COVID Data API - Implementation Report

Introduction

A Python FastAPI application called the COVID Data API was created to give users access to COVID-19 data through an easy-to-use interface. The programme loads information from a CSV file containing COVID-19 statistics and exposes endpoints to retrieve nations, WHO regions, and the overall death toll based on different filters.

Implementation Process

The implementation of the COVID Data API involved the following steps:

Project Setup: The project was set up by installing the required dependencies and starting a FastAPI application. FastAPI, uvicorn, and csv were the three primary dependencies used.

Data Loading: The COVID-19 data from the CSV file was loaded into memory and kept in the database list. The file was read and the data rows were extracted using the csv module. To remove the header row from the data, it was skipped.

Endpoint Implementation: For the needed functions, the required endpoints were put in place. The API offers endpoints for retrieving distinct nations, distinct WHO regions, and the overall number of deaths based on filters such nation, region, and year.

Documentation and Testing: Using the built-in documentation generation capability of FastAPI, the API was documented. Information on the accessible endpoints, their parameters, and anticipated answers is provided in the documentation. To assure the accuracy of the functionalities implemented, the API underwent a thorough testing process.

Challenges Faced

During the implementation of the COVID Data API, the following challenges were encountered:

Data Parsing: Careful handling of the data rows and information extraction were necessary when parsing the data from the CSV file. For accurate data retrieval, proper indexing and data type conversion were essential.

Filtering Logic: Complex conditional statements were used to implement the filtering logic to extract the total deaths by country, region, and year. It was difficult to ensure proper filter application while retaining code readability.

Error Handling: To give API consumers useful error messages, it was crucial to gracefully handle errors and exceptions. To ensure the API's resilience, appropriate error handling and exception management were put into place.

Additional Functionalities

In addition to the core functionalities, the following enhancements were made to improve the usability and functionality of the COVID Data API:

API Redirection: Users of the API are now automatically sent to the documentation page via the endpoint. The ease of access to the API documentation enhances the user experience.

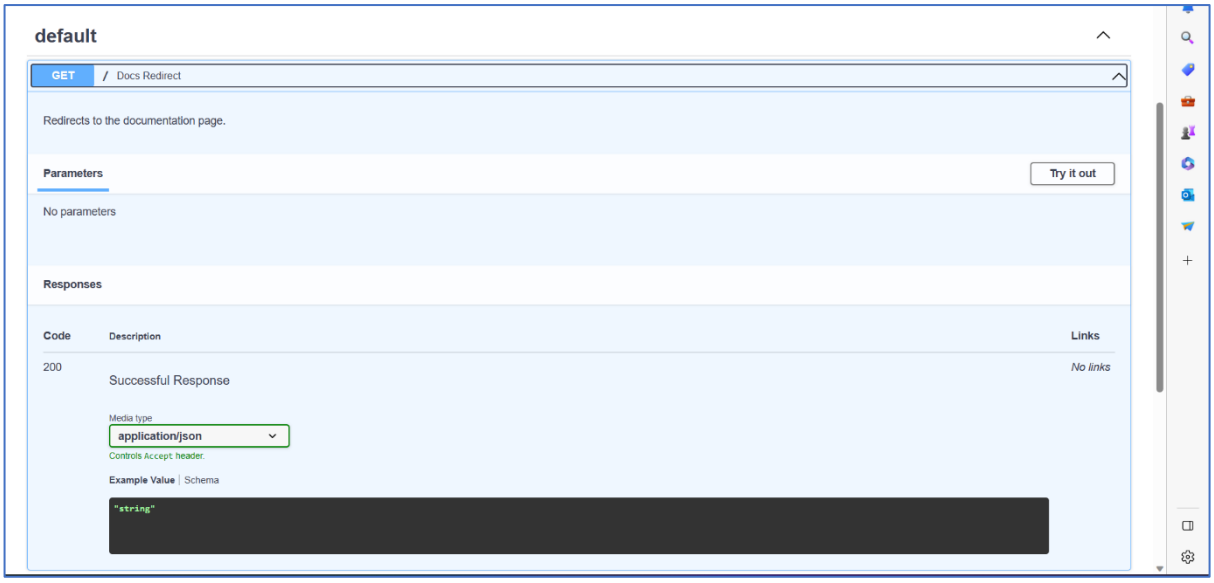
Error Reporting: In order to give instructive error messages in the event of exceptions or failures, detailed error reporting was implemented. This provides crucial information for troubleshooting and aids users in understanding the cause of the failure.

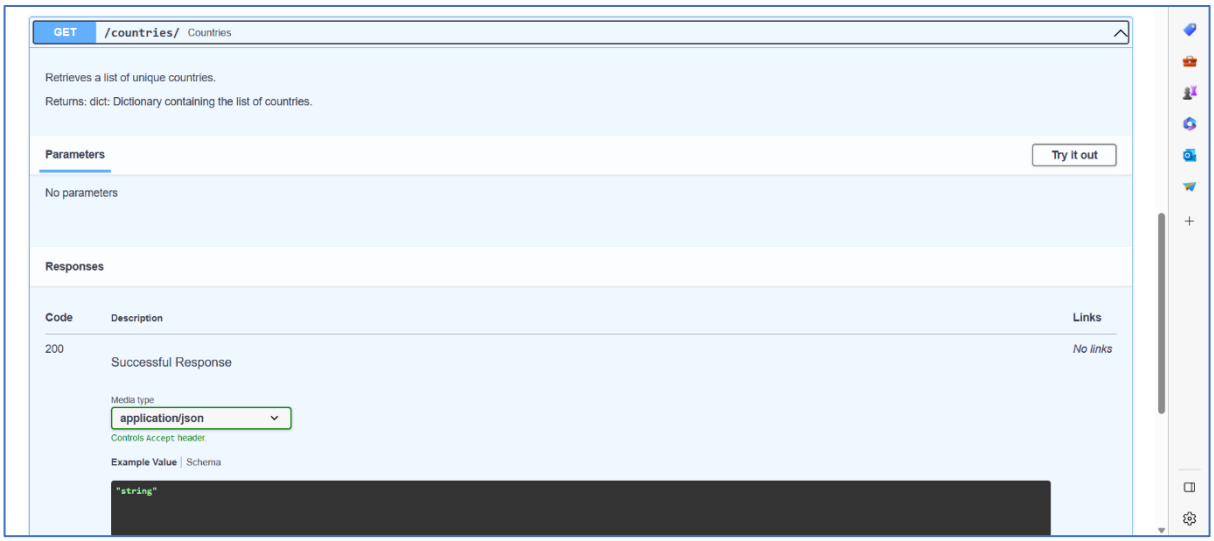
Enhanced Documentation: Each endpoint's parameters, expected replies, and thorough descriptions were added to the API documentation. This aids in the comprehension of the functionality and the efficient usage of the API by users.

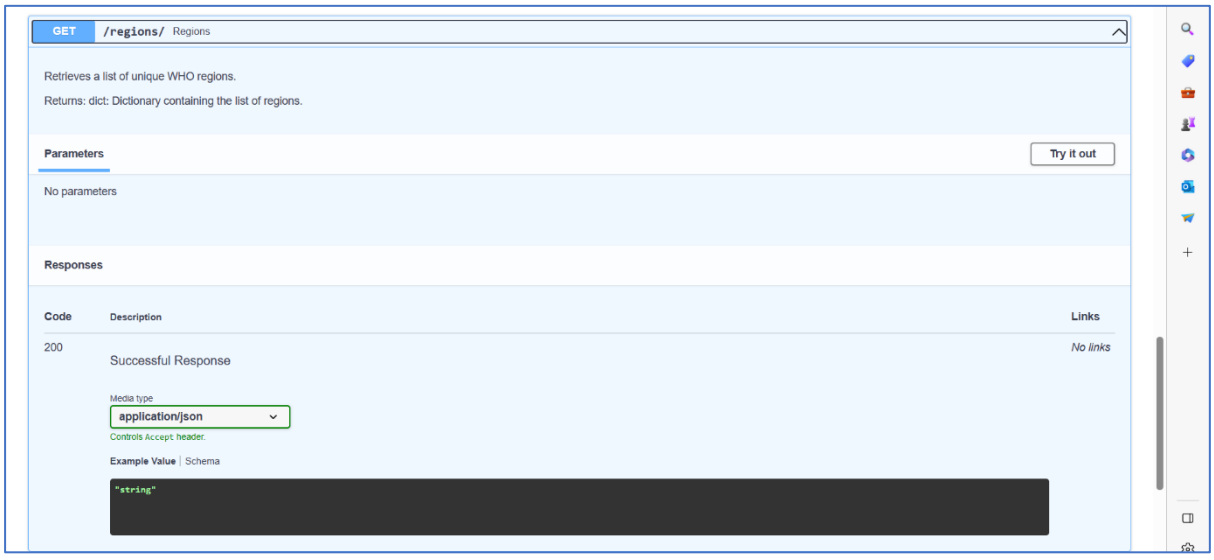
Conclusion

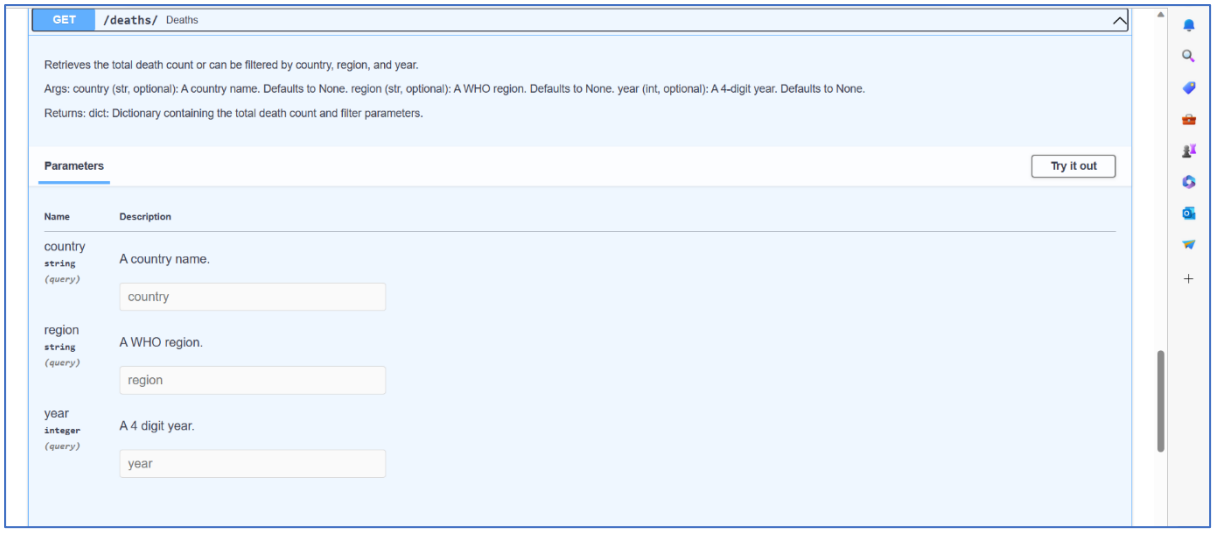
A Python FastAPI application called COVID Data API effectively implements the required functionality to give users access to COVID-19 data. Data loading, endpoint implementation, documentation, and testing were all part of the implementation process. During the development process, issues with data processing, filtering logic, and error handling were resolved. To enhance the usage of the API, other features including API redirection and improved error reporting were put in place. Accessing COVID-19 data is made simple and dependable through the COVID Data API.

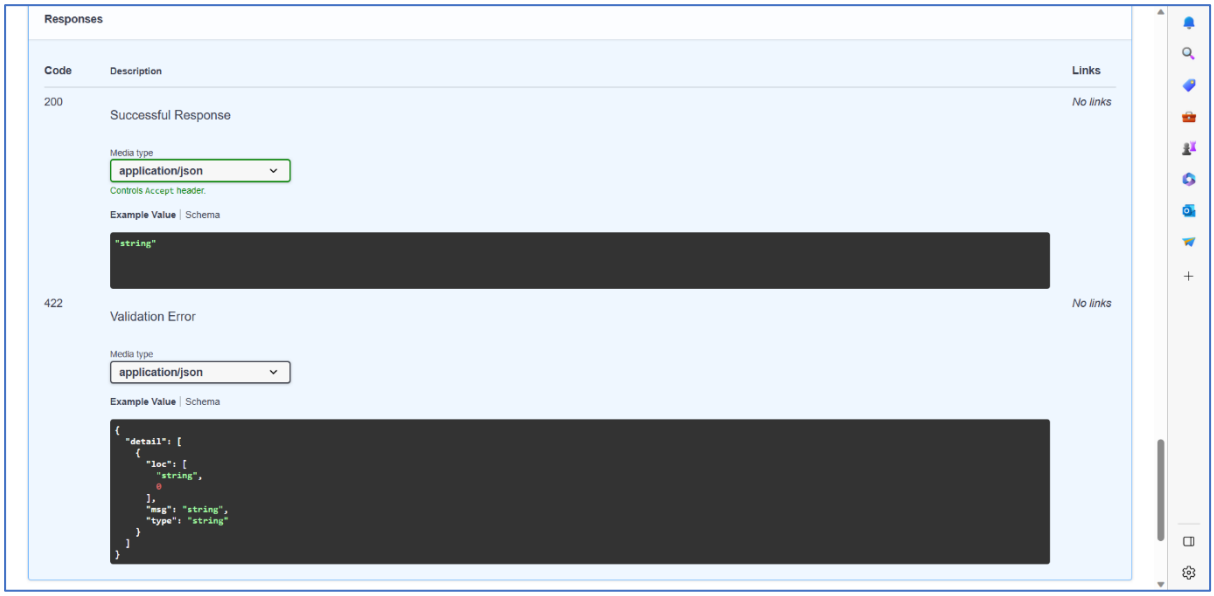
Screenshots

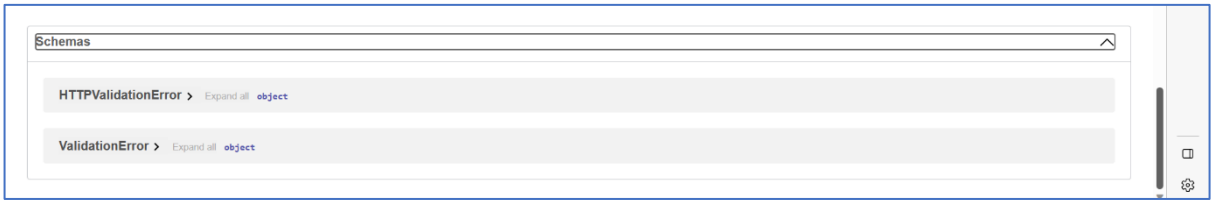

default
GET / Docs Redirect
Redirects to the documentation page.
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"


GET /countries/ Countries
Retrieves a list of unique countries.
Returns: dict: Dictionary containing the list of countries.
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"


GET /regions/ Regions
Retrieves a list of unique WHO regions.
Returns: dict: Dictionary containing the list of regions.
Parameters
Try it out
No parameters
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"


GET
/deaths/ Deaths
Retrieves the total death count or can be filtered by country, region, and year.
Args: country (str, optional): A country name. Defaults to None. region (str, optional): A WHO region. Defaults to None. year (int, optional): A 4-digit year. Defaults to None.
Returns: dict: Dictionary containing the total death count and filter parameters.
Parameters
Try it out
Name
Description
country
string
(query)
A country name.
country
region
string
(query)
A WHO region.
region
year
integer
(query)
A 4 digit year.
year


Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"
422
Validation Error
No links
Media type
application/json
Example Value | Schema
{
"detail": [
{
"loc": [
"string",
0
],
"msg": "string",
"type": "string"
}
]
}


Schemas
HTTPValidationError
Expand all
object
ValidationError
Expand all
object