User Manual

Online Restaurant Ordering System (OROS) API

Group 8
May 6, 2025

Table of Contents

Purpose

This User Manual is designed to help end-users understand how to interact with the Online Restaurant Ordering System (OROS) API effectively.

Overview

Online Restaurant Ordering System (OROS)

The Online Restaurant Ordering System (OROS) API enables the customer's restaurant to offer a seamless and user-friendly online food ordering experience. Designed to boost revenue and enhance customer satisfaction, this API supports accountless ordering, dynamic menu browsing, real-time order tracking, and multiple payment options. It also streamlines operations through role-based access for staff, chefs, and customers, while providing tools for promotions, feedback collection, and data-driven decision-making.

Setup Instructions

Required Tools and Technologies

To begin working with the project, ensure that you have the following tools installed:

- **Python 3.10+**
- **MySQL** (ensure you have your password and access credentials ready)
- **FastAPI** (for building APIs)
- **PyCharm** (recommended IDE, but any terminal or integrated development environment will work)
- **GitHub Integration** in your IDE (to pull/push code from the repository)
- **Git** (for version control and cloning the repository)

Setup Instructions (Also referenced in the Technical Documentation)

1. **Clone the Github repository**

   Start by cloning the project repository to your local machine.

```
Sophia@DESKTOP-O5E9T11 MINGW64 ~/IntroToSoftwareEng
$ git clone https://github.com/LiesAndDeception/ITSC3155_051_GroupProject
Cloning into 'ITSC3155_051_GroupProject'...
remote: Enumerating objects: 267, done.
remote: Counting objects: 100% (267/267), done.
remote: Compressing objects: 100% (199/199), done.
remote: Total 267 (delta 177), reused 123 (delta 66), pack-reused 0 (from 0)
Receiving objects: 100% (267/267), 44.41 KiB | 1.85 MiB/s, done.
Resolving deltas: 100% (177/177), done.

Sophia@DESKTOP-O5E9T11 MINGW64 ~/IntroToSoftwareEng
$ ls
ITSC3155_051_GroupProject/

Sophia@DESKTOP-O5E9T11 MINGW64 ~/IntroToSoftwareEng
$ cd ITSC3155_051_GroupProject/
```

   You can access the GitHub repository here: [LiesAndDeception/ITSC3155_051_GroupProject: Group Project Repository for ITSC3155-051]

2. **Install Python Dependencies**

   Install the required Python packages using pip:

```
pip install fastapi
pip install "uvicorn[standard]"
pip install sqlalchemy
pip install pymysql
pip install pytest
pip install pytest-mock
pip install httpx
pip install cryptography
```

You may also choose to store and install dependencies via a requirements.txt file for ease of use:

```
pip install -r requirements.txt
```

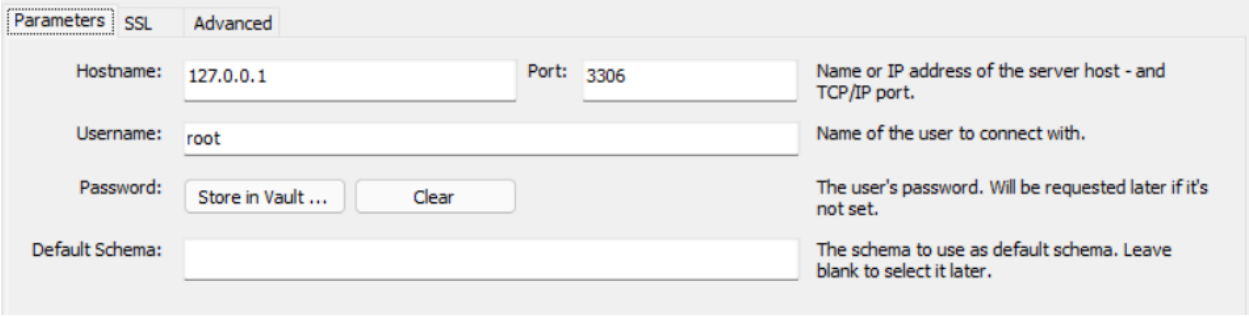
3. **Configure MySQL**

   In api\dependencies\config.py, replace “db_name” with a chosen name for the api (i.e. “restaurant_system_api”), and replace “db_password” with the password for your MySQL Workbench local instance connection.

   api\dependencies\config.py with a placeholder password and “sandwich_maker_api” as a database name:

```
class conf:  9 usages  Peter *
    db_host = "localhost"
    db_name = "sandwich_maker_api"
    db_port = 3306
    db_user = "root"
    db_password = "rootroot"
    app_host = "localhost"
    app_port = 8000
```

   Other “db_xxx” variable values should reflect the MySQL local connection information, such as the instance below.



4. **Run the Development Server**

   Use Uvicorn to start the FastAPI server with hot reloading enabled:

```
uvicorn api.main:app --reload
```

5. **Access the API Documentation**

Once the server is running, navigate to the following URL in your browser to interact with the API via the built-in documentation interface:

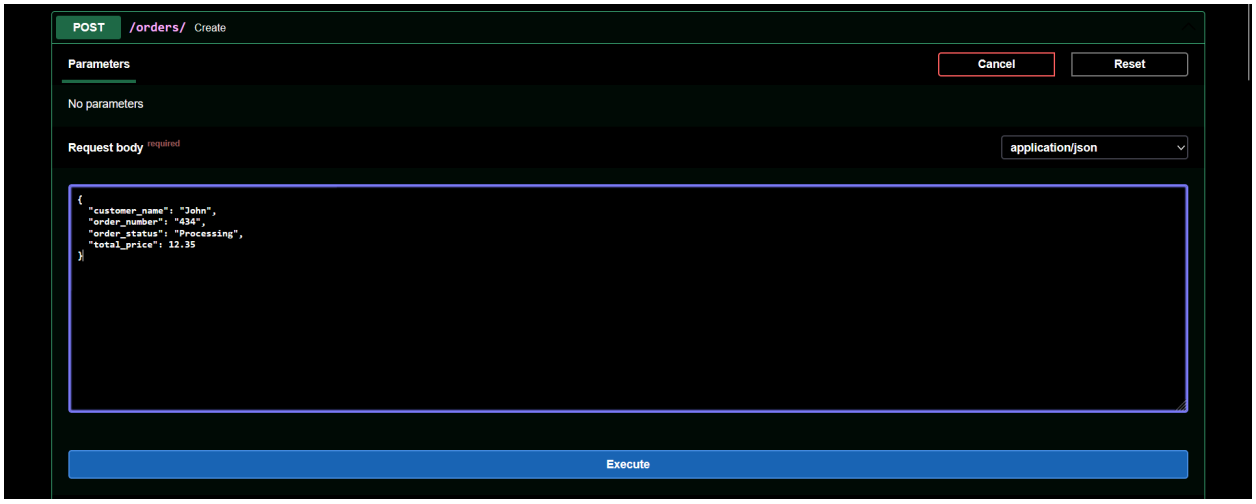
http://127.0.0.1:8000/docs

This interface allows you to test API endpoints and view schema definitions directly from the browser.

Usage Examples

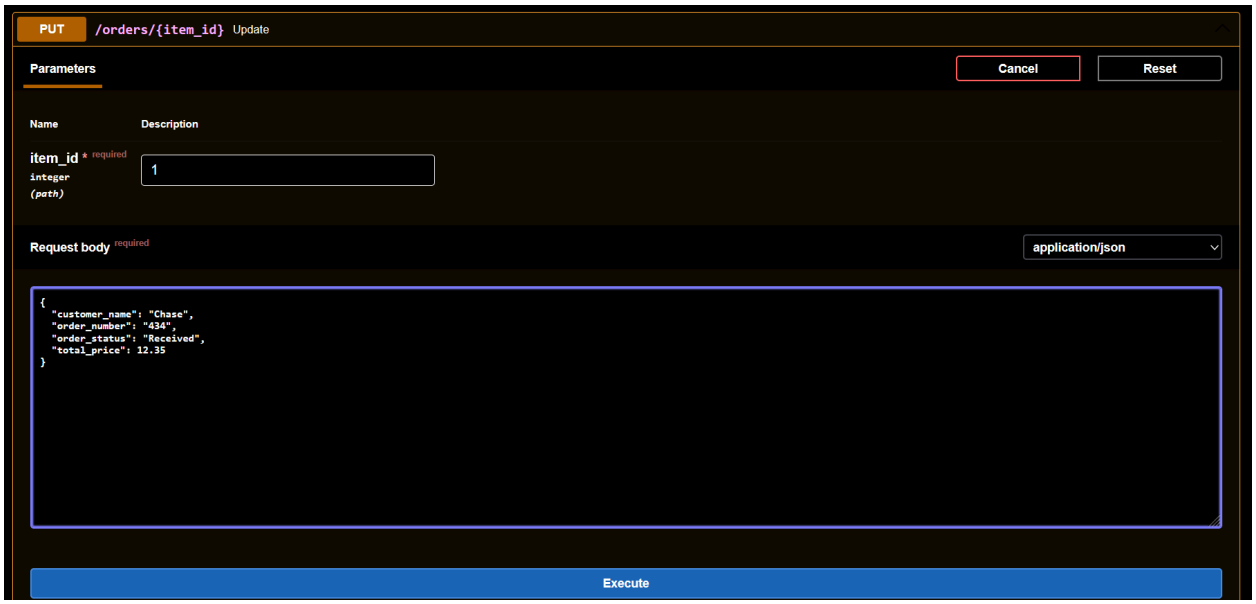
1. **Create and Update an Order**

Under the “POST /orders/ Create” endpoint, replace “string” in “customer_name” and “order_number” with the related information. Replace 0 with the price for the order:



Now, click Execute, and the order should be added to the database. You can check that the order posted by using any of the GET buttons:

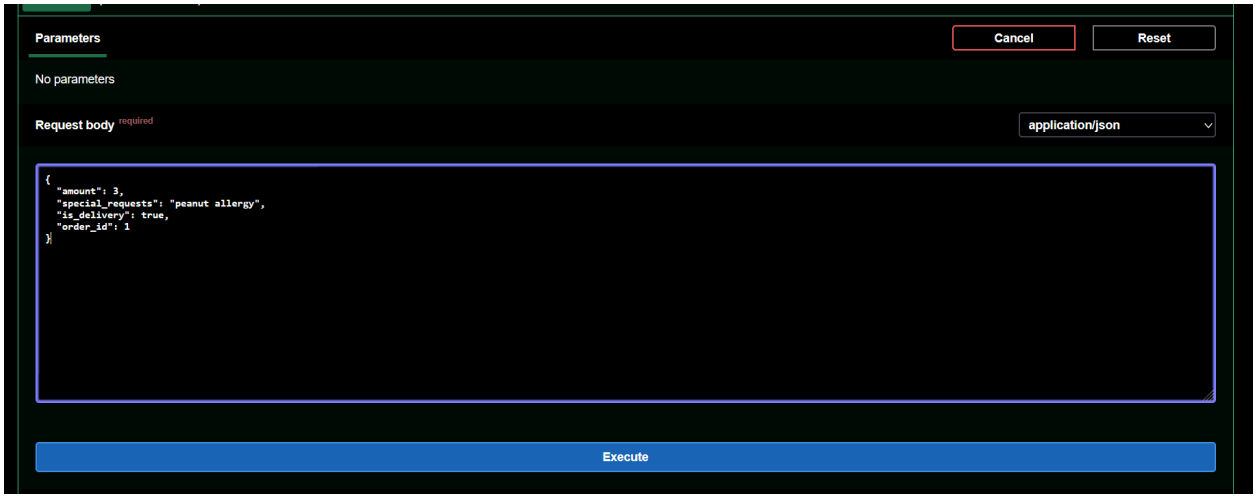
To update an order using the “PUT /orders/ Update” endpoint, use the order’s “id” number [circled in the above image] and fill in the updated information. Then, click Execute:



Updated order viewed in the database:

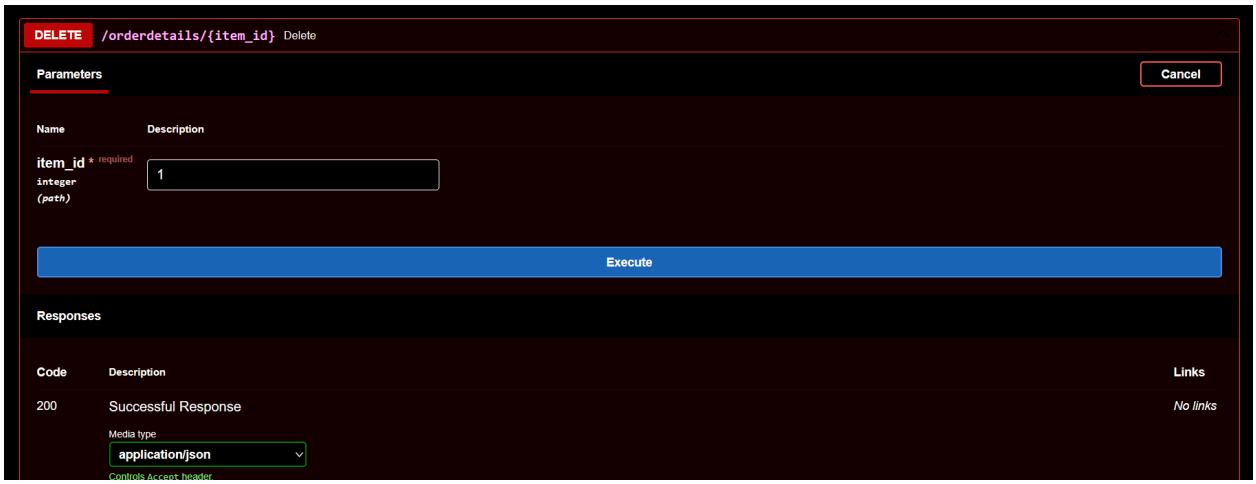
2. Add Details to an Order (Order Details)

After an order has been created with the Order endpoint, you can add more information using Order Details. Under the "*POST /orders/ Create*" endpoint, fill out the required information and replace the number next to "*order_id*" with the "*id*" number from the order these details will be attached to:



3. Delete an Order

When deleting an order, first delete the details using the "*DELETE /orderdetails/{item_id} Delete"* endpoint using the id number for the order. Then, do the same thing with the "*DELETE /orders/{item_id} Delete*" endpoint.



View endpoint showing that the order was removed (thus, an empty database):


GET
/orders/read_all_by_date
Read All By Date
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/orders/read_all_by_date' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/orders/read_all_by_date
Server response
Code
Details
200
Response body
[]
Download