Documentación Entorno Moodle + API Tutor

API Tutor Virtual

Descripción General

Requisitos del Sistema

- **GPU**: NVIDIA GeForce RTX 4050 Laptop con 6 GB de VRAM
- **Controlador NVIDIA**: Versión 572.47 con soporte para CUDA 12.8.
- **Python**: Versión 3.11.11
- **Espacio en Disco**: Aproximadamente 10-15 GB para el entorno virtual y las dependencias.
- **Conexión a Internet**: Necesaria para descargar las dependencias y el modelo.

Configuración del Entorno

Crear el Entorno Virtual

Se utiliza **Conda** para crear un entorno virtual con las dependencias especificadas en un archivo YAML.

Paso: Ejecuta el siguiente comando en la consola para crear el entorno virtual:

```
conda create -n env_api_tutor python=3.11 -y
```

- **Nombre del entorno**: Reemplaza env_api_tutor con el nombre deseado para el entorno (por ejemplo, tutor_api).

Instalar PyTorch con Soporte para CUDA

PyTorch debe instalarse desde el índice oficial de PyTorch para garantizar compatibilidad con CUDA 12.8 y la GPU NVIDIA RTX 4050.

```
pip install torch==2.6.0+cu124 torchaudio==2.6.0+cu124 torchvision==0.21.0+cu124 --index-url https://download.pytorch.org/whl/cu124
```

Instalar Dependencias Adicionales

Las siguientes dependencias son necesarias para ejecutar la API y el modelo. Se deben instalar en el entorno virtual activado:

```
pip install accelerate==1.5.2 aiohappyeyeballs==2.6.1 aiohttp==3.11.14 aiosignal==1.3.2 annotated-types==0.7.0 anyio==4.9.0 attrs==25.3.0 bitsandbytes==0.45.4 certifi==2025.1.31 charset-normalizer==3.4.1 click==8.1.8 cut-cross-entropy==25.1.1
```

```
datasets==3.5.0 diffusers==0.32.2 dill==0.3.8 docstring-parser==0.16 fastapi==0.115.12 filelock==3.18.0 frozenlist==1.5.0 fsspec==2024.12.0 h11==0.14.0 hf-transfer==0.1.9 huggingface-hub==0.29.3 idna==3.10 iniconfig==2.1.0 jinja2==3.1.6 markdown-it-py==3.0.0 markupsafe==3.0.2 mdurl==0.1.2 mpmath==1.3.0 multidict==6.2.0 multiprocess==0.70.16 networkx==3.4.2 numpy==2.2.4 pandas==2.2.3 peft==0.15.1 pillow==11.1.0 pluggy==1.6.0 propcache==0.3.1 protobuf==3.20.3 pyarrow==19.0.1 pydantic==2.11.0 pydantic-core==2.33.0 pypdf2==3.0.1 pytest==8.3.5 pytz==2025.2 pyyaml==6.0.2 regex==2024.11.6 requests==2.32.3 rich==13.9.4 safetensors==0.5.3 sentencepiece==0.2.0 shtab==1.7.1 sniffio==1.3.1 starlette==0.46.1 sympy==1.13.1 tokenizers==0.21.1 tqdm==4.67.1 transformers==4.50.2 triton-windows==3.2.0.post17 trl==0.15.2 typeguard==4.4.2 typing-inspection==0.4.0 tyro==0.9.17 tzdata==2025.2 unsloth==2025.3.19 unsloth-zoo==2025.3.17 urllib3==2.3.0 uvicorn==0.34.0 xformers==0.0.29.post3 xxhash==3.5.0 yarl==1.18.3
```

Nota: Asegurarse de que el entorno virtual esté activado antes de ejecutar este comando:

```
conda activate env_api_tutor
```

Código de la API

La API está implementada en un archivo Python (por ejemplo, app.py) y utiliza FastAPI para manejar solicitudes HTTP. A continuación, se describe su estructura y funcionalidad.

Estructura del Código

El código de la API realiza las siguientes tareas:

1. **Inicialización**:
    - Crea una aplicación FastAPI con el título "API de Modelo Afinado".
    - Configura CORS para permitir solicitudes desde orígenes específicos (http://localhost y http://192.168.67.76, para integración con Moodle).
2. **Carga del Modelo**:
    - Carga un modelo afinado (lora_model_3Instruct_bnb_new) usando la biblioteca **Unsloth**.
    - Utiliza cuantización de 4 bits (load_in_4bit=True) para optimizar el uso de memoria en la GPU.
    - Configura el modelo para inferencia con FastLanguageModel.for_inference.
3. **Endpoints**:
    - **GET /**: Ruta raíz que devuelve un mensaje de confirmación de que la API está funcionando.
    - **POST /generar**: Recibe una solicitud con una instrucción, una entrada opcional y el número máximo de tokens a generar, y devuelve una respuesta generada por el modelo.

Código Fuente

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from unsloth import FastLanguageModel
import torch
from transformers import TextStreamer
import traceback

app = FastAPI(title="API de Modelo Afinado")

# Habilitar CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost", "http://192.168.67.76"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class GenerationRequest(BaseModel):
    instruccion: str
    entrada: str = ""
    max_nuevos_tokens: int = 200

# Cargar el modelo y tokenizador
max_seq_length = 2048
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="lora_model_3Instruct_bnb_new",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)

FastLanguageModel.for_inference(model)

alpaca_prompt = """A continuación hay una instrucción que describe una tarea, junto con una entrada que proporciona más contexto. Escribe una respuesta que complete adecuadamente la solicitud.

### Instrucción:
{}

### Entrada:
{}

### Respuesta:
{}"""

@app.post("/generar")
async def generar_texto(request: GenerationRequest):
    try:
```

```
        prompt = alpaca_prompt.format(request.instruccion, request.entrada, "")
        inputs = tokenizer([prompt], return_tensors="pt").to("cuda")
        text_streamer = TextStreamer(tokenizer)
        outputs = model.generate(
            **inputs,
            max_new_tokens=request.max_nuevos_tokens,
            use_cache=True,
            streamer=text_streamer
        )
        texto_generado = tokenizer.batch_decode(outputs, skip_special_tokens=True)[0]
        inicio_respuesta = texto_generado.find("### Respuesta:") + len("### Respuesta:\n")
        respuesta = texto_generado[inicio_respuesta:].strip()
        return {"respuesta": respuesta}
    except Exception as e:
        traceback.print_exc()
        raise HTTPException(status_code=500, detail=f"Error al generar texto: {str(e)}")

@app.get("/")
async def raiz():
    return {"mensaje": "La API está funcionando. Usa POST /generar para generar texto."}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Dependencias Principales

- **FastAPI (0.115.12)**: Framework para construir la API web.
- **Unsloth (2025.3.19)**: Biblioteca para cargar y optimizar modelos de lenguaje con cuantización.
- **Transformers (4.50.2)**: Proporciona herramientas para manejar modelos y tokenizadores.
- **PyTorch (2.6.0+cu124)**: Framework de aprendizaje profundo con soporte para CUDA.
- **Pydantic (2.11.0)**: Validación de datos para las solicitudes de la API.
- **Uvicorn (0.34.0)**: Servidor ASGI para ejecutar la API.

Configuración del Modelo

- **Nombre del Modelo**: lora_model_3Instruct_bnb_new (debe estar disponible localmente o en Hugging Face).
- **Cuantización**: 4 bits (load_in_4bit=True) para reducir el uso de memoria.
- **Longitud Máxima de Secuencia**: 2048 tokens.
- **Dispositivo**: CUDA (la GPU NVIDIA RTX 4050 se utiliza para acelerar la inferencia).

Ejecución de la API

1. Activar el Entorno Virtual:

```
conda activate new_env_name
```

2. Ejecutar la API:

- Asegúrate de que el archivo Python (por ejemplo, app.py) esté en el directorio actual.
- Ejecuta

```
python app.py
```

- La API estará disponible en http://0.0.0.0:8000.

3. Probar la API:

- **Ruta GET /**: Accede a http://localhost:8000 para verificar que la API está funcionando.
- **Ruta POST /generar**: Envía una solicitud POST con el siguiente formato JSON:

```
{
   "instruccion": "Escribe un resumen sobre inteligencia artificial.",
   "entrada": "La inteligencia artificial es un campo en rápido crecimiento...",
   "max_nuevos_tokens": 200
}
```

Ejemplo de Respuesta

Solicitud

```
{
   "instruccion": "Escribe un resumen sobre inteligencia artificial.",
   "entrada": "La inteligencia artificial es un campo en rápido crecimiento...",
   "max_nuevos_tokens": 200
}
```

Respuesta:

```
{
    "respuesta": "La inteligencia artificial (IA) es un campo de la informática que se centra en desarrollar sistemas capaces de realizar tareas que requieren inteligencia humana, como el aprendizaje, la resolución de problemas y la toma de decisiones. Con avances en el aprendizaje profundo y el procesamiento de datos masivos, la IA ha transformado industrias como la medicina, la logística y el entretenimiento."
}
```

Notas Adicionales

- **CORS**: Configurado para permitir solicitudes desde http://localhost y http://192.168.67.76 (ideal para integraciones con Moodle).
- **Errores**: La API maneja excepciones y devuelve un código de estado 500 con detalles si ocurre un error durante la generación de texto.
- **Rendimiento**: La cuantización de 4 bits y el uso de CUDA optimizan el rendimiento en la GPU RTX 4050, pero asegúrate de que la VRAM no se sature.
- **Modelo Local**: El modelo lora_model_3Instruct_bnb_new debe estar disponible en el directorio local.

Solución de Problemas

- **Error de CUDA**: Si aparece un error relacionado con CUDA, verifica que el controlador NVIDIA (572.47) y PyTorch con soporte para CUDA 12.8 estén correctamente instalados.
- **Modelo no encontrado**: Asegúrate de que el modelo lora_model_3Instruct_bnb_new esté disponible o especifica la ruta correcta en el código.
- **Problemas de CORS**: Si la API no responde a solicitudes desde Moodle, verifica que la URL (http://192.168.67.76) esté correctamente configurada en allow_origins.

API BD guardado de mensajes

Base de Datos - Descripción General

La base de datos tutor_db está diseñada para almacenar información sobre usuarios registrados y los mensajes intercambiados entre ellos y el tutor virtual. La base de datos permite registrar mensajes de entrada (del usuario) y salida (del tutor), así como datos de los usuarios para futuras estadísticas. Utiliza el motor de almacenamiento **InnoDB** de MySQL/MariaDB, lo que garantiza soporte para claves foráneas y transacciones.

- **Nombre de la Base de Datos**: tutor_db
- **Codificación**: latin1 con colación latin1_swedish_ci
- **Versión del Servidor**: MariaDB 10.4.28 (compatible con MySQL 8.0.42)
- **Propósito**: Almacenar datos de usuarios y mensajes para un sistema de tutoría, con índices optimizados para consultas frecuentes y soporte para análisis estadístico.

Requisitos del Sistema

- **Servidor de Base de Datos**: MySQL 5.5.5-10.4.28-MariaDB o superior.
- **Sistema Operativo**: Windows 64 bits (según el script, ejecutado en Win64).
- **Espacio en Disco**: Depende del volumen de mensajes y usuarios, pero inicialmente menos de 100 MB para estructuras básicas.

- **Permisos**: El usuario de MySQL debe tener privilegios para crear bases de datos, tablas, índices y claves foráneas. Además, se debe crear un usuario específico (tutoruser) para gestionar la base de datos.

Configuración del Usuario de la Base de Datos

Para gestionar la base de datos tutor_db, se debe crear un usuario específico con los permisos adecuados. El usuario designado es tutoruser con la contraseña tutorpassword.

Pasos para Crear el Usuario:

Conéctate al servidor MySQL con un usuario con privilegios administrativos (por ejemplo, root):

```
mysql -u root -p
```

Ejecutar los siguientes comandos SQL para crear el usuario y otorgarle permisos:

```
-- Crear el usuario tutoruser con la contraseña tutorpassword
CREATE USER 'tutoruser'@'localhost' IDENTIFIED BY 'tutorpassword';

-- Otorgar todos los privilegios sobre la base de datos tutor_db
GRANT ALL PRIVILEGES ON tutor_db.* TO 'tutoruser'@'localhost';

-- Aplicar los cambios
FLUSH PRIVILEGES;
```

Notas sobre el Usuario:

- El usuario tutoruser tiene permisos completos sobre tutor_db (SELECT, INSERT, UPDATE, DELETE, CREATE, etc.), lo que lo hace adecuado para manejar todas las operaciones de la base de datos desde la API.
- La conexión está limitada a localhost, asumiendo que la API y el servidor MySQL se ejecutan en la misma máquina. Si la API se ejecuta en otra máquina, reemplaza 'localhost' por '%' o la IP correspondiente (por ejemplo, '192.168.67.76').
- Por seguridad, considera usar una contraseña más fuerte en un entorno de producción y limitar los privilegios según las necesidades específicas de la API.

Estructura de la Base de Datos

La base de datos tutor_db contiene dos tablas principales: users y chatmessages. A continuación, se detalla cada una.

Tabla users

Descripción: Almacena información sobre los usuarios registrados en el sistema, incluyendo su identificador, nombre de usuario, rol, nombre completo y fecha de creación.

Estructura:

Campo	Tipo de Dato	Descripción	Restricciones
user_id	INT(11)	Identificador único del usuario.	PRIMARY KEY, NOT NULL
username	VARCHAR(255)	Nombre de usuario único.	NOT NULL, UNIQUE KEY
role	VARCHAR(50)	Rol del usuario (por ejemplo, "estudiante", "tutor", "admin").	NOT NULL
userfullname	VARCHAR(255)	Nombre completo del usuario (opcional).	NULL permitido
created_at	TIMESTAMP	Fecha y hora de creación del usuario.	NOT NULL, DEFAULT current_timestamp()

Índices:

- PRIMARY KEY (user_id): Garantiza unicidad y optimiza búsquedas por user_id.
- UNIQUE KEY (username): Evita nombres de usuario duplicados.
- INDEX idx_users_username (username): Mejora el rendimiento de consultas que buscan por nombre de usuario.
- INDEX idx_users_userfullname (userfullname): Facilita búsquedas por nombre completo (si se usa).

Propósito: Esta tabla sirve como registro central de usuarios, permitiendo identificar quién envía mensajes y analizar estadísticas basadas en roles o fechas de registro.

Tabla chatmessages

Descripción: Almacena los mensajes intercambiados entre usuarios y el tutor, incluyendo el tipo de mensaje (entrada o salida), el texto, y la fecha de envío.

Estructura:

Campo	Tipo de Dato	Descripción	Restricciones
message_id	INT(11)	Identificador único del mensaje.	PRIMARY KEY, NOT NULL, AUTO_INCREMENT
user_id	INT(11)	Identificador del usuario asociado al mensaje.	NOT NULL, FOREIGN KEY a users(user_id)
message_type	ENUM('input','output')	Tipo de mensaje: "input" (enviado por el usuario) o "output" (respuesta del tutor).	NOT NULL
message_text	TEXT	Contenido del mensaje.	NOT NULL
sent_at	TIMESTAMP	Fecha y hora en que se envió el mensaje.	NOT NULL, DEFAULT current_timestamp()

Índices:

- PRIMARY KEY (message_id): Garantiza unicidad y optimiza búsquedas por message_id.
- INDEX idx_sent_at (sent_at): Mejora el rendimiento de consultas ordenadas por fecha de envío (útil para estadísticas temporales).
- INDEX idx_user_id (user_id): Optimiza consultas que filtran mensajes por usuario.
- FOREIGN KEY chatmessages_ibfk_1 (user_id) REFERENCES users(user_id): Garantiza que cada mensaje esté asociado a un usuario válido en la tabla users.

Propósito: Esta tabla registra la interacción entre usuarios y el tutor, permitiendo rastrear conversaciones completas y analizar patrones de uso o métricas de interacción.

Consideraciones

- **Codificación**: La base de datos usa latin1 por defecto, pero las tablas se crean con utf8mb4 durante la ejecución del script. Si planeas almacenar caracteres no latinos (por ejemplo, emojis o texto en otros idiomas), considera cambiar la codificación de la base de datos a utf8mb4 en el script:

```
CREATE DATABASE IF NOT EXISTS `tutor_db` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

- **Integridad Referencial**: La clave foránea en chatmessages asegura que los mensajes estén vinculados a usuarios válidos, evitando datos huérfanos.
- **Índices**: Los índices en username, userfullname, user_id, y sent_at están optimizados para consultas comunes en sistemas de tutoría (búsquedas por usuario o por fecha).
- **Escalabilidad**: La estructura es adecuada para aplicaciones pequeñas a medianas. Para grandes volúmenes de mensajes, considera particionamiento por fecha en chatmessages o un motor de búsqueda como Elasticsearch para análisis.
- **Seguridad del Usuario**: El usuario tutoruser tiene permisos completos sobre tutor_db. En un entorno de producción, evalúa limitar los privilegios (por ejemplo, solo SELECT, INSERT, UPDATE) según las necesidades de la API.

Solución de Problemas

- **Error de Clave Foránea**: Si la tabla users no existe antes de crear chatmessages, el script fallará debido a la restricción de clave foránea. Asegúrate de que las tablas se creen en el orden correcto (users primero).
- **Problemas de Codificación**: Si encuentras errores con caracteres especiales, verifica que la conexión del cliente MySQL use utf8mb4:

```
SET NAMES utf8mb4;
```

- **Permisos Insuficientes**: Si el usuario tutoruser no puede acceder a tutor_db, verifica que los privilegios se otorgaron correctamente:

```
SHOW GRANTS FOR 'tutoruser'@'localhost';
```

- **Error de Conexión**: Asegúrate de que el servidor MySQL esté en ejecución y que las credenciales de tutoruser sean correctas.

Notas Finales

La base de datos tutor_db está diseñada para integrarse con una API que maneje el guardado de mensajes y la gestión de usuarios. La estructura es simple pero efectiva para registrar interacciones y permite generar estadísticas basadas en los datos de users y chatmessages. El usuario tutoruser con la contraseña tutorpassword debe ser utilizado para todas las operaciones de la base de datos desde la API, garantizando un acceso controlado y seguro.

API BD Tutor - Descripción General

La API está diseñada para interactuar con la base de datos tutor_db, permitiendo registrar usuarios, guardar mensajes de entrada y salida (interacciones entre usuarios y el tutor), y obtener estadísticas sobre las interacciones. Está construida con **FastAPI** y utiliza **SQLAlchemy** para la gestión de la base de datos MySQL/MariaDB. La API soporta operaciones CRUD básicas para usuarios y mensajes, así como consultas analíticas para estadísticas.

- **Base de Datos**: tutor_db (MySQL/MariaDB, con tablas users y chatmessages).
- **Propósito**: Gestionar usuarios registrados, almacenar mensajes de interacción con el tutor, y proporcionar estadísticas como el usuario más activo, la hora pico de mensajes, y la pregunta más frecuente.
- **Tecnologías**:
    - **FastAPI** (0.110.0): Framework para la API web.
    - **SQLAlchemy** (2.0.29): ORM para interactuar con la base de datos.
    - **PyMySQL** (1.1.0): Driver para conectar con MySQL/MariaDB.
    - **Uvicorn** (0.29.0): Servidor ASGI para ejecutar la API.
- **Usuario de Base de Datos**: tutoruser con contraseña tutorpassword, configurado para gestionar tutor_db.

Requisitos del Sistema

- **Sistema Operativo**: Windows 64 bits (compatible con Linux/macOS con ajustes mínimos).
- **Servidor de Base de Datos**: MySQL 5.5.5-10.4.28-MariaDB o superior.
- **Python**: Versión 3.11.11
- **Espacio en Disco**: Aproximadamente 1-2 GB para el entorno virtual y dependencias.
- **Conexión a Internet**: Necesaria para instalar las dependencias.

Configuración del Entorno Virtual

Se utiliza un entorno virtual para aislar las dependencias de la API. Crearlo de manera similar al entorno del tutor.

```
conda create -n bd_api_env python=3.11
```

Activa el entorno virtual:

```
conda activate bd_api_env
```

Instalar Dependencias

Las dependencias están especificadas en el archivo requirements.txt:

```
fastapi==0.110.0
uvicorn==0.29.0
sqlalchemy==2.0.29
pymysql==1.1.0
```

Con el entorno virtual activado, instala las dependencias:

```
pip install -r requirements.txt
```

Notas:

- Asegúrate de que pip esté vinculado al entorno virtual (puedes verificar con pip --version).
- Si encuentras problemas con pymysql, verifica que el servidor MySQL/MariaDB esté instalado y en ejecución.

Estructura del Proyecto

La API está organizada en los siguientes archivos:

- schemas.py: Define los esquemas Pydantic para validar datos de entrada y salida (creación de usuarios, mensajes, y estadísticas).
- database.py: Configura la conexión a la base de datos con SQLAlchemy y proporciona una dependencia para sesiones de base de datos.
- models.py: Define los modelos SQLAlchemy para las tablas Users y ChatMessages.
- main.py: Contiene la lógica de la API, incluyendo los endpoints y la configuración de FastAPI.
- requirements.txt: Lista las dependencias necesarias.

Detalles de los Archivos

schemas.py

Define los modelos Pydantic para validar las solicitudes y respuestas:

- UserCreate: Esquema para registrar un usuario (user_id, username, role).
- MessageCreate: Esquema para guardar un mensaje (user_id, message_type, message_text).

- StatisticsResponse: Esquema para las estadísticas, incluyendo el usuario más activo, hora pico, y pregunta más frecuente.

database.py

Configura la conexión a MySQL/MariaDB usando SQLAlchemy:

- **DATABASE_URL**: mysql+pymysql://tutoruser:tutorpassword@localhost/tutor_db.
- Proporciona una función get_db para gestionar sesiones de base de datos.

models.py

Define los modelos SQLAlchemy que mapean las tablas Users y ChatMessages:

- User: Representa la tabla users con campos user_id, username, role, userfullname, y created_at.
- Message: Representa la tabla chatmessages con campos message_id, user_id, message_type, message_text, y sent_at.

main.py

Contiene la aplicación FastAPI con los endpoints y la lógica principal:

- Configura CORS para permitir solicitudes desde múltiples orígenes (por ejemplo, http://192.168.67.76, http://localhost).
- Crea las tablas en la base de datos al iniciar la API usando models.Base.metadata.create_all.

Endpoints de la API

La API proporciona los siguientes endpoints:

Registrar un Usuario

- **Método**: POST
- **Ruta**: /api/users/register
- **Descripción**: Registra un nuevo usuario en la base de datos.
- **Cuerpo de la Solicitud**:

```
{
    "user_id": 1,
    "username": "juanperez",
    "role": "estudiante",
    "userfullname": "Juan Pérez"
}
```

Respuesta Exitosa (200):

```
{
   "user_id": 1,
   "username": "juanperez",
   "role": "estudiante",
   "userfullname": "Juan Pérez",
   "created_at": "2025-05-28T20:31:00"
}
```


Errores:

- 200 (si el usuario ya existe): {"message": "User already exists"}

Obtener Todos los Usuarios

- **Método**: GET
- **Ruta**: /api/users
- **Descripción**: Devuelve una lista de todos los usuarios registrados.
- **Respuesta Exitosa** (200):


```
[
   {
      "user_id": 1,
      "username": "juanperez",
      "role": "estudiante",
      "userfullname": "Juan Pérez",
      "created_at": "2025-05-28T20:31:00"
   },
   ...
]
```


Obtener un Usuario por ID

- **Método**: GET
- **Ruta**: /api/users/{user_id}
- **Descripción**: Devuelve los detalles de un usuario específico.
- **Parámetros**: user_id (entero, en la URL).
- **Respuesta Exitosa** (200):


```
{
   "user_id": 1,
   "username": "juanperez",
   "role": "estudiante",
   "userfullname": "Juan Pérez",
   "created_at": "2025-05-28T20:31:00"
}
```


Errores:

- 404: {"detail": "User not found"}

Guardar un Mensaje

- **Método**: POST
- **Ruta**: /api/messages/save
- **Descripción**: Guarda un mensaje (entrada o salida) en la base de datos.
- **Cuerpo de la Solicitud**:

```
{
   "user_id": 1,
   "message_type": "input",
   "message_text": "Hola, ¿cómo funciona la IA?"
}
```

Respuesta Exitosa (200):

```
{
   "message": "Message saved successfully",
   "message_id": 1
}
```

Obtener Estadísticas

- **Método**: GET
- **Ruta**: /api/statistics
- **Descripción**: Devuelve estadísticas sobre los mensajes, incluyendo:
    - Usuario más activo (top_user).
    - Total de mensajes (total_messages).
    - Hora pico de mensajes (peak_hour).
    - Total de mensajes en la hora pico (total_peak_hour_messages).
    - Pregunta más frecuente (top_question).
    - Total de preguntas (total_questions).
- **Respuesta Exitosa** (200):

```
{
   "top_user": {
      "userfullname": "Juan Pérez",
      "total_messages": 10
   },
   "total_messages": 50,
   "peak_hour": {
      "hour_of_day": 14,
      "message_count": 15
   },
   "total_peak_hour_messages": 50,
   "top_question": {
      "message_text": "Hola, ¿cómo funciona la IA?",
      "frequency": 5
   },
   "total_questions": 30
```

```
}
```

Ejecución de la API

Ejecuta

```
uvicorn main:app --reload --host 0.0.0.0 --port 8080
```

Configuración de CORS

La API está configurada para aceptar solicitudes desde los siguientes orígenes, lo que permite integraciones con plataformas como Moodle:

- http://192.168.67.76, http://192.168.67.76:80, http://192.168.67.76:8080, http://192.168.67.76:3000
- http://127.0.0.1, http://127.0.0.1:80, http://127.0.0.1:8080
- http://localhost, http://localhost:80, http://localhost:3000, http://localhost:8080

Notas:

- Todos los métodos HTTP (GET, POST, etc.) y encabezados están permitidos.
- Las credenciales (por ejemplo, cookies) están habilitadas para solicitudes autenticadas.

Integración con la Base de Datos

- **Conexión**: La API se conecta a tutor_db usando el usuario tutoruser y la contraseña tutorpassword (definido en database.py).
- **Tablas**:
    - Users: Mapeada al modelo User en models.py.
    - ChatMessages: Mapeada al modelo Message en models.py.
- **Operaciones**: La API realiza operaciones CRUD (crear, leer) en las tablas y consultas analíticas para estadísticas.
- **Sincronización**: Las tablas se crean automáticamente al iniciar la API (models.Base.metadata.create_all).

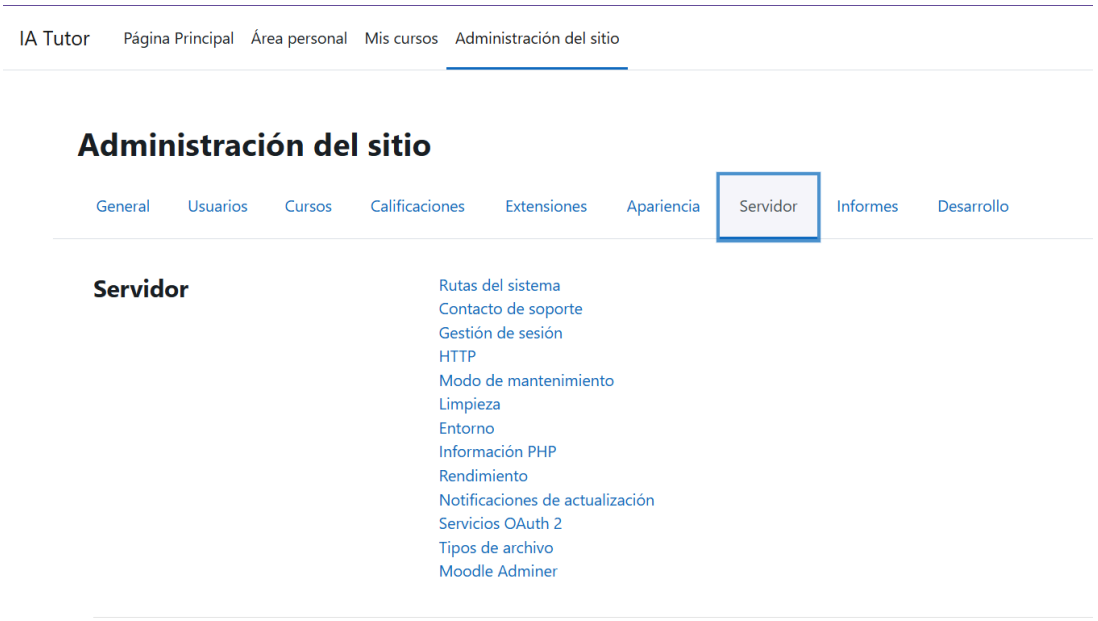
Notas Finales

La API proporciona una interfaz robusta para gestionar usuarios y mensajes en tutor_db, con soporte para estadísticas analíticas. Está optimizada para integrarse con aplicaciones cliente (como Moodle) y puede extenderse para incluir más funcionalidades, como autenticación de usuarios o endpoints adicionales para análisis. El usuario tutoruser con contraseña tutorpassword garantiza un acceso seguro a la base de datos, pero considera

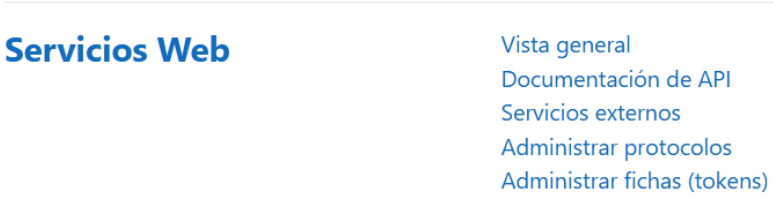
reforzar la seguridad en producción (por ejemplo, usando variables de entorno para credenciales).

Activación de Web Services Moodle

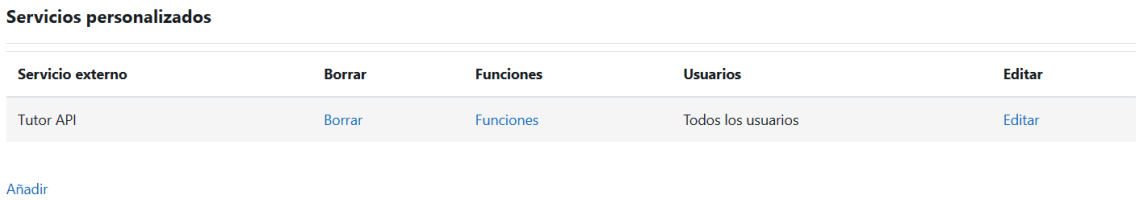
Como administrador del sitio navegar a la sección Administración del sitio, luego a la parte del Servidor.



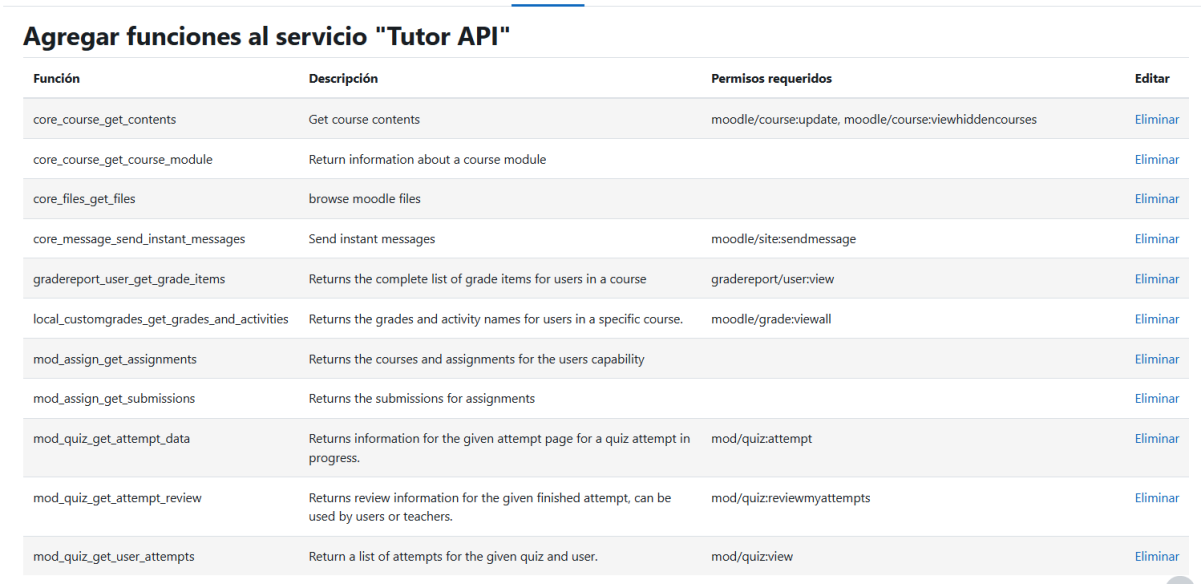
En esta sección navegar al apartado Servicios Web e ingresamos a Servicios Externos



En la sección Servicios Externos se añade un nuevo Servicio Personalizado



Y se añaden las siguientes funciones:



Agregar funciones al servicio "Tutor API"

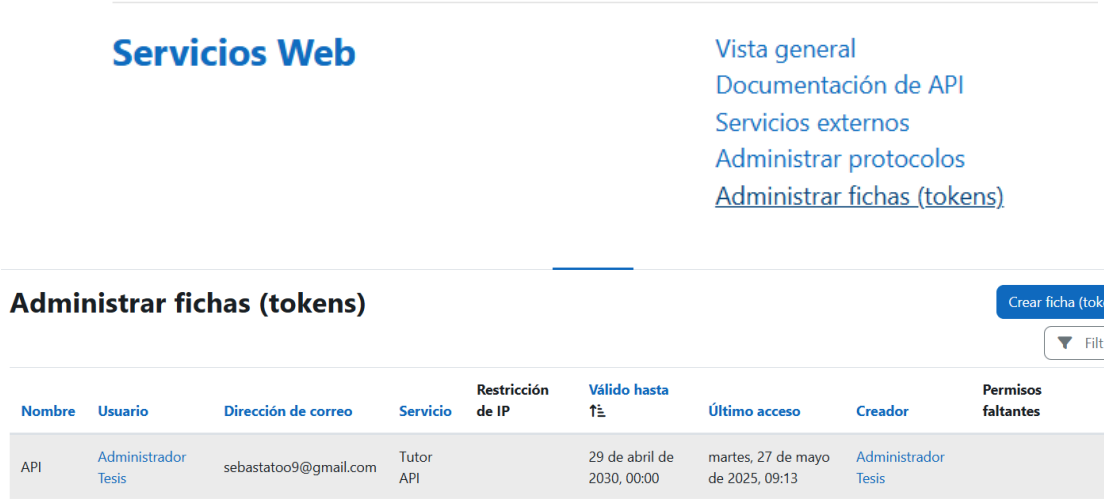
Función	Descripción	Permisos requeridos	Editar
core_course_get_contents	Get course contents	moodle/course:update, moodle/course:viewhiddencourses	Eliminar
core_course_get_course_module	Return information about a course module		Eliminar
core_files_get_files	browse moodle files		Eliminar
core_message_send_instant_messages	Send instant messages	moodle/site:sendmessage	Eliminar
gradereport_user_get_grade_items	Returns the complete list of grade items for users in a course	gradereport/user:view	Eliminar
local_customgrades_get_grades_and_activities	Returns the grades and activity names for users in a specific course.	moodle/grade:viewall	Eliminar
mod_assign_get_assignments	Returns the courses and assignments for the users capability		Eliminar
mod_assign_get_submissions	Returns the submissions for assignments		Eliminar
mod_quiz_get_attempt_data	Returns information for the given attempt page for a quiz attempt in progress.	mod/quiz:attempt	Eliminar
mod_quiz_get_attempt_review	Returns review information for the given finished attempt, can be used by users or teachers.	mod/quiz:reviewmyattempts	Eliminar
mod_quiz_get_user_attempts	Return a list of attempts for the given quiz and user.	mod/quiz:view	Eliminar

Lista y Descripción de Funciones del Servicio "Tutor API"

Función	Descripción	Permisos Requeridos
core_course_get_contents	Obtiene los contenidos de un curso. Retorna información detallada sobre las secciones y actividades de un curso específico en Moodle.	moodle/course:update, moodle/course:viewhiddencourses
core_course_get_course_module	Devuelve información sobre un módulo de curso específico. Permite acceder a detalles como el tipo de módulo (por ejemplo, foro, tarea) y su configuración.	-
core_files_get_files	Permite explorar y recuperar archivos almacenados en Moodle, como recursos subidos por usuarios o profesores.	-
core_message_send_instant_messages	Envía mensajes instantáneos a usuarios en Moodle. Útil para notificaciones o interacciones directas entre el tutor y los estudiantes.	moodle/site:sendmessage

graderreport_user_get_grade_items	Retorna la lista completa de ítems de calificación para los usuarios en un curso, incluyendo notas y actividades evaluadas.	graderreport/user:view
local_customgrades_get_grades_and_activities	Devuelve los grados y los nombres de las actividades para los usuarios en un curso específico.	moodle/grade:view
mod_assign_get_assignments	Retorna los cursos y asignaciones para las que los usuarios tienen capacidad, mostrando detalles como fechas límite y descripciones.	-
mod_assign_get_submissions	Recupera las entregas realizadas por los estudiantes para las asignaciones, incluyendo archivos o texto enviado.	-
mod_quiz_get_attempt_data	Devuelve información para una página específica de un intento de un cuestionario en progreso, útil para seguimiento en tiempo real.	mod/quiz:attempt
mod_quiz_get_attempt_review	Retorna información de revisión para un intento terminado de un cuestionario, accesible por usuarios o profesores para análisis.	mod/quiz:reviewmyattempts
mod_quiz_get_user_attempts	Devuelve una lista de intentos realizados por un usuario específico en un cuestionario dado.	mod/quiz:view

Una vez hecho esto y para poder usar los endpoints desde otro lugar, se debe crear un token con permisos del administrador del curso.



Una vez hecho esto se puede usar los endpoints y el token para recuperar la información del curso.

Configuración de los bloques

Para instalar los bloques en Moodle, sigue estos pasos de manera clara y ordenada:

Copiar las carpetas de los bloques:

- Dirígete a la ruta C:\moodle\server\moodle\blocks en tu instalación de Moodle.
- Copia las carpetas de los bloques que deseas instalar: mi_api_block, mi_actividad_feedback_block y tutor_statistics.
- **mi_api_block**: Permite la interacción con una API personalizada, gestionando mensajes y registros de usuarios. Incluye las variables API_BD_TUTOR_BASE y API_tutor al inicio para las URLs (http://localhost:8080/api/ y http://localhost:8000/generar).
- **mi_actividad_feedback_block**: Proporciona retroalimentación educativa para actividades como cuestionarios y tareas, analizando calificaciones y generando recomendaciones. Incluye las variables API_tutor y API_Moodle al inicio para las URLs (http://localhost:8000/generar y http://localhost/webservice/rest/server.php).
- **tutor_statistics**: Muestra estadísticas de uso (usuario más activo, hora pico, pregunta frecuente) con gráficos y una conclusión generada por IA. Incluye las variables API_BD_TUTOR_BASE y API_tutor al inicio para las URLs (http://localhost:8080/api/ y http://localhost:8000/generar).

Cada bloque tiene sus URLs definidas como constantes al inicio para facilitar la mantenibilidad.

Reconstruir los archivos JavaScript para los bloques:

- Navega a la carpeta raíz de tu instalación de Moodle: C:\moodle\server\moodle.
- Abre una terminal en esta ubicación (puedes usar el símbolo del sistema o una terminal integrada si usas un editor como VS Code).

- Ejecuta el siguiente comando para compilar los archivos JavaScript de los bloques usando Grunt:

```
grunt amd --force
```

Este comando procesa los archivos JavaScript de los bloques (como los módulos AMD definidos en los códigos) y los prepara para su uso en Moodle. La opción --force asegura que el proceso se complete incluso si hay advertencias.

Finalizar la instalación:

- Una vez que el comando grunt amd --force se ejecute correctamente, accede a tu sitio de Moodle como administrador.
- Moodle detectará automáticamente los nuevos bloques. Para activarlos, ve a **Administración del sitio > Plugins > Actualizar base de datos** (si es necesario) y luego configúralos desde **Administración del sitio > Plugins > Bloques**.

Notas Adicionales

- Asegúrate de tener **Node.js** y **Grunt** instalados en tu sistema, ya que son necesarios para ejecutar el comando grunt. Si no están instalados, puedes instalarlos siguiendo la documentación oficial de Moodle para desarrolladores.
- Verifica que las carpetas de los bloques tengan los permisos correctos para que Moodle pueda acceder a ellas.