Lab 1815 - Creating the OpenAPI documentation

## Introduction to OpenAPI

OpenAPI is a specification that enables you to describe your API in a machine-readable format. It offers clear documentation and assists in development by generating client libraries and server stubs. OpenAPI adheres to RESTful principles and is typically used with JSON or YAML. The specification will be written in JSON for the purposes of this lab as this is required by watsonx Assistant. If you would prefer to write the spec in YAML, please ensure to convert it to JSON before importing it into the Assistant.

??? note Working with YAML specifications will be outside the scope of this lab.

## Installation

To start, ensure you have a text editor or integrated development environment (IDE) installed. You’ll also need a basic understanding of JSON. Visual Studio Code has already been installed on the virtual machine being used for this lab.

## Defining API Information

Begin by defining general information about your API, such as its title, version, and description. This metadata helps users comprehend the API’s purpose and usage.

{  
 "openapi": "3.0.0",  
 "info": {  
 "title": "My Awesome API",  
 "version": "1.0.0",  
 "description": "An API for demonstrating OpenAPI specifications."  
 }  
}

## Defining Paths and Operations

Define the paths (endpoints) of your API and the operations (HTTP methods) that can be performed on them. Include request and response details, parameters, and more.

{  
 "paths": {  
 "/users": {  
 "get": {  
 "summary": "Get a list of users",  
 "responses": {  
 "200": {  
 "description": "Successful response",  
 "content": {  
 "application/json": {  
 "example": {  
 "users": [  
 { "id": 1, "name": "John Doe" },  
 { "id": 2, "name": "Jane Smith" }  
 ]  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
}

## Handling Parameters

Specify parameters that can be passed in the URL, query string, or request body. Parameters help customize API requests.

{  
 "paths": {  
 "/users/{user\_id}": {  
 "parameters": [  
 {  
 "name": "user\_id",  
 "in": "path",  
 "required": true,  
 "schema": {  
 "type": "integer"  
 },  
 "description": "ID of the user"  
 }  
 ],  
 "get": {  
 "summary": "Get user by ID",  
 "responses": {  
 "200": {  
 "description": "Successful response",  
 "content": {  
 "application/json": {  
 "example": {  
 "id": 1,  
 "name": "John Doe"  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
}

## Adding Request Bodies

Describe request bodies for operations that require data to be sent in the request.

{  
 "paths": {  
 "/users": {  
 "post": {  
 "summary": "Create a new user",  
 "requestBody": {  
 "required": true,  
 "content": {  
 "application/json": {  
 "schema": {  
 "type": "object",  
 "properties": {  
 "name": { "type": "string" },  
 "email": { "type": "string" }  
 }  
 }  
 }  
 }  
 },  
 "responses": {  
 "201": {  
 "description": "User created successfully"  
 }  
 }  
 }  
 }  
 }  
}

## Adding Responses

Define different response scenarios with corresponding HTTP status codes and content.

{  
 "paths": {  
 "/users/{user\_id}": {  
 "get": {  
 "summary": "Get user by ID",  
 "responses": {  
 "200": {  
 "description": "Successful response",  
 "content": {  
 "application/json": {  
 "example": {  
 "id": 1,  
 "name": "John Doe"  
 }  
 }  
 }  
 },  
 "404": {  
 "description": "User not found"  
 }  
 }  
 }  
 }  
 }  
}

## Generating Documentation

With your OpenAPI specification ready, you can generate interactive documentation using tools like Swagger UI or ReDoc.

## Conclusion

Congratulations! You’ve learned the fundamentals of writing OpenAPI 3.0 specifications using JSON examples. For more advanced features and detailed documentation, refer to the official OpenAPI Specification.

Practice is key. Experiment with different features and create more complex API specifications to enhance your skills.