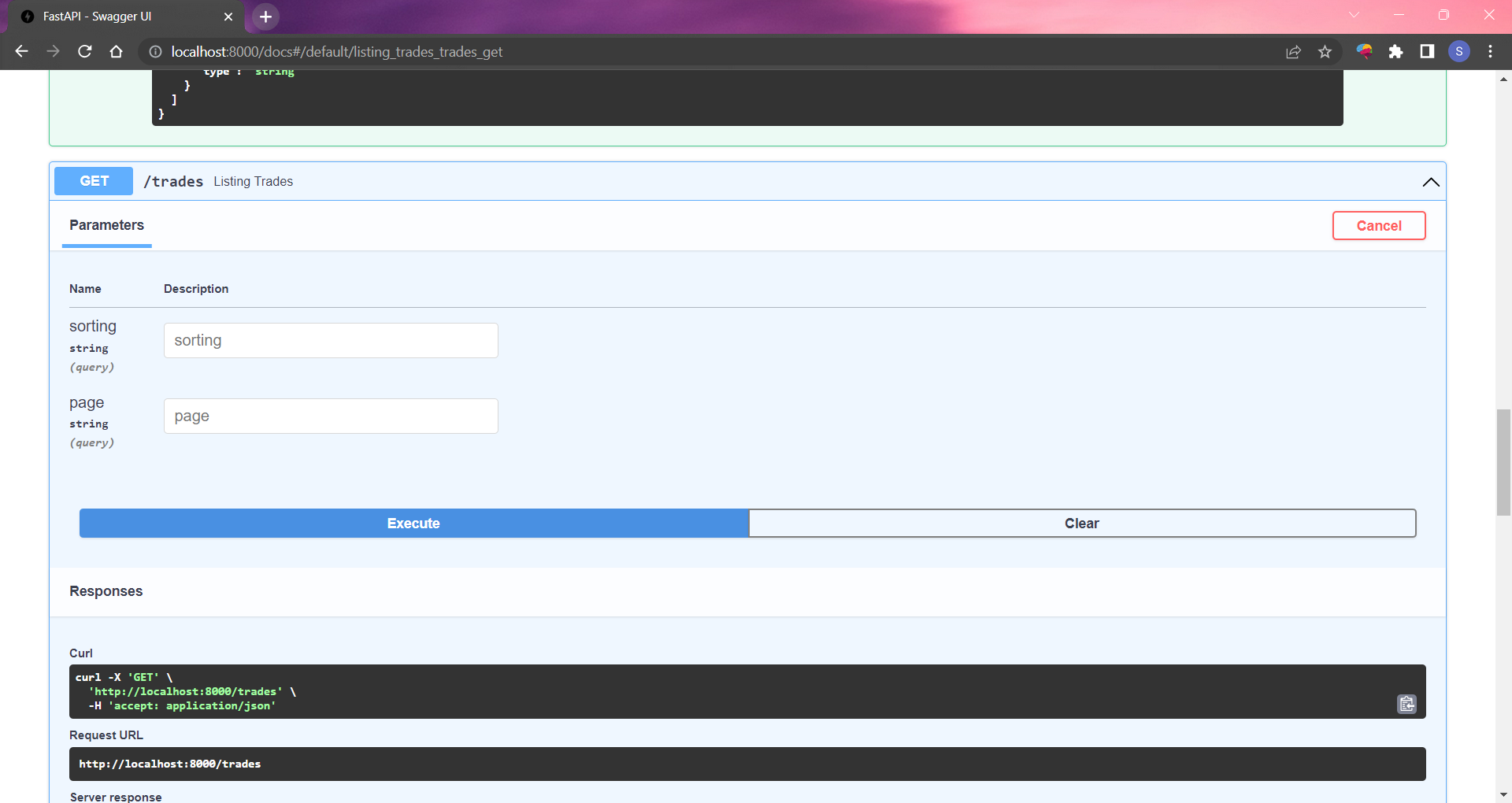
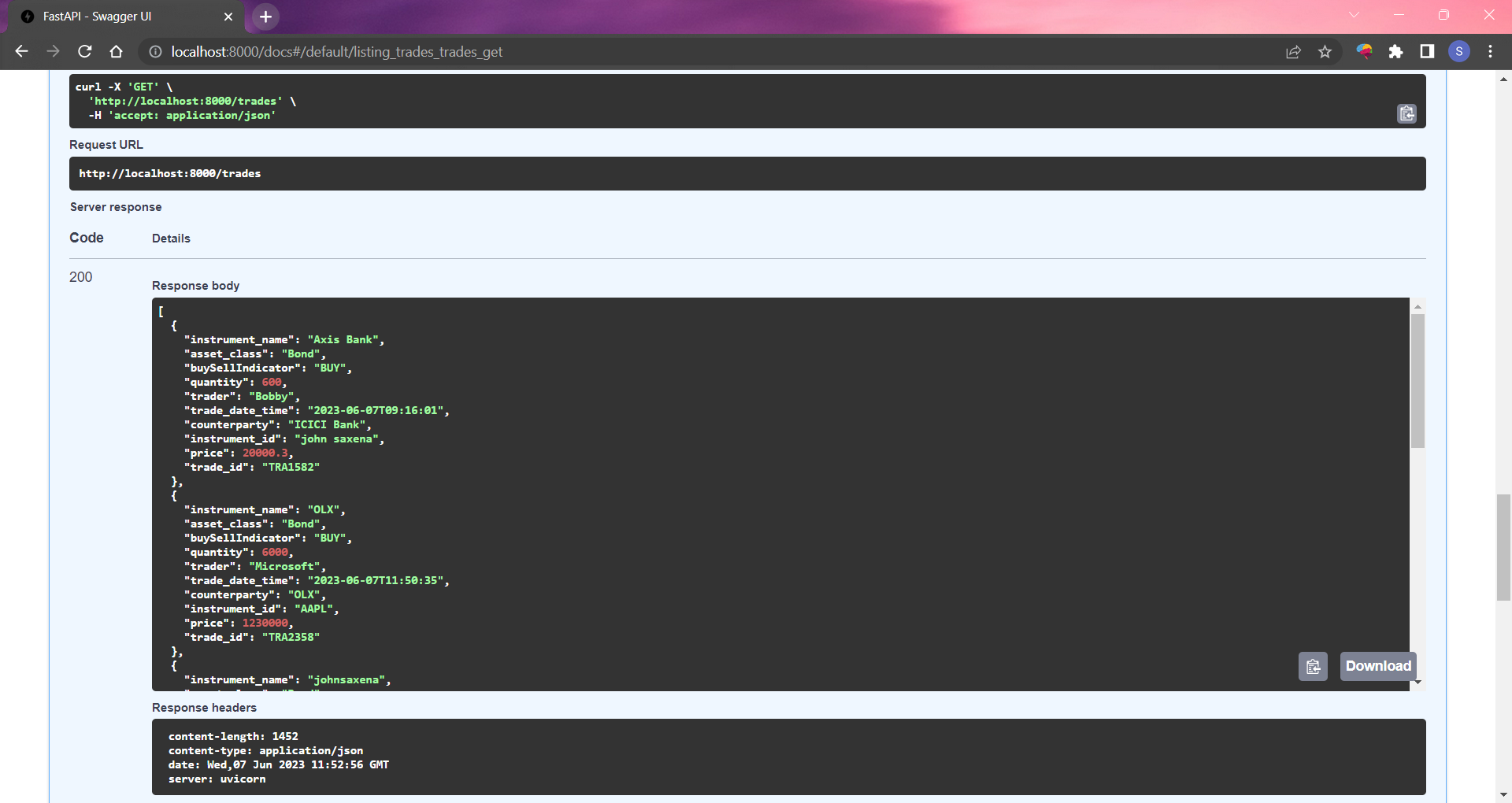
**Name: Sanyam Jain**

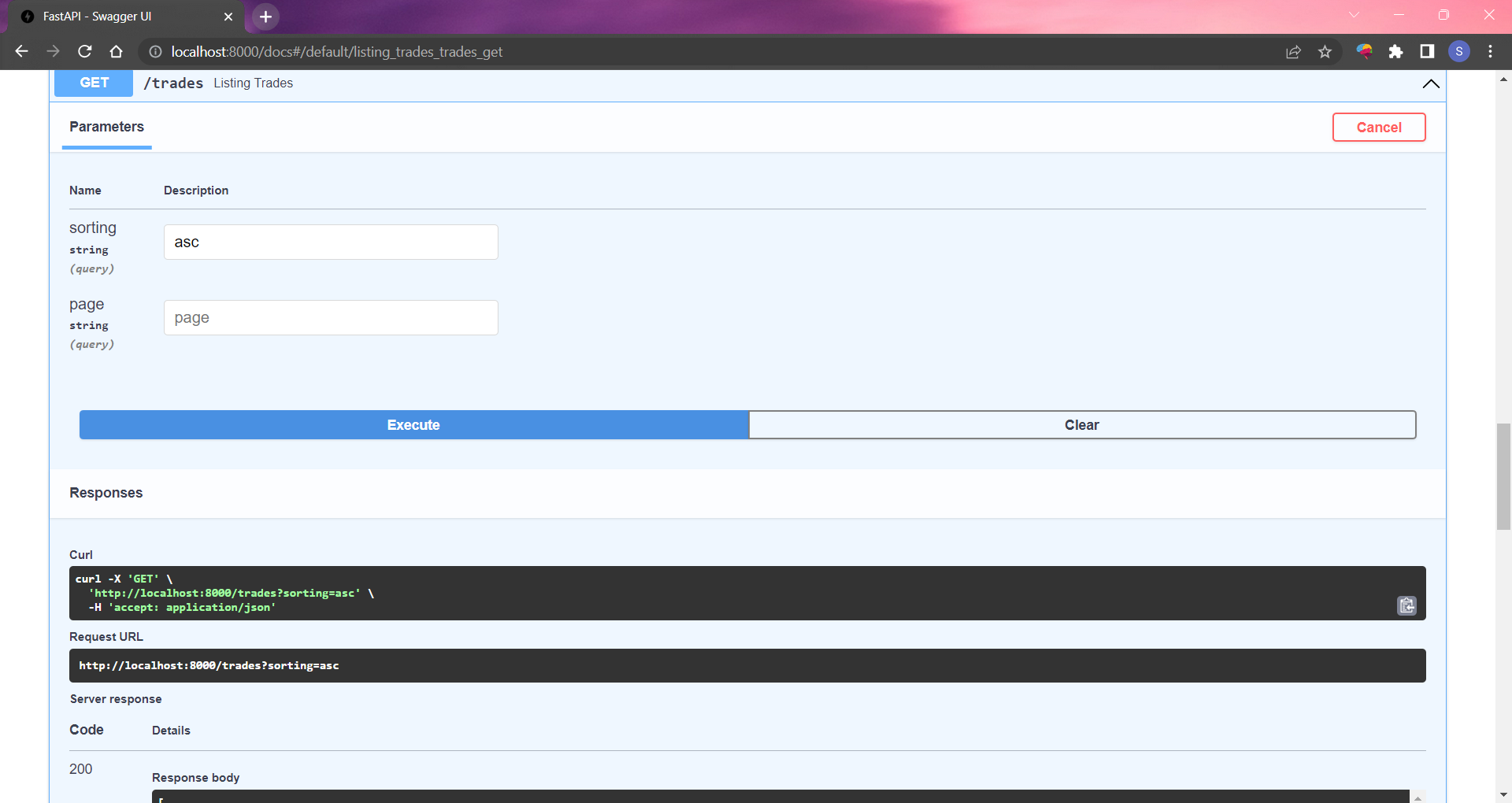
I have used FastAPI, Pydantic, and MySQL in the Python language. MySQL was an easily compatible database for Python so I have used it. I have made the schema as informed and saved it in the schema.py file. Then I made the model representing the table and the features inside the table I will be adding to the MySQL database. The datatype was used based on schema and so was the required attribute also. The trade\_id was made as the primary key in the database based on its uniqueness. To avoid primary key error I am generating the trade\_id randomly in the format “TRAdddd”, where the last four digits are randomly generated by the code.

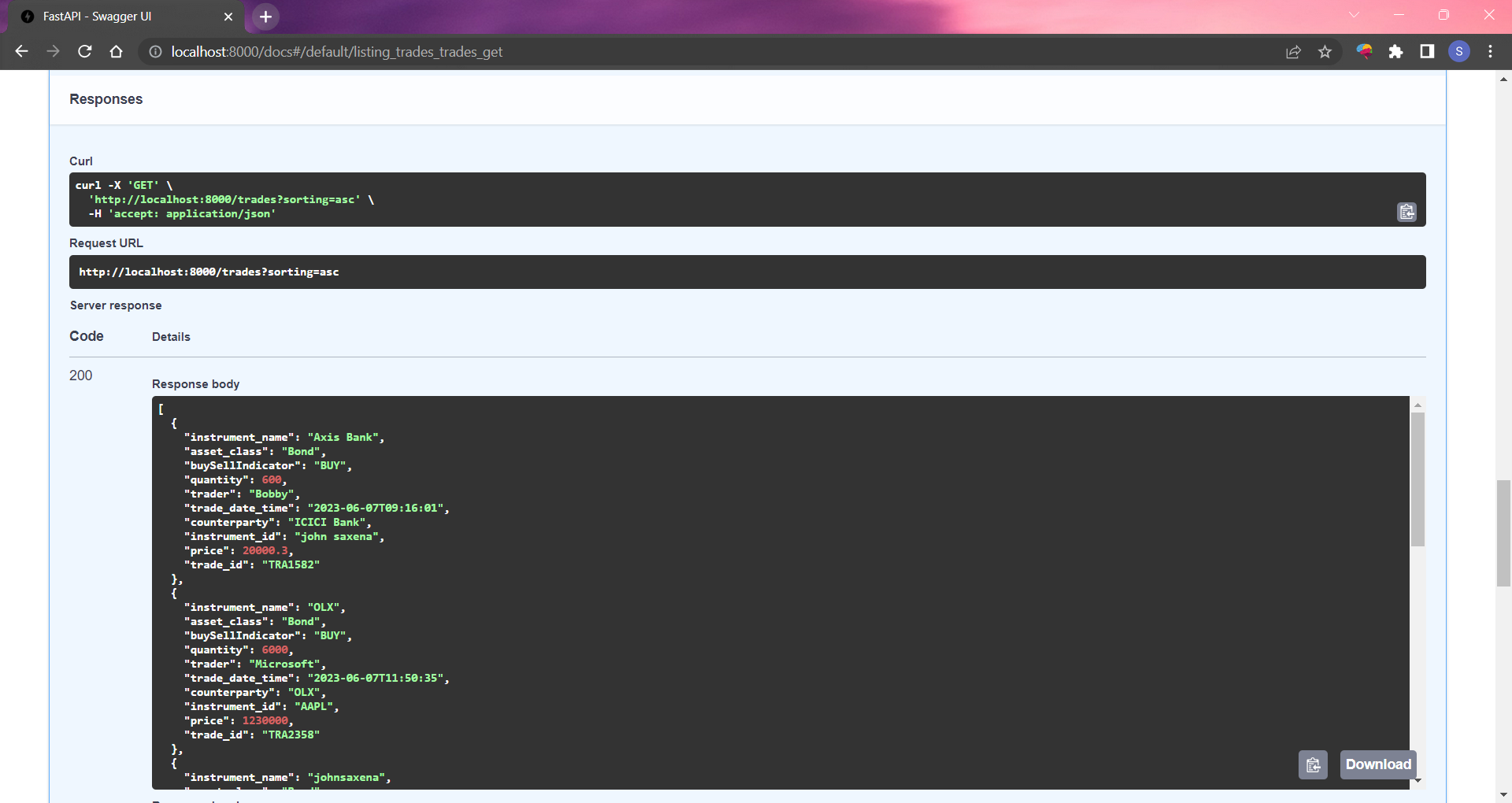
The Listing Trades API has features for also sorting and viewing them based on pages. I have taken the number of rows as constant number 5 which will list 5 trade objects on each page moreover the sorting is done both for ascending order and descending order on trade\_id. The sorting and paging are the optional parameters that can add up in the listing trade URL to get the desired results. The below image shows the listing trade result having all the trade items listed below.



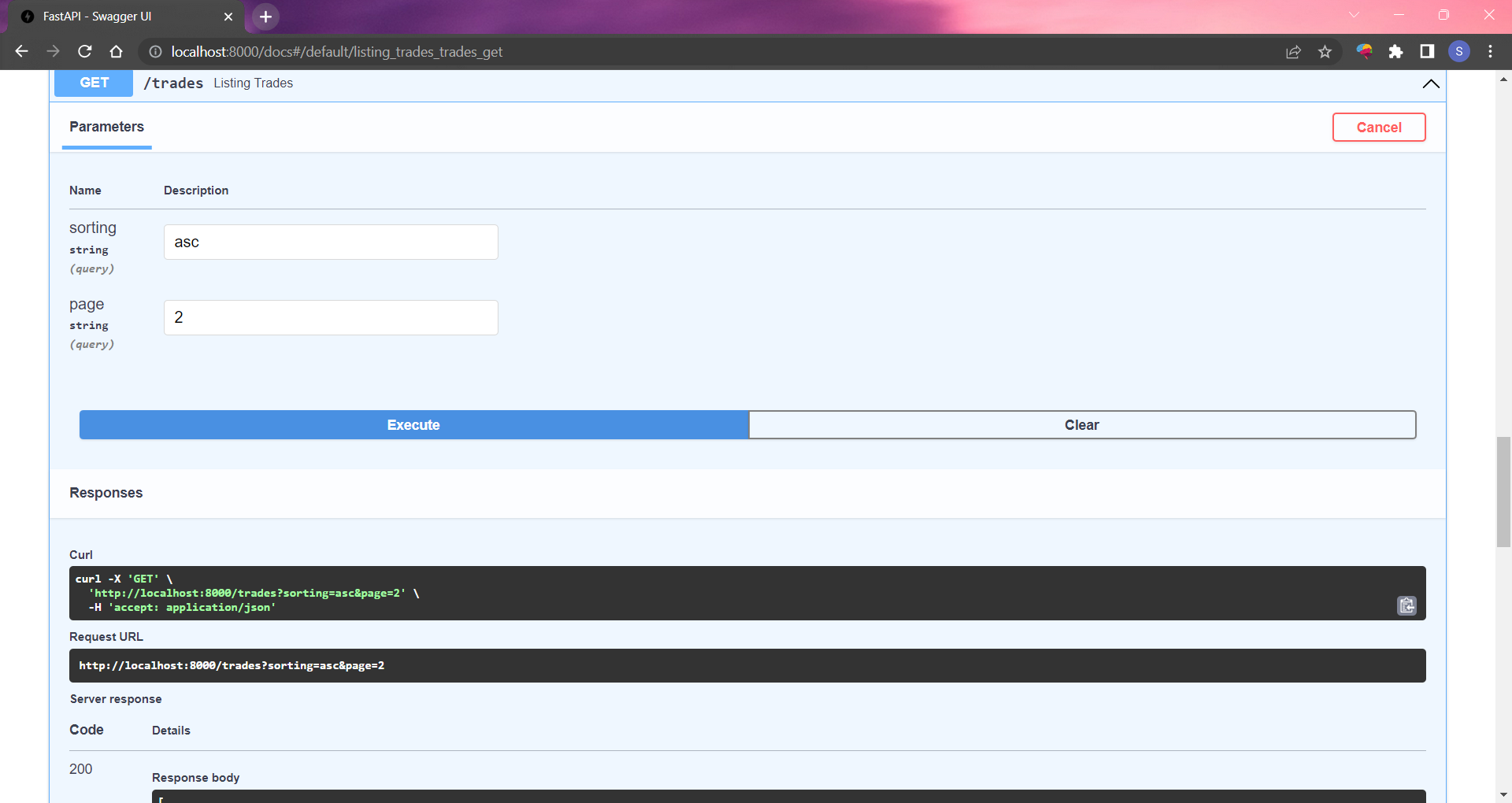


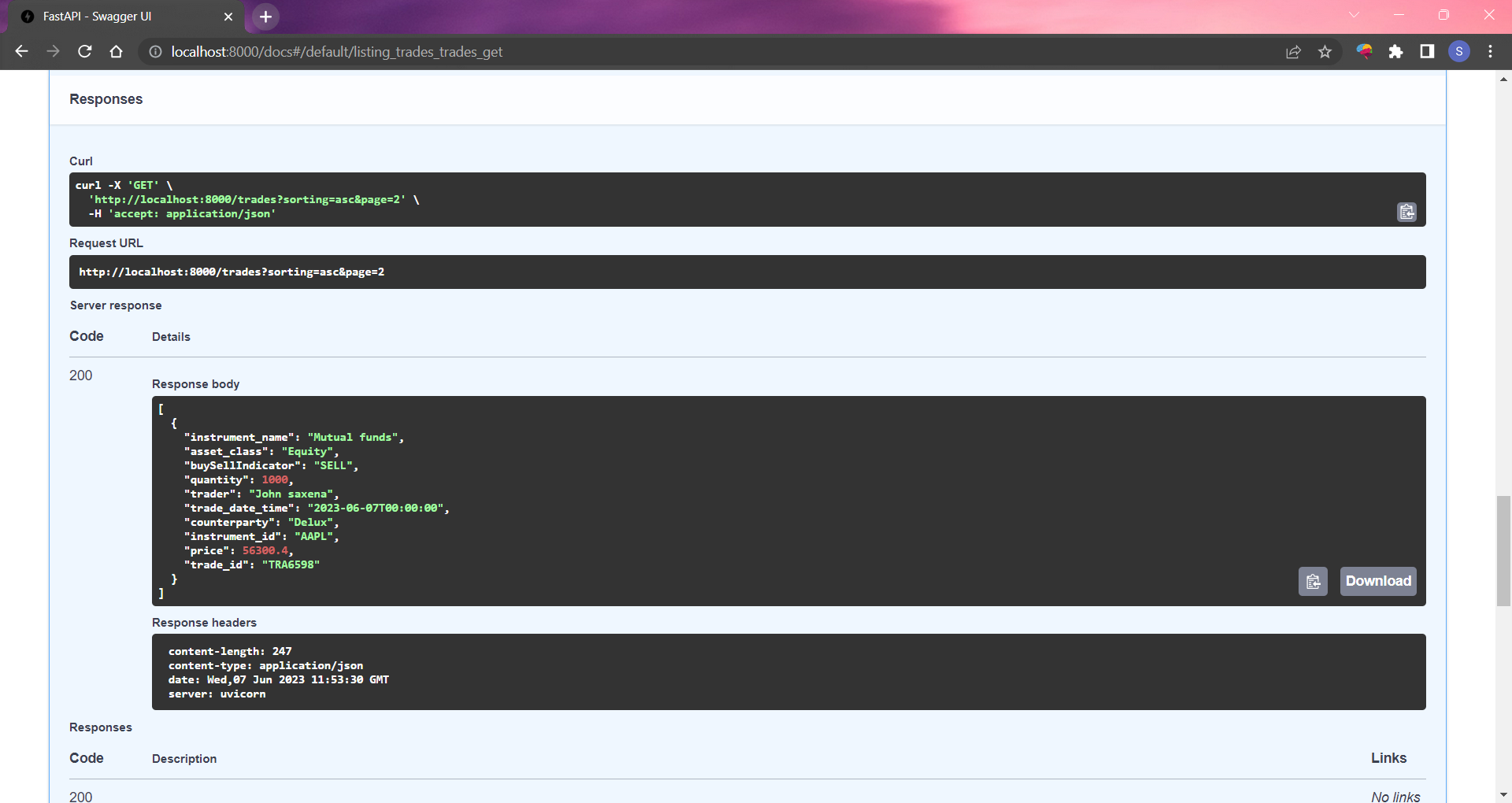
The picture below shows the sorting result of the trade items sorted in ascending order by the trade\_id.



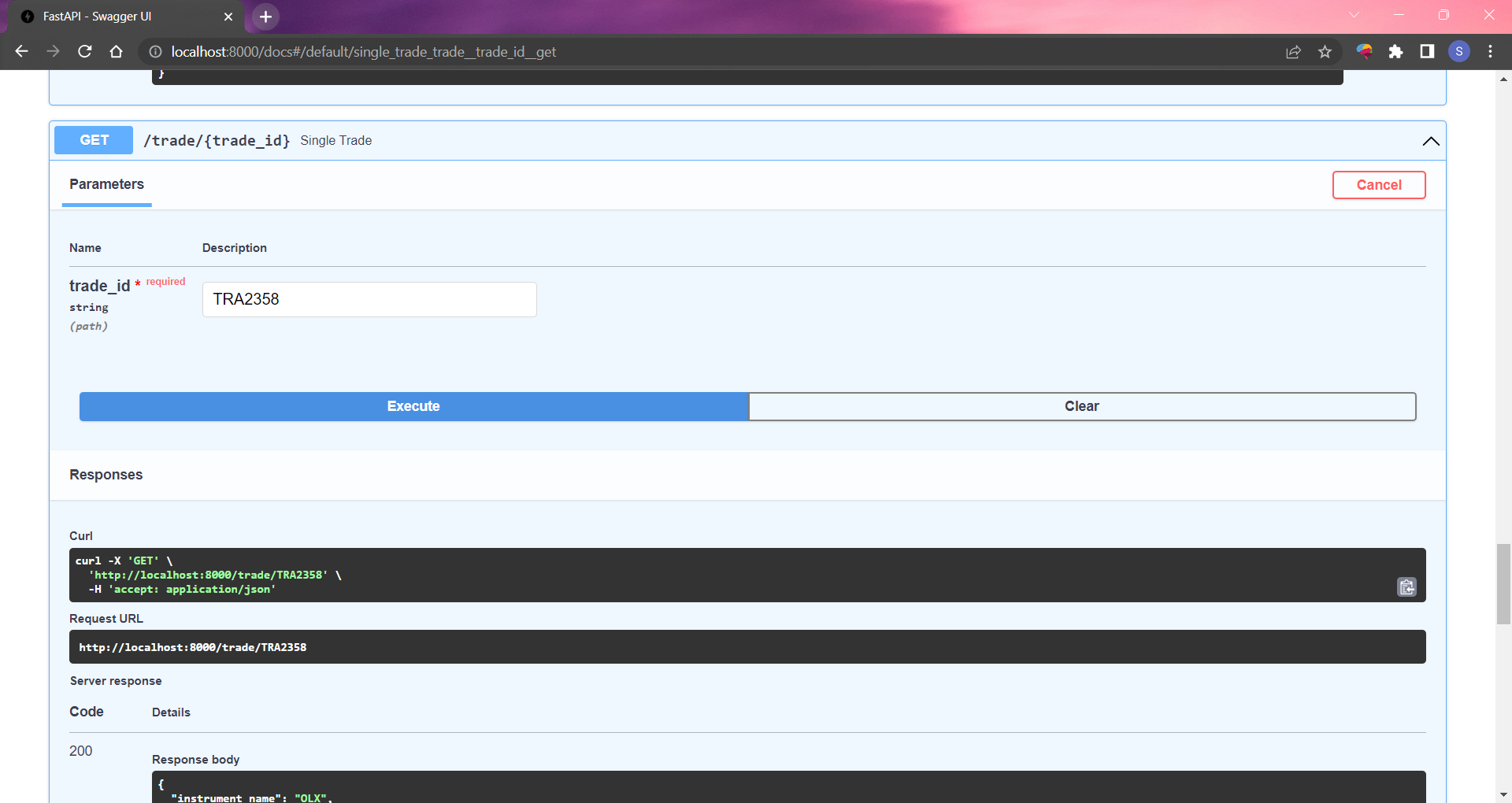


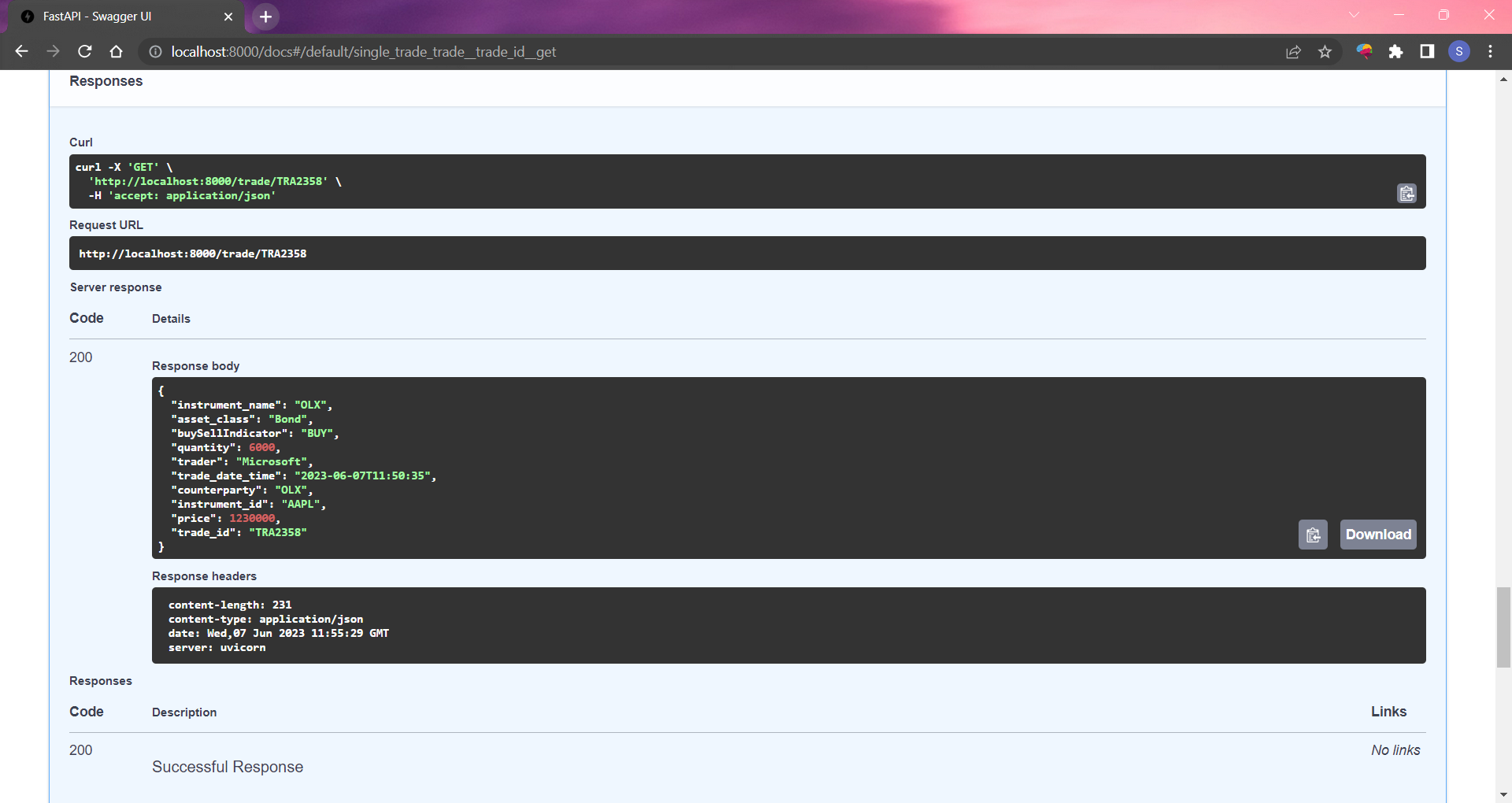
The picture below shows the sorting with the paging result of the trade items and this will show the trade items after the 5th item as the page number was selected as 2 and the number of items shown on the page is 5 (constant).



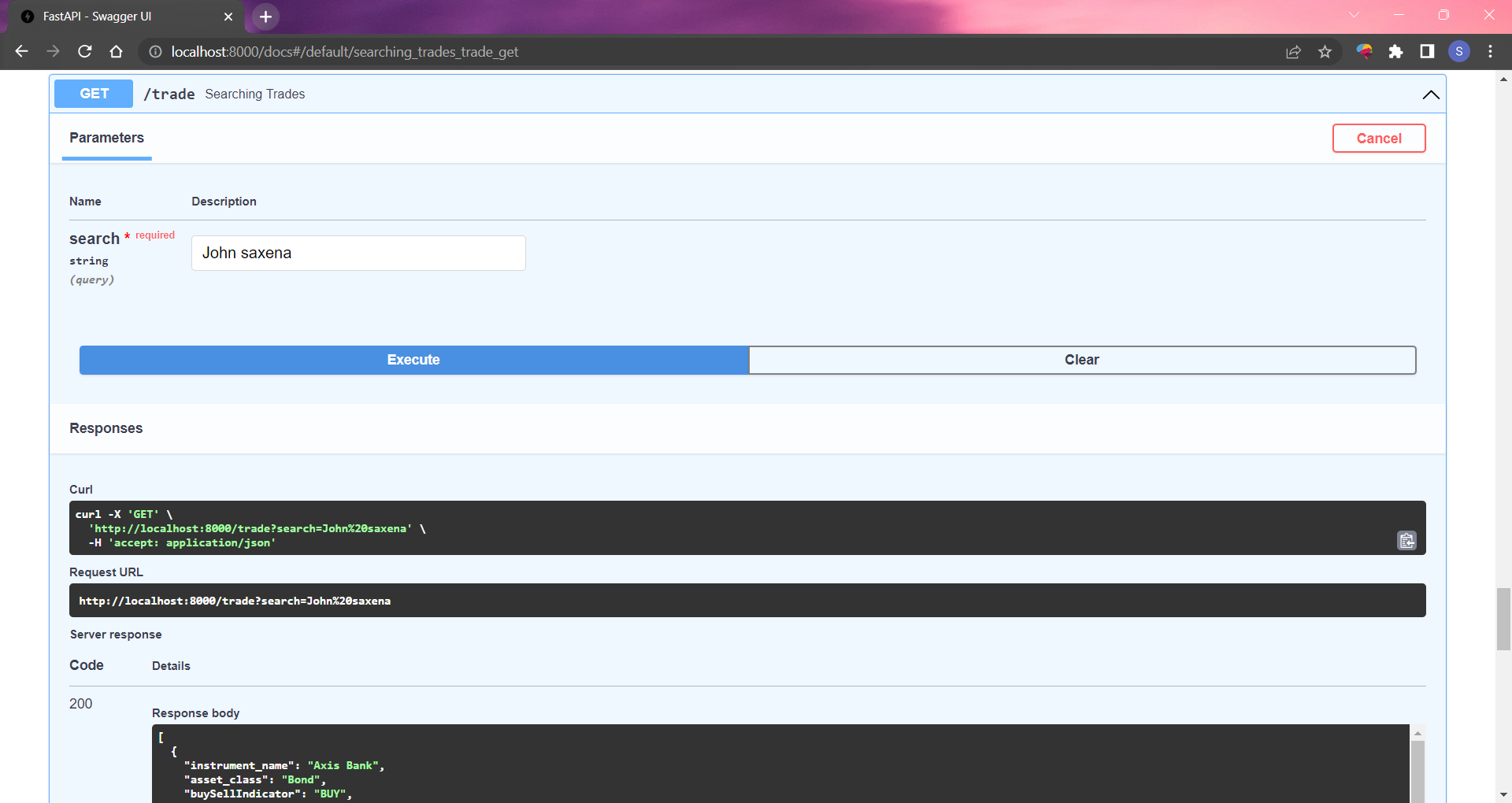


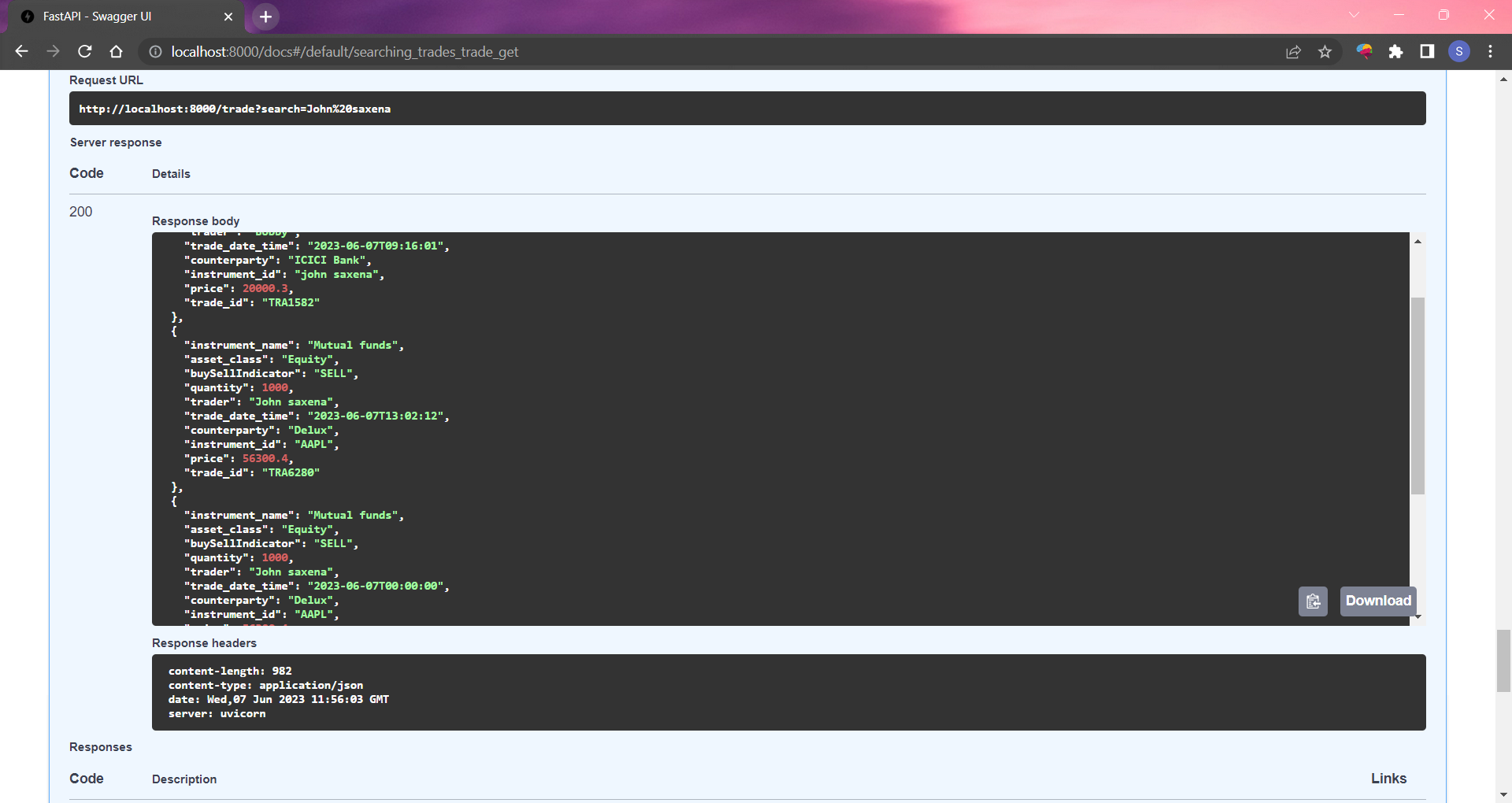
Then for the single trade, I am taking the trade\_id from the user and display the trade details for the specified trade\_id.



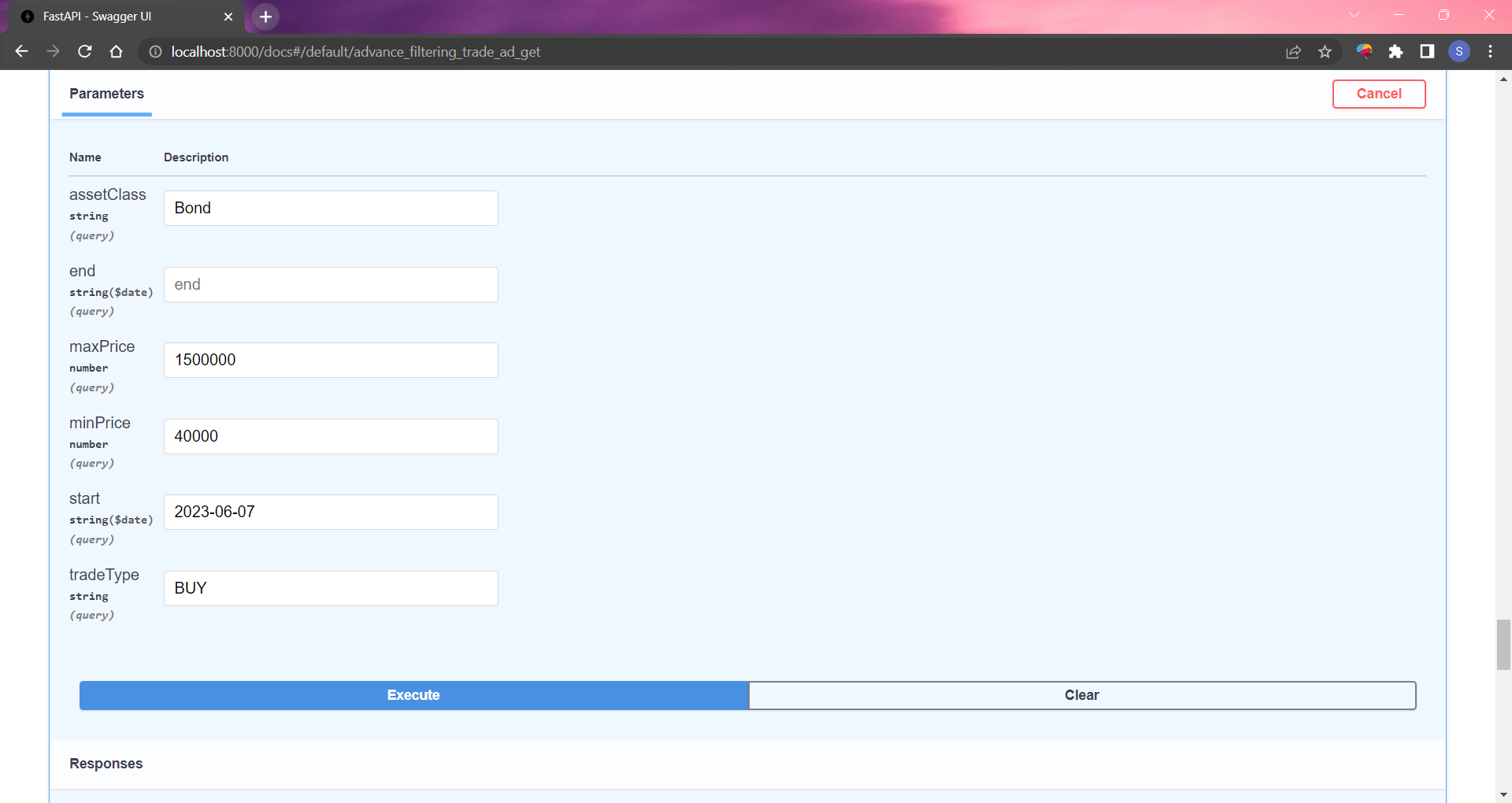


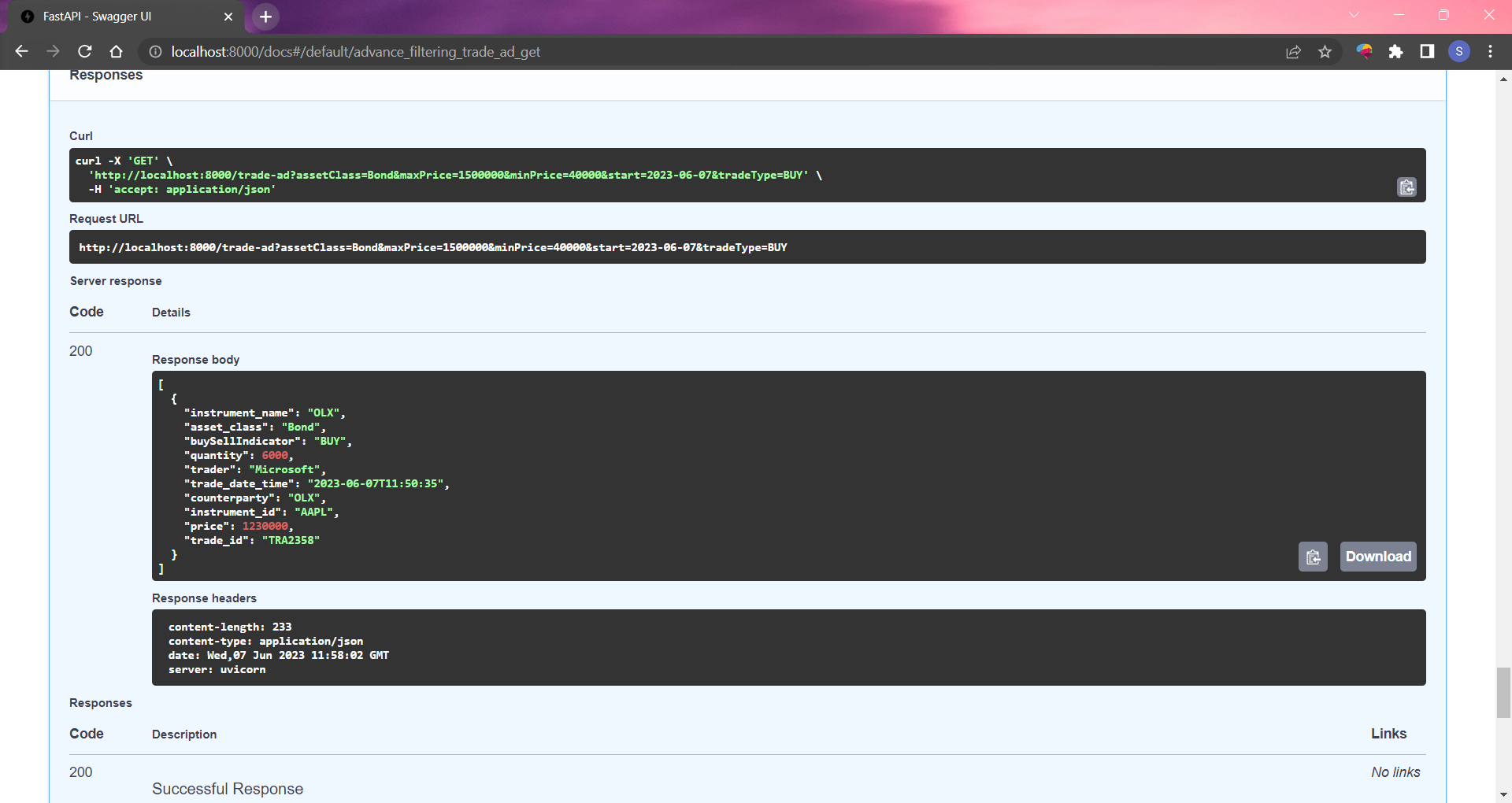
Then for searching trades, I have the search query parameter which takes the string value from the user and searches for the string in the columns counterparty, instrument\_id, instrument\_name, and trader moreover it checks for all of them together without case sensitivity. It will filter the search query for each column and then combine the results using the “or” method to union all the filtered items. The below pictures show the result for the searching trade API call.





Then for the advanced filtering I have used all the mentioned query parameters which can be optional and the result for each query parameter is also checked for all the other query parameters mentioned by the user and make the filtering for the items which satisfy all the query parameters. The query parameter assestClass is directly checked with the trade items assest\_class attribute, for the start and end query parameters which are date will be compared with the trade\_date\_time and selected all the items which lie inside the start and end parameters where either start can be given as input or the end and even both of them, then the minPrice and the maxPrice parameters are compared with price attribute in the trade items moreover here the input can be either minPrice or maxPrice and even both of them, then for the last query parameter tradeType it is checked with the buySellIndicator attribute in the trade items with case insensitivity option.





This is an additional post API for the trade to dynamically add the rows in the model.

