Bioinformatics

Final documentation for project “MelanoQ”

A REST-based implementation for the Melanoma Questionnaire

Students

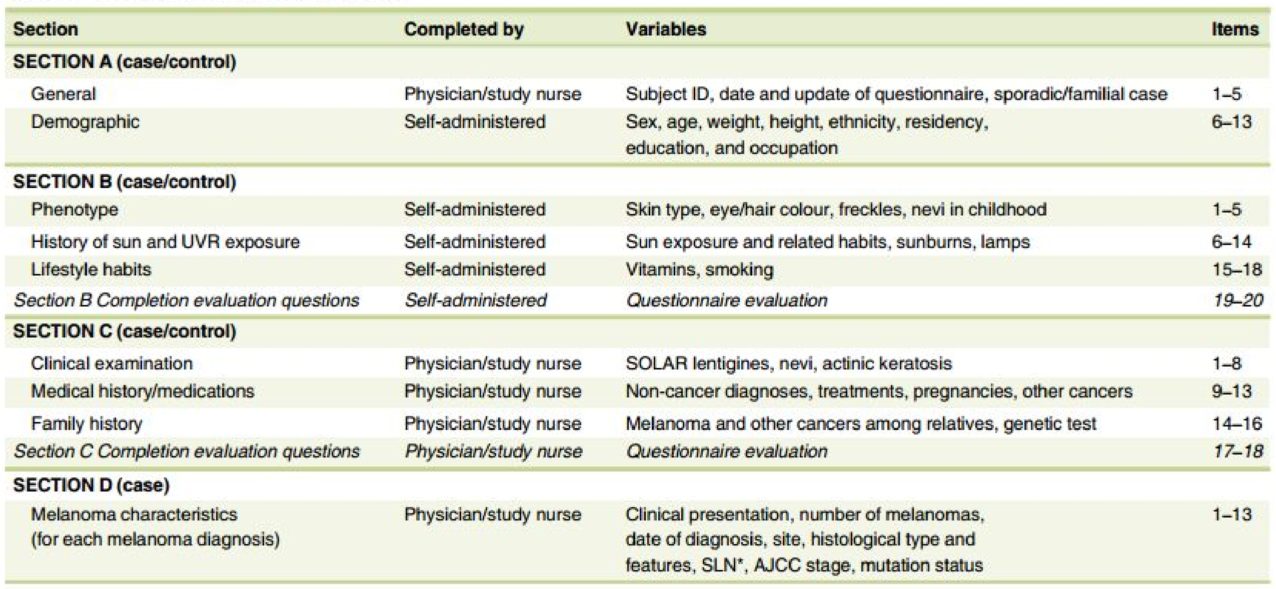
- Cerone Luigi, number: 261212
- D’Ascenzo Andrea, number: 261123

Introduction

MelanoQ is a questionnaire developed with the purpose of standardize the information gathering of melanomas. It has been created by a consortium composed of universities and research departments called **Melanostrum**.

The structure of the survey is the following:

Table 1 Design and variables of MelanoQ.



Section	Completed by	Variables	Items
SECTION A (case/control)			
General	Physician/study nurse	Subject ID, date and update of questionnaire, sporadic/familial case	1–5
Demographic	Self-administered	Sex, age, weight, height, ethnicity, residency, education, and occupation	6–13
SECTION B (case/control)			
Phenotype	Self-administered	Skin type, eye/hair colour, freckles, nevi in childhood	1–5
History of sun and UVR exposure	Self-administered	Sun exposure and related habits, sunburns, lamps	6–14
Lifestyle habits	Self-administered	Vitamins, smoking	15–18
Section B Completion evaluation questions	*Self-administered*	*Questionnaire evaluation*	*19–20*
SECTION C (case/control)			
Clinical examination	Physician/study nurse	SOLAR lentigines, nevi, actinic keratosis	1–8
Medical history/medications	Physician/study nurse	Non-cancer diagnoses, treatments, pregnancies, other cancers	9–13
Family history	Physician/study nurse	Melanoma and other cancers among relatives, genetic test	14–16
Section C Completion evaluation questions	*Physician/study nurse*	*Questionnaire evaluation*	*17–18*
SECTION D (case)			
Melanoma characteristics (for each melanoma diagnosis)	Physician/study nurse	Clinical presentation, number of melanomas, date of diagnosis, site, histological type and features, SLN*, AJCC stage, mutation status	1–13

For further information related to the meaning of the different fields you should refer to the official scientific paper available at : https://onlinelibrary.wiley.com/doi/full/10.1111/jdv.15208 .

One important thing to notice is that the information are organized in sections, some sections have to be filled directly by the case/control, other form the physician/study nurse.

Adopted technologies:

The team has decided to use a client/server paradigm based on a server that exposes a REST-based interface and a client that allows the final user to visualize the required information. The code (both client-side and server-side) is organized according to the MVC paradigm. In the following paragraphs we provide further information about the technologies used.

Server

The server-side code is written in Java by using the Spring Boot framework that allowed us to develop a stable, MVC-based application with ease and with lots of integration with different plugins. The backend is responsible of handling the storage and retrieving of the information in a database. One requirement from the client was to utilize a noSQL database, the team has decided to use MongoDB. The reasons behind this choice are the knowledge of the team members about this tool and the existence of a plugin for Spring Boot specific for the integration of Spring with MongoDB databases. As far as concerned the authentication and the security of the user credential management, the team has used the JSON Web Token standard. This technology expects that the end-user provides valid credentials to login in the system, then it receives a digitally signed token with an expiry date that must be inserted in each following request made to the server. If the token contained is the request is valid the server responds to the client, otherwise an exception is launched.

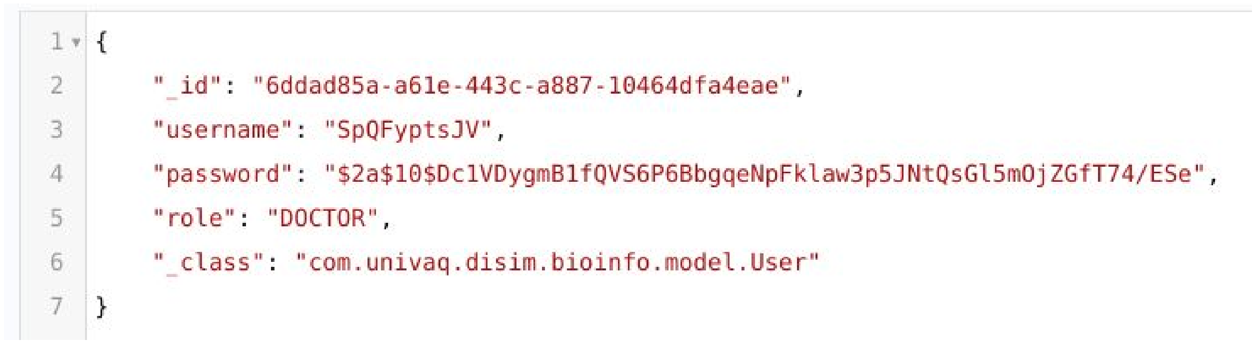
Client

The client-side code is written in Javascript with the Angular framework. The team has decided to use this technology for multiple reasons, the most important are that it has been already used for developing both web client and Single-page Web application and because it is a very popular framework that perfectly fitted the project requirements. Also the source code in Angular has to be written in TypeScript, a typed Javascript.

Data Model

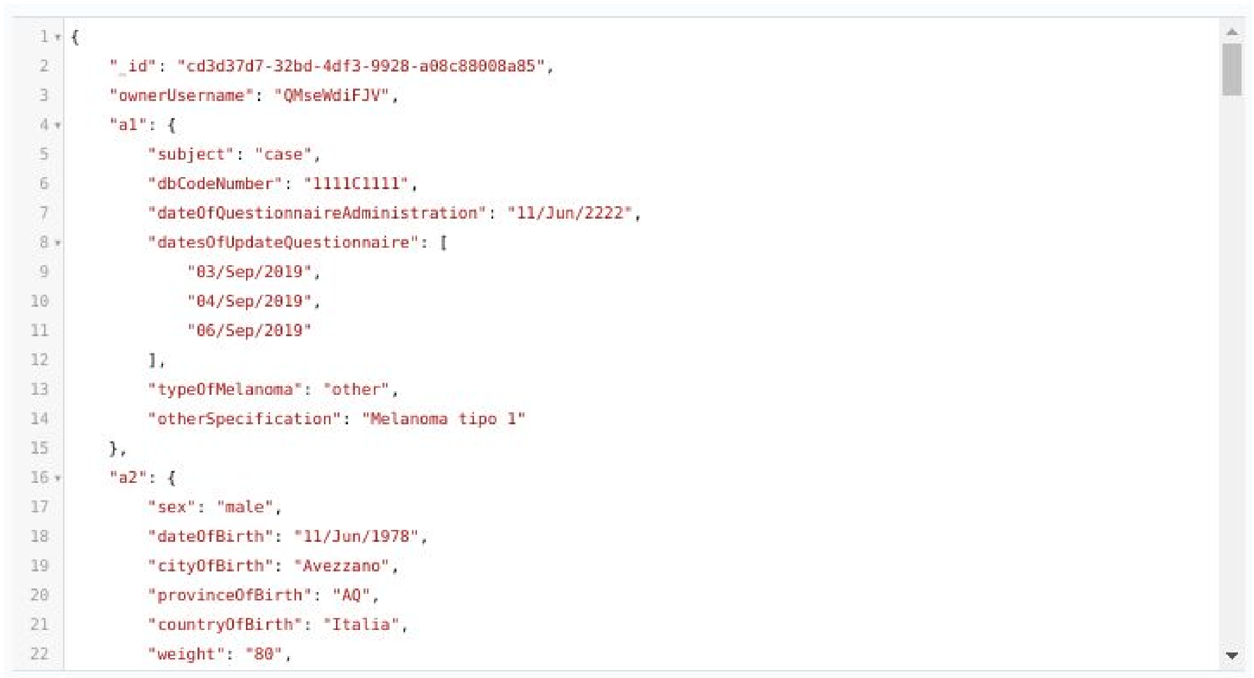
Given the fact that we are using a noSQL database, the information are inserted into a document in the database. The whole database is composed by two different collections: one is called “user” and contains the information of a specific user with, for example, its username and password. The other is the collection “questionnaire” that contains the questionnaire associated to a specific user.

The structure of a user is the following:



```
{
    "_id": "6ddad85a-a61e-443c-a887-10464dfa4eae",
    "username": "SpQFyptsJV",
    "password": "$2a$10$Dc1VDygmB1fQVS6P6BbgqeNpFklaw3p5JNtQsGl5mOjZGfT74/ESe",
    "role": "DOCTOR",
    "_class": "com.univaq.disim.bioinfo.model.User"
}
```

The structure of a questionnaire is the following:



```
{
    "_id": "cd3d37d7-32bd-4df3-9928-a08c88008a85",
    "ownerUsername": "QMseWdiFJV",
    "a1": {
        "subject": "case",
        "dbCodeNumber": "1111C1111",
        "dateOfQuestionnaireAdministration": "11/Jun/2222",
        "datesOfUpdateQuestionnaire": [
            "03/Sep/2019",
            "04/Sep/2019",
            "06/Sep/2019"
        ],
        "typeOfMelanoma": "other",
        "otherSpecification": "Melanoma tipo 1"
    },
    "a2": {
        "sex": "male",
        "dateOfBirth": "11/Jun/1978",
        "cityOfBirth": "Avezzano",
        "provinceOfBirth": "AQ",
        "countryOfBirth": "Italia",
        "weight": "80",
```

Code

Server

The JAVA code is organized in packages:

- model: It contains all the class that describe the entities of our project such as the User, the Questionnaire, the different Sections and a subpackage, called nested, that contains all the classes used to describe the complex object contained in the various sections.
- controller: It contains the classes that handle the data of our application, there are the AuthenticationController, the QuestionnaireController and the QueryController.
- service: It's the business layer of our application, there are different services for the various sections that compose one questionnaire. In a service class we can find the logic for the CRUD operation and other operations performed on the data.
- repository: It contains all the logic related to the database logic used to communicate with MongoDB.
- configuration: It contains all the classes needed for the JSON Web Token authentication such as the creation of the token, the validation and so on.

Endpoints

Our implementation is REST-based, so it offers different endpoints, in the following we list them:

- GET /api/questionnaire/user/{username}: Return the questionnaire associated to the user with username equals to given username.
- POST /api/questionnaire/user/{username}/{section}: Upsert the section in the questionnaire associated to user with username equals to username. If it is not found the questionnaire is created, if the section is already present it performs an update on the previous information.
- GET /api/questionnaire/user/{username}/{section}: Return the section of the questionnaire associated to user with username equals to username. If not found, it return null.

- POST /api/auth/login: Used for the login, if the provided credentials are valid this method return a valid JWT token.
- POST /api/auth/signup: If the provided credential are valid (es. no duplicated information), a new user instance is created in the db.ù
- POST /api/query: Given a query in the body of the request, this endpoint returns the result based on the data stored in the database.

Client

The angular project in the directory code/gui of our application. In the subfolder code/gui/src/app there is the code related to our web client application. The folders are organized in the following way:

- model: It contains all the class that describe the entities of our project such as the User, the Questionnaire and the different Sections.
- guards: It contains the “guards” that angular uses to handle the permission access to the different routes of the application.
- services: It contains all the classes used to interact with the REST-based interface provided by the server.
- pages: It contains the different pages of our application with its components.
- interceptors: It contains an interceptor that basically adds the JWT token to the header of each request made by our client app.

The client application supports is multi language, the final user can choose between english and italian.

Deployment

If you want to run our application you have to follow these steps:

- Download and install all the tools used to develop the application: npm, angular-cli, mongodb, maven, java.
- In order to run the client side:
  - git clone https://github.com/LuigiCerone/BioInfo_Survey.git
  - cd BioInfo_Survey/code/gui

    - npm install
    - ng serve
- In order to run the server side:
    - git clone https://github.com/LuigiCerone/BioInfo_Survey.git
    - cd BioInfo_Survey/code/rest
    - mvn spring-boot:run