UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

SNP Web-portal

Autori
U. Buonadonna, D. Sicignani, D. Tosoni

Anno Accademico
2013/2014

Capitolo 1

Introduction

In this chapter it is provided a brief introduction about the project described in the context of this essay.

1.1 Medical Background

The project focuses on the concept of **SNP**. SNP, *Single Nucleotide Polymorphism*, is defined as a *DNA sequence variation* occurring when a Single Nucleotide — A, T, C or G — in the genome (or other shared sequence) differs between members of a biological species or paired chromosomes. These variations can either be pathogenic, causing diseases, or completely harmless.
The main purpose for which the project is carried out is to *help biologists understanding, having an ad-hoc organized database, when a particular SNP can cause diseases or not.* The Web Portal allows them to **store** and **retrieve** Single Nucleotide Polymorphism genomics variants from a common database. This portal will then be used by biologists for this task, so it is realized so as to provide an intuitive and functional interface, which will act as an intermediary between the biologist and the actual database, facilitating their work as much as possible.

The main functions of the portal are divided into two groups, depending on the privilege level of the user who use the system at the time:

Super User

The *Super User* is the user with the *highest action privilege*, in fact it is in full possession of all the functionalities of the system.
As such, it can:

- create a family with one or more members
- authorize users
- modify inserted values

Basically, it can add, modify and enter the values it wants to. Obviously, the Super User can perform all the activities that Authorized User can do.

Authorized User

The *Authorized User* is the one with a *lower degree of privilege*, so it can not perform all the typical actions of the Super User, such as edit and insert.
It only can:

- search for a patient
- search for a gene
- search for a mutation
- ...

The authorized user is someone who can query the portal, but who can not modify any values.

The distinction between types of users has been designed to *provide a control mechanism against unwanted anomalies*; in fact, if a biologist is interested in making simple queries to the system, he is able do it in a way that no changes or accidental deletions occurs.

1.2 Project Structure

We will now describe the structure of the project.

The project is divided into three parts, which represent the basic structure of the portal:

1. **Database**
2. **Database - Website interaction**
3. **Website (User Interface)**

1.2.1 Database

The database is the most important part of the whole project, in fact the portal acts as an *intermediary* between the database itself and the biologist. **All informations about the SNPs are stored in the database**. It follows that a poorly designed database can invalidate all the work related to the portal; therefore the good output of the same is conditioned first of all by a good design of the database.

We will not focus on a detailed description of the structure of the database (that will be explained in *Chapter 1*).
We can see, however, that the most important part of the database is represented by the concept of **variation**, which represents nothing more that *a variation within the genome*. All other elements that need to be managed refer to the variant; we mention by way of example the gene relative to the variant, or the patient on which refers the same variant.

1.2.2 Database - Website interaction

Interaction between Database and Website is based on a *series of queries*, that the user can send to the system to retrieve the information it wants. These queries are then executed on the database and the result returned to the user.

Items covered by those queries may be different; for example, the system allows searching for:

- patient's SNPs
- gene's SNPs
- region's SNPs
- all SNPs with certain Mutation, Genotype, Freq alt,...
- patients with same SNP or Genotype
- SNPs within a genomic region
- specific SNP
- ...

The interactions can be carried out, as explained earlier, by all users (either Autenticated or Super User). The interactions for editing and adding data may relate to any entity in the database.

1.2.3 Website (User Interface)

The portal interface is represented by web pages, containing forms allowing to submit queries.
The fundamental characteristics that this interface must have are directly related to the it's purpose; we have to make sure that the biologists are facilitated as much as possible in their work of storing and retrieving information, to ensure that their work is as fast as possible. This affects also the intuitiveness and ease of use of the interface, that have to be taken into primary consideration.

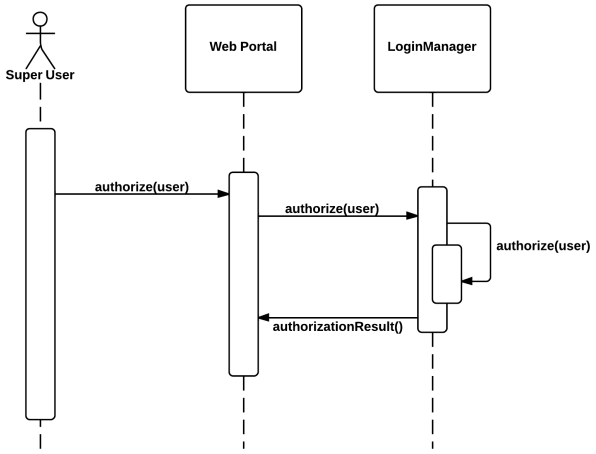
1.3 Use cases

In this section we will discuss the use cases, illustrating the use of the system.
They will show how the user interacts with the system, from the point of view of the

messages exchanged with it, and how the system handles these messages. You can see that the patterns of the use cases are similar to each other, since all gets the message from the user, and run it to the database by sending a special message for each request submitted.

We will now analyse briefly each use case.

1.3.1 Super User authorizes an User



In this use case the actors are the **Super User**, the **Web Portal** and the **Login manager**.The user's intention is to allow another user to have privilege for using the portal.

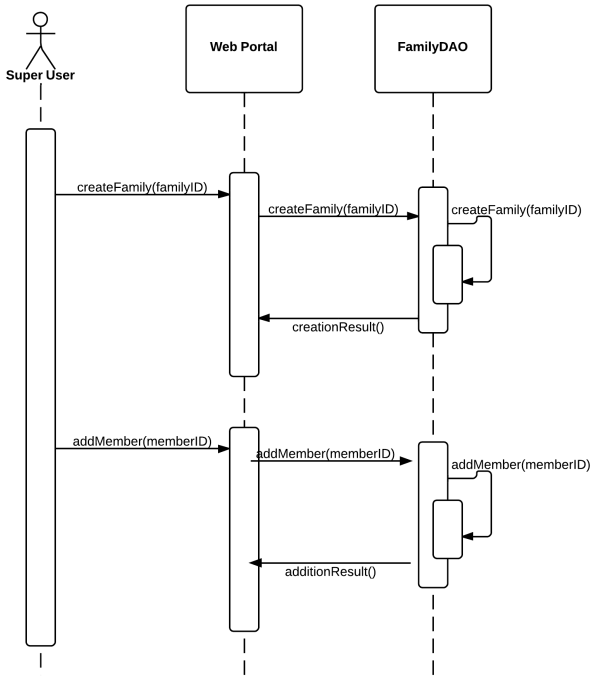
The interaction begins with the super user that sends a message *authorize(user)*to the Web portal. The latter sends the same message to the login manager, wich will be responsible to communicate with the database to record the granted permission. Successicely, the login manager sends a message to the web portal *authorization(result)*, containing the result of the operation, so that the Web Portal can know whether the authorization was granted or not.

1.3.2 Super User creates/populates a family

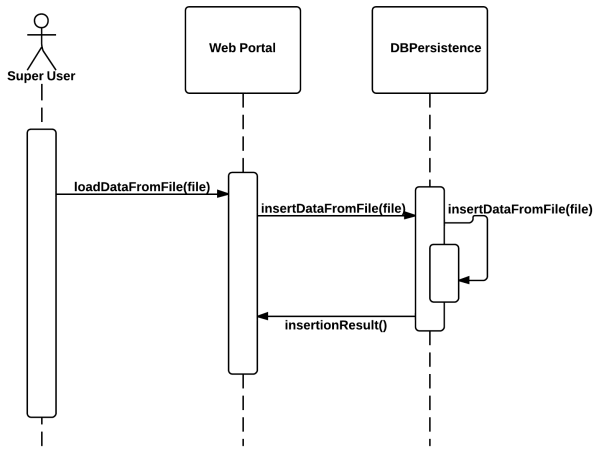
In this use case the actors are the **Super User**, the **Web Portal** and the **Family DAO**. The user's intention is to create a new family, and populate it with a member.

The interaction starts with the user sending a message *createFamily (familyID)* to Web Portal, which sends a message similar to FamilyDAO, the class responsible for managing / creating families and its interaction with the database. This class then communicates with the database itself, and sends a message containing the create result to the Web Portal.

If the family creation is successful, then the user can populate it, through a message *addMember(memberID)* sent to the Web Portal. The portal then dispatches the message to the FamilyDAO, who will add the member to the family, and returns the result to the Web Portal.



1.3.3 Super User loads data



In this use case the actors are the **Super User**, the **Web Portal** and the **DBpersistence**. The user's intention is to load the data that have to be stored in the database, for later querying. Note that data is submitted to the system by using a comma-separated file.

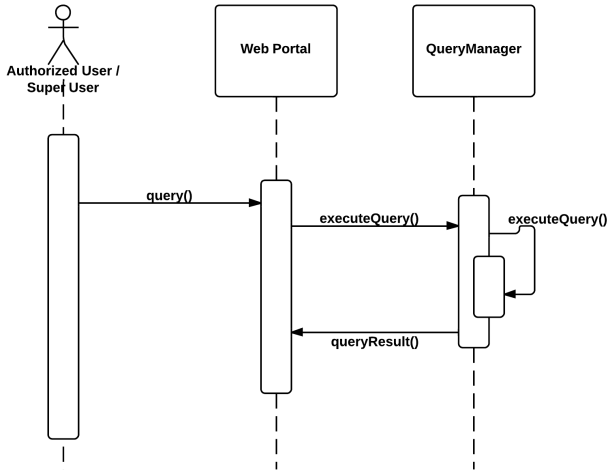
To accomplish this task, the user sends a message to the web portal *lodaDataFromFile(file)*, which file, as mentioned before, is a comma-separated file containing all the necessary data. Then the web portal sends message and data to the DBperistence class that both parse the file, and pass all information contained in file to the database, which will then store them.

Successively, this class will send the result of this insertion to the web portal through a message *insertionResult()*.

1.3.4 Authorized User executes a query

In this use case the actors are the **Authorized User**, the **Web Portal** and the **QueryManager**. The user's intention is to run a query on the system, in order to obtain the information he wants to.

The user then sends a message *query()* to the web portal, which will contain a generic query submitted by the user.The portal will send a message *executeQuery()* to QueryManager, who will actually run the query in the database, and return the results to the portal. It does this through a message *QueryResult()*



1.4 Software architecture and tecnologies

In this section we will discuss the software technologies used in the project, both for the front-end and back-end.

1.4.1 Programming language

The only programming language used throughout the project is **JavaScript**. This is because we want to use frameworks supporting the JavaScript language (that can be considered one of the most utilised languages relating to dynamic content in web pages) that ensure a great rapidity of coding.

1.4.2 Technologies

Technologies used all ri around chosen language. In particular:

Database: MongoDB

Website: AngularJS

Model: NodeJS

Framework: ExpressJS

MongoDB is a *NoSQL object-oriented DBMS*, which is based on the use of documents to represent objects. AngularJS allows to *extend HTML with instructions for building dynamic web pages*. NodeJS is an *event-driven, non-blocking network platform for building applications*. ExpressJS is a *web application framework for NodeJS*, that extends some of its features, to make development faster.

These four components form the so-called **MEAN** stack; it has been chosen for its scalability and for the quickness of development it can offer.

Capitolo 2

Database Description

In this chapter we will explain the creation of the database, from code point of view as well as from design point of view.
The **database** is the core part of the back end of the application, because it is the structure that keeps stored all the data necessary for the operation of the portal. Because of its importance, we spent much of our time to decide its structure (tables, relationships, multiplicity, ...) and only after a thorough analysis of results we moved to implementation.

As explained in the introductory chapter, the software implementation of database was performed using **MongoDB** through **Mongoose** (a MongoDB object modeling tool). MongoDB is an open-source document database, and the leading NoSQL database. It is based on *Document-Oriented Storage* and unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections do not enforce document structure. This flexibility facilitates the mapping of documents to an entity or an object.
Mongoose helps to have a more efficient management of the objects; in fact, using mongoose's schemas to define database's objects, permits to treat database's models as commons JavaScript's objects.

Throughout the chapter, we will see the objects that compose the model of our da-

tabase, but, first of all, let us focus on the syntax used to communicate with MongoDB and Mongoose.

2.1 Mongoose's syntax

To realize an object using mongoose, we need to follow a specific syntax.
First of all, we should place all files related to the models (one file for each model) in a specific folder of the project; this folder is as follows:

/server/models

At the beginning of each file, we have to inform the database that all data passed to builders, but that are not present in the schema that we will define below, must not be stored.
To do this we use the following line of code:

```
1 'use strict';
```

Next, we must specify needed *modules dependencies*. In our project, we have to insert the dependency from mongoose and crypto (a module for encryption of information):

```
1 var mongoose = require('mongoose'),
2 Schema = mongoose.Schema,
3 crypto = require('crypto');
```

Now we move to define the *schema*, specifying all necessary components (attributes and relationships). Note that MongoDB will automatically add an ID field to each schema and assigns it using an internal policy.

For example, if we want to model a *User* with an attribute *name* of type *string*:

⟨ *Example schema ?* ⟩ ≡

```
UserSchema = new Schema ({
        //attribute
        name: {
                type: String,
                required: true,
                validate: [validatePresenceOf, 'Name can not be blank
    ']
        },

        //relationship
        patiens: [{
                type: Schema.Type.ObjectId,
                ref: 'Patient'
        }]
        );
```

◇

Fragment never referenced.

Next, we can define a number of *additional features* to our model, such as:

- **Validations**: functions called to validate a set of attributes or data specified by the programmer
- **Virtuals**: virtual attributes, not persisted
- **Pre-save hooks**: functions called at the time immediately before the model is saved to the database
- **Methods**: all methods needed for the operations on our model

Finally, we must specify the actual name of the model described by the schema specified as previously shown:

```
mongoose.model ('User', UserSchema);
```

2.2 Model Schemas

These are the model schemas that compose our database.

2.2.1 Variants

The model *Variant* represents the key concept that we want to represent in our work: **genomic variants**.
This table will always be accessed during any kind of consultation, in fact are variants what biologists are looking for or insert through the portal.
The centrality of the concept and, therefore, of the table is also demonstrated by the fact that in the database schema (see chapter *Introduction*) Variant table is linked to almost all the other tables.

Let us look at the code components:

Attributes

⟨ *Variant's attributes ?* ⟩ ≡

```
/* Variant schema */
var VariantSchema = new Schema({
    chr: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'CHR cannot be blank']
    },
    start: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'start cannot be blank']
    },
    end: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'End cannot be blank']
    },
    ref: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Ref cannot be blank']
    },
    alt: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Alt cannot be blank']
    },
```

◇

Fragment never referenced.

- **chr**: it is a five-character string describing the chromosome affected by the variation

- **start, end**: integers that describe the coordinate of the mutation within the chromosome; if they match, it means that we have a punctiform mutation
- **ref**: represents the correct bases, as they would be if there were no mutation
- **alt**: represents what we actually find due to the mutation; a "." or a "-" represent a deletion, while a sequence, for example CTTG..., represents an insertion

Relationships

⟨ *Variant's relationships ?* ⟩ ≡

```
    // Here start relations
    gene: {
        type: Schema.Types.ObjectId,
        ref: 'Gene'
    },

    pathogenicity: {
        type: Schema.Types.ObjectId,
        ref: 'Pathogenicity'
    },

    patients: [{
        type: Schema.Types.ObjectId,
        ref: 'Patient'
    }],

    dbSNPs: [{
        type: Schema.Types.ObjectId,
        ref: 'DbSNP'
    }],

    esps: [{
        type: Schema.Types.ObjectId,
        ref: 'Esp'
    }],

    variantDetails: [{
        type: Schema.Types.ObjectId,
        ref: 'VariantDetail'
    }]

},{ versionKey: false });
```

◇

Fragment never referenced.

- **gene**: links the variant with the gene affected by the mutation
- **pathogenicity**: describes the variant pathogenicity, that is if the variant cause diseases or not
- **patients**: links to patients affected by this varant
- **dbSNPs**: connects the variant to its dbSNPs
- **esps**: Exome Sequencing Project
- **sequencings**: all sequencings affected by this variation

2.2.2 dbSNP

dbSNP is the key concept of the database, that is the **Single Nucleotide Polymorphism Database** (a free public archive for genetic variation within and across different species developed and hosted by the *National Center for Biotechnology Information (NCBI)* in collaboration with the *National Human Genome Research Institute (NHGRI)*).

Let us look at the code components:

Attributes

⟨ *dbSNP's attributes ?* ⟩ ≡

```
    function validatePresenceOf(x) {return true;}
/* DbSNP schema */
var DbSNPSchema = new Schema({
    dbSNP: {
        //key
        type: String,
        required: true,
        unique: true,
        validate: [validatePresenceOf, 'DbSNP cannot be blank']
    },
    freqAlt: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'freqAlt cannot be blank']
    },
    freqRef: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'freqRef cannot be blank']
```

◇

Fragment never referenced.

- **dbSNP**: identifier of the variation in the dbSNP database
- **freqAlt**: frequency of the alternative allele (variant) in the population 1000 Genomes
- **freqRef**: frequency of the reference allele

Relationships

⟨ *dbSNP's relationships ?* ⟩ ≡

```
1
2     //Relationship
3     variants: [{
4         type: Schema.ObjectId,
5         ref: 'Variant'
6     }]
```

◇

Fragment never referenced.

- **variants**: links dbSNP to its variants

2.2.3 Gene

A *Gene* is the molecular unit of heredity of a living organism. Genes hold the information to build and maintain an organism's cells and pass genetic traits to offspring. All organisms have genes corresponding to various biological traits, some of which are instantly visible, such as eye color or number of limbs, and some of which are not, such as blood type, increased risk for specific diseases, or the thousands of basic biochemical processes that comprise life.

Let us look at the code components:

Attributes

⟨ *Gene's attributes ?* ⟩ ≡

```
/* Gene schema */
var GeneSchema = new Schema({
    genes: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Gene cannot be blank']
    },
    region: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Region cannot be blank']
    },
    mutation: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Mutation cannot be blank']
    },
    annotation: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Annotation cannot be blank']
    },
```

◇

Fragment never referenced.

- **gene**: identifier of the gene
- **region**: the region in which the gene is located
- **mutation**: the mutation that affects the gene
- **annotation**: any annotations

Relationships

⟨ *Gene's relationships ?* ⟩ ≡

```
    // Relationship
    variants: [{
        type: Schema.ObjectId,
        ref: 'Variant'
    }]

},{ versionKey: false });
```

◇

Fragment never referenced.

- **variants**: links gene to its variants

2.2.4 Pathogenicity

Pathogenicity indicates whether a mutation is pathogenic (able to create damages to the organism) or not.

Let us look at the code components:

Attributes

⟨ *Pathogenicity's attributes ?* ⟩ ≡

```
/* Pathogenicity schema */
var PathogenicitySchema = new Schema({
    SIFT: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'SIFT cannot be blank']
    },

    polyPhen: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'polyPhen cannot be blank']
    },

    mutationTaster: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'mutationTaster cannot be
    blank']
    },

    mutationAssessor: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'mutationAssessor cannot be
    blank']
    },

    GERpp: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'GERPpp cannot be blank']
    },

    phyloP: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'pyoloP cannot be blank']
    },
```

◇

Fragment never referenced.

⟨ *Pathogenicity's attributes - 2 ?* ⟩ ≡

```
    siPhy: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'siPhy cannot be blank']
    },
```

◇

Fragment never referenced.

- **SIFT, polyPhen, mutationTaster, mutationAssessor**: all four are predictors of pathogenicity of the mutation. The numbers are the scores of the pathogenicity.
- **GERPpp, pyoloP, SiPhy**: all three are predictors of conservation of the mutation. The numbers are the scores of conservation.

Relationships

⟨ *Pathogenicity's relationships ?* ⟩ ≡

```
    //Relationship
    variant: {
        type: Schema.ObjectId,
        ref: 'Variant'
    }

},{ versionKey: false });
```

◇

Fragment never referenced.

- **variant**: links pathogenicity to its variant

2.2.5 Sequencing

DNA Sequencing is the process of determining the nucleotide order of a given DNA fragment. The sequence of DNA encodes the necessary information for living things to survive and reproduce, so determining the sequence is therefore useful in fundamental research into why and how organisms live, as well as in applied subjects.

Let us look at the code components:

Attributes

⟨ *Sequencing's attributes ?* ⟩ ≡

```
/* Sequencing schema */
var SequencingSchema = new Schema({
    //patient id and date are keys
    patientId: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'patient cannot be blank']
    },
    date: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'date cannot be blank']
    },
    patientHealthStatus: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'patientHealthStatus cannot be
     blank']
    },
    sequencerName: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'sequencerName cannot be blank
    ']
    },
    sequencerModel: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'sequencerModel cannot be
    blank']
    },
    referenceGenome: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'referenceGenome cannot be
    blank']
    },
```

◇

Fragment never referenced.

⟨ *Sequencing's attributes - 2 ?* ⟩ ≡

```
detail :{
    qual: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'qual cannot be blank']
    },
    filter: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'filter cannot be blank']
    },
    genotype: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'genotype cannot be blank']
    },
    genotypeQuality: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'genotypeQuality cannot be blank']
    },
    readsDepth: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'readsDepth cannot be blank']
    },
    ref: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'ref cannot be blank']
    },
    altFilterReads: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'readsDepth cannot be blank']
    },
    genotypeLikelihood: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'genotypeLikelihood cannot be blank']
```

- **patientId**: identifier of the patient.
- **date**: the date when sequencing was performed.
- **patientHealthStatus**: if the patient is diseased or not.
- **sequencerName**: name of the sequencer.
- **sequencerModel**: model of the sequencer.
- **referenceGenome**: the reference genome.
- **qual**: describes the quality of the sequencing.
- **filter**: reference allele.
- **genotype**
- **genotypeQuality**: quality of the genotype.
- **readsDepth**: the depth of the reads.
- **ref**: string that indicates whether the quality of the variant is good (PASS).
- **altFilterReads**: number of reads that do not contain the mutation to their inside.
- **genotypeLikelihood**: probability that the determined genotype is correct.
- **haplotypeScore**: indicate the presence of misaligned reads in the neighborhood of the variant.
- **strandBias**: all four are predictors of pathogenicity of the mutation.

Relationships

⟨ *Sequencing's relationships ?* ⟩ ≡

```
    //Relationship
    variants: [{
        type: Schema.ObjectId,
        ref: 'Variant'
    }]
},{ versionKey: false });
```

◇

Fragment never referenced.

- **variants**: links sequencing to its variants

2.2.6 ESP

ESP is the part of the database that represents locally the interesting portion of the database of the **Exome Sequencings Project**, the goal of which is to discover novel genes and mechanisms contributing to heart, lung and blood disorders by pioneering the application of next-generation sequencing of the protein coding regions of the human genome across diverse, richly-phenotyped populations and to share these datasets and findings with the scientific community to extend and enrich the diagnosis, management and treatment of heart, lung and blood disorders.

Let us look at the code components:

Attributes

⟨ *esp's attributes ?* ⟩ ≡

```
/* Esp schema */
var EspSchema = new Schema({
    ESP6500_ALL: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'ESP6500_ALL cannot be blank']
    },
    ESP6500_AA: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'ESP6500AA cannot be blank']
    },
    ESP6500_EA: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'ESP6500_EA cannot be blank']
    },
```

◇

Fragment never referenced.

- **ESP6500_ALL**: frequency of the mutation in the EVS database and in the overall population
- **ESP6500_AA**: frequency of the mutation in the EVS database and in the American/African population
- **ESP6500_EA**: frequency of the mutation in the EVS database and in the European /American population

Relationships

⟨ *esp's relationships ?* ⟩ ≡

```
    //Relationship
    variants: [{
        type: Schema.ObjectId,
        ref: 'Variant'
    }]

},{ versionKey: false });
```

◇

Fragment never referenced.

- **variants**: links esp to its variants

2.2.7 Patient

A *Patient* describes a patient in the real world. Can only be created by a user with Admin privileges.

Let us look at the code components:

Attributes The Patient object has no attributes to be specified; it only has an identifier (**ID**)that is added from MongoDB by default and a **Name**.

⟨ *Patient's attributes ?* ⟩ ≡

```
/* PAtient schema */
var PatientSchema = new Schema({

        name: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Name cannot be blank']
         },
```

◇

Fragment never referenced.

Relationships

⟨ *Patient's relationships ?* ⟩ ≡

```
1  //Relationship
2  variants: [{
3    type: Schema.ObjectId,
4    ref: 'Variant'
5  }],
6
7  family: {
8    type: Schema.ObjectId,
9    ref: 'Family'
10 }
11
12 });   //,{ versionKey: false });
```

◇

Fragment never referenced.

- **variants**: links patient to its variants
- **family**: links patient to its family

2.2.8 Family

A *Family* represents a set of one or more *patients* (modeled in patient schema). It can be created only by an user with admin privileges. This schema allow to group patients with similar genomic mutations as well as real families (father, mother, son...).

Let us look at the code components:

Attributes

⟨ *Family's attributes ?* ⟩ ≡

```
/* Family schema */
var FamilySchema = new Schema({

    name: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Name cannot be blank']
    },
```

◇

Fragment never referenced.

- **name**: describes family's name

Relationships

⟨ *Family's relationships ?* ⟩ ≡

```
    //Relationship
    patients: [{
      type: Schema.ObjectId,
      ref: 'Patient'
    }]

},{ versionKey: false });
```

◇

Fragment never referenced.

- **patients**: links family to its members

2.2.9 User

The *User* object describes a user of the system, a *biologist* who accesses the portal to carry out one of the supported operations (described in the introductory chapter).

Let us look at the code components:

Attributes

⟨ *User's attributes ?* ⟩ ≡

```
/**
 * User Schema
 */
var UserSchema = new Schema({
    name: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Name cannot be blank']
    },
    firstName: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'First Name cannot be blank']
    },
    lastName: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Last Name cannot be blank']
    },
    email: {
        type: String,
        required: true,
        match: [/.+\@.+\..+/, 'Please enter a valid email'],
        validate: [validatePresenceOf, 'Email cannot be blank']
    },
    address: {
        type: String,
    },
    phone: {
        type: String,
    },
    username: {
        type: String,
        unique: true,
        validate: [validatePresenceOf, 'Username cannot be blank']
    },
    roles: {
        type: Array,
        default: ['authenticated']
    },
```

◇

Fragment never referenced.

⟨ *User's attributes - 2 ?* ⟩ ≡

```
    hashed_password: {
        type: String,
        validate: [validatePresenceOf, 'Password cannot be blank']
    },
    provider: {
        type: String,
        default: 'local'
    },
    salt: String,
    facebook: {},
    twitter: {},
    github: {},
    google: {},
    linkedin: {}
});

/**
 * Virtuals
 */
```

◇

Fragment never referenced.

These attributes are all self-explanatory.

Methods

⟨ *User's validation ?* ⟩ ≡

```
    this._password = password;
    this.salt = this.makeSalt();
    this.hashed_password = this.hashPassword(password);
}).get(function() {
    return this._password;
});

/**
 * Pre-save hook
 */
UserSchema.pre('save', function(next) {
    if (this.isNew && this.provider === 'local' && this.password && !
    this.password.length)
        return next(new Error('Invalid password'));
    next();
});

/**
 * Methods
 */
UserSchema.methods = {

    /**
     * HasRole - check if the user has required role
     *
     * @param {String} plainText
     * @return {Boolean}
     * @api public
     */
    hasRole: function(role) {
        var roles = this.roles;
        return roles.indexOf('admin') !== -1 || roles.indexOf(role)
    !== -1;
    },

    /**
     * IsAdmin - check if the user is an administrator
     *
     * @return {Boolean}
```

◇

Fragment never referenced.

⟨ *User's methods ?* ⟩ ≡

```
     * @api public
     */
    isAdmin: function() {
        return this.roles.indexOf('admin') !== -1;
    },

    isLicensed: function(){
        return this.roles.indexOf('licensed') !== -1;
    },

    /**
     * Authenticate - check if the passwords are the same
     *
     * @param {String} plainText
     * @return {Boolean}
     * @api public
     */
    authenticate: function(plainText) {
        return this.hashPassword(plainText) === this.hashed_password;
    },

    /**
     * Make salt
     *
     * @return {String}
     * @api public
     */
    makeSalt: function() {
        return crypto.randomBytes(16).toString('base64');
    },

    /**
     * Hash password
     *
     * @param {String} password
```

◇

Fragment never referenced.

⟨ *User's methods - 2 ?* ⟩ ≡

```
     * @return {String}
     * @api public
     */
    hashPassword: function(password) {
        if (!password || !this.salt) return '';
        var salt = new Buffer(this.salt, 'base64');
        return crypto.pbkdf2Sync(password, salt, 10000, 64).toString('base64');
    }
};

mongoose.model('User', UserSchema);
```

◇

Fragment never referenced.

Functions performed by these methods are explained in the code comments.

Capitolo 3

Backend-Frontend communication

In this chapter we will explain the realization of the interaction between application's Backend and Frontend.
The interaction occurs through the implementation of **RESTful APIs that allow to interact with a MongoDB instance**.

3.1 The idea

We built some APIs (one for each item that the application handles) that:

- Handle CRUD for the item
- Use the proper HTTP verbs to make it RESTful (GET, POST, PUT, and DELETE)
- Return JSON data
- Log all requests to the console

All of this is pretty standard for RESTful APIs.

3.2 Implementation

To realize APIs, we needed to:

1. define our Node packages
2. define our models
3. declare our routes using Express
4. test our API

3.2.1 Node packages

Node Packages are the list of the support packages that are used for the realization of the application; therefore it also contains the packages needed for the APIs.

This is our package.json file:

⟨ *package.json ?* ⟩ ≡

```
{
  "name": "mean",
  "description": "MEAN.io: A fullstack javascript platformed powerd
    by MongoDB, ExpressJS, AngularJS, NodeJS.",
  "version": "0.3.3",
  "private": false,
  "author": "Linnovate <mean@linnovate.net>",
  "contributors": [
    { "name": "Lior Kesos", "mail": "lior@linnovate.net" },
    { "name": "Yonatan Ellman", "mail": "yonatan@linnovate.net" },
    { "name": "Ehud Shahak", "mail": "ehud@linnovate.net" },
    { "name": "Amos Haviv", "mail": "mail@amoshaviv" },
    { "name": "Drew Fyock", "mail": "drew@steelbisondev.com" }
  ],
  "mean": "0.3.3",
  "repository": {
    "type": "git",
    "url": "https://github.com/linnovate/mean.git"
  },
  "engines": {
    "node": "0.10.x",
    "npm": "1.3.x"
  },
  "scripts": {
    "start": "node node_modules/grunt-cli/bin/grunt",
    "test": "node node_modules/grunt-cli/bin/grunt test",
    "postinstall": "node node_modules/bower/bin/bower install"
  },
  "dependencies": {
    "angular-md5": "*",
    "assetmanager": "0.1.2",
    "body-parser": "^1.2.0",
    "bower": "^1.3.3",
    "compression": "^1.0.1",
    "connect-flash": "^0.1.1",
    "consolidate": "^0.10.0",
    "cookie-parser": "^1.1.0",
    "dependable": "0.2.5",
    "errorhandler": "^1.0.0",
    "express": "^4.2.0",
    "express-session": "^1.1.0",
    "express-validator": "^2.1.1",
    "forever": "^0.11.1",
    "grunt-cli": "^0.1.13",
    "grunt-concurrent": "^0.5.0",
```

⟨ *package.json - 2 ?* ⟩ ≡

```
    "grunt-concurrent": "^0.5.0",
    "grunt-contrib-clean": "^0.5.0",
    "grunt-contrib-csslint": "^0.2.0",
    "grunt-contrib-cssmin": "^0.9.0",
    "grunt-contrib-jshint": "^0.10.0",
    "grunt-contrib-uglify": "^0.4.0",
    "grunt-contrib-watch": "^0.6.1",
    "grunt-env": "^0.4.1",
    "grunt-nodemon": "0.2.1",
    "load-grunt-tasks": "^0.4.0",
    "lodash": "^2.4.1",
    "mean-connect-mongo": "0.4.3",
    "gridfs-stream": "^0.5.1",
    "mean-logger": "0.0.1",
    "meanio": "0.4.x",
    "method-override": "^1.0.0",
    "mers": "*",
    "mongoose": "^3.8.8",
    "morgan": "^1.0.0",
    "passport": "^0.2.0",
    "passport-facebook": "^1.0.3",
    "passport-github": "^0.1.5",
    "passport-google-oauth": "^0.1.5",
    "passport-linkedin": "^0.1.3",
    "passport-local": "^1.0.0",
    "passport-twitter": "^1.0.2",
    "serve-favicon": "^2.0.0",
    "superagent": "*",
    "supertest":"*",
    "swig": "^1.3.2",
    "time-grunt": "^0.3.1",
    "view-helpers": "^0.1.4"
  },
  "devDependencies": {
    "assetmanager": "^0.1.2",
    "grunt-karma": "^0.8.2",
    "grunt-mocha-test": "^0.10.2",
    "karma": "^0.12.10",
    "karma-chrome-launcher": "^0.1.3",
    "karma-coffee-preprocessor": "^0.2.1",
    "karma-coverage": "^0.2.1",
    "karma-firefox-launcher": "^0.1.3",
    "karma-html2js-preprocessor": "^0.1.0",
    "karma-jasmine": "^0.1.5",
    "karma-ng-html2js-preprocessor": "^0.1.0",
```

⟨ *package.json - 3 ?* ⟩ ≡

```
    "karma-ng-html2js-preprocessor": "^0.1.0",
    "karma-ng-scenario": "^0.1.0",
    "karma-phantomjs-launcher": "^0.1.4",
    "karma-requirejs": "^0.2.1",
    "karma-script-launcher": "^0.1.0",
    "requirejs": "^2.1.11",
    "should": "3.3.1",
    "supertest": "0.11.0"
  }
}
```

◇

Fragment never referenced.

In our selection of packages, those necessary for the implementation of the APIs are:

- **express**, the Node framework
- **mongoose**, the ORM we used to communicate with our MongoDB database
- **body-parser**, to pull POST content from our HTTP request

3.2.2 Model

Models required for the API are the same as those described in *Chapter 2*, to which was added an additional model: **User**. It is necessary to be able to properly handle user information as well as the levels of privilege that a user owns.

This is the User model:

⟨ *User's attributes ?* ⟩ ≡

```
/**
 * User Schema
 */
var UserSchema = new Schema({
    name: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Name cannot be blank']
    },
    firstName: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'First Name cannot be blank']
    },
    lastName: {
        type: String,
        required: true,
        validate: [validatePresenceOf, 'Last Name cannot be blank']
    },
    email: {
        type: String,
        required: true,
        match: [/.+\@.+\..+/, 'Please enter a valid email'],
        validate: [validatePresenceOf, 'Email cannot be blank']
    },
    address: {
        type: String,
    },
    phone: {
        type: String,
    },
    username: {
        type: String,
        unique: true,
        validate: [validatePresenceOf, 'Username cannot be blank']
    },
    roles: {
        type: Array,
        default: ['authenticated']
    },
```

◇

Fragment never referenced.

⟨ *User's attributes - 2 ?* ⟩ ≡

```
    hashed_password: {
        type: String,
        validate: [validatePresenceOf, 'Password cannot be blank']
    },
    provider: {
        type: String,
        default: 'local'
    },
    salt: String,
    facebook: {},
    twitter: {},
    github: {},
    google: {},
    linkedin: {}
});

/**
 * Virtuals
 */
```

◇

Fragment never referenced.

⟨ *User's validation ?* ⟩ ≡

```
    this._password = password;
    this.salt = this.makeSalt();
    this.hashed_password = this.hashPassword(password);
}).get(function() {
    return this._password;
});

/**
 * Pre-save hook
 */
UserSchema.pre('save', function(next) {
    if (this.isNew && this.provider === 'local' && this.password && !
    this.password.length)
        return next(new Error('Invalid password'));
    next();
});

/**
 * Methods
 */
UserSchema.methods = {

    /**
     * HasRole - check if the user has required role
     *
     * @param {String} plainText
     * @return {Boolean}
     * @api public
     */
    hasRole: function(role) {
        var roles = this.roles;
        return roles.indexOf('admin') !== -1 || roles.indexOf(role)
    !== -1;
    },

    /**
     * IsAdmin - check if the user is an administrator
     *
     * @return {Boolean}
```

◇

Fragment never referenced.

⟨ *User's methods ?* ⟩ ≡

```
     * @api public
     */
    isAdmin: function() {
        return this.roles.indexOf('admin') !== -1;
    },

    isLicensed: function(){
        return this.roles.indexOf('licensed') !== -1;
    },

    /**
     * Authenticate - check if the passwords are the same
     *
     * @param {String} plainText
     * @return {Boolean}
     * @api public
     */
    authenticate: function(plainText) {
        return this.hashPassword(plainText) === this.hashed_password;
    },

    /**
     * Make salt
     *
     * @return {String}
     * @api public
     */
    makeSalt: function() {
        return crypto.randomBytes(16).toString('base64');
    },

    /**
     * Hash password
     *
     * @param {String} password
```

◇

Fragment never referenced.

⟨ *User's methods - 2 ?* ⟩ ≡

```
     * @return {String}
     * @api public
     */
    hashPassword: function(password) {
        if (!password || !this.salt) return '';
        var salt = new Buffer(this.salt, 'base64');
        return crypto.pbkdf2Sync(password, salt, 10000, 64).toString('base64');
    }
};

mongoose.model('User', UserSchema);
```

◇

Fragment never referenced.

3.2.3 Routes

We used an instance of the *Express Router* to handle all of our routes.
Here is an overview of the routes we made, what they do, and the HTTP Verb used to access it. *"model"* represent one of the models of our application.

Route	HTTP Verb	Description
/api/model	GET	Get all the "model"
/api/model	POST	Create a "model"
/api/model/:model_id	GET	Get a single "model"
/api/model/:model_id	POST	Update a "model" with new info
/api/model/:model_id	DELETE	Delete a "model"

To achieve this, we used the package **MERS** (*Mongoose Express Rest Service*): it is a plugin for express to expose mongoose finders as simple crud/rest operations.

3.2.4 Tests

Capitolo 4

Frontend

In this chapter we will explain the realization of the Frontend.

4.1 The idea

The application is divided into *two parts*: the *first part* is also viewable without logging in; the *second part* is visible only after the authentication.
In the first part, you can view general information about the application, along with information about those who built it and a tutorial to help you use the application.
In the second part, instead, there are all the necessary view to perform operations that users can perform using the application.

To implement the frontend we set goal: to *have the most possible compatibility with any screen, at any resolution or aspect ratio.* To achieve this it was decided to use the Twitter **Bootstrap**, which greatly facilitates the work.

4.2 First part

The first part of the application, as mentioned, is appointed to *provide the basic informations* in order to facilitate the use of the services provided. It consists of the following pages:

- **Home**
- **Login**
- **About us**
- **Info**

4.2.1 Home

This is the landing page, the first page that appears when you open the application.
It contains a link to the Login page and a link that allows you to contact an administrator to request an account if you do not have one.
From the graphical point of view, what is particularly noticeable is the background, consisting of a video that is an animation of a DNA strand.

This is the code used to implement it:

```
1 <link href="public/system/assets/css/login.css" rel="stylesheet">
2 <link rel="stylesheet" type="text/css" href="/public/system/assets/css/
    video_background.css">
3
4 <video id="video_background" preload="auto" autoplay="true" loop="loop"
    muted="muted" volume="0">
5 <source src="/public/system/assets/videos/background.mp4" type="video/mp4"
    >
6 Video not supported </video>
7
8 <div class="page-header" data-ng-controller="HeaderController">
9
10    <section class="main content-section">
```

```
11          <div class="row message-top col-xs-9" data-ng-hide="global.
    authenticated">
12              Welcome to the SNP Web-Portal!
13          </div>
14
15          <div class="row message-bottom col-xs-6" data-ng-hide="global.
    authenticated">
16              Please
17              <a ui-sref='auth.login' role="button" class="btn btn-primary">
    Login</a>
18              to use this portal...
19          </div>
20
21          <div class="row message-email col-xs-9" data-ng-hide="global.
    authenticated">
22              If you do not have an account yet, please
23              <a href="mailto:admin@admin.it" role="button">contact an
    administrator</a> to have it!
24          </div>
25
26      </section>
27  </div>
```

4.2.2 Login

This page is *the most important of the first part of the application*, in fact it is the one that *allows* a user with logon credentials *to access the second part of the application.* Its structure is very simple; it only consists of a form that contains the necessary fields to authenticate to the system.

This is the code used to implement it:

```
1 <link href="public/system/assets/css/login.css" rel="stylesheet">
2
3 <section class="main content-section">
4     <div data-ng-controller="LoginCtrl" class="content-section-login col-
    md-5 col-md-offset-1">
```

```
5         <div class="alert alert-danger animated fadeIn" ng-show="
    loginerror">{{loginerror}}</div>
6
7         <form  ng-submit="login()" class="form-horizontal">
8             <div class="text-left"><h1>Login</h1></div>
9             <div class="form-group">
10                <div class="col-md-10">
11                    <input required id="email" type="email" name="email"
    placeholder="Email" class="form-control" ng-model="user.email"/>
12                </div>
13            </div>
14            <div class="form-group">
15                <div class="col-md-10">
16                    <input required id="password" type="password" name="
    password" placeholder="Password" class="form-control" ng-model="user.
    password"/>
17                </div>
18            </div>
19            <div class="form-group" style="margin-bottom: 275px">
20                <div class="col-md-8">
21                    <button type="submit" class="btn btn-primary">Sign in
    </button> 
22                </div>
23            </div>
24        </form>
25    </div>
26</section>
```

4.2.3 About Us

This page is a sort of "self-congratulatory page"; in fact *contains the description of one who has made the application and of the person who commissioned it, the CSS Mendel.* On the page there is a button that allows you to contact technical support in case of problems with the application and another button, aimed primarily at technicians, which allows you to view the source code of the application using a GitHub repository.

This is the code used to implement it:

```
<link href="public/system/assets/css/aboutUs.css" rel="stylesheet">


<section class="main content-section" style="margin-top: 50px">

    <div class="row text-center container-about">
      <h1><b>Realized by...</b></h1>

      <div class="col-md-4 text-center" style="margin-top: 50px">
        <img class="img-circle img-responsive" src="public/system/assets/
    img/Davide_Sicignani.jpg">
        <h2><b>Davide Sicignani</b></h2>
      </div>
      <div class="col-md-4 text-center" style="margin-top: 50px">
        <img class="img-circle img-responsive" src="public/system/assets/
    img/Ugo_Buonadonna.png">
        <h2><b>Ugo Buonadonna</b></h2>
      </div>
      <div class="col-md-4 text-center" style="margin-top: 50px">
        <img class="img-circle img-responsive" src="public/system/assets/
    img/Damian_Tosoni.jpg">
        <h2><b>Damian Tosoni</b></h2>
      </div>
    </div>

    <div class="row text-center" style="margin-top: 100px">
      <div class="col-lg-10 col-lg-offset-1">
        <h1><b>CSS Mendel</b></h1>
        <h3>Studiare, diagnosticare e prevenire le malattie genetiche ed
    eredo-familiari</h3>
        <div class="row">
          <div class="col-md-4">
            <img src="public/system/assets/img/RicercaScientifica.jpg"
    class="img-responsive">
            <h3>Scientific research</h3>
          </div>
          <div class="col-md-4">
```

```
33            <img src="public/system/assets/img/Laboratorio.jpg" class="img
    -responsive">
34            <h3>Specialized laboratory of Medical Genetics</h3>
35          </div>
36          <div class="col-md-4">
37            <img src="public/system/assets/img/Ambulatori.jpg" class="img-
    responsive">
38            <h3>Ambulatories</h3>
39          </div>
40        </div>
41        <p>Visit our official site by clicking <a href="http://www.css-
    mendel.it/" target="_blank">here</a>.</p>
42      </div>
43    </div>
44
45    <div class="row">
46      <!-- Small modal -->
47      <div class="container">
48          <div class="row controls-groups">
49              <h1 class="text-center"><a href="mailto:dav.sicignani@stud.
    uniroma3.it" role="button" class="btn btn-primary btn-lg" data-toggle=
    "modal">Contact Us</a></h1>
50              <h1 class="text-center"><a href="https://github.com/
    thekage91/web-snp" class="btn btn-default btn-lg"><i class="fa fa-
    github fa-fw"></i> <span class="network-name">Github</span></a>
51              </h1>
52          </div>
53      </div>
54    </div>
55</section>
```

4.2.4 Info

The Info page provides *general information about the application*, how it is structured and what you can do based on their level of authorization (admin user or licensed user).

4.3 Second part

The second part of the application is the core. is structured by way of a **dashboard**, divided into several **tabs** that are visible only if you reach the required permission level.

The tab of which it is composed are the following:

- **Home**
- **My Profile**
- **Execute Query**
- **Sequencing Upload**
- **Sequencing Upload History**
- **Authorize Users**
- **Families**

In particular, an Admin User can view all of the tabs, the Licensed User only the first three.

4.3.1 Home

This page is the first page that appears immediately after login.
It is a *welcome page for the user* and thus contains a greeting, the profile picture, and a suggestion on how to use the application. Regarding the image of the profile, the application refers to **Gravatar**; it is therefore necessary to have an account on the site or register a new one using the same email used for login to be able to have an image of the profile other than the default one.

This is the code used to implement it:

```
1 <div class="col-lg-11">
2
3 <div class="row">
```

```
4   <h1 class="section-header">Home</h1>
5
6   <hr>
7 </div>
8
9 <div class="row user-menu-container square" ng-controller="ProfileCtrl">
10    <div class="col-md-12 user-details">
11        <div class="row coralbg white" style="margin-top: 30px">
12            <div class="col-md-6 no-pad">
13                <div class="user-pad" style="margin-left: 30px">
14                    <h3>Welcome back, </h3><h1>{{global.user.username}}</
    h1>
15
16                    <h4 style="margin-top: 75px"><span class="fa fa-
    lightbulb-o"></span> <b>TIP</b>: Click on a button in the sidebar
     to start.</h4>
17
18                </div>
19            </div>
20            <div class="col-lg-6 no-pad">
21                <div class="user-image">
22                    <span gravatar name="{{ global.user.name }}" email-
    hash="{{ emailHash }}" width="300" height="300"></span>
23                </div>
24
25                <a style="align-right" data-toggle="modal" data-target="#
    myModal">
26                    How can I set my profile picture?
27                </a>
28            </div>
29        </div>
30
31
32        <!-- Modal -->
33        <div class="modal fade" id="myModal" tabindex="-1" role="dialog"
    aria-labelledby="myModalLabel" aria-hidden="true">
34          <div class="modal-dialog">
35            <div class="modal-content">
```

```
36              <div class="modal-body">
37                <div class="alert-message alert-message-info">
38                    <h4>
39                        How can I set my profile picture?</h4>
40                    <p>
41                        In this application are used <b>Gravatar</b>. A
    Gravatar is an image that follows you from site to site and will
    appear next to your name being associated to your email address.<br>
42                        To change your profile picture, sign up on
    Gravatar.com (<a href="https://gravatar.com/" target="_blank">link</a
    >) using the email address you use to log on to this platform, or, if
    you are already registered, you can simply change your Gravatar image
    at <a href="https://gravatar.com/emails/" target="_blank">this link</a
    >.
43                    </p>
44                </div>
45              </div>
46              <div class="modal-footer">
47                <button type="button" class="btn btn-default" data-dismiss
    ="modal">OK</button>
48              </div>
49            </div>
50          </div>
51        </div>
52
53    </div>
54</div>
55
56</div>
57
58<style>
59.alert-message
60{
61    margin: 20px 0;
62    padding: 20px;
63    border-left: 3px solid #eee;
64}
65.alert-message-info
```

```
{
    background-color: #f4f8fa;
    border-color: #5bc0de;
}
.alert-message-info h4
{
    color: #5bc0de;
}
</style>
```

4.3.2 My Profile

This page contains all the *informations about the user logged in* at the time; there is also a link to another page where you can edit this informations.

This is the code used to implement it:

```
<div class="row">
  <h1 class="section-header">My profile
  <button type="button" class="btn btn-primary btn-xs" ui-sref="
  dashboard.editProfile">
      <span class="glyphicon glyphicon-edit"></span> Edit profile
  </button></h1>

  <hr>
</div>

<div class="row" ng-controller="ProfileCtrl">
  <div class="col-xs-12 col-sm-12 col-lg-9 col-xs-offset-0 col-sm-offset
  -0 toppad" >
    <div>
      <div class="panel-heading">
        <h2>{{ _user.name }}</h2>
      </div>
      <div class="panel-body">
        <div class="row">
          <div class="col-md-4 col-lg-4" align="center">

```

```
20          <span gravatar name="{{ global.user.name }}" email-hash="{{
    emailHash }}" width="150" height="150"></span>
21
22          <br></br></div>
23
24          <div class="col-md-8 col-lg-8">
25            <table class="table table-user-information">
26              <tbody>
27                <tr>
28                  <td class="col-lg-4"> Username:</td>
29                  <td>{{ _user.username }}</td>
30                </tr>
31                <tr>
32                  <td>First Name:</td>
33                  <td id="firstNameField">{{ _user.firstName }}</td>
34                </tr>
35                <tr>
36                  <td>Last Name:</td>
37                  <td id="lastNameField">{{ _user.lastName }}</td>
38                </tr>
39
40                <tr>
41                <tr>
42                  <td>Address:</td>
43                  <td id="addressField">{{ _user.address }}</td>
44                </tr>
45                <tr>
46                  <td>Email:</td>
47                  <td id="emailField">{{ _user.email }}</td>
48                </tr>
49                  <td>Phone Number:</td>
50                  <td id="phoneField">{{ _user.phone }}
51                  </td>
52
53                </tr>
54
55              </tbody>
56            </table>
```

```
57
58              </div>
59            </div>
60          </div>
61      </div>
62      </div>
63    </div>
```

4.3.3 Execute Query

This view allows you to *query the database*. It is possible to run two types of queries: simple queries and range queries.For this reason it is divided into two tabs.
Once queried, at the bottom is shown a table containing the results of the query itself (for details on the query that you can do see previous chapters).

This is the code used to implement it:

```
1 <div id="main" class="col-lg-11">
2
3 <h1 hidden id="iddi">{{global.user._id}}</h1>
4
5 <div class="row">
6   <h1 class="section-header">Execute Query</h1>
7   <hr>
8 </div>
9
10 <script type="text/javascript">
11    function changeLabel(value) {
12        var e = document.getElementById('search_concept');
13        e.innerHTML = value;
14    }
15
16    function changeLabel2(value) {
17        var e = document.getElementById('refine_value');
18        e.innerHTML = value;
19    }
20
21    function changeLabel3(value) {
```

```
22          var e = document.getElementById('refine_value_first');
23          e.innerHTML = value;
24      }
25
26      function changeLabel4(value) {
27          var e = document.getElementById('refine_value_second');
28          e.innerHTML = value;
29      }
30  </script>
31
32  <ul class="nav nav-tabs" ng-controller="ExecuteQueryCtrl">
33        <li class="text-left"><a href="#basic" data-toggle="tab" ng-click="
      clear()">Basic SNPs Search</a></li>
34        <li class="text-left"><a href="#complex" data-toggle="tab" ng-click=
      "clear()">Range SNPs Search</a></li>
35
36      </ul>
37
38      <!-- First tab -->
39      <div id="myTabContent" class="tab-content" ng-controller="
      ExecuteQueryCtrl">
40        <div class="tab-pane active in" id="basic">
41          <div class="row bot15">
42                  <div id="custom-search-input">
43                  <h4>Choose what to search and then insert keywords:</h4>
44                  <div class="input-group col-md-6">
45                    <div class="input-group">
46                      <div class="input-group-btn search-panel">
47                        <button type="button" class="btn btn-default
      dropdown-toggle" data-toggle="dropdown">
48                          <span id="search_concept">Search by...</span> <
      span class="caret"></span>
49                        </button>
50                        <ul class="dropdown-menu" role="menu">
51                          <li><a href="#patient" ng-click="elValue('patient
      ')" onClick="changeLabel('Patient')">Patient</a></li>
52                        <li><a href="#gene" ng-click="elValue('gene')"
      onClick="changeLabel('Gene')">Gene</a></li>
```

```
53                      <li><a href="#region" ng-click="elValue('region')"
    onClick="changeLabel('Region')">Region</a></li>
54                      <li><a href="#mutation"  ng-click="elValue('
    mutation')" onClick="changeLabel('Mutation')">Mutation</a></li>
55                      <li><a href="#genotype"  ng-click="elValue('
    genotype')" onClick="changeLabel('Genotype')">Genotype</a></li>
56                      <li><a href="#freqAlt"  ng-click="elValue('freqAlt
    ')" onClick="changeLabel('Freq alt')">Freq alt</a></li>
57                      <li><a href="#dbsnp"  ng-click="elValue('dbsnp')"
    onClick="changeLabel('dbSNP')">dbSNP</a></li>
58                    </ul>
59                  </div>
60                  <input type="hidden" name="search_param" value="all"
    id="search_param" />
61                  <input id = "key" ng-model="query.keyword" type="
    search" class="search-query form-control" placeholder="Type Here" />
62                  <span class="input-group-btn">
63                    <button class="btn btn-danger" type="button" ng-
    click="submitBase()">
64                      <span class="glyphicon glyphicon-search"></span>
65                    </button>
66                  </span>
67                </div>
68              </div>
69            </div>
70          </div>
71
72          <div class="row input-group-btn search-panel">
73            <div id="firstCombobox" class="col-md-4">
74              <button type="button" class="btn btn-default dropdown-
    toggle" data-toggle="dropdown">
75                <span id="refine_value_first">Choose Mutation...</span>
    <span class="caret"></span>
76              </button>
77              <ul class="dropdown-menu" role="menu">
78                <li><a ng-click="elValue('patient')" onClick="
    changeLabel3('Patient');">Patient</a></li>
```

```
79                  <li><a ng-click="elValue('gene')" onClick="changeLabel3
    ('Region');">Region</a></li>
80                  <li><a ng-click="elValue('gene')" onClick="changeLabel3
    ('Mutation');">Mutation</a></li>
81                </ul>
82              </div>
83
84              <div id="secondCombobox" class="col-md-4">
85                <button type="button" class="btn btn-default dropdown-
    toggle" data-toggle="dropdown">
86                  <span id="refine_value_second">Choose Region...</span> <
    span class="caret"></span>
87                </button>
88                <ul class="dropdown-menu" role="menu">
89                  <li><a ng-click="elValue('patient')" onClick="
    changeLabel4('Patient');">Patient</a></li>
90                  <li><a ng-click="elValue('gene')" onClick="changeLabel4
    ('Region');">Region</a></li>
91                  <li><a ng-click="elValue('gene')" onClick="changeLabel4
    ('Mutation');">Mutation</a></li>
92                </ul>
93              </div>
94            </div>
95
96
97        <div class="row" style="height: 500px; overflow-y: auto; margin-
    top: 50px">
98            <h4>Results:</h4>
99        <h4 ng-hide="elements.length"> No Results! </h4>
100         <div ng-show="elements && elements.length">
101           <table class="table table-striped table-fixed-header">
102             <thead>
103               <tr>
104                 <th>Chr</th>
105                 <th>Start</th>
106                 <th>End</th>
107                 <th>Ref</th>
108                 <th>Alt</th>
```

```
109                 <!--<th style="width: 75px;"></th>-->
110               </tr>
111             </thead>
112             <tbody>
113                 <tr ng-repeat="el in elements">
114                 <td>{{el.chr}}</td>
115                 <td>{{el.start}}</td>
116                 <td>{{el.end}}</td>
117                 <td>{{el.ref}}</td>
118                 <td>{{el.alt}}</td>
119                 <!--<td>
120                   <button type="button" class="btn btn-primary btn-sm"
    data-ng-hide="user.isLicensed"
121                   data-toggle="modal" data-target="#myModalLicensed">
122                   <span class="glyphicon glyphicon-eye-open"></span>
    View
123                   </button>
124                 <br/>
125                 </td>-->
126                 </tr>
127             </tbody>
128           </table>
129         </div>
130       </div>
131     </div>
132     <!-- First tab end -->
133
134
135
136     <!-- Second tab -->
137     <div class="tab-pane fade" id="complex">
138     <div class="tab-pane active in" id="basic">
139
140       <div>
141           <div class="tab-pane" id="region">
142           <div class="row bot15">
143                     <div id="custom-search-input">
144                     <h4>Insert required fields:</h4>
```

```
145                     <div class="input-group col-md-12">
146                       <div class="input-group">
147                         <div class="input-group-btn search-panel">
148                           <input type="text" ng-model="chr" class="form-
    control" name="x" placeholder="chr">
149                         </div>
150                         <div class="input-group-btn search-panel">
151                           <input type="text" ng-model="start" class="
    form-control" name="x" placeholder="start">
152                         </div>
153                         <div class="input-group-btn search-panel">
154                           <input type="text" ng-model="end" class="form-
    control" name="x" placeholder="end">
155                         </div>
156                       <span class="input-group-btn">
157                           <button class="btn btn-danger" type="button"
    ng-click="submitByRegion()">
158                             <span class="glyphicon glyphicon-search"></
    span>
159                           </button>
160                         </span>
161                       </div>
162                       </div>
163                     </div>
164                   </div>
165
166                   <div class="row input-group-btn search-panel">
167                     <div id="value" class="col-md-4">
168                     <button type="button" class="btn btn-default
    dropdown-toggle" data-toggle="dropdown">
169                       <span id="refine_value">Refine by...</span> <span
    class="caret"></span>
170                     </button>
171                     <ul class="dropdown-menu" role="menu">
172                       <li><a ng-click="elValue('patient')" onClick="
    changeLabel2('dnSNP not specified'); document.getElementById('
    valueFreq').style.display='none';">dnSNP not specified</a></li>
```

```
173                         <li><a ng-click="elValue('gene')" onClick="
    changeLabel2('FreqRef smaller than...'); document.getElementById('
    valueFreq').style.display='block';">FreqRef smaller than...</a></li>
174                         <li><a ng-click="elValue('gene')" onClick="
    changeLabel2('FreqAlt smaller than...'); document.getElementById('
    valueFreq').style.display='block';">FreqAlt smaller than...</a></li>
175                       </ul>
176                     </div>
177
178                     <div style="display:none;" id="valueFreq" class="col-
    md-4">
179                       <input type="text" class="form-control" placeholder=
    "value">
180                     </div>
181
182                   </div>
183
184                    <div class="row">
185             <h4>Results:</h4>
186         <h4 ng-hide="elements.length"> No Results! </h4>
187           <div ng-show="elements && elements.length">
188             <table class="table table-striped  ">
189               <thead>
190                 <tr>
191                   <th>Chr</th>
192                   <th>Start</th>
193                   <th>End</th>
194                   <th>Ref</th>
195                   <th>Alt</th>
196                   <!--<th style="width: 75px;"></th>-->
197                 </tr>
198               </thead>
199               <tbody>
200                   <tr ng-repeat="el in elements">
201                   <td>{{el.chr}}</td>
202                   <td>{{el.start}}</td>
203                   <td>{{el.end}}</td>
204                   <td>{{el.ref}}</td>
```

```
205                     <td>{{el.alt}}</td>
206                     <!--<td>
207                         <button type="button" class="btn btn-primary btn-sm"
     data-ng-hide="user.isLicensed"
208                         data-toggle="modal" data-target="#myModalLicensed">
209                         <span class="glyphicon glyphicon-eye-open"></span>
    View
210                         </button>
211                     <br/>
212                 </td>-->
213                 </tr>
214               </tbody>
215             </table>
216             </div>
217             </div>
218                 </div>
219
220             </div>
221           </div>
222           </div>
223           </div>
224           <!-- Second tab end -->
225
226   </div>
227 </div>
```

4.3.4 Sequencing Upload

This view is probably the most important part of any application. In fact it is one that *allows an administrator to load sequencing*, the datas that the application manages.
To upload a file, simply drag and drop the file to be loaded into the panel pointed out; however, the panel will appear only after you specify the name of the patient and press the "OK" button.
Once the loading is done, a table that displaying the contents of the file is shown. Next, you will be redirected to a view through which you can edit the information loaded and then save them.

This is the code used to implement it:

```
<div  id="main" class="col-lg-11" ng-show="global.user.roles.indexOf('
    admin') > -1">

  <div class="row">
    <h1 class="section-header">Sequencing Upload</h1>

    <hr>
  </div>


  <style>
  #drop{
     border:2px dashed #bbb;
     -moz-border-radius:5px;
     -webkit-border-radius:5px;
     border-radius:5px;
     padding:100px;
     text-align:center;
     font:20pt bold,"Vollkorn";color:#bbb
  }
  #b64data{
     width:100%;
  }
  </style>

<div class="row bot15" ng-controller="UploaderCtrl">
  <h4>Insert patient's name to start:</h4>

  <div id="custom-search-input" class="col-md-10 bot15">
      <div class="input-group col-md-6 pull-left">
        <input type="search" class="search-query form-control" placeholder
    ="Patient's name" ng-model="patient" />
        <button hidden data-ng-show="false" id="btn_save" type="button"
    class="btn btn-success btn-xs ng-hide"  ng-click="saveResult()" />
        <span class="input-group-btn">
```

```
33          <button class="btn btn-danger" type="button" onClick="document.
    getElementById('uploadMain').style.display='block';">
34            <span>OK</span>
35          </button>
36        </span>
37      </div>
38  </div>
39 </div>
40
41 <div style="display:none;" id="uploadMain" class="row bot15" ng-controller
    ="UploaderCtrl">
42   <h4>Drag and drop into the box below your sequencing file to start
     uploading it:</h4>
43
44   <div id="drop" style="margin-bottom: 50px; margin-top: 50px">Drop an XLS
     or XML (2003) file here...</div>
45
46   <!--   <span class="glyphicon glyphicon-upload"></span> Show result
47   </button> -->
48
49 </div>
50
51 <!--<pre id="out"></pre>-->
52
53
54 <!-- Per progress bar -->
55 <script src="public/system/lib/jquery/dist/jquery.js"></script>
56 <script src="public/system/lib/jquery-ui/jquery-ui.js"></script>
57 <script>
58
59 $(function() {
60     var progressTimer,
61       progressbar = $( "#progressbar" ),
62       progressLabel = $( ".progress-label" ),
63       dialog = $( "#dialog" ).dialog({
64         autoOpen: false,
65         closeOnEscape: false,
66         resizable: false,
```

```
67          beforeClose: function() {
68            uploadButton.button( "option", {
69              disabled: false,
70            });
71          }
72        }),
73        uploadButton = $( "#uploadButton" )
74          .button()
75          .on( "click", function() {
76            $( this ).button( "option", {
77              disabled: true,
78            });
79            dialog.dialog( "open" );
80          });
81
82      progressbar.progressbar({
83        value: false,
84      });
85
86    });
87
88 </script>
89
90 <div id="dialog" title="File Upload">
91   <div class="progress-label">Uploading... Please wait...</div>
92   <div id="progressbar"></div>
93 </div>
94 <button hidden id="uploadButton">Start Upload</button>
95
96 <!-- Fine progress bar -->
97
98 </div>
99
100
101 <br/>
102 <script src="public/system/assets/js/xls.js"></script>
103 <script>
```

```
104 var rABS = typeof FileReader !== "undefined" && typeof FileReader.
      prototype !== "undefined" && typeof FileReader.prototype.
      readAsBinaryString !== "undefined";
105
106 function fixdata(data) {
107   var o = "", l = 0, w = 10240;
108   for(; l<data.byteLength/w; ++l) o+=String.fromCharCode.apply(null,new
        Uint8Array(data.slice(l*w,l*w+w)));
109   o+=String.fromCharCode.apply(null, new Uint8Array(data.slice(o.length)))
        ;
110   return o;
111 }
112
113
114 function get_radio_value( radioName ) {
115   var radios = document.getElementsByName( radioName );
116   for( var i = 0; i < radios.length; i++ ) {
117     if( radios[i].checked || radios.length === 1 ) {
118       return radios[i].value;
119     }
120   }
121 }
122
123 function to_json(workbook) {
124   var result = {};
125   workbook.SheetNames.forEach(function(sheetName) {
126     var roa = XLS.utils.sheet_to_row_object_array(workbook.Sheets[
        sheetName]);
127     if(roa.length > 0){
128       result[sheetName] = roa;
129     }
130   });
131   return result;
132 }
133
134 function to_csv(workbook) {
135   var result = [];
136   workbook.SheetNames.forEach(function(sheetName) {
```

```
137     var csv = XLS.utils.sheet_to_csv(workbook.Sheets[sheetName]);
138     if(csv.length > 0){
139       result.push("SHEET: " + sheetName);
140       result.push("");
141       result.push(csv);
142     }
143   });
144   return result.join("\n");
145 }
146
147 function to_formulae(workbook) {
148   var result = [];
149   workbook.SheetNames.forEach(function(sheetName) {
150     var formulae = XLS.utils.get_formulae(workbook.Sheets[sheetName]);
151     if(formulae.length > 0){
152       result.push("SHEET: " + sheetName);
153       result.push("");
154       result.push(formulae.join("\n"));
155     }
156   });
157   return result.join("\n");
158 }
159
160 var tarea = document.getElementById('b64data');
161 function b64it() {
162   if(typeof console !== 'undefined') console.log("onload", new Date());
163   var wb = XLS.read(tarea.value, {type: 'base64'});
164   process_wb(wb);
165 }
166
167 var output;
168 function process_wb(wb) {
169   output = "";
170   switch("json") {
171     case "json":
172       output = JSON.stringify(to_json(wb), 2, 2);
173       btn_save.click();
174       //filteredJson = filterOnlyAttributes(parse(output));
```

```
175       break;
176     case "form":
177       output = to_formulae(wb);
178       break;
179     default:
180       output = to_csv(wb);
181   }
182
183
184   document.getElementById('dialog').style.display = 'none';
185   document.getElementById('ui-id-1').innerHTML = "Done!";
186   document.getElementById('btn_save').disabled = "enable";
187
188   if(typeof console !== 'undefined') console.log("output", new Date());
189 }
190
191 var drop = document.getElementById('drop');
192 function handleDrop(e) {
193   e.stopPropagation();
194   e.preventDefault();
195   rABS = false;
196   var files = e.dataTransfer.files;
197   var i,f;
198
199   button = document.getElementById('uploadButton');
200   button.click();
201
202   for (i = 0, f = files[i]; i != files.length; ++i) {
203     var reader = new FileReader();
204     var name = f.name;
205     reader.onload = function(e) {
206
207
208       if(typeof console !== 'undefined') console.log("onload", new Date(),
      rABS, false);
209       var data = e.target.result;
210
211       var wb;
```

```
212       if(rABS) {
213         wb = XLS.read(data, {type: 'binary'});
214       } else {
215         var arr = fixdata(data);
216         wb = XLS.read(btoa(arr), {type: 'base64'});
217       }
218       process_wb(wb);
219     };
220     if(rABS) reader.readAsBinaryString(f);
221     else reader.readAsArrayBuffer(f);
222   }
223 }
224
225 function handleDragover(e) {
226   e.stopPropagation();
227   e.preventDefault();
228   e.dataTransfer.dropEffect = 'copy';
229 }
230
231 if(drop.addEventListener) {
232   drop.addEventListener('dragenter', handleDragover, false);
233   drop.addEventListener('dragover', handleDragover, false);
234   drop.addEventListener('drop', handleDrop, false);
235 }
236 </script>
```

4.3.5 Sequencing Upload History

Basically, this page displays a table containing all the *uploads that were made on the system*; so, for example, when you load an incorrect file, you can delete it using this table.

This is the code used to implement it:

```
1 <div class="col-lg-11" ng-show="global.user.roles.indexOf('admin') > -1">
2
3 <div class="row">
4   <h1 class="section-header">Sequencing Upload History</h1>
```

```
5
6   <hr>
7 </div>
8
9 <div class="row bot15" ng-controller="UploaderCtrl">
10   <h4>These are all the uploads you have done:</h4>
11
12   <div class="row" style="height: 500px; overflow-y: auto; margin-top: 50
       px">
13
14   <table class="table table-bordered" ng-show="!showEditFamily">
15     <thead>
16       <tr>
17         <th>Name</th>
18         <th>Date</th>
19         <th> </th>
20       </tr>
21     </thead>
22
23     <tbody>
24       <tr ng-repeat="upload in uploads">
25         <td>{{upload.name}}</td>
26         <td>{{upload.date}}</td>
27         <td>
28           <button type="button" class="btn btn-primary btn-sm">
29             <span class="glyphicon glyphicon-trash" ></span> Delete
30           </button>
31         </td>
32       </tr>
33       <tr>
34       </tr>
35     </tbody>
36   </table>
37 </div>
38
39 </div>
40
41 </div>
```

4.3.6 Authorize Users

Show a table of *all members registered to the system*; you can register a new user, promote a user to the Admin role, demote a user to the Licensed role, filter users by means of a textbox. For each user is also shown his avatar, in addition to basic informations.

This is the code used to implement it:

```
<div class="col-lg-11" ng-controller="AuthorizerUserCtrl" ng-show="global.
    user.roles.indexOf('admin') > -1">

<div class="row">
  <h1 class="section-header">Authorize Users</h1>

  <hr>
</div>

<div class="row">

  <div id="custom-search-input" class="col-md-10 bot15">
    <div class="input-group col-md-4 pull-left">
      <input type="search" class="search-query form-control" placeholder="
    Type here" ng-model="searchUser" />
      <span class="input-group-btn">
        <button class="btn btn-danger" type="button">
          <span class="glyphicon glyphicon-search"></span>
        </button>
      </span>
    </div>
  </div>

  <div class="col-md-2">
    <button type="button" class="btn btn-success btn-sm" data-toggle="
    collapse" data-parent="#operation" href="#formInsert">
      <span class="glyphicon glyphicon-plus"></span> Add new User
    </button>
  </div>
</div>

```

```
29
30 <!-- appearance panel -->
31 <div id="formInsert" class="top10 bot10 panel-collapse collapse out row
       col-md-6 col-md-offset-3 text-center panel panel-primary appearance">
32
33   <h4>Create user</h4>
34
35   <div class="panel-body">
36     <form class="form-horizontal">
37       <fieldset>
38
39         <div class="control-group">
40           <label class="control-label" for="username">Username</label>
41           <div class="controls">
42             <input id="username" name="username" type="text" placeholder="
       Username" class="form-control input-large" required=""
43               ng-model="user.username">
44
45           </div>
46         </div>
47
48         <div class="control-group">
49           <label class="control-label" for="name">First Name</label>
50           <div class="controls">
51             <input id="name" name="name" type="text" placeholder="First
       Name" class="form-control input-large" required=""
52                   ng-model="user.firstName">
53
54           </div>
55         </div>
56
57         <div class="control-group">
58           <label class="control-label" for="secondname">Last Name</label>
59           <div class="controls">
60             <input id="secondname" name="secondname" type="text"
       placeholder="Last Name" class="form-control input-large" required=""
61                   ng-model="user.lastName">
62
```

```
63          </div>
64        </div>
65
66        <div class="control-group">
67          <label class="control-label" for="email">Email</label>
68          <div class="controls">
69            <input id="email" name="email" type="text" placeholder="Email"
     class="form-control input-large" required=""
70                   ng-model="user.email">
71
72          </div>
73        </div>
74
75        <div class="control-group">
76          <label class="control-label" for="passwordUser">Password</label>
77          <div class="controls">
78            <input id="passwordUser" name="passwordUser" type="password"
    placeholder="Password" class="form-control input-large" required="" ng
    -model="user.password">
79
80          </div>
81        </div>
82
83        <div class="control-group">
84          <label class="control-label" for="confirmpasswordUser">Confirm
    Password</label>
85          <div class="controls">
86            <input id="ConfpasswordUser" name="ConffpasswordUser" type="
    password" placeholder="Password" class="form-control input-large"
    required="" ng-model="user.confirmPassword">
87
88          </div>
89        </div>
90
91        <div class="control-group">
92          <label class="control-label" for="saveUser"></label>
93          <div class="controls">
```

```
94            <button id="saveUser" name="saveUser" class="btn btn-success"
    ng-click="authorizeUser(user)">Save</button>
95            <button id="cancelUser" name="cancelUser" class="btn btn-
    danger" ng-click="cancelOperation()">Cancel</button>
96          </div>
97        </div>
98
99      </fieldset>
100    </form>
101  </div>
102</div>
103
104<div class="row" style="height: 500px; overflow-y: auto; margin-top: 50px"
    >
105  <div id="searchResults" class="row">
106
107      <table class="table table-bordered">
108        <thead>
109          <tr>
110            <th style="width: 100px;"></th>
111            <th>Username</th>
112            <th>First Name</th>
113            <th>Last Name</th>
114            <th style="width: 75px;"></th>
115          </tr>
116        </thead>
117        <tbody >
118          <tr ng-repeat="user in users | filter:searchUser">
119            <td>
120              <div class="col-md-3 col-lg-3 " align="center">
121                <span gravatar name="{{ user.name }}" email-hash="{{ user.
    emailHash }}" width="100" height="100"></span>
122              </div>
123            </td>
124            <td>{{ user.username }}</td>
125            <td>{{ user.firstName }}</td>
126            <td>{{ user.lastName }}</td>
127            <td>
```

```
128                <div class="control-groups">
129                  <button data-dz-remove class="btn btn-success delete btn-
    sm bot5" ng-click="promote2Admin(user)"
130                          ng-show="user.roles.indexOf('admin') === -1">
131                    <i class="glyphicon glyphicon-stair"></i>
132                    <span>Promote</span>
133                  </button>
134                  <button data-dz-remove class="btn btn-danger delete btn-sm
     bot5" ng-click="cancelUser(user)">
135                    <i class="glyphicon glyphicon-ban-trash"></i>
136                    <span>Dismiss</span>
137                  </button>
138                  <button data-dz-remove class="btn btn-primary delete btn-
    sm" ng-click="degrade2licensed(user)"
139                          ng-show="user.roles.indexOf('admin') !== -1">
140                    <i class="glyphicon glyphicon-ban-circle"></i>
141                    <span>Degrade</span>
142                  </button>
143                </div>
144              </td>
145            </tr>
146          </tbody>
147        </table>
148
149    </div>
150 </div>
151
152 </div>
```

4.3.7 Families

Is the view that enables you to *manage families* (groups of patients, relatives or not). All the actions actions you can do are performed by pop-up; you can: create a family, remove a family, modify a family, modify the members that make up the family.

This is the code used to implement it:

```
<div id="main" ng-controller="FamilyCtrl" ng-show="global.user.roles.indexOf('admin') > -1">

  <div class="row">

    <h1 class="section-header">Families</h1>

    <hr>
  </div>


      <div class="row bot10" id="operation">
        <div class="col-md-6">
          <div class="input-group col-md-8">
            <input type="search" class="search-query form-control" placeholder="Type here"
              ng-model="searchFamily" />
            <span class="input-group-btn">
              <button class="btn btn-danger" type="button">
                <span class="glyphicon glyphicon-search"></span>
              </button>
            </span>
          </div>
        </div>

        <div class="controls-group col-md-3 pull-right">
          <button type="button" class="btn btn-success btn-sm"
                  data-toggle="collapse" data-parent="#operation" href="#formInsert">
            <span class="glyphicon glyphicon-plus"></span> Add new
          </button>

          <button id="btn_edit" type="button" class="btn btn-primary btn-sm" ng-click="showEditFamily = !showEditFamily"  href="#patientsPanel" >
            <div ng-show="!showEditFamily"><span class="glyphicon glyphicon-pencil"></span> Edit</div>
```

```
32          <div ng-show="showEditFamily"><span class="glyphicon glyphicon
    -floppy-save"></span>Finish</div>
33        </button>
34
35        <!--
36        <button id="btn_finish" type="button" class="btn btn-primary btn
    -sm" ng-click="finishedOperation()"
37                ng-disabled="!showEditFamily">
38          <span class="glyphicon glyphicon-ok" href="#patientsPanel"></
    span> Finish
39        </button>-->
40      </div>
41    </div>
42
43    <!-- appearance panel -->
44    <div id="formInsert" class="top10 bot10 panel-collapse collapse out
    row col-md-6 col-md-offset-3 text-center panel panel-primary
    appearance">
45
46      <h4>Create family</h3>
47
48      <div class="panel-body">
49        <form class="form-horizontal">
50          <fieldset>
51
52            <div class="control-group">
53              <label class="control-label" for="nameFamily">Name</label>
54              <div class="controls">
55                <input id="nameFamily" name="nameFamily" type="text"
    placeholder="" class="form-control input-xlarge" ng-model="family.name
    ">
56              </div>
57            </div>
58
59            <div class="control-group top10">
60              <div class="controls">
61                <button id="save_btn" name="save_btn" class="btn btn-
    success" ng-click="createFamily(family)">Save</button>
```

```
62                  <button id="cancel_btn" name="cancel_btn" class="btn btn
    -danger" ng-click="cancel()">Cancel</button>
63                </div>
64              </div>
65
66            </fieldset>
67          </form>
68        </div>
69      </div>
70
71      <div id="patientsPanel" class="top10 bot10 panel-collapse collapse
    out row appearance">
72        <div class="panel panel-primary col-md-6">
73          <h4>Patients out family</h3>
74
75          <ul class="list-group">
76            <li class="list-group-item" ng-repeat="patient in outPatients"
    >
77              <div class="row">
78                  <div class="pull-left">{{ patient.name }}</div>
79                  <div class="pull-right">
80                    <button type="button" class="btn btn-success btn-sm"
    ng-click="addPatient(patient)">
81                          <span class="glyphicon glyphicon-plus"></span>
82                    </button>
83                  </div>
84              </div>
85            </li>
86          </ul>
87
88        </div>
89
90        <div class="panel panel-primary col-md-6">
91          <h4>Patients in family</h3>
92
93          <ul class="list-group" >
94              <li class="list-group-item" ng-repeat="patient in inPatients
    ">
```

```
95              <div class="row">
96                <div class="pull-left">{{ patient.name }}</div>
97                <div class="pull-right">
98                  <button type="button" class="btn btn-danger btn-sm" ng
    -click="removePatient(patient)">
99                          <span class="glyphicon glyphicon-trash"></span
    >
100                 </button>
101               </div>
102             </div>
103           </li>
104
105       </ul>
106     </div>
107   </div>
108
109   <div id="showPanel" class="top10 bot10 panel-collapse collapse out
    row appearance">
110     <div class="panel panel-primary col-md-6 col-md-offset-3">
111       <h4>Panel in family</h3>
112         <table class="table table-bordered pull-center">
113           <thead>
114             <tr class="">
115               <th>Name</th>
116             </tr>
117           </thead>
118           <tbody>
119             <tr ng-repeat="patient in inPatients">
120               <td>{{ patient.name }}</td>
121             </tr>
122           </tbody>
123         </table>
124     </div>
125   </div>
126
127
128   <div class="row" style="height: 500px; overflow-y: auto; margin-top:
     50px">
```

```
129         <div>
130           <table class="table table-bordered" ng-show="showEditFamilies
    (!showEditFamily)">
131             <thead>
132               <tr>
133                 <th>Name</th>
134                 <th>Members</th>
135               </tr>
136             </thead>
137             <tbody>
138               <tr ng-repeat="family in families | filter:searchFamily ">
139                   <td>{{ family.name }}</td>
140                   <td id="parents">
141                     <button type="button" class="btn btn-default btn-sm
    col-md-3 col-md-offset-4"
142                             ng-click="showPatients(family)"
143                             data-toggle="collapse" data-parent="#parents
    " href="#showPanel">
144                             <span class="glyphicon glyphicon-user"></
    span> Show Patients
145                     </button>
146                   </td>
147               </tr>
148               <tr>
149               </tr>
150             </tbody>
151           </table>
152
153           <table class="table table-bordered" ng-show="showEditFamilies(
    showEditFamily)">
154             <thead>
155               <tr>
156                 <th>Name</th>
157                 <th>Operation</th>
158               </tr>
159             </thead>
160             <tbody>
161               <tr ng-repeat="family in families | filter:searchFamily ">
```

```
162                     <td>
163                       <div>{{ family.name }}</div>
164                       <!--
165                       <input id="nameUpdate" name="nameUpdate" type="text"
    value="{{ family.name }}" class="form-control input-xlarge" ng-model=
    "familyUpdate.name" ng-show="showConfirm">-->
166                     </td>
167                     <td>
168                       <div class="control-group pull-right" ng-show="!
    showConfirm">
169                         <div class="controls" id="controlsBtn">
170                           <!--<button type="button" class="btn btn-success
     btn-sm" ng-click="showConfirm = !showConfirm">
171                             <span class="glyphicon glyphicon-pencil"></
    span> Edit Name
172                           </button>-->
173                           <button type="button" class="btn btn-primary btn
    -sm"
174                                   data-toggle="collapse" data-parent="#
    controlsBtn" href="#patientsPanel"
175                                   ng-click="showPatients(family)">
176                             <div ><span class="glyphicon glyphicon-pencil"
    ></span> Edit Patients</div>
177                             <!--<div ng-show="showEditFamily"><span class=
    "glyphicon glyphicon-pencil"></span> Finishs Patents</div>-->
178                           </button>
179                           <button type="button" class="btn btn-danger btn-
    sm" ng-click="removeFamily(family)">
180                             <span class="glyphicon glyphicon-trash"></span
    > Remove
181                           </button>
182                         </div>
183                       </div>
184
185                       <div class="control-group pull-right" ng-show="
    showConfirm">
186                         <div class="controls">
```

```
187                             <button type="button" class="btn btn-success btn
    -sm" ng-click="updateFamily(family, familyUpdate)">
188                               <span class="glyphicon glyphicon-floppy-save"
    ></span> OK
189                             </button>
190                             <button type="button" class="btn btn-danger btn-
    sm" ng-click="showConfirm = false">
191                               <span class="glyphicon glyphicon-ban-circle"
    ></span> Cancel
192                             </button>
193                           </div>
194                         </div>
195                       </td>
196                   </tr>
197                   <tr>
198                   </tr>
199                 </tbody>
200               </table>
201             </div>
202         </div>
203 </div>
```