**Visión y Alcance del Proyecto**

Plataforma Web de Gestión de Tutorías Académicas

Información General del Proyecto

Nombre del Proyecto: Plataforma Web de Gestión de Tutorías Académicas

Fecha de Elaboración: Junio 2025

Versión del Documento: 1.0

**1. VISIÓN DEL PROYECTO**

1.1 Declaración de Visión

Desarrollar una plataforma web integral que revolucione la gestión de tutorías académicas, creando un ecosistema digital que conecte eficientemente a estudiantes, tutores y coordinadores académicos, facilitando el acceso equitativo a la educación personalizada y optimizando los recursos educativos institucionales.

1.2 Propósito del Proyecto

Crear una solución tecnológica que automatice y optimice el proceso completo de gestión de tutorías académicas, desde la oferta de disponibilidad por parte de los tutores hasta la evaluación final del servicio, eliminando las barreras administrativas y mejorando la experiencia educativa de todos los participantes.

1.3 Justificación del Proyecto

***Problemática Actual***

Gestión manual ineficiente de horarios y solicitudes de tutoría

Falta de transparencia en la disponibilidad de tutores

Procesos administrativos lentos que retrasan el acceso a tutorías

Ausencia de métricas para evaluar la calidad del servicio

Comunicación fragmentada entre estudiantes, tutores y coordinadores

Dificultad para hacer seguimiento del progreso académico

Oportunidades Identificadas

Digitalización de procesos educativos post-pandemia

Demanda creciente de educación personalizada

Disponibilidad de tecnologías web maduras y accesibles

Necesidad institucional de optimizar recursos académicos

Gracias por el contexto. A continuación te redacto el Alcance del Proyecto en un estilo formal, claro y alineado con la visión, propósito y justificación que has proporcionado:

**2. ALCANCE DEL PROYECTO**

2.1 Definición del Alcance

El presente proyecto contempla el diseño, desarrollo e implementación de una Plataforma Web de Gestión de Tutorías Académicas, que permitirá automatizar y centralizar todos los procesos relacionados con la solicitud, asignación, seguimiento y evaluación de tutorías académicas dentro de una institución educativa.

El sistema estará orientado a tres perfiles principales de usuarios: estudiantes, tutores y coordinadores académicos, brindando a cada uno herramientas específicas para cumplir sus funciones de manera eficiente, transparente y orientada a resultados. La plataforma se desarrollará como una aplicación web responsive, accesible desde distintos dispositivos y navegadores modernos.

2.2 Funcionalidades Incluidas

Para Estudiantes:

* Registro y autenticación en la plataforma.
* Consulta de tutores disponibles y sus horarios.
* Solicitud de tutorías seleccionando materia, tutor y franja horaria.
* Seguimiento del estado de sus solicitudes (pendiente, aceptada, rechazada).
* Visualización del historial de tutorías realizadas.
* Calificación y comentarios sobre las tutorías recibidas.
* Notificaciones automáticas sobre cambios en el estado de las solicitudes.

Para Tutores:

* Gestión de perfil profesional y materias que imparte.
* Configuración de disponibilidad horaria para tutorías.
* Recepción y gestión de solicitudes de tutoría.
* Aprobación o rechazo de solicitudes, con inclusión de motivos.
* Acceso al historial de tutorías impartidas.
* Visualización de evaluaciones recibidas por parte de estudiantes.
* Acceso a métricas personales relacionadas con su desempeño.

Para Coordinadores Académicos:

* Supervisión y administración general del sistema.
* Gestión de usuarios: alta, baja y modificación de estudiantes y tutores.
* Generación de reportes sobre uso de la plataforma, participación y desempeño.
* Análisis de indicadores de satisfacción estudiantil y calidad del servicio.
* Configuración de parámetros generales de funcionamiento del sistema.
* Acceso a bitácoras y registros de auditoría para seguimiento de actividades.

Funcionalidades Generales del Sistema:

* Sistema de roles y permisos diferenciados por perfil de usuario.
* Arquitectura basada en una base de datos relacional optimizada.
* Exposición de una API REST para futuras integraciones con otros sistemas.
* Sistema completo de logging y auditoría para trazabilidad de acciones.
* Interfaz web adaptable a dispositivos móviles y de escritorio.
* Despliegue mediante contenedores Docker para facilitar su portabilidad y mantenimiento.

2.3 Límites del Alcance

Este proyecto no incluye el desarrollo de aplicaciones móviles nativas, integraciones inmediatas con plataformas externas de comunicación o LMS, ni la implementación de sistemas de pago o monetización. Estas funcionalidades podrán ser consideradas en fases futuras.

**Dockerfile**

Este Dockerfile define cómo construir una imagen de Docker que contendrá todo lo necesario para ejecutar el proyecto GestorTutorias, basado en Django y Python 3.12.

***Imagen base***

FROM python:3.12-slim

¿Qué hace? Utiliza como base una imagen oficial de Python 3.12 con una variante reducida (slim) para que la imagen final sea más ligera.

¿Por qué? Esto asegura que tenemos un entorno limpio y controlado con Python 3.12 preinstalado.

***Configuración de variables de entorno***

ENV PYTHONDONTWRITEBYTECODE=1

Evita que se creen archivos .pyc (archivos bytecode) innecesarios al ejecutar el código Python.

ENV PYTHONUNBUFFERED=1

Evita el buffering de la salida estándar, lo que facilita el seguimiento de los logs en tiempo real desde el contenedor.

***Directorio de trabajo***

WORKDIR /app

Define /app como directorio de trabajo dentro del contenedor.

Todos los comandos posteriores se ejecutarán desde esta ruta.

***Instalación de dependencias del sistema***

RUN apt-get update \

&& apt-get install -y build-essential libpq-dev gcc \

&& rm -rf /var/lib/apt/lists/\*

Actualiza el índice de paquetes.

Instala:

build-essential: compila paquetes Python con extensiones en C.

libpq-dev: permite compilar el conector psycopg2 para PostgreSQL.

gcc: compilador de C necesario para dependencias nativas.

Limpia cachés para reducir el tamaño final de la imagen.

***Copiar archivos del proyecto***

COPY . /app/

Copia todos los archivos desde el directorio actual del host (tu máquina) al contenedor, dentro de /app.

***Instalación de dependencias de Python***

RUN pip install --upgrade pip \

&& pip install -r requirements.txt

Actualiza pip.

Instala todas las dependencias del proyecto listadas en requirements.txt.

***Cambio de directorio al proyecto Django***

WORKDIR /app/GestorTutorias

Cambia el directorio de trabajo a donde está el proyecto Django principal (manage.py, settings.py, etc.).

***Aplicación de migraciones de Django***

RUN python manage.py makemigrations \

&& python manage.py makemigrations Sistema \

&& python manage.py migrate

Ejecuta los comandos para:

Generar archivos de migración para todos los modelos (makemigrations).

Asegurar que la app Sistema también tiene sus migraciones.

Aplicar las migraciones a la base de datos (migrate).

Esta parte no es común hacerla en tiempo de build, ya que las migraciones requieren acceso a una base de datos activa. Idealmente, estas se deberían ejecutar al iniciar el contenedor, no al construirlo.

***Exposición del puerto***

EXPOSE 8000

Informa que el contenedor escucha por el puerto 8000.

Este es el puerto por defecto del servidor de desarrollo de Django.

***Comando para arrancar el servidor***

CMD ["sh", "-c", " python manage.py runserver 0.0.0.0:8000"]

Define el comando por defecto que se ejecutará cuando el contenedor se inicie.

Lanza el servidor de desarrollo de Django en la dirección 0.0.0.0, para que esté disponible desde fuera del contenedor.

Diagrama

El contenido generado por IA puede ser incorrecto.**Análisis Técnico del Diagrama ER - Plataforma de Tutorías Académicas**

1. **ARQUITECTURA DE HERENCIA Y ESPECIALIZACIÓN**

Patrón de Herencia Implementado

El diseño utiliza herencia por composición (Table-per-Type) en lugar de herencia por tabla única:

USUARIO (Tabla base)

├── TUTOR (Tabla especializada)

├── ALUMNO (Tabla especializada)

└── COORDINADOR (Tabla especializada)

¿Por qué es correcto?

Normalización: Mantiene los datos comunes en una sola tabla (USUARIO)

Flexibilidad: Permite extender cada rol sin afectar otros

Integridad referencial: Las claves foráneas garantizan consistencia

Escalabilidad: Fácil agregar nuevos roles sin modificar estructura existente

Ventajas sobre alternativas:

vs. Tabla única con discriminador: No hay campos NULL innecesarios

vs. Tablas completamente separadas: Evita duplicación de datos de usuario

vs. Herencia múltiple: Mantiene la simplicidad y claridad

1. **DISEÑO DE RELACIONES**

Relación TUTOR → HORARIO (1:N)

TUTOR ||--o{ HORARIO

Justificación técnica:

* Un tutor puede ofertar múltiples horarios (cardinalidad correcta)
* Permite flexibilidad en la disponibilidad del tutor
* La FK en HORARIO evita redundancia
* Facilita consultas de horarios por tutor

Relación ALUMNO → SOLICITUD (1:N)

ALUMNO ||--o{ SOLICITUD

Justificación técnica:

* Un estudiante puede hacer múltiples solicitudes (histórico)
* Permite tracking del comportamiento del estudiante
* Facilita reportes y análisis de demanda

Relación HORARIO → SOLICITUD (1:N)

HORARIO ||--o{ SOLICITUD

Justificación técnica:

* Un horario puede recibir múltiples solicitudes (competencia)
* Permite manejo de lista de espera
* El campo disponibilidad controla la asignación final

1. **ANÁLISIS DE INTEGRIDAD REFERENCIAL**

Claves Primarias

Todas las entidades tienen PK auto-incrementales:

Ventaja: Garantiza unicidad sin dependencias externas

Performance: Los índices sobre enteros son más eficientes

Flexibilidad: Permite cambios en datos de negocio sin afectar relaciones

Claves Foráneas y Cascadas

usuario = models.OneToOneField(usuario, on\_delete=models.CASCADE)

tutor = models.ForeignKey(tutor, on\_delete=models.CASCADE)

CASCADE en relaciones de especialización (TUTOR, ALUMNO, COORDINADOR):

Correcto: Si se elimina un usuario, su rol específico debe eliminarse

Consistencia: Evita registros huérfanos

Integridad: Mantiene la coherencia del modelo de herencia

CASCADE en relaciones de negocio:

HORARIO → TUTOR: Si se elimina tutor, sus horarios se eliminan (correcto)

SOLICITUD → ALUMNO/HORARIO: Si se elimina alumno u horario, sus solicitudes se eliminan (correcto)

FEEDBACK → SOLICITUD: Si se elimina solicitud, su feedback se elimina (correcto)

1. **MANEJO DE ESTADOS Y FLUJO DE NEGOCIO**

Campo estado en SOLICITUD

estado = models.CharField(choices=[('pendiente', 'Pendiente'), ('aceptada', 'Aceptada'), ('rechazada', 'Rechazada')])

* Diseño correcto porque:
* Máquina de estados simple: Flujo claro pendiente → aceptada/rechazada
* Constraint a nivel de BD: Los CHOICES de Django se validan
* Extensibilidad: Fácil agregar nuevos estados
* Auditoría: Con auditlog se trackean cambios de estado
* Campo disponibilidad en HORARIO

Función dual correcta:

* Control de oferta: El tutor puede marcar/desmarcar disponibilidad
* Control de sistema: Se actualiza automáticamente al aceptar solicitud
* Optimización: Evita consultas complejas para mostrar horarios disponibles

1. **VALIDACIONES Y RESTRICCIONES**

Validación de calificación en FEEDBACK

calificacion = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)])

***Implementación correcta:***

Constraint a nivel de aplicación: Django valida antes de guardar

Rango lógico: 1-5 es estándar para calificaciones

Integridad de datos: Evita valores inválidos

Campos opcionales bien definidos

motivo = models.TextField(blank=True, null=True)

Flexibilidad: No todos los rechazos requieren motivo detallado

Usabilidad: No obliga al usuario a completar campos innecesarios

1. **NORMALIZACIÓN Y EFICIENCIA**

Análisis de Formas Normales

Primera Forma Normal (1FN):

* Todos los campos son atómicos
* No hay grupos repetitivos

Segunda Forma Normal (2FN):

* Todas las dependencias no-clave dependen completamente de la PK
* No hay dependencias parciales

Tercera Forma Normal (3FN):

* No hay dependencias transitivas
* Cada campo no-clave depende directamente de la PK
* Optimización de consultas

Campos calculados evitados correctamente:

* No se almacena "cantidad de solicitudes por horario"
* No se almacena "promedio de calificación por tutor"
* Se calculan en tiempo de consulta (más flexible)

1. **EXTENSIBILIDAD Y MANTENIBILIDAD**

Audit Trail implementado

auditlog.register(todas\_las\_entidades)

Ventajas del diseño:

* Trazabilidad completa: Todos los cambios se registran
* Compliance: Necesario para entornos académicos
* Debugging: Facilita identificar problemas de datos
* Patrón de Soft Delete
* Eliminado = models.BooleanField(default=False)

Implementación correcta:

* Solo en USUARIO: Los roles se eliminan por CASCADE
* Preservación de datos: Mantiene integridad referencial
* Reversibilidad: Permite "deshacer" eliminaciones

1. **CONSIDERACIONES DE SEGURIDAD**

Separación de responsabilidades:

* Autenticación: Delegada a AbstractUser de Django
* Autorización: Implementable via roles (tutor/alumno/coordinador)
* Auditoría: Centralizada en auditlog

Prevención de inconsistencias:

Constraints FK: Evitan referencias inválidas

Validadores: Previenen datos inválidos

Estados definidos: Evitan transiciones ilegales

**CONCLUSIÓN TÉCNICA**

El diagrama ER presenta un diseño técnicamente sólido porque:

* Cumple principios SOLID: Responsabilidad única por entidad
* Sigue patrones establecidos: Herencia por composición, audit trail
* Mantiene integridad: Constraints apropiadas, cascadas correctas
* Es escalable: Fácil extensión sin romper funcionalidad existente
* Optimiza performance: Estructura eficiente para consultas comunes
* Facilita mantenimiento: Código limpio y relaciones claras

El diseño demuestra comprensión profunda de:

* Modelado de datos relacional
* Patrones de Django/ORM
* Requisitos de negocio académico
* Mejores prácticas de desarrollo