The Generated Web API

1 CONTENTS

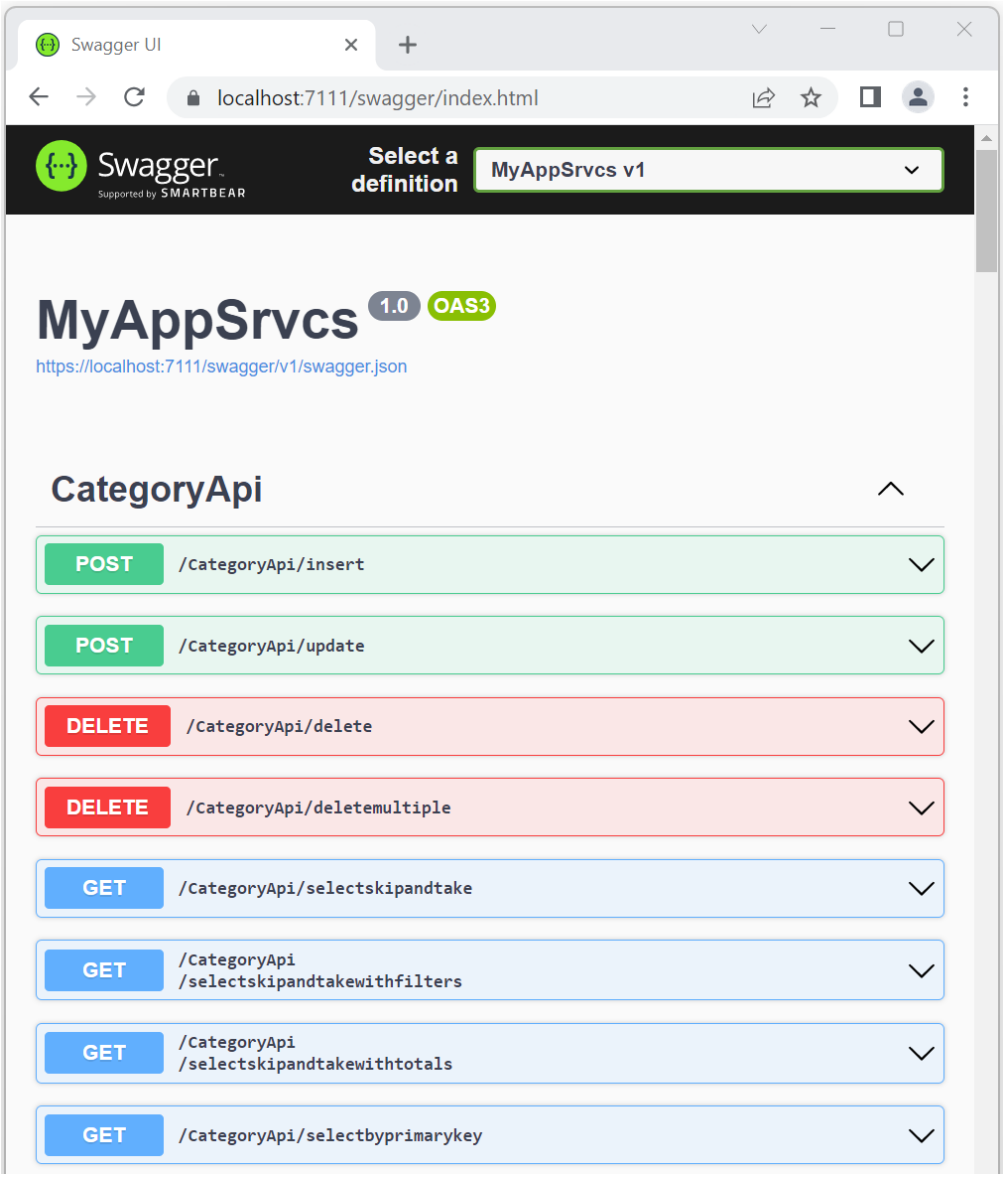
The Generated Web API

1 INTRODUCTION

This topic will show you how to test the generated Web API endpoints.

2 SWAGGER INDEX PAGE

When you run the generated *Web Application* and *Web API* projects by pressing F5 in Visual Studio, two browsers are launched, one for each of the projects respective. The *Web API project* will launch the *Swagger Index Page* by default, shown below. The *Swagger Index Page* contains *Web API Endpoints* for each of the database table that you generated code for.



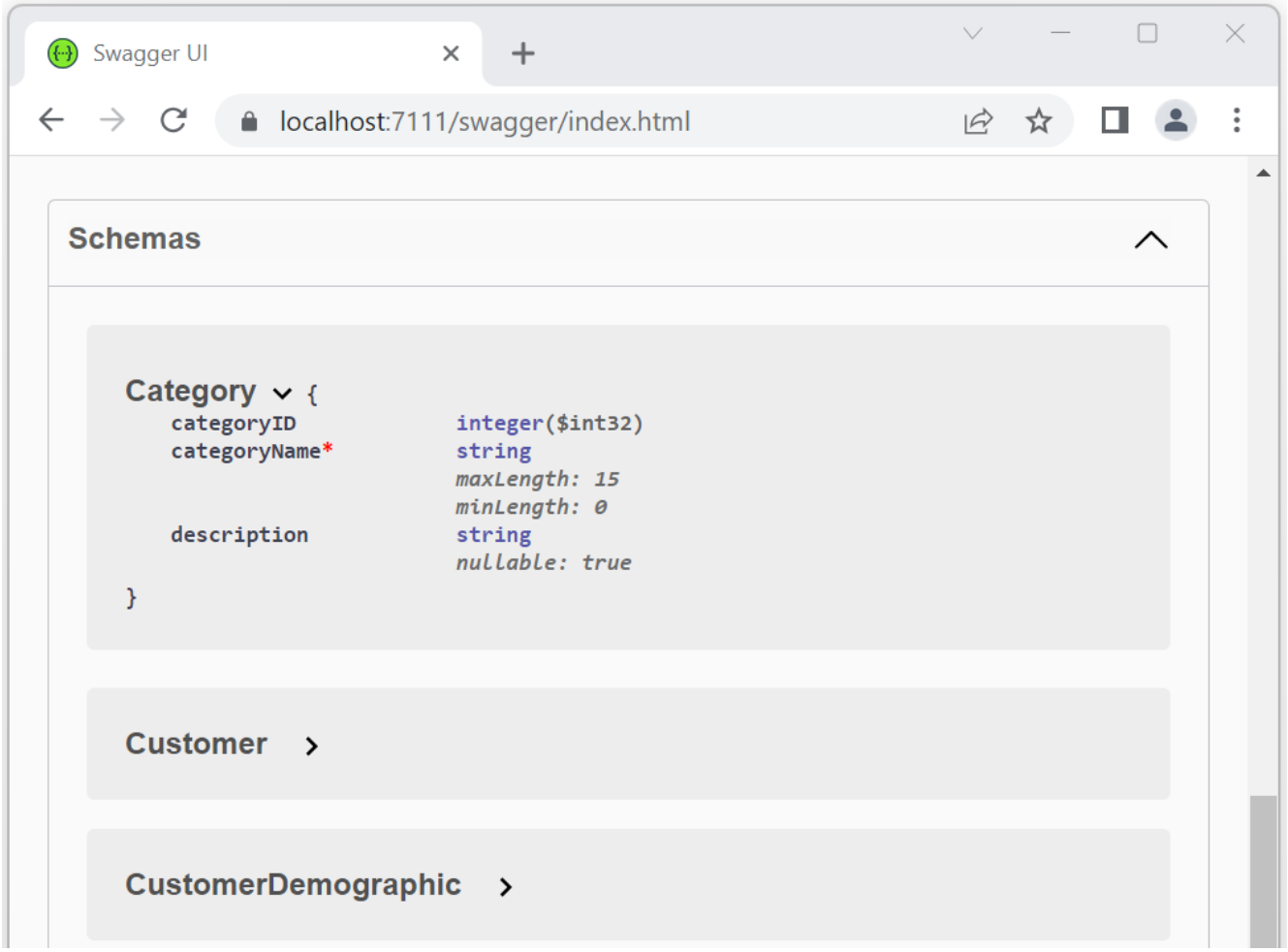
2.1 WEB API ENDPOINTS?

In the example above, the ***Category****API* which is derived from the *Categories* table in the *Northwind* database shows the *Web API Endpoints* available for the *Categories* table. If you move further down the web page you'll see other Web APIs such as *CustomerAPI, CustomerAPI, EmployeeAPI, ProductAPI*, etc, where each also have their own *Web API Endpoints* just like the *CategoryAPI*.

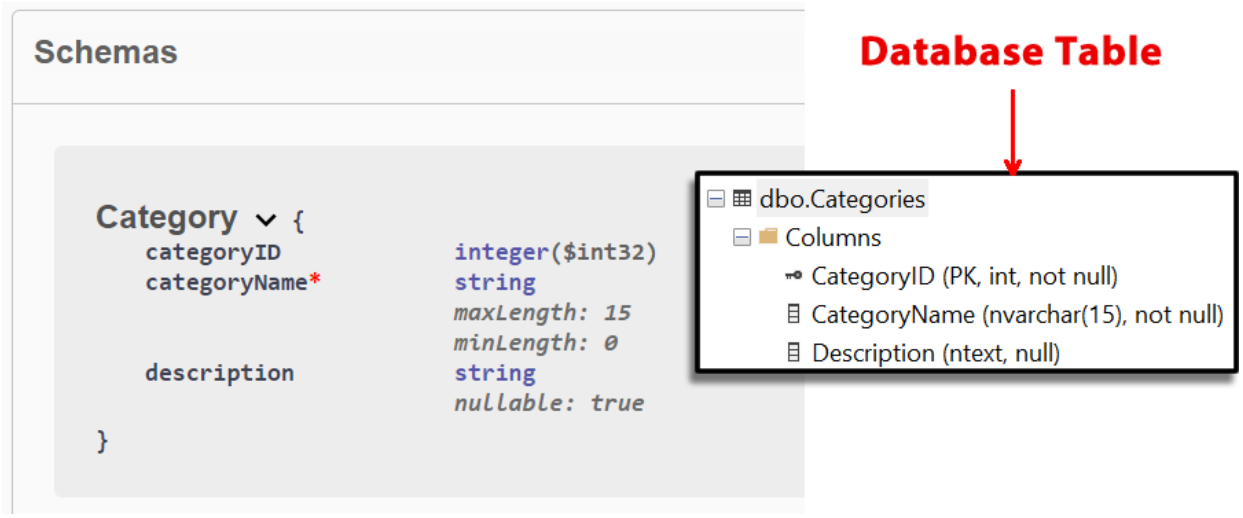
2.2 WEB API SCHEMA?

Further down the web page you'll notice that a *Schema* for each *Web API* is shown. The *Schema* shows the structure of the respective Web API. This is how we know how to interact (what type of data to pass) with the *Web API Endpoints*. For example, the ***Category****API* has 3 fields, *categoryID*, *categoryName*, and *description*. Each of these fields shows their type and other restrictions.

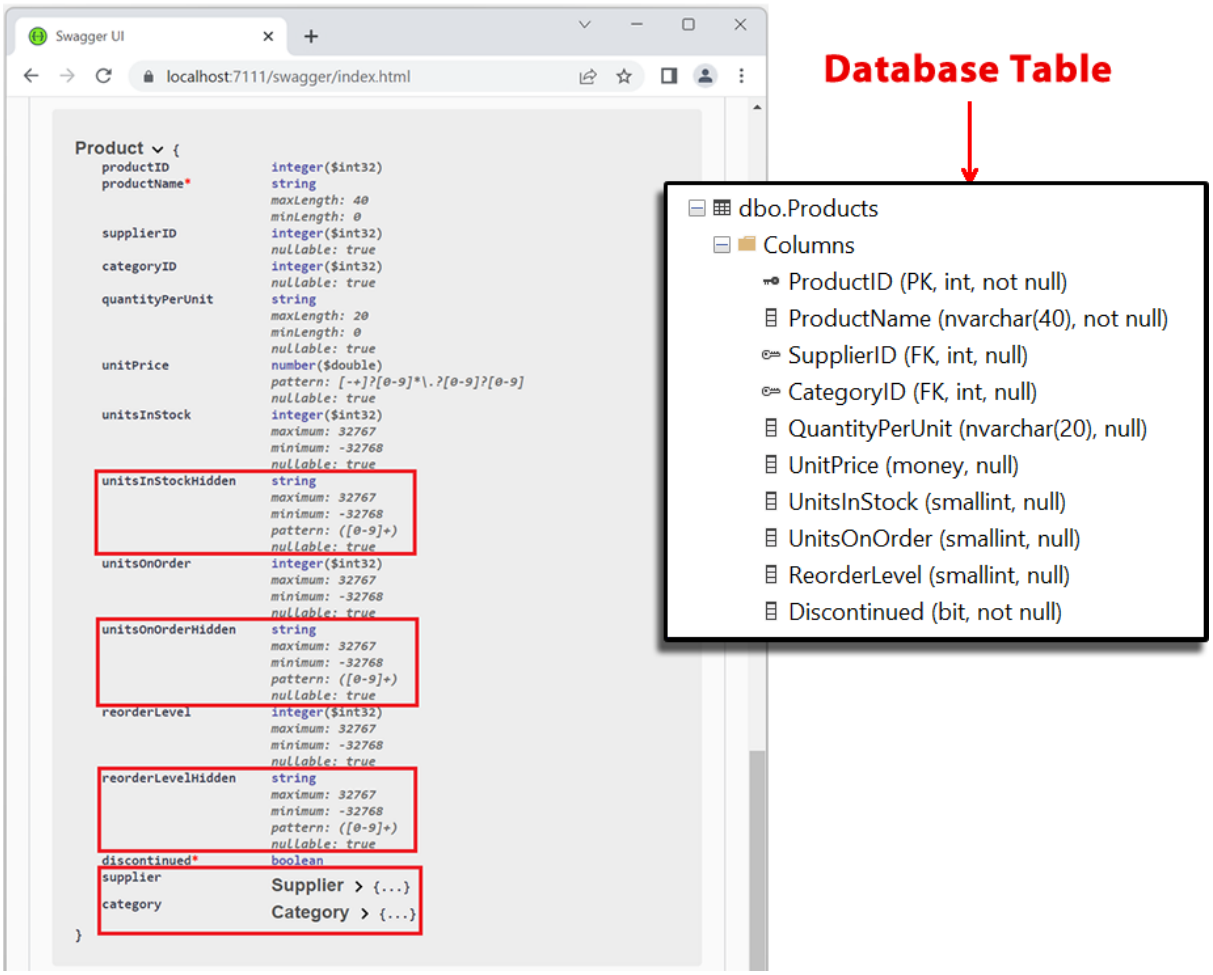
a. **categoryID**: int32 (required)
b. **categoryName**: string (required), maximum length 15 characters.
c. **description**: string, no maximum length (nullable – not required)



For most parts the *Schema's Fields* are direct reflection of the database table fields it represents. For example, the *Category Schema's Fields* are directly related to the *Categories Database Table Fields*.



Some *Schemas* have more fields in them than the *Database Table* it represents, like the *Product Schema*. When interacting with the ***Product****API Schema* all that is required are the *Fields* available in the *Products Database Table*. We will show this later when we try to insert a record in the *Products Tab*le.

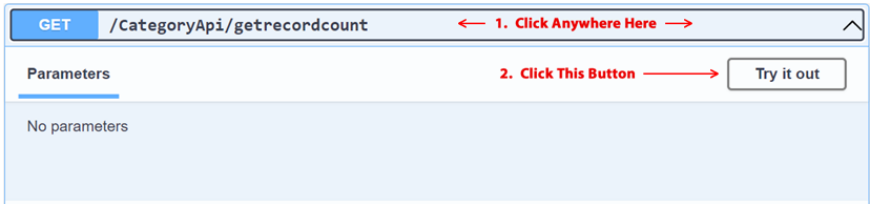


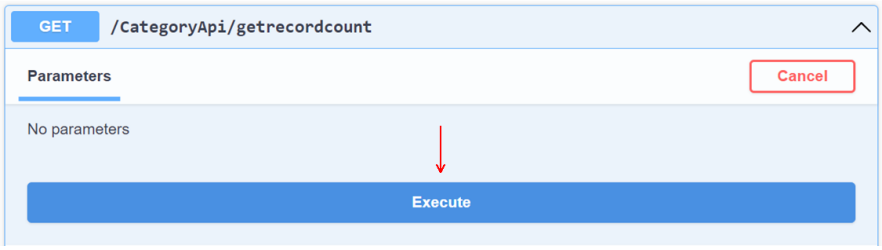
3 TESTING WEB API ENDPOINTS

As developers this is probably self-explanatory. However, we will show how to test some of these endpoints.

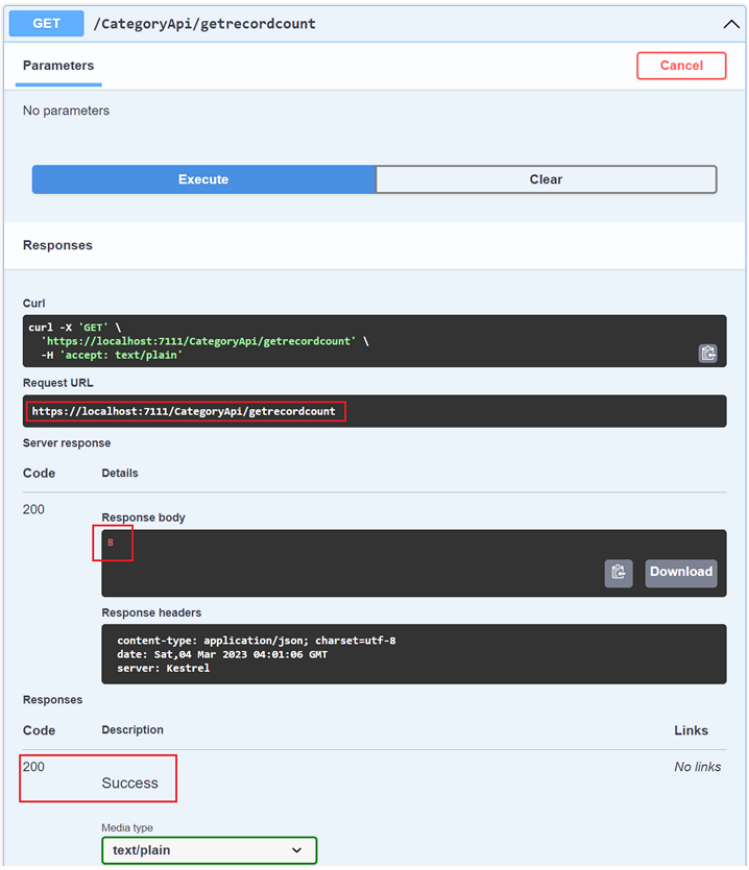
3.1 GET RECORD COUNT ENDPOINT

Click the ***/CategoryApi/getrecordcount***. And then click the *Try it out* button. And then click the *Execute* button.





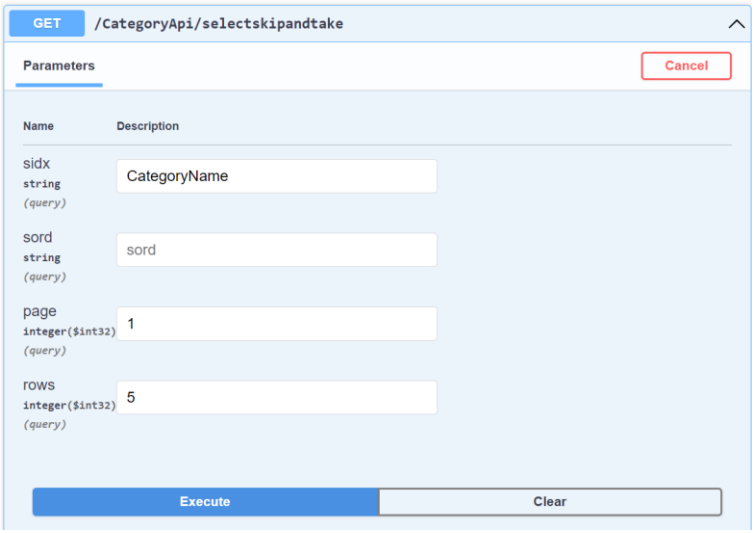
The result will show the *Request URL*, you can use this in your code to call this endpoint. Also shows the *Response Body* **8** (*getrecordcount* endpoint simply returns a number), which is the *total number of records* in the *Categories* database table. And the *Response Code*, *200* means *Success* (the web API call was successful).



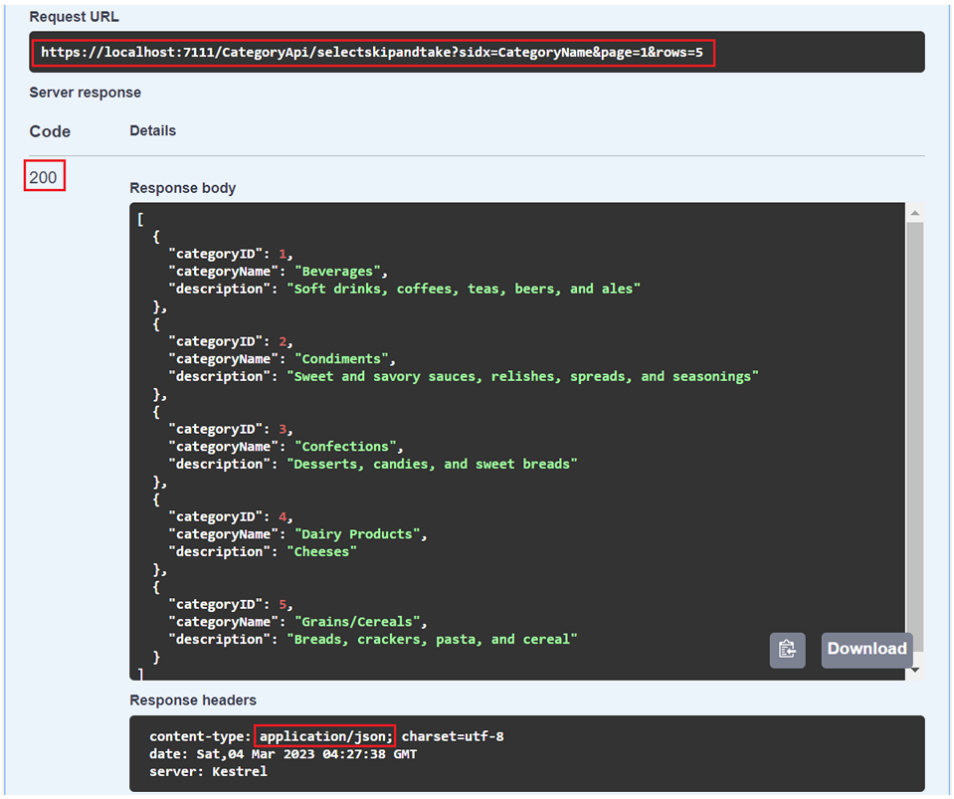
3.2 SELECT, SKIP, AND TAKE ENDPOINT

Now try the ***/CategoryApi/selectskipandtake*** endpoint. Click the *Try it out* button. This endpoint requires 4 parameters:

1. **sidx** – Field to sort.
2. **sord** – Sort order. ***asc*** or ***blank*** (nothing) for ascending order, and ***desc*** for descending order.
3. **rows** – Number of rows you want returned.
4. **page** – based on the number of rows you're requesting, there may be several pages to return, this is the *page number* you want returned. For example, if there are 37 records in the *Categories* database table, and the *rows* you requested is 5, then there will be 7 pages, you can request any number from 1 to 7.



The *Response* shows a *Code 200* (success), *5 records* returned (in descending order by *CategoryName*) in *json format*.

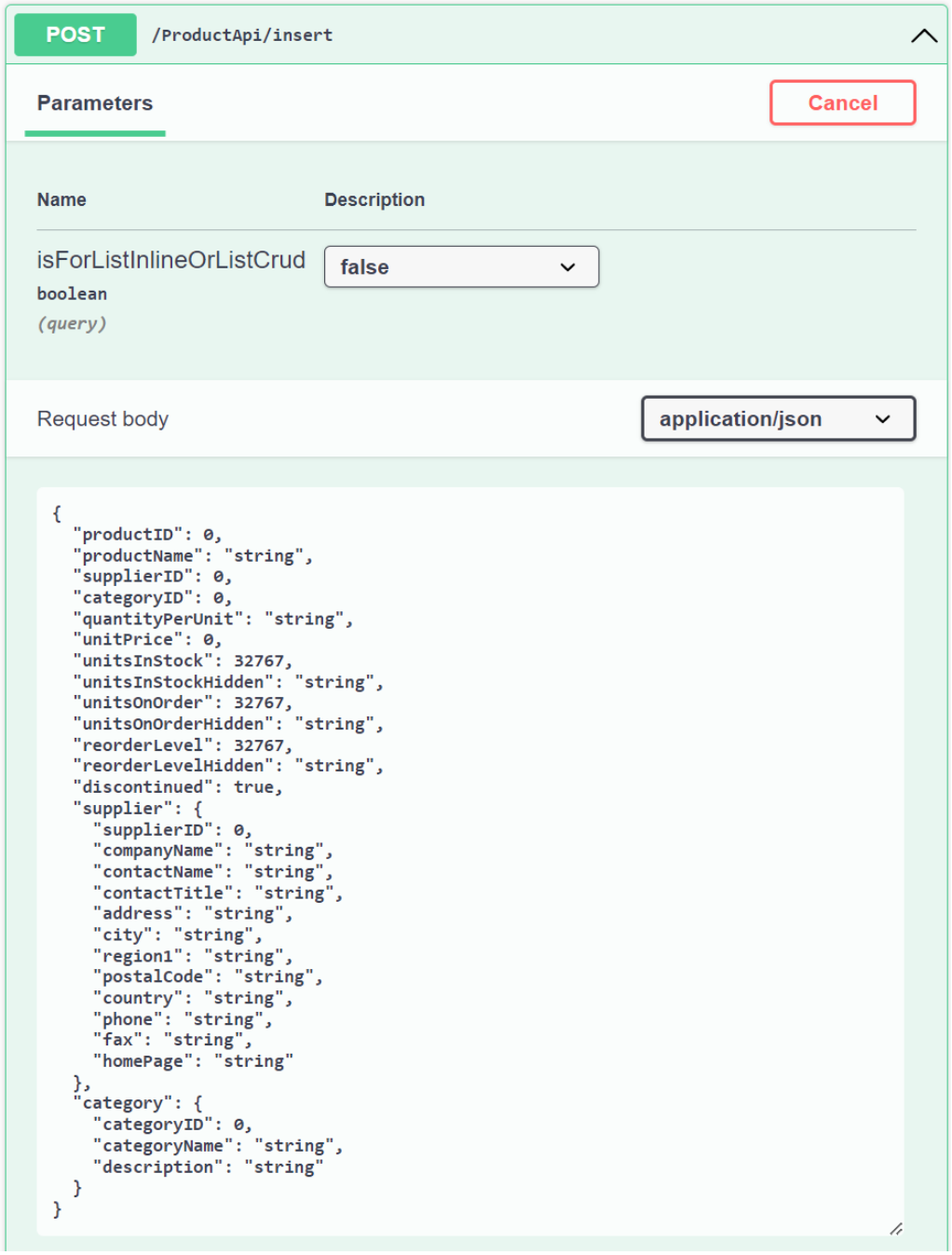


3.3 INSERT ENDPOINT

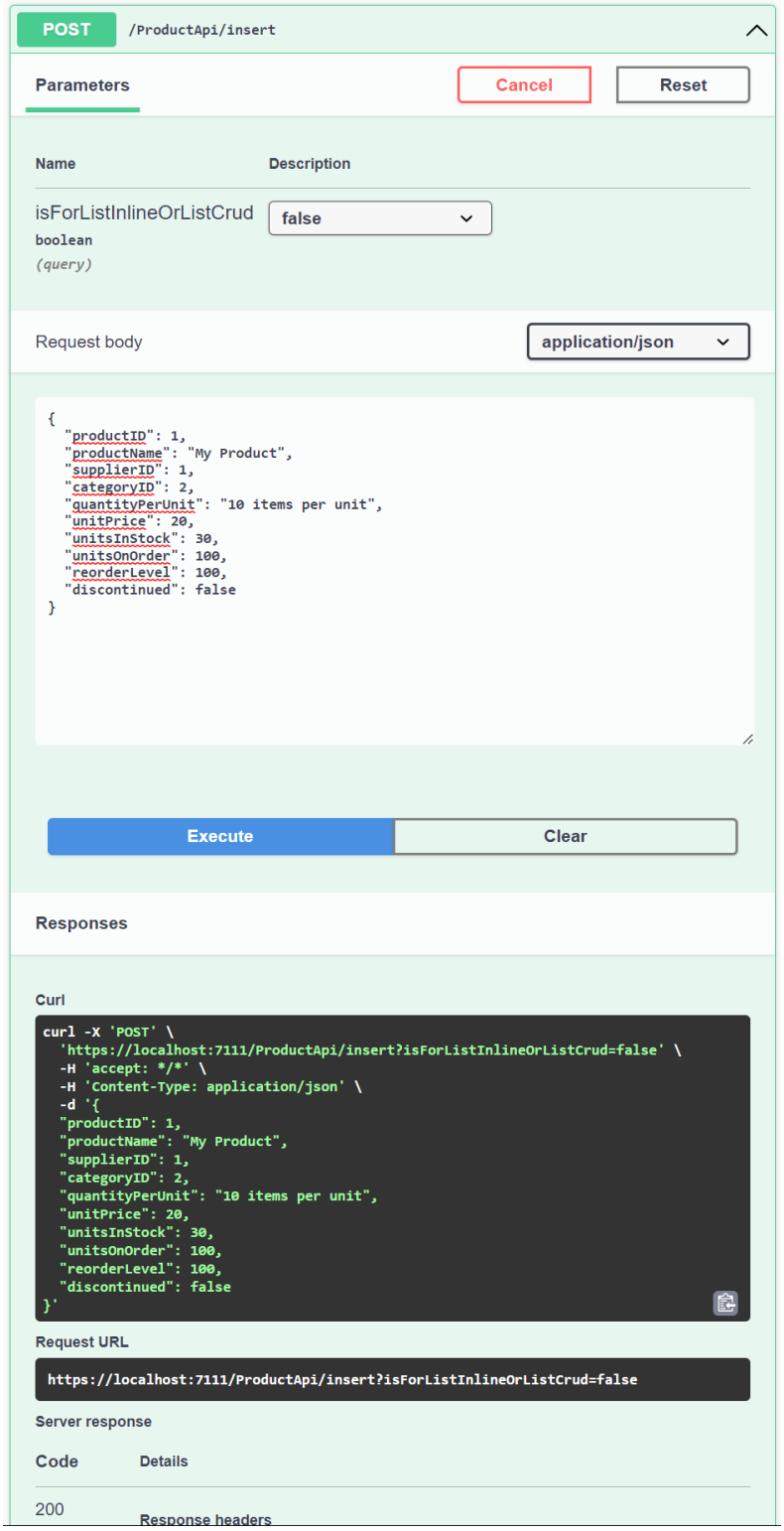
Under the *ProductAPI*, click the ***/ProductApi/insert*** endpoint. Click the *Try it out* button. This endpoint only has 1 parameter: *isForListInlineOrListCrud* parameter. This parameter is used in the web application, so we don't really care about this one for this example.

What's more important is the *Request body*. Because the operation is a POST, we need to pass the parameters via the *Request body*, and it's showing here that it's expecting a json formatted body. It's also showing the type of each parameter.

Note: for a more detailed information on the requirements for each of the parameters in the *Request Body*, please refer to the respective *Schema* of the *Web API* as shown in 2.2.



Like discussed in Page 4, under the *Web API Schema* section, some of these fields are not required. We need to remove some of the parameters that are not required and change the values we want inserted in the Products table, and then click *Execute*.



We got a *Server Response Code 200*, which means *Success*, this record has been inserted in the database.
Note: Although the *productID* was sent on this Web API post, it is disregard by the Data Repository code on the backend because it is an *Identity Field* on the database, which means the database will generate the *productID* on the fly everytime we add a new record on the *Products table* in the database.

You can read end-to-end tutorials on more subjects on using AspCoreGen 9.0 MVC Professional Plus that came with your purchase. These tutorials are available to customers and are included in a link on your invoice when you purchase AspCoreGen 9.0 MVC Professional. Download example shown here at: https://junnark.com/CustomProjectSamples/acg6mvc/StoredProcWa.zip

Note: Some features shown here are not available in the Express Edition. The code in this tutorial is available for download for paying customers only, please email us at Software Support for more information.

End of tutorial.