Documentación de Uso - AWS NODEJS

Introducción

Este documento proporciona una guía de uso para el proyecto XYZ. El proyecto XYZ es una aplicación web que permite gestionar y administrar personajes de una historia.

Requisitos previos

Antes de comenzar a utilizar el proyecto **aws-nodeJS**, asegúrate de cumplir con los

siguientes requisitos:
Tener instalado Node.js en tu sistema.
Tener acceso a una base de datos MySQL para almacenar los datos.
cli de aws
instalar serverless de manera global

Instalación
Sigue los pasos a continuación para instalar y configurar el proyecto **aws-nodeJS**

en tu **entorno local:**

Clona el repositorio del proyecto desde GitHub:
https://github.com/eduvinTrigos/prueba_serverless_softe.git
Accede al directorio del proyecto: cd **prueba_serverless_softe**

Ejecuta el comando npm install para instalar las dependencias del proyecto.

Configura la conexión a la base de datos editando el archivo **config/db.js** y proporcionando los detalles de conexión adecuados.

comandos para iniciar el proyecto
sls offline --reloadHandler =manera offline

para desplegar en aws no se olvide de configurar su cli de aws con sus credenciales y la zona donde se desplegará el servicio

sls deploy --stage demo
sls deploy --stage development

Una vez que hayas completado estos pasos, el proyecto **aws-nodeJS** estará listo para su uso.

Uso
El proyecto **aws-nodeJS** ofrece una API RESTful para gestionar los personajes de la historia. A continuación, se detallan las rutas disponibles y los métodos correspondientes:

Obtener todos los personajes
Método: GET
Ruta: /personajes

- **Descripción:** Obtiene todos los personajes almacenados en la base de datos.
- **Parámetros de consulta:** Ninguno.
- **Respuesta exitosa:** Código de estado 200 (OK) y un array de objetos JSON que representan los personajes.
- **Respuesta de error**: Código de estado 500 (Error del servidor).
-
- **Obtener un personaje específico**
- **Método: GET**
- **Ruta: /personajes/{id}/details**
-
- **Descripción:** Obtiene los detalles de un personaje específico.
- **Parámetros de ruta:** {id}: El identificador único del personaje.
- **Respuesta exitosa:** Código de estado 200 (OK) y un objeto JSON que representa el personaje solicitad**o.**
- **Respuesta de error**: Código de estado 404 (No encontrado) si el personaje no existe, o código de estado 500 (Error del servidor).

Crear un nuevo personaje
Método: POST
Ruta: /personajes

- **Descripción:** Crea un nuevo personaje con los datos proporcionados.

- **Parámetros de cuerpo**: Un objeto JSON con los siguientes campos obligatorios:
- **per_name**: Nombre del personaje (cadena de texto).
- **per_last_name:** Apellido del personaje (cadena de texto).
- **per_age**: Edad del personaje (número entero).
- **per_gender**: Género del personaje (cadena de texto).
- **per_description:** Descripción del personaje (cadena de texto).
- **per_skills:** Habilidades del personaje (cadena de texto).
-
- **Respuesta exitosa:** Código de estado 201 (Creado) y un objeto JSON que representa **el nuevo personaje creado.**
- **Respuesta de error:** Código de estado 400 (Solicitud incorrecta) si faltan campos obligatorios en el cuerpo de la solicitud, o código de estado500 (Error del servidor).

Actualizar un personaje existente
Método: PUT
Ruta: /personajes/{id}

- **Descripción:** Actualiza los datos de un personaje existente.
- **Parámetros de ruta:** {id}: El identificador único del personaje.
- **Parámetros de cuerpo:** Un objeto JSON con los campos opcionales que se deseen actualizar:
- **per_name:** Nuevo nombre del personaje (cadena de texto).
- **per_last_name:** Nuevo apellido del personaje (cadena de texto).
- **per_age**: Nueva edad del personaje (número entero).
- **per_gender:** Nuevo género del personaje (cadena de texto).
- **per_description:** Nueva descripción del personaje (cadena de texto).
- **per_skills:** Nuevas habilidades del personaje (cadena de texto).
-
- **Respuesta exitosa:** Código de estado 200 (OK) y un objeto JSON que representa el personaje actualizado.
- **Respuesta de error:** Código de estado 400 (Solicitud incorrecta) si hay errores de validación en los datos proporcionados, código de estado 404 (No encontrado) si el personaje no existe, o código de estado 500 (Error del servidor).

Eliminar un personaje
Método: DELETE
Ruta: /personajes/{id}
Descripción: Elimina un personaje existente.

Parámetros de ruta: {id}: El identificador único del personaje.

Respuesta exitosa: Código de estado 200 (OK) y un objeto JSON que indica que el personaje ha sido eliminado correctamente.
Respuesta de error: Código de estado 404 (No encontrado) si el personaje no existe, o código de estado 500 (Error del servidor).

Peticiones a SWAPI (Star Wars API)
El proyecto **aws-nodeJS** también ofrece integración con la Star Wars API (SWAPI) para obtener información relacionada con los personajes, planetas, películas, especies, naves espaciales y vehículos de Star Wars. A continuación, se detallan las rutas disponibles y los métodos correspondientes para acceder a la SWAPI:

Obtener datos de SWAPI por tipo y ID
Método: GET
Ruta: /swapi/{type_data}/{id_data}
Descripción: Obtiene los datos de SWAPI correspondientes al tipo y ID especificados.
Parámetros de ruta:

- **{type_data}:** El tipo de dato deseado (people, planets, films, species, starships, vehicles).
- **{id_data}:** El ID del dato específico.
- **Parámetros de consulta:** Ninguno.
- **Respuesta exitosa:** Código de estado 200 (OK) y un objeto JSON que representa los datos de SWAPI solicitados.
- **Respuesta de error:** Código de estado 400 (Solicitud incorrecta) si el tipo de solicitud es desconocido, o código de estado 500 (Error del servidor)**.**

Ejemplo de uso:

GET /swapi/people/1
GET /swapi/planets?page=2

Obtener datos de SWAPI por tipo
Método: GET
Ruta: /swapi/{type_data}

- **Descripción:** Obtiene los datos de SWAPI correspondientes al tipo especificado.
- Parámetros de ruta:

- **{type_data}:** El tipo de dato deseado (people, planets, films, species, starships, vehicles).
- **Parámetros de consulta:**
- **page (opcional):** El número de página para paginar los resultados.
- **Respuesta exitosa:** Código de estado 200 (OK) y un objeto JSON que representa los datos de SWAPI solicitados.
- **Respuesta de error:** Código de estado 400 (Solicitud incorrecta) si el tipo de solicitud es desconocido, o código de estado 500 (Error del servidor).

Nota: Los datos de SWAPI son proporcionados por la API pública de Star Wars y pueden estar sujetos a cambios o disponibilidad limitada.

Troubleshooting (Solución de Problemas)

Si encuentras algún problema durante la instalación o el uso del proyecto **aws-nodeJS**, prueba las siguientes soluciones comunes:
Asegúrate de haber configurado correctamente la conexión a la base de datos en el archivo config/db.js.

Verifica que todos los paquetes y dependencias estén instalados correctamente utilizando el comando npm install.

Reinicia el servidor y asegúrate de que esté funcionando correctamente.
Si recibes errores de validación al realizar una solicitud, asegúrate de proporcionar los datos correctamente formateados y completos.