FastAPI Application Documentation (General)

Overview

This document outlines the structure and functionality of a FastAPI application designed for handling file operations, data processing, and API interactions which returns JSON data. This documentation will walk through the Python modules, the logging setup, initialization of the FastAPI application, data models, and endpoint configurations. Each component and endpoint will be explained in detail for ease of understanding and usage.

Endpoints:

The application defines the following API endpoints:

1. **/validationbody** - Validates files.
2. **/databasevaluesbody** - Processes and returns template-based data.
3. **/tablebody** - Returns table structure after processing file content.

Prerequisites:

To run this FastAPI application, ensure the following are set up on your machine:

1. Python: Version 3.6 or higher is required. You can download it from python.org.

Required Packages:

Install the necessary packages using pip

```
pip install fastapi pandas uvicorn
```

Steps to Run the Application:

1. **Save the Code**: Store the FastAPI code in a Python file, e.g., FastAPI_filename.py .

2. **Run the Service:** Open a terminal and navigate to the directory where FastAPI_ file_name.py is saved.

3. **Start the FastAPI Server:** Use the following command to launch the server:

```
uvicorn FastAPI_ file_name:app --reload
```

The --reload flag automatically reloads the server when you make changes to the code.

4. **Access the API:** Once the server is running, access the API by visiting http://127.0.0.1:8000 in a browser.

Code Overview

Modules and Imports:

The following key Python modules are used:

- **FastAPI:** A modern, fast web framework for building APIs.
- **Request:** Represents an HTTP request in FastAPI, providing access to various parts of the HTTP request.
- **HTTPException**: Allows for raising exceptions that the FastAPI app can catch to return standardized error responses.
- **typing (List, Optional)**: Provides runtime support for type hints.
- **os**: Provides a portable way of using operating system-dependent functionality like reading or writing to the filesystem.
- **tempfile**: Generates temporary files and directories.
- **datetime (as dt)**: Manipulates dates and times.
- **logging**: Logs events for a library or application.
- **uvicorn**: A lightning-fast ASGI server for Python.
- **pandas (as pd)**: Offers data structures and operations for manipulating numerical tables and time series.

Setup and Configuration:

Logging:

Logs are stored in the ./Logs directory, which is automatically created if it doesn’t exist. The log filename follows the format YYYY-MM-DD.log, and logging captures all info-level events, providing detailed timestamps and messages.

FastAPI App Initialization:

The FastAPI application is instantiated using:

app = FastAPI()

API Endpoints:

1. /validationbody Endpoint

Description: This endpoint is designed to handle the validation of incoming files by reading the file content, and storing it temporarily, the system validates the data and responds with a status indicating the result. This method accepts a binary file and if the file is valid, it returns a

JSON object with Code: 0. Otherwise, when it is invalid or an error occurs, it logs the error and returns a JSON object with Code: non-zero and the error message.

Request:

- Method: POST
- Body: data as a post body

Validation Response (in case of successful validation):

```
{
    "Code": 0,
    "Message": null,
    "Warning": null
}
```

Validation Response (In case of occurring an error):

```
{
    "Code": 500,
    "Message": "System.Exception column 0: MA (unknown column name)",
    "Warning": null
 }
```

or

```
{
    "Code": 500,
    "Message": "This file is not valid: Table of measurement was not found",
    "Warning": null
 }
```

Upon successful validation, the "message" field should always be empty or null, as this indicates the document is valid. The "warning" field may contain optional comments or suggestions on potential improvements to the document.

The "message" field serves as an error indicator, while the "warning" field provides additional feedback when necessary. In cases where validation is successful, the "message" field must remain empty or null, and it is required to be included regardless of the outcome.

2. /databasevaluesbody Endpoint

Description: This endpoint processes incoming requests and returns a predefined template data. The response is automatically validated and serialized based on the predefined data model.

Request:

- Method: POST
- Body: data as a post body

Database Values Response (Example: EDX file):

```
{
	"Compositions": [
	{
	"CompositionElements": [
		{"CompoundIndex": 0, "ElementName": "V", "ValueAbsolute": null,
"ValuePercent": 31.299999237906547},
		{"CompoundIndex": 0, "ElementName": "Mn", "ValueAbsolute": null,
"ValuePercent": 2.700000047683716},
		{"CompoundIndex": 0, "ElementName": "Co", "ValueAbsolute": null,
"ValuePercent": 22.600000381469727},
		{"CompoundIndex": 0, "ElementName": "Ni", "ValueAbsolute": null,
"ValuePercent": 6},
		{"CompoundIndex": 0, "ElementName": "Ho", "ValueAbsolute": null,
"ValuePercent": 37.400001525878906}
	],
	"DeletePreviousProperties": true,
	"Properties": [
		{
		"PropertyID": 0,
		"Type": 2,
		"Name": "Measurement Area",
		"Value": 1,
		"ValueEpsilon": null,
		"SortCode": 10,
		"Row": null,
		"Comment": null
		}
	]
	}
```

```
        ]
}
```

Database Values Response (Example: Resistance file):

```
{
  "CompositionsForSampleUpdate": [
        {
        "Predicate": {
        "Properties": [
        {
        "Type": 2,
        "Name": "Measurement Area",
        "Value": 1
        }
        ]
        },
        "DeletePreviousProperties": false,
        "Properties": [
        {
        "PropertyId": 0,
        "Type": 1,
        "Name": "R",
        "Value": 185942.15625,
        "ValueEpsilon": null,
        "SortCode": 10,
        "Row": null,
        "Comment": "Resistance (Room Temperature) for Measurement Area 1"
        }
        ]
        },
```

Compositions:

The document includes a collection of compositions, and each composition should contain one or more CompositionElement objects to define its chemical composition. Each CompositionElement specifies the individual components that contribute to the overall chemical composition of the composition. It represents an array of composition objects, each describing a distinct composition entity.

- **CompositionElements**: A list of elements that make up the sample's composition.
- **CompoundIndex**: Identifies a specific compound within a database; the same index across elements indicates part of the same compound/mixture. It designates the optimal sequence to display those elements
- **ElementName**: Name of the element (e.g., V for Vanadium, Mn for Manganese).
- **ValueAbsolute**: Absolute quantity or concentration of the element.
- **ValuePercent**: Percentage concentration of the element relative to the entire composition. This shows the proportion of each element relative to the entire composition.

DeletePreviousProperties:

- Boolean indicates whether to delete previous properties before updating the data.
- true deletes old data; false preserves existing data.

Properties:

An array containing additional attributes related to the composition.

- **PropertyID**: Unique identifier for the property.(if it is null, it should be specified). If a property should be created, then please specify the propertyID as 0 for the new property.
- **Type**: Integer representing the property type (e.g., 2 for integers).
- Other types: 0 for None, 1 for float, 3 for string, 4 for BigString.
- **Name**: Name of the property (e.g., "Measurement Area" or "R" for resistance at room temperature)

- R Can be modified based on conditions (e.g., "R - Heating" for resistance measured while the system is being heated or "R -20 Cooling" for resistance measured at -20°C during cooling).
- **Value**: Value of the property (e.g., 1 for "Measurement Area" or 185942.15625 for “resistance”).
- **ValueEpsilon**: Represents a margin of error or uncertainty (set to null if not applicable, meaning no tolerance or error margin is provided.).
- **SortCode**: Numeric code used for sorting or categorizing properties.
- **Row**: Reference to a specific row in the database (set to null if not used).
- **Comment**: Additional context or notes (e.g., "Resistance (Room Temperature) for Measurement Area 1" or null in this sample suggesting no comments are provided).

CompositionsForSampleUpdate:

It refers to updates for pre-existing compositions, such as modifications to properties in various Measurement Areas (MAs).
Since we already have EDX data and multiple compositions (MAs) created, this structure is used to modify the properties inside each MA. Each MA can be treated as a container for measurement data, such as resistance at different temperatures.

Predicate:

- Filters or identifies the specific Measurement Area (MA) to update.
- Used to target a particular MA from a list of multiple MAs (e.g., 342 MAs) for an update.
- The system can use this to refer to the correct MA based on the conditions set in the predicate.
- Properties (inside Predicate): Identifies the "Measurement Area" (e.g., Value = 1 refers to MA 1).

The provided JSON in the Resistance file represents an updated structure for compositions in a system with pre-existing Energy Dispersive X-ray (EDX) data, specifically focusing on Measurement Areas (MAs). With 342 compositions already created, each MA can have additional properties, such as resistance at room temperature, added or updated. The CompositionsForSampleUpdate allows updates to specific MAs using the Predicate to target them. This flexible structure supports tracking multiple resistance values under different thermal conditions, enabling precise measurement management across the sample.

3. / tablebody Endpoint

Description: This endpoint is designed to handle the processing of incoming files by reading the file content, and returning a predefined table data structure as a response.

Request:

- Method: POST
- Body: data as a post body

Tablebody Response (Example: EDX file):

```
table_data = {
     "DataTable": [
           {
          "Spectrum": "Spektrum 1 {1}",
           "In stats.": "Yes",
           "X (mm)": -44.6,
           "Y (mm)": -40.3,
          "C": 80.36562,
          "O": 17.7799,
          "Si": 0.331076,
          "V": -0.3328189,
          "Mn": 0.09434319,
          "Co": 1.12593,
          "Ni": 1.070188,
          "Ho": -0.4342265
           }
     ]
     }
```

This block sets up table_data as a dictionary containing a list of dictionaries, each representing a row in a data table.

- **DataTable:** This is the key for the main data collection, which contains a list (array) of compositions. In this example, it contains a single object, but in practice, there could be more.
- Each object in the "DataTable" list represents a spectrum's data, which includes the following fields:
- **Spectrum**: Describes the spectrum's name or identifier, in this case, "Spektrum 1 {1}". This refers to the first spectrum of some datasets.
- **In stats.**: Indicates whether the spectrum is included in statistical analysis, or not.
- **X (mm)**: Represents the X-coordinate position in millimeters.

- **Y (mm)**: Represents the Y-coordinate position in millimeters.
- **Elements (e.g., C, O, Si, V, Mn, Co, Ni, Ho)**: These are the chemical elements detected in the spectrum, with their corresponding amounts or concentrations in percentages or relative values.

Tablebody Response (example: Resistance file):

```
{
  "DataTable": [
    {
      "x": 0,
      "y": 0,
      "R1": 1820709.375,
      "R2": 1816450.625,
      "R3": 1816368.5,
      "unknown_1": 9.9845686,
      "unknown_2": 0.0000055,
      "R_ave": 1817842.8333333333,
      "R_median": 1816450.625
    },....
```

Each object in the Data table array contains:

- **x**: The X-coordinate of the measurement.
- **y**: The Y-coordinate of the measurement.
- **R1, R2, R3**: These are three different resistance values.
- **R_ave:** average of the three resistance values.
- **R_median**: This is the median value of resistance values which is the middle value of them when sorted.