**Online Restaurant Ordering System**

**Group 15**

**UNC Charlotte**

**Restaurant Ordering System Technical Document - Group 15**

***Introduction:***

This API is designed to function as an online restaurant ordering system simulation using CRUD framework as a baseline. This is accomplished by using [FastAPI](https://fastapi.tiangolo.com/) docs to emulate the API calls in real time. Group 15’s [Python](https://www.python.org/downloads/) API setup is designed to connect with a [MySQL](https://www.mysql.com/) database via [SQLAlchemy](https://www.sqlalchemy.org/). It is set up to implement [PyTest](https://docs.pytest.org/en/stable/) as necessary to ensure proper execution of functions. Each object of the API is divided into four sections - model, schema, controller, and router. These four sections, and remaining sections, are described below:

* ***Model***: This is the way in which data is stored inside the database, ie. tables, rules, attributes, etc.
* ***Schema***: These pieces connect the database to the API by defining the correct structure for requests and responses.
* ***Controller***: Logical functions go here to manipulate data as needed by the object. This includes database interaction.
* ***Router***: This is the connection between the controller and the rest of the API. It defines endpoints from the CRUD operations.
* ***Database:*** *This folder houses the two initial queries used for database setup and population. One of these queries is the Test and Setup Query, used to reset the database. The other is the Populate Tables Query, used to populate the database with information for testing.*
* ***Dependencies:*** *Here is the location of the config.py and database.py files. Config.py is the details pertaining to the database credentials of the environment. Database.py connects to the database and ensures that interacts are able to transpire between the Python and MySQL environments.*
* ***Tests:*** *The PyTest files in this location are used to directly test API endpoints instantaneously, without the need for FastAPI docs requests.*
* ***main.py:*** *This is the main file in which the app is run from and FastAPI operates from.*

Using this combination of tools, the backend of the restaurant ordering system accomplishes user needs and creates implementations to streamline the restaurant processes.

**For User Setup, see User Guide at:**

**"Python\_Online\_Restaurant\_Ordering\_System\_Final\Product Documents\Documentation\Final Documentation - User Guide.docx"**

***High Abstraction Architecture:***

| **Component:** | **Architecture:** | **Use:** |
| --- | --- | --- |
| ***Frontend*** | * *HTTP Web Client* | *Interaction Interface for SImulation* |
| ***Backend API*** | * *FastAPI* * *Pydantic* | *Routing, Requests, Response, Input Validation, Modeling* |
| ***Database*** | * *MySQL Workbench 8.0* * *SQLAlchemy* * *PyMySQL* | *Connects Python to MySQL Tables, Connects to Database, Stores Data from Client* |
| ***Host*** | * *Localhost (port 8000 default)* | *Web Server Hosting Locally* |
| ***Version Control*** | * *Git* | *Controls Changes and Updates* |
| ***Local Deployment*** | * *Uvicorn* | *Seamless FastAPI-to-server synchronization* |
| ***Security*** | * *Cryptography* | *Password Hashing, Encryption* |
| ***Testing*** | * *Pytest* * *Pytest-mock* * *httpx* | *Unit Testing, Simulate HTTP Requests/Endpoints* |

***Data Flow Architecture:***

**[HTTP Web Client]**

**⬇**

**[FastAPI, Pydantic]**

**⬇**

**[Controllers/Logic]**

**⬇**

**[SQLAlchemy Models]**

**⬇**

**[PyMySQL Validation]**

**⬇**

**[MySQL Database]**

**⬇**

**[DataBase Response]**

**⬇**

**[API Response]**

***Testing Data Flow Architecture:***

**[PyTest Function]**

**⬇**

**[PyTest-mock]**

**⬇**

**[httpx Endpoint]**

**⬇**

**[FastAPI Response]**

***Local Deployment:***

**uvicorn main:app –reload**

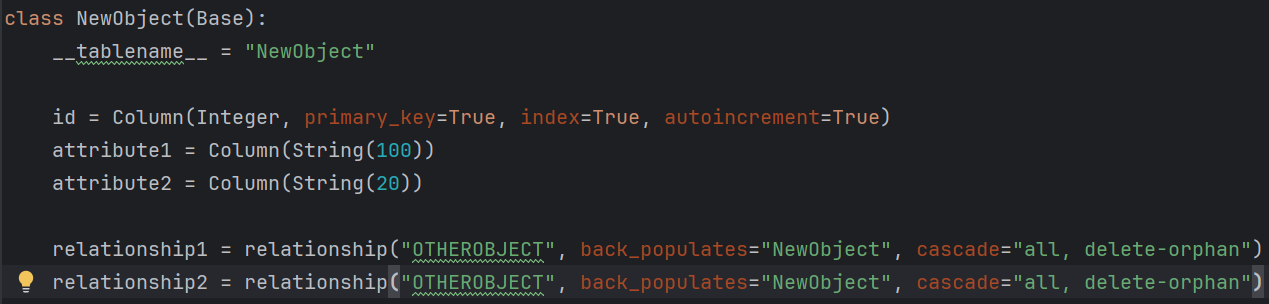
***Development Environment and New Object Creation:***

After completing the User Guide Setup, The Development Environment is entirely set up as well. It is important to note that Pycharm v.2024.3.1.1 or higher is required for development.

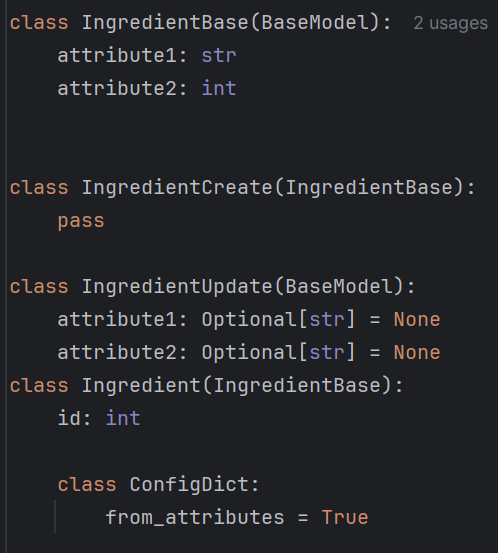
To begin creation of a new object in the database, do the following;

1. Create a new Model, Schema, Controller, and Router file as the new object’s name
2. Beginning with the Model, reference the existing models to create a new object with attributes, and table(\_\_tablename\_\_=) and relationships if necessary. Ensure the correct imports from SQLAlchemy, and dependencies.database are present.

**Model NewObject.py:**



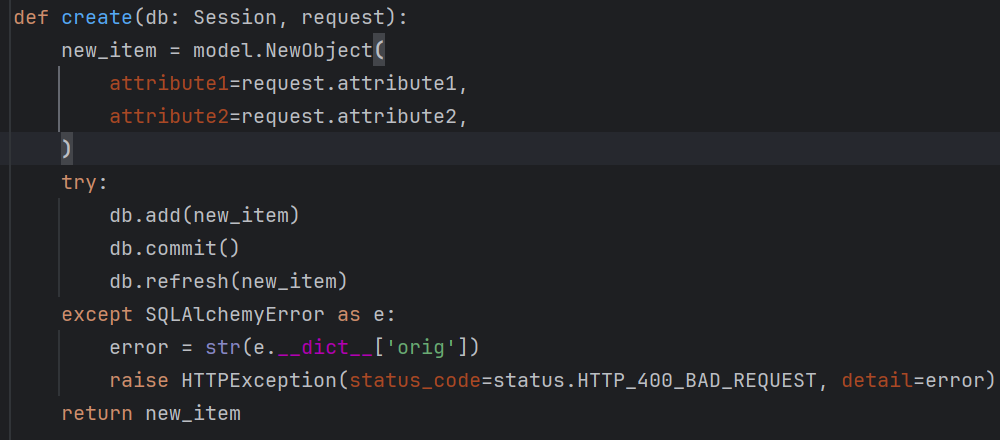
1. Moving to the Schema, reference the existing schemas to create a new object schema. This will require four classes: NewObjectBase, NewObjectCreate, NewObjectUpdate, and NewObject. This is the structure that passed information should be formatted to prior to its addition to the database. Be sure to import the relationship object if necessary and the Pydantic BaseModel.

**Schema NewObject.py**

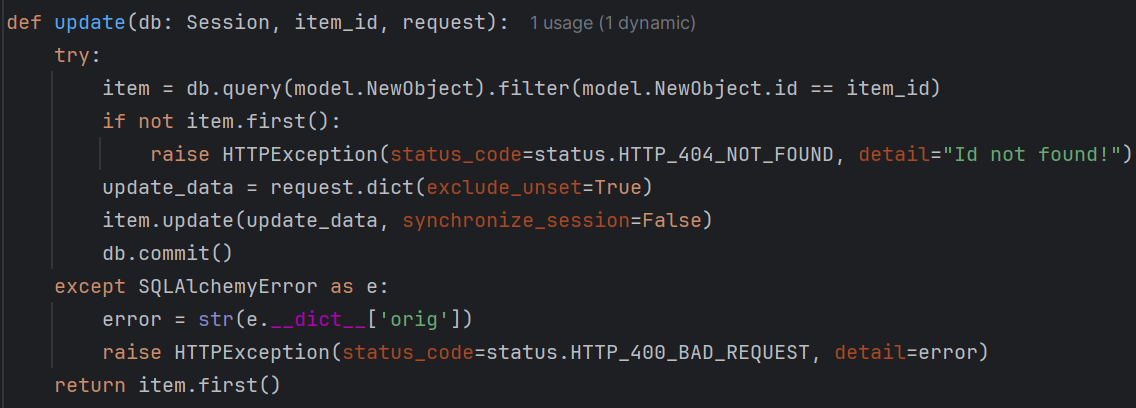
1. For the Controller file, reference the given controllers to formulate one for the new object. Ensure to import the referenced model, SQLAlchemy, and FastAPI necessities. Also ensure to use try, except statements to catch API request eros when necessary. This project uses CRUD API implementation as defined below:

**Controller NewObject.py:**

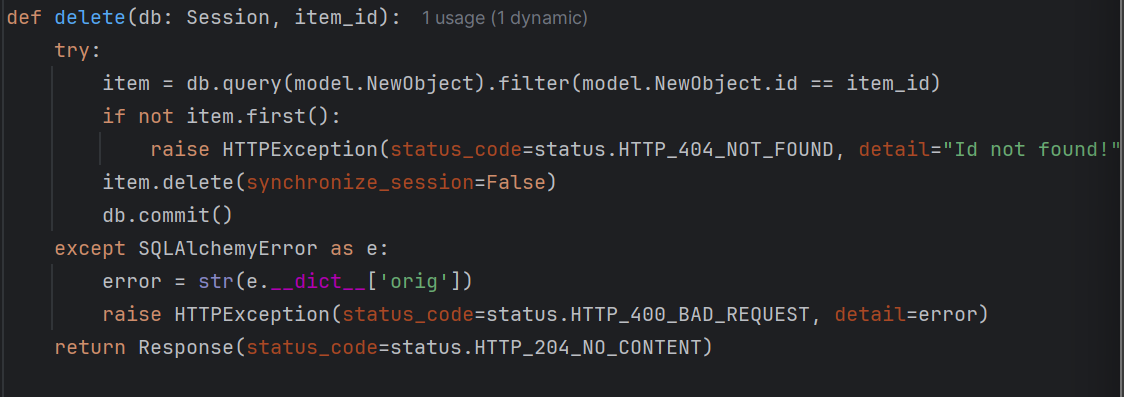
• Create: Function creates new data in the table.



• Update: Function handles updates to existing data.

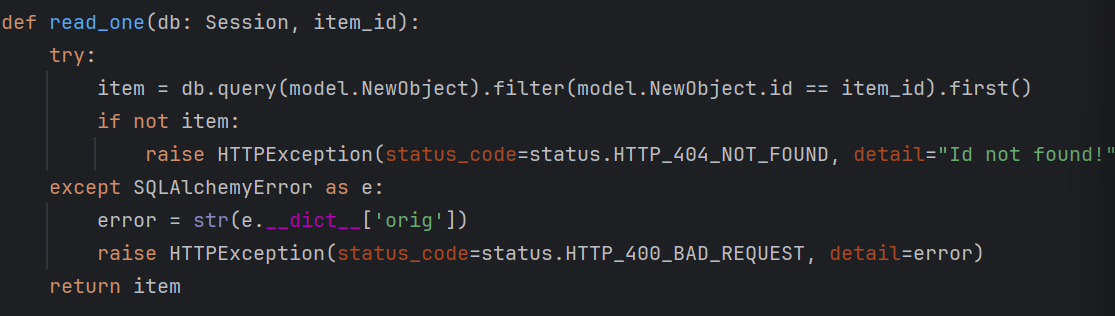


• Delete: Function to delete data from the table.



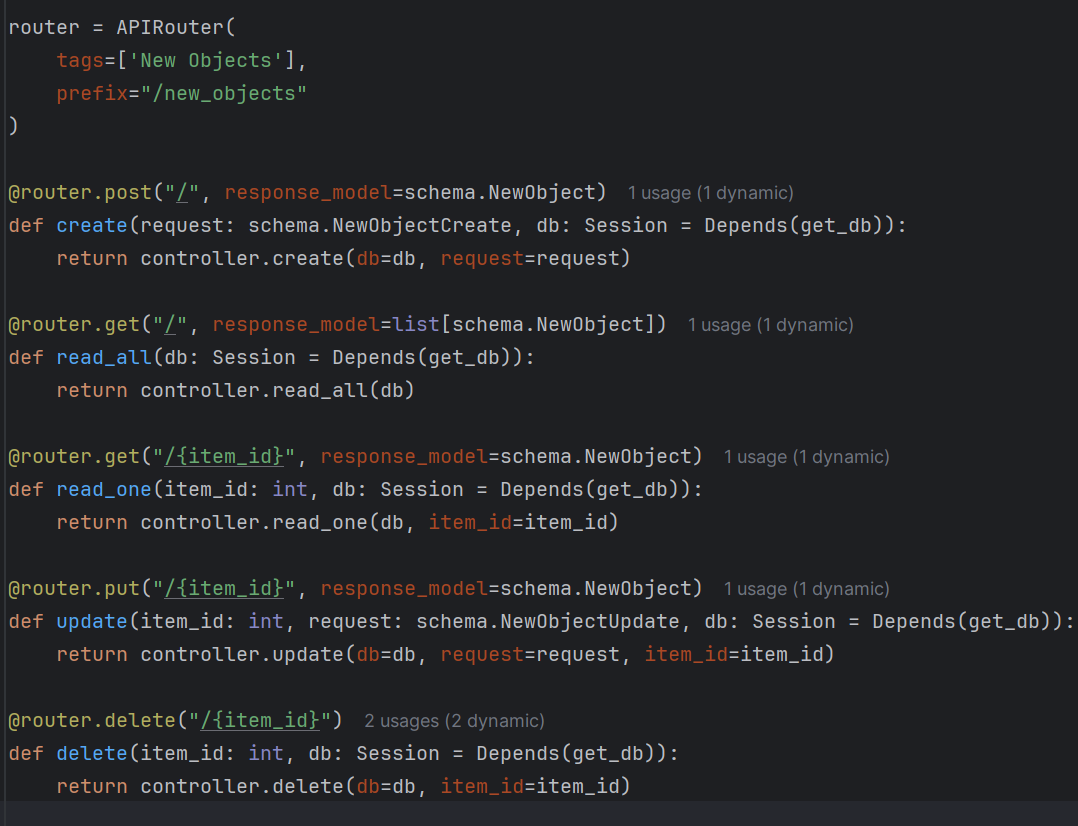
• Read\_all: Function to retrieve all rows in table.



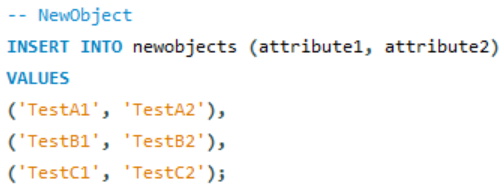
• Read\_one: Function to retrieve a specific data by its id.

1. The routers files are very similar for all objects and can be nearly copied with the adjustment to the NewObject naming conventions.

**Router NewObject.py:**

****

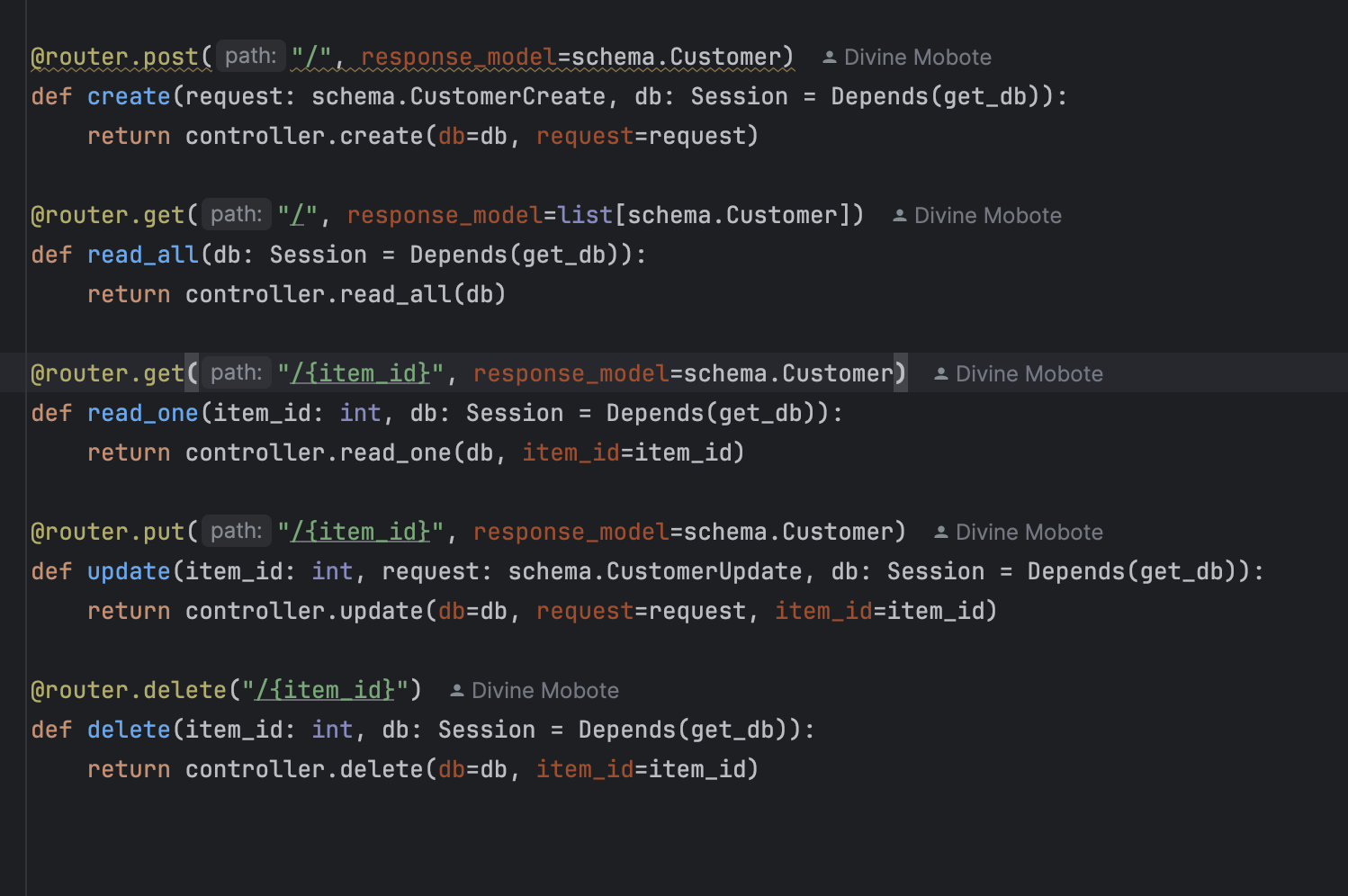
1. Now, in models/model\_loader.py, import the new model and add the NewObject.Base.metdata.create\_all(engine) line in the def index():
2. In routers/index.py, include the new import router and include\_router line for the NewObject.
3. Finally, in the Populate\_Table Database in MySQL, create a table to correspond with the new object and add preliminary data entries if necessary.



***Endpoints:***

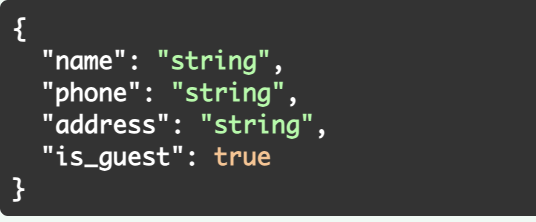
***All Objects have create, read, read all, update and delete endpoints. Examples for the customer object are shown below. Objects with different endpoints are also shown.***

**Customers Endpoints:**

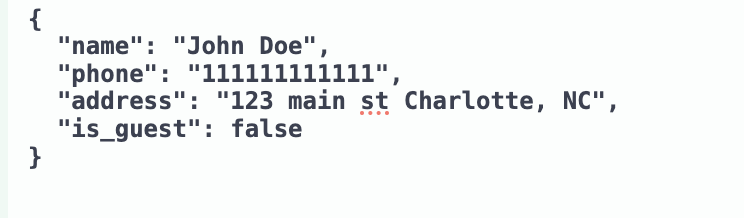
****

**Post:/customers:** Creates a new customer record in the system.

* Parameters: Customer Schema



* Sample Request:



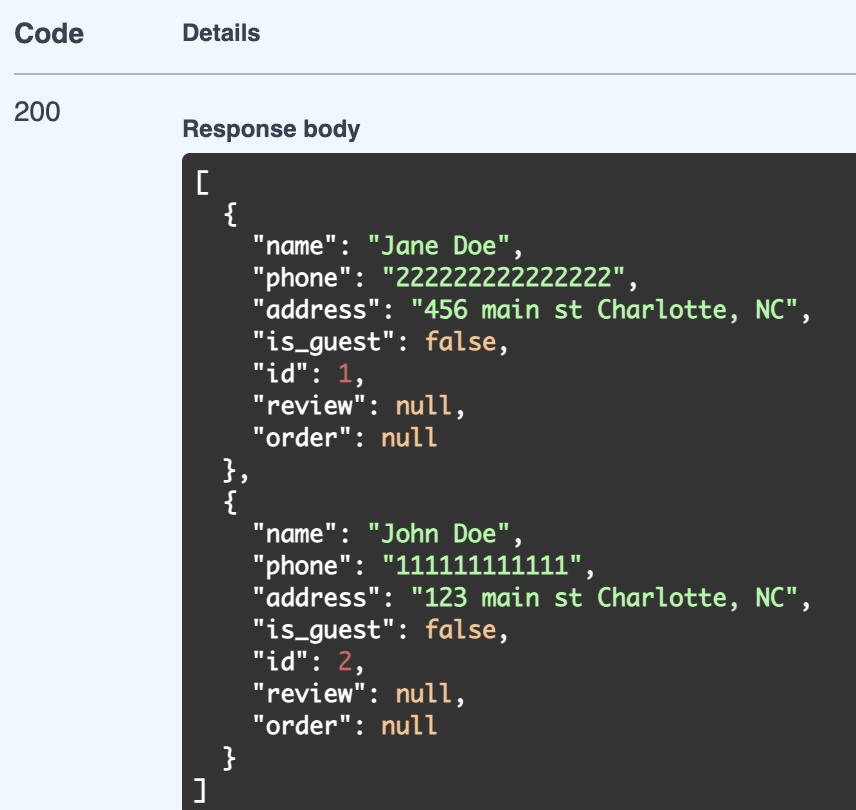
* Expected Response:
  + **200 OK:** Returns newly created Customer object



* + **400 Bad Request:** Invalid input / missing information

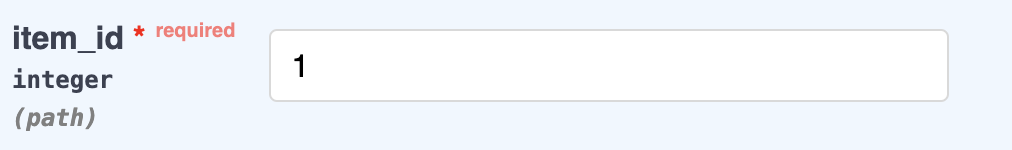
**Get:/customers:** Retrieves a list of all customers.

* Parameters: None
* Expected Response:
  + **200 OK:** Returns an array of customer objects

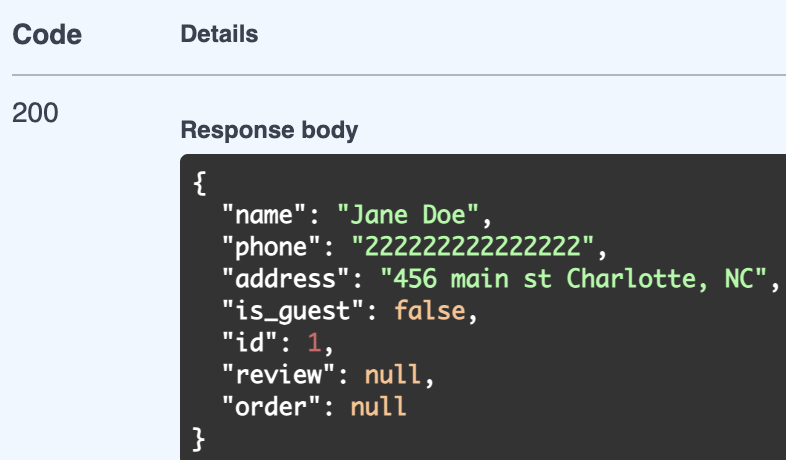


**Get:/customers/{item\_id}:** Retrieves details of a specific customer.

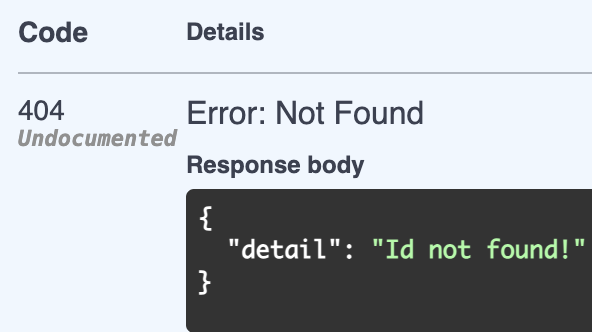
* Parameters: customer\_id:integer
* Sample request:



* Expected Response:
  + **200 OK:** Returns specific customer object



* + **404 Not Found:** ID not found

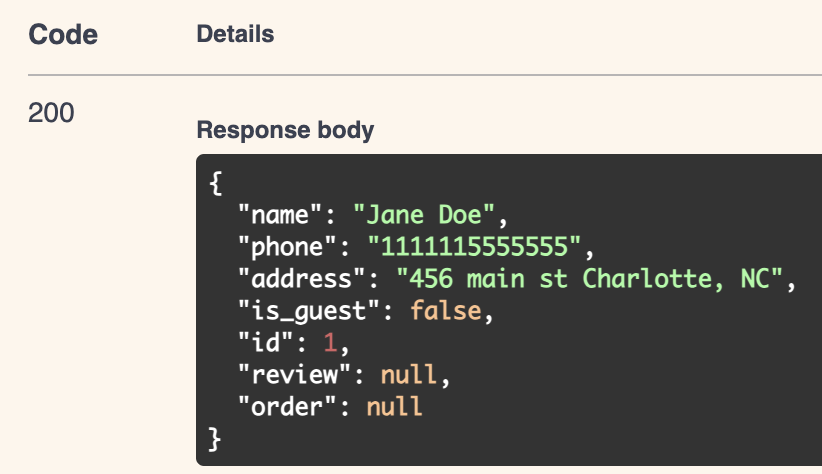


**Put:/customers:** Updates an existing customer's information.

* Parameters: customer\_id:integer, customer schema:all fields are optional
* Example request:



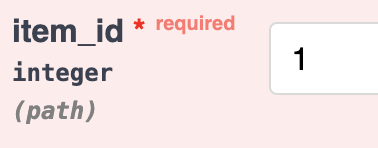
* Expected Response:
  + **200 OK:** Returns updated customer object



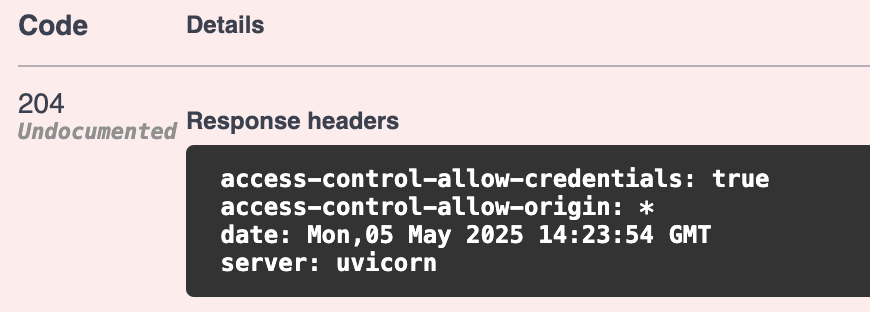
* + **401 Not Found**

**Delete:/customer:** Removes a customer record from the system.

* Parameters: customer\_id:integer
* Example Request:



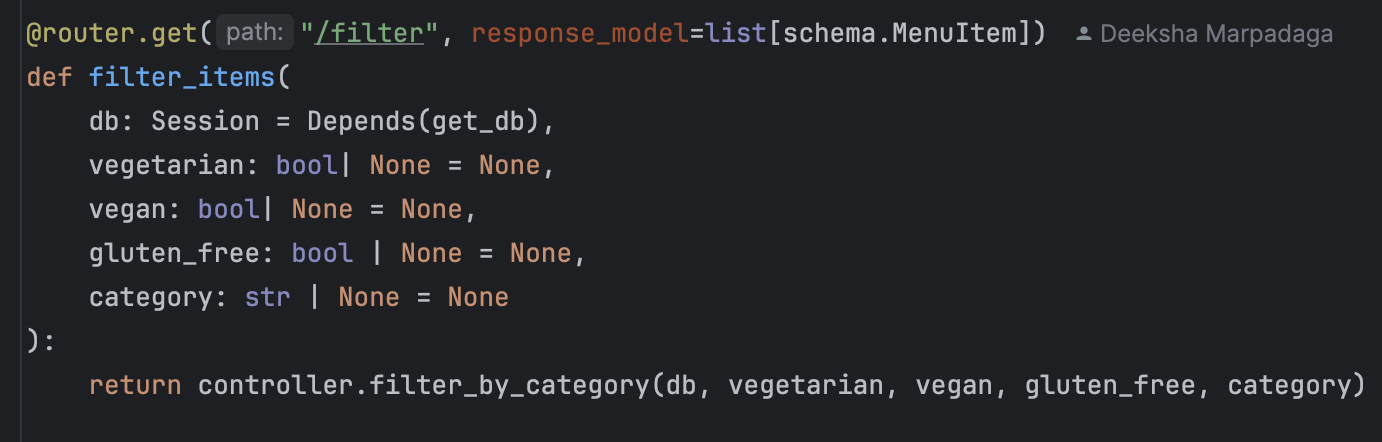
* Expected Response:
  + **204 Successful Deletion:** (no body returned)



* + **404 Not found:** ID not found

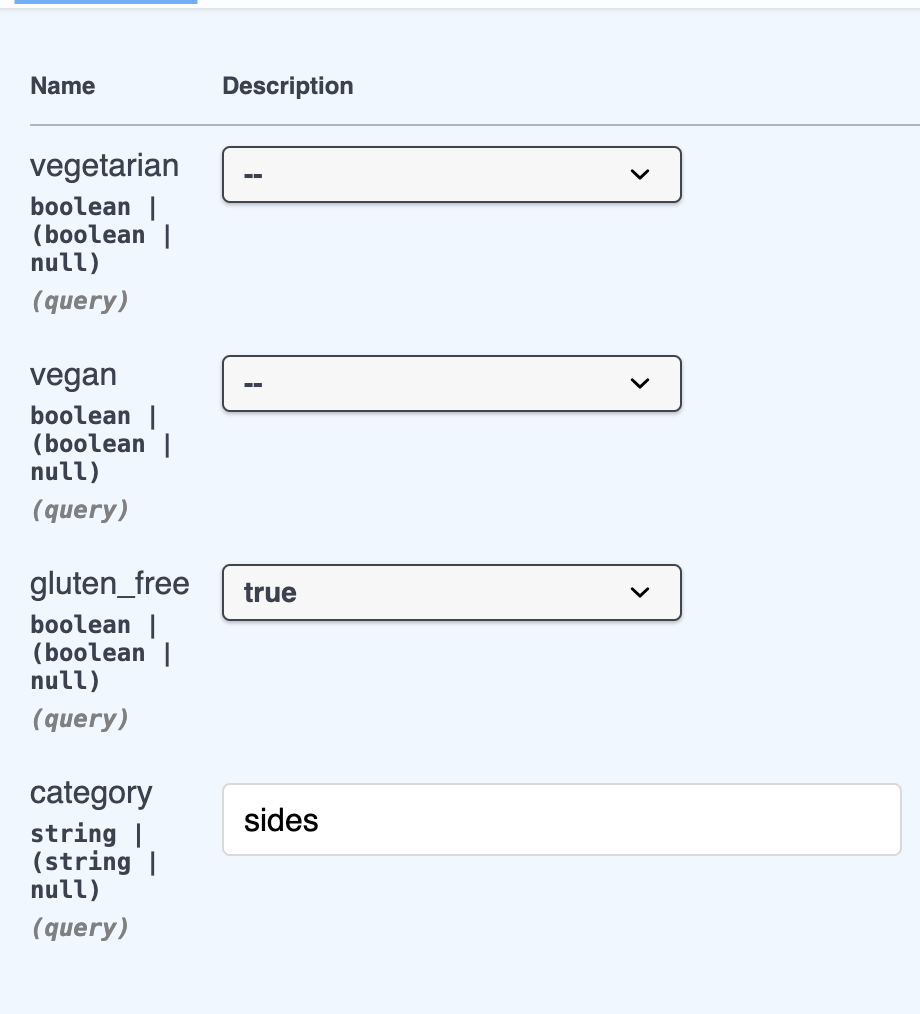


**Menu Item Endpoint:**

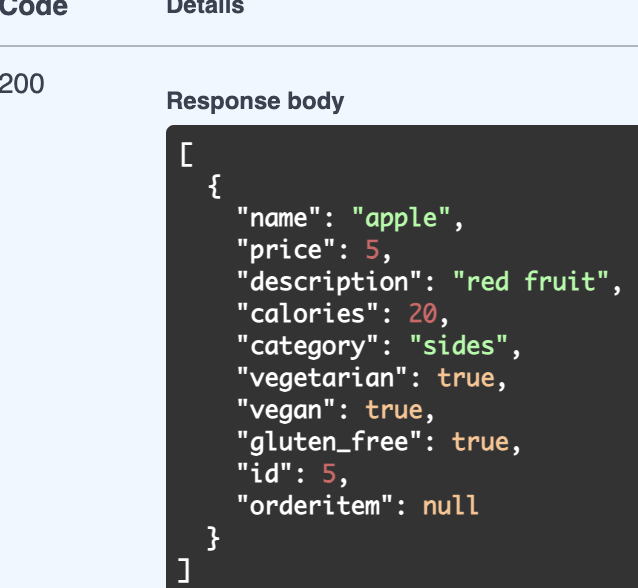


**Get:/menu\_items/filter:** Retrieves menu items filtered by dietary preferences and category.

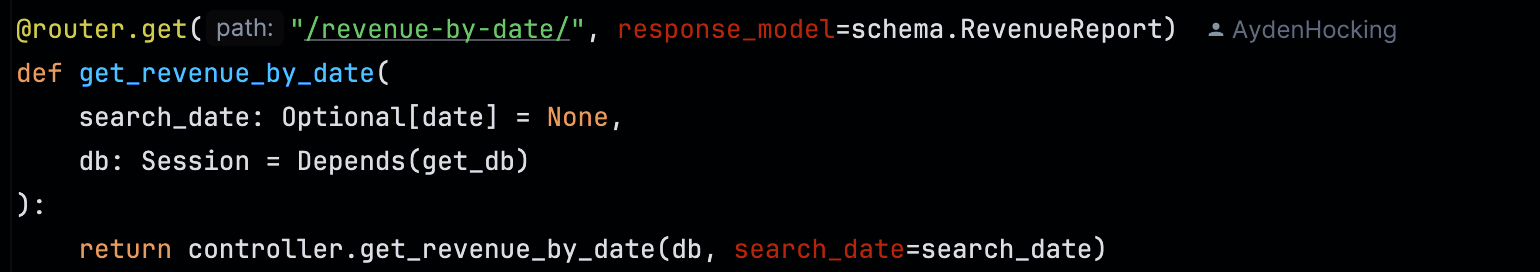
* Parameters: vegetarian:boolean, vegan:boolean, gluten-free:boolean, category:str
* Example request:



* Expected Response:
  + **200 OK:** Returns filtered menu items

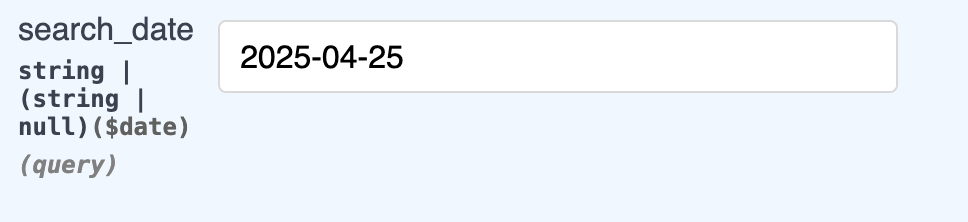


**Payment Item Endpoint:**

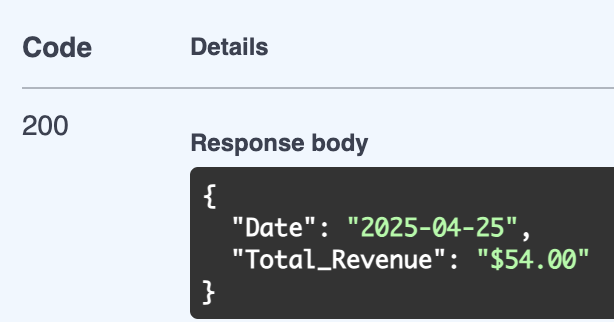


**Get:/payment/revenue-by-date:** Retrieve the total amount of payment by date.

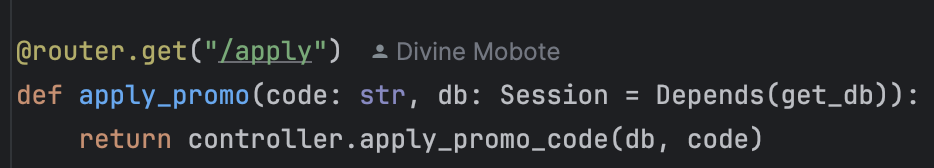
* Parameters: search\_date:date(YYYY-MM-DD)
* Example request:



* Expected Response:
  + **200 OK:** Returns sum of payments for date

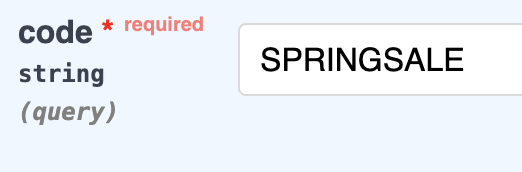


**Promo Endpoint:**



**Get:/promo/apply:** Checks if promo code is valid

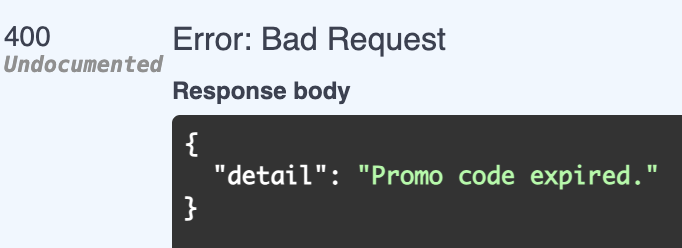
* Parameters: code:str
* Example request:



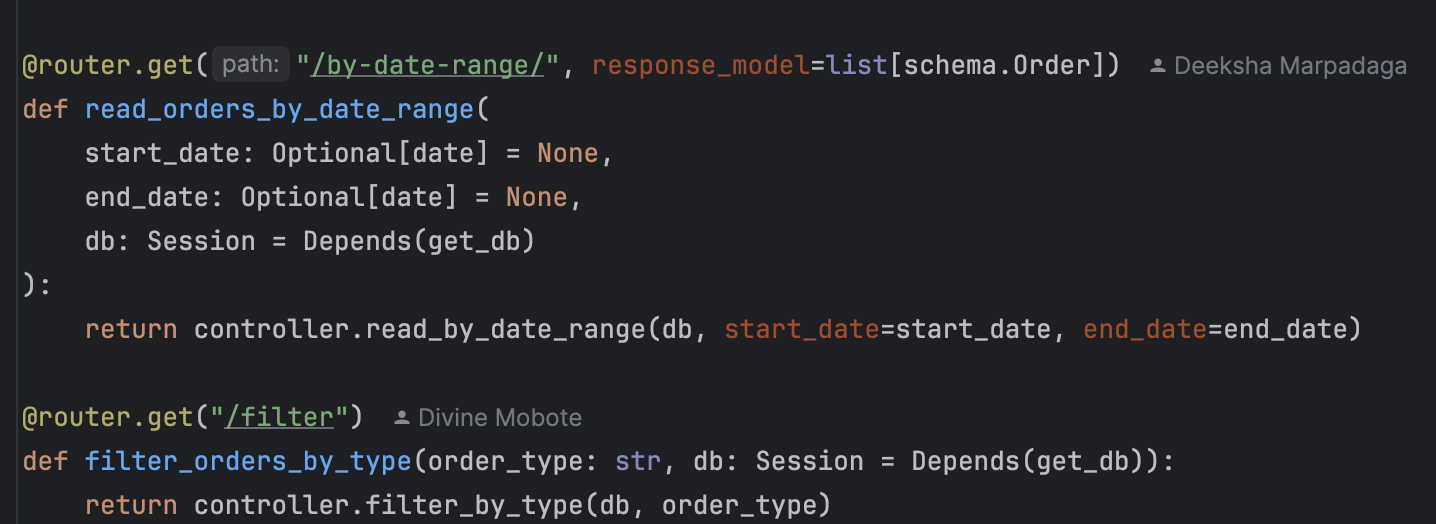
* Expected response:
  + **200 OK:** Returns promo with code



* + **400 Bad Request:** Not found/Expired

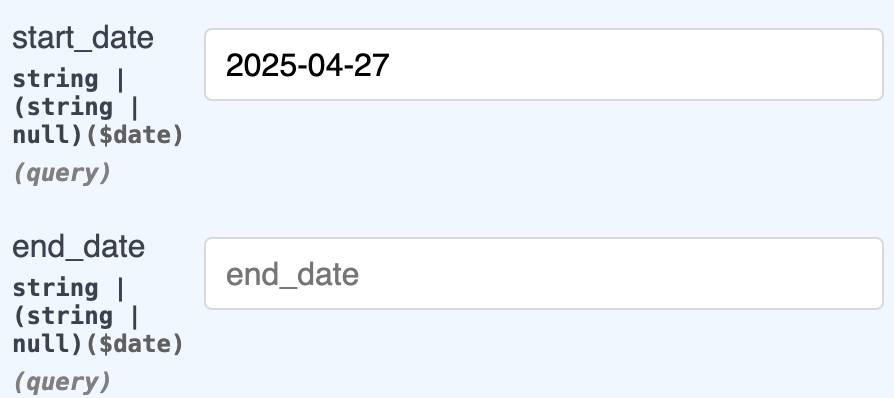


**Orders Endpoints:**

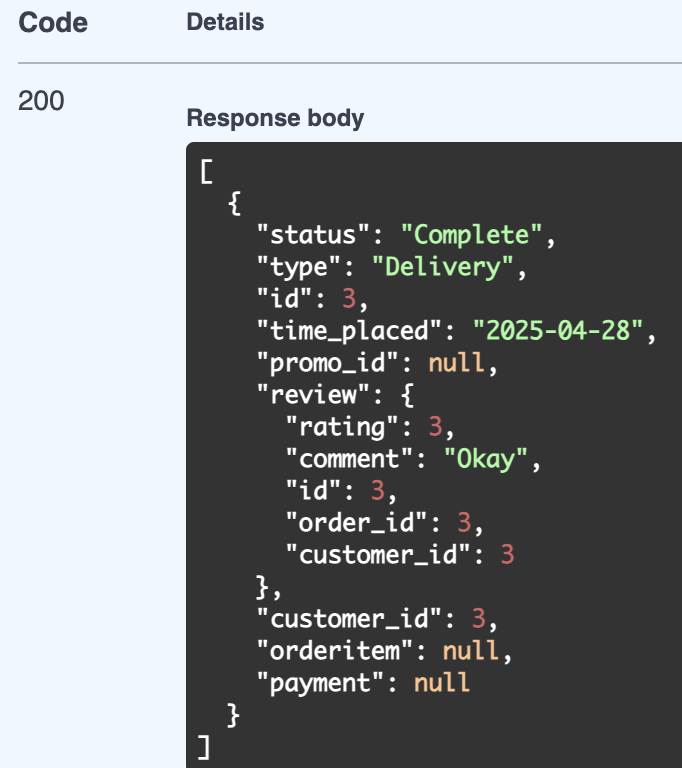


**Get:/orders/by-date-range:** retrieves orders filtered by date placed.

* Parameters: start date:date(YYYY-MM-DD), end date:date(YYYY-MM-DD), Optional
* Example request:

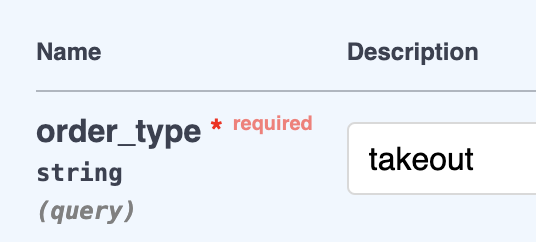


* Expected Response:
  + **200 OK:** Returns orders in date range

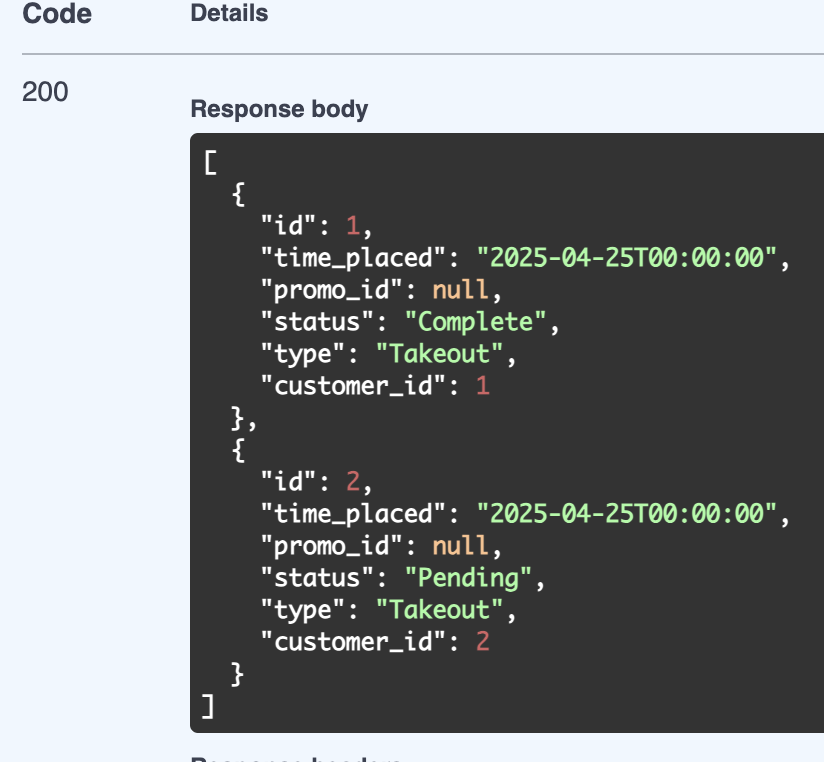


**Get:/orders/filter:** Retrieves orders filtered by order type.

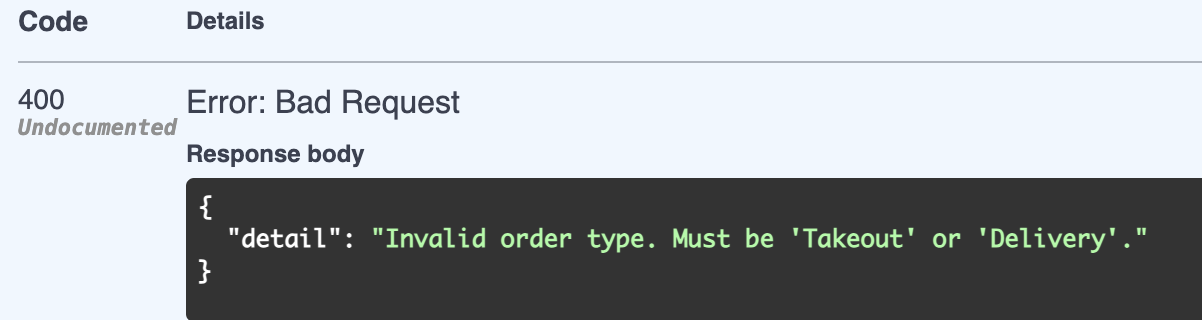
* Parameters: order\_type:str
* Example request:



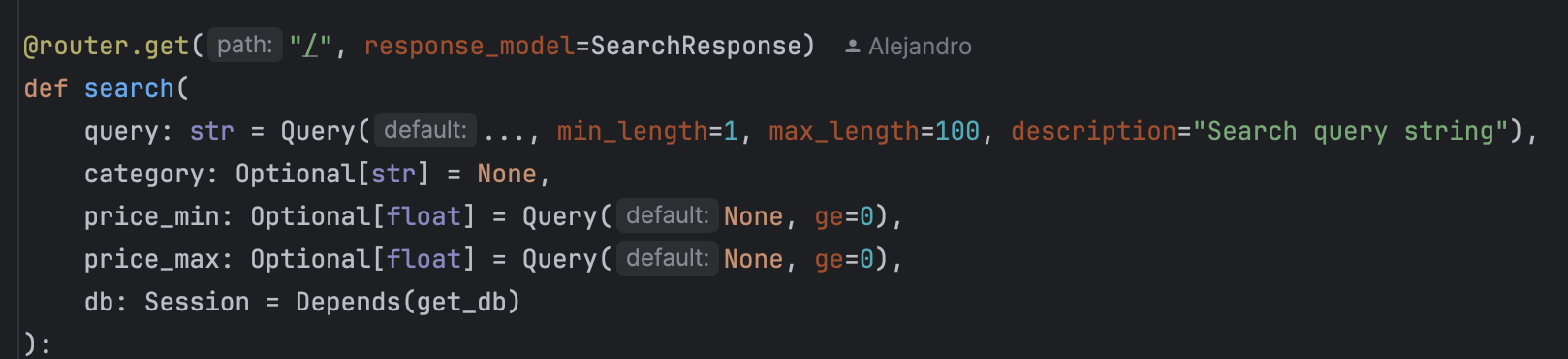
* Expected response:
  + **200 OK:** Returns list of orders with filters



* + **400 Bad Request:** Invalid order type



**Search Endpoint:**

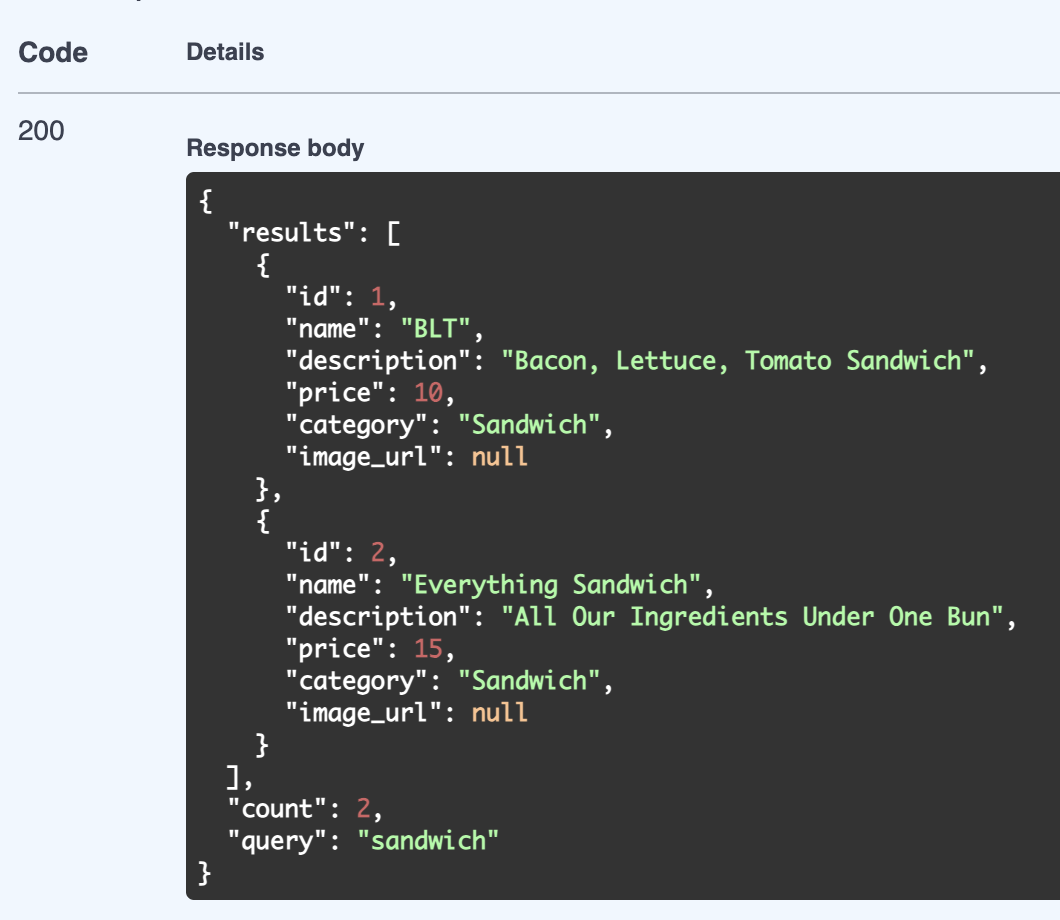


**Get:/search:** Retrieve menu items filtered by price, category and keywords

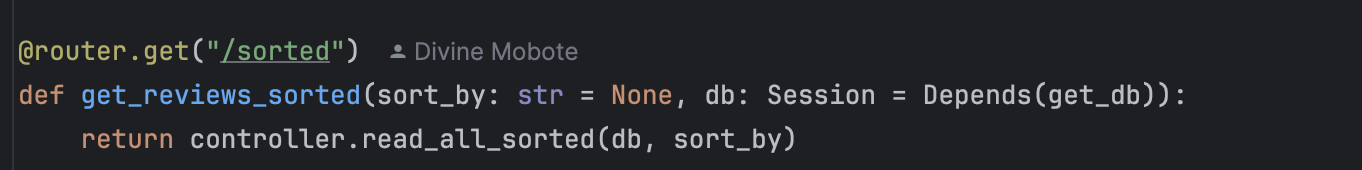
* Parameters: query:str, price\_min:float, price\_max:float, category:str
* Example Request:



* Example response:
  + **200 OK:** Return a list of filtered menu\_item objects



**Review Endpoint:**

****

**Get:/reviews/sorted:** Retrieve review sorted by rating or date

* Parameter: sort\_by:str(date or rating)
* Example request:



* Expected response:
  + **200 OK:** list of review object sorted