MINISTRY MAGIC SYSTEM

Este proyecto consiste en un sistema avanzado de gestión de hechizos y eventos mágicos de un ministerio de magia.

Para el implementamos un sistema backend modular en Python utilizando fast api. El sistema esta principalmente desarrollado siguiendo una programación orientada a aspectos para manejar acciones transversales sin contaminar lógica de negocio principal. Además, utilizamos la concurrencia ya que varios hechizos pueden ser lanzados a la vez.

Luego se realizó un frontend de prueba en la que puedes seleccionar o crear un personaje y probar sus hechizos en una simulación de batalla.

Arquitectura

-Patrón Factory:

Para la creación de hechizos y eventos se utiliza este patrón. Primero en models/ se implementan los modelos de cada uno y luego utilizamos las fabricas para crear los diferentes tipos de hechizos o eventos.

-AOP:

- “@validate_magic_permission” (seguridad dinamica): Intercepta la llamada verifica el rol de usuario contra el catalogo de hechizos y eventos y permite o deniega el acceso.

- “@audit_log”: Registra automáticamente las acciones que ocurren y quien las realiza. Se guarda cada dato en un archivo .log además de imprimirse por consola.

- “@magic_transaction”: Garantiza la estabilidad si un hechizo o evento falla, captura el error y simula un rollback.

-Concurrencia:

El sistema es asíncrono, permite que múltiples magos lancen hechizos simultáneamente sin bloquear el servidor. (simulamos el tiempo con asyncro.sleep).

-Inyección de dependencias:

Utilizamos el mecanismo nativo de FastAPI para inyectar dependencias.

La función get_current_user inyecta el objeto User actual en los endpoints basándose en los headers HTTP, desacoplando así la lógica de autentificación de la lógica del endpoint.

Componentes Clave

-magic_catalog.py: Es donde están definidos los hechizos y eventos disponibles y sus propiedades además de los que roles que puede acceder a ellos. También tenemos las propiedades que se le da a cada rol.

-database.py: Una simulación de base de datos donde se guarda cada personaje.

-decorators.py: Este es el componente clave en el programa. En el encontramos:

| **validate_magic_permission**: Su trabajo es decidir si la petición debe ser procesada o rechazada antes de que llegue al servicio.

Extrae los argumentos user y magic_name que se le pasan a la función decorada. Una vez encontrado el hechizo, mira su campo perm en el catálogo ej: "Fireball" requiere cast_advanced. Llama al método user.has_permission() comprobando si el usuario tiene el rol suficiente para lanzar ese hechizo. Si lo pasa permite la ejecución.

| audit_log: Esta función es la encargada de registrar lo que pasa y el tiempo que tarda.

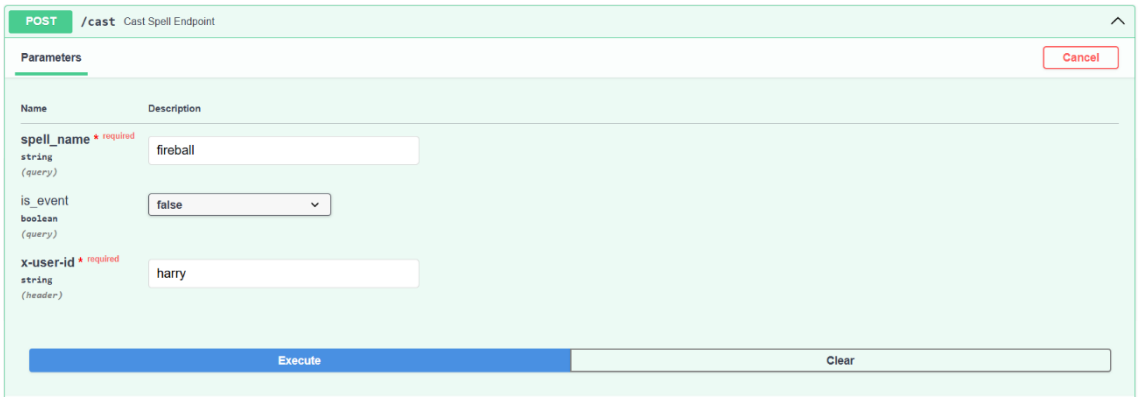
Intenta ejecutar la función original (await func(...)). Si tiene éxito, escribe AUDIT [SUCCESS] en el log. Si la función falla, captura el error, escribe AUDIT [FAILURE] en el log y vuelve a lanzar el error (raise e) para que no se oculte. calcula la duración (end_time - start_time) y llama a record_spell_metric. Esto envía los datos a Prometheus y al array que alimenta tu gráfica en el Dashboard.

| magic_transaction: Esta es la encargada de simular el comportamiento de una base de datos, implementa el manejo de Transacciones. Si la función se ejecuta bien, retorna el resultado. Esto simularía un commit en base de datos. Si ocurre cualquier error dentro del servicio (o en los decoradores anteriores), entra al except.

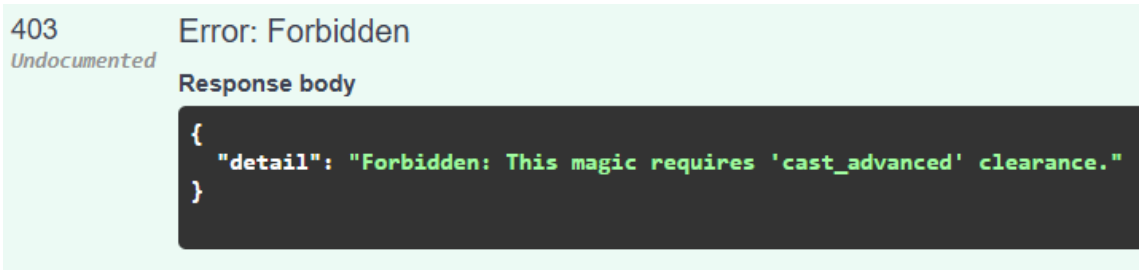
-monitoring.py / dashboard.py: Se utiliza Prometheus para medir cuantos hechizos se lanzan y cuanto tardan. Tambien utilizamos Matplotlib para generar gráficos en tiempo real convertidos a Base64 para mostrarlos en un dashboard HTML simple en /dashboard/stats

DOCS DE FASTAPI

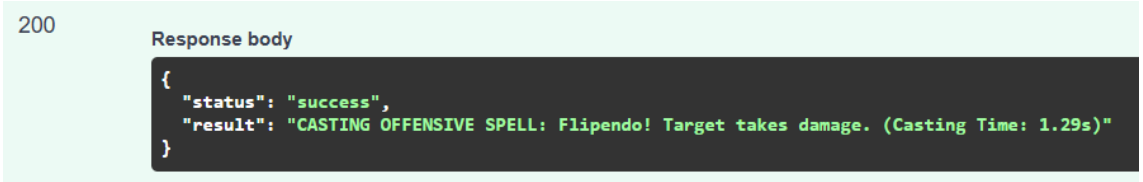
En /docs podemos probar la función de que un usuario lance un hechizo de la siguiente manera:



En este caso el usuario no tiene el rol suficiente para lanzar un hechizo por lo que devuelve un error.



En el caso de ser un hechizo que si pueda lanzar la respuesta seria:



Frontend

El frontend esta construido principalmente en static/index.html

En el se puede elegir un personaje o crearlo (mete un nuevo personaje en la base de datos temporalmente).

Luego hay un sistema de batallas de prueba que tiene un sistema de turnos con bloqueo.
1.Turno Jugador: Selecciona acción -> Selecciona objetivo (si es ofensivo).

2.Bloqueo: La interfaz se deshabilita visualmente.

3.Turno Enemigo: Los enemigos (IA simple aleatoria) atacan secuencialmente con retardos para mejorar la UX.

Esta integrado con el backend, se llama a /cast en segundo plano al lanzar un hechizo o evento. Esto activa los decoradores del servidor además de que se registraran las acciones en los logs.

Flujo de petición

Se envia POST /cast?spell_name=Fireball con header x-user-id: harry

||

FastApi recibe la peticion

||

get_current_user busca "usuario" en database.py y crea el objeto User

||

Se pasan los datos al MAgicService

||

Decorador Transaction: Abree contexto de transacción

||

Decorador Audit: Registra la acción

||

Decorador Security: Busca la acción verifica que el usuario tenga permiso y lanza la respuesta

Ejecución

Para la ejecución se tiene que instalar las siguientes dependencias:

fastapi, uvicorn, psutil, prometheus-client matplotlib, httpx

Se inicia el servidor desde la carpeta raíz con:

``` uvicorn app.main:app --reload```

Rutas dentro del Sistema:

-Simulador de Batalla: http://127.0.0.1:8000/static/index.html

-Dashboard de Métricas: http://127.0.0.1:8000/dashboard/stats

-Documentación API (Swagger): http://127.0.0.1:8000/docs
```