadores y gestiona la base de datos mediante Supabase.

En esta versión, se realizó un cambio en donde se organizó el sistema en tres capas claras (presentación, lógica y desarrollo), facilitando el mantenimiento, la escalabilidad y la separación de responsabilidades, además de que se agregó la etiqueta de Socio.