

**Instituto Profesional Duoc UC**

**Escuela de Informática y Telecomunicaciones**

**Carrera de Ingeniería en Informática**

**Diseño e Implementación de Vistas 4+1**

**Nombre del profesor:** Fernando Herrera

**Nombres de los estudiantes:** Eduardo Álvarez

Felipe Silva

**Sección:** 707V - 706V

OCTUBRE – 2025

SANTIAGO DE CHILE

**ÍNDICE**

[**1) Introducción 3**](#_heading=h.ak7b8ber5dyf)

[1.1. Propósito del Documento 3](#_heading=h.n8h3gsmz7mcz)

[1.2. Alcance del Proyecto (StudIA) 3](#_heading=h.wpnh2balpyz)

[1.3. Actores Clave 4](#_heading=h.bry4dslhv04d)

[1.4. Atributos de Calidad (Requisitos No Funcionales) 4](#_heading=h.e8k5s9x9o3r0)

[**2) Vista Lógica (El "Qué") 5**](#_heading=h.i7y77y50acim)

[2.1. Descripción General 5](#_heading=h.ifx7kxlqtlf7)

[2.2. Modelo de Dominio (Diagrama de Clases) 5](#_heading=h.d1l2z8jrsiu8)

[2.3. Modelo de Datos (Diagrama Entidad-Relación - E-R) 7](#_heading=h.bzchx2gjzown)

[2.4. Diagramas de Secuencia (Interacción) 8](#_heading=h.5oyv4i2tmhb2)

[2.4.1. Secuencia: Inscripción a una Ayudantía (Flujo del Estudiante) 8](#_heading=h.6r41lrblnptj)

[**3) Vista de Implementación (Desarrollo) 9**](#_heading=h.fdmtayulm8z)

[3.1. Descripción General 9](#_heading=h.3wzw4rmlaici)

[3.2. Diagrama de Componentes 9](#_heading=h.ds1ubzw7zq9b)

[3.3. Diagrama de Paquetes (Estructura de Código) 11](#_heading=h.iexxifbwhv2q)

[**4) Vista de Procesos (El "Cuándo") 12**](#_heading=h.kr0m85prqkdk)

[4.1. Descripción General 12](#_heading=h.ptc54larw0t7)

[4.2. Diagrama de Actividad: "Asignar Ayudantía" 12](#_heading=h.kun983oyj8pl)

[**5) Vista Física (de Despliegue) 14**](#_heading=h.n5ae3rjtcjrb)

[5.1. Descripción General 14](#_heading=h.8gkwnzx5lh4l)

[5.2. Topología de Despliegue 15](#_heading=h.7tci60pkxibt)

[**6) Vista de Escenarios (Casos de Uso) 16**](#_heading=h.na7tfxw1gokt)

[6.1. Descripción General 16](#_heading=h.vvuskavpnww5)

[6.2. Actores Identificados 16](#_heading=h.wm6rqwukcy8r)

[6.3. Diagrama: "Caso de uso General" 17](#_heading=h.rhupc1sj3s0a)

[6.4. Análisis del Diagrama: "Administrador de Usuarios" 18](#_heading=h.d4if65k7rr7d)

[6.5. Análisis del Diagrama: "Agendamiento" 20](#_heading=h.1km04o8e7v6z)

[6.6. Análisis del Diagrama: "Ver Asignaciones" 21](#_heading=h.7yhlt5d6otjx)

[6.7. Análisis del Diagrama: "Mis Ayudantías" 22](#_heading=h.z9vd5kuihx9r)

[6.8. Análisis del Diagrama: "Reportería" 22](#_heading=h.c6q5t6xx15q3)

[**7) Decisiones de Arquitectura y Tecnologías 23**](#_heading=h.wysbgtxj34rn)

[7.1. Patrón de Arquitectura 23](#_heading=h.ll38q4hox0e8)

[7.2. Stack Tecnológico (Confirmado por Diagramas) 24](#_heading=h.or03m49e42pc)

[7.3. Justificación de Decisiones 24](#_heading=h.v159oalpahmq)

[**8) Conclusiones 25**](#_heading=h.oku7wie3jy4d)

[8.1. Resumen de la Arquitectura 25](#_heading=h.ivugi4f7xht3)

[8.2. Futuras Mejoras o Extensiones Posibles 25](#_heading=h.jh254rgun3e5)

# **1) Introducción**

### **1.1. Propósito del Documento**

El presente documento define y describe la arquitectura de software para el sistema "Portal de Gestión de Ayudantías – StudIA". Su propósito principal es servir como guía técnica fundamental para el equipo de desarrollo, alinear la visión del producto con la implementación técnica y comunicar las decisiones de diseño estructural a todos los *stakeholders* involucrados.

Este informe detalla los componentes del sistema, sus relaciones, sus interacciones y las vistas arquitectónicas clave (Lógica, de Implementación, de Procesos, Física y de Escenarios). El enfoque principal es establecer una base sólida para el **Producto Mínimo Viable (MVP)**, asegurando que la solución sea técnica y funcionalmente viable, al mismo tiempo que se establecen los cimientos para una futura escalabilidad.

### **1.2. Alcance del Proyecto (StudIA)**

**StudIA** se concibe como una solución integral (web y móvil) para la gestión y coordinación de ayudantías académicas en instituciones de educación superior.

**El Problema:** El proyecto aborda una problemática común en el entorno educativo:

* La falta de una herramienta institucional centralizada para coordinar tutorías.
* La dificultad en la comunicación efectiva entre estudiantes que necesitan apoyo y tutores que pueden ofrecerlo.
* La deserción académica frecuentemente ligada al bajo rendimiento en asignaturas clave.
* La falta de oportunidades económicas estructuradas para estudiantes con buen rendimiento que desean enseñar.

**La Solución (MVP):** Se desarrollará un Producto Mínimo Viable (MVP) que permita optimizar la forma en que los estudiantes gestionan, ofrecen y acceden a ayudantías. Este MVP se centrará en validar la viabilidad técnica y funcional del flujo completo del servicio.

Las funcionalidades centrales del MVP incluyen:

* Registro, autenticación y gestión de perfiles de usuario.
* Publicación y gestión de ayudantías (por parte de tutores).
* Búsqueda e inscripción a ayudantías (por parte de estudiantes).
* Un panel administrativo básico para la supervisión y gestión de usuarios.

**Objetivo Estratégico:** El objetivo final de StudIA es conectar conocimiento, facilitar la gestión académica y fortalecer el aprendizaje colaborativo. El MVP busca probar esta hipótesis de valor en un entorno real, ofreciendo una solución tecnológica ágil y accesible.

### **1.3. Actores Clave**

El sistema define tres roles de usuario principales (actores) que interactúan con la plataforma:

* **Estudiante (Solicitante):** Es el consumidor principal del servicio. Este actor puede buscar ayudantías filtrando por asignatura, horario o tutor; inscribirse en sesiones disponibles; y gestionar su perfil y sus inscripciones.
* **Tutor (Ayudante):** Es el proveedor del servicio. Este actor (previamente validado o autorizado) puede crear y publicar sus propias ayudantías, establecer horarios, definir cupos máximos, gestionar la lista de inscritos y administrar su perfil público.
* **Administrador (Supervisor):** Es el rol de gestión y control. Este actor tiene permisos para validar y autorizar nuevas solicitudes de tutores, gestionar la base de datos de usuarios (ej. bloquear o eliminar), y revisar indicadores básicos de uso y participación (ej. número de ayudantías creadas, tasa de inscripción).

### **1.4. Atributos de Calidad (Requisitos No Funcionales)**

Para que el MVP de StudIA sea exitoso y siente las bases para el futuro, la arquitectura debe priorizar los siguientes atributos de calidad:

* **Accesibilidad:** Dado que la solución es web y móvil, la arquitectura debe soportar el acceso desde múltiples dispositivos (navegadores de escritorio y *smartphones*), garantizando una experiencia de usuario coherente.
* **Mantenibilidad:** Al ser un MVP, se espera que el sistema evolucione rápidamente. La arquitectura debe ser modular y limpia, permitiendo al equipo de desarrollo agregar nuevas funcionalidades (post-MVP) con el mínimo esfuerzo y riesgo.
* **Escalabilidad:** Aunque el MVP se probará en un entorno controlado, la arquitectura debe diseñarse con la capacidad de crecer. Debe poder manejar un aumento futuro en el número de usuarios, sesiones y datos sin degradar el rendimiento.
* **Seguridad:** El sistema manejará datos personales y académicos de los estudiantes. Es fundamental garantizar la correcta autenticación (quién eres) y autorización (qué puedes hacer) para cada actor, así como la protección de los datos almacenados.
* **Rendimiento:** Las funciones clave del MVP (búsqueda de ayudantías, proceso de inscripción) deben ser rápidas y eficientes para no frustrar al usuario y validar correctamente el flujo del servicio.

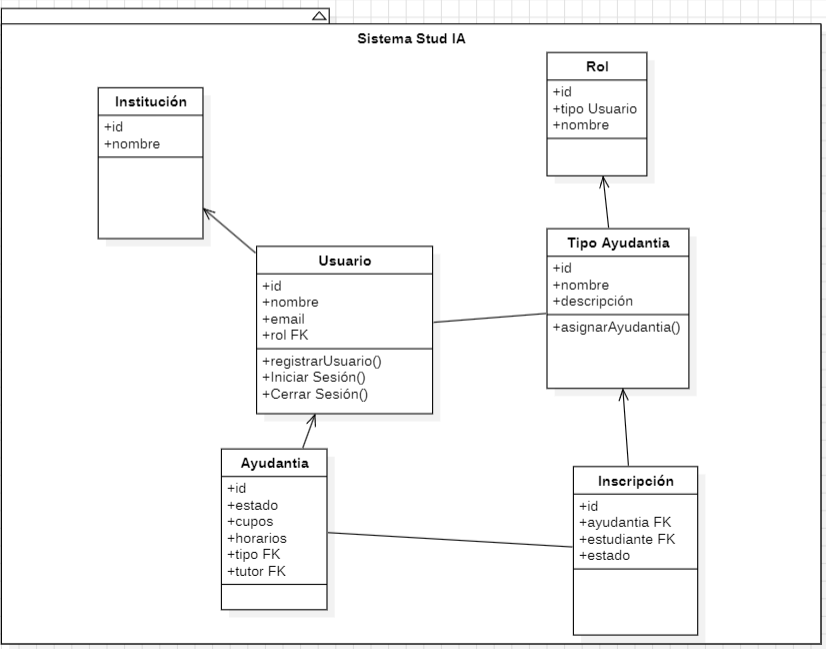
# **2) Vista Lógica (El "Qué")**

### **2.1. Descripción General**

La Vista Lógica describe la estructura estática del sistema en términos de sus clases, objetos y las relaciones entre ellos. Responde a la pregunta: **¿De qué está hecho el sistema?**

Esta vista es fundamental para el MVP de StudIA, ya que define el vocabulario del dominio (las "entidades" del negocio) y cómo se relacionan. Utilizamos el Diagrama de Clases para mostrar los conceptos y responsabilidades, y el Diagrama Entidad-Relación (E-R) para detallar la implementación específica en la base de datos.

### **2.2. Modelo de Dominio (Diagrama de Clases)**

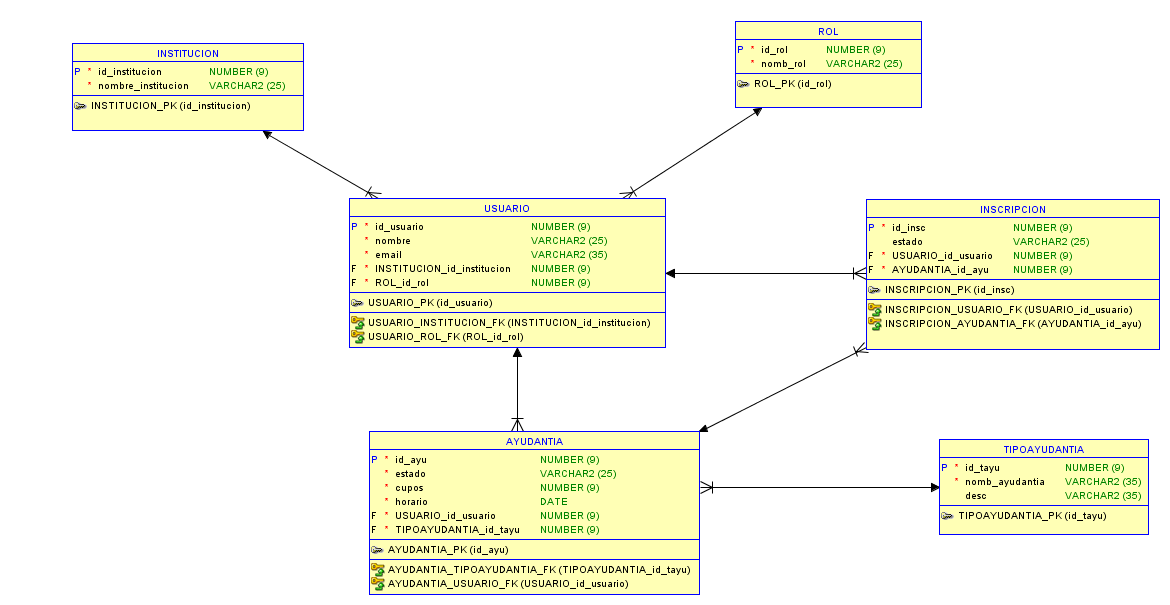


El diagrama de clases modela las principales abstracciones del sistema. Define las responsabilidades y atributos de cada concepto clave:

* **Usuario:** Es la entidad central que representa a cualquier actor que interactúa con el sistema. Concentra las operaciones de gestión de identidad (registrarUsuario(), IniciarSesion(), CerrarSesion()). Está correctamente asociado a una Institucion y a un Rol.
* **Rol:** Entidad que define los permisos y el tipo de usuario (Estudiante, Tutor, Administrador).
* **Institucion:** Permite agrupar o segmentar a los usuarios por su afiliación (ej. universidad, facultad).
* **Ayudantia:** Representa la "oferta" de tutoría. Define los atributos clave del servicio, como estado (ej. Abierta, Cerrada, Llena), cupos y horarios.
* **Tipo Ayudantia:** Funciona como la categoría de la ayudantía (ej. "Cálculo I", "Programación Orientada a Objetos", "Ensayo SIMCE").
* **Inscripcion:** Es la clase asociativa que conecta a un Usuario (en su rol de Estudiante) con una Ayudantia. Registra el estado de dicha inscripción (ej. Confirmada, Pendiente, Cancelada).

### 

### **2.3. Modelo de Datos (Diagrama Entidad-Relación - E-R)**



El diagrama E-R detalla la implementación física de la base de datos relacional y clarifica las relaciones del modelo de dominio. El diseño es robusto y refleja fielmente la lógica del negocio:

* **Tablas Centrales:** USUARIO, AYUDANTIA, INSCRIPCION.
* **Tablas de Catálogo (Lookup):** ROL, INSTITUCION, TIPOAYUDANTIA.

El análisis de las claves foráneas (FK):

1. **Relación Tutor-Ayudantía:** La tabla AYUDANTIA tiene una FK USUARIO\_id\_usuario. Esta clave identifica al **Tutor** que crea y dicta la ayudantía.
2. **Relación Estudiante-Inscripción:** La tabla INSCRIPCION tiene una FK USUARIO\_id\_usuario. Esta clave identifica al **Estudiante** que se inscribe en la ayudantía.
3. **Relación Ayudantía-Inscripción:** La tabla INSCRIPCION tiene una FK AYUDANTIA\_id\_ayu, vinculando la inscripción con la ayudantía específica.

Este diseño separa correctamente los roles: un USUARIO actúa como "Tutor" al *crear* una AYUDANTIA y actúa como "Estudiante" al *crear* una INSCRIPCION.

### 

### 

### 

### 

### **2.4. Diagramas de Secuencia (Interacción)**

#### **2.4.1. Secuencia: Inscripción a una Ayudantía (Flujo del Estudiante)**

Este diagrama ilustra cómo colaboran los objetos del sistema cuando un estudiante (actor) decide inscribirse en una sesión.

1. **Actor (Estudiante)** interactúa con la VistaInscripcion (Frontend).
2. La Vista invoca al ControladorInscripcion.inscribir(id\_usuario, id\_ayudantia).
3. El ControladorInscripcion llama al ServicioInscripcion.crearInscripcion(...).
4. El ServicioInscripcion *primero* debe validar la lógica de negocio. Para ello, llama al RepositorioAyudantia.obtenerPorId(id\_ayudantia) para obtener los cupos y el estado de esa ayudantía.
5. El ServicioInscripcion *también* llama al RepositorioInscripcion.contarPorAyudantia(id\_ayudantia) para saber cuántos inscritos hay.
6. **Lógica de Negocio:** El Servicio compara inscritos\_actuales vs cupos.
7. Si hay cupos disponibles, el ServicioInscripcion crea un nuevo objeto Inscripcion (con estado = 'Confirmado') y llama al RepositorioInscripcion.guardar(nuevaInscripcion).
8. La BaseDatos ejecuta el INSERT.
9. Se retorna un mensaje de "Éxito" al Estudiante.

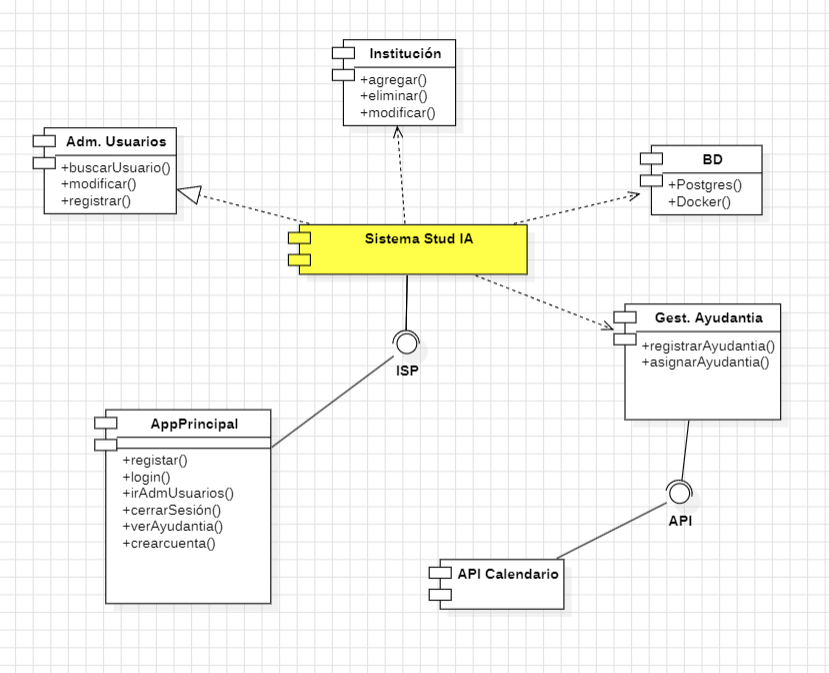
# **3) Vista de Implementación (Desarrollo)**

### **3.1. Descripción General**

La Vista de Implementación (o de Desarrollo) describe cómo se organiza el sistema desde la perspectiva del desarrollador. Responde a la pregunta: **¿Cómo se organiza y ensambla el código fuente?**

Esta vista se centra en los componentes de software (partes compilables y desplegables) y los paquetes (agrupaciones lógicas de código) que conforman el sistema StudIA.

### **3.2. Diagrama de Componentes**



El diagrama de componentes ilustra una arquitectura orientada a servicios o microservicios, donde un sistema central se apoya en componentes especializados.

**Análisis de Componentes:**

* **Sistema Stud IA:** Es el componente central o *facade* del backend. Actúa como el orquestador principal que consume las interfaces de los demás módulos de negocio. Expone la interfaz principal (ISP) que será consumida por los clientes.
* **AppPrincipal:** Representa la aplicación cliente (el *frontend* web o la aplicación móvil). Es el punto de entrada para el usuario (login(), registrar(), verAyudantia()). Este componente consume la interfaz (ISP) expuesta por el Sistema Stud IA para operar.
* **Adm. Usuarios:** Es un componente de servicio especializado. Toda la lógica de negocio para la gestión de usuarios (buscarUsuario, modificar, registrar) está encapsulada aquí. El Sistema Stud IA depende de este componente.
* **Gest. Ayudantia:** Otro componente de servicio clave. Es responsable de toda la lógica de las ayudantías (registrarAyudantia, asignarAyudantia). Es una dependencia del sistema central.
* **API Calendario:** Un componente externo que representa un servicio de terceros (ej. Google Calendar, Microsoft Graph API). El componente Gest. Ayudantia depende de esta API para gestionar la funcionalidad de horarios (ver disponibilidad, agendar, etc.).
* **BD (Base de Datos):** Representa la capa de persistencia. En el diagrama, se identifica el uso de **Postgres** y **Docker**, indicando la tecnología de base de datos y su método de despliegue. Todos los componentes de negocio (implícitamente a través del Sistema Stud IA) dependen de este componente para almacenar y recuperar datos.

**Flujo de Interacción (según el diagrama):**

1. El usuario interactúa con la AppPrincipal.
2. La AppPrincipal envía una solicitud a la interfaz ISP del Sistema Stud IA.
3. El Sistema Stud IA recibe la solicitud y, según corresponda, delega la tarea al componente Adm. Usuarios o Gest. Ayudantia.
4. Si se gestiona una ayudantía, Gest. Ayudantia puede necesitar comunicarse con la API Calendario.
5. Los componentes de servicio interactúan con la BD para persistir o consultar información.

### 

### 

### 

### 

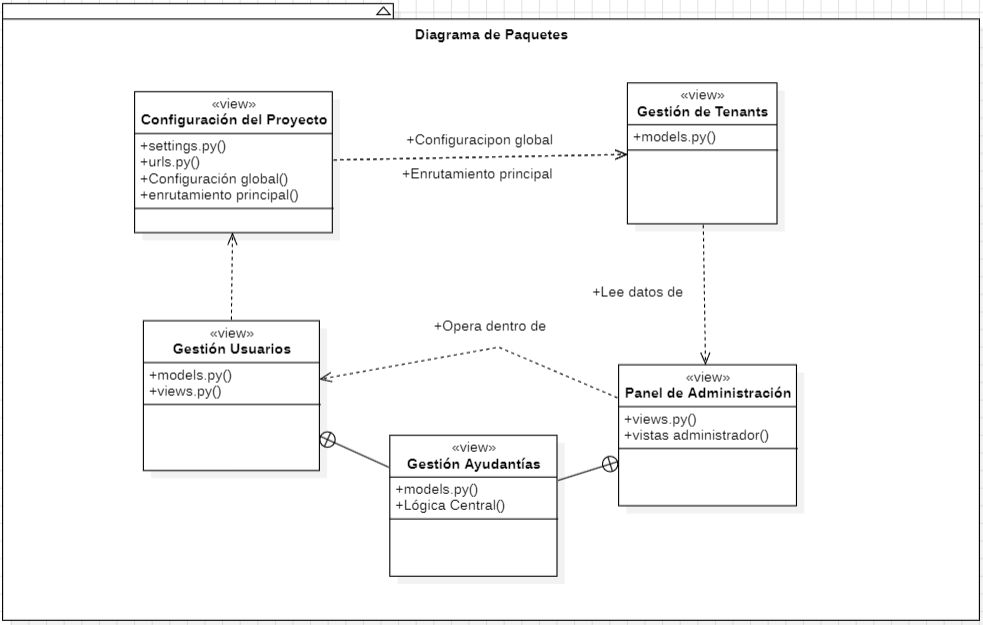
### 

### 

### 

### 

### **3.3. Diagrama de Paquetes (Estructura de Código)**



El diagrama de paquetes ilustra la arquitectura de código de alto nivel de la aplicación Django, mostrando cómo el sistema se divide lógicamente en "apps" modulares.

**Análisis de los Paquetes (Apps de Django):**

El sistema está estructurado en cinco paquetes principales:

1. **Configuración del Proyecto:** Es el paquete raíz del proyecto Django. Contiene los archivos settings.py y urls.py. Su responsabilidad es la "+Configuración global" y la definición del "+Enrutamiento principal" de la aplicación.
2. **Gestión de Tenants:** Este paquete es crucial para la arquitectura, ya que maneja la lógica *multi-tenant*. Contiene los models.py (probablemente el modelo Institucion) que define a cada "cliente" o institución.
3. **Gestión Usuarios:** Es la "app" de Django estándar que gestiona los perfiles, la autenticación y los roles. Contiene los models.py (Usuario, Rol) y los views.py (Login, Registro, Perfil).
4. **Gestión Ayudantías:** Es el corazón del negocio. Esta "app" contiene los models.py (Ayudantia, Inscripcion) y la +Lógica Central del proyecto StudIA.
5. **Panel de Administración:** Es una "app" dedicada que contiene los views.py específicos para las +vistas administrador(), como reportes o validaciones.

**Análisis de Dependencias:**

El diagrama ilustra las siguientes relaciones de dependencia (indicadas por las flechas discontinuas y etiquetas):

* El paquete Configuración del Proyecto actúa como el integrador principal, con dependencias directas hacia Gestión de Tenants y Gestión Usuarios para cargarlos y enrutar las peticiones.
* El Panel de Administración es un consumidor de datos. Depende directamente de Gestión de Tenants (para "+Lee datos de" las instituciones) y de Gestión Ayudantías (para "+Operar dentro de" ellas y mostrar sus datos).
* El diagrama también muestra una fuerte asociación (usando símbolos de agregación) que vincula al Panel de Administración tanto con Gestión Usuarios como con Gestión Ayudantías, lo que indica que el panel se construye funcionalmente sobre los datos y la lógica de estos dos módulos centrales.

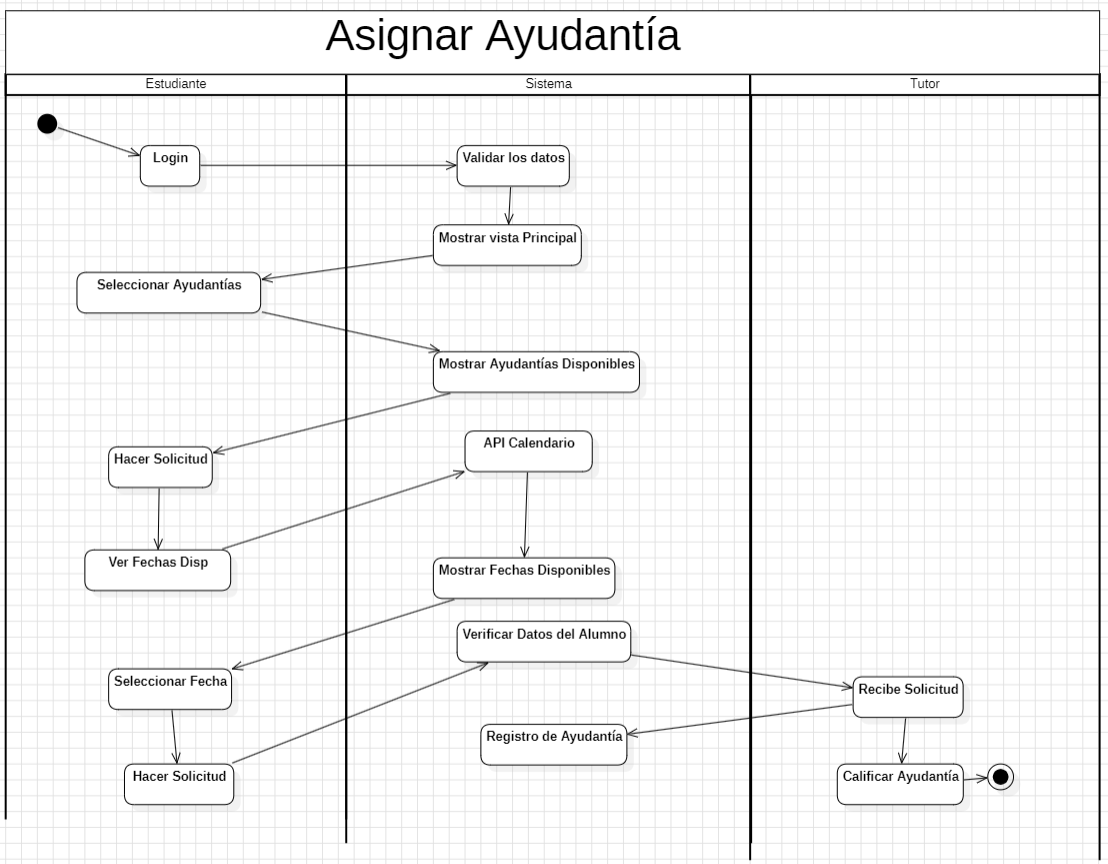
# **4) Vista de Procesos (El "Cuándo")**

### **4.1. Descripción General**

La Vista de Procesos describe el comportamiento dinámico del sistema. Responde a la pregunta: **¿Cómo se ejecuta e interactúa el sistema en tiempo real?**

Esta vista se enfoca en los flujos de trabajo (workflows), la secuencia de eventos y la comunicación entre los diferentes componentes y actores para completar una tarea. El Diagrama de Actividad es la herramienta principal para modelar estos flujos de negocio.

### **4.2. Diagrama de Actividad: "Asignar Ayudantía"**



El siguiente diagrama de actividad modela el flujo de negocio principal para que un estudiante solicite y asigne una ayudantía, mostrando la colaboración entre el Estudiante, el Sistema y el Tutor.

**Análisis del Flujo:**

El proceso se divide en tres carriles (*swimlanes*) que muestran las responsabilidades de cada parte:

1. **Actor Estudiante (Inicio):**
   * El flujo comienza cuando el estudiante realiza un Login.
   * Una vez autenticado, selecciona la opción Seleccionar Ayudantías.
   * El estudiante visualiza las fechas (Ver Fechas Disp) y procede a Seleccionar Fecha y Hacer Solicitud.
2. **Sistema (Mediador y Lógica):**
   * Tras el Login, el sistema realiza la Validar los datos y, si es exitoso, Mostrar vista Principal.
   * En respuesta a Seleccionar Ayudantías, el sistema Mostrar Ayudantías Disponibles.
   * **Integración Clave:** El diagrama muestra que para obtener la disponibilidad, el sistema consulta la API Calendario (un componente externo identificado en la vista anterior).
   * Luego, Mostrar Fechas Disponibles al estudiante.
   * Una vez que el estudiante selecciona una fecha, el sistema Verificar Datos del Alumno.
   * Tras la confirmación final (Hacer Solicitud), el sistema realiza el Registro de Ayudantía en la base de datos (actualizando la tabla INSCRIPCION).
3. **Actor Tutor (Validación):**
   * Después de que el sistema verifica los datos del alumno (o quizás después del registro), el Tutor Recibe Solicitud.
   * El tutor debe realizar una acción de Calificar Ayudantía (que en este contexto de MVP se interpreta como **"Aprobar" o "Confirmar"** la solicitud de inscripción).
   * El proceso finaliza.

Este diagrama es crucial porque valida el Caso de Uso "Inscripción a una Ayudantía", mostrando cómo el Sistema actúa como un intermediario que consume servicios externos (API Calendario) y coordina las acciones entre el Estudiante y el Tutor.

# **5) Vista Física (de Despliegue)**

### **5.1. Descripción General**

La Vista Física, también conocida como Vista de Despliegue, describe la topología del hardware y el software en ese hardware. Responde a la pregunta final: **¿Dónde y cómo se ejecuta el sistema?**

Esta vista mapea los componentes de software (los artefactos compilados o ejecutables) a nodos físicos o virtuales (servidores, contenedores, dispositivos cliente), mostrando cómo se comunican a través de la red. El diagrama proporcionado ilustra una arquitectura de despliegue clásica y robusta para una aplicación web basada en Django.

### 

### 

### 

### 

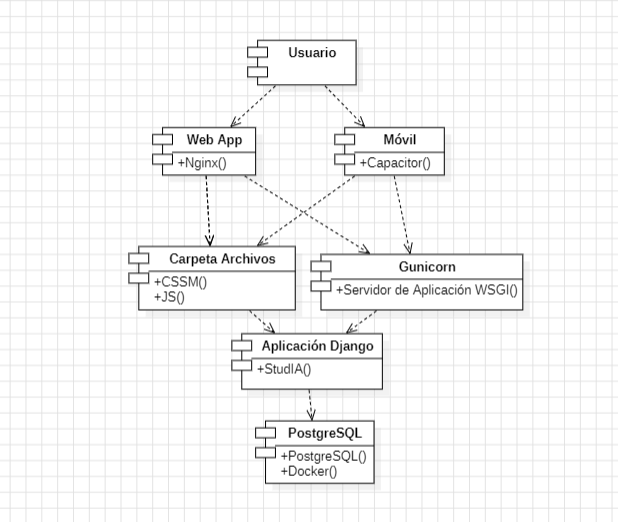
### 

### 

### 

### 

### **5.2. Topología de Despliegue**



El diagrama muestra una arquitectura multicapa que separa claramente las responsabilidades, desde la interacción con el usuario hasta el almacenamiento de datos.

**Análisis de la Topología:**

1. **Capa Cliente (Acceso del Usuario):**
   * **Usuario**: El actor que inicia todas las interacciones.
   * **Web App**: El punto de acceso para usuarios en navegadores de escritorio o móviles. Este componente es servido por **Nginx**, un servidor web de alto rendimiento.
   * **Móvil**: La aplicación nativa instalada en los dispositivos de los usuarios. El diagrama especifica el uso de **Capacitor**, lo que indica que es una aplicación híbrida construida con tecnologías web.
2. **Capa de Servidor Web y Proxy Inverso:**
   * **Nginx (Web App)**: Este componente tiene una doble función crítica:
     1. **Servir Archivos Estáticos:** Entrega directamente el contenido de la Carpeta Archivos (CSS, JavaScript, imágenes) al navegador del usuario. Esto es muy eficiente y descarga de trabajo a la aplicación principal.
     2. **Proxy Inverso:** Recibe las peticiones dinámicas (ej. login, inscribir\_ayudantia) y las reenvía de forma segura y gestionada al servidor de aplicaciones (Gunicorn).
   * **Carpeta Archivos**: Representa el nodo o volumen de almacenamiento donde residen los activos estáticos del frontend.
3. **Capa de Aplicación (Lógica de Negocio):**
   * **Gunicorn**: Es el **Servidor de Aplicación WSGI** (Web Server Gateway Interface). Su función es actuar como intermediario entre el servidor web (Nginx) y la aplicación Python (Django). Gestiona los procesos de trabajo que ejecutan el código de la aplicación de manera concurrente y eficiente.
   * **Aplicación Django (StudIA)**: Este es el corazón del backend. Aquí reside todo el código de la lógica de negocio (los servicios Adm. Usuarios, Gest. Ayudantia, etc.), la conexión a la base de datos y los endpoints de la API que son consumidos tanto por la Web App (a través de Nginx) como por la app Móvil.
4. **Capa de Datos (Persistencia):**
   * **PostgreSQL**: Es el Sistema Gestor de Base de Datos relacional elegido para almacenar toda la información del sistema (usuarios, ayudantías, inscripciones).
   * **Docker**: Se indica que PostgreSQL se ejecuta dentro de un contenedor Docker. Esto facilita enormemente la portabilidad, el despliegue y la gestión de la base de datos, asegurando que el entorno sea consistente desde el desarrollo hasta la producción.

# **6) Vista de Escenarios (Casos de Uso)**

### **6.1. Descripción General**

Esta vista es el "+1" del modelo de arquitectura (4+1 Vistas) y es la que cohesiona a todas las demás. Su propósito es capturar los **requisitos funcionales** del sistema desde la perspectiva de los usuarios. Responde a la pregunta: **¿Qué debe hacer el sistema para los actores?**

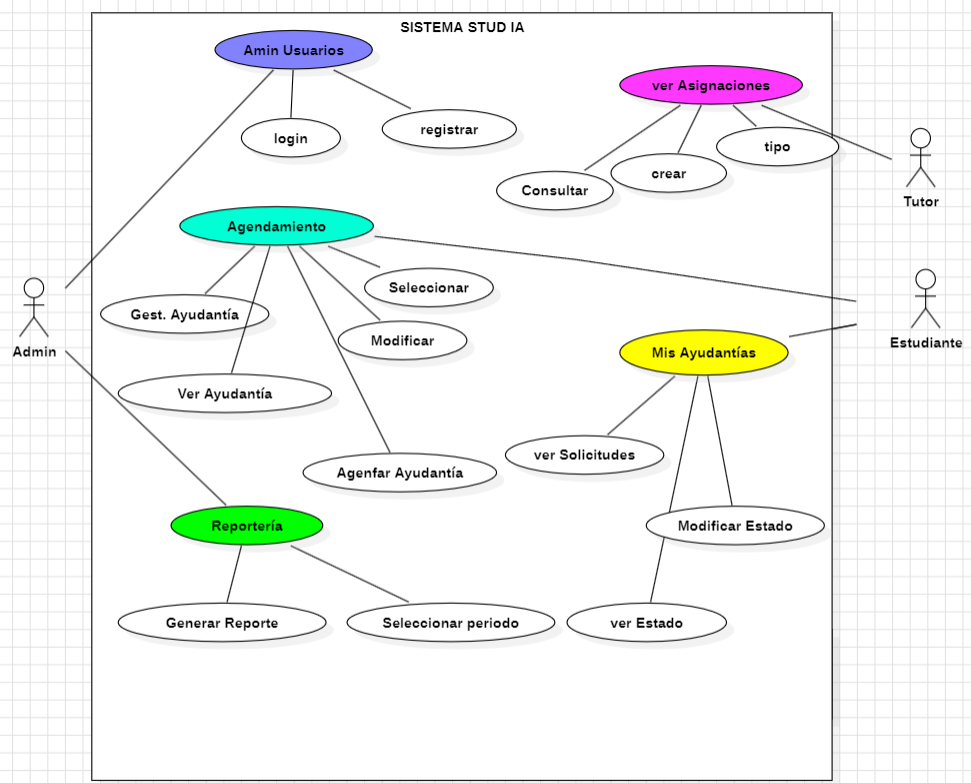
Los casos de uso definen las interacciones entre los actores y el sistema para lograr un objetivo específico.

### **6.2. Actores Identificados**

El diagrama de casos de uso proporcionado identifica a los tres actores clave del sistema StudIA:

* **Admin**: El administrador del sistema, con privilegios de gestión.
* **Tutor**: El usuario que ofrece las ayudantías.
* **Estudiante**: El usuario que consume las ayudantías.

### **6.3. Diagrama: "Caso de uso General"**



El **SISTEMA STUD IA** se organiza en cinco módulos clave. El módulo **"Admin Usuarios"** (azul) es la entrada universal, manejando el login y registrar para todos.

El Tutor utiliza **"ver Asignaciones"** (rosa) para gestionar las materias que imparte (crear, Consultar).

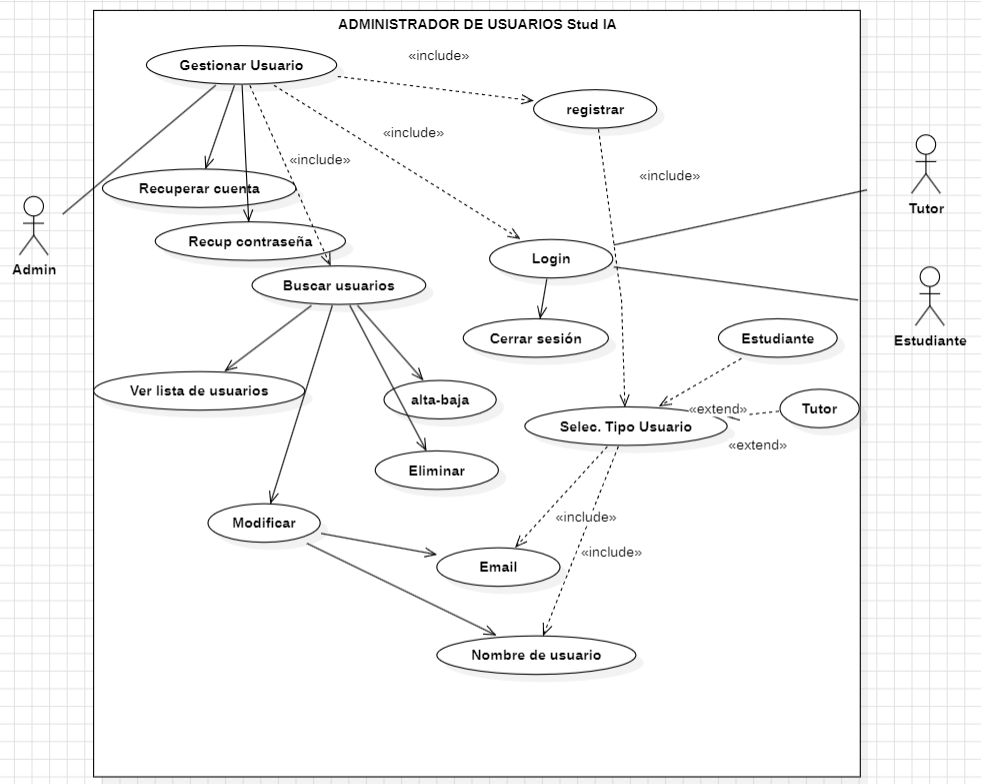
El Estudiante usa **"Agendamiento"** (turquesa) para Seleccionar una ayudantía y **"Mis Ayudantías"** (amarillo) para revisar el ver Estado de sus solicitudes.

El Admin tiene un rol de supervisor: utiliza **"Agendamiento"** para la gestión general (Gest. Ayudantía) y **"Reporteria"** (verde) para Generar Reporte del sistema.

### 

### 

### **6.4. Análisis del Diagrama: "Administrador de Usuarios"**



El diagrama "Administrador de Usuarios" modela el subsistema completo de gestión de identidad y perfiles de la plataforma.

**Funcionalidad Central:**

* **Login (Autenticación):** Es el caso de uso base para todos los actores (Admin, Tutor, Estudiante). El diagrama muestra que es un paso incluido (<<include>>) en acciones protegidas como Gestionar Usuario y registrar.
* **Cerrar sesión:** Complementa al Login, permitiendo a los usuarios salir del sistema de forma segura.
* **registrar (Registro):** Es el punto de entrada para nuevos usuarios. Este caso de uso es fundamental y se detalla a continuación.

**Flujo de Registro (Extensibilidad):**

El proceso de registrar es genérico, pero se especializa según el tipo de usuario:

1. El registro <<include>> obligatoriamente el caso de uso Selec. Tipo Usuario.
2. Selec. Tipo Usuario es un punto de extensión (<<extend>>). Esto significa que el flujo base de selección de tipo *puede ser extendido* por funcionalidades específicas:
   * Si se selecciona Tutor, se pueden ejecutar pasos adicionales (ej. solicitar validación de credenciales).
   * Si se selecciona Estudiante, se pueden ejecutar otros pasos (ej. asociar a una carrera).

**Funcionalidad del Administrador (Admin):**

El actor Admin tiene capacidades de gestión exclusivas, agrupadas en el caso de uso principal Gestionar Usuario:

* **Recuperar cuenta / Recup contraseña:** El admin puede iniciar flujos de recuperación de acceso para otros usuarios.
* **Buscar usuarios:** Esta es la función central de gestión del admin, que a su vez habilita otras acciones:
  + Ver lista de usuarios: Una consulta simple.
  + alta-baja: Permite activar o desactivar una cuenta de usuario sin borrarla.
  + Eliminar: Permite borrar permanentemente a un usuario.
  + Modificar: Permite editar la información de un usuario.

**Flujo de Modificación (Inclusión):**

El caso de uso Modificar es un agrupador que <<include>> acciones más pequeñas y específicas, como la modificación del Email o del Nombre de usuario.

### 

### 

### 

### 

### 

### 

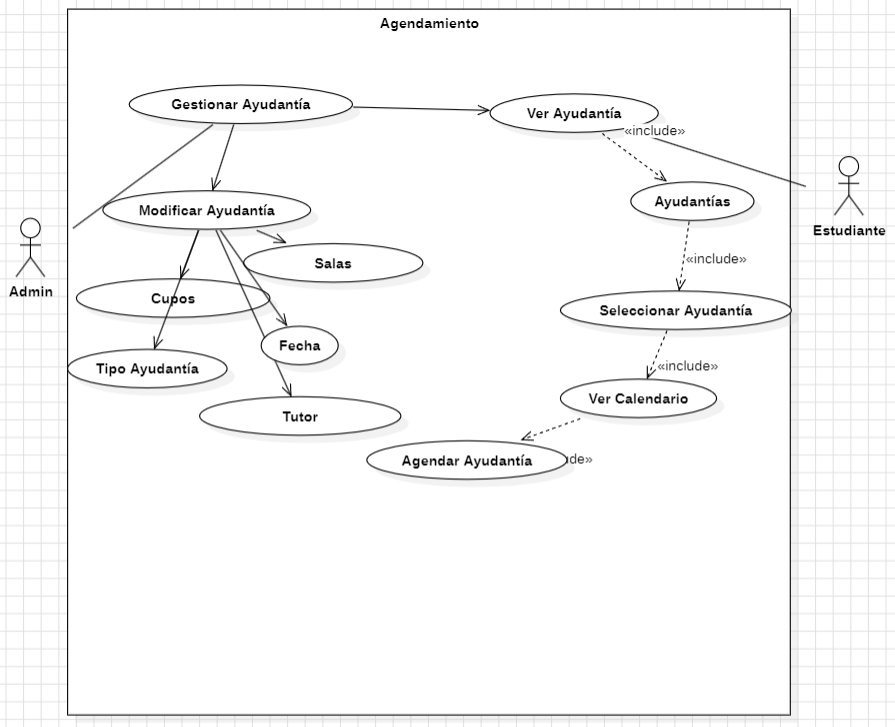
### 

### 

### 

### 

### **6.5. Análisis del Diagrama: "Agendamiento"**



El módulo de **"Agendamiento"** define los flujos de gestión y reserva. El Admin tiene el control total a través de **Gestionar Ayudantía**, que le permite **Modificar Ayudantía** en todos sus detalles (como Cupos, Salas, Fecha, Tipo Ayudantía y Tutor).

El Estudiante inicia su flujo en **Ver Ayudantía**, lo cual <<include>> la visualización de Ayudantías disponibles. Desde allí, puede **Seleccionar Ayudantía**, un proceso que <<include>> la consulta de **Ver Calendario** para verificar fechas, permitiéndole finalmente **Agendar Ayudantía**.

### 

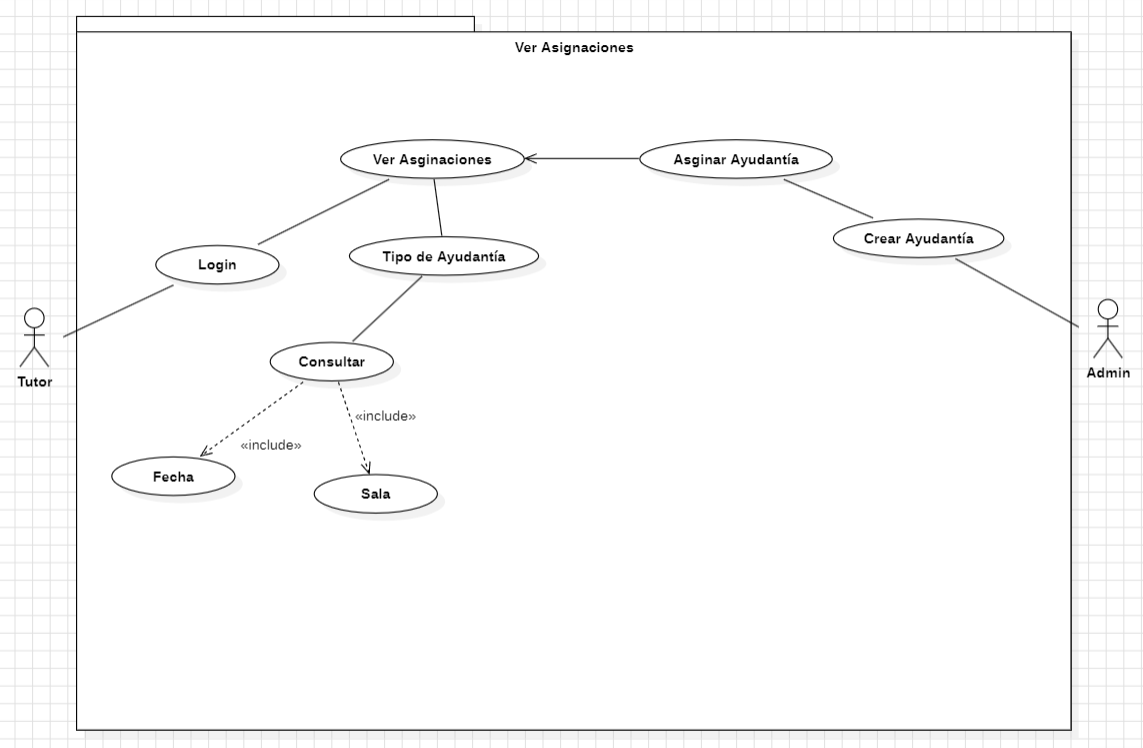
### 

### 

### 

### 

### **6.6. Análisis del Diagrama: "Ver Asignaciones"**



Este módulo define los flujos de creación y consulta de asignaciones. Por un lado, el Admin es responsable de **Crear Ayudantía**, lo que desencadena el proceso de **Asignar Ayudantía**.

Por otro lado, el Tutor inicia su flujo con **Login** para acceder a **Ver Asignaciones**. Desde allí, puede **Consultar** los detalles, un proceso que obligatoriamente <<include>> la revisión de la **Fecha** y la **Sala** de la ayudantía. El diagrama sugiere que las ayudantías asignadas por el Admin son las que el Tutor puede ver.

### 

### 

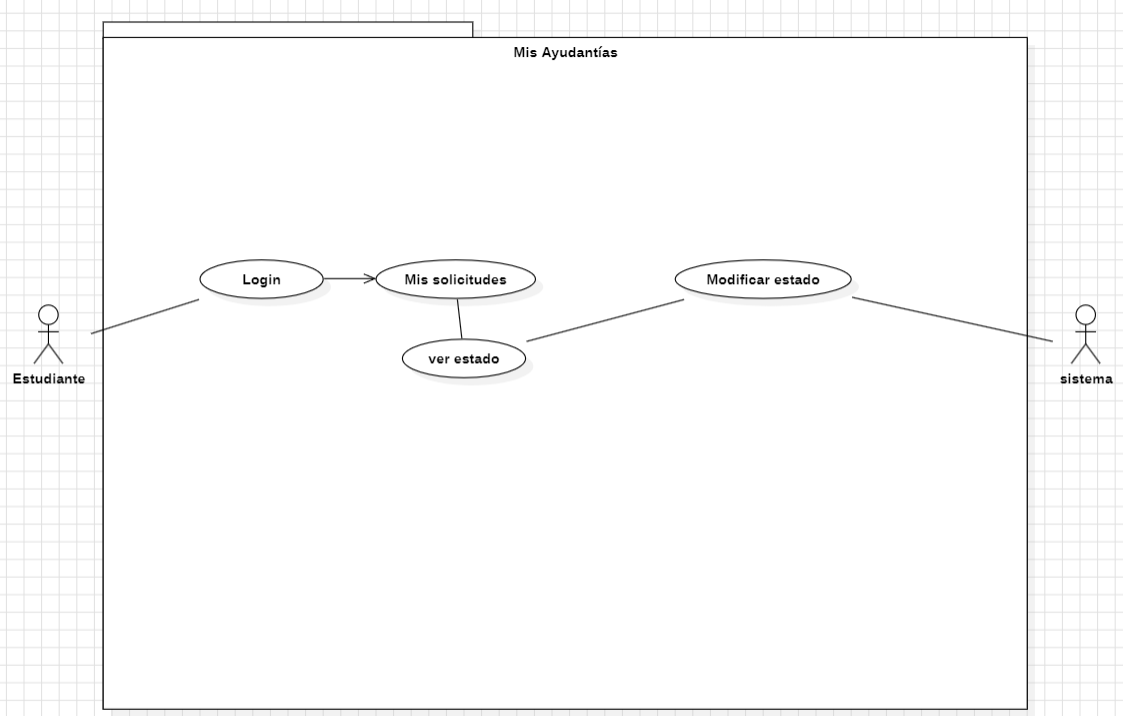
### 

### 

### 

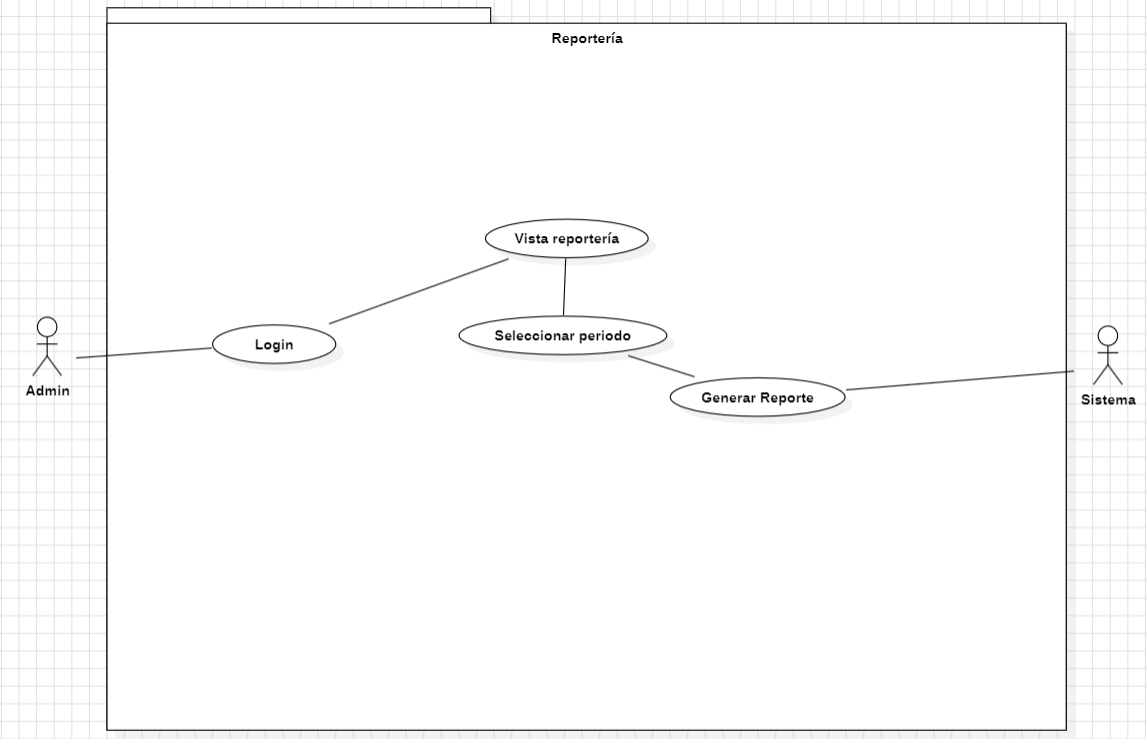
### 

### **6.7. Análisis del Diagrama: "Mis Ayudantías"**



Este módulo representa el panel personal del Estudiante. El flujo comienza cuando el Estudiante realiza **Login** para acceder a **Mis solicitudes**. Desde allí, puede **ver estado** (ej. "Pendiente", "Confirmada"). El diagrama indica que el **sistema** (System) es el actor que ejecuta la acción de **Modificar estado**, lo que sugiere que el estado que ve el estudiante es actualizado automáticamente por el sistema (por ejemplo, cuando un admin aprueba la solicitud).

### **6.8. Análisis del Diagrama: "Reportería"**



Este módulo describe el flujo de generación de informes. El Admin es el único que inicia este caso de uso, comenzando con **Login** para acceder a la **Vista reportería**. Desde allí, el admin puede **Seleccionar periodo** (filtrar por fechas). Una vez seleccionado el periodo, el **Sistema** (System) es el actor que ejecuta la acción de **Generar Reporte**.

# **7) Decisiones de Arquitectura y Tecnologías**

Esta sección consolida y justifica las decisiones arquitectónicas clave tomadas para el **Producto Mínimo Viable (MVP)** de StudIA, basándose en el análisis de las vistas de escenarios, lógica, de implementación, de procesos y de despliegue.

### **7.1. Patrón de Arquitectura**

Se ha seleccionado una combinación de patrones para satisfacer los requisitos del MVP y la escalabilidad futura:

1. **Arquitectura Cliente-Servidor N-Capas:** Es el patrón macro. Como se observa en la Vista Física (Diagrama 5), el sistema está claramente dividido en:
   * **Capa Cliente:** (Web App y Móvil con Capacitor).
   * **Capa Servidor Web / Proxy Inverso:** (Nginx).
   * **Capa de Aplicación:** (Gunicorn + Aplicación Django).
   * **Capa de Datos:** (PostgreSQL en Docker).
2. **Monolito Modular (Backend):** Aunque la Vista de Implementación (Diagrama 3) muestra componentes separados (Adm. Usuarios, Gest. Ayudantia), la Vista Física (Diagrama 5) revela que todos se ejecutan dentro de una única Aplicación Django. Este patrón es una decisión estratégica:
   * **Ventaja para el MVP:** Permite un desarrollo rápido, ya que toda la lógica de negocio comparte la misma base de código, autenticación y capa de datos.
   * **Preparado para Escalar:** Al diseñar el código en módulos cohesivos (como "apps" de Django), se facilita que en el futuro, si el sistema crece, estos módulos (Adm. Usuarios o Gest. Ayudantia) puedan ser extraídos y convertidos en microservicios independientes con un refactor mínimo.

### **7.2. Stack Tecnológico (Confirmado por Diagramas)**

El stack tecnológico se deriva directamente de los diagramas de implementación y despliegue:

* **Backend (Capa de Aplicación):** **Django**. Este *framework* de Python (identificado en la Vista Física) es ideal para un MVP, ya que incluye "baterías incluidas" como un ORM (que mapea al Diagrama E-R), un sistema de autenticación (para el Caso de Uso de Login) y un panel de administrador (Admin) robusto y automático.
* **Servidor de Aplicación (WSGI):** **Gunicorn**. (Vista Física). Es el estándar de oro para ejecutar aplicaciones Django (WSGI) en producción, gestionando los *workers* que atienden las peticiones.
* **Servidor Web / Proxy Inverso:** **Nginx**. (Vista Física). Se utiliza para servir los archivos estáticos (CSSM/JS) de forma eficiente y para actuar como proxy inverso, recibiendo el tráfico HTTPS y pasándolo a Gunicorn.
* **Base de Datos:** **PostgreSQL**. (Vista Física y de Implementación). Una base de datos relacional potente, ideal para manejar las relaciones (FKs) definidas en el Diagrama E-R entre usuarios, ayudantías e inscripciones.
* **Frontend Móvil:** **Capacitor**. (Vista Física). Esta es una decisión clave. Indica que la aplicación móvil será **híbrida (Web-Native)**, probablemente construida con un *framework* como React, Angular o Vue. Esto permite reutilizar la mayor parte del código de la Web App para la Móvil, acelerando drásticamente el desarrollo del MVP.
* **Contenerización:** **Docker**. (Vista Física y de Implementación). Se utiliza para ejecutar PostgreSQL. Esto garantiza la portabilidad y consistencia del entorno de la base de datos entre desarrollo y producción.
* **Integración Externa:** **API Calendario** (genérica). (Vista de Implementación y Procesos). El sistema se integrará con un servicio de calendario externo para gestionar la disponibilidad de horarios, tal como lo requiere el flujo de "Asignar Ayudantía".

### **7.3. Justificación de Decisiones**

Las elecciones tecnológicas están alineadas con los objetivos del proyecto:

* **Django + Gunicorn + Nginx + Postgres** es una de las arquitecturas más probadas, seguras y escalables en el ecosistema de Python. Es perfecta para un MVP que necesita ser robusto desde el primer día. El panel de administrador de Django satisface la mayoría de los requisitos del actor Admin (Diagrama de Casos de Uso) casi sin desarrollo adicional.
* La elección de **Capacitor** es la decisión más estratégica para un **MVP**. En lugar de construir y mantener tres bases de código (backend, web, móvil nativo), se mantienen solo dos (backend y web-híbrida). Esto reduce costos y tiempo de desarrollo, permitiendo validar la hipótesis del producto en ambos canales (web y móvil) rápidamente.
* El uso de un **Monolito Modular** en lugar de microservicios puros en la etapa de MVP reduce la complejidad operacional. No hay necesidad de gestionar redes complejas entre servicios, descubrimiento de servicios o autenticación duplicada, lo que permite al equipo enfocarse en entregar las funcionalidades centrales del MVP.

# **8) Conclusiones**

### **8.1. Resumen de la Arquitectura**

La arquitectura definida para el Producto Mínimo Viable (MVP) de **StudIA** es un sistema **Cliente-Servidor N-Capas**, diseñado para ser robusto, probado y rápido de implementar.

El *backend* se consolida como un **Monolito Modular** desarrollado en **Django**. Esta elección (vista en el Punto 5 y 7) permite un desarrollo ágil al aprovechar las "baterías incluidas" del *framework*, como el ORM (que mapea directamente al Modelo E-R del Punto 2) y el panel de administración (que satisface los Casos de Uso del Admin del Punto 6).

La arquitectura de despliegue (Punto 5) es clara: **Nginx** actúa como *proxy* inverso y servidor de archivos estáticos (CSS/JS), mientras que **Gunicorn** gestiona la ejecución de la aplicación Django. La base de datos relacional **PostgreSQL**, ejecutada en **Docker**, proporciona la persistencia de datos.

Una decisión estratégica clave es el uso de **Capacitor** para el *frontend*. Esto habilita una arquitectura de **aplicación híbrida**, permitiendo que una única base de código sirva tanto a la Web App como a la Móvil, optimizando los recursos de desarrollo del MVP.

En conjunto, esta arquitectura satisface todos los requisitos funcionales (Casos de Uso) y de procesos (Diagrama de Actividad, Punto 4), conectando a los tres actores (Admin, Tutor, Estudiante) a través de componentes de software bien definidos (Punto 3).

### **8.2. Futuras Mejoras o Extensiones Posibles**

Si bien esta arquitectura es ideal para el MVP, está diseñada explícitamente para evolucionar. El diseño como "Monolito Modular" es el primer paso hacia futuras mejoras:

1. **Evolución a Microservicios:** A medida que la plataforma crezca, los módulos más utilizados o complejos (como Adm. Usuarios o Gest. Ayudantia, identificados en el Punto 3) pueden desacoplarse del monolito principal de Django y convertirse en microservicios independientes. Esto permitirá que escalen de forma autónoma.
2. **Panel de Analítica Avanzada:** El "panel administrativo básico" (mencionado en el Punto 1) puede evolucionar a un *dashboard* de *Business Intelligence* (BI). Este módulo podría analizar patrones de uso, ayudantías más demandadas, tasas de inscripción y su correlación con la deserción académica.
3. **Funcionalidades de Comunidad y Tiempo Real:** Se puede implementar un sistema de **chat en tiempo real** (usando WebSockets) para la comunicación directa entre tutores y estudiantes, así como un **sistema de calificación y reseñas** (5 estrellas) para las ayudantías, fomentando la calidad.
4. **Despliegue Nativo en la Nube:** La arquitectura de despliegue (Punto 5) puede migrar de contenedores Docker gestionados manualmente a plataformas de **orquestación (como Kubernetes)** o servicios PaaS (Platform-as-a-Service como AWS Elastic Beanstalk o Google Cloud Run), para obtener autoescalado, alta disponibilidad y un despliegue continuo (CI/CD) más robusto.

# 