UNIVERSITÀ DEGLI STUDI ROMA TRE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Web-SNP

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

Anno Accademico
2013/2014

Introduction

In this chapter, we will explain creation of database, from code point of view and design point of view. As explained in introduction chapter, the impl
lementaion of the database is made using MongoDB, the noSQL DBMS, through mongoose, the JavaScript interface beetween the program and the database.
The approach followed by this technique allows a more efficient way of handling those objects. In fact, using mongoose's schemas to define objects of the
database, permits to treat database models like any JavaScript object. To achieve this, MongoDB uses documents, a set of key-value pairs, stored in
collections, held by a database.
In the following part, we will see objects that form our database model.

Mongoose's syntax

To realize an object using mongoose, we need to follow a specific syntax.
First of all, you must place all model files related to the diagrams of the model in a specific folder within the project. Each of these files must be placed in a specific folder within the project /server/models/.// At the beginning of each file, we need to inform the database of the fact that any values passed to the builders of the models, which are not present in the schema should not be persisted. We do this by defining a string `'use strict'`
. then you must specify the modules dependencies required to create the schema. In our case, we must satisfy the dependency on the modules (bold type) 'mongoose' and 'crypto (a module for encrypting information.) //* We do this through the following code:

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

At this point you switch to define the schema itself, which consists of all the necessary components for the definition of the object in question, with the attributes and relationships. The syntax for doing this is explicated in the example, where it is supposed to want to model a user with an attribute **name** of type string:

```

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

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

Next, you 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, ie not persisted
- Pre-save hooks: functions called at the time immediately before the model is saved to the database
- Methods: all methods necessary for the operation of our model.

Finally, one must specify the actual name of the model from which the schema specified as previously shown, in addition to its creation proocedere real. We do this

through the following code: begin lstlisting mongoose.model ('User', UserSchema); end lstlisting

Model Schemas

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 /..).
Here it is the code to implement the family object:

```
'use strict';

/**
 * Module dependencies.
 */
var mongoose = require('mongoose'),
    Schema = mongoose.Schema,
    crypto = require('crypto');


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

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

    //Relationship
    patiens: [{
      type: Schema.Type.ObjectId,
      ref: 'Patient'
    }]

});
```

```
27
28
29 /**
30  * Validations
31  */
32 // nothing for now
33
34 /**
35  * Virtuals
36  */
37
38 // no not-persisted attributes
39 /*
40  * Pre-save hook
41  */
42 //still nothing
43
44 /**
45  * Methods
46  */
47 // no method required here. Query class
48
49
50 mongoose.model('Family', FamilySchema);
```

ATTRIBUTES

- The first item
- The second item
- The third etc ...

ciao