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 **2**

2 Hypotheses **2**

3 Conceptual model **5**

4 Sample dataset **7**
4.1 Example of documents 7
4.2 Import the dataset 9

5 Queries and Commands **9**
5.1 Queries 9
5.2 Commands 14

6 User Interface **16**

7 User guide **16**

8 References and sources **17**

9 Image gallery **17**

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. As countries progressively re-open spaces and activities, it is necessary to adopt preventive measures to keep the pandemic under control: one of the main approaches is to introduce individual certificates containing relevant information about tests and vaccines, and allow individuals to access public places and activities accordingly to requirements such as being vaccinated or tested.
To support these rules, it is necessary to implement a system able to manage all the necessary information relative to a certificate and to allow external applications to quickly verify its validity. In particular, such system must be scalable enough to manage information about COVID-19 tests and vaccinations for a huge number of individuals, 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 schema-less approach also allows to store different information for each certificate, allowing an easy management of vaccines and tests of each individual. 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 valid 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, 2 or 3 doses of vaccine (it should be noticed that, up to now, the fourth 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 vaccine, each one with different characteristics in terms of the validity of the certificate (here, for the sake of simplicity, the vaccines are listed using the name of the brand):
  1. Pfizer/BioNTech, 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 that no person can take more than one vaccine dose if she has taken the Johnson & Johnson vaccine as the first one);

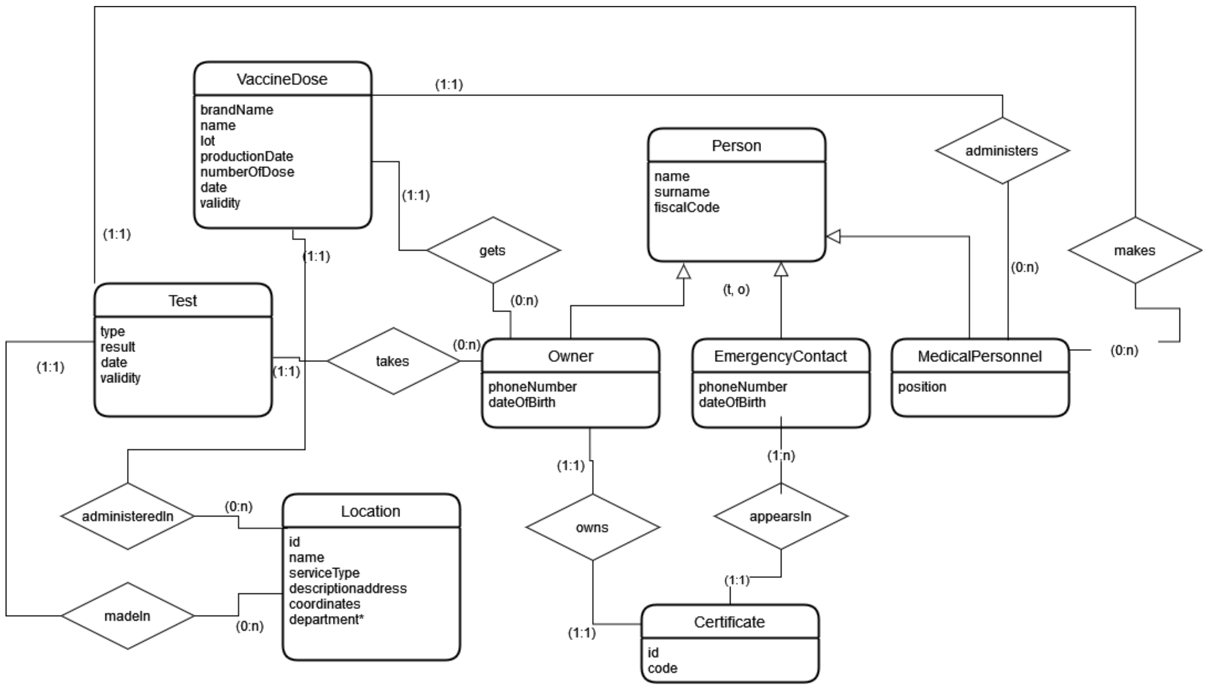
  It should be noticed that the flexibility offered by a schema-less database like `MongoDB` can be very useful in this case, since the details of the certificates with regards to the type of vaccine can change over time;
- the validity of the first dose of vaccine expires 3 months in case the person does not undergo 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 (taken);
   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 (this assumption is rather strong, but we consider it meaningful and useful, therefore we decided to add it).

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 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 acquires 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 write in detail 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). The *LOCATION*s will lie in a different collection;

- *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. **Notice that there is a 1-to-1 relationship between *PERSON* and *CERTIFICATE*: in our database, every time a person takes a test (or a vaccine dose), it is added to the same certificate of the person. In this way, there is not an annoying change of the 6-digits code (QR code) every time the certificate changes (like in GREEN PASS). If a certificate is not valid, it will be simply parsed as non-valid.**
- *APPEARSIN* relates the *EMERGENCYCONTACT*s to the certificates in which they appear as such.

4 Sample dataset

4.1 Example of documents

The sample dataset we provide, created through some PHP scripts, is made of the following two documents (and all the nested subdocuments). As previously said, the decision to 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": "L7R43T",
        "owner": {
            "name": "John",
            "surname": "Morris",
            "dateOfBirth": "1986-07-14",
            "fiscalCode": "JHNMRS86L14G797S",
            "phoneNumber": 3155102921
        },
        "recovered": null,
        "emergencyContact": {
            "name": "Jean",
            "surname": "Beaudoin",
            "dateOfBirth": "1993-09-12",
            "fiscalCode": "JNEBDN93P521595H",
            "phoneNumber": 3498952472
        },
        "vaccines": [
            {
                "type": {
                    "name": "Spikevax",
                    "brandName": "Moderna",
                    "lot": "HJLTXE5E",
                    "productionDate": 2021-01-07T21:13:06.000+00:00
                },
                "numberOfDose": 1,
                "validity": 2160,
                "date": 2021-01-28T21:13:06.000+00:00,
                "medicalPersonnel": {
                    "name": "Scott",
                    "surname": "Mckinney",
                    "fiscalCode": "SCTMKN69P029128N",
                    "position": "nurse"
                },
                "locationId": "location_35"
            }
        ],
        "tests": [
            {
                "type": "rapid",
                "validity": 48,
                "result": false,
                "date": 2020-08-09T11:18:43.000+00:00,
                "medicalPersonnel": {
                    "name": "Robert",
                    "surname": "Savage",
                    "fiscalCode": "RBRSVG76R210999T",
                    "position": "doctor"
                },
                "locationId": "location_1014"
            }
        ]
    }
```

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 is intended to simulate the function of the QR code in the `GREEN PASS`, providing a "fast" way to retrieve the certificate from the database);
- 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.

Notice that the validity is expressed in hours; so, for instance, the Pfizer/BioN-Tech vaccine which gives a validity of 9 months as second dose is represented with the field: $validity = 9 * 30 * 24 = 6480$. This is a design choice.

On the other side, an example of `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).

4.2 Import the 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 (since certificates and locations are different kinds of document, they will be imported into different collections):

1. open a command prompt and type

   ```
       mongod
   ```

   to start MongoDB locally. In alternative, you can use services such as MongoDB Atlas to host your database;

2. open `MongoDBCompass` and type the string "mongodb://127.0.0.1:27017/" if MongoDB is running on your machine, otherwise insert the URI provided by the hosting platform;

3. create a new database called "db" and two new collections called "certificates" and "locations" inside it;

4. import the two `JSON` files "certificates.json" and "locations.json" respectively in the collections certificates and locations (exploit the *ADD DATA* button).

5 Queries and Commands

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

5.1 Queries

We have identified the following queries:

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

   ```
   db.certificates.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.certificates.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 healed from the disease (verified through a negative test after a positive one) -> 6 months ≈ 4320 hours of validity;

   (b) the owner took a (negative) test in the last hours -> 48h or 72h 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.certificates.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.certificates.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.certificates.find({$or: [
    {vaccines: {$elemMatch: {validity: 2160, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 2160 +
     1000 * 60 * 60 * 24 * 15)}}}},
    {vaccines: {$elemMatch: {validity: 4320, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 4320)
    }}}},
    {vaccines: {$elemMatch: {validity: 6480, date: {$gte:
    new Date(ISODate().getTime() - 1000 * 60 * 60 * 6480)
    }}}}
]});

```

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, these checks will be performed by applications and not by the database itself.

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 with `$unwind`, which allows to expand the content of the array ”vaccines”; next, `$group` allows to group the documents by brandName, and finally we sort the results through `$sort` and take the maximum using `$limit`.

```
db.certificates.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.certificates.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.certificates.find({
            "$and":[{"$expr":{$eq:[{$size:"$vaccines"},0]}},{"
    $expr":{$gte:[{$size:"$tests"},1]}}]
    });
```

7. **Find the 10 locations with the highest number of vaccines administered**.
   Like two previous queries, once 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.certificates.aggregate([
    {
        '$unwind': {
            'path': '$vaccines'
        }
    }, {
        '$group': {
            '_id': '$vaccines.locationId',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }, {
        '$limit': 10
    }
]);

```

Then the details about the locations can be retrieved by querying the `locations` collection with the returned `_id`s.

8. **Find how many vaccinated people there are in the system**.
   For this query we use the aggregation construct; we match all the certificates with at least one vaccine dose (size greater than or equal to 1) and then we count how many certificates we have. Clearly, we have to use the `certificates` collection:

```
db.certificates.aggregate([
            {
                '$match': {
                    '$expr': {
                        '$gte': [
                            {
                                '$size': '$vaccines'
                            }, 1
                        ]
                    }
                }
            }, {
                '$count': 'numOfVaccinatedPeople'
            }
        ]);
```

9. **Find all the vaccinated people and, for each of them, the number of vaccine doses**.
   We use an aggregation construct in which we apply the `$project` operator to take only the persons along with the number of doses done, then we get rid of the people with 0 vaccine doses:

```
        db.certificates.aggregate([
            {
               '$project': {
              'person': '$owner',
                  'numberOfDoses': {
                      $size: '$vaccines'
                }
             }
         },
         {
         '$match': {
                    '$expr': {
                        '$gte': [
                                '$numberOfDoses', 1
                        ]
                    }
                }}]);

```

5.2 Commands

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

1. **Update a certificate because of a new vaccine dose**.
   We use the `updateOne()` function in which we look for the certificate to update through the unique 6-digits code (≈ QR code) and then we push inside the `vaccines` array the new dose. It should be noticed that the use of the construct `$date` is essential in order to insert dates instead of strings:

```
        db.certificates.updateOne(
            {
                code: 'L7R43T'
                }, {
                    $push: {
                        'vaccines': {
                        'type' : {
                            'name': 'Vaxzevria',
                            'brandName': 'AstraZeneca',
                            'lot': 'K60GB2VP',
                               'productionDate': {
                        '$date': '2021-01-07T22:13:06+01:00'
                    }
                            },
                        'numberOfDose': 1,
                        'validity': 2160,
                        'date': {
                        '$date': '2021-01-07T22:13:06+01:00'
                    },
                        'medicalPersonnel': {
                            'name': 'Harry',
                            'surname': 'Cooper',
                            'fiscalCode': 'HRRCPR45H08A966A',
                            'position': 'other'
                               },
                        'locationId': 'location_1175'
                        }
                    }
            });

```

2. **Update a certificate because of a new test**.
   This query is quite similar to the previous one: we find a certificate through the unique 6-digits code and we push into the `tests` array the new test. For the sake of simplicity, the task of checking whether the owner has previous tests and whether the last test taken is positive is performed by the application; in such a case, if the new test is negative the application will automatically update the field `recovered` of the document with the date of the new test.

```
        db.certificates.updateOne({
            code: 'L7R43T'
            }, {
            $push : {
                'tests': {
                    'type': 'rapid',
                            'validity': 48,
                            'result': false,
                            'date': {
                            '$date': '2021-01-07T22
    :13:06+01:00'},
                            'medicalPersonnel': {
                               'name': 'Albert',
                                'surname': 'Saults',
                                'fiscalCode': '
    LBRSTS17S13W706O',
                                'position': 'nurse'
                                },
                            'locationId': 'location_1988'
                }
            }

        });

```

3. **insert a certificate because a new person has received the first vaccine dose or has made the first test**.
   In this case, the person does not have a certificate yet, therefore we exploit the `insert()` function to insert a new certificate in the database (in the example Michael Penn has just taken his first vaccine dose):

```
db.certificates.insert({
    'code': 'QSA4GP',
    'owner': {
        'name': 'Michael',
        'surname': 'Penn',
        'dateOfBirth': '1986-06-03',
        'fiscalCode': 'MCHPNN86H03C751C',
        'phoneNumber': {
            '$numberLong': '3121784343'
        }
    },
    'recovered': null,
    'emergencyContact': {
        'name': 'Esther',
        'surname': 'Bernard',
        'dateOfBirth': '1968-06-30',
        'fiscalCode': 'STHBNR68H70R928L',
        'phoneNumber': {
            '$numberLong': '3585672957'
        }
    },
    'vaccines': [{
        'type': {
            'name': 'Janssen',
            'brandName': 'J&J',
            'lot': '5S3QLBDA',
            'productionDate': {
                            '$date': '2021-01-07T22
    :13:06+01:00'},
        },
        'numberOfDose': 1,
        'validity': 4320,
        'date': {
                '$date': '2021-01-07T22:13:06+01:00'},
        'medicalPersonnel': {
            'name': 'Vincent',
            'surname': 'Vang',
            'fiscalCode': 'VNCVNG55R160373I',
            'position': 'nurse'
        },
        'locationId': 'location_1006'
    }],
    'tests': []
});

```

6 User Interface

We have provided a demo of a User Interface in order to implement some real-life use cases and to show some meaningful interaction between the database and external applications. The demo consists of a simple client-server application implemented in `node.js`, using `express.js` and the *MongoDB Node Driver* for the back-end and the `React` framework for the front-end.
The UI aims to simulate a real-life system for the verification of COVID-19 certificates: by inserting the 6-digits certification code into the system, the user will be able to verify whether the corresponding certificate is valid or not and to visualize the certificate's data. Furthermore, the UI provides some other functionalities to interact in various ways with the database, that are:

- verify whether a certificate is valid or not;
- insert a new vaccine or a new test in a given certificate;
- visualize a list of locations, their information and statistics.

7 User guide

In order to run the application, you need a MongoDB database with the provided data (see Section 4.2) and *Node.js* installed on the machine.
After having installed *Node.js*, follow these steps to run the application:

1. move to the `webapp` folder;
2. in the `server` folder, edit `config.env` inserting username, password and name of the database in the corresponding placeholders (remember to remove `'<'` and `'>'`);

3. open the terminal in the `webapp` folder;
4. to install dependencies and build the application, run `npm run build`. This command will install the necessary *node modules* and will build the front-end application;
5. to start the application, run `npm start`;
6. to use the application, open your browser and go to `http://localhost:5000`.

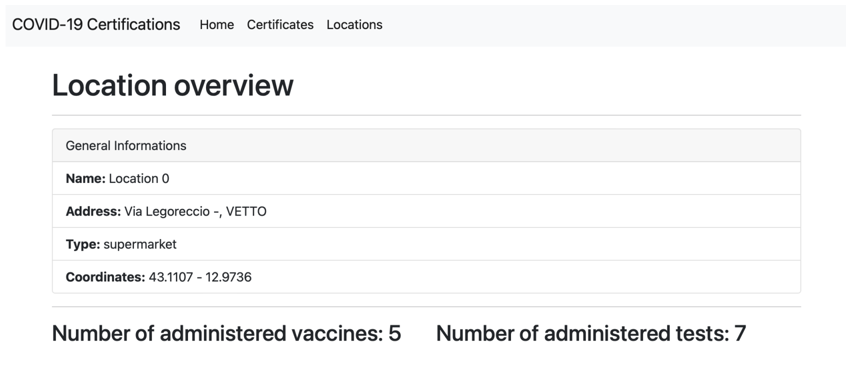
N.B.: if you are using macOS, you could be required to disable *Airplay Receiver* before running the code.

8 References and sources

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

9 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.



Location overview

General Informations

Name: Location 0

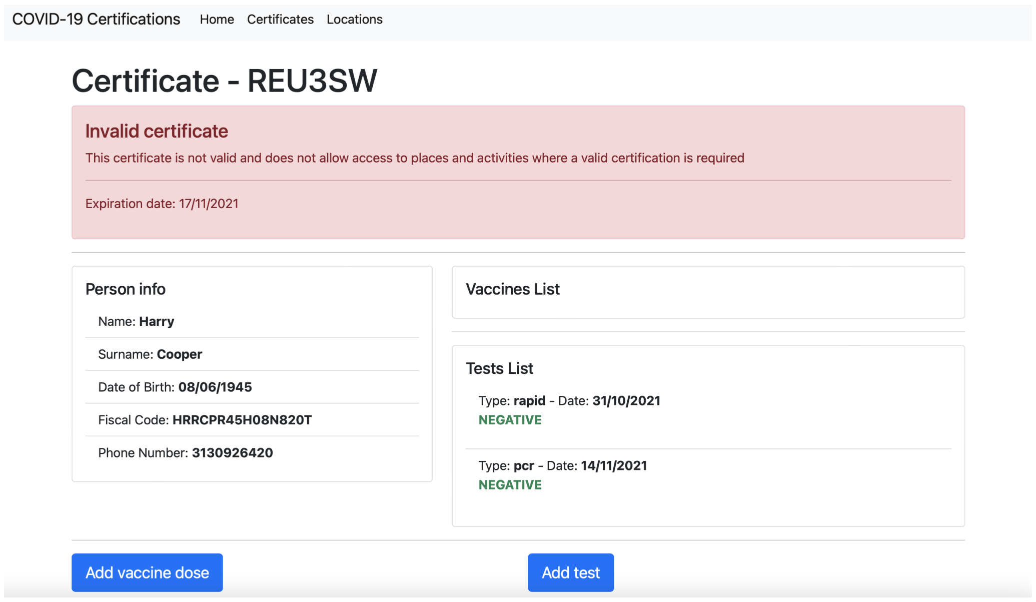
Address: Via Legoreccio -, VETTO

Type: supermarket

Coordinates: 43.1107 - 12.9736

Number of administered vaccines: 5   Number of administered tests: 7

Figure 1: User interface for visualizing a location in the database.



Certificate - REU3SW

Invalid certificate

This certificate is not valid and does not allow access to places and activities where a valid certification is required

Expiration date: 17/11/2021

Person info

Name: **Harry**

Surname: **Cooper**

Date of Birth: **08/06/1945**

Fiscal Code: **HRRCPR45H08N820T**

Phone Number: **3130926420**

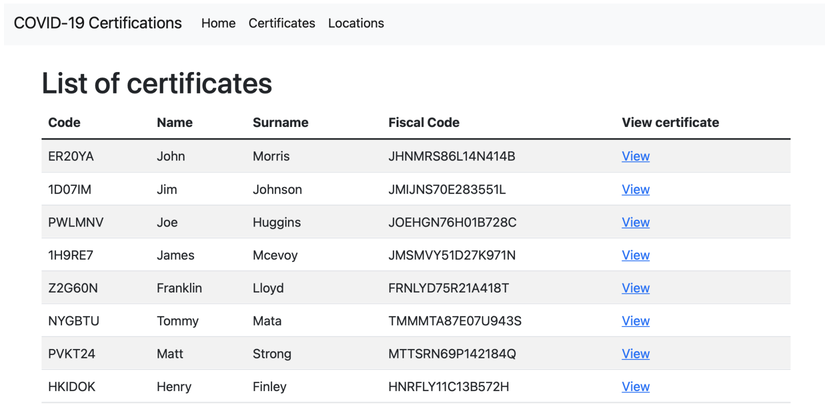
Vaccines List

Tests List

Type: **rapid** - Date: **31/10/2021**
NEGATIVE

Type: **pcr** - Date: **14/11/2021**
NEGATIVE

Figure 2: How the app shows the overview of a (invalid) certificate.



List of certificates

Code	Name	Surname	Fiscal Code	View certificate
ER20YA	John	Morris	JHNMRS86L14N414B	View
1D07IM	Jim	Johnson	JMIJNS70E283551L	View
PWLMNV	Joe	Huggins	JOEHGN76H01B728C	View
1H9RE7	James	Mcevoy	JMSMVY51D27K971N	View
Z2G60N	Franklin	Lloyd	FRNLYD75R21A418T	View
NYGBTU	Tommy	Mata	TMMMTA87E07U943S	View
PVKT24	Matt	Strong	MTTSRN69P142184Q	View
HKIDOK	Henry	Finley	HNRFLY11C13B572H	View

Figure 3: How the user interface shows the list of certificates.

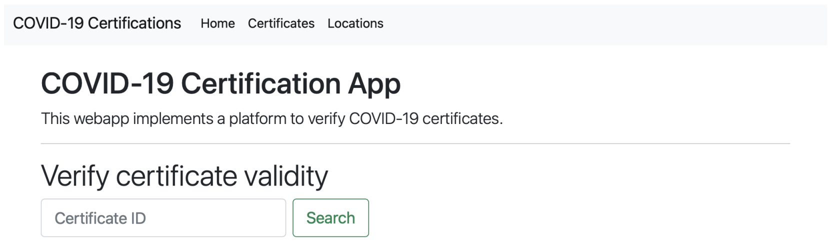

COVID-19 Certifications
Home
Certificates
Locations
COVID-19 Certification App
This webapp implements a platform to verify COVID-19 certificates.
Verify certificate validity
Certificate ID
Search

Figure 4: Homepage of the application.

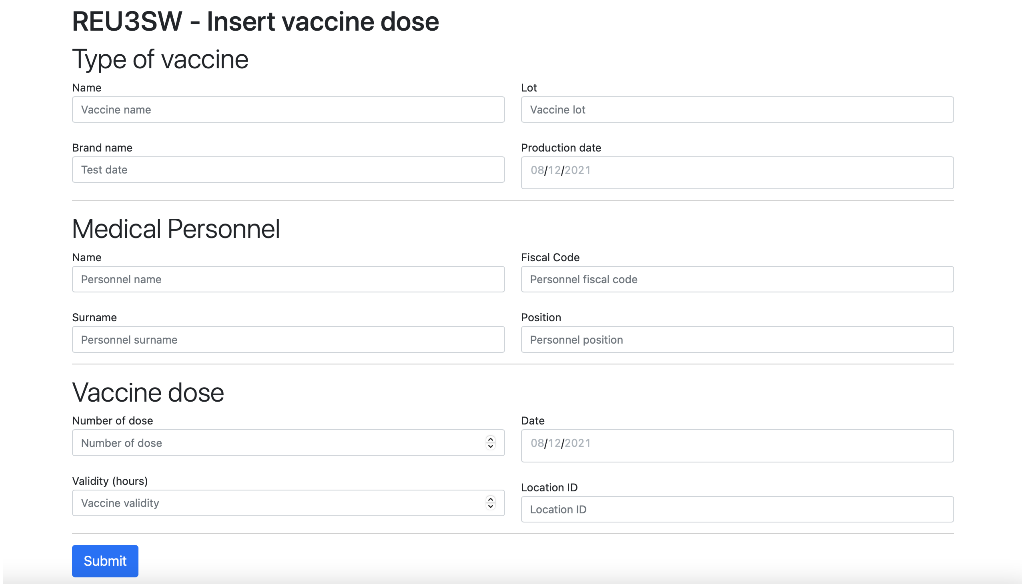


Figure 5: Page for the insertion of a new vaccine dose to a person.