Table of Contents

Introduction

Many melanoma observational studies have been carried out across different countries and geographic areas using heterogeneous assessments of epidemiologic risk factors and clinical variables.
The aim is to develop a consensus questionnaire to standardize epidemiologic and clinical data collection for melanoma risk
assessment.
Melanoq is the product of the work of the MelanoStrum Consortium.

This project's goal is to implement a digital version of the aforementioned questionnaire by providing a way to *compile* the questionnaire itself and to expose the physicians, researchers and authorized personnel a way to *query* the data stored in the questionnaires' database.

Adopted technologies

To offer the best decoupling and deployment flexibility we decided to follow the classic **MVC** web-application architectural pattern by developing a **REST Server** and a **Web Client**, while storing the data on a **NoSQL documental database**.
The excution flow follows the MVC paradigm: the data stored in the database is presented to the Web Client, from it it is possible to invoke functions on the REST Server that will then modify the aforementioned data, updating the Web Client view.

The reasons by which we decided to adopt a certain kind of tool or technology is explained in the following paragraphs.

Database

CouchBase Server NoSQL database: since the project aims to store a large and complex questionnaire for each patient adhering to the supervision programme of the melanoma cancer, we tried to find a database that was able to store a large amount of data whithout significantly changing the structure of the questionnaire itself. Couchbase is a documental NoSQL database whose datastructure is not composed by columns and rows as in the relational one, but documents are stored in the database in `.json` format, that is a simple, lightweight notation, compact and human readable. In Couchbase, a document usually represents a single instance of an object in the application code and each document can contain nested structures (sub-documents). This allows developers to naturally express many-to-many relationships without requiring a "reference table" or "junction table". So in this database we can store the entire questionnaire and all its sub-sections in an easy way and as naturally as possible. Morevover, the nested structure comes in handy when a certain degree of kinship is sought.
Morover, Couchbase offers the possibility distribute the storage: because of Couchbase's architecture, an application can be developed at small scale, but also deployed to a distributed cluster, or separate clusters of varied topologies, without any architecture or behavioral changes to that application. This feature seemed to us to be very advantageous for a future distributed deployment of the application, as it emerged from the given documentation, this study was carried out in collaboration between three countries: Italy, Greece and Spain with the possibility of extension to other countries. In addition, each country has a network of centres where the collection of questionnaires is made, which could in turn be distributed in the country.
The last feature that made us decide to adopt this technology is its query lenguage: `N1QL` (pronounced "*nickel*") is Couchbase's next-generation query language. `N1QL` aims to meet the query needs of distributed document-oriented databases. It is very similar to `SQL`. `N1QL` gives application developers an expressive, powerful, and complete declarative language with industry standard ANSI joins for querying, transforming, and manipulating `.json` data – just like `SQL`.

REST Server

Spring: it is the standard-de-facto Java framework for what concerns the development of applications based on the MVC architectural pattern; it also comes with a generous amount of plugins that allow to easly connect to several types of database, deploy security filters, etc.
The following are the components that have been exploited:

- **Spring Boot**: Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". It directly embeds the principal web servers without need to deploy WAR files, provides opinionated 'starter' dependencies to simplify the build configuration, configure Spring and 3rd party libraries whenever possible in automatic way and it needs no code generation and no requirement for XML configuration.

- **Spring Data JPA for CouchBase:** Spring Data for Couchbase is part of the umbrella Spring Data project which aims to provide a familiar and consistent Spring-based programming model for new datastores while retaining store-specific features and capabilities. The Spring Data Couchbase project provides integration with the Couchbase Server database. Key functional areas of Spring Data Couchbase are a POJO centric model for interacting with Couchbase Buckets and easily writing a Repository style data access layer. It has been choosed to use this framework because of the many features provided (configuration support using Java based *Configuration* classes or an XML namespace for the Couchbase driver, *CouchbaseTemplate* helper class that increases productivity performing common Couchbase operations, annotation based mapping metadata but extensible to support other metadata formats, automatic implementation of *Repository* interfaces including support for custom finder methods...) and because already known and used before by the components of the group.

- **Spring Security and Json Web Token-based authentication**: Spring Security is a powerful and highly customizable authentication and access-control framework. It is a framework that focuses on providing both authentication and authorization to Java applications. The real power of Spring Security is found in how easily it can be extended to meet custom requirements.
  JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains

Web Client

VueJS: it is yet another Javascript framework that recently exploded in popularity; it is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.
In a nutshell: Vue allows to bind the `.html` view to the data model, and to interactively modify the second by interacting with the first. Obviously, as explained in the following sections, Vue actually offers way more powerfull tools, such as a routing engine, local store, etc.

Data Model

The structure of how each questionnaire is stored is the following `.json` format:

```
{
    ...

    "a2": {
        "education": "text",
        "ethnicity": "text",
        "currentOccupationalStatus": "text",
        "sex": "text",
        "weight": "numeric",
        "historyOfOccupations": {
            "occupationStartingTime": "text",
            "occupationEndingTime": "text"
        },
        "provinceOfBirth": "text",
        "countryOfBirth": "text",
        "residencies": {
            "residencyEndingTime": "text",
            "residencyStartingTime": "text",
            "residencyCountry": "text",
            "residencyCity": "text"
        },
        "height": "numeric"
    },
    ...
}
```

We tried to build a 1 to 1 mapping between the data model and the given questionnaire representation:

- the entire questionnaire is between the curly brackets;
- inside, each questionnaire has a certain number of "childrens" that match to the sections of the original questionnaire representation;
- each child is identified by a string representig the name of the respective section (so section A1 is identified by `a1`, section A2 is identified by `a2` and so on.);
- each section has a certain number of fields, where the left side is the identifier of the field that is in the given questionnaire model and the right side is the value that contains the answer to that specific field. We can have different type of values:
    - `text` (see for example `a2.education`),
    - `numeric` (see for example `a2.weight`)
    - lists that we represent as child components (see for example `a2.residencies`).

To access a particular field it is simply needed to follow the key hierarchy; e.g., to access the field `occupationStartingTime` of the section `a2`, we need to type `a2.historyOfOccupations.occupationStartingTime`.

REST Server

Structure

The server side of our project is structured with the following packages: `model`, `controller`, `repository` and `service`.
In the `model` package there is the "description" of the entities used by the application:

- `User` that is the representation of user's types that can log in in the application and add the questionnaires or consult them.
- `Questionnaire` where is explained of which sections a questionnaire is composed.
- `Query` is the model of how the application can be questioned.
- `ErrorMessage` is the structure that will have any error message in the application.
- `section` subpackage contains all the models of the all sections of which a questionnaire is composed by (i.e., section A1, A2, B1, B2, B3, C1, C2, C3, D and Evaluation).
- `nested` subpackage contains all the complex objects nested in the various section.

The `controller` orchestrates the data flow of the entire application:

- Trough `QuestionnaireController` and `UserController` we indicate the CRUD operations to perform on data in the database.
- `Response` defines the format of how a general response from the server to the client must be.
- `JwtAuthenticationController` is the controller that manage the authentication of the users in the application.
- `ExceptionControllerAdvice` deals with the management of the exceptions throw, eventually, by the functioning of the application.

The `repository` package contains all the interfaces needed to communicate with the database, these interfaces will be automatically implemented by the Spring Data JPA module.

The `service` package contains the business logic of the application: it contains all the sections' services and the user service, where are specified the CRUD operation to perform on the database and the server internal logic.

In addition to the packeges characterising the functionnalities of the application, there are:

- the `configuration` package whose contains all the classes necessarely to configure the application to the use of the additional modules;
- the `BusinessException` class that handles the exeptions of the application;
- the `Application` class that is the starting point of the entire application.

REST Endpoints

- GET `/questionnaire/{dbCodeNumber}`:
  retrives the questionnaire identified by `dbCodeNumber`.
- POST `/questionnaire/query`:
  retrives all the questionnaires in the database according to the filters setted in the query.
- GET `/questionnaire`:
  retrives all the questionnaires.
- GET `/questionnaire/validate/{dbCodeNumber}`:
  verifies if the given `dbCodeNumbe` exists or not.
- POST `/questionnaire/{dbCodeNumber}/{section}`:
  inserts the section indicated in the `section` parameter to he questionnaire indentified by the `dbCodeNumber`.
- GET `/questionnaire/{dbCodeNumber}/{section}`:
  returns the section indicated in the `section` parameter of the questionnaire indentified by the `dbCodeNumber`.
- PATCH `/questionnaire/{dbCodeNumber}/{section}`:
  updates the section indicated in the `section` parameter to he questionnaire indentified by the `dbCodeNumber`.
- DELETE `/questionnaire/{dbCodeNumber}/{section}`:
  deletes the section indicated in the `section` parameter to he questionnaire indentified by the `dbCodeNumber`.
- POST `/user`:
  inserts an user in the database.
- GET `/user/{username}`:
  returns the user whose username matches with the given `username`.
- POST `/authentication`:
  performs the authentication of a user and assigns it a token.

Web Client

The **web client** has been implemented using the VueJS framework.
The internal project structure is as follows:

- **assets**: contains the static resources, such has the stylesheets, images, etc.
- **common**: contains the services that are accessed by rest of the application, including the api services for the connection to the api server, the locale serviec that allows to switch between the various languages, the authentication service that is used to store the token and the logged user's data on the browser's local storage (so to survive to page refreshes and between sessions), etc.
- **components**: here are contained the components used by VueJS to build the various views, these components can be either used as singletons (i.e., they appear at most one time in a given view, these components are prefixed by the `The` keyword, such as the Header component), or classic components that can be used as many times as required in a given view (such as the Message component)).
- **locales**: contains the `.json` files that store the translation data (the key of an entry of these files points to a word/phrase to be translated, the value of that key contain the respective translation).
- **plugins**: here are located the VueJS plugins (*datepicker*, *bootstrap*, *vuelidate*, etc.).
- **router**: since VueJS implements the *single page web application* design pattern, there is the need to have a structured way to navigate between the views, this is achieved by the *router* plugin, this directory contains its configuration: a series of paths pointing to the respective view files.
- **store**: to survive between views data needs to be stored in some way, this directory contains the services that are responsible of implementing the VueJS *vuex* store interface. In these files there is a *model* description of the data, several *action* methods and several *mutation* methods; *actions* expose asynchronous entry points to *mutation*, *mutation* modify the data contained in the model (the cycle is the following: a component invokes an *action*, the *action* triggers a *mutation*, the *mutation* modifies the *model* data, the component sees the modified data; all of this while the component is free to perform other actions, since the call to the action is asynchronous).
- **views**: this directory contains the VueJS views; a *view* is a collection of components plus some stand-alone template code, the template is followed by the logic of the view (what happens when the user presses a certain button, etc.).
- **App.js**: is the entry point of the application and the first view.
- **main.js**: is the glue that connects al the various components, views, plugins, etc.

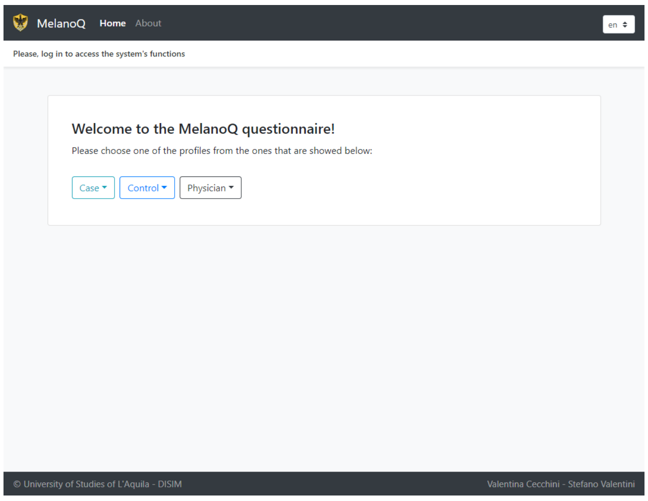
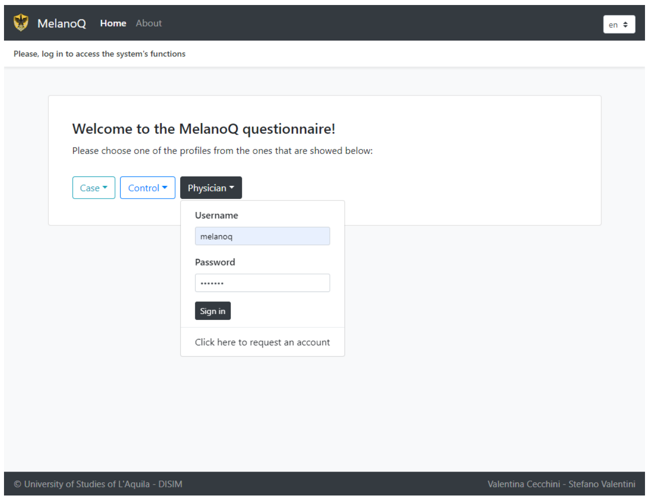
Views

All the screenshots that are showed here can be found inside the `/docs/views/` directory.

Home

This is the view that is presented to the user at the first access: here the system prompts the user to log in to access the functionnalities.
The user can log in with its credientals using three different *profiles*, the credentials are the same but the choosen profile will influnce the functionnality to which the user can access during its session: if a physician needs to have a patient (maybe a control one) to fill a questionnaire section then it can log in with its physician credentials but with the *control* profile.

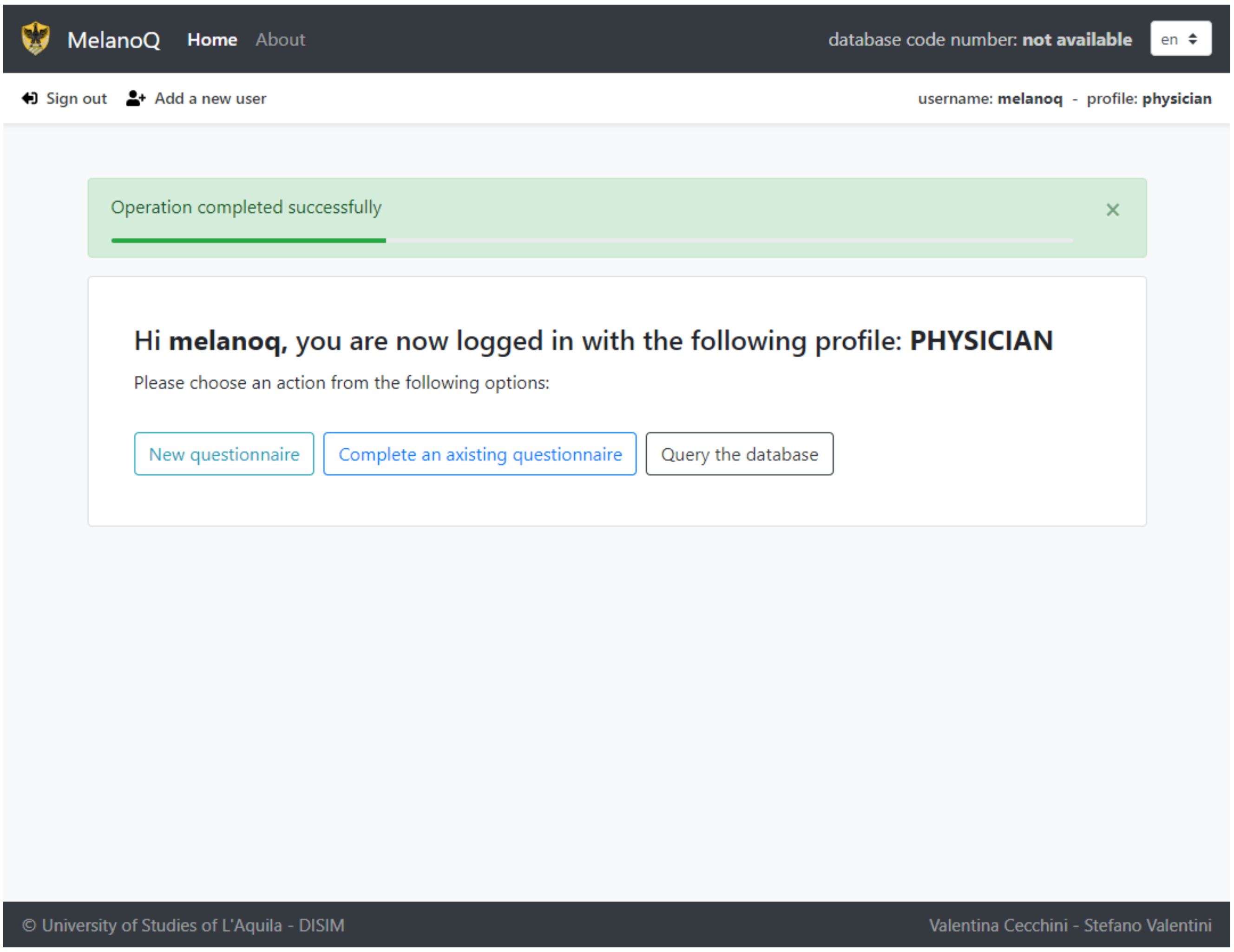
After having logged in, a series of actions become available, based on the choosen profile (case/control profiles cannot access to the query engine and database code number retival).

Home view as guest	Home view: log in
	

MelanoQ
Home
About
en
Please, log in to access the system's functions
Welcome to the MelanoQ questionnaire!
Please choose one of the profiles from the ones that are showed below:
Case
Control
Physician
© University of Studies of L'Aquila - DISIM
Valentina Cecchini - Stefano Valentini

MelanoQ
Home
About
en
Please, log in to access the system's functions
Welcome to the MelanoQ questionnaire!
Please choose one of the profiles from the ones that are showed below:
Case
Control
Physician
Username
melanoq
Password
•••••••
Sign in
Click here to request an account
© University of Studies of L'Aquila - DISIM
Valentina Cecchini - Stefano Valentini

Home view as logged user



MelanoQ
Home
About
database code number: **not available**
en
Sign out
Add a new user
username: **melanoq** - profile: **physician**

Operation completed successfully

Hi **melanoq,** you are now logged in with the following profile: **PHYSICIAN**

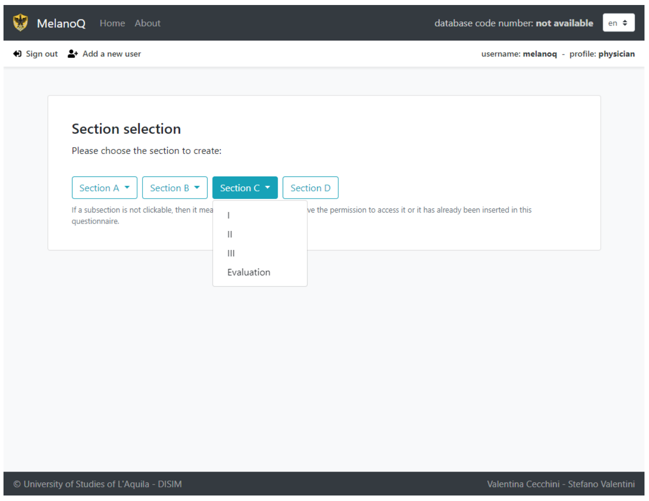
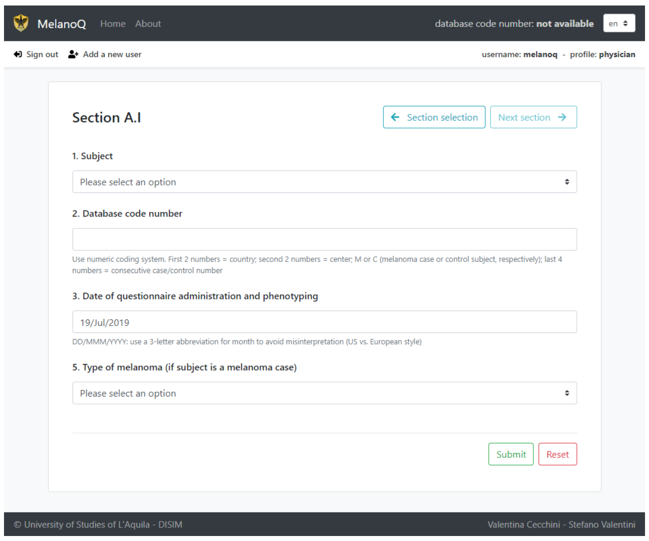
Please choose an action from the following options:

New questionnaire
Complete an axisting questionnaire
Query the database
© University of Studies of L'Aquila - DISIM
Valentina Cecchini - Stefano Valentini

Section submission

Based on which section has already been inserted and on which profile the user is currently logged in a series of section will be available.

After having choosen a section, the user will be redirected to the respective form (if, for any reason, a user with an invalid profile reaches a restricted access form section, it is automatically redirected to the home page with an error message).
Each form section has a validator function for each field, so to avoid the insertion of invalid data.
Once the form has been submitted the user can follow the top-right corner arrow to reach the next section to fill, if authorized.

Section selection	**Section form example**
	

MelanoQ Home About database code number: **not available** en
Sign out Add a new user username: **melanoq** - profile: **physician**

Section selection
Please choose the section to create:
Section A Section B Section C Section D
I
II
III
Evaluation
If a subsection is not clickable, then it mea ve the permission to access it or it has already been inserted in this questionnaire.

© University of Studies of L'Aquila - DISIM Valentina Cecchini - Stefano Valentini

MelanoQ Home About database code number: **not available** en
Sign out Add a new user username: **melanoq** - profile: **physician**

Section A.I
Section selection Next section

1. Subject
Please select an option

2. Database code number

Use numeric coding system. First 2 numbers = country; second 2 numbers = center; M or C (melanoma case or control subject, respectively); last 4 numbers = consecutive case/control number

3. Date of questionnaire administration and phenotyping
19/Jul/2019
DD/MMM/YYYY: use a 3-letter abbreviation for month to avoid misinterpretation (US vs. European style)

5. Type of melanoma (if subject is a melanoma case)
Please select an option

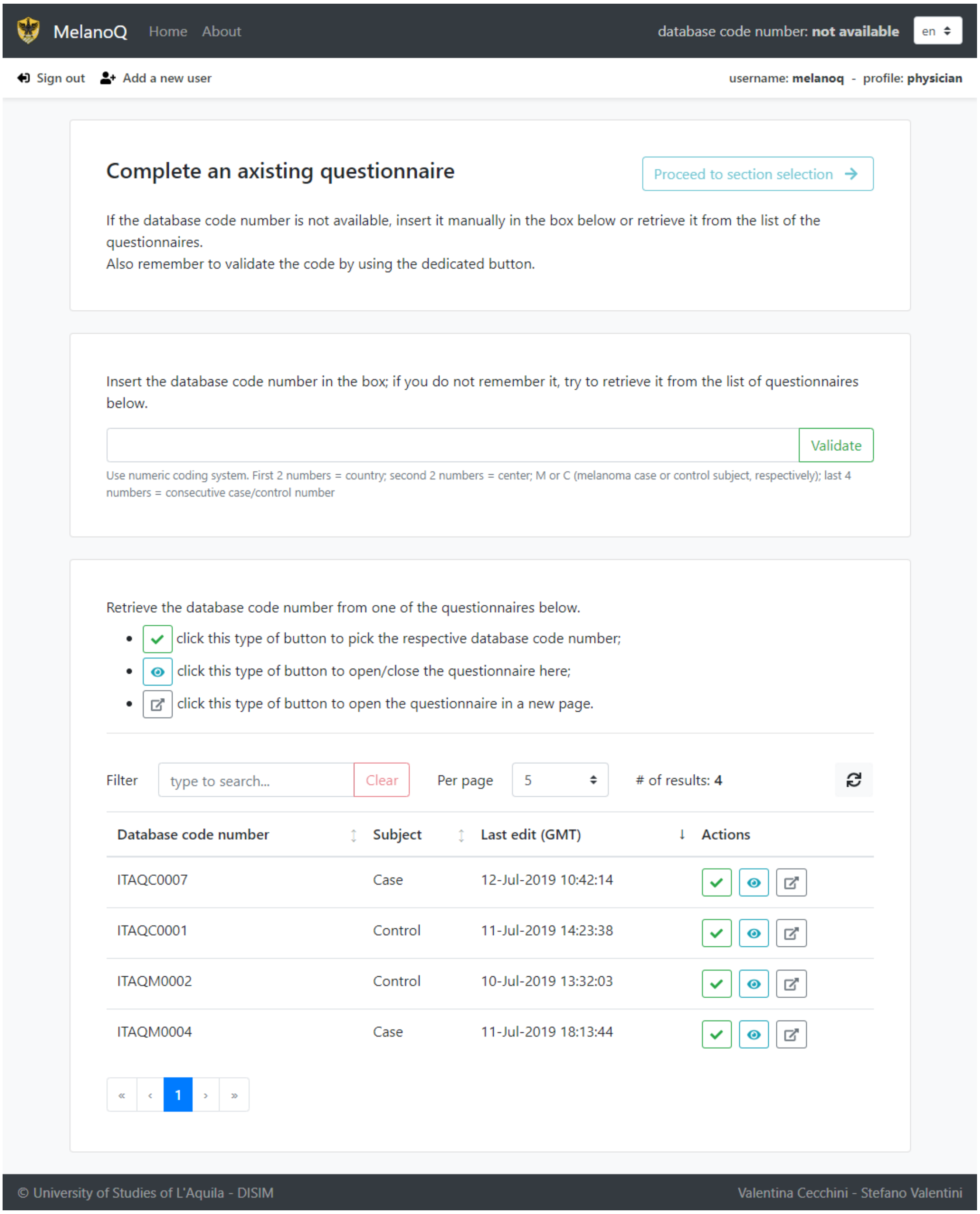
Submit Reset

© University of Studies of L'Aquila - DISIM Valentina Cecchini - Stefano Valentini

To access a section sumbission form it is needed that the questionnaire database code number of the questionnaire that the user is trying to fill is stored in the session (with an exception for the section A1, that is the one that creates the database code number); this can happen in two ways:

- either the subsequent forms are submitted in the same session as the section A1 (the database code number is stored after having submitted a new A1 section)
- or the user has to either manually write the database code number or it can choose it from the list of the questionnaires whose section A1 has already been submitted.

Database code number retival



database code number: **not available**

username: **melanoq** - profile: **physician**

Complete an axisting questionnaire

If the database code number is not available, insert it manually in the box below or retrieve it from the list of the questionnaires.
Also remember to validate the code by using the dedicated button.

Insert the database code number in the box; if you do not remember it, try to retrieve it from the list of questionnaires below.

Use numeric coding system. First 2 numbers = country; second 2 numbers = center; M or C (melanoma case or control subject, respectively); last 4 numbers = consecutive case/control number

Retrieve the database code number from one of the questionnaires below.

- ✓ click this type of button to pick the respective database code number;
- click this type of button to open/close the questionnaire here;
- click this type of button to open the questionnaire in a new page.

Filter: type to search... Per page: 5 # of results: **4**

Database code number	Subject	Last edit (GMT)	Actions
ITAQC0007	Case	12-Jul-2019 10:42:14	
ITAQC0001	Control	11-Jul-2019 14:23:38	
ITAQM0002	Control	10-Jul-2019 13:32:03	
ITAQM0004	Case	11-Jul-2019 18:13:44	

© University of Studies of L'Aquila - DISIM Valentina Cecchini - Stefano Valentini

Query builder

After having successfully logged in with the physician profile, it is possible to access the query builder view: here the user can create its query by composing a series of *rules*; all the added rules can be considered as *any of them must be valid* or *all of them must be valid*. It is possible to add one or more rules for each one of the fields of the questionnaire.

The following operators are available for the **textual** fields (all *case insensitive*):

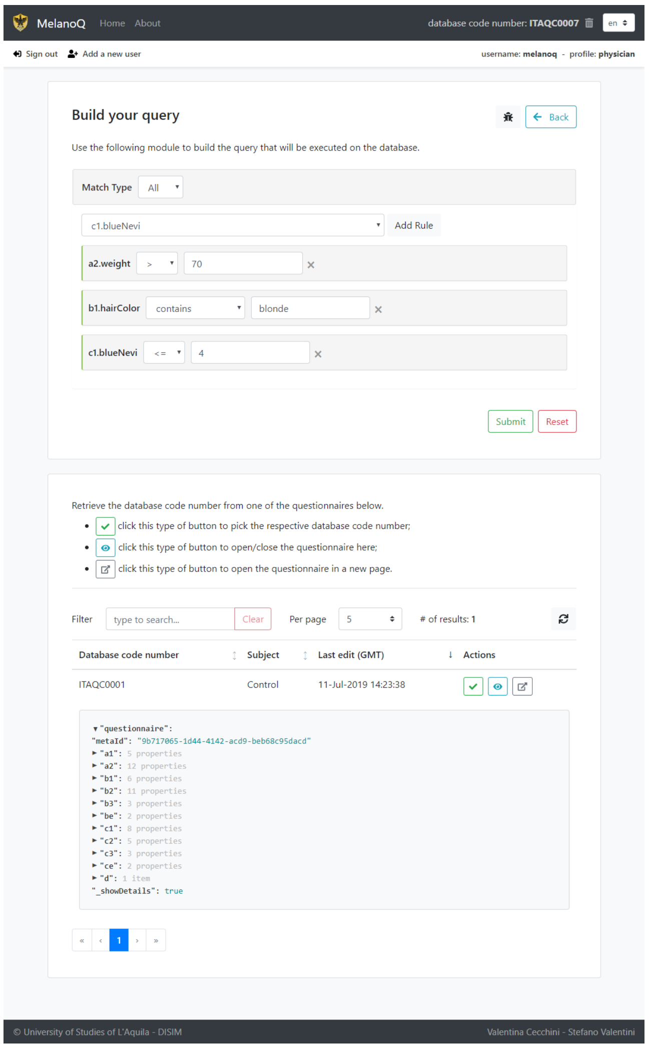
- `contains`: *true* if the field contains the submitted value inside it
- `does not contain`: when *contains* is not *true*
- `equals`: *true* when the submitted value is *exactly* the same as the value contained in the field
- `does not equal`: when *equals* is not *true*

The following operators are available for the **numeric** fields:

- `=` (equals)
- `<>` (does not equal)
- `>` (greather than)
- `>=` (greather or equal than)
- `<` (lesser than)
- `<=` (lesser or equal than)

Once the query has been submitted it is possible to see the results in the table at the bottom of the page. It is also possible to see the resulting questionnaires direcly in the same page as the query builder or to open them in a new page.

Query builder and local view



database code number: **ITAQC0007**

username: **melanoq** - profile: **physician**

Build your query

Use the following module to build the query that will be executed on the database.

Match Type: All

c1.blueNevi

a2.weight	>	70
b1.hairColor	contains	blonde
c1.blueNevi	<=	4

Retrieve the database code number from one of the questionnaires below.

- ✓ click this type of button to pick the respective database code number;
- click this type of button to open/close the questionnaire here;
- click this type of button to open the questionnaire in a new page.

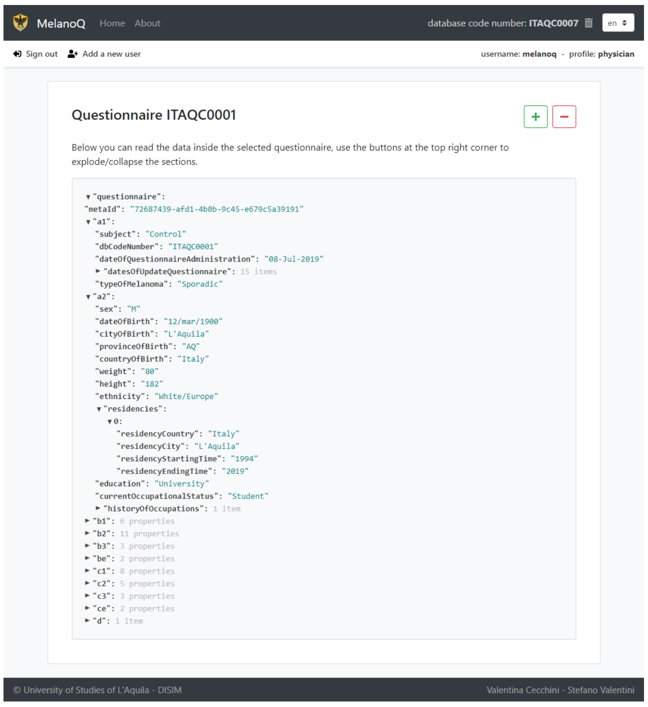
Filter: type to search... Per page: 5 # of results: **1**

Database code number	Subject	Last edit (GMT)	Actions
ITAQC0001	Control	11-Jul-2019 14:23:38	

```
▼"questionnaire":
"metaId": "9b717065-1d44-4142-acd9-beb68c95dacd"
► "a1": 5 properties
► "a2": 12 properties
► "b1": 6 properties
► "b2": 11 properties
► "b3": 3 properties
► "be": 2 properties
► "c1": 8 properties
► "c2": 5 properties
► "c3": 3 properties
► "ce": 2 properties
► "d": 1 item
"_showDetails": true
```

© University of Studies of L'Aquila - DISIM   Valentina Cecchini - Stefano Valentini

Standalone view



database code number: **ITAQC0007**

username: **melanoq** - profile: **physician**

Questionnaire ITAQC0001

Below you can read the data inside the selected questionnaire, use the buttons at the top right corner to explode/collapse the sections.

```
▼"questionnaire":
"metaId": "72687439-afd1-4b0b-9c45-e679c5a39191"
▼"a1":
  "subject": "Control"
  "dbCodeNumber": "ITAQC0001"
  "dateOfQuestionnaireAdministration": "08-Jul-2019"
  ► "datesOfUpdateQuestionnaire": 15 items
  "typeOfMelanoma": "Sporadic"
▼"a2":
  "sex": "M"
  "dateOfBirth": "12/mar/1900"
  "cityOfBirth": "L'Aquila"
  "provinceOfBirth": "AQ"
  "countryOfBirth": "Italy"
  "weight": "80"
  "height": "182"
  "ethnicity": "White/Europe"
  ▼"residencies":
    ▼0:
      "residencyCountry": "Italy"
      "residencyCity": "L'Aquila"
      "residencyStartingTime": "1994"
      "residencyEndingTime": "2019"
  "education": "University"
  "currentOccupationalStatus": "Student"
  ► "historyOfOccupations": 1 item
► "b1": 6 properties
► "b2": 11 properties
► "b3": 3 properties
► "be": 2 properties
► "c1": 8 properties
► "c2": 5 properties
► "c3": 3 properties
► "ce": 2 properties
► "d": 1 item
```

© University of Studies of L'Aquila - DISIM   Valentina Cecchini - Stefano Valentini

Deployment

Requirements

- CouchBase Server - https://www.couchbase.com/
- Apache Tomcat - http://tomcat.apache.org/
- Apache Maven - https://maven.apache.org/
- Yarn - https://yarnpkg.com/en/ or NPM (Node Package Manager) - https://www.npmjs.com/
- VueJS CLI - https://cli.vuejs.org/

Database

Download and install CouchBase Server Community or Enterprise from here: https://www.couchbase.com/downloads.

After having installed and run it, its web interface will be available at the address http://machine-ip:8091, configure the server according to your preferences and then create a user and a bucket (remember that the bucket name and the user's credentials must then be provided to the MelanoQ api server), finally add all the permissions to the created user for the created bucket.

Go to the `Query` section of the web interface and run:

```
CREATE PRIMARY INDEX ON bucketname USING VIEW;
```

where `bucketname` is the name of the created bucket.

Now the database is empty, so you need to manually add an admin user for the MelanoQ api server:

- generate a `bcrypt` password from this online tool: https://www.browserling.com/tools/bcrypt using *12 rounds* (be sure to check that the generated encrypted password starts with the `$2a` prefix)

- execute the following query:

```
INSERT INTO bucketname (KEY, VALUE)
 VALUES
 (
    "k999",
    {
        "role": "admin",
        "username": "name",
        "password": "generated-password",
        "_class": "it.univaq.disim.bioinformatics.melanoq.model.User"
     }
 )
```

where `k999` is any random alphanumeric string, `name` is a name of your choice and `generated-password` is the bcrypt password you previously generated.

If you need to backup the database content run:

```
> cbbackup http://machine-ip:8091 path-to-dest-folder -u username -p password -b
bucket-name
```

if you need to restore a backup run:

```
> cbrestore path-to-source-folder http://machine-ip:8091 -u username -p password -
b bucket-name -x rehash=1
```

where `cbbackup` and `cbrestore` are tools that are located in the installation directory of CouchBase Server.

REST Server

```
> git clone https://gitlab.com/valent0ne/melanoq.git
> cd ./melanoq/code/api
> mvn clean package
```

Now inside the directory `./melanoq/code/api/target/` there will be a file named `melanoq-x.y.z.jar`, move this file inside your *Apache Tomcat* `webapp` directory.
The MelanoQ api server will be available at the address http://machine-ip:8080/melanoq/api. To change the application parameters (e.g., couchbase server credentials and address), before executing the command `mvn clean package` edit the file `application.properties` located in the directory `./melanoq/code/api/src/main/resources`.

Web Client

```
> git clone https://gitlab.com/valent0ne/melanoq.git
> cd ./melanoq/code/webclient
> yarn
> yarn build
```

or (if you are using `npm` instead of `yarn`):

```
> git clone https://gitlab.com/valent0ne/melanoq.git
> cd ./melanoq/code/webclient
> npm
> npm run build
```

Before executing the `build` command, remember to edit the `config.js` file inside the directory `./melanoq/code/webclient/src/common` with the correct ip address of the MelanoQ api server.

Now you have two options:

- double-click `index.html` inside the directory `./melanoq/code/webclient/dist` to directly access the local instance of the client
- deploy the client (copy the content of the directory `./melanoq/code/webclient/dist`) into an http server (Apache Tomcat or others) to expose the client to the public.