SAPIENZA
UNIVERSITÀ DI ROMA

Data Management Project

Mamo Art Gallery

Faculty of Information Engineering, Computer Science and Statistics

M.Sc. Engineering in Computer Science

Presented by:
Gloria Marinelli, 2054014
Mario Morra, 2156770

Contents

1 Introduction

The idea is a web-based application which serves as an online marketplace for the search and purchase of artworks created by a number of different artists. Potential users are afforded the opportunity to explore a variety of paintings prior to making a purchase.

The application's architectural framework consists of the following components:

- A **database**, managed by MongoDB.
- A **backend**, constructed using Python with the Flask web framework.
- A **frontend**, developed using the ReactJS framework.

The entire application is based on two datasets: the Edvard Munch Paintings dataset and the Museum of Modern Art (MoMA) Collection dataset.

The web application is structured as follows:

- The **login page**
- the **sign up page**
- The **homepage**: displays all available paintings at MaMo Art gallery. Selecting a painting will display a summary of relevant information about the artwork, including the title, name, date, medium, dimensions, acquisition date, and other pertinent details. Furthermore, users have the option to purchase the selected artwork directly from this page, which will then be added to their list of orders.
- The **orders page**: displays all orders placed by the user, including the Order ID, the Artwork ID and the date of the order.
- The **artists page**: displays all artists associated with MaMo Art Gallery. Clicking on an artist displays their biography (nationality, gender, birth year, death year) and the list of their relative paintings.

2 Dataset Description

The application is built upon two datasets: the Edvard Munch Paintings dataset and the Museum of Modern Art (MoMA) Collection dataset.

The **first** dataset includes all known paintings created by the Norwegian artist Edvard Munch. This dataset consists of 1,789 entries, each providing metadata related to Munch's paintings. The columns in the dataset are as follows:

- **number**: index number of the painting
- **name**: title of the painting
- **year**: year of creation
- **location**: current location of the painting
- **status**: indicates whether the painting is lost (no image exists)
- **technique**: technique used in creating the painting
- **size**: original dimensions of the painting
- **filename**: image file name in the dataset

The **second** dataset is the MoMA Collection, which includes comprehensive records of artworks and artists represented in the museum. This dataset is divided into two CSV files: one containing artworks, with 218,000 records, and the other containing artists, with 67,700 records.

- `artworks.csv` contains the following columns:
  - **artwork id**: unique identifier for the artwork
  - **title**: title of the artwork
  - **artist id**: identifier for the artist
  - **name**: name of the artist
  - **date**: date of the artwork
  - **medium**: material or technique used
  - **dimensions**: physical dimensions of the artwork
  - **acquisition date**: date when the artwork was acquired by moma
  - **credit**: source of acquisition

  - **catalogue**: catalogue information
  - **department**: department within moma responsible for the artwork
  - **classification**: artwork classification
  - **object number**: unique identifier for the object in the museum's system
  - **diameter (cm)**, **circumference (cm)**, **height (cm)**, **length (cm)**, **width (cm)**, **depth (cm)**, **weight (kg)**, **duration (s)**: physical measurements of the artwork, where applicable

- `artists.csv` contains the following columns:
  - **artist id**: unique identifier for the artist
  - **name**: name of the artist
  - **nationality**: nationality of the artist
  - **gender**: gender of the artist
  - **birth year**: year of birth
  - **death year**: year of death (if applicable)

3 Database: MongoDB

MongoDB is an open-source database designed for flexibility and scalability. It stores data in documents, making it easy to handle complex data. MongoDB can handle large amounts of unstructured data and can be scaled horizontally. MongoDB also offers features like replication, high availability, and indexing for better performance. It can be scaled horizontally through sharding, which distributes data across multiple servers for better performance and reliability.

3.1 Why use MongoDB?

MongoDB is a good choice for the **MaMo Art Gallery** web app. The datasets contain different types of data, including paintings, artists, dimensions and acquisition history. MongoDB's **document-oriented approach** is ideal for this application because it can combine different datasets without any predefined relationships. MongoDB also supports quick searches and indexing, which is important for the marketplace. Furthermore, users can browse, order and view order histories in real time.

3.2 Analysis and management of data

The `database.py` script loads painting and artist data from CSV files into a MongoDB database. It cleans the data, removes duplicate painting titles, and inserts the records into a *paintings* collection. It also creates separate collections for each artist to avoid duplicates, with Edvard Munch's paintings stored in both general and specific collections. Finally, it loads the artist data into an *artists* collection, resulting in the MaMo-Art database containing *paintings*, *artists*, and individual collections for each artist.

3.2.1 CSV file reading and parsing

The data pertaining to paintings and artists is extracted from CSV files (`paintings.csv`, `edvard_munch.csv`, `artists.csv`) using the *CSV* library. These files contain information on paintings and artists, and each row is processed in turn, with any unnecessary data removed.

3.2.2 Text cleaning

The CSV files are cleaned by removing unnecessary spaces and commas to ensure data consistency before insertion and normalization. Furthermore, any special characters in collection names are removed, such as dots at the end of artist names, to prevent any potential naming issues when creating MongoDB collections.

To avoid repetition of entries for paintings and artists within the database, the titles of paintings are stored in a set (`paintings_titles`), thus simplifying the identification of duplicates during data processing.

3.2.3 Inserting data into MongoDB

Once the data has been cleaned and deduplicated, it is inserted into MongoDB collections. Each painting or artist is represented as a document in the database. This is achieved using the *insert_one* method, which allows for the addition of each record individually.

3.2.4 Managing MongoDB collections for specific artists

Paintings are inserted not only into the aggregate collection of *paintings*, but also into distinct collections for specific artists using the `update_one` method. For instance, Edvard Munch's artworks are included in both the main collection and a dedicated collection for the artist, *Edvard Munch*.

3.2.5 Handling collections

To prevent the replication of paintings during updates, the `update_one` method with the `upsert=True` parameter guarantees the insertion of new paintings into the collection if they do not already exist. This is employed when associating paintings with specific artists within their respective collections.

3.2.6 Handling duplicate checks for Edvard Munch

A specific check is carried out to determine whether a painting by Edvard Munch is already present in one of the collections. In the case that a duplicate is identified, the insertion is cancelled.

3.3 Sharding

The technique of **sharding** in MongoDB is employed to distribute data across multiple servers, thereby enhancing the performance, scalability, and reliability of large-scale databases. When a dataset reaches the limits of a single server's capacity, sharding enables horizontal scaling through the partitioning of data into smaller units, called **shards**, which are then distributed across multiple servers. Each shard contains a portion of the complete dataset.

In the context of the **MaMo Art Gallery**, the implementation of sharding could prove particularly beneficial as the size of the dataset (including the large number of paintings, artist details, and user orders) increases over time.

3.3.1 Configuring sharding

This section explains how to set up sharding in MongoDB. It involves creating multiple shards, setting up configuration servers, starting replica sets, and enabling sharding for a specific database. Here's an explanation of each step.

- **Creating directories for shards and config servers**

  ```
  mkdir C:\data\shard1
  mkdir C:\data\shard2
  mkdir C:\data\shard3
  mkdir C:\data\config1
  mkdir C:\data\config2
  mkdir C:\data\config3
  ```

  Directories are created to store data for shards and config servers. Shards store parts of the database, while config servers manage the information for the sharded cluster.

- **Starting MongoDB instances for each shard**

  ```
  .\mongod --shardsvr --replSet shard1 --port
     27001 --dbpath C:\data\shard1
  .\mongod --shardsvr --replSet shard2 --port
     27002 --dbpath C:\data\shard2
  .\mongod --shardsvr --replSet shard3 --port
     27003 --dbpath C:\data\shard3
  ```

  Three MongoDB instances are started on different ports (27001, 27002, and 27003), each representing a shard in the cluster. The `--replSet` option names each replica set (shard1, shard2, shard3).

- **Initiating replica sets for the shards**

  ```
  .\mongosh --port 27001
  rs.initiate()

  .\mongosh --port 27002
  rs.initiate()
  ```

```
.\mongosh --port 27003
rs.initiate()
```

Each shard is configured as a replica set, ensuring that data is copied for safety and high availability.

- **Starting and configuring config servers**

```
.\mongod --configsvr --replSet configReplSet
    --port 27020 --dbpath C:\data\config1
.\mongod --configsvr --replSet configReplSet
    --port 27021 --dbpath C:\data\config2
.\mongod --configsvr --replSet configReplSet
    --port 27022 --dbpath C:\data\config3
```

Three MongoDB instances store metadata for the sharded cluster. The `--configsvr` flag designates these instances as config servers. They are part of a replica set called *configReplSet* to ensure high availability.

- **Initiating the config server replica set**

```
.\mongosh --port 27020
rs.initiate({
   _id: "configReplSet",
   configsvr: true,
   members: [
      { _id: 0, host: "localhost:27020" },
      { _id: 1, host: "localhost:27021" },
      { _id: 2, host: "localhost:27022" }
   ]
})
```

The config server replica set is initiated by specifying its members on ports 27020, 27021, and 27022.

- **Starting the `mongos` router**

```
.\mongos --configdb configReplSet/localhost
   :27020,localhost:27021,localhost:27022 --
   port 27018
```

A `mongos` instance is started on port 27018. The `mongos` routes queries to the appropriate shard and connects to the config servers to get cluster information.

- **Adding shards to the cluster**

  ```
  .\mongosh --port 27018

  sh.addShard("shard1/localhost:27001")
  sh.addShard("shard2/localhost:27002")
  sh.addShard("shard3/localhost:27003")
  ```

  Each shard is added to the cluster using the `addShard` command. The system now recognizes shards 1, 2, and 3 as part of the sharded database.

- **Enabling sharding for the database**

  ```
  sh.enableSharding("MaMo-Art")
  ```

  Sharding is enabled for the `MaMo-Art` database, allowing the distribution of collections across the shards.

- **Sharding the paintings collection**

  ```
  use MaMo-Art
  sh.shardCollection("MaMo-Art.paintings", { "
     _id": "hashed" })
  ```

  The `MaMo-Art` database is sharded based on the `_id` field. A hashed sharding key ensures that data is distributed evenly across shards.

- **Checking sharding status and starting the balancer**

  ```
  sh.status()
  sh.startBalancer()
  ```

  The status of the sharded cluster is checked using `sh.status()`, and the balancer is started. The balancer helps distribute data across shards to prevent any one shard from becoming overloaded.

4 API Development: Flask

5 Frontend: React