

EPITA GRADUATE SCHOOL OF COMPUTER SCIENCE



# IDENTITY ACCESS MANAGEMENT WEB APPLICATION

December 2016

Submitted by

Vishwas Nagaraj

Prof. Thomas Broussard

# TABLE OF CONTENTS

1. Project Overview
2. Scope and Purpose
3. Utilities and Configuration
4. Process Overview
5. Source Code
6. Screenshots

## 1. PROJECT OVERVIEW

This project revolves around building a web application.

This application comprises of the index page, the welcome page after the successful login by a user and the pages to search a user and modify a user. A link to registering a user is included in the index page.

To give a clear view of the application -

First page is the index page where the user is asked to login or register the user.

The second page is displayed on successful login, which is the welcome page.

The third page is where a user can search for an existing entry by using the username

The fourth page is where a user can modify the existing entries. The username, e-mail address and other details attributed to an entry can be modified in this page.

The application allows or denies a users' authentication based on whether the entry exists in the database or not.

A manual check the existence and details of a record by accessing the IAM system database.

## 2. SCOPE AND PURPOSE

The Identity Access Management System is an administrative application that deals with manipulation and management of individual identities, their authentication within or across system. The document helps to understand the basic routine working of the Identity access management and its functionalities.

On the application, the major features include creating an identity, modifying an identity, searching an identity and deleting an identity (DAO inherited from lamCore)

The server aspect of the web application is exhibited here. The design is limited despite the use of Bootstrap framework and there is no JavaScript applied.

### *Limitations-*

A more comprehensive validation of user input could be implemented both on the client side as well as the server side since the project, at this stage is neither very flexible nor all that scalable when it comes to adding or changing attributes in the activity model.

### 3. UTILITIES AND CONFIGURATION

The project uses the following frameworks –

- Maven
- Spring
- Derby
- Hibernate

An Apache Tomcat server is used for the implementation of this project

- *Maven Framework –*

It is used for a better clarity on the overview and knowing what is needed. This is responsible for clarification on dependencies.

- *Spring Framework –*

It is used to control the ApplicationContext and assists in managing the creation of instances of different classes as needed. This is called dependency injection and by using @autowired, Spring framework will inject dependencies as per users' needs.

- *Derby Framework –*

Derby is an easy to use framework which creates a local database by using embedded mode. The configuration is easily maintained in the ApplicationContext.xml

- *Hibernate Framework –*

Hibernate uses HQL which is much simpler compared to the complex SQL statements. The codes, thus, are simpler and compact and the configuration for the framework is maintained in the ApplicationContext.xml

- *Apache Tomcat –*

This is an open source server available on the internet and is easy to set up with the web application. The main features of the application will be realized using servlets, classes and JSP files.

## 4. PROCESS OVERVIEW

The application works in the following way –

When a user enters the username and password, if the entry matches the one in the database, the user is taken to the welcome page of the application. If the user login fails, the user is redirected to the login page with a login option giving an error message saying ‘Login failed’. After successful login of a user, the user will be able to search for an entry which is displayed in the form of a list and once the entries are displayed, the entries will be available for editing by the logged in user. At any point during the execution of any of the actions, a message is displayed informing the user whether the action is successful or not.

Sessions in Java handles the process of user authorization (login/logout) which is done by checking if the users’ session is stored in the current session. Servlets controlling the request must perform a step-by-step algorithm to create users. This algorithm checks for the correct input and creates a duplicate identity in the database.

The same process takes place during the updating and deletion of an identity. The servlet checks for the correct input from the user and either updates or deletes the identity as per the request.

The search function is carried out by parsing the entered search criteria by the user and calls the implemented search method in lamCore. The search method looks for identities in the database that matches the criteria entered.

For modifydelete, validation is made for all user inputs and an update in the database will be requested by the servlet. If no input is provided, an empty record will be made in the database.

## ApplicationContext

The applicationcontext.xml consists of the overall pre-configuration of the database properties and maps the different classes needed for the dependency injection engine (used for auto wiring).

By mapping the different classes, it is not needed to create instances, but only indicates where to put instances to make your application run.

The application context consists of the following classes –

- UserModel
- SessionFunctions
- IdentityHibernateDAO
- AuthenticationServiceDAO
- BeanBasedSessionFactory
- DataSourceBean
- HibernateProperties

The UserModel, SessionFunctions, AuthenticationServiceDAO, IdentityHibernateDAO are all places as beans in the applicationcontext.xml file because of the mapping needed to use the



auto wiring mechanism.

The BeanBasedSessionFactory is mapped as well, but is used as a session factory for the connection of the database. Furthermore, both the identity and user model is mapped using annotation configuration, which means that the classes are defined as entities, and each field or attribute is mapped, to tell hibernate, how to create the entities in the database, with its corresponding columns, DataSourceBean has the properties of the JDBC driver to the database. Properties like DriverClassName, URL, and Username are configured here.

The HibernateProperties are the properties of the hibernate configuration to access the database through the framework. In this project, hibernate is set up with the DerbyDialect, which gives the ability to convert HQL into something the database will understand. Furthermore, the configuration is set to display SQL queries in the console.

Additionally, the property hibernate.hbm2ddl.auto is used to endure object representation will synchronize with the database. When starting the application, it is set as create, to start the database from scratch, with no users or identities in the database. After the first execution, the property should be changed to 'update' to keep the created objects in the database.

## 5. SOURCE CODE

### Create module

```

<%@page import="fr.tbr.iamcore.datamodel.Identity"%>
<%@page import="java.util.Collection"%>
Collection<Identity> idList = (Collection<Identity>)session.getAttribute("identity");

session.removeAttribute("identity");
Identity identity = null;
if(idList!=null){
for (Identity identity1 : idList){
    identity =identity1;
}
}
%>
<head>

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>jQuery UI Datepicker - Default functionality</title>
<link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-
ui.css">
<link rel="stylesheet" href="/resources/demos/style.css">
<script src="https://code.jquery.com/jquery-1.12.4.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
<script>
$( function() {
    $( "#datepicker" ).datepicker();
} );
</script>
<link xmlns="http://www.w3.org/1999/xhtml" rel="stylesheet"
href="css/bootstrap.min.css" />
</head>

<body>
<h1>Identity Creation</h1>
<div class="container">
    <div xmlns="http://www.w3.org/1999/xhtml" class="bs-example">
        <form role="form" method="post" action="IdAction">
            <div class="form-group">

                <label for="exampleInputEmail1">Display Name</label>
                <input name="displayName" type="text" class="form-
control" id="exampleInputEmail1"

                    placeholder="Enter Login" <%
                    if(identity!=null){

                        %>
                        value= '<%=identity.getDisplayName()%>'
                        <%
                    }
                }
            </div>
        </form>
    </div>
</div>

```

```

        %> />
    </div>
    <div class="form-group">
        <label for="exampleInputEmail1">Email</label>
        <input name="email" type="text" class="form-control"
id="exampleInputEmail1"
            placeholder="Enter Login"
            <%
            if(identity!=null){
                %>
                value='<%=identity.getEmail() %>'
                <%
            }
            %> />
    </div>
    <div class="form-group">
        <label for="exampleInputEmail1">UID</label>
        <input name="uid" type="text" class="form-control"
id="exampleInputEmail1"
            placeholder="Enter Login"
            <%
            if(identity!=null){
                %>
                value='<%=identity.getId() %>'
                <%
            }
            %>/>
    </div>
    <div class="form-group">
        <label for="exampleInputEmail1">BirthDate</label>
        <input name="birthDate" type="date" class="form-
control" id="datepicker"
            placeholder="Enter Login"
            <%
            if(identity!=null){
                %>
                value='<%=identity.getBirthDate() %>'
                <%
            }
            %>/>
    </div>

    <button type="submit" class="btn btn-
default">Save</button>
    </form>
</div>
</div>
</body>
</html>

```

## Login module

```
1 package fr.tbr.iam.web.servlets;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class Login
7  */
8
9 @WebServlet(name="Login", urlPatterns="/Login")
10 public class Login extends GenericSpringServlet {
11     private static final long serialVersionUID = 1L;
12
13     @Inject
14     UserDAOInterface dao;
15
16     IAMLogger logger = IAMLogManager.getIAMLogger(Login.class);
17
18     /**
19      * Default constructor.
20      */
21     public Login() {}
22
23     /**
24      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
25      */
26     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
27     }
28
29     /**
30      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
31      */
32     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
33         String login = request.getParameter("login");
34         String password = request.getParameter("password");
35         try {
36             dao.save(new User("test", "test"));
37         } catch (DAOSaveException e1) {
38             // TODO Auto-generated catch block
39             e1.printStackTrace();
40         }
41         logger.debug(login);
42         logger.debug(password);
43         try {
44             if (dao.authenticate(login, password)){
45                 response.sendRedirect("welcome.jsp");
46             }else{
47                 response.sendRedirect("reconnect.jsp");
48             }
49         } catch (DAOSearchException | DAOSaveException e) {
50             // TODO Auto-generated catch block
51             e.printStackTrace();
52         }
53     }
54 }
55 }
```

## Search Module

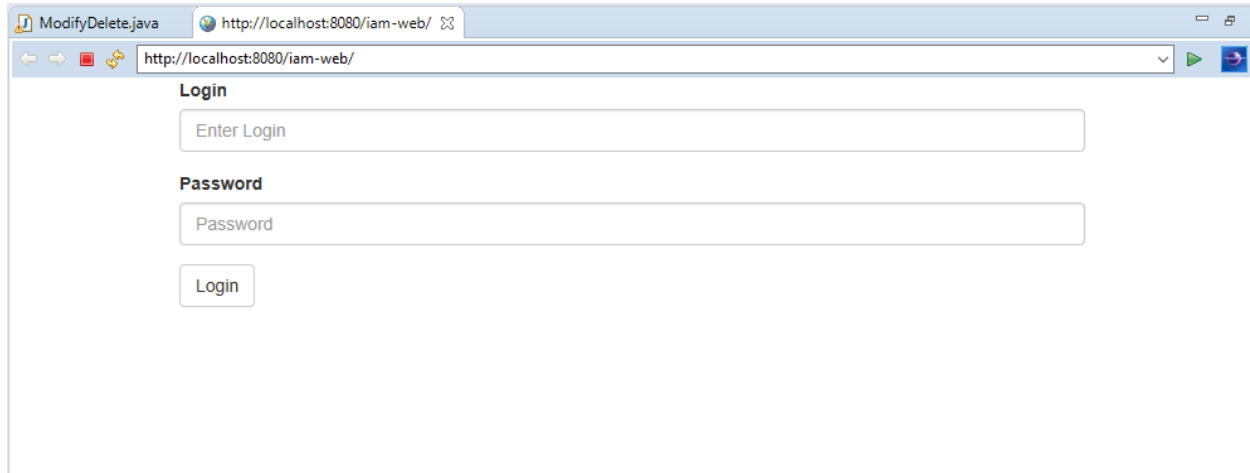
```
1 package fr.tbr.iam.web.servlets;
2
3 import java.io.IOException;
18
19 /**
20  * Servlet implementation class Login
21  */
22
23 @WebServlet(name="Search", urlPatterns={"/Search"})
24 public class Search extends GenericSpringServlet {
25     private static final long serialVersionUID = 1L;
26
27     IAMLogger logger = IAMLogManager.getIAMLogger(Search.class);
28
29     @Inject
30     IdentityDAOInterface dao;
31
32
33     /**
34      * Default constructor.
35      */
36     public Search() {
37     }
38
39     /**
40      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
41      */
42     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
43         String displayName = request.getParameter("displayName");
44         String email = request.getParameter("email");
45         logger.info("received this query : = displayName" + displayName + " email = " + email);
46         try {
47             Collection<Identity> idList = dao.search(new Identity(displayName, email, null, null));
48             request.getSession().setAttribute("idList", idList);
49         } catch (DAOSearchException e) {
50             // TODO Auto-generated catch block
51             e.printStackTrace();
52         }
53
54         response.sendRedirect("search.jsp");
55
56     }
57
58
59     /**
60      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
61      */
62     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
63         doGet(request, response);
64     }
65
66 }
67
68 }
```

## ModifyDelete Module

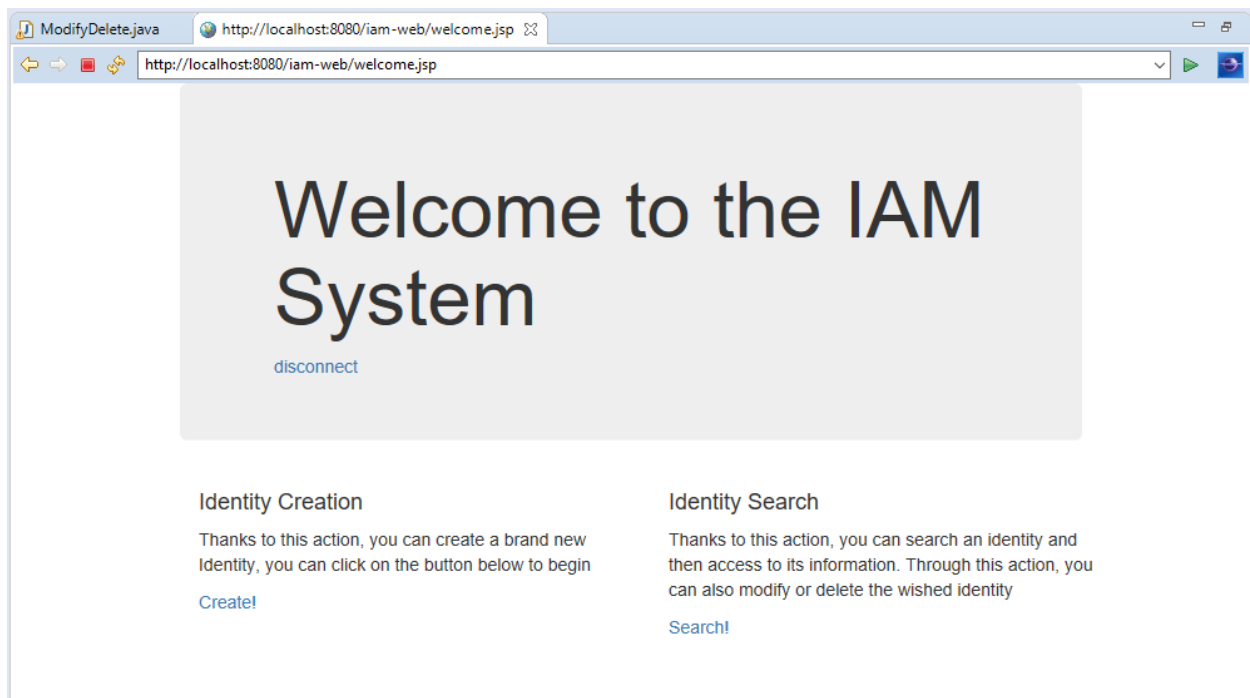
```
1 package fr.tbr.iam.web.servlets;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class ModifyDelete
7  */
8 @WebServlet(name="ModifyDelete", urlPatterns={"/ModifyDelete"})
9 public class ModifyDelete extends GenericSpringServlet {
10     private static final long serialVersionUID = 1L;
11
12     IAMLogger logger = IAMLogManager.getIAMLogger(ModifyDelete.class);
13
14     @Inject
15     IdentityDAOInterface dao;
16
17     /**
18      * @see HttpServlet#HttpServlet()
19      */
20     public ModifyDelete() {
21         super();
22         // TODO Auto-generated constructor stub
23     }
24
25     /**
26      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
27      */
28     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
29         // TODO Auto-generated method stub
30         response.getWriter().append("Served at: ").append(request.getContextPath());
31     }
32
33     /**
34      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
35      */
36     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
37         doGet(request, response);
38         String value=request.getParameter("modification");
39         long a=Long.valueOf(request.getParameter("selection"));
40         logger.info(value);
41         if(value.equals("modify")){
42             try {
43                 Collection<Identity> idList = dao.get_id (a);
44                 request.getSession().setAttribute("identity", idList);
45                 request.getSession().setAttribute("id", request.getParameter("selection"));
46                 request.getRequestDispatcher("create.jsp").forward(request, response);
47             } catch (DAOSearchException e) {
48                 e.printStackTrace();
49             }
50         }else if(value.equals("delete")){
51             try {
52                 Collection<Identity> idList = dao.get_id (a);
53                 for(Identity id : idList){
54                     dao.delete(id);
55                     response.sendRedirect("search.jsp");
56                 }
57             } catch (DAOSearchException | DAODeleteException e) {
58                 e.printStackTrace();
59             }
60         }else{
61             response.sendRedirect("search.jsp");
62         }
63     }
64 }
65
66 }
```

## 6. SCREENSHOTS

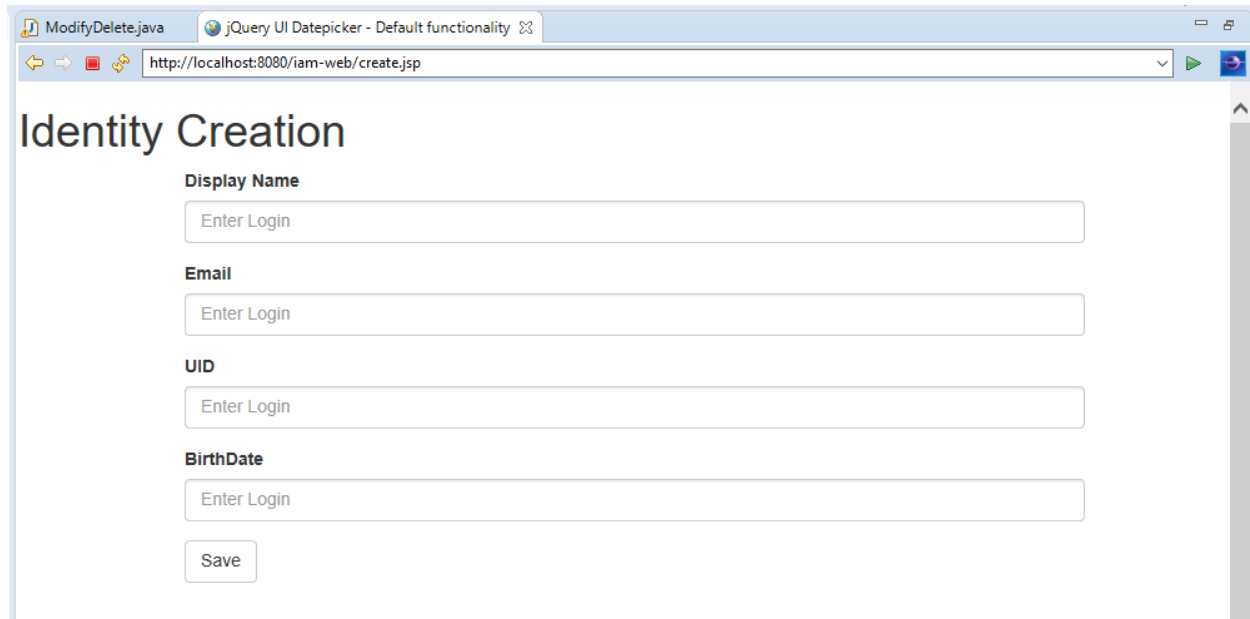
### Index page



### Welcome page on successful login



## Identity creation

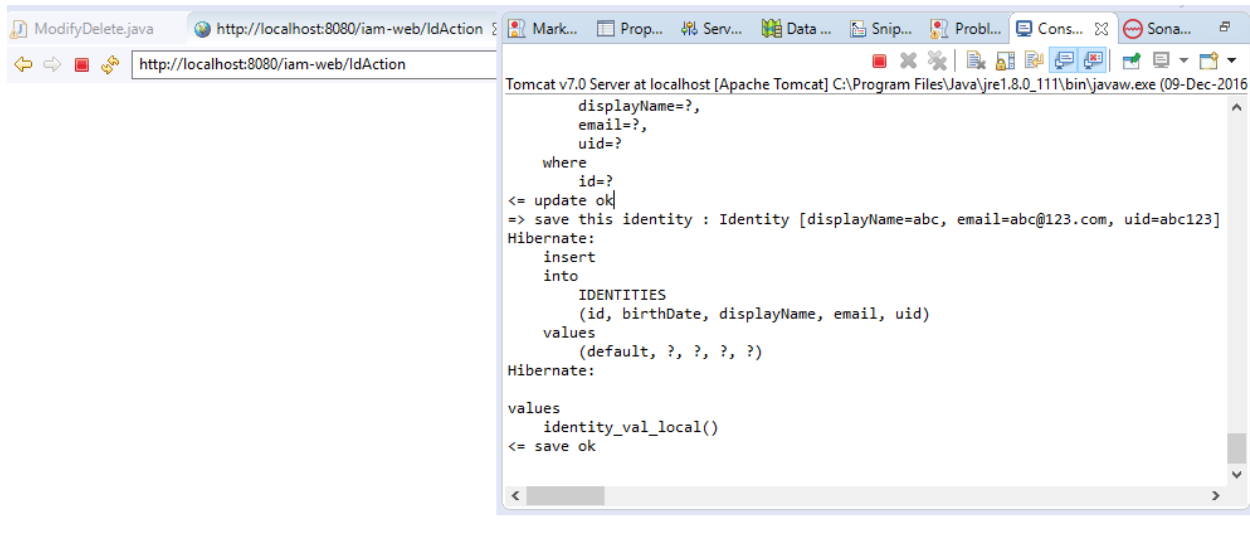


The screenshot shows a web browser window with the URL `http://localhost:8080/iam-web/create.jsp`. The page title is "Identity Creation". It contains four input fields, each with the placeholder text "Enter Login":

- Display Name**
- Email**
- UID**
- BirthDate**

Below the input fields is a "Save" button.

## Successful creation of identity



The screenshot shows the Tomcat v7.0 Server console output for the URL `http://localhost:8080/iam-web/IdAction`. The output indicates a successful save operation:

```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (09-Dec-2016)
    displayName=?,
    email=?,
    uid=?
    where
        id=?
<= update ok
=> save this identity : Identity [displayName=abc, email=abc@123.com, uid=abc123]
Hibernate:
    insert
    into
        IDENTITIES
    (id, birthDate, displayName, email, uid)
    values
        (default, ?, ?, ?, ?)
Hibernate:
values
    identity_val_local()
<= save ok
```



## Search page

The screenshot shows a web browser window with the URL `http://localhost:8080/iam-web/search.jsp`. The page title is "Identity Search". Below the title is a "<< back" link. The "Search Criteria" section contains two input fields: "Display Name" and "Email". A "Search" button is located to the right of the "Email" field. Below the search criteria is the "Search Results" section, which contains a table with the following headers: "Selection", "UID", "Display Name", and "Email". The table currently shows "No result". At the bottom right of the page are three buttons: "Modify", "Delete", and "Cancel".

## Successful search

The screenshot shows a Tomcat v7.0 Server console window. The output text is as follows:

```
values
  (default, ?, ?, ?, ?)
Hibernate:
values
  identity_val_local()
<= save ok
received this query : = displayNameabc email =
Hibernate:
select
  identity0_.id as id1_0_,
  identity0_.birthDate as birthDat2_0_,
  identity0_.displayName as displayN3_0_,
  identity0_.email as email4_0_,
  identity0_.uid as uid5_0_
from
  IDENTITIES identity0_
where
  identity0_.displayName=?
```

## Search results displayed in the form of a List

### Search Results

Selection	UID	Display Name	Email
<input type="radio"/>	abc123	abc	abc@123.com
<input type="radio"/>	abc123	abc	abc@123.com
<input type="radio"/>	abc123	abc	abc@123.com

ModifyDeleteCancel

## Modification page

### Identity Creation

Display Name

Email

UID

BirthDate

## Successful modification

```
=> update this identity : Identity [displayName=abcd, email=abc@123.com, uid=abc123]
Hibernate:
    update
      IDENTITIES
    set
      birthDate=?,
      displayName=?,
      email=?,
      uid=?
    where
      id=?
<= update ok
```

## Search result after modification

### Search Results

Selection	UID	Display Name	Email
<input type="radio"/>	abc123	abcd	abc@123.com

## Deletion of an entry

```
identity0.id=?  
=> delete this identity : Identity [displayName=abcd, email=abc@123.com, uid=abc12  
Hibernate:  
    delete  
    from  
        IDENTITIES  
    where  
        id=?  
<= delete ok
```