SMBUD

Vaccination and Certificates

Specification, Entity-Relationship model and MongoDB CRUD operations

POLITECNICO
MILANO 1863

Filippo Lazzati (10629918) - Martina Magliani (10682333) - Christian Grasso (10652464) - Sofia Martellozzo (10623060) - Giacomo Lombardo (10674987)

Year: 2021/2022

Contents

1 Problem specification

The purpose of this project is to build an information system that is able to manage pandemic information for a given country. In particular, such system must be scalable enough to manage all the information about COVID-19 tests and vaccinations, and it should provide data at the granularity level required by a business perspective. Moreover, the storage system should be flexible enough in order to comply with the expiration dates of the certificates and, potentially, with the evolution of the rules.
Since `MongoDB` is a document-oriented database with the property of being schema-less, it perfectly matches this use case. Furthermore, because of its document-oriented structure, it solves the impedance mismatch problem and, thus, allows to retrieve data also through object-oriented code. The main drawback is the duplication of the data, which is, in a measure, acceptable in the scenario of the certificates.

2 Hypotheses

The database is implemented under the following hypotheses:

- for each person owning a certificate some demographic details are known;
- for each person owning a certificate is also known an emergency contact;
- there are 3 different ways through which a person can receive a certificate:
    1. healing from the disease; the validity of the certificate is fixed and assumed to be of 6 months after the recovery. A recovery is certified through one negative test after a positive one;
    2. taking at least 1 dose of vaccine; see below for details about the validity of the certificate;
    3. taking a test; the validity varies between the 48 and the 72 hours;
- by definition, a certificate is valid if it contains at least one valid vaccine/test or a certified recovery from the illness;
- each person can receive 0, 1 or 2 doses of vaccine (it should be noticed that, up to now, the third dose is not considered; however, since `MongoDB` is schema-less, the introduction of the n-th dose can be easily introduced in the database);
- there are various types of vaccines, each one of them has different characteristics in terms of validity of the certificate (here, for the sake of simplicity, the vaccines are listed using the name of the brand):
    1. Pfizer, which provides a valid certificate after 15 days from the first dose and up to 9 months after the administration of the second dose;

2. Moderna, whose certificates have the same characteristics of the Pfizer's one;
3. AstraZeneca, whose certificates have the same characteristics of the Pfizer's one;
4. Johnson & Johnson, which provides a valid certificate after 15 days from the first dose and valid for 6 months (for the time being we assume this is
5. Sputnik, which does not provide any valid certificate because of UE regulations.

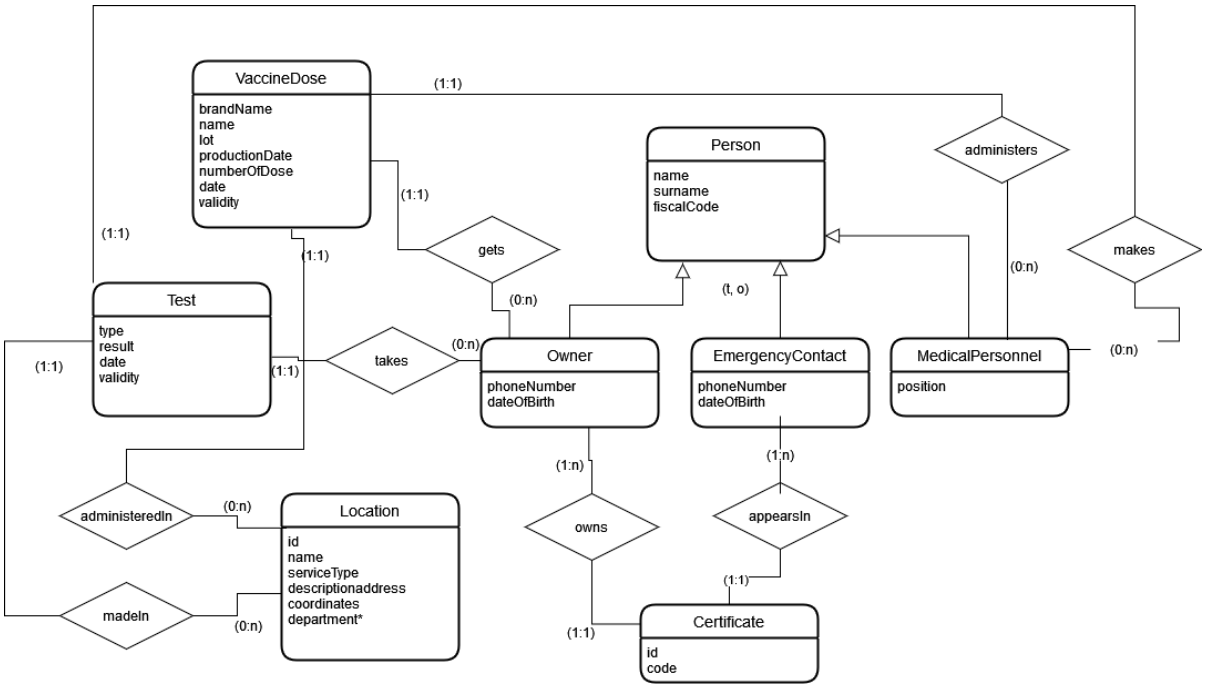
It should be noticed that the flexibility offered by a schema-less database like `MongoDB` can be very useful in this case, since the details of the certificates with regards to the type of vaccine can change over time;

- the validity of the first dose of vaccine expires 3 months in case the person does not undergoes a second dose;
- each vaccine dose is characterized by:
  1. the name of the brand that produces it;
  2. the name of the vaccine;
  3. the lot of the vaccine;
  4. the production date;
- a temporary certificate can be obtained also via a COVID-19 test, which can be:
  1. rapid, which provides a valid certificate for 48 hours;
  2. PCR, which provides a valid certificate for 72 hours;
- each test is registered when the result is available and is characterized by the type of the test (rapid/PCR);
- vaccines and COVID-19 tests can be done in various locations. Each location is characterized by:
  1. the name of the hospital/health service (or whatever other place like, for instance, a supermarket or the hairdresser) where the vaccine (test) has been administered (performed);
  2. the type of the service (e.g. hospital, local healthcare facility, pharmacy, supermarket etc.);
  3. a full text description of such place;
  4. the address of the location;
  5. the GPS coordinates of the location;
  6. optionally, the department (this can be present only for hospitals and big health structures);

- vaccines and tests are administered by authorized health personnel such as doctor and nurses, characterized by name, surname, fiscal code and the position (e.g. nurse, doctor);
- for each vaccine/test that is administered, the certificate includes the health personnel attending the administration;
- a doctor/nurse can administer a vaccine in a different place than the workplace (e.g. supermarket or another health structure);
- each certificate has a unique code of 6 digits that allows to retrieve it; such code does the same job of the QR code of the `GREEN PASS`;
- no person takes another test before that the 48h (or 72h in case of a pcr test) of a test resulted negative expire.

3 Conceptual model

The following is the Entity-Relationship (E-R) model of our database:



Notice that the optional attributes have been marked with a star (*).
The Entity-Relationship model contains 5 main different entities, that are related to each other through various relationships.

- *PERSON* represents a person. Every *PERSON* has a name, a surname and a fiscal code. Then there are 3 entities that inherit from *PERSON*, namely *OWNER*, *EMERGENCYCONTACT* and *MEDICALPERSONNEL*. It has been assumed that it is not relevant to insert the date of birth of the *MEDICALPERSONNEL* in the certificate, and that it should be better to avoid to insert its phone number in it. Of course the attributes of *PERSON* could be enlarged, but as a sample dataset we have believed that these three are enough;
- *CERTIFICATE* represents the main concept of the whole database. The *CERTIFICATE* will be the document on which we put more or less all the data we have, because the *CERTIFICATE* is at the level of granularity with which users want to work with this data. We have preferred to represent such entity with only an identifier and the unique code of 6 digits, since it gains all the data it owns through its relationships;
- *LOCATION* represents a place where a vaccine or a test can be done. It

should be remarked that we have decided to detail in-depth such entity, because it is relevant for this scenario. Because of the many attributes it has, you will notice that this is the only document that has not been added to the *CERTIFICATE* document to avoid a useless waste of storage space (as a matter of fact, we have assumed that the *LOCATION* is relevant but it is not required so much from the "business", therefore when it is asked they can wait some additional time);

- *VACCINEDOSE* represents one single dose of vaccine received by a person; it has a name, the name of the brand, the lot number and the date of production;
- *TEST* represents one single test taken by a person; it is characterised by a type (rapid or PCR), a result, a date and the validity (48 hourse or 72 hours).

Among these entities, some relations hold. In particular:

- *ADIMINISTERS* and *MAKES* represent the relation between a medical personnel and the vaccine/test he does;
- *GETS* and *TAKES* bind a person (*OWNER*) with its doses of vaccine/tests;
- *ADMINISTEREDIN* and *MADEIN* specify in which location a certain vaccine dose/test has been administered;
- *OWNS* simply relates a person (*OWNER*) to her certificate;
- *APPEARSIN* relates the *EMERGENCYCONTACT*s to the certificates in which they appear as such.

4 Sample dataset

The sample dataset we provide is made by the following 2 documents (and all the nested subdocuments). As previously said, the decision of avoid to embed the `LOCATION` documents inside the `CERTIFICATE` ones has been taken in order to prevent a huge and useless waste of storage space. The following is an example of a `CERTIFICATE` document:


```
{
        "code": "3U4162",
        "owner": {
            "name": "John",
            "surname": "Morris",
            "dateOfBirth": "1986-07-14",
            "fiscalCode": "JHNMRS86L14S677A",
            "phoneNumber": 3313567314
        },
        "recovered": null,
        "emergencyContact": {
            "name": "Michelle",
            "surname": "Hampton",
            "dateOfBirth": "2006-03-31",
            "fiscalCode": "MCHHPT06C71Q216S",
            "phoneNumber": 3002475590
        },
        "vaccines": [
            {
                "type": {
                    "name": "Comirnaty",
                    "brandName": "Pfizer\/BioNTech",
                    "lot": "NRZZZHOH",
                    "productionDate": "2021-03-03"
                },
                "numberOfDose": 1,
                "validity": 6480,
                "date": "2021-04-06",
                "medicalPersonnel": {
                    "name": "Peter",
                    "surname": "Davis",
                    "fiscalCode": "PTRDVS48P04L453S",
                    "position": "doctor"
                },
                "locationId": "location_949"
            },
            {
                "type": {
```

```
                "name": "Comirnaty",
                "brandName": "Pfizer\/BioNTech",
                "lot": "KDTULQPG",
                "productionDate": "2021-04-15"
            },
            "numberOfDose": 2,
            "validity": 6480,
            "date": "2021-05-01",
            "medicalPersonnel": {
                "name": "Robert",
                "surname": "Savage",
                "fiscalCode": "RBRSVG76R210827S",
                "position": "doctor"
            },
            "locationId": "location_1922"
        }
    ],
    "tests": [
        {
            "type": "pcr",
            "validity": 72,
            "result": false,
            "date": "2021-02-02",
            "medicalPersonnel": {
                "name": "James",
                "surname": "Williams",
                "fiscalCode": "JMSWLM94C24X515S",
                "position": "doctor"
            },
            "locationId": "location_1924"
        }
    ]
}
```


A `CERTIFICATE` document is recognized (like every document in `MongoDB`) by an "_id" and contains:

- a unique 6-digits code different from the "_id", that can be used to identify the certificate (such code has the same role of the QR code in the `GREEN PASS`);
- the owner of the document;
- the emergency contact of the owner of the document;
- a list of all the doses of vaccine received (in this way if the rules evolve, additional doses can be added to new documents);

- a list of all the tests taken by the owner of the certificate.

The `LOCATION` document is the following:

```
{
        "_id": "location_0",
        "name": "Location 0",
        "address": "VIA PANORAMICA 42, BORGO TOSSIGNANO",
        "typeOfService": "vaccinationCenter",
        "coordinates": [
            43.2881,
            12.12
        ],
        "department": null
    }
```

It is recognized by an "_id" and contains all the attributes identified in the conceptual model (E-R) of the data. The `CERTIFICATE` documents contain links to it (when needed).
The sample dataset has been created through some `PHP` scripts.

5 Import the sample dataset

The sample dataset is given through two different `JSON` files, "certificates.json" and "locations.json". The former contains an array of certificate documents while the latter an array of location documents. For the sake of simplicity we have decided to generate on our own the "_id" attribute of the locations.
To import the dataset, you have to follow these steps :

1. open a command prompt and type

```
1    mongod
```

2. open `MongoDBCompass` and type the string "mongodb://127.0.0.1:27017/";
3. create a new database called "db" and a new collection called "collection" inside it, then click on *ADD DATA*;
4. import the two `JSON` files "certificates.json" and "locations.json".

6 Queries and Commands

The following queries and commands have been developed in order to provide an example of usage of the system for typical usage scenarios.

6.1 Queries

We have identified the following different queries:

1. **find all the people who have done the disease**.
   We simply have to check whether the "recovered" attribute is null or not:

```
db.collection.find({"recovered":{$ne:null}}, {"owner": 1});

```

2. **find all the people who have undergone 3 vaccine doses**.
   Thanks to the $size operator, we can make a query on the length of an array:

```
db.collection.find({vaccines: {$size: 3}}, {"owner": 1});

```

3. **find all the valid certificates**.
   According to the hypotheses (see section 2), a certificate is valid in 3 cases:

   (a) the owner took a (negative) test in the last hours -> 48h or 72h of validity;

   (b) the owner healed from the disease (verified through a negative test after a positive one) -> 6 months $\approx$ 4320 hours of validity;

   (c) the owner underwent at least one vaccination dose -> 3 months of validity for the first dose, then the validity varies according to the type of vaccine.

   Therefore, we can write 3 different queries to retrieve certificates that are valid for different reasons.

   (a) Certificates of people who have recovered in the last 6 months:

```
db.collection.find({"recovered": {$gte: new Date(ISODate()
    .getTime() - 1000 * 60 * 60 * 24 * 30 * 6)}});

```

   (b) certificates of people who have taken a (negative) test in the last hours (48 or 72 according to the type of test):

```
db.collection.find({$or: [
{tests: {$elemMatch: {validity: 48, result: false, date: {
    $gte: new Date(ISODate().getTime() - 1000 * 60 * 60 *
    24 * 2)}}}},
{tests: {$elemMatch: {validity: 72, result: false, date: {
    $gte: new Date(ISODate().getTime() - 1000 * 60 * 60 *
    24 * 3)}}}}]});

```

   N.B.: it should be remarked that this query is correct only thanks to the assumption that states that no person can take another test while having taken a negative test in the last 48/72 hours.

(c) certificates of people who have undergone vaccination not too much time ago and therefore the certificates are valid (notice that, since the first dose gives a valid certificate after 15 days, we have to add such value in the conditions):

```
db.collection.find({$or: [
    {vaccines: {$elemMatch: {validity: 2160, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 2160 +
     1000 * 60 * 60 * 24 * 15)}}}},
    {vaccines: {$elemMatch: {validity: 6480, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 6480)
    }}}},
    {vaccines: {$elemMatch: {validity: 4320, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 4320)
    }}}}
]});

```

It should be noticed that this query is not exactly correct, because there is the possibility that a person is tested positive after having received the vaccine, and thus her certificate is not valid anymore. However, for the sake of simplicity, we will check this fact with `JavaScript` in the user interface.

4. **find the type of vaccine which has been administered the most**. It is clear that we have to use aggregations for this query. The pipeline starts from `$unwind`, which allows to expand the content of the array "vaccines"; next, `$group` allows to group the documents by brandName, and finally we sort the results through `$sort` and take the maximum using `$limit`.

```
db.collection.aggregate([
    {
        '$unwind': {
            'path': '$vaccines'
        }
    }, {
        '$group': {
            '_id': '$vaccines.type.brandName',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }, {
        '$limit': 1
    }
]);

```

5. **find the medical personnel who has administered more vaccines**.

This query is rather similar to the previous one; indeed, we need again to use aggregation and the pipeline proceeds along the same operators:

```
db.collection.aggregate([
    {
        '$unwind': {
            'path': '$vaccines'
        }
    }, {
        '$group': {
            '_id': {
                'fiscalCode': '$vaccines.medicalPersonnel.fiscalCode',
                'name': '$vaccines.medicalPersonnel.name',
                'surname': '$vaccines.medicalPersonnel.surname',
                'position': '$vaccines.medicalPersonnel.position'
            },
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }, {
        '$limit': 1
    }
]);
```

6. **find the people who have taken only tests**.
   This query allows to retrieve all the people who have never undergone vaccination but who have taken at least one test. Since the owner of a certificate is inside the certificate, we simply have to project the owner:

```
    db.collection.find(
  {"$and":[{"vaccines.date":{$exists:false}},{"tests.date":{$exists:true}}]},
  {"owner": 1});
```

7. **find the 10 locations with the highest number of vaccines administered**.
   Like two previous queries, again we are looking for the maximum, therefore the pipeline is the same. The only difference is that now we are looking for the top 10 locations, thus the `$limit` operator has a 10 instead of a 1:

```
    db.collection.aggregate([
    {
        '$unwind': {
            'path': '$vaccines'
        }
    }, {
        '$group': {
```

```
            '_id': '$vaccines.locationId',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }, {
        '$limit': 10
    }
]);

```

6.2 Commands

We have identified the following commands to show how the system works:

1. **insert a new certificate because of a vaccine dose**.
   .... Explanation of the command .....

```

```

2. **insert a new certificate because of a test**.
   .... Explanation of the command .....

```

```

3. **insert a certificate because a new person has received the first vaccine dose**.
   .... Explanation of the command .....

```

```

7 User Interface

7.1 Description

We have provided a demo of a User Interface in order to show some use cases of the database and how to apply some sample queries/commands to it. We have implemented a simple client-server application with some basic functionalities to query and update the database. The application is implemented in `JavaScript`, using *node.js* and *express.js* to run the server and the **React** framework for the front-end.

7.2 Functionalities

The application allows to:

- ...
- ...

8 User guide

8.1 Requirements

In order to run the application, the following software need to be installed on your machine:

- ...
- ...

8.2 Database

To run the database:

1.

8.3 Server

After having installed *Node.js*, to run the server do:

1. ...
2. ...

8.4 Client

After having started the server, to run the front-end of the application do:

1. ...
2. ...

9 References and sources

- MongoDB
- draw.io
- node.js
- express.js
- React

10 Image gallery

In this section we include some screenshots showing the appearance of the User Interface (UI). Read the captions for details about what each image represents.