# SOCIALDEVELOP

CARLOS AVENDAÑO – KELWIN PAYARES
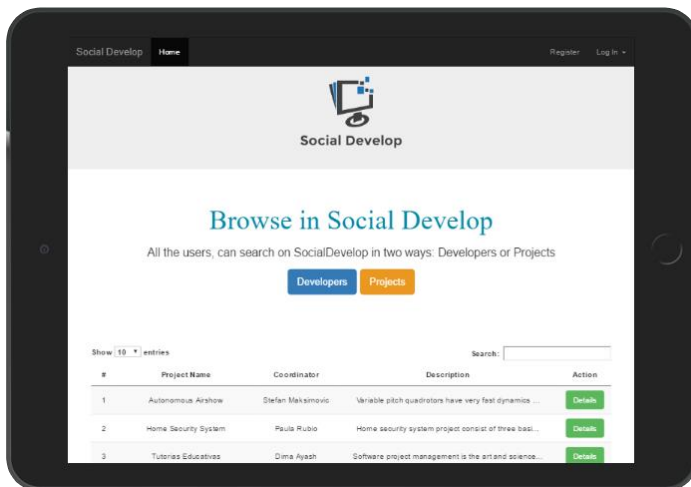
# Tabla de contenido

# 1. Description

Social Develop has been developed with SPRING Framework version 3.1, because we can write clean, testable code against the infrastructure components, keep it portable, and code with confidence because Spring provides an open programming model that is comprehensive, cohesive, widely understood and well-supported[1].

So we are talking about MVC framework.

All the views extend of a file called "layout.jsp", it has the following structure.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:set var="contextPath" value="${pageContext.request.contextPath}"/>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE html>
<html <c:if test="${init != null and init == 'home'}">ng-app="app"</c:if>>
    <head>
        <tiles:insertAttribute name="stylecss" />
        <title><tiles:insertAttribute name="title" /></title>
        <tiles:insertAttribute name="scriptjs" />
    </head>

    <body <c:if test="${init != null and init == 'home'}">ng-controller="AppController"</c:if>>
        <tiles:insertAttribute name="nav" />

        <tiles:insertAttribute name="header" />

        <tiles:insertAttribute name="body" />

        <tiles:insertAttribute name="footer" />
    </body>
</html>
```

*Figure 1. Layout.jsp*

In that sense we just have a nav, header and footer, and the unique thing that changes at any view will be the body that is good way for programing because allow an easier code maintainability.
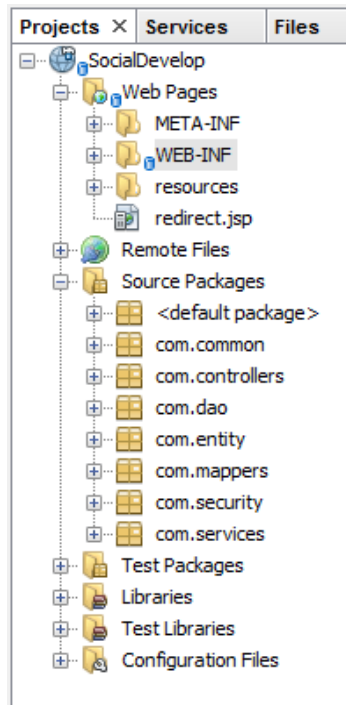
The code structure is clear as well.

---

[1] https://spring.io/

*Figure 2. Code structure*

It is possible to see MVC component in the code structure, we have the folder "Web Pages" for the Front end and "Source Packages" for the Back end, and we can see how are conformed the Model and the Controller, in "Web-INF" we have all the views.

In the figure 3 we can see how is conformed a body of a view and the respective validation.



*Figure 3. Body of - register.jsp*

```
1    function jsRegister() {
2        $('#NAME_USER').mask('AAAAAAAAAAAAAAAAAAAAA', {'translation':
3            A: {pattern: /[A-Za-z ]/}
4        }
5    });
6        $('#SURNAME').mask('AAAAAAAAAAAAAAAAAAAAA', {'translation': {
7            A: {pattern: /[A-Za-z ]/}
8        }
9    });
10
11        $('#PHONE').mask('000000000000');
12        $('#DOB').mask('0000/00/00', {placeholder: "____/__/__"});
13
14        $("#form_register").validate({
15            rules: {
16                NAME_USER: {
17                    required: true,
18                    minlength: 3
19                },
20                SURNAME: {
21                    required: true,
22                    minlength: 3
23                },
24                DOB: {
25                    required: true,
26                    dateISO: true,
27                    datePermit: true
28                },
29                PHONE: {
30                    required: true,
31                    minlength: 10
32                },
33                EMAIL: {
34                    required: true,
35                    email: true,
```

*Figure 3. Validation file – JsRegister ().*

We can see that our view we have only the body of them and each element has an id, that allow us give them toward a .js a validation, all the restriction that we have in the client side are assumption that we establish for all the field in the database.

*Figure 4. Register view*

But, what happened when we click in register and all the data are going to the controller.

```
110    @RequestMapping(value = "/register", method = RequestMethod.POST)
111    public String addUser(@RequestParam("NAME_USER") String NAME_USER,
112            @RequestParam("SURNAME") String SURNAME,
113            @RequestParam("DOB") String DOB,
114            @RequestParam("SEX") String SEX,
115            @RequestParam("PHONE") String PHONE,
116            @RequestParam("EMAIL") String EMAIL,
117            @RequestParam("PASSWORD") String PASSWORD,
118            @RequestParam("CURRICULUM") MultipartFile file,
119            ModelMap model) {
120
121        model.put("file", "Register");
122        model.put("menu", "register");
123        model.put("init", "home");
124
125        StringBuilder pathCurrent = new StringBuilder(request.getServletContext().getRealPath("/"));
126        pathCurrent.delete(pathCurrent.length() - 10, pathCurrent.length());
127        pathCurrent.append("web/resources/curriculums/");
128        if (!file.isEmpty()) {
129
130            try {
131                String CURRICULUM = file.getOriginalFilename();
132                File f = new File(pathCurrent.toString(), CURRICULUM);
133                if (f.exists()) {
134                    Calendar calendario = Calendar.getInstance();
135                    int anio = calendario.get(Calendar.YEAR),
136                        mes = calendario.get(Calendar.MONTH),
137                        dia = calendario.get(Calendar.DAY_OF_MONTH),
```

*Figure 5. Method register in the controller - 1*

When we have all the data they need to be inside in a try and catch in case of any problem with the file of the curriculum, because the file is mandatory and the validation has to be in the server side, after that all the controller request to a service for communicate with the model.

```
156                    usr.setPASSWORD(encryptPass(PASSWORD));
157                    usr.setCURRICULUM(CURRICULUM);
158
159                    TaskResult result = serv_user.getCurrentUser(usr.getEMAIL());
160
161                    if (result.isSuccess()) {
162                        if (result.getResultado() == null) {
163                            TaskResult result2 = serv_user.addUser(usr);
164                            if (result2.isSuccess()) {
165                                model.put("sucess", "Successfully registered");
166                            } else {
167                                model.put("error", result2.getMensaje());
168                            }
169                        } else {
170                            model.put("error", "Error: Mail already exists.");
171                        }
172                    } else {
173                        model.put("error", result.getMensaje());
174                    }
175
176                    return "Register-page";
                  } catch (Exception ex) {
178                    model.put("error", "Error: You failed to upload " + ex.getMessage());
179                  }
180              } else {
181                  model.put("error", "Error: You failed to upload curriculum because the file was empty");
182              }
183
184          return "Register-page";
185      }
```

*Figure 6. Method register in the controller – 2*

After we create the object user with all the data arriving from the view, we use a service (see in the line 163), addUser.

In the figure 7 we can see what that service does.

```
23   public class UserService implements UserDAO{
24
25       @Autowired
26       UserMapper mapper;
27
28       public TaskResult<Integer> addUser(User usr) {
29           try{
30               return TaskResult.FromResult(mapper.addUser(usr));
31           }catch(DataAccessException ex){
32               Logger.getLogger(UserService.class.getName()).log(Level.ERROR, "Failed add user. "+ ex);
33               return TaskResult.Failed("Failed add user. "+ex.getMessage());
34           }
35           catch(Exception e){
36               Logger.getLogger(UserService.class.getName()).log(Level.ERROR, "Failed add user. "+ e);
37               return TaskResult.Failed("Failed add user. "+e.getMessage());
38           }
39       }
```

*Figure 7. AddUser service – 2*

The service allow the communication between the controller and the model and the validation if any problem happen then will return to the controller if the request is right an integer otherwise will return a message with the exception that the model return.

We use a mapper addUser (see in the line 30), in the figure 8 we see how is conformed.

```
22   public interface UserMapper {
23
24       @Insert("INSERT INTO user(ID_USER, NAME_USER, SURNAME, DOB, SEX, PHONE, EMAIL, PASSWORD, CURRICULUM) VALUES (#{ID_USER},#{NAME_USER},#{SURNAME},#{DOB}
25       public Integer addUser(User usr)throws DataAccessException;
26
27       @Select("SELECT * FROM user WHERE EMAIL = #{email} AND STATE='1' ")
28       @Results(value = {
29           @Result (property = "ID_USER" , column = "ID_USER"),
30           @Result (property = "NAME_USER" , column = "NAME_USER"),
31           @Result (property = "SURNAME" , column = "SURNAME"),
32           @Result (property = "DOB" , column = "DOB"),
33           @Result (property = "SEX" , column = "SEX"),
34           @Result (property = "PHONE" , column = "PHONE"),
35           @Result (property = "EMAIL" , column = "EMAIL"),
36           @Result (property = "PASSWORD" , column = "PASSWORD"),
37           @Result (property = "CURRICULUM" , column = "CURRICULUM"),
38           @Result (property = "USER_TYPE" , column = "USER_TYPE")
39       })
40       public User getCurrentUser(String email)throws DataAccessException;
41
42       @Select("SELECT * FROM user WHERE ID_USER = #{ID_USER} AND STATE='1' ")
43       @Results(value = {
44           @Result (property = "ID_USER" , column = "ID_USER"),
45           @Result (property = "NAME_USER" , column = "NAME_USER"),
46           @Result (property = "SURNAME" , column = "SURNAME"),
47           @Result (property = "DOB" , column = "DOB"),
48           @Result (property = "SEX" , column = "SEX"),
```

*Figure 8. AddUser mapper – 2*

This contain the query in this case for insert the user to the database.

And after if going to return to the controller toward the service if the query was executed successfully or no, in each case we send a message to view (see in figure 6, after the line 164 until the end of the figure).

All the views, controllers and models follow the same behavior in this project.

All functionalities have been developed, mandatories and optional.

The project work with JavaScript disabled only for register and we use to login as amazon. One when JavaScript is enabled and another one when JavaScript disabled, it means allow you to access to the core even with JavaScript disabled.

# 2. Relational Schema of the Database.
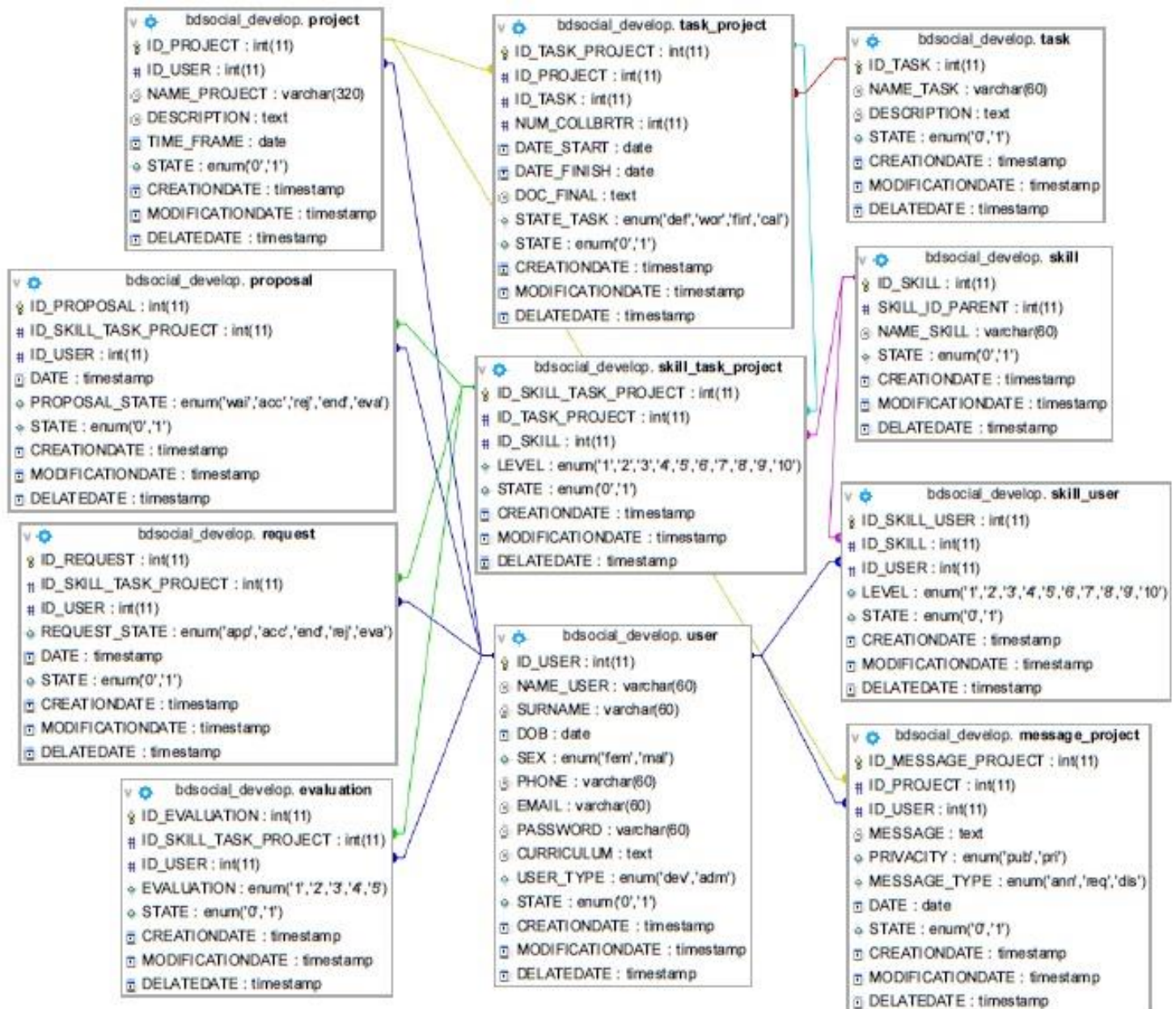


*Figure 9. Relational Schema of the Database*

This is the last version of the relational schema of the database, we use XAMPP 5.6.3 and MySQL as DBMS.

# 3. Diagram Structure Site

## 3.1. Home



*Figure 11. Home*

In home anonymous user can register or browse in social develop in two different ways as described in the figure 11, and user can log in.
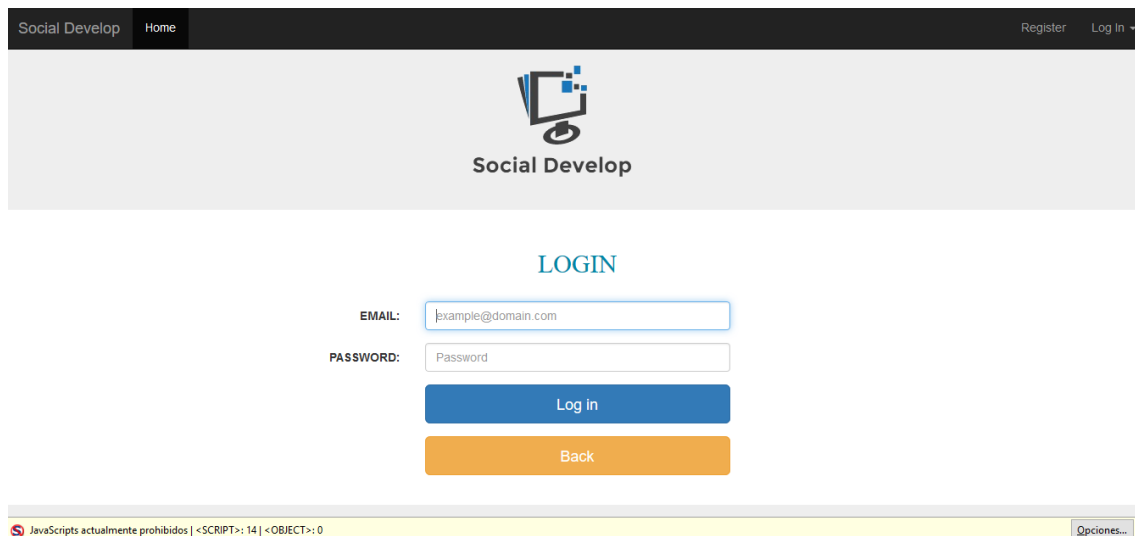


*Figure 12. Home responsive*
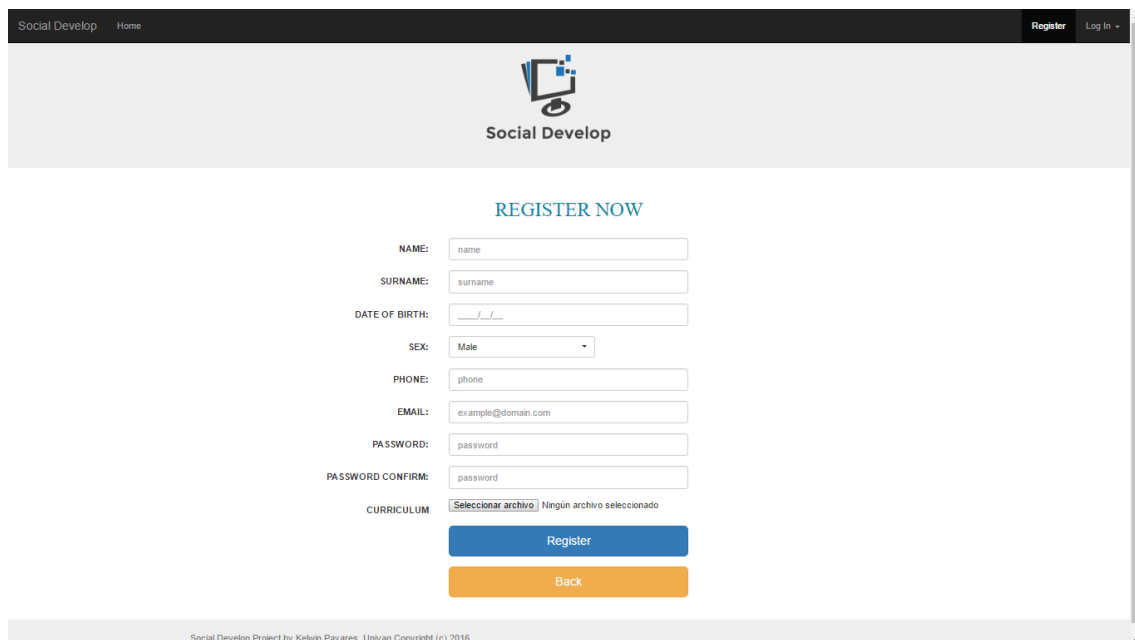
All the project is completely responsive. And when JavaScript is disabled and the click in Lon in, they will be redirected to the special login.



*Figure 13. Login when JavaScript is disabled*

Anonymous user can register themselves on social develop.



*Figure 14. Register*

Toward this view everyone can register.

## 3.2. Admin User

When an admin in logged we have the home as well but we have two option else.
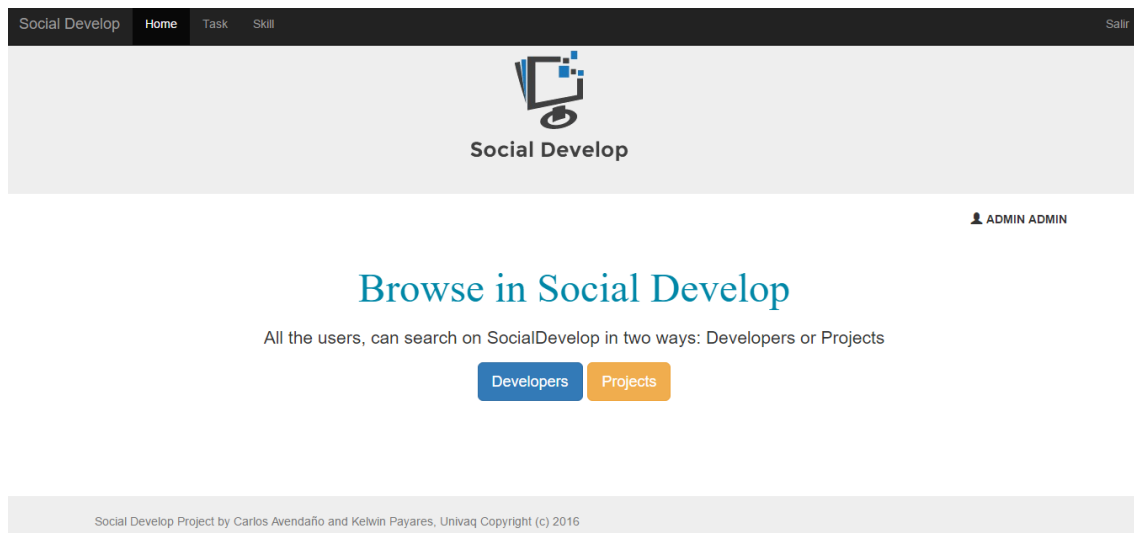


*Figure 14. Home Admin*

The other two options are task and skill.
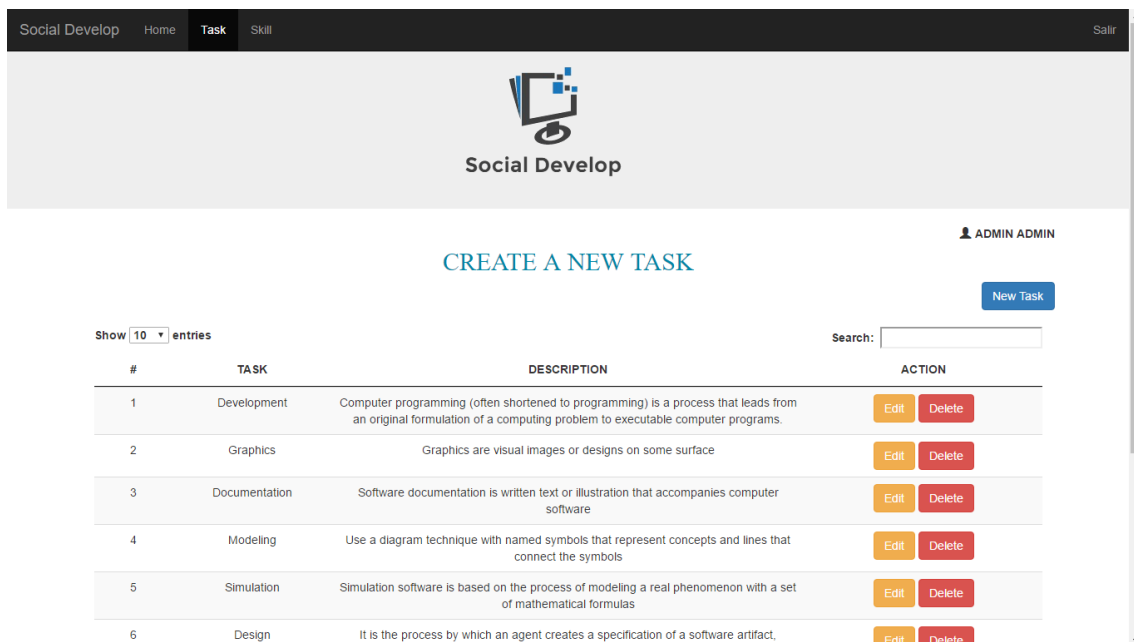


*Figure 15. Task Admin*

In this view, the admin can create, list, edit and delete task.

All our table have, a filter, a list for show how many items you want and pagination.
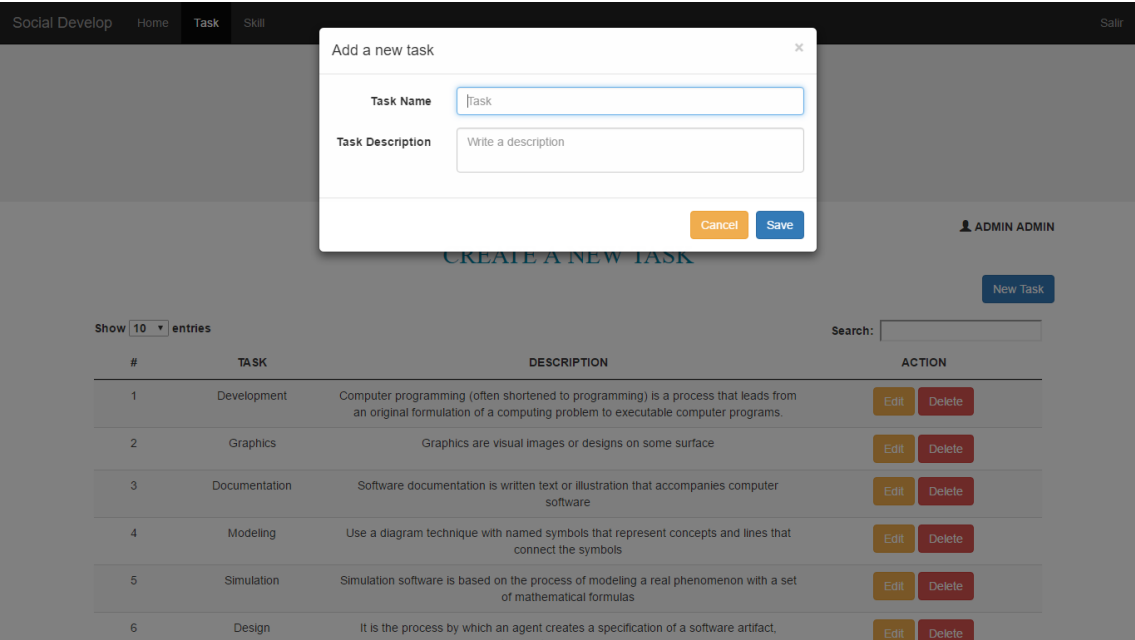


*Figure 16. Create a new task*
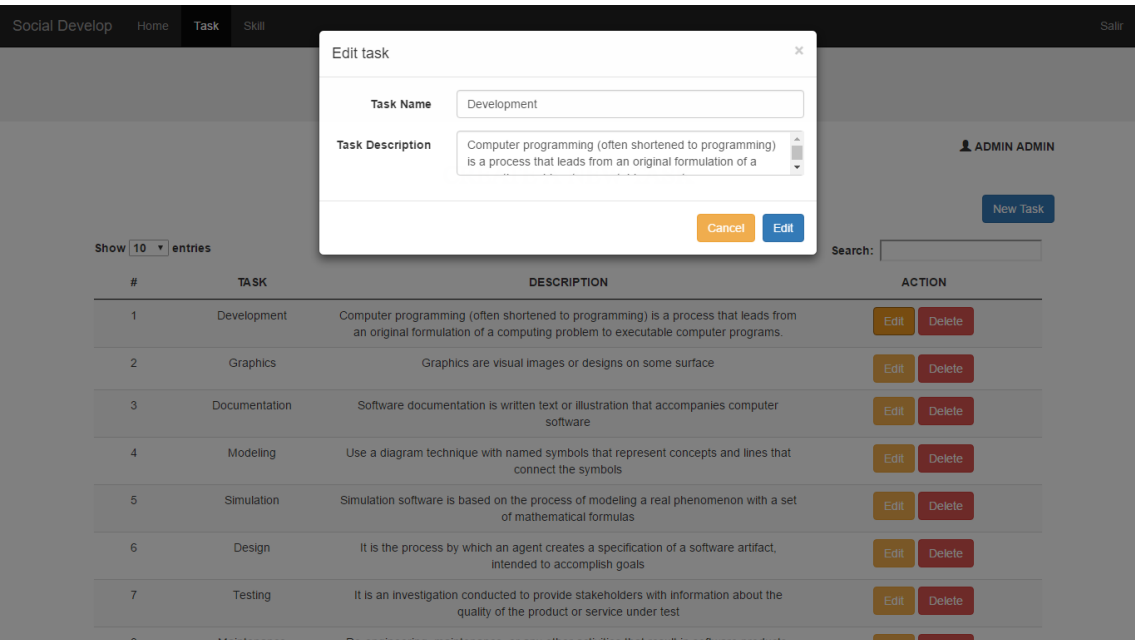


*Figure 17. Edit task*
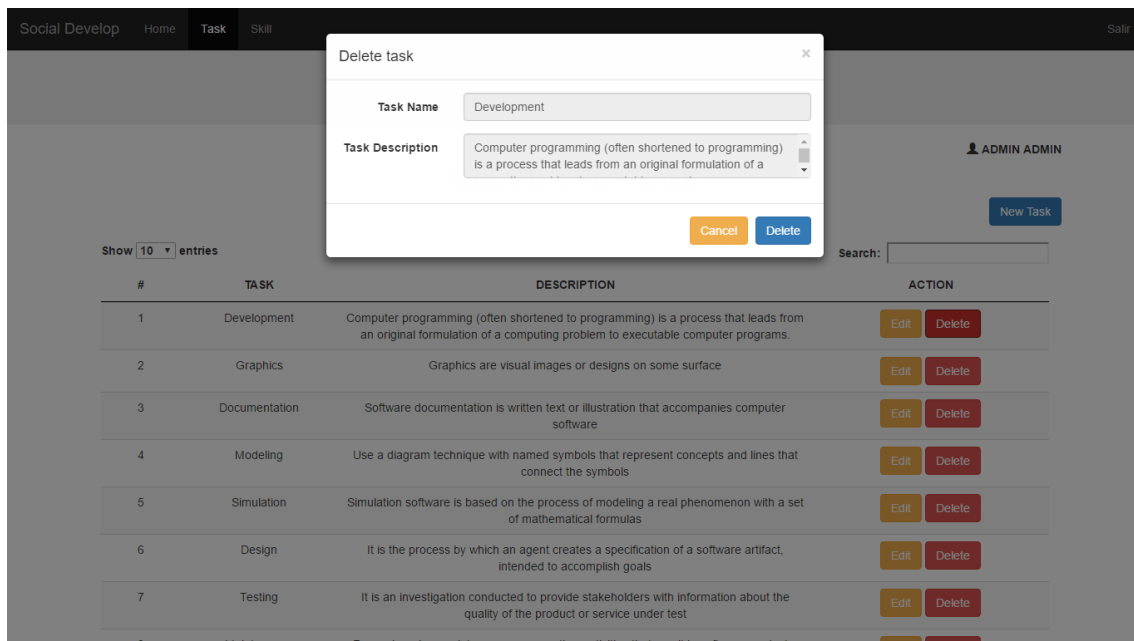
*Figure 18. Edit task*

We just add the functionality edit and delete in task view because in the same process for the skill (see image 19).
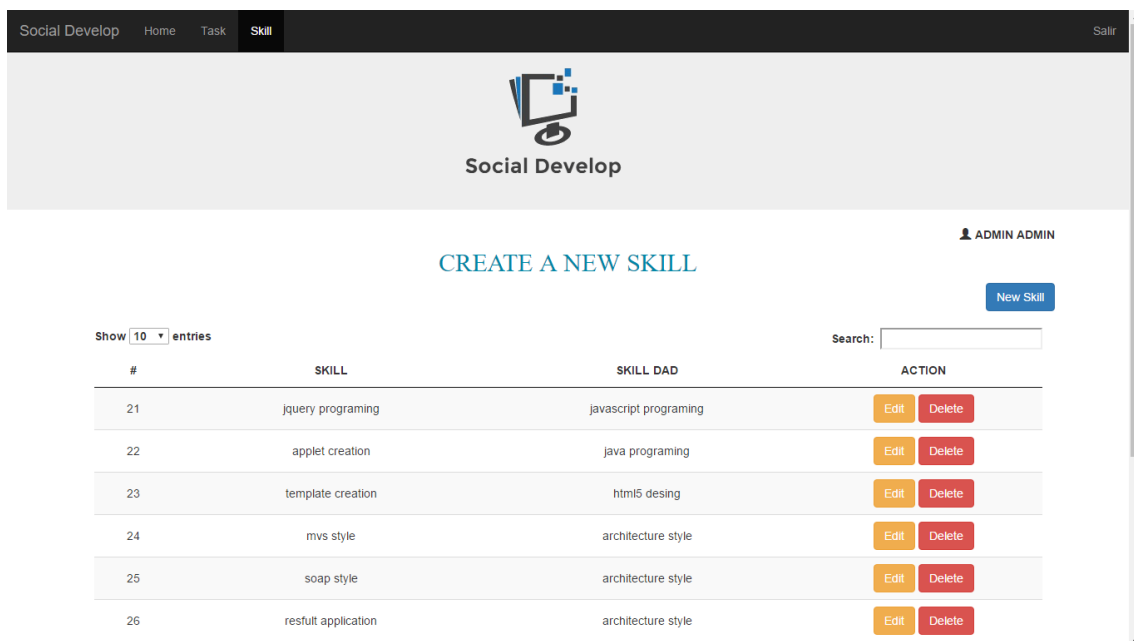


*Figure 19. Skill view*

It has been developed as an optional (Skills may optionally have a hierarchical structure).

## 3.3. User Logged

When an user is logged for the first time, he/she must associate skill to her/him profile, in this view it is possible list and associate skill.



*Figure 20. Associate skill view*



*Figure 21. Associate skill*
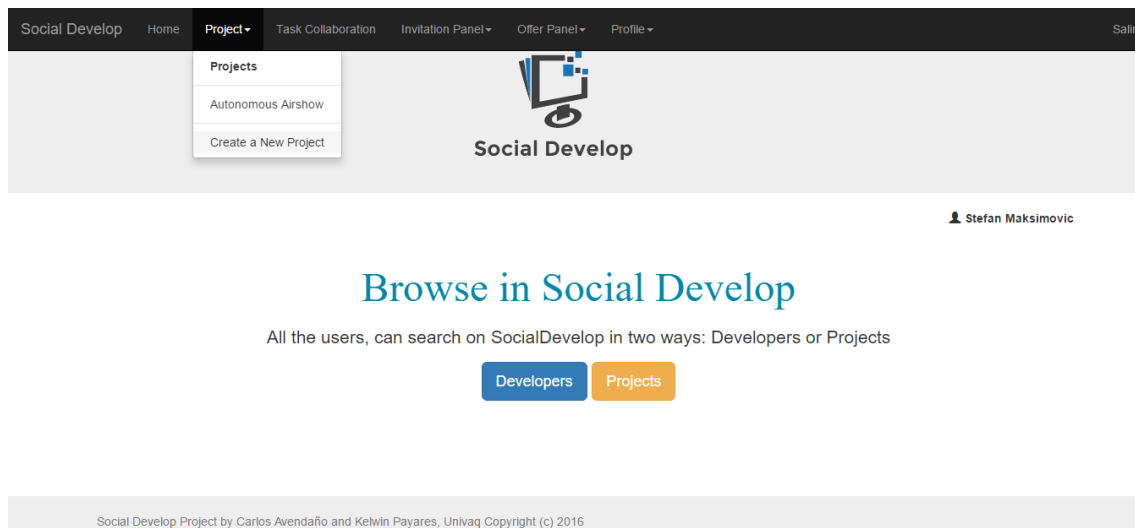
*Figure 22. Home user logged*

When the user is logged, it is going appear the complete menu with all the feature that a user can do, the first view is home, but from here the user can create a project or can entry to the projects already created example "Autonomous Airshow".



*Figure 23. Home user logged*

Any logged user can create a project. With those fields.

## 3.4. Project Definition



*Figure 24. Project definition view*

This view show the project definition, all the data that were using for create the project, the definition is composed, adding task in a project (see figure 25).



*Figure 25. Associate task*

And we can see that are listed all the task associated, inside the task we can associate skills (see figure 26).



*Figure 26. Skills inside a task*

And we can associate a new skill inside a task as well.



*Figure 27. Associate skill*

We are going to talk about collaborator after.

### 3.5.  Invitation Panel



*Figure 28. Invitation Panel*

The project coordinator can send to any registered developer a collaboration proposal for a particular task. He/she can see the profile of the developer.



*Figure 29. Proposal panel*

Each developer has a "proposals panel" that lists all the collaboration proposals sent by project coordinators as just described. And they can see detail of the project, accept or reject the proposal.
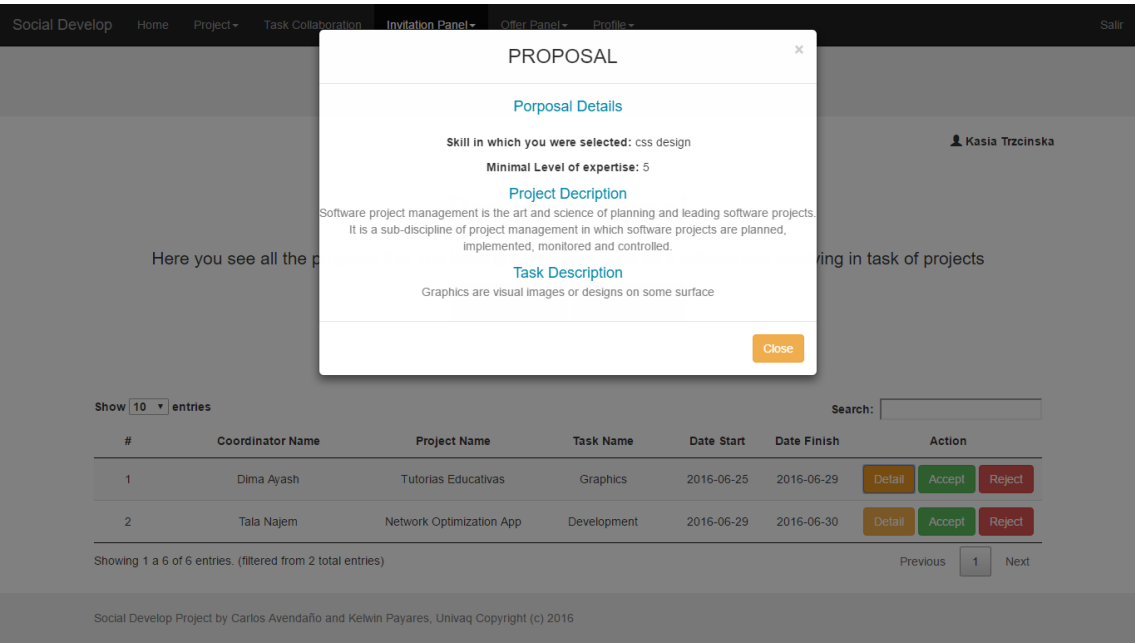


*Figure 30. Proposal details*

As optional feature we add the possibility to see the record of proposal that has been accepted or rejected.
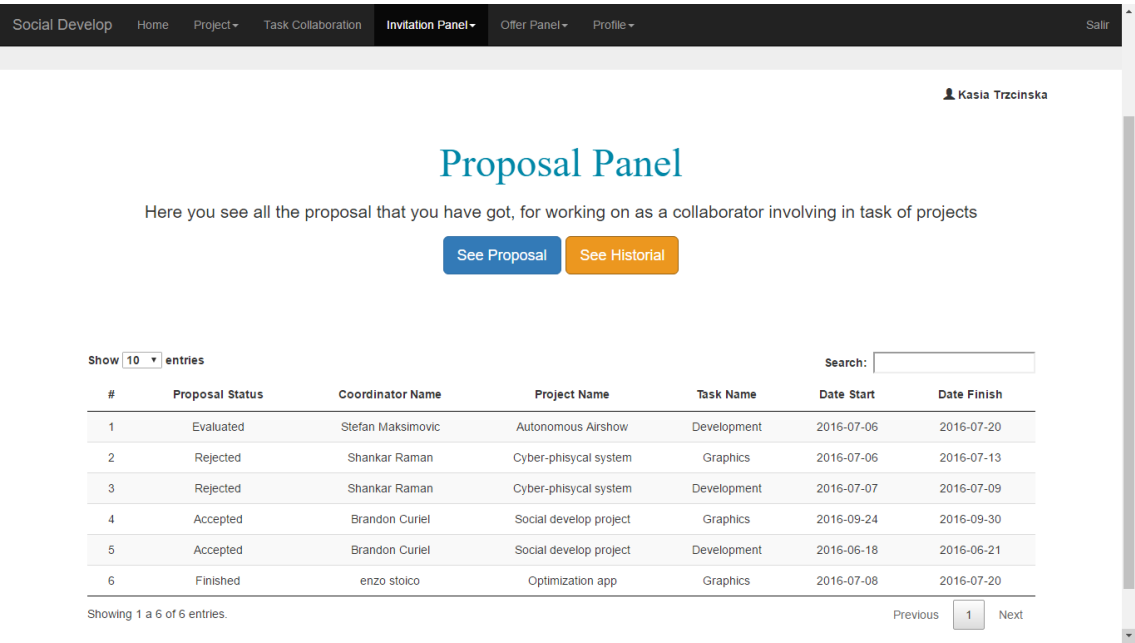


*Figure 30. Record of proposal panel*

## 3.6.  Offer Panel

Social Develop give you the opportunity for get job as a developer, showing you a list of open task according with your skills for work in any project, apply now this is your opportunity.



*Figure 30. Request application panel*



*Figure 31. Record of Request application panel*

Collaborator can see the coordinator's decision status as "accepted" or "rejected".

### 3.7. Task Collaboration Panel

Task Collaboration panel allow you, see all the tasks in which you are working, here you can finish the task collaboration, only uploading a PDF file.



*Figure 32. Task collaboration panel*

You can see the task collaboration record as optional feature we added three tree more status, "working", "waiting for evaluation" and "evaluated" for this section.

*Figure 33. Record of Task collaboration panel*

It time to talk in the definition of the project about the button collaborators of a task.



*Figure 34. Collaborators of a task-1*

We can see the collaborators of a development task, and two of them have been evaluated and "Jose Sevilla" finished its task, because he is waiting for be evaluated.
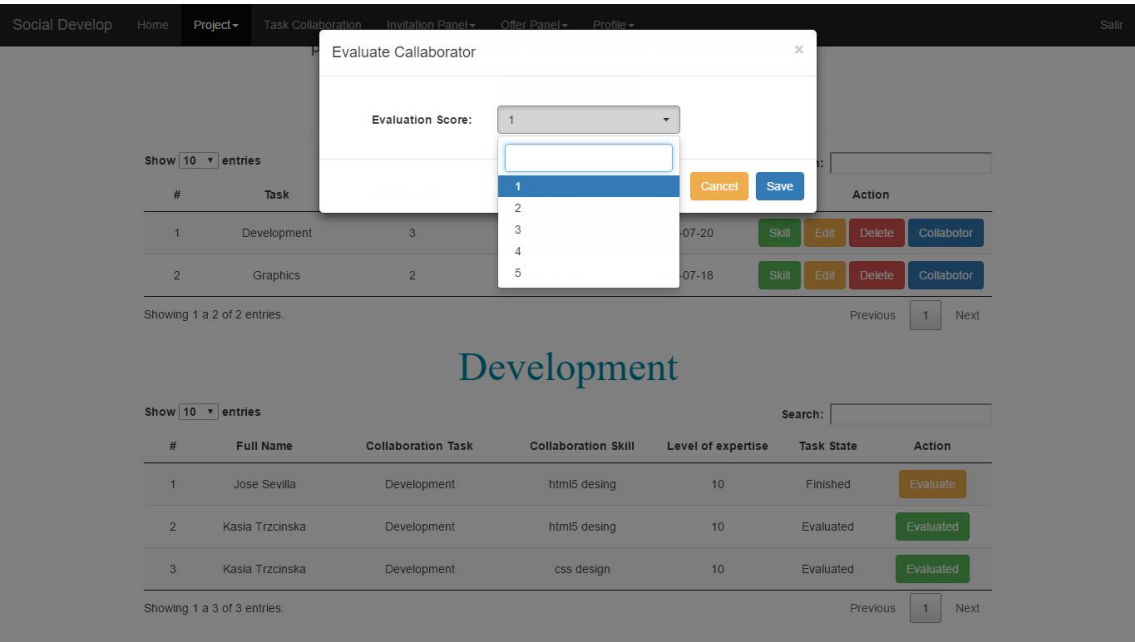


*Figure 35. Evaluation of a collaborator*

The coordinator of the project can evaluate from 1 to 5 to collaborator
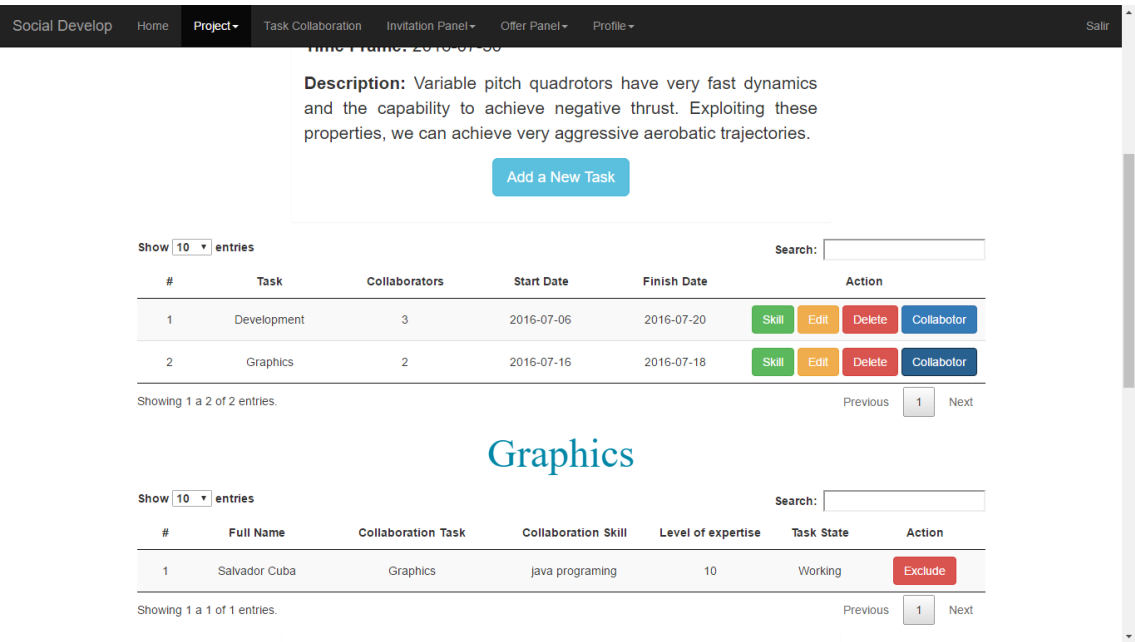


*Figure 36. Collaborators of a task-2*

We can see the collaborator of a graphics task, and he is working so it means that he can be excluded.

## 3.8. Messages

Coordinator and developer can write and see messages public or private inside of the project.
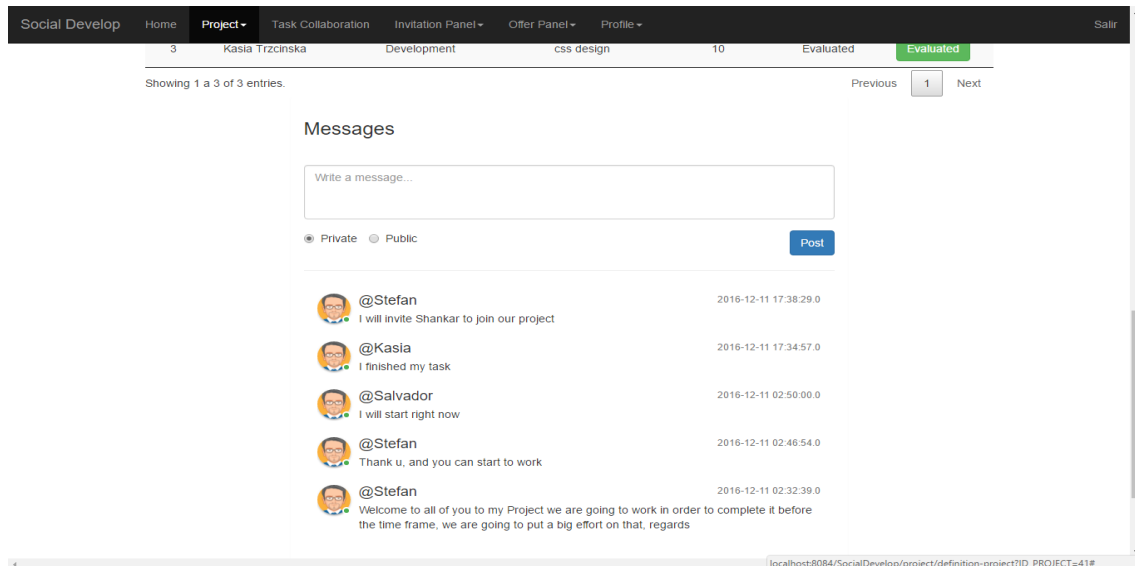


*Figure 37. Messages inside a project*

Developers can see their profile, that include to see all their data download, change their curriculum and see task in which they have been evaluated and in which they are working on.
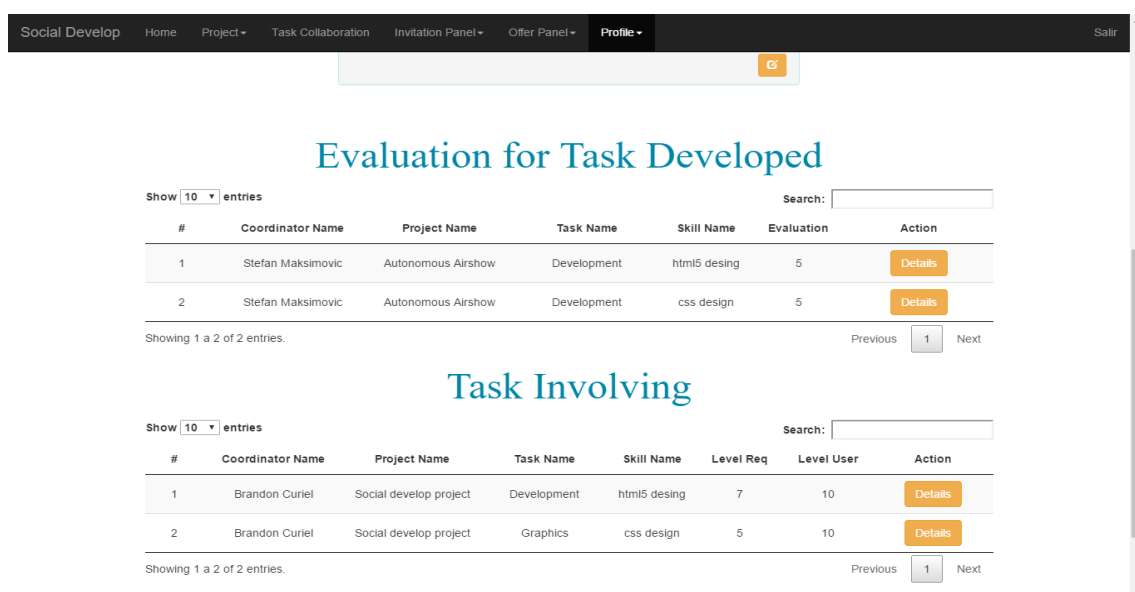


*Figure 38. Profile on computer*
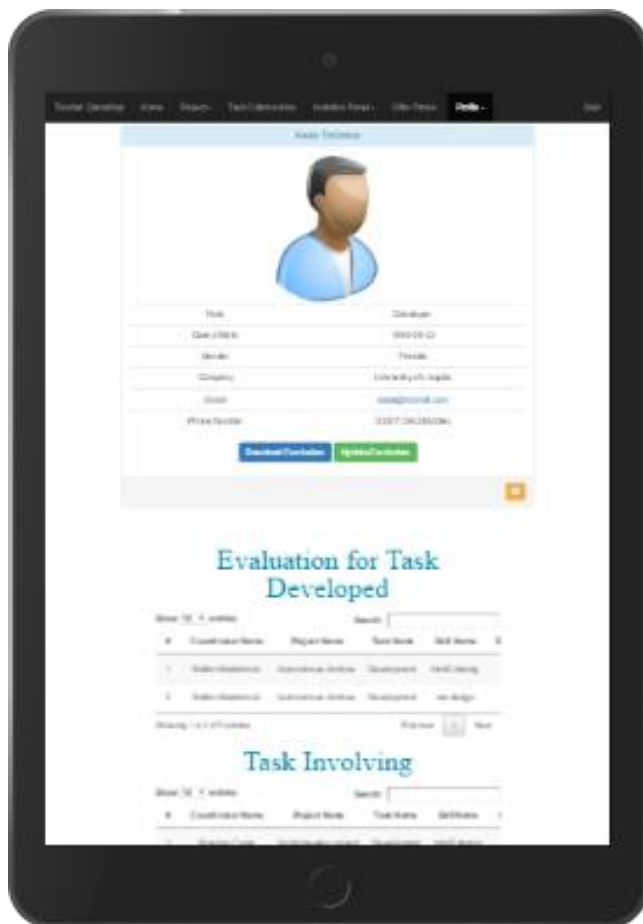
*Figure 39. Profile on tablet*
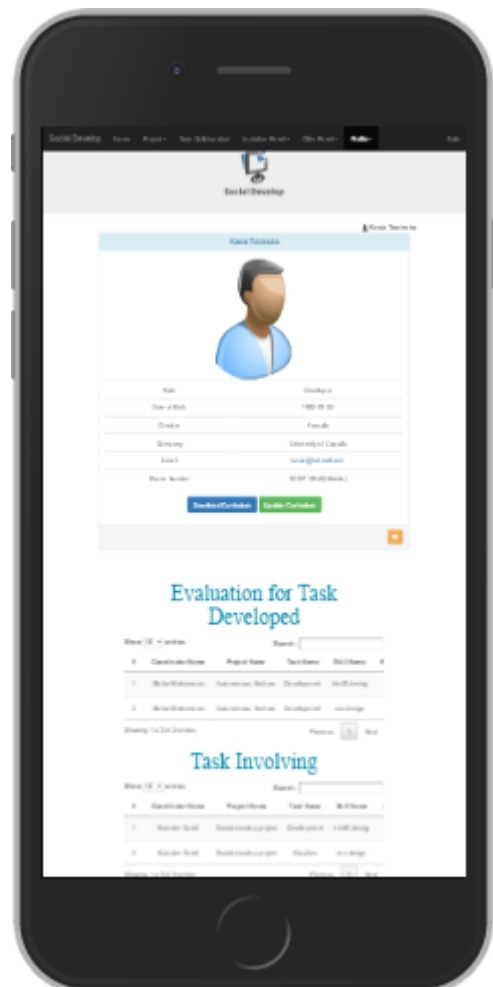
*Figure 40. Profile on smartphone*

## 3.9. Search on SocialDevelop
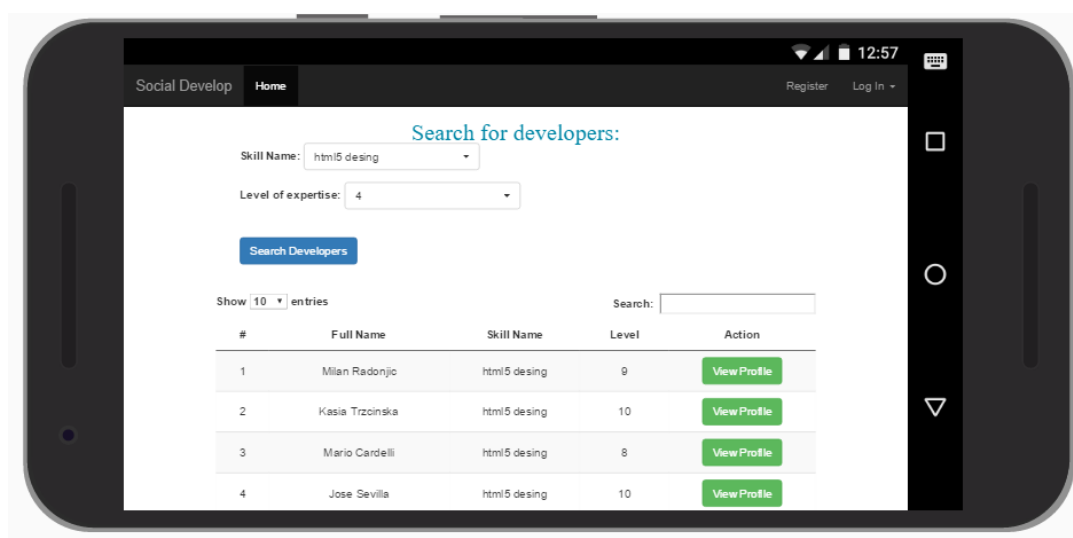


*Figure 41.  Search by developer in the home view*

Search for developers: the user indicates "HTML 5" a "4" as minimum level of expertise and the system returned the list of developer profiles having the skill, with a level of expertise greater than or equal to the given.
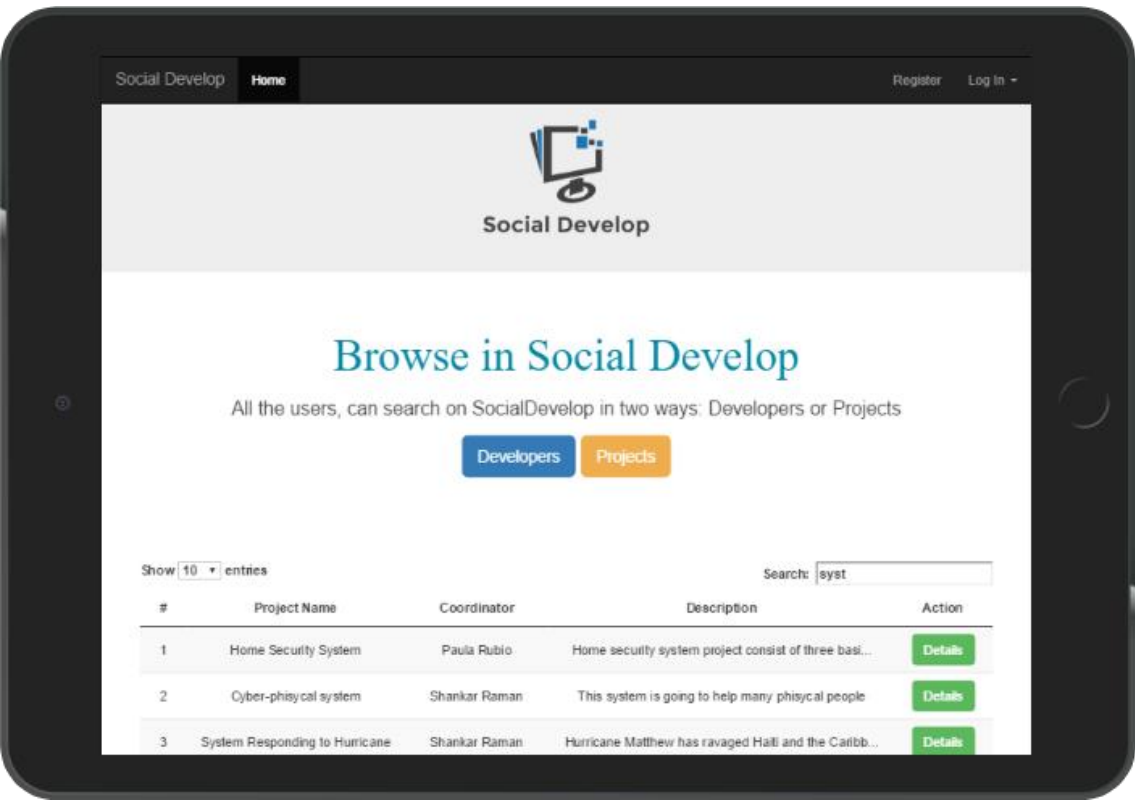


*Figure 42.  Search by projects in the home view*

Search for projects: the user entered "Syst" as keywords, and the system returned a list of projects having that word in the name or description.

In the end we show that we are inside of the same project of the figure 37, "Autonomous Airshow" as an anonymous user and only we can see the public messages and we cannot write any message because only user that are part of the project can write a message.

*Figure 43.  Messages and project definition for an anonymous user*

We added email notification as well, we are going to show in the oral examination.

## 4.  Browse supported

We are using Bootstrap version 3 and Bootstrap is built to work best in the latest desktop and mobile browsers, meaning older browsers might display differently styled, though fully functional, renderings of certain components[2].



*Figure 44.  Supported browsers*

---

[2] http://getbootstrap.com/getting-started/#support

# 5. Libraries used in the project

## 5.1. Back-End

| Libraries | Version | Web Site | Description |
|---|---|---|---|
| **Spring Framework** | 3.1.1.RELEASE | https://projects.spring.io/spring-framework/ | Spring was selected as the Social Develop Framework. The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform. |
| **Spring-Security** | 3.1.1 | https://projects.spring.io/spring-security/ | Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. |
| **MySQL-connector-java** | 5.1.26-bin | http://dev.mysql.com/downloads/connector/j/3.0.html | Library for connect to the data base MySQL. |
| **MyBatis-Spring** | 3.2.2 | http://www.mybatis.org/mybatis-3/es/ | MyBatis-Spring integrates MyBatis seamlessly with Spring. This library allows MyBatis to participate in Spring transactions, takes care of building MyBatis mappers and SqlSessions and inject them into other beans, translates MyBatis exceptions into Spring DataAccessExceptions, and finally, it lets you build your application code free of dependencies on MyBatis, Spring or MyBatis-Spring. |
| **BoneCP** | 0.8 | http://www.jolbox.com/ | BoneCP is a fast, free, open-source, Java database connection pool (JDBC Pool) library. If you are familiar |

| | | | with C3P0 and DBCP then you already know what this means. |
|---|---|---|---|
| **JSTL** | 1.2 | http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/ http://www.oracle.com/technetwork/java/index-jsp-135995.html | JavaServer Pages Standard Tag Library (JSTL) encapsulates as simple tags the core functionality common to many Web applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. |
| **Apache-tiles** | 2.2.2 | https://tiles.apache.org/ | Apache Tiles allows authors to define page fragments which can be assembled into a complete page at runtime. These fragments, or tiles, can be used as simple includes in order to reduce the duplication of common page elements or embedded within other tiles to develop a series of reusable templates. |
| **Jackson** | 2.0 | http://wiki.fasterxml.com/JacksonHome | Inspired by the quality and variety of XML tooling available for the Java platform (StAX, JAXB, etc.), the Jackson is a multi-purpose Java library for processing JSON data format. Jackson aims to be the best possible combination of fast, correct, lightweight, and ergonomic for developers. |
| **Logger** | 1.2.14 | https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html | A Logger object is used to log messages for a specific system or application component |
| **Javax-mail** | 1.4 | http://www.oracle.com/technetwork/java/javamail/index.html | The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications. The JavaMail API is available as an optional package for use with the Java SE platform and is also included in the Java EE platform. |

| Free-Marker | 2.3.20 | http://freemarker.org/ | Apache FreeMarker is a *template engine*: a Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data. |
|---|---|---|---|

*Table 1.  Backend Table*

## 5.2.   Front End

| Libraries | Version | Web Site | Description |
|---|---|---|---|
| jQuery | 2.1.4 | https://jquery.com/ | JQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. |
| AngularJS | 1.4.7 | https://angularjs.org/ | AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop. |
| Bootstrap | 1.10.0 | http://getbootstrap.com/ | Bootstrap makes front-end web development faster and easier. It's made for folks of all skill levels, devices of all shapes, and projects of all sizes. |
| JQuery-confirm | 2.5.1 | https://craftpip.github.io/jquery-confirm/ | A jQuery plugin that provides great set of features like, Auto-close, Ajax-loading, Themes, Animations and more. This plugin is actively developed, I would love you have your suggestions. |

| jQuery-validate | 1.15.0 | https://jqueryvalidation.org/ | JQuery plugin makes simple clientside form validation easy, whilst still offering plenty of customization options. It makes a good choice if you're building something new from scratch, but also when you're trying to integrate something into an existing application with lots of existing markup. |
|---|---|---|---|
| DataTables | 1.10.12 | https://datatables.net/ | DataTables is a plug-in for the jQuery JavaScript library. It is a highly flexible tool, based upon the foundations of progressive enhancement, and will add advanced interaction controls to any HTML table. |
| JQuery.Mask | 1.13.4 | https://plugins.jquery.com/mask/ | A jQuery Plugin to make masks on form fields and html elements. |

*Table 2.  Frontend Table*