RESTful Diner

A 5-stars Open Data repository to handle restaurants order data

Enrico Stefanel [SM3500554]
enrico.stefanel@studenti.units.it

Data Science and Scientific Computing Master's Course
Department of Mathematics and Geosciences
University of Trieste

A.Y. 2024–2025

Contents

A brief overview of what we are about to see:

The Problem

In a far away land, there is a restaurant that is serving every day thousands of hungry customers. The restaurant owner needs a system to keep track of all the orders that are coming in, and that can also help him to analyze the data in order to improve the service that he is offering to his customers.
The restaurant owner understands that the data that he is collecting is very valuable, and **he wants to make it available to the public**[1], so that other willing Data Scientists can help him to improve his business.

[1] The data will have to be Findable, Accessible, Interoperable and Reusable. Does this sound familiar?

Envisioned Solution

The solution of the problem is to design a system that can handle the restaurant orders data, and that can also provide an API to access the data in a simple and efficient way.
In a few words, an **Open Data API system** is exactly what the restaurant owner needs.

Use Case Diagram

The first step of our development to identify the use cases of our system. We have identified four main use cases:

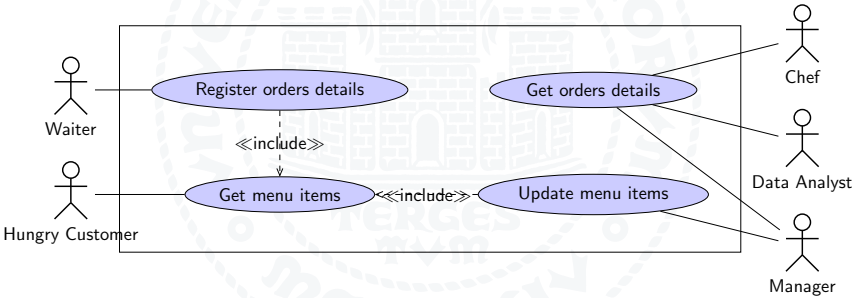

Figure: Use cases diagram

E/R Diagram

Next, the E/R diagram was designed to represent the data model of the system:

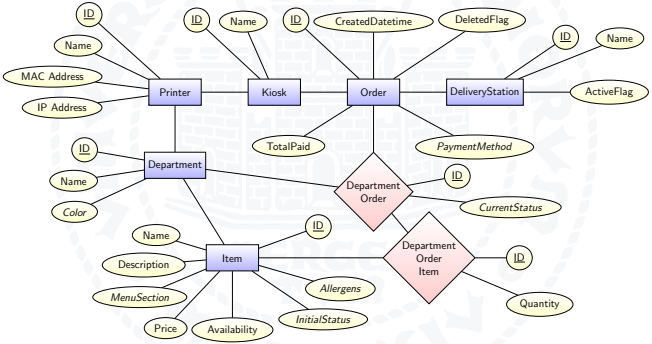

Figure: E/R diagram

Data model implementation I

The data model has been implemented using the SQLAlchemy ORM, which provides a high-level interface to the (PostgreSQL, in this case) database.

```
1 class BaseModel(DeclarativeBase):
2     __abstract__ = True
3     metadata = metadata
4     id: Mapped[UUIDType] = mapped_column(
5         cd.ID,
6         default=uuid,
7         primary_key=True,
8         comment="Unique object identifier",
9     )
```

Any class inherits from the `BaseModel` class and is automatically mapped to a table in the database.

Data model implementation II

An example of a model class is the `Department` class:

```python
class Department(BaseModel):

    id = BaseModel.id
    name: Mapped[str] = mapped_column(
        cd.SHORT_NAME,
        unique=True,
    )
    color: Mapped[Optional[str]] = mapped_column(
        cd.SHORT_NAME,
    )
    printer_id: Mapped[Optional[UUIDType]] = mapped_column(
        cd.ID,
        ForeignKey("printer.id"),
        nullable=True,
    )

    printer = relationship("Printer", back_populates="department")
    items = relationship("Item", back_populates="department")
    department_orders = relationship(
        "DepartmentOrder", back_populates="department"
    )
```

API resources I

The RESTful API has been implemented using the Flask web framework.

```python
class DepartmentResource(ProtectedResource):

    def get(
        self, *, _id: str | None = None, name: str | None = None
    ) -> tuple[dict, int]:
        if _id or name:
            department = (
                db.session.query(Department)
                .where(
                    Department.id == _id if _id else Department.name == name
                )
                .one_or_none()
            )
            if department:
                return DepartmentDTO.from_model(department), 200
            return {"message": "Department was not found"}, 404

        departments = db.session.query(Department).all()
        return DepartmentDTO.from_model_list(departments), 200
```

Code: `GET /api/v1/departments[/<string:_id>]` GET method definition

API resources II

```
1  "@context": {
2    "schema": "https://schema.org/",
3    "self": "@id",
4    "type": "@type",
5    "name": "schema:name",
6    "printer": {
7      "@id": "schema:isRelatedTo",
8      "@type": "@id"
9    },
10   "license": {
11     "@id": "schema:license",
12     "@type": "@id"
13   }
14 },
15 "license": "https://creativecommons.org/licenses/by/4.0/",
16 "self": "http://127.0.0.1/api/v1/departments/851c8de8-fb55-11ef-9aea-0242ac120003",
17 "type": "schema:Organization",
18 "name": "Kitchen",
19 "printer": "http://127.0.0.1/api/v1/printers/67936882-fb55-11ef-b5b9-0242ac120003"
```

Code: `GET /api/v1/departments/851c8de8-fb55-11ef-9aea-0242ac120003` response

Open Data principles[1] I

Make your stuff available on the Web (whatever format) under an open license.

Make it available as structured data (e.g., Excel instead of image scan of a table).

Make it available in a non-proprietary open format (e.g., CSV instead of Excel).

These requirements are satisfied by our RESTful API that allows publicly access to data in JSON-LD[2] format and provide them under Creative Commons license.

Open Data principles[1] II

Use URIs to denote things, so that people can point at your stuff.

Link your data to other data to provide context.

Every resource in the system is identified by a URI, and can be accessed by a simple HTTP GET request to that URI. A relation between resources is established by linking them in the JSON response.

The real world scenario

The system we presented today is not a requirements from a very *far away land*, after all...

`https://www.sagrevolution.it`

Q&A

Questions?

References

[1] Tim Berners-Lee. *Linked Data Design Issues*. W3.org. 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 2025-03-17).

[2] Manu Sporny, Gregg Kellogg, and Marcus Langhaler. *JSON-LD 1.0*. W3.org. 2014. URL: https://www.w3.org/TR/2014/REC-json-ld-20140116/ (visited on 2025-03-17).