SMBUD Delivery #2
2021/22

Prof. Brambilla Marco

Reale Simone	(Personal Code: 10616980)
Shalby Hazem	(Personal Code: 10596243)
Somaschini Marco	(Personal Code: 10561636)
Urso Giuseppe	(Personal Code: 10628602)
Vitobello Andrea	(Personal Code: 10608395)

12 December 2021

POLITECNICO
MILANO 1863

Contents

1 Problem specification

The project purpose is to design a documental DB for supporting a certification app for COVID-19. The database must be designed to store the following data:

- The certificates (with all correlated info, such as vaccinations, tests, doctors/nurses attending the vaccination/test)
- The authorized bodies (with description, gps location, address, type ecc.)

2 Hypothesis and assumptions

In this second delivery, the following hypothesis/assumptions have been made:

- The expiration date of a certificate is stored in the certificate document itself
- After a vaccination and/or a test, the expiration date of a certificate is updated (if needed)
- The fiscal code of a person is unique
- The 'partita iva' of an authorized body is unique
- Only four types of covid vaccine exist:
  - PFIZER
  - MODERNA
  - ASTRAZENECA
  - SPUTNIK
- Only three types of covid test exist:
  - Molecular test;
  - Antigen test;
  - Antibody test.
- An authorized body can be active or inactive (it depends on whether it still have vaccinations/tests or not)

3 ER diagram

Certificate document

```
{
  _id: <ObjectId1>,
  name: "xyz",
  surname: "xyz",
  cf: "123xyz",
  birthdate: "yyyy-mm-dd",
  details: "xyz",
  list_of_vaccinations:[
    {
        brand:"vaccine_brand"
        lot:1234
        date:"aaaa-mm-dd"
        production_date:"aaaa-mm-dd"
        location:"xyz"
        doctor:"name surname"
        piva:"xyz"
    }
  ],
  list_of_tests:[
    {
        test_type:"test_type"
        date:"aaaa-mm-dd"
        location:"xyz"
        result:boolean
        doctor:"name surname"
        piva:"xyz"
    }
  ],
  validity_date: "yyyy-mm-dd",
}
```

The above schema describes the structure of the `certificate collection`. The document can be divided into three main parts:

- **General information**: personal information (name, surname, birth date, ...) and the validity date of the certificate;
- **List of vaccinations**: array of the vaccinations made. In particular, each element of the list contains information on the brand, location and date of the vaccine injection, with the lot used and the doctor who supervised the injection.
- **List of tests**: array of the tests made. In particular, each element of the list contains information on the type, result,location and date the test was executed, along with the doctor who supervised it.

Authorized Body document

```
{
  _id: <ObjectId1>,
  piva: "123xyz",
  name: "xyz"
  gps: "yyyy-mm-dd",
  type: "xyz",
  location: "xyz"
  description: "xyz",
  active: "xyz",
  department: "xyz"
  list_of_doctors:[...]
}
```

The above schema describes the structure of the `Authorized_Body collection`. The above is the general schema of an authorized body but reflecting the use of documents and the advantages of document databases, some fields are left unfilled on purpose. Those fields can however be added through commands.

4 Data generation

The generation of data is done using a python script. For a better understanding of the data generation process go to Simulation section.

5 Queries

In this section we will discuss some of the most significant queries, but the database is designed in a way that facilitate any type of query.

5.1 Number of people by number of doses

```
results = col_cert.aggregate(
    [
        {
            "$project": {
                "size_of_vaccines_array": {
                    "$cond": {
                        "if": {"$gte": [{"$size": "$list_of_vaccinations"}, 5]},
                        "then": ">5",
                        "else": {"$toString": {"$size": "$list_of_vaccinations"}},
                    }
                }
            }
        },
        {
            "$group": {
                "_id": "$size_of_vaccines_array",
                "count": {"$sum": 1}
            }
        },
        {
            "$sort": {"_id": 1}
        }
    ]
)
```

The query simply gets the number of people divided by number of doses. The query is executed using an aggregation pipeline of three stages that process the documents:

- `$project`: calculate the number of doses and using `$cond` aggregates all the people with more than five doses executed. Since the data are used to be visualized on a pie chart, all the people with more than five doses are aggregated.
- `$group`: groups the results using the count calculated in the previous stages
- `$sort`: this is used only for the visualization part

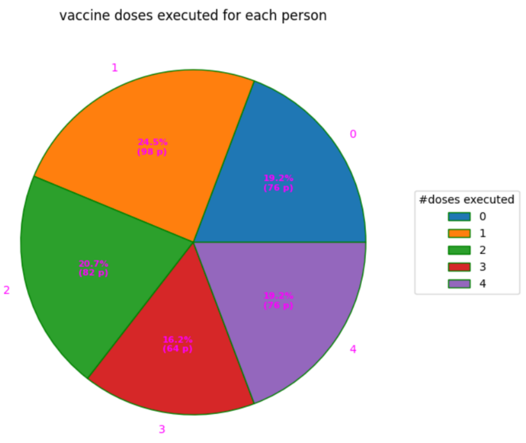

Figure 1: Example of the result of the above query plotted used the application described later

5.2 Number of people by number of tests

```
results = col_cert.aggregate(
    [
        {
            "$project": {
                "size_of_tests_array": {
                    "$cond": {
                        "if": {"$gte": [{"$size": "$list_of_tests"}, 5]},
                        "then": ">5",
                        "else": {"$toString": {"$size": "$list_of_tests"}},
                    }
                }
            }
        },
        {
            "$group": {
                "_id": "$size_of_tests_array",
                "count": {"$sum": 1}
            }
        },
        {
            "$sort": {"_id": 1}
        }
    ]
)
```

The query gets the number of people divided by number of tests executed. The query is executed using an aggregation pipeline of three stages that process the documents:

- `$project`: calculates the number of tests and using `$cond` aggregates all the people with more than five tests executed. Since the data are used to be visualized on a pie chart, all the people with more than five tests executed are aggregated.
- `$group`: groups the results using the count calculated in the previous stages
- `$sort`: this is used only for the visualization part

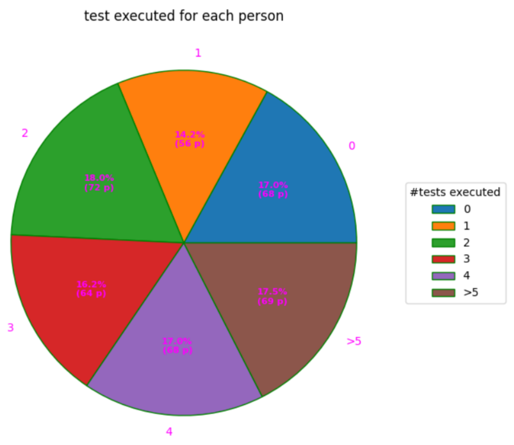

Figure 2: Example of the result of the above query plotted used the application described later

5.3 Number of doses per vaccine

```
results = col_cert.aggregate(
    [
        {
            "$unwind": {
                "path": "$list_of_vaccinations",
                "preserveNullAndEmptyArrays": False,
            }
        },
        {
            "$group": {
                "_id": "$list_of_vaccinations.brand",
                "count": {"$sum": 1}
            }
        }
    ]
)
```

The query gets the number of doses for each vaccine. The query is executed using an aggregation pipeline of two stages that process the documents:

- `$unwind`: deconstructs the `list_of_vaccine` field of each person.
- `$group`: groups the results based on the vaccine type and calculates the count for each vaccine type.

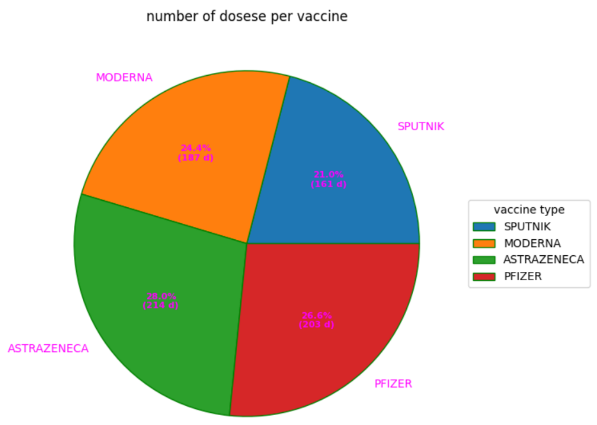

Figure 3: Example of the result of the above query plotted used the application described later

5.4 Get person vaccine and tests log

This query allows to retrieve the list of vaccinations and/or tests of a person, given his/her name and his/her surname.
The query is relatively simple. It only retrieves a document (if any) from the certificate collection where the field name and surname have as values the input of user in the UI interface. The fundamental lines of code are the following:

```
query = {"cf": person_cf}
dict_person = col_cert.find_one(query)
```

If the variable `dict_person` is non-empty, then for each possible value of the fields `list_of_vaccinactions` or `list_of_tests` (depends on what button the user clicked) it will be added a row into the table showed in the UI frame.

5.5 Get location with more vaccines/tests

The query retrieves the top 3 Authorized Bodies that performed the most vaccinations and most tests. This is achieved through an aggregation of commands:

1. The list of vaccinations/test for each individual is unwound
2. Authorized Bodies are grouped based on their "piva" attribute and their appearances are counted
3. The resulting tuples are then sorted based on the number of appearances and the first 3 are returned

Note that an analogous query is performed for vaccines.

```
db.Certificate_Collection.aggregate(
    [
        {
            "$unwind" : "$list_of_tests"
        },
        {
            "$group": {
                "_id": { "piva" : "$list_of_tests.piva" },
                "count": {"$sum": 1}
            }
        },
        {"$sort": {"count": -1}},
        {"$limit": 3}
    ]
)
```

5.6 Search Green Pass

The query "Search Green Pass" does not more than retrieving the certificate of the searched person, it displays the expiration date of the certificate and a qr code that summarises some personal details, the result of the search is always up to date in a non-concurrent situation.

5.7 Find the group of people that was vaccinated with a certain lot

This simple query aims at finding all the people that were vaccinated with a certain lot. In this way, if the lot is then found faulty people can be easily reached. This query uses the "$elemmatch" operator to select the documents that contain the specified value inside the list of vaccines.
The query then returns the list of whole certificates, then the python program projects only name, surname, birthdate and CF to simplify the visualization process for the end-user.

```
db.Certificate_collection.find({"list_of_vaccinations" : { "$elemMatch" : { "lot" :  int(lot)
↪  }}})
```

6 Commands

6.1 Add authorized body

As mentioned in the specification document of the second delivery of the project activity, in the database we have, in addition to certificates, info also about the authorized bodies. So, a natural command is the possibility to add a new authorized body to the database. The interface provides many text fields to fill with the authorized body details (name, “partita iva”, type, address, gps location, department issuing vaccines, description and a list of doctors) and, after have filled all this text fields, a user can add to the db the new authorized body. The command is straightforward, as it is shown in the following lines of code:

```
new_auth_body = func.createGoodAuthorizedBody(name=insert_name.get(),
↪  piva=int(insert_piva.get()),type=insert_type.get(), address=insert_addr.get(),
↪  gps=insert_location.get(), department=insert_dept.get(),
↪  description=insert_description.get(), doctors=insert_docs.get())
col_authBody.insert_one(new_auth_body)
```

Where the function `createGoodAuthorizedBody` is the following:

```
def createGoodAuthorizedBody(name, piva, type, address, gps, department, description,
↪  doctors):
    doctors_array = doctors.split(";")
    dict_auth = {'active' : True, 'name': name, 'piva': piva, 'type' : type, 'location' :
    ↪  address, 'gps': gps,'department': department, 'description': description,
    ↪  'list_of_doctors': doctors_array}
    return dict_auth
```

6.2 Add Person

This command allows to enter a new individual in the database by specifying its Name, Surname, Birthdate and Details. All these are verified not to be empty, otherwise an error is raised. The vaccination/test lists are initialized as empty since those can be added to the newly inserted person through their specific command.

```
db.Certificate_Collection.insertOne({
    'name' : name,
    'surname' : surname,
    'birthdate' : birthdate,
    'details' : details,
    'list_of_vaccinations' : [],
    'list_of_tests' : [],
    'cf': cf,
    'validity_date': validity_date
})
```

6.3 Add vaccine/test

The command ”Add vaccine/test” not only allows the user to add a test or a vaccine to the medical record of the patient, but also modifies the validity of his certificate according to the newly updated data. Everything is coordinated by the function `returnCertificateExpirationDate` that analyzes the dictionary that represent the certificate of a person and returns its expiration date.

6.4 Update authorized body

This command aims at updating an authorized body. The selection is performed through the PIVA value, while the fields to be updated are ”active” and ”description”.

- the PIVA is unique, thus it will only select the one authorized body to update.
- the ”description” field contains a recap of the activities of the body. Note that each time a new description is uploaded, the new sting will be appended to the previous description. In such manner it is possible to create a log-like structure with the whole history of the authorized body: as public entities, in fact, these bodies should be as transparent as possible. Note that the description field does not exist when the authorized body is created by the python script, thus it is necessary to check through the ”$ifNull” operator if the ”$description”, the value of the ”description” field, is null or not. If the value is null, then a new field is created.
- the ”active” field tells if the body is still providing its services. True means that the body is still operating.

Note that it is possible to leave the ”active” field untouched as the python program implements a variant of this command where only the description is updated.

```
db.Authorized_Body.updateOne(
 { "piva" : int(PIVA)},
 [{ "$set" : { "active" : active, "description" :  {"$concat": [ {"$ifNull" : ["$description",
 ↪  ""]}, description ]} }}]
)
```

7 Application

To make all the above queries and commands easy to use for the users, we developed an application with the related GUI.
The application GUI is simply used to get the parameters necessary to run the query/command and to visualize the results of the queries.
For a better understanding of the application, a copy of the whole project is available on the Github repository of the project and also a small demo is available on YouTube link.