IamWeb

Advanced Java

Niels Christian Dawartz

November 27, 2015

Contents

List of Figures

List of Tables

1 Subject Description

This subject is a continuation of the project IamCore, which can be found here: (http://thomas-broussard.fr/work/java/courses/project/index.xhtml). It is a web-application written in the java language (jsp, servlets, jstl), css, html and makes use of a variety of frameworks, Maven, Spring, Derby, Hibernate and Bootstrap.

2 Subject Analysis

2.1 Major features

The webapplication makes it possible to create a user which is able to login/-logut on a website (iamWeb). On the website it is possible to create, search, modify and delete Identities (DAO from IamCore).

2.2 Application Feasibility

The project is using Maven, Spring, Derby, Hibernate and a Tomcat server. Maven assist in having a great overview and a definition of what is needed (dependencies), in order to build and use the project, which is controlled in the xml file Pom.xml (*Maven website* n.d.).

The Spring framework is used to control the applicationcontext, and assist in managing when to create instances of different classes (beans), when needed. This is called dependency injection, and by using the @autowired annotation spring will inject the instances, where it is decided to inject them (*Spring website*).

Derby is used because of the ease of use. By using the embedded mode, a local database is created/connected, which is running inside the application. The configuration is easely maintained in the ApplicaitonContext.xml file (dataSource) (*Derby website* n.d.).

The main reason Hibernate is applied to this project, is because it helps to get rid of too complex SQL statements, and uses HQL instead. This results in more simple, less lines of code and it is again controlled in the Application-Context.xml file (*Why i choose Hibernate for my project?*).

The server is an Apache Tomcat, becuase is the free and easy to set up with the webapplication.

The webapplications main features will be realized through the use of servlets, Java classes and JSP files.

2.3 Data description

In the following an overview of the data description is presented, through the usage of UML. The specific implementation and elaborations of the different methods can be found in the javaDoc, within the source code.

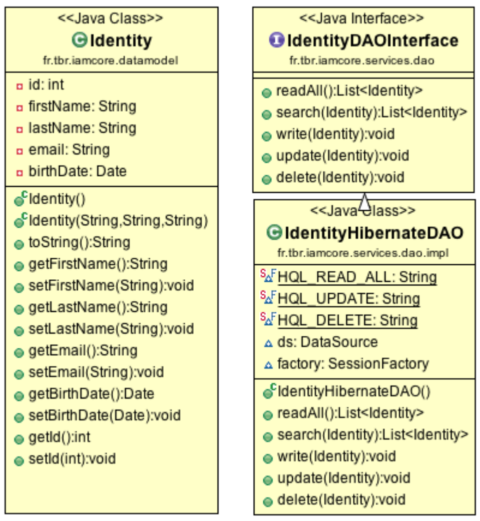
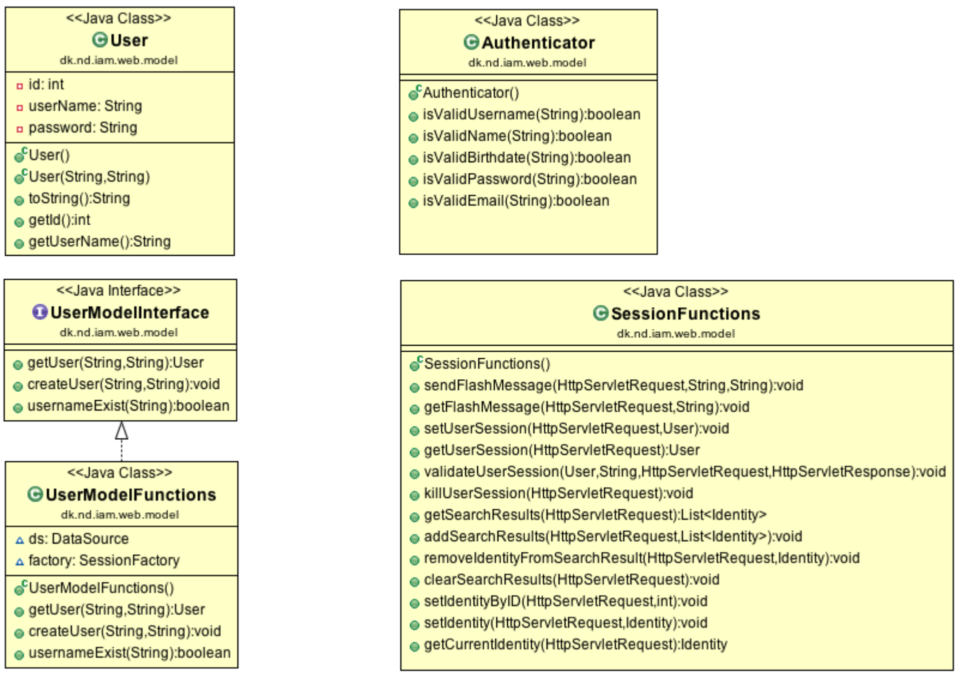


Figure 1: An overview of IamCore DAO

As can be seen in figure 1, the Identity consist of an id (int), firstName (String), lastName (String), email (String), and a birthDate (Date) - implemented with corresponding getter's and setter's. The `IdentityHibernateDAO` class is an implementaion of the `IdentityDAOInterface`, and makes use of the Hibernate Framework, in order to make sessions in the database. The HQL String constants are HQL queries, for the methods `readAll`, `update` and `delete`.



<<Java Class>>
User
dk.nd.iam.web.model
id: int
userName: String
password: String
User()
User(String,String)
toString():String
getId():int
getUserName():String
<<Java Class>>
Authenticator
dk.nd.iam.web.model
Authenticator()
isValidUsername(String):boolean
isValidName(String):boolean
isValidBirthdate(String):boolean
isValidPassword(String):boolean
isValidEmail(String):boolean
<<Java Interface>>
UserModelInterface
dk.nd.iam.web.model
getUser(String,String):User
createUser(String,String):void
usernameExist(String):boolean
<<Java Class>>
UserModelFunctions
dk.nd.iam.web.model
ds: DataSource
factory: SessionFactory
UserModelFunctions()
getUser(String,String):User
createUser(String,String):void
usernameExist(String):boolean
<<Java Class>>
SessionFunctions
dk.nd.iam.web.model
SessionFunctions()
sendFlashMessage(HttpServletRequest,String,String):void
getFlashMessage(HttpServletRequest,String):void
setUserSession(HttpServletRequest,User):void
getUserSession(HttpServletRequest):User
validateUserSession(User,String,HttpServletRequest,HttpServletResponse):void
killUserSession(HttpServletRequest):void
getSearchResults(HttpServletRequest):List<Identity>
addSearchResults(HttpServletRequest,List<Identity>):void
removeIdentityFromSearchResult(HttpServletRequest,Identity):void
clearSearchResults(HttpServletRequest):void
setIdentityByID(HttpServletRequest,int):void
setIdentity(HttpServletRequest,Identity):void
getCurrentIdentity(HttpServletRequest):Identity

Figure 2: An overview of models in iamWeb

In the iamWeb project four models can be seen: `User`, `UserModelFunctions`, `Authenticator`, `SessionFunctions` and an interface `UserModelInterface` (figure 2). The User model consist of an id (int), username (String) and a password (String) - again with corresponding getter's and setter's. The interaction with the `User` model, is made through the `UserModelInterface`, where an implementation is found in the `UserModelFunctions` class. The implementation consist of methods, that make requests to the database, and makes it possible to create and retrieve User objects from the database.
An `Authenticator` model is created to validate userinput on the serverside. The `SessionFunctions` model consist of methods, that manages the sessions in the running webapplication. The methods makes it possible to set, retrieve and delete objects in the sessions.

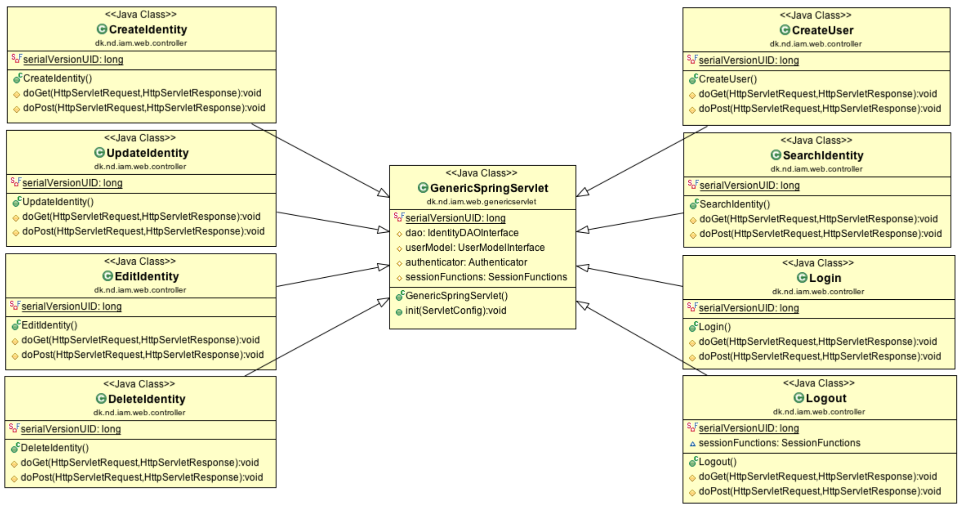

<<Java Class>>
CreateIdentity
dk.nd.iam.web.controller
serialVersionUID: long
CreateIdentity()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
UpdateIdentity
dk.nd.iam.web.controller
serialVersionUID: long
UpdateIdentity()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
EditIdentity
dk.nd.iam.web.controller
serialVersionUID: long
EditIdentity()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
DeleteIdentity
dk.nd.iam.web.controller
serialVersionUID: long
DeleteIdentity()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GenericSpringServlet
dk.nd.iam.web.genericservlet
serialVersionUID: long
dao: IdentityDAOInterface
userModel: UserModelInterface
authenticator: Authenticator
sessionFunctions: SessionFunctions
GenericSpringServlet()
init(ServletConfig):void
<<Java Class>>
CreateUser
dk.nd.iam.web.controller
serialVersionUID: long
CreateUser()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
SearchIdentity
dk.nd.iam.web.controller
serialVersionUID: long
SearchIdentity()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
Login
dk.nd.iam.web.controller
serialVersionUID: long
Login()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
Logout
dk.nd.iam.web.controller
serialVersionUID: long
sessionFunctions: SessionFunctions
Logout()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void

Figure 3: An overview of servlets in IamWeb

As can be seen in figure 3, all the servlets extends from the `GenericSpringServlet`. This servlet makes it possible to use the @Autowiring functionality from the Spring framework, by @overriding the `intit()` method. Additionally, the `GenericSpringServlet` extends the `HttpServlet` class (not shown in 3, and has @autowire fields of the texttIdentityDAOInterface, the `UserModelInterface`, the `Authenticator` and the `SessionFunctions`. Servlet extending from the `GenericSpringServlet`, will therby have access to the 4 different @autowired fields, as well as the doPost and doGet methods.

2.4 Algorithms study

The login/logut mechanish is managed by using sessions, where it must check if a user session is stored in the current session.
To achieve the functionality of creating users and identities, servlet's controlling the request, must peform a stepwise algorithm, that check for correct userinput, dublicates, and create the Identity in the database.
Almost the same approach must be used in the functionality of updating and deleting an identity. Again a servlet must check for correct userinput from the client, and update or delete the identity.
The search functionality is achieved by, parsing the search criteria from the client to the call of the implemented search method from the iamCore project. The search method look for identities, that matches the search criteria (firstname, lastname or email).

2.5 Scope of the application

The project is mainly focusing on the aspect of the server side. Even though the project use the Bootstrap framework to achieve a quick and decent frontend design, the design is very limited and no javascript is applied. Furthermore, it is not very flexible/scalable regarding the changing/adding of attributes in the model of Identity. Additionally, a more comprehensive validation of userinput could be implemented, both on the the client-side as well as the server-side.

3 Conception

3.1 Chosen algorithm

The following is a description of the algorithms applied in the different servlets in IamWeb. To clarify, all the input given by the acutal user (client-side), is sent to the requests through the use of html forms.

As mentioned in section 2.4 the login mechanish is managed by using sessions. Here the `Login` servlet call the the SessionFunctions model, to check if a user session is set in the current session. If it's not set, the response will be redirected to the login (index) page, with a message asking the user to login. When a user succeed in logging in (valid userinput, and exist in the database), the corresponding `User` object is set in the session. A request to the `Logut` servlet will check if a user is set in the current session, and remove it if present. Otherwise the response will be redirected to the index page.

When a request is made to create a user, the servlet and identities, servlet `CreateUser` peform a validatoin check of the differnt input sent in the request. The validation check if the username is valid or already exist in the databe and if the password is valid. If the requirements are met, a new user will be created and the response is redirected to the login page, with a succesfull "user created" message displayed. Otherwise the response will update the current page with an error message displayed to the user.

When a user submits to create an identity a request to the `CreateIdentity` servlet is made, and it will peform a very simular stepwise algorithm, as the `CreateUser` servlet does. The sent parameters in the request is validated by the servlet, and if the requirements are met, a new `Identity` is created in the database.

When a user sends a request to the `SearchIdentity` servlet, it will create a new `Identity`, with the given parameters, and search for Identities that matches any of the search criteria, wheter it is firstname, lastname or email. The result is returned in a list, which is stored in the session.

For `UpdateIdentity` validation is made for all userinput and an update in the database will be requested by the servlet. If no input is given, an update will be made as well, but with no side-effects, since the Identity won't be changed.

Regarding `DeleteIdentity`, the servlet get's an ID from a hidden form field in the jsp file `search-identity`, which is associated with the corresponding Idenitity. The a new Identity is created with the id requested, and it's deleted from the database (using the id).

3.2 Data structures

The datastructures used in this project is primarily Java's build-in List, which is used to store Identity objects. An example of it's usage is in the `IdentityHibernateDAO` implemenation of IamCore, where a call to the `search` method will store all found Identities in a List, which is returned from the method. Through the servlet `SearchIdentity` the list is set in the current session, as an attribute. The result will then be desplayed on the corresponding view (jsp file), where jstl will access and print the result.

3.3 Global application flow

The overall architecture of the webapplication is implemented using the model-view-controller (MVC) design pattern. The views display information from the model and shapes what the user sees. The model is the collection of database requests, methods and classes, which defines the program and its functionality. Finally the controller requests and dispatches the content of the model through Hypertext transfer protocol (HTTP). The controller handles the user-requests, defines the corresponding requests to the model, which subsequently defines the view (*MVC design pattern* n.d.).

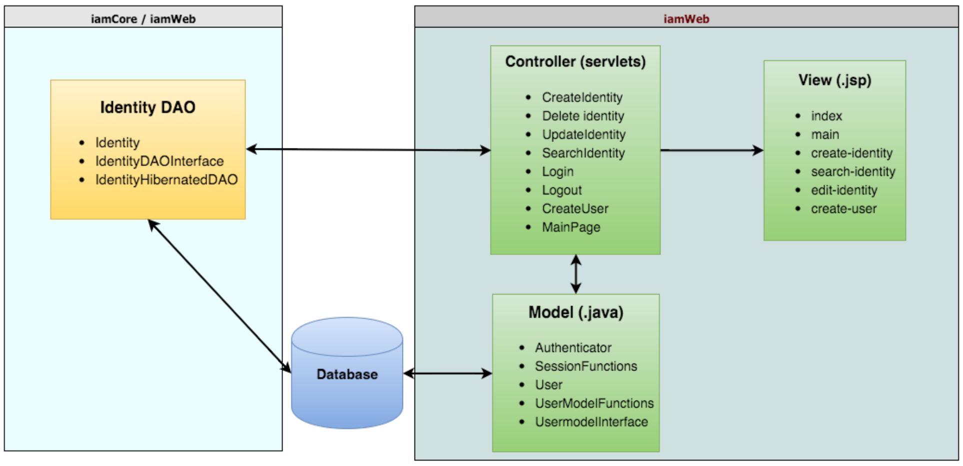


Figure 4: An overview of the flow of the global webapplication

As can be seen in figure 4 the different Models in the iamWeb project interact with the Database and the controllers. The `UserModelFunctions` invoke calls to the datebase, and the `SessionFunctions` is managing methods/functions of sessions. The model `Authenticator` is used as an authentication mechanisms (serverside validations of user input).

The application makes use of the `DAO` (hibernate implementation) in the iamCore project.The Hibernate implementation is chosen, as mentioned earlier, which is used for creating/searching/editing/deleting `Identites` in the database.

The servlets work as controllers, and handles the user-requests and calling the correct models, whether it's in the `IdentityHibernateDAO` or in the Model package in the iamWeb project, and chose and send the correct view as the response. The views is displayed using jsp files, and by using jstl it is possible to avoid too much java code within the files.

4 GUI Description

In this the GUI of the webaplication is presented by six screenshots. On each of the six screenshots there is indicated different Uid's (in red), which is descriped in a following table below the image.

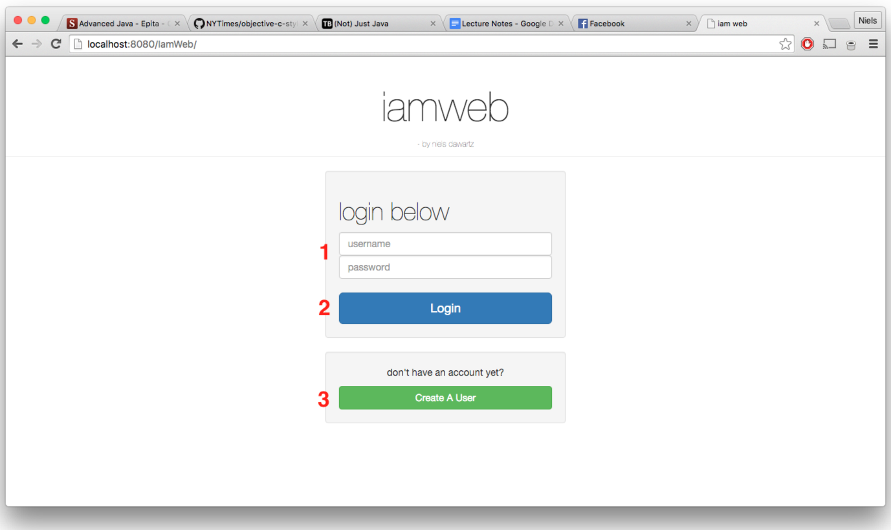
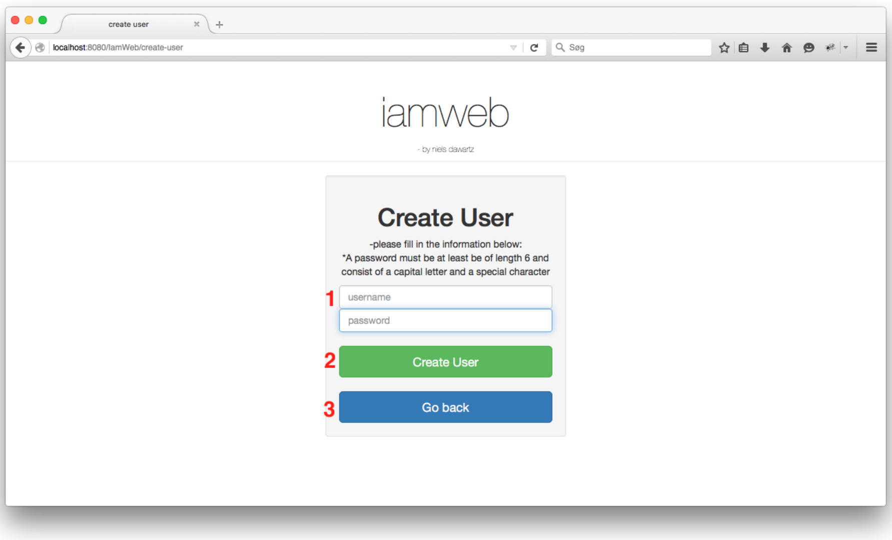


Figure 5: The index page (login)

Uid	Description
1	Html form with fields for username and password
2	The submit button of the form - request the `Login` servlet using POST method
3	Html form to request the `CreateUser` servlet using GET method

Table 1: Description of login page



Create User

-please fill in the information below:
*A password must be at least be of length 6 and consist of a capital letter and a special character

Figure 6: The create user page

Uid	Description
1	Html form with fields for creating a `User` with username and password. A password must consist of password must be at least be of length 6 and consist of a capital letter and a special character
2	The submit button of the form - requesting the `CreateUser` servlet using POST method
3	Html form to request the `Login` servlet using GET method

Table 2: Description of create-user page

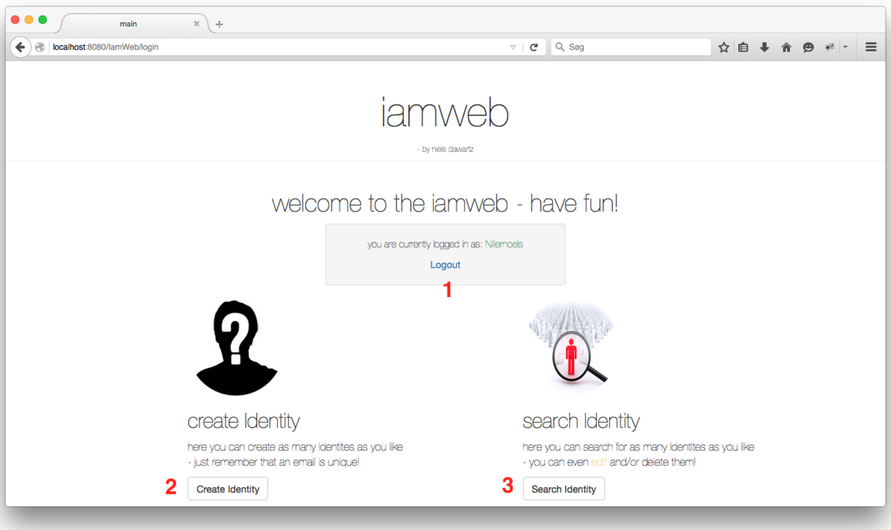

main
localhost:8080/IamWeb/login
Søg
iamweb
- by niels dawartz
welcome to the iamweb - have fun!
you are currently logged in as: Nilemoels
Logout
1
create Identity
here you can create as many Identites as you like
- just remember that an email is unique!
2
Create Identity
search Identity
here you can search for as many Identites as you like
- you can even edit and/or delete them!
3
Search Identity

Figure 7: The main page

Uid	**Description**
1	html anchor link to request the `Logout` servlet
2	Html form to request the `CreateIdentity` servlet using GET method
3	Html form to request the `SearchIdentity` servlet using GET method

Table 3: Description of main page

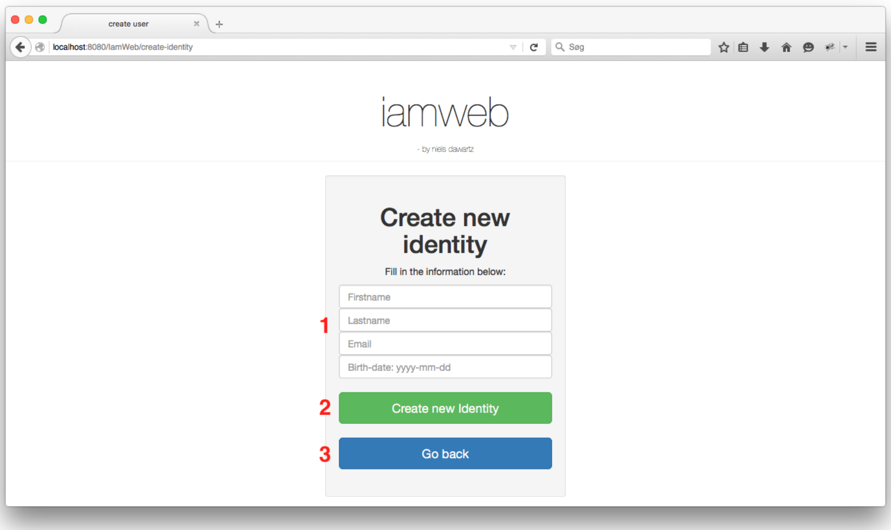
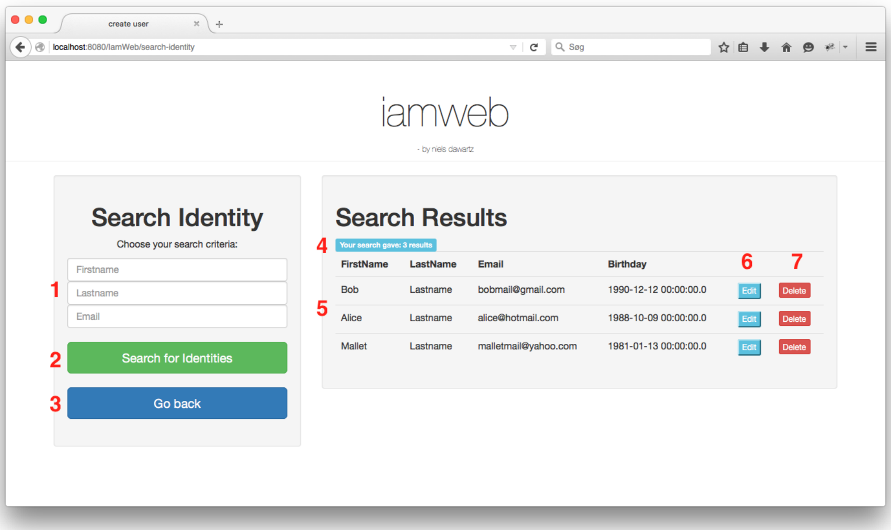


Figure 8: The create-identity page

Uid	Description
1	Html form with fields for creating an `Identity` firstname, lastname, email and birthdate
2	The submit button of the form - requesting the `CreateIdentity` servlet using POST method
3	Html form to request the `MainPage` servlet using GET method

Table 4: Description of the create-identity page



Search Results

Your search gave: 3 results

FirstName	LastName	Email	Birthday		
Bob	Lastname	bobmail@gmail.com	1990-12-12 00:00:00.0	Edit	Delete
Alice	Lastname	alice@hotmail.com	1988-10-09 00:00:00.0	Edit	Delete
Mallet	Lastname	malletmail@yahoo.com	1981-01-13 00:00:00.0	Edit	Delete

Figure 9: The create-identity page

Uid	Description
1	Html form with fields for searching fo an `Identity` firstname, lastname and email
2	The submit button of the form - requesting the `SearchIdentity` servlet using POST method
3	Html form to request the `MainPage` servlet using GET method
4	A paragraph that makes use of jstl to retrieve the a message to display for the user.
5	A table to display the search results for the user, with the 4 fields FirstName, LastName, Email and Birthday
6	Html form to request the `EditIdentity` servlet using POST method. To identify the selected `Identitty` a hidden input field with the current ID of the `Identity` is placed here.
7	Html form to request the `DeleteIdentity` servlet using POST method. To identify the selected `Identitty` a input hidden field with the current ID of the `Identity` is placed here. If the delete button is clicked, the selected `Identity` will be deleted from the database, and the page will be refreshed

Table 5: Description of the search-identity page

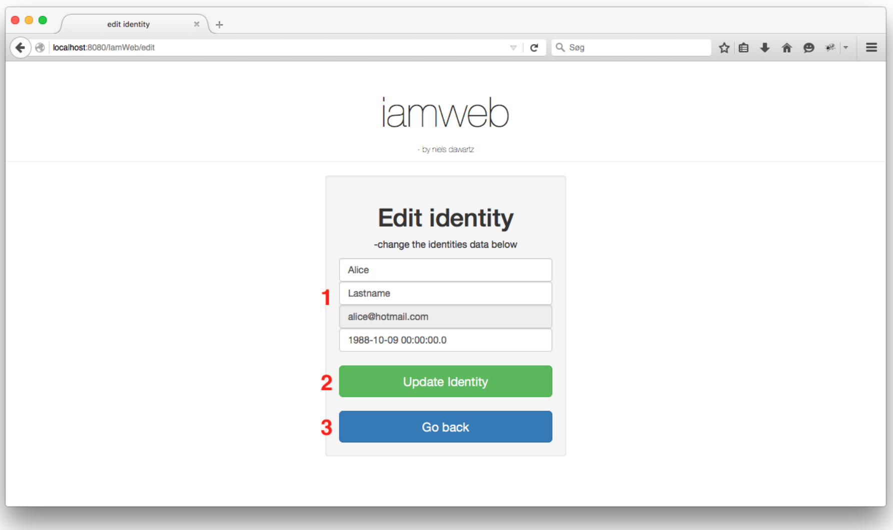


Figure 10: The create-identity page

Uid	Description
1	Html form with fields for editing an `Identity` firstname, lastname, email and birthdate. The email field is read-only, since it cannot be changed
2	The submit button of the form - requesting the `UpdateIdentity` servlet using POST method
3	Html form to request the `MainPage` servlet using GET method

Table 6: Description of the edit-identity page

5 Configuration Instructions

5.1 applicationContext

The applicationContex.xml consist of the overall preconfigurations of the database properties, and maps the differnt beans (clases) needed for the dependency injection engine (used for @autowiring). By mapping the different beans it's not needed to create instanceses, but only indicate where to put instances in order to make your application run. The application context consist of the following beans:

- userModel
- sessionFuncitons
- authenticator
- IdentityHibernateDAO
- beanBasedSessionFactory
- dataSourceBean
- hibernateProperties

mappings The `userModel`, `sessionFunctions`, `authenticator`, `IdentityHibernateDAO` are all placed as beans in the applicationContext.xml file, because of the mapping, needed in order to use the @autowiring 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 (attribute) is mapped, in order to tell hibernate, how to create the entities in the database, with it's corresponding coloumns.

dataSourceBean The dataSourceBean is the properties of the jdbc driver to the database. Properties like driverClassName, url, username and username are configured here.

hibernateProperties The hibernateProperties is the properties of the hibernate configuration in order 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 (derby) will understand. Furthermore, the config is set to display sql queries in the consol. Additionally the property `hibernate.hbm2ddl.auto` is used to ensure object representation will synchronize with database. When starting the application, it's set as create, to start the database from scratch, with no User's or Identities in the database. After the first execution, the property should be changed to "update" in order to keep the created objects in the database.

References

Derby website (n.d.). URL: https://db.apache.org/derby/.
Maven website (n.d.). URL: https://maven.apache.org/.
mkYoung. *Why i choose Hibernate for my project?* URL: http://www.mkyong.com/hibernate/why-i-choose-hibernate-for-my-project/.
MVC design pattern (n.d.). URL: https://developer.chrome.com/apps/app_frameworks.
Spring website. URL: https://spring.io/.