RESTful Diner

A Restaurant orders management APIs system

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

Table of Contents

The presentation is divided into the following sections:

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**, so that other willing Data Scientists can help him to understand any hidden insight on the data, and to suggest him how to improve his business.

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.
The whole code of the project is available on GitHub at `https://github.com/enstit/RESTfulDiner`.

Use Case Diagram

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

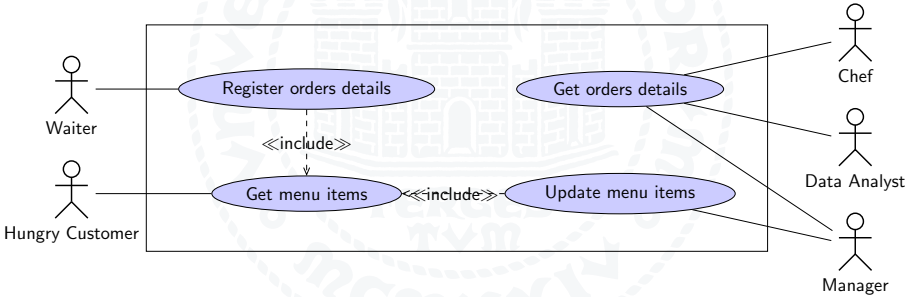

Figure: Use cases diagram

E/R Diagram

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

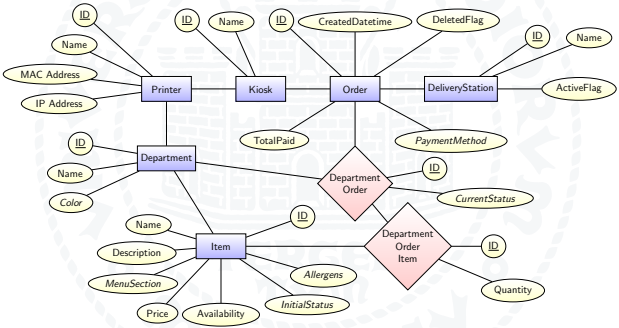

Figure: E/R diagram

Data model (logical) 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.

```python
class BaseModel(DeclarativeBase):

    __abstract__ = True
    metadata = metadata

    @declared_attr
    def id(cls) -> Mapped[UUIDType]:
        return mapped_column(
            cd.ID,
            default=lambda: uuid8(domain=cls.__name__),
            primary_key=True,
            comment=f"Unique {cls.__name__} instance identifier",
            sort_order=-1, # Ensure the id is the first column in the table
        )
```

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

Data model (logical) implementation II

```python
class Department(BaseModel):

    name: Mapped[str] = mapped_column(
        cd.SHORT_NAME,
        unique=True,
        comment="Unique name of the department",
    )
    color: Mapped[Optional[str]] = mapped_column(
        cd.SHORT_NAME,
        comment=(
            "A color between the Recognized color keyword names. "
            "See also https://www.w3.org/TR/SVG11/types.html#ColorKeywords"
        ),
    )
    printer__id: Mapped[Optional[UUIDType]] = mapped_column(
        cd.ID,
        ForeignKey("printer.id"),
        nullable=True,
        default=None,
        comment="Identifier of the printer the department is equipped with",
    )
```

Data model (physical) implementation

```
1   CREATE TABLE department (
2     id UUID NOT NULL,
3     name VARCHAR(63) NOT NULL,
4     color VARCHAR(63),
5     printer__id UUID,
6     PRIMARY KEY (id),
7     UNIQUE (name),
8     FOREIGN KEY(printer__id) REFERENCES printer (id)
9   );
10  COMMENT ON COLUMN department.id IS 'Unique Department instance identifier';
11  COMMENT ON COLUMN department.name IS 'Unique name of the department';
12  COMMENT ON COLUMN department.color IS 'A color between the Recognized color keyword names.
      See also https://www.w3.org/TR/SVG11/types.html#ColorKeywords';
13  COMMENT ON COLUMN department.printer__id IS 'Identifier of the printer the department is
     equipped with';
```

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
```

API resources II

```
    "@context": {
        "schema": "https://schema.org/",
        "self": "@id",
        "type": "@type",
        "name": "schema:name",
        "printer": {
            "@id": "schema:isRelatedTo",
            "@type": "@id"
        },
        "license": {
            "@id": "schema:license",
            "@type": "@id"
        }
    },
    "license": "https://creativecommons.org/licenses/by/4.0/",
        "self": "http://127.0.0.1:5000/api/v1/departments/5b11618c-51f5-8000-8000-6496d5c5c0cf
    ",
        "type": "schema:Organization",
        "name": "Kitchen",
        "printer": "http://127.0.0.1:5000/api/v1/printers/5b11618c-51f5-8000-8000-2a5553677712
    "
```

Demo

A live demo of the system is available at `https://diner.enricostefanel.it`.

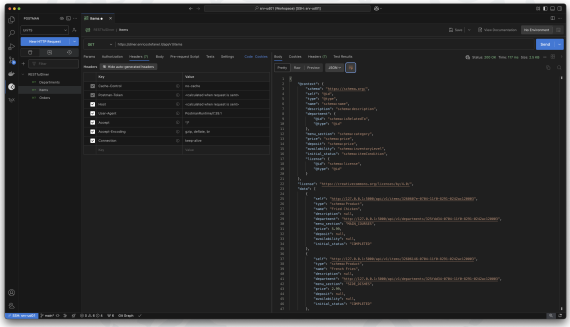

Figure: Demo of the system using Postman

Linked Open Data[1] requirements 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.

Linked Open Data[1] requirements 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. We are not fully inter-operable with other datasets, given the nature of the system, but we are open to the possibility of linking our data with other datasets in the future.

FAIR[3] principles requirements I

Findable

- **F1.** (Meta)data are assigned a globally unique and persistent identifier
- **F2.** Data are described with rich metadata (defined by R1 below)
- **F3.** Metadata clearly and explicitly include the identifier of the data they describe
- **F4.** (Meta)data are registered or indexed in a searchable resource

FAIR[3] principles requirements II

Accessible

- **A1.** (Meta)data are retrievable by their identifier using a standardized communications protocol
- **A1.1** The protocol is open, free, and universally implementable
- **A1.2** The protocol allows for an authentication and authorization procedure, where necessary
- **A2.** Metadata are accessible, even when the data are no longer available

FAIR[3] principles requirements III

Interoperable

- **I1.** (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation
- **I2.** (Meta)data use vocabularies that follow FAIR principles
- **I3.** (Meta)data include qualified references to other (meta)data

FAIR[3] principles requirements IV

Reusable

- **R1.** Meta(data) are richly described with a plurality of accurate and relevant attributes
- **R1.1** (Meta)data are released with a clear and accessible data usage license
- **R1.2** (Meta)data are associated with detailed provenance
- **R1.3** (Meta)data meet domain-relevant community standards

Data Management Plan I

1. **How will the data be created?** The data will be created by the restaurant staff, who will use the system to manage the restaurant's departments, tables, and printers.
2. **How will the data be documented?** The data will be documented in the system's database, and the API will provide a JSON-LD representation of the data.
3. **Who will access the data?** The data will be publicly accessible through the RESTful API.
4. **How will the data be stored?** The data will be stored by the system's database, which is a PostgreSQL database. In case, the database can also be installed on a cloud service.

Data Management Plan II

5. **How will the data be shared?** The data will be shared through the RESTful API, which will provide a JSON-LD representation of the data for basic HTTP requests (GET, POST, PUT, DELETE).
6. **How will the data be preserved?** The data will be preserved by the system's database, which will be backed up regularly to ensure data integrity.
7. **Who will back up the data?** The data will be backed up by the system administrator, who will ensure that the database is backed up regularly.

Storage metrics I

Let's try to imagine some storage metrics for our system:

- **Bandwidth** (volume of data read and written in a second):
  - **How many orders per day?** About 200 orders per day, which means less than a order per minute.
  - **How many items per order?** An average of 5 items per order can be considered a good approximation.

  We are talking about less than 1 MB of data per day, which is not a big deal for a modern database. $\rightarrow$ We do not need a distributed database, a single instance is enough.

Storage metrics II

- **Latency** (time taken to read and write data): since the system does not imply heavy computation and it is not required to guarantee real-time responses, we can assume that the latency is not a big deal. A good approximation is that the latency is less than 1 second for a single request. $\rightarrow$ Even an infrastructure based on hard disk drives is fast enough for our purpose.
- **Scalability** (ability to handle increasing load): the system could, in theory, be designed to already foreseen a future increase in the number of users and restaurants. $\rightarrow$ We can use a cloud service to host the database in future, which can be easily scaled up or down. The migration will be easy and fast, and the API will only require to change the connection string to the new database.

Storage metrics III

- **Reliability** (ability to recover from failures): since we are using the system in a running business, it is strictly required to have a reliable system. The database must be backed up regularly to ensure data integrity. $\rightarrow$ We can use a cloud service to host the database, which can be easily backed up and restored in case of failure. Furthermore, the filesystem could be based on a RAID 1 configuration, which assures the replication of the data on two different disks and rapid recovery in case of failure.

The real world scenario

The system we presented today is not a requirements from a very *far away land*, after all...
More information on
`https://www.sagrevolution.it`

Q&A

Questions?

References

[1] Tim Berners-Lee. *Linked Data*. W3.org. 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 2025-03-19).

[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 2025-03-19).

[3] Wilkinson Mark D., Dumontier Michel, Aalbersberg IJsbrand Jan, et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3.160018 (2016). DOI: 10.1038/sdata.2016.18.