Documentación MADRE:

Texto

Descripción generada automáticamente

**2. Descripción de cada directorio:**

* **main/**: Contiene todo el código fuente de la aplicación principal.
  + **config/**: Aquí van todas las configuraciones del proyecto, como los settings, la configuración de logs, etc.
  + **api/**: Aquí se organizan las rutas y vistas de la API.
    - **routes/**: Módulos que definen las rutas/endpoints de la API.
    - **views/**: Módulos que gestionan la lógica de cada endpoint.
  + **services/**: Servicios que incluyen la lógica de negocio, y manejo de bases de datos.
  + **models/**: Definición de los modelos que interactúan con las bases de datos MySQL y MongoDB.
  + **tests/**: Todos los tests de la aplicación, organizados por funcionalidad.
  + **utils/**: Funciones de utilidad que pueden ser usadas en cualquier parte de la aplicación.
* **docs/**: Documentación relacionada con el proyecto. Esto incluye la documentación técnica, API, y arquitectura.
* **scripts/**: Scripts útiles para tareas administrativas, como la migración o inicialización de bases de datos.
* **.env**: Archivo para las variables de entorno sensibles.
* **requirements.txt**: Archivo que contiene las dependencias del proyecto.
* **README.md**: Documento que describe el proyecto, cómo instalarlo y cómo usarlo.

**3. Sugerencias adicionales:**

1. **Frameworks y Librerías**:
   * Si decides utilizar **Django**, algunas de estas estructuras cambiarán debido a la naturaleza del framework, ya que Django tiene una estructura propia que incluye apps y un modelo más definido.
   * **Flask** podría ser una buena alternativa si prefieres un enfoque más modular y menos estructurado que Django.
2. **BBDD**:
   * Considera usar un ORM (Object-Relational Mapping) como SQLAlchemy si decides no usar Django, para facilitar la interacción con MySQL.
3. **Manejo de la API**:
   * Puedes considerar el uso de **FastAPI** si deseas un framework moderno, rápido, y con la posibilidad de generar documentación de la API de forma automática.

**Sugerencias adicionales para FastAPI:**

* **CORS**: No olvides configurar los CORS si tus frontales van a hacer llamadas a la API desde dominios diferentes.
* **Documentación automática**: FastAPI genera automáticamente la documentación Swagger y Redoc. Asegúrate de que tus modelos Pydantic y anotaciones de tipo estén bien definidos para aprovechar esta funcionalidad.
* **Middleware**: Considera agregar middleware para logging, manejo de excepciones globales, y autenticación.

**1. Middleware para Logging**

Este middleware se encargará de registrar cada petición y respuesta. Puedes usar el middleware nativo de FastAPI o crear uno personalizado.

**2. Middleware para Manejo de Excepciones Globales**

El manejo de excepciones globales en FastAPI es generalmente manejado con un ExceptionHandler. Sin embargo, puedes usar un middleware si necesitas un control más fino.

**3. Middleware para Autenticación**

Para la autenticación, puedes usar OAuth2, JWT, o cualquier otro método de autenticación. Este middleware se encargará de verificar la autenticación en cada petición.

**4. Registro de Middleware en FastAPI**

Después de implementar los middleware, debes registrarlos en tu aplicación FastAPI en main.py:

**5. Configuración Adicional**

1. **Logging**:
   * Configura el logging en app/config/logging\_config.py:
2. **Autenticación Avanzada:**
   * Puedes integrar OAuth2, JWT, o cualquier otro esquema de autenticación utilizando dependencias de FastAPI en lugar de un middleware si necesitas más flexibilidad.

Si tus frontales van a realizar llamadas a la API desde diferentes dominios o subdominios, es esencial configurar **CORS (Cross-Origin Resource Sharing)** en tu aplicación FastAPI. CORS permite que los navegadores controlen qué recursos pueden ser solicitados de un dominio diferente al de la página que está haciendo la solicitud.

**Cómo Configurar CORS en FastAPI**

FastAPI tiene soporte nativo para CORS a través de CORSMiddleware, lo que facilita mucho su configuración.

**Paso 1: Instalar starlette**

FastAPI ya depende de starlette, que incluye CORSMiddleware. Así que normalmente no necesitas instalar nada adicional. Pero, si por alguna razón no lo tienes, puedes instalarlo con:

**Paso 2: Configurar CORSMiddleware**

Añade el siguiente código en tu main.py o en el archivo principal de tu aplicación FastAPI:

**Explicación de los Parámetros:**

* **allow\_origins**: Una lista de dominios o subdominios permitidos para hacer solicitudes a tu API. Puedes usar comodines (\*) para permitir todos los subdominios de un dominio específico.
* **allow\_credentials**: Si necesitas permitir el envío de cookies o cabeceras de autorización con las solicitudes (por ejemplo, para JWT o sesiones), ponlo en True.
* **allow\_methods**: Especifica los métodos HTTP permitidos (GET, POST, PUT, etc.). Puedes usar ["\*"] para permitir todos los métodos.
* **allow\_headers**: Especifica qué cabeceras HTTP pueden ser enviadas por el frontend. Puedes usar ["\*"] para permitir todas las cabeceras.

**Paso 3: Probar la Configuración**

Una vez configurado, deberías probar desde tus frontales para asegurarte de que las solicitudes a tu API funcionan correctamente y que no hay problemas relacionados con CORS.

**Consideraciones de Seguridad**

* **Restricción de Orígenes**: Intenta ser lo más restrictivo posible con los orígenes permitidos en allow\_origins para evitar posibles riesgos de seguridad.
* **Autenticación y Cookies**: Si estás usando autenticación basada en cookies o tokens, asegúrate de que allow\_credentials=True esté configurado y que tus dominios estén correctos.

Configurar CORS es un paso esencial cuando tus frontales y tu backend se comunican a través de diferentes dominios. La configuración anterior debería cubrir la mayoría de los escenarios comunes.

# Uvicorn

1. **if \_\_name\_\_ == '\_\_main\_\_':**:
   * **¿Qué es?**: Este es un bloque condicional en Python que se ejecuta solo si el archivo Python es ejecutado directamente, es decir, no se está importando como un módulo en otro archivo.
   * **¿Por qué es útil?**: Permite que el código dentro de este bloque solo se ejecute cuando el script es ejecutado directamente, no cuando es importado. Esto es útil para definir comportamientos que solo deben ocurrir en ciertos contextos, como iniciar un servidor web.
2. **import uvicorn**:
   * **¿Qué es?**: uvicorn es un servidor ASGI (Asynchronous Server Gateway Interface) que se usa comúnmente para ejecutar aplicaciones web construidas con FastAPI.
   * **¿Por qué es útil?**: uvicorn es rápido y está optimizado para aplicaciones asincrónicas como las construidas con FastAPI. Este import se realiza aquí porque uvicorn se usará para iniciar el servidor que ejecutará la aplicación FastAPI.
3. **uvicorn.run(app, host="0.0.0.0", port=8000, log\_level=settings.LOG\_LEVEL)**:
   * **uvicorn.run(...)**: Este es el comando que inicia el servidor uvicorn y pone en marcha tu aplicación FastAPI.
   * **Parámetros**:
     + **app**: Aquí, app es la instancia de tu aplicación FastAPI. Es el objeto que define todas tus rutas y lógica de la API.
     + **host="0.0.0.0"**: Este parámetro indica en qué direcciones IP el servidor debe estar escuchando. 0.0.0.0 significa que la aplicación estará disponible en todas las direcciones IP de la máquina, es decir, será accesible desde cualquier red que pueda conectarse a la máquina donde se ejecuta.
     + **port=8000**: Este es el puerto en el que el servidor estará escuchando. 8000 es el puerto predeterminado, pero puede ser cambiado según tus necesidades.
     + **log\_level=settings.LOG\_LEVEL**: Este parámetro define el nivel de detalle en los logs que uvicorn generará. El valor settings.LOG\_LEVEL probablemente venga de un archivo de configuración en tu proyecto y puede ser algo como "info", "debug", "warning", etc.

**Resumen**

Este bloque de código se usa para iniciar la aplicación FastAPI utilizando uvicorn como servidor cuando ejecutas directamente el script Python. Es un método estándar para lanzar aplicaciones web con FastAPI y asegurar que tu servidor web esté escuchando en el puerto y dirección IP correctos, con la configuración de logging adecuada.

Si quieres ejecutar la aplicación FastAPI para desarrollo o despliegue, este código es esencial, ya que pone en marcha el servidor que responderá a las solicitudes HTTP.

Documentación

**Docstring para el Método call\_proc\_bbdd\_records**

|  |  |
| --- | --- |
| **Texto  Descripción generada automáticamente** | **Explicación del Docstring**   1. **Descripción General**: Explica qué hace el método de manera clara y concisa. 2. **Parámetros**: Describe cada parámetro, incluyendo su tipo y propósito. 3. **Retorno**: Indica qué tipo de valor devuelve el método y describe brevemente lo que contiene. 4. **Excepciones**: Menciona las excepciones que el método podría lanzar, con una breve explicación. 5. **Ejemplo de Uso**: Proporciona un ejemplo simple de cómo usar el método. 6. **Notas**: Incluir notas adicionales puede ser útil para describir comportamientos particulares o consideraciones importantes.   **Beneficios de Documentar Así**   * **Claridad**: Ayuda a cualquier desarrollador (incluido tú mismo en el futuro) a entender rápidamente qué hace el método y cómo usarlo. * **Automatización**: Si usas una herramienta como Sphinx, estos docstrings pueden ser utilizados para generar documentación automática de tu código. * **Mantenimiento**: Con una buena documentación, es más fácil mantener y modificar el código sin introducir errores. |