Manual técnico MyMathx

Esteban Rafael Perez Maya

Análisis y Desarrollo de Software

Sena

2025

1. Introducción

1.1. Propósito del Documento

Este manual técnico proporciona una descripción detallada de la arquitectura, componentes, configuración y procedimientos de mantenimiento de la aplicación web MyMathX. Está dirigido a desarrolladores, administradores de sistemas y personal técnico encargado del despliegue y la gestión de la plataforma.

1.2. Descripción General del Sistema

MyMathX es una página web educativa diseñada para ayudar a estudiantes a practicar y mejorar sus habilidades en cálculo, específicamente en temas de límites y derivadas. La aplicación combina un entorno de aprendizaje gamificado, a través de un juego de tablero interactivo, con un tutor de inteligencia artificial que proporciona explicaciones paso a paso de los problemas.

1.3. Audiencia

Este documento está dirigido a:

- **Desarrolladores Backend:** Para entender la lógica de negocio, la estructura de la base de datos y la integración con servicios externos.
- **Desarrolladores Frontend:** Para comprender cómo interactuar con los endpoints del API y la lógica del juego.
- **Administradores de Sistemas (SysAdmins):** Para realizar el despliegue, la configuración del servidor y el mantenimiento de la aplicación en un entorno de producción.

2. Arquitectura del Sistema

2.1. Arquitectura General

MyMathX sigue una arquitectura de cliente-servidor monolítica basada en el framework Django.

- **Backend (Servidor):** Gestiona toda la lógica de negocio, autenticación de usuarios, interacciones con la base de datos, y sirve tanto las plantillas HTML como los datos a través de una API RESTful.

- **Frontend (Cliente):** Se renderiza en el navegador del usuario. Utiliza HTML, CSS y JavaScript para crear la interfaz de usuario, la interactividad del juego y para comunicarse con el backend de forma asíncrona (AJAX/Fetch) para obtener datos dinámicos.
- **Servicios Externos:** Se integra con la API de Google Gemini para la generación de explicaciones matemáticas.

2.2. Stack Tecnológico

- **Lenguaje Backend:** Python 3.x
- **Framework Web:** Django 4.x / 5.x
- **Base de Datos:** MySQL (a través del conector pymysql)
- **Servidor de Desarrollo:** Django Development Server
- **Librerías Python Clave:**
    - django: Framework principal.
    - google-generativeai: Cliente para la API de Gemini.
    - bcrypt: Hashing de contraseñas.
    - pymysql: Conector para la base de datos MySQL.
- **Tecnologías Frontend:**
    - HTML5, CSS3, JavaScript (ES6+).
    - **Three.js:** Biblioteca para el renderizado del dado en 3D.
    - **Oimo.js:** Motor de físicas para la simulación del comportamiento del dado.
    - Fetch API: Para la comunicación asíncrona con el backend.

2.3. Estructura de Directorios del Proyecto

La estructura del proyecto sigue la lógica de Django:

```
mymathx_django/
├── .venv/                  # Entorno virtual de Python
├── mymathx_django_proyecto/ # Directorio de configuración del proyecto
│   ├── settings.py         # Configuración principal
│   ├── urls.py             # URLs globales
│   └── wsgi.py             #
├── app_mymathx/            # App principal de Django
│   ├── migrations/
│   ├── static/             # Archivos estáticos (CSS, JS, imágenes)
│   ├── templates/          # Plantillas HTML
│   ├── models.py           # Modelos de la base de datos
│   ├── views.py            # Lógica de las vistas
│   ├── urls.py             # URLs de la app
│   └── admin.py
├── tutor_ia/               # App para la integración con IA
│   ├── views.py            # Vista para generar explicaciones
```

3. Configuración del Entorno de Desarrollo

1. **Prerrequisitos:**
    - Python 3.10 o superior.
    - Git.
    - Un servidor de base de datos MySQL en ejecución.
2. **Crear y Activar Entorno Virtual:** python -m venv .venv source .venv/bin/activate (En Linux/macOS) o .\.venv\Scripts\Activate.ps1 (En Windows PowerShell)
3. **Instalar Dependencias:** Crear un archivo requirements.txt con las librerías necesarias y ejecutar: pip install -r requirements.txt
4. **Configurar Base de Datos:** En mymathx_django_proyecto/settings.py, configurar la sección DATABASES con las credenciales de MySQL.
5. **Aplicar Migraciones:** python manage.py migrate
6. **Ejecutar Servidor de Desarrollo:** python manage.py runserver

4. Descripción de Módulos (Apps de Django)

4.1. App Principal (app_mymathx)

Esta app contiene el núcleo de la funcionalidad de MyMathX.

- **models.py (Modelos Principales):**
    - UsuarioCuentaSuscripcion: Almacena la información de la cuenta principal (administrador), incluyendo datos de suscripción y credenciales.
    - ProgresoCuentaJugador: Contiene todos los datos de progreso de un jugador (nivel, EXP, monedas, esmeraldas, etc.) y está enlazada a una UsuarioCuentaSuscripcion.
    - Objeto: Define los ítems que se pueden obtener en el juego, como fotos de perfil, con su precio y tipo.

    - ProgresoObjeto: Tabla intermedia que registra qué objetos posee cada jugador.
    - EjercicioTablero: Almacena los ejercicios matemáticos (enunciado, LaTeX, respuesta, dificultad).

- **views.py (Vistas Clave):**
    - **Autenticación y Cuentas:**
        - register: Procesa el registro de una nueva cuenta de administrador, creando también una cuenta de jugador personal asociada.
        - auth: Maneja el inicio de sesión, diferenciando entre cuentas de administrador y cuentas de jugador.
        - enviar_correo_verificacion y verificar_email: Gestionan el flujo de verificación de correo electrónico.
    - **Juego y Perfil:**
        - mapa_juego: Renderiza la página principal del juego.
        - perfil_estudiante y perfil_admin: Muestran los perfiles correspondientes.
        - tienda: Genera los objetos disponibles en la tienda para un jugador.
    - **API Interna (Endpoints):**
        - obtener_ejercicio_aleatorio: Devuelve un ejercicio aleatorio en formato JSON.
        - comprar_objeto: Procesa la compra de un ítem de la tienda, verificando el saldo y actualizando el progreso.
        - guardar_campo: Endpoint genérico para actualizar datos del jugador (como la foto de perfil) en la base de datos.

4.2. App de IA (tutor_ia)

Módulo dedicado exclusivamente a la interacción con la IA generativa.

- **views.py (Vistas Clave):**
    - generar_explicacion_ia:
        - **Función:** Actúa como un proxy seguro entre el frontend y la API de Google Gemini.
        - **Entrada:** Recibe una petición POST con un JSON que contiene enunciado, ejercicio (en LaTeX) y respuesta.
        - **Proceso:** Construye un promete combinado (un SYSTEM_PROMPT predefinido y los datos del ejercicio) y lo envía al modelo gemini-2.5-flash.
        - **Salida:** Devuelve la respuesta de la IA (un JSON con la explicación paso a paso) al cliente.
    - **Gestión de API Key:** La clave de la API de Google **NO DEBE** estar hardcodeada en el código. Se debe gestionar a través de variables de entorno cargadas en settings.py.

5. API y Endpoints Clave

La comunicación dinámica del frontend se realiza a través de los siguientes endpoints:

- **POST /generate-explanation/**
    - **Descripción: Solicita una explicación detallada de un ejercicio a la IA.**
    - **Request Body: {"enunciado": "...", "ejercicio": "...", "respuesta": "..."}**
    - **Response: {"pasos": [{"explicacion": "...", "ecuacion": "..."}, ...]} o {"error": "..."}**
- **GET /obtener_ejercicio_aleatorio/**
    - **Descripción: Obtiene un ejercicio aleatorio de la base de datos.**
    - **Response: {"status": "success", "ejercicio": {"id": ..., "ejercicio": "...", ...}}**
- **POST /comprar_objeto/**
    - **Descripción: Procesa la compra de un ítem de la tienda.**

    - **Request Body: {"id_objeto": 123}**
    - **Response: {"success": true, "nuevas_monedas_oro": ..., "nuevas_esmeraldas": ...}**
- **POST /guardar_campo/**
    - **Descripción: Actualiza un campo específico del progreso del jugador.**
    - **Request Body: {"campo": "imagen_perfil", "valor": "05.png"} o {"campo": "progreso_completo", "valor": {"monedas_oro": ..., ...}}**
    - **Response: {"success": true, "message": "..."}**
    -

6. Mantenimiento y Solución de Problemas Técnicos

- **Logs de Django:** La consola donde se ejecuta runserver es la primera fuente para depurar errores 500 (Internal Server Error) durante el desarrollo. En producción, los logs deben ser redirigidos a un archivo.
- **Errores Comunes:**
    - **ModuleNotFoundError:** Generalmente indica que el entorno virtual no está activado o que una dependencia no está instalada.
    - **Errores de API de Gemini (4xx/5xx):**
        - **404 Model Not Found:** El nombre del modelo (gemini-2.5-flash) es incorrecto o no está disponible.
        - **403/401 Permission Denied:** La API Key es inválida, ha sido revocada o no tiene permisos para usar el modelo.
    - **Errores de Base de Datos:** Verificar que el servicio de MySQL esté en ejecución y que las credenciales en settings.py sean correctas. Si se realizan cambios en models.py, es necesario ejecutar python manage.py makemigrations y python manage.py migrate.