Spring 2017

Advanced Java Project

IAM System

Vanessa Vargas Garnica

Table of Contents

[Subject description 3](#_Toc484564088)

[Subject analysis 3](#_Toc484564089)

[Major features 3](#_Toc484564090)

[Frameworks 3](#_Toc484564091)

[Application Feasibility 4](#_Toc484564092)

[Data description 4](#_Toc484564093)

[Expected results 4](#_Toc484564094)

[Scope of the application 5](#_Toc484564095)

[Conception 5](#_Toc484564096)

[Data structures 5](#_Toc484564097)

[Identities 5](#_Toc484564098)

[IAMUser 6](#_Toc484564099)

[Address 6](#_Toc484564100)

[Global application flow 6](#_Toc484564101)

[Create an identity 9](#_Toc484564102)

[Search Screen 9](#_Toc484564103)

[Identity update 10](#_Toc484564104)

[Delete an Identity 11](#_Toc484564105)

[Identity Details 11](#_Toc484564106)

[Success 12](#_Toc484564107)

[Global schema 13](#_Toc484564108)

[GUI description 13](#_Toc484564109)

[Login Page 13](#_Toc484564110)

[Welcome page 14](#_Toc484564111)

[Create Identity Page 14](#_Toc484564112)

[Search Page 15](#_Toc484564113)

[Identity Details Page 15](#_Toc484564114)

[Identity Edit Page 16](#_Toc484564115)

[Configuration instructions 16](#_Toc484564116)

[Prerequisites 16](#_Toc484564117)

[Get project from GitHub 16](#_Toc484564118)

[Database Connection Configuration 17](#_Toc484564119)

[Schema creation 20](#_Toc484564120)

[Database Configuration 22](#_Toc484564121)

# Subject description

The IAM Project is the final project presented by the students of the Advanced Java class.

This project is the implementation of an Identity Management Software as a web solution. The project was partially developed during the semester’s class and has been completed by the student.

# Subject analysis

## Major features

The IAM Project has 4 major features.

* Authentication: A User should be authenticated before being able to work and manage the Identities.
* Manage Identities: the user should be able to
  + Create,
  + Update,
  + Delete Identities from the application.

To do this, the Identities and the users are stored as application data. We use Derby as the Database engine for managing the entities.

### Frameworks

Maven

A framework that allows us to package our projects in a more portable way. Maven helps handle jar dependencies easily through the pom.xml file.

Junit

Helps us create test classes. With special tags to better handle the initialization and end of test classes. Tags las @BeforeClass and @AfterClass allows us to initialize values and terminate sessions or open database statements. This framework allows us to develop with TDD and is integrated with Maven.

Log4j2

A tool to log to the console. Here, we can handle 5 levels of logs: error, warning, info, debug and trace. To use it we need to create a pattern, this patter would be used at the moment of the logging and will also have a threshold that will indicate up from which level we will have the logs printed.

Spring

A configuration framework that will allows us to implement in an easy way IoC providing a list of predefined instance providers that will be used by the code. Annotations are used to identify the instances that are to be automatically recognized by the framework. To use spring a configuration file has to be made created, in this file we will place as beans all the classes we will want to have available.

Hibernate

This framework is an ORM which allows for mapping an object to a table in the database. The base of hibernate is the SessionFactory, it is here where we specify to what database engine and the database instance and schema, as well as the data base authentication parameters. Since the database engine is configured in the factory session there is a level of isolation as to which engine we are using, so it is easy to do changes.

## Application Feasibility

With the knowledge obtained from the Java class during the semester and considering the development done in class we can easily see that this application is completely feasible under the Java platform.

Also, we can clearly see the utility of building such an application that allow us to clearly see the process of managing entities.

## Data description

In this case, we must represent two different types of entities: Identities and Users.

The requirements do not specify anything regarding the managing of users, only the Identities, for simplicity no implementation has been made to manage users, the only user is created when running the test that creates the database.

Both, users and identities are stored in a relational database, along with address entities that are related to the identities.

The tables for these entities are automatically generated thanks to the Hibernate framework when the configuration test is running.

## Expected results

The user is expected to interact with the application using the web UI.

The three basic management functions should be completely functional so the software to be useful and complaint with specifications.

## Scope of the application

As stated above, the application is limited to only manage Identities and not to manage the Users. This is of course one of the first steps to be improved in a next revision.

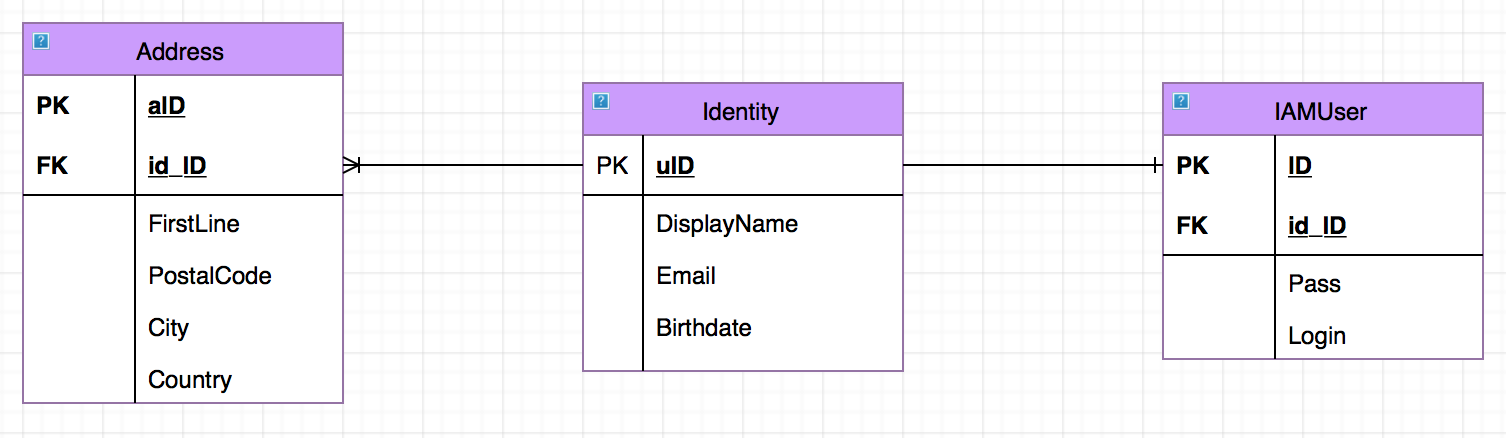
Also, the Identities are only going to be managed on a database, there is no current way to select a file as storage mode for Identities, so the user must have installed a Database engine software.

Also, during classes the concept of a List of addresses related to an identity was introduced. As for the scope of this project, an address cannot be created or edited, these again is functionality that fits a revision and upgrade of the current project.

# Conception

## Data structures

As mentioned Hibernate is the framework selected to deal with the data model - database relationship. So, we have a data model package where all our tables are described as classes, once the application builds Hibernate takes the tags found in the data model classes and creates the respective tables in the database. Here is an entity relationship diagram of our three tables.



### Identities

Identities are the entities that are going to be managed, within the application we should be able to create, update and delete these. All identities are going to be stored in a Derby database.

Per the requirements an identity can have:

* UId: User Id, a unique id auto-generated and auto-incremented as a Primary Key in the Identity table.
* Display Name: represented as a string.
* Email: represented as a string also.
* Birthdate: stored as a Date.

### IAMUser

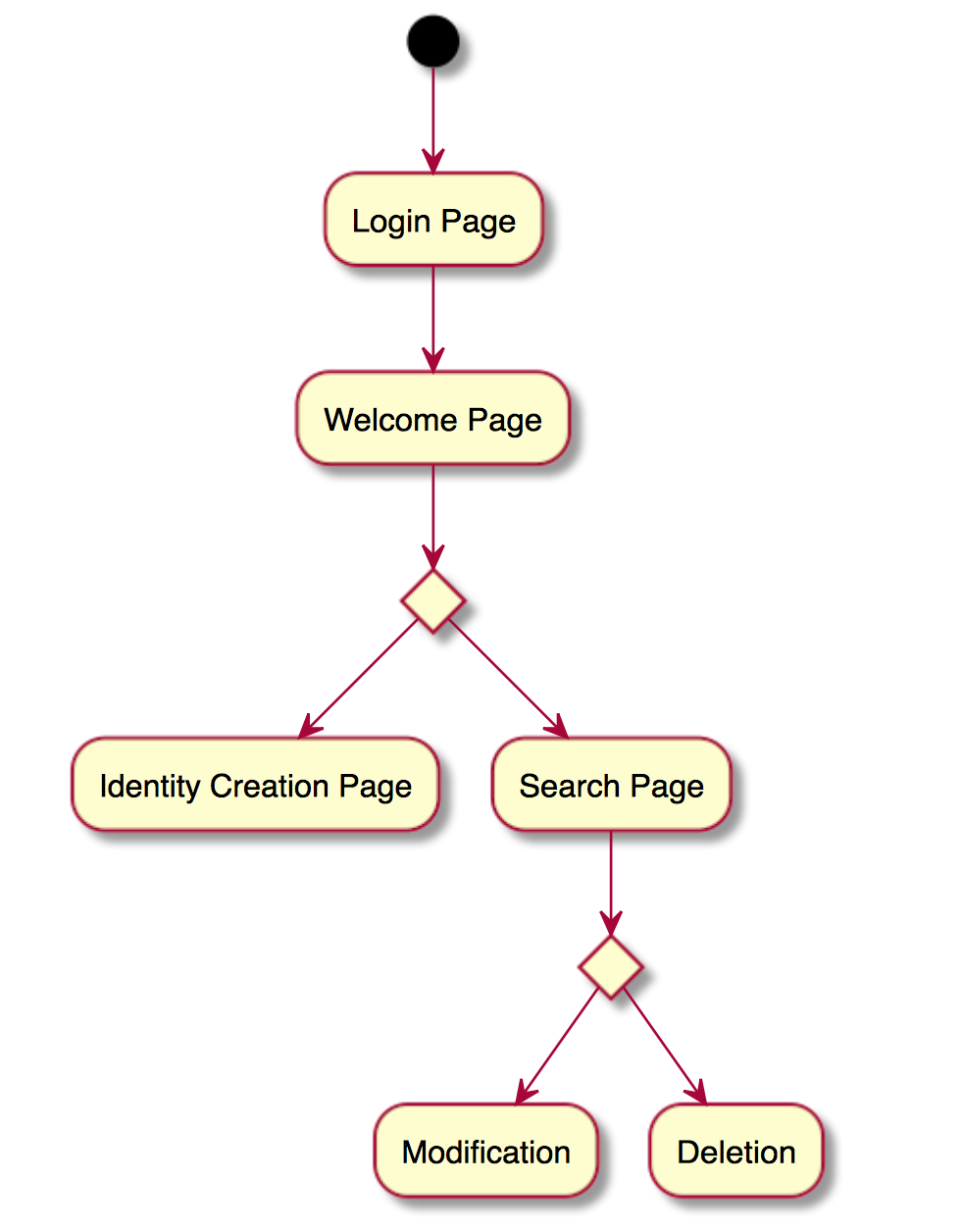
Users are not supposed managed by the application. A user will have an Id, login and a password, both are stored as string. Because of the one to one relationship a user holds with an identity has a foreign key to this table.

### Address

Address table has its own id as primary key and as with the user table, we have a foreign key to identity table, however, in this case the relations ship is many to one.

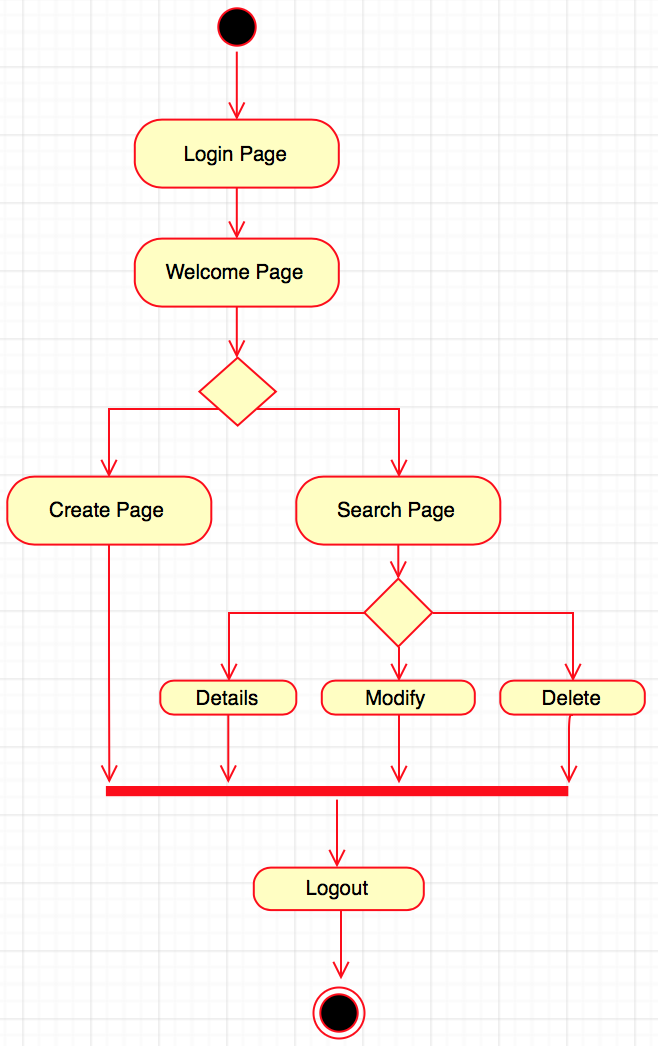
## Global application flow

The application flow was presented in the requirements of the project as the following diagram.



So, the implementation respected this diagram to fulfill the requirements, though a new screen was added at the Modification / Deletion level, the new screen is a Details screen that will allows us to see the addresses a user may have.

Also, a logout functionality was implemented and this can be accessed from any of the tasks. The following is the modified application flow.



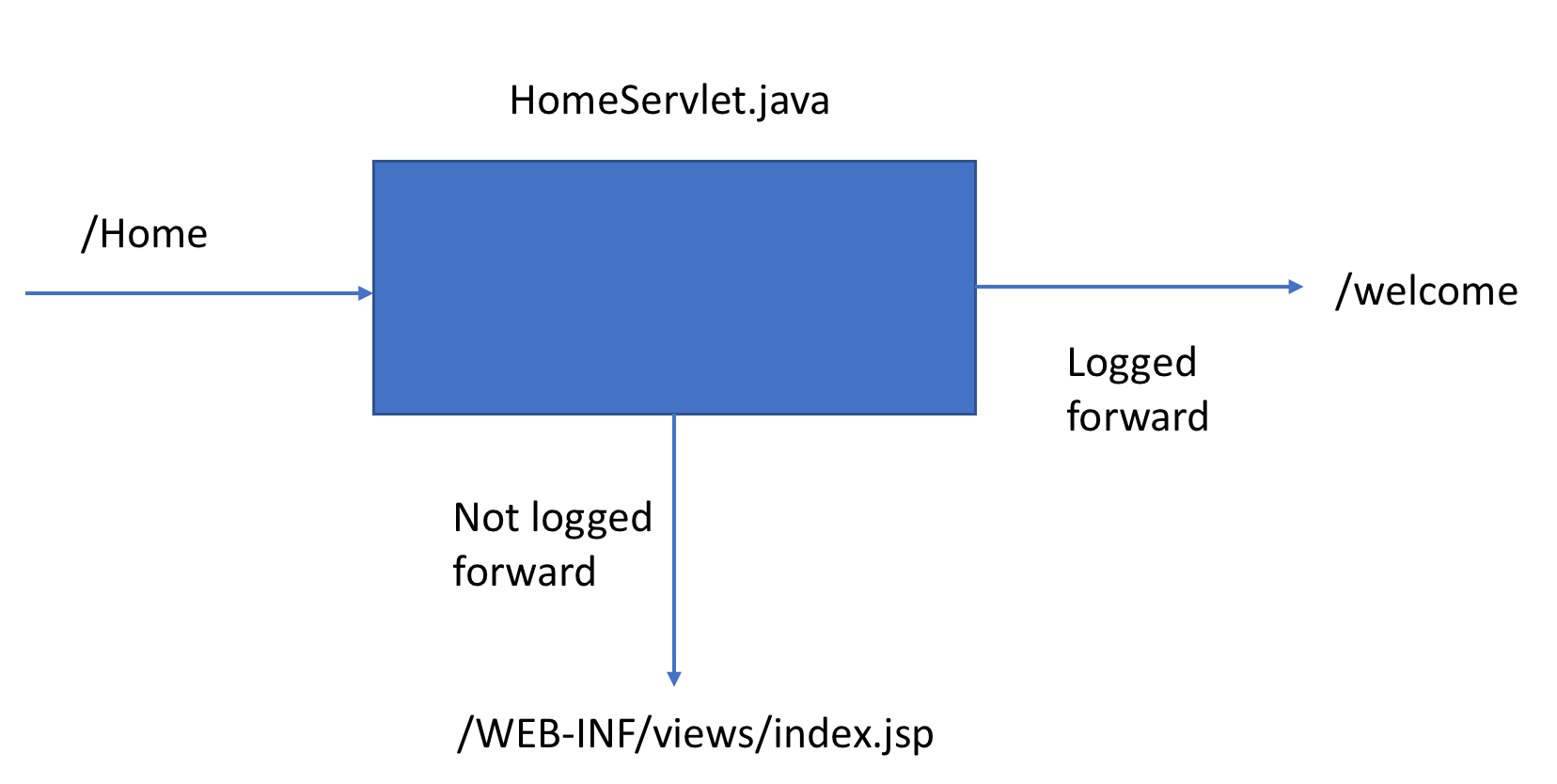
The entry point for our web application is the ‘\’ local path. Using the servlet architecture, a path /Home was declared in the web.xml file:

<welcome-file-list>

   <welcome-file>home</welcome-file>

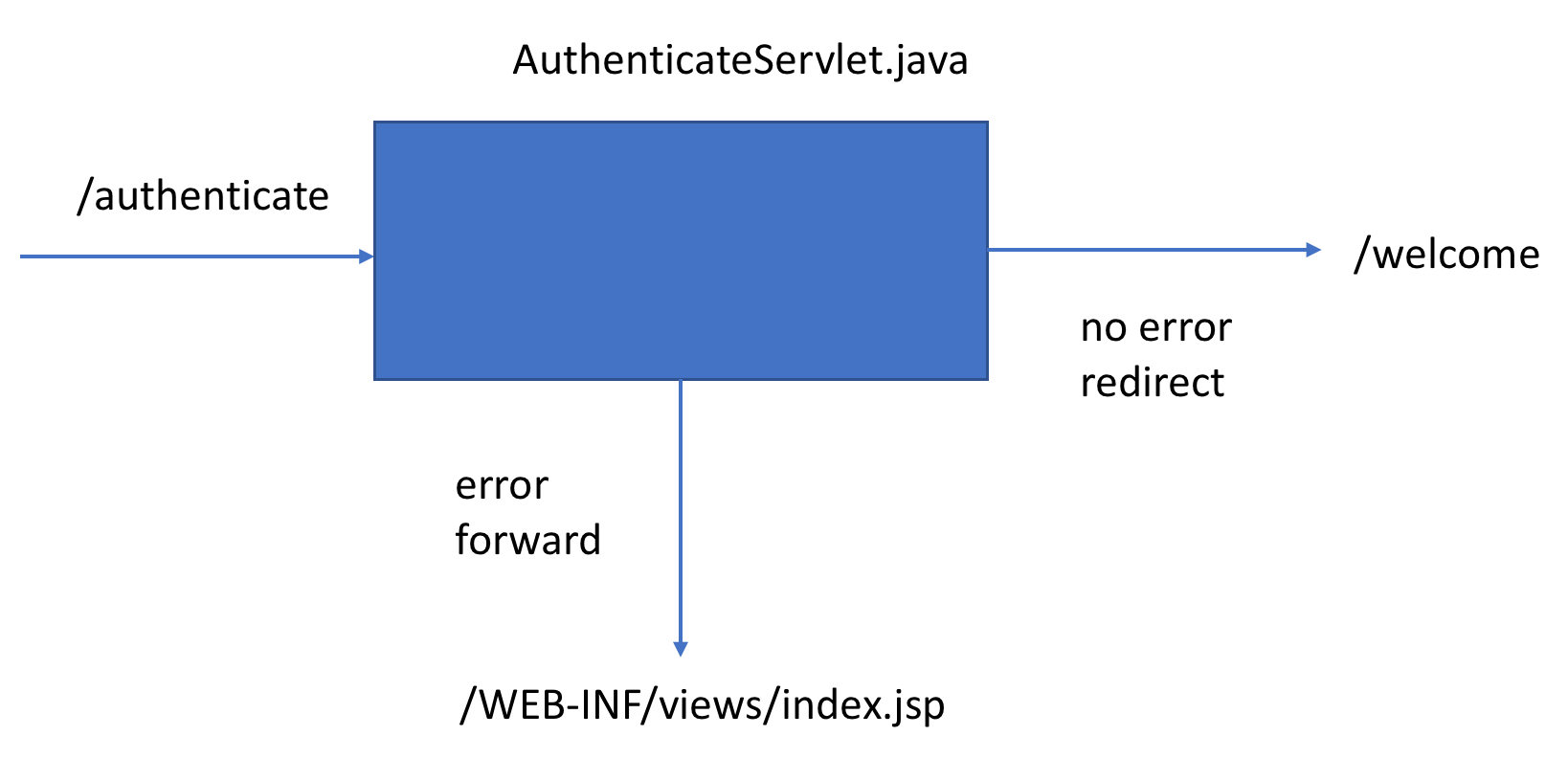
 </welcome-file-list>

The following diagram illustrates how the servlet implementation was done for the rendering of the login screen as entry point of our application.



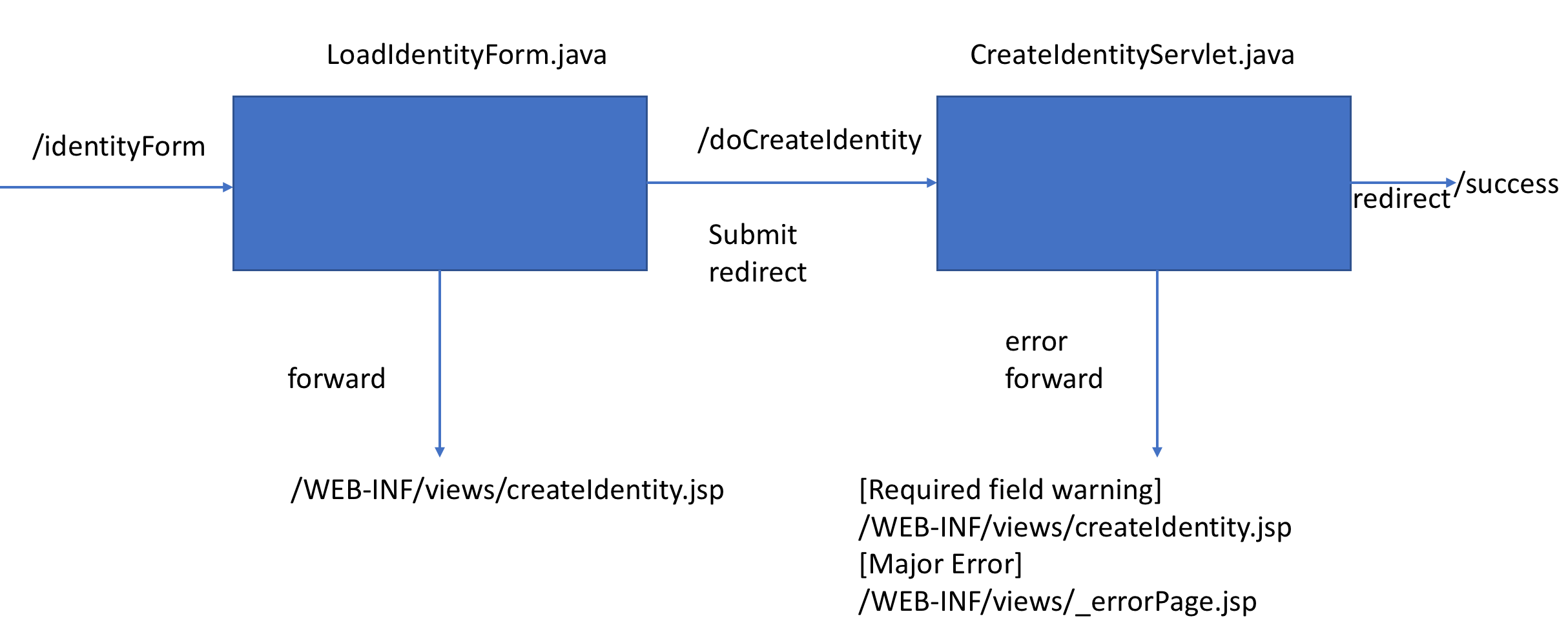
To determinate if a user is logged we are storing the user in the session, in that way, if we can retrieve it from the session we know the user has already logged in previously and we redirect him directly to the welcome page.

When the user is on the index.jsp page he can try to authenticate with a valid login and password. Once the submit button of the form is clicked, this will reach the authentication servlet process which will try to compare the given credentials whit those stored on the database, in case of failure the user would stay on the index.jsp page with an error message, otherwise it will be redirected to the welcome screen.



### Create an identity

The following is a description of the process that occurs when clicking on the “Create” button from the welcome screen.

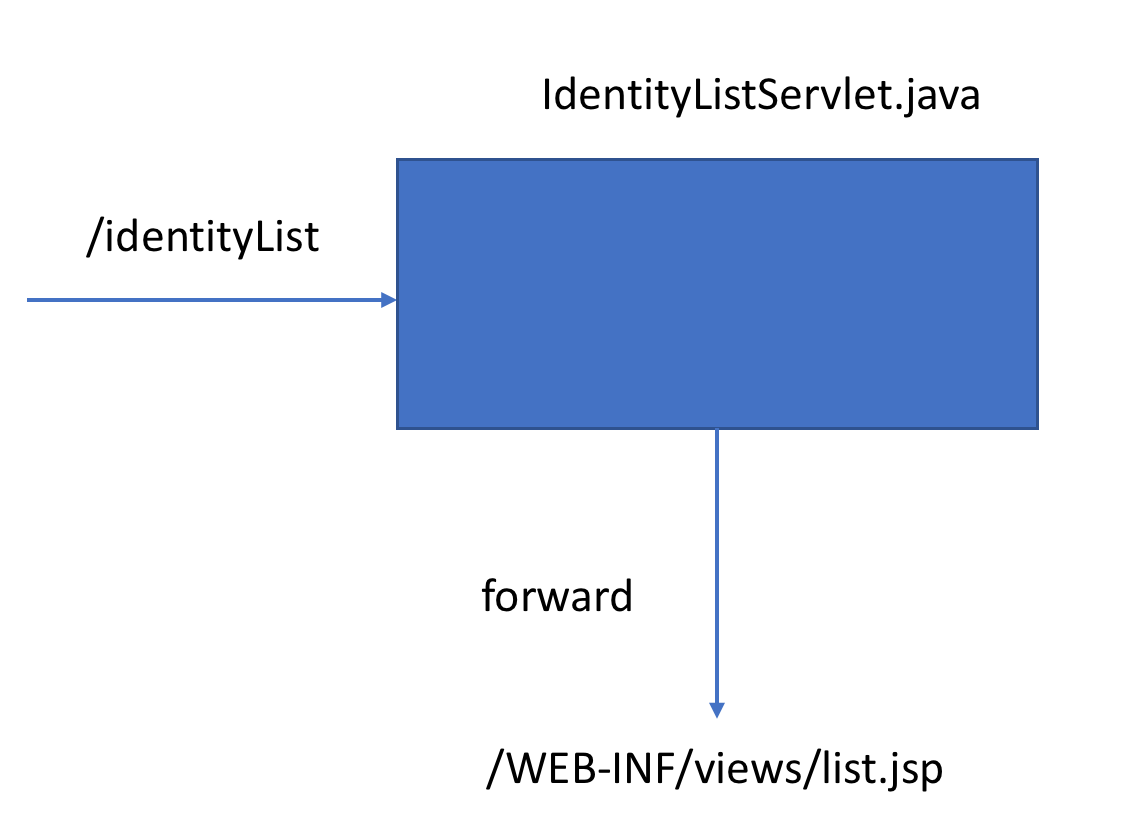


So basically, we are first requesting to load the Identity form view. Once it is loaded and we have done the changes and we want to submit the form, we can either have an error on the form, or encounter an error while saving which will forward us to different pages. If everything goes well, we will be forwarded to a success servlet that centralizes the success process for all our actions in the application. We’ll describe the behavior of this servlet later.

### Search Screen

The search screen is the central part of the application since from it we can view, modify and delete other identities. The search screen will always execute the same code behind, if there are no parameters passed to it to search upon, it will do a blank search returning all the identities on the database. Otherwise, according to the selected field it will filter the results based on the parameter from the search text.

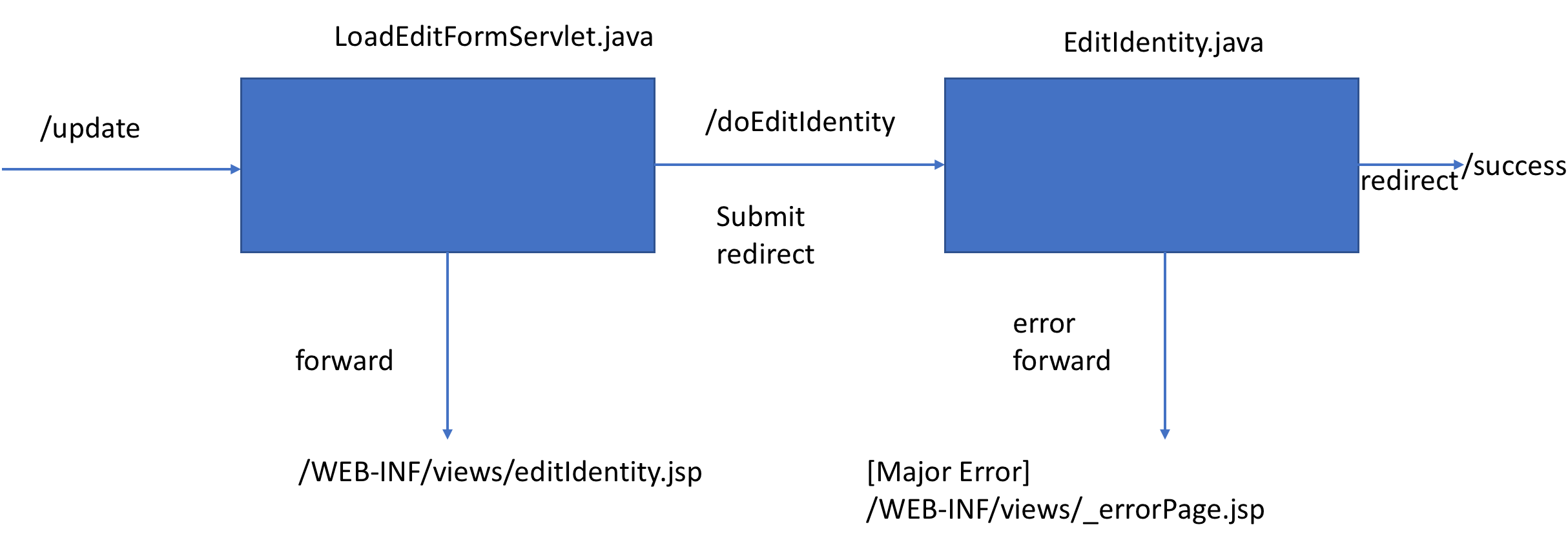
The servlet architecture is presented on the following image.



### Identity update

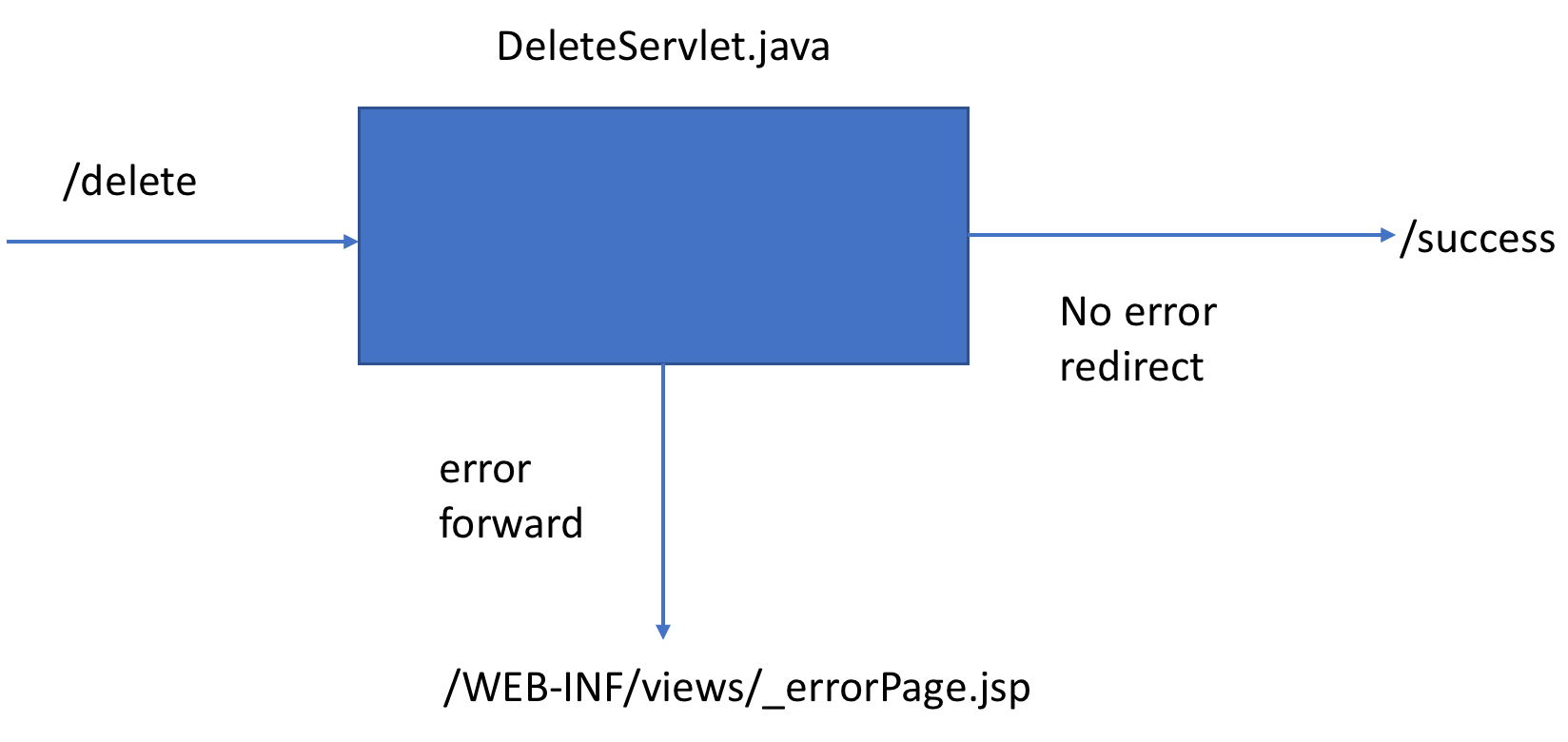
Like what happens for the creation of the identity, on the update identity process we involve two servlet classes. One to load the form, and a second one to process the submission of the form. The diagram bellow illustrates this process.

The /update has as parameter the uId of the identity we are going to edit. so, the LoadEditServlet.java class is the one to load the information from the database and prepare it on the jsp page to be edited.



### Delete an Identity

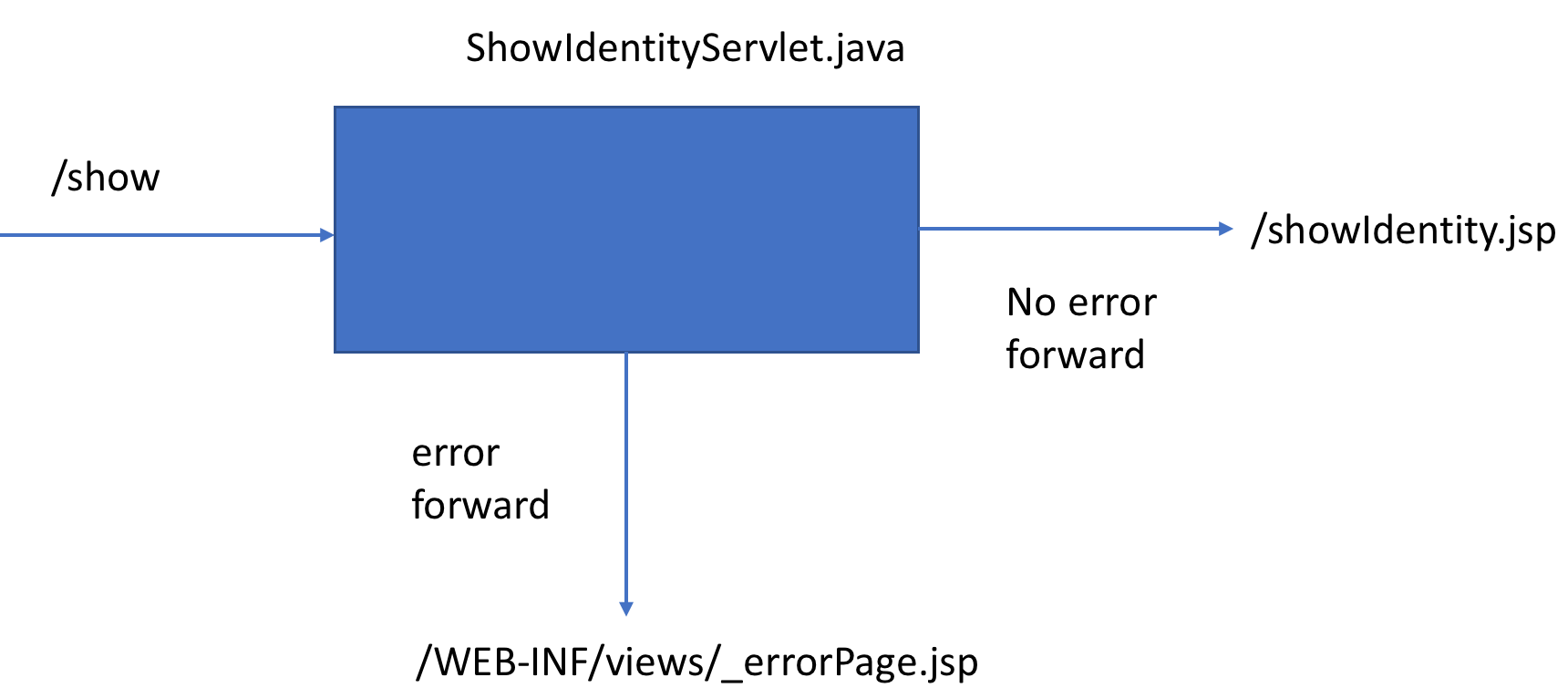
Delete Identity will be a straight forward process compared to create or update, mainly because we do not need to pre-load a form screen in the middle of the process. This process again, has as parameter the uId of the identity to be deleted. The servlet is in charge to look for the identity on the database and delete it using the Identity Data Service. If the operation succeeds we will be redirected to the success servlet, else we will be forwarded to the error page.



### Identity Details

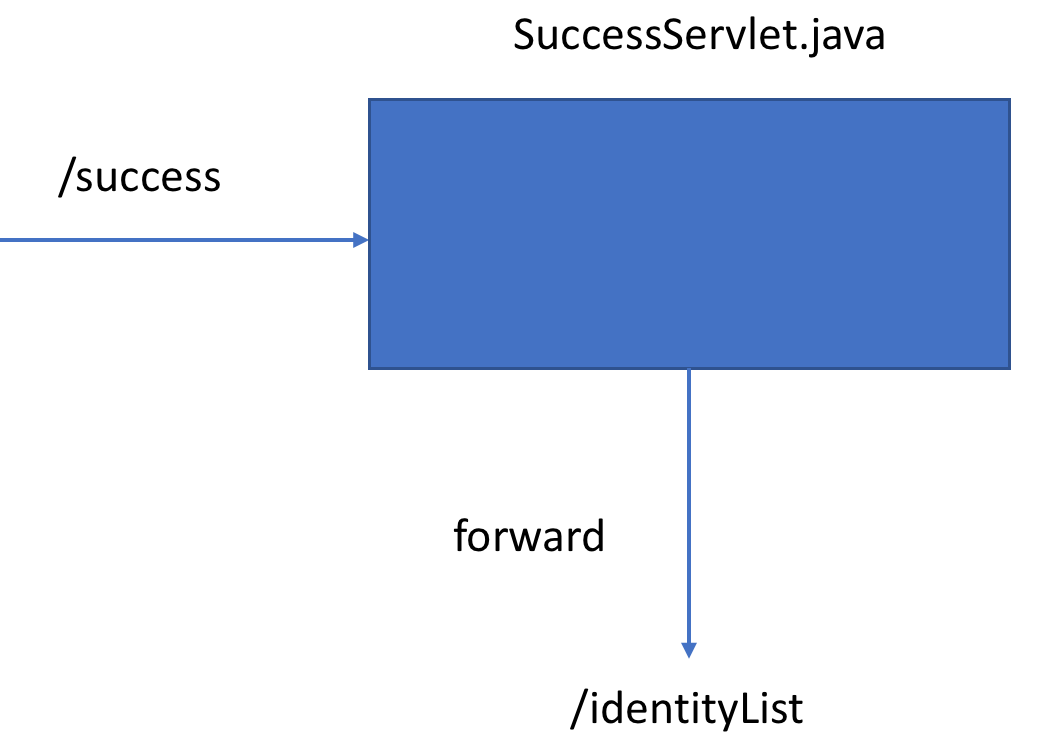
This option was not on the original requirements; however, because of the addition of an address objet to be tied up to the identities it was a necessary new screen to be able to show these properties to the user.

As with the update and delete cases, we pass as a parameter to the servlet the uId of the Identity we want to see. The servlet will then retrieve it from the database and load the appropriate details so they can be retrieved by the jsp view.



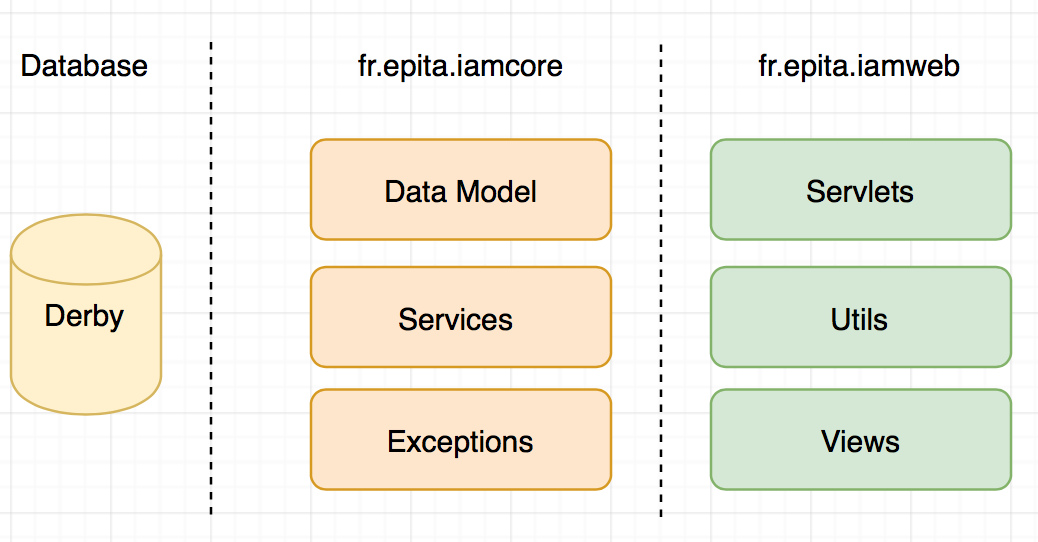
### Success

Lastly, the success in case of a creation/update/deletion is a single servlet which will forward us to the search screen and show us a success message. This implementation was put in place to avoid the user to have the capacity of re-send an already completed request, like trying to delete the same identity again.



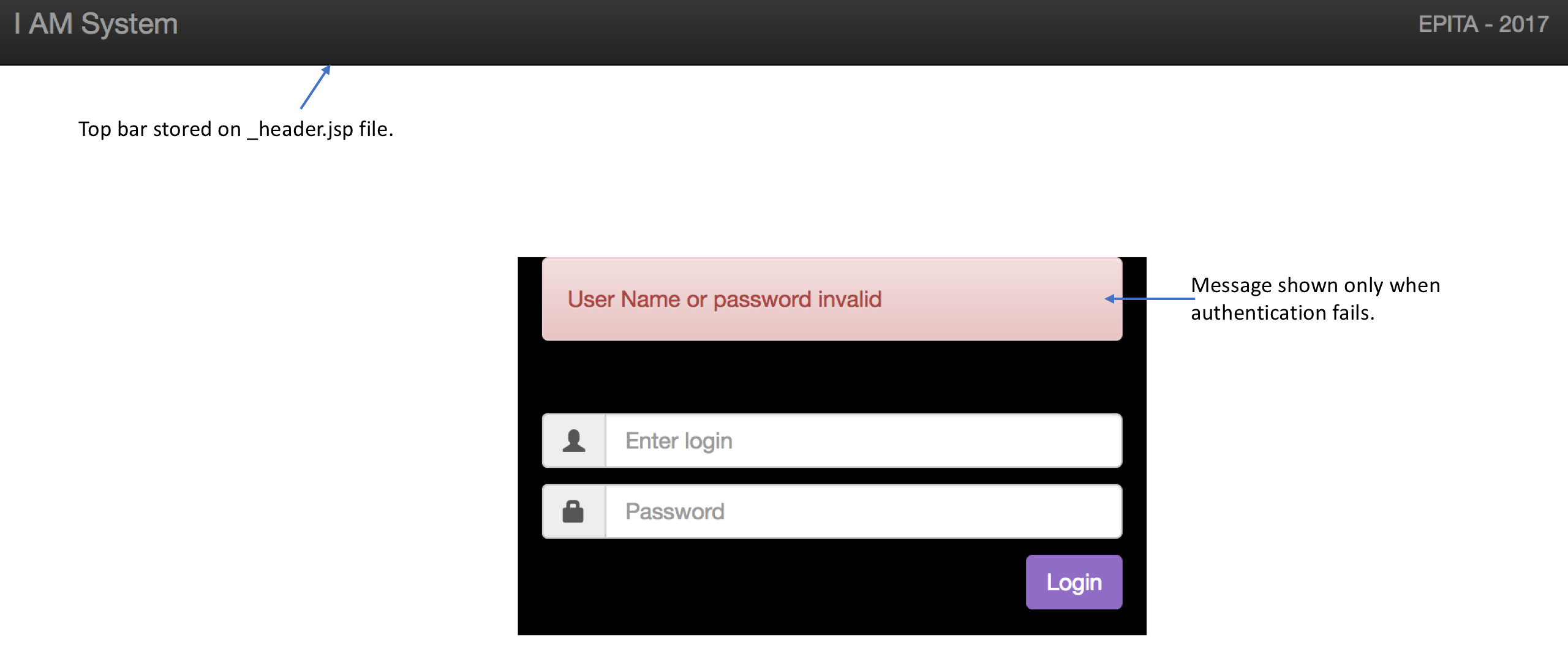
## Global schema

The image shows the global schema of the application. So we have isolated the data model and services processing in a “core” project, whereas we have all the web parts of the system in a “web” project, this provides isolation and flexibility for technology changes, so we could be able to use the backbone or “core” project with a no “web” solution.

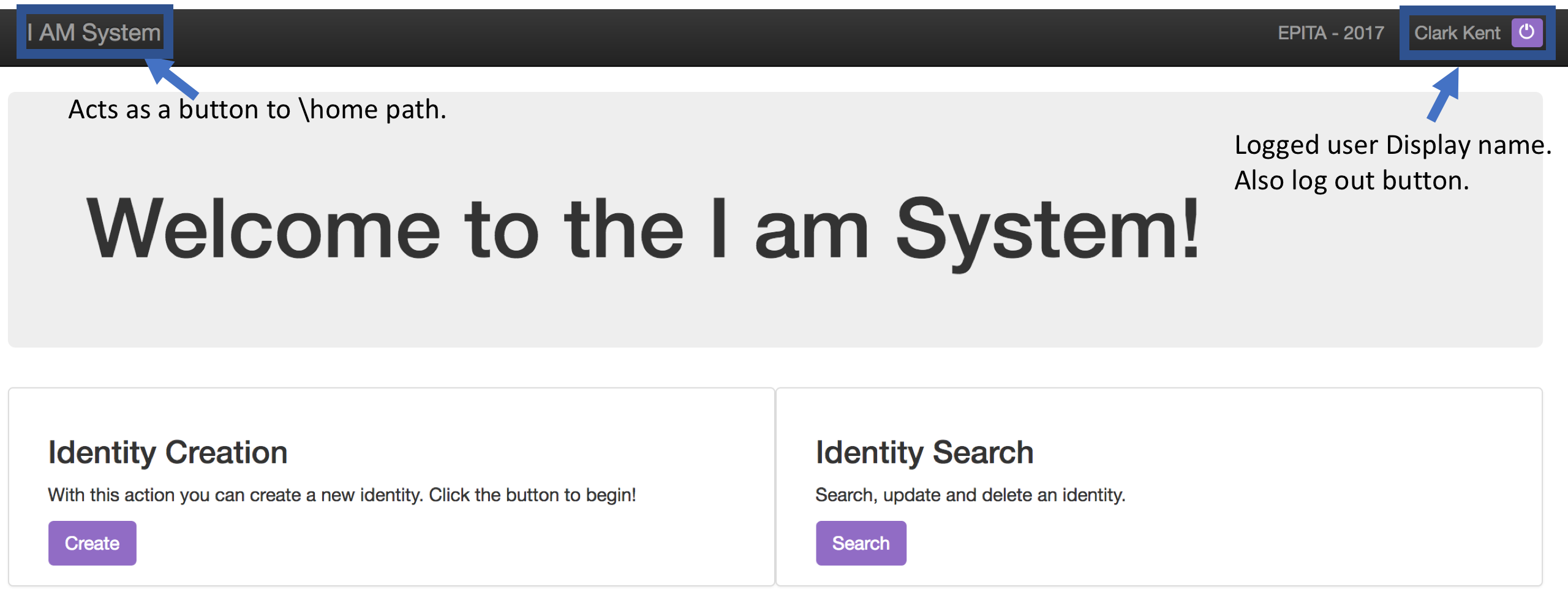


# GUI description

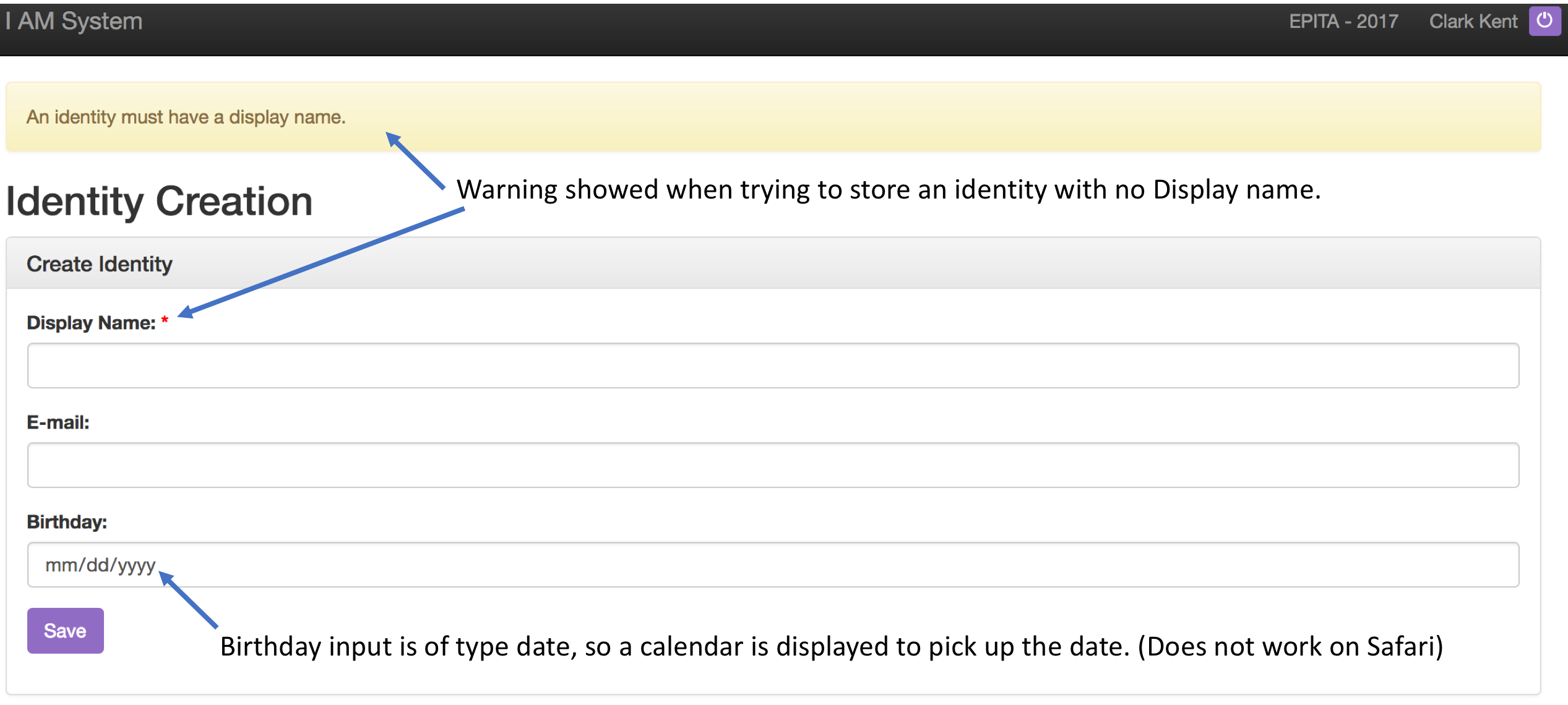
## Login Page



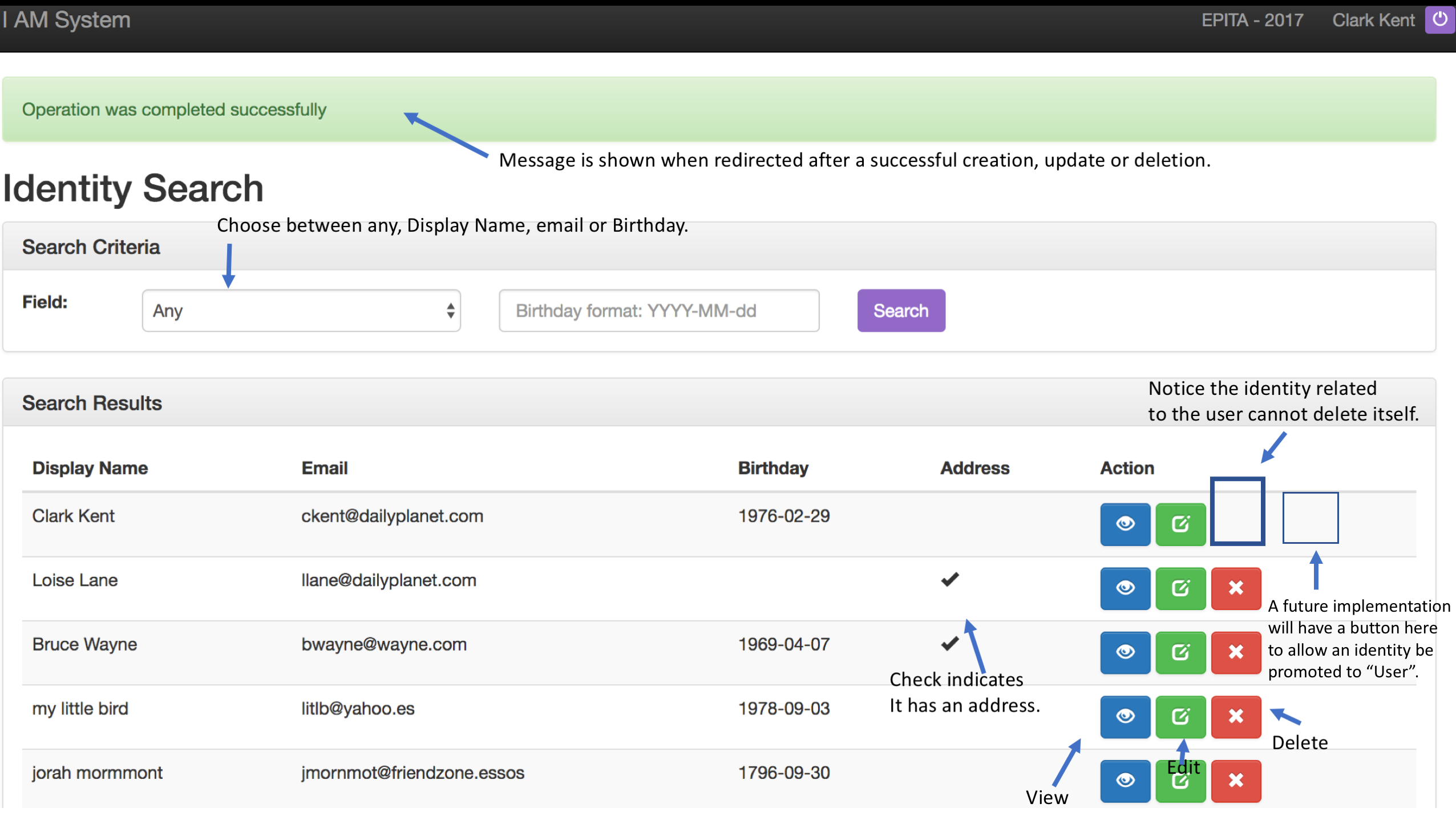
## Welcome page



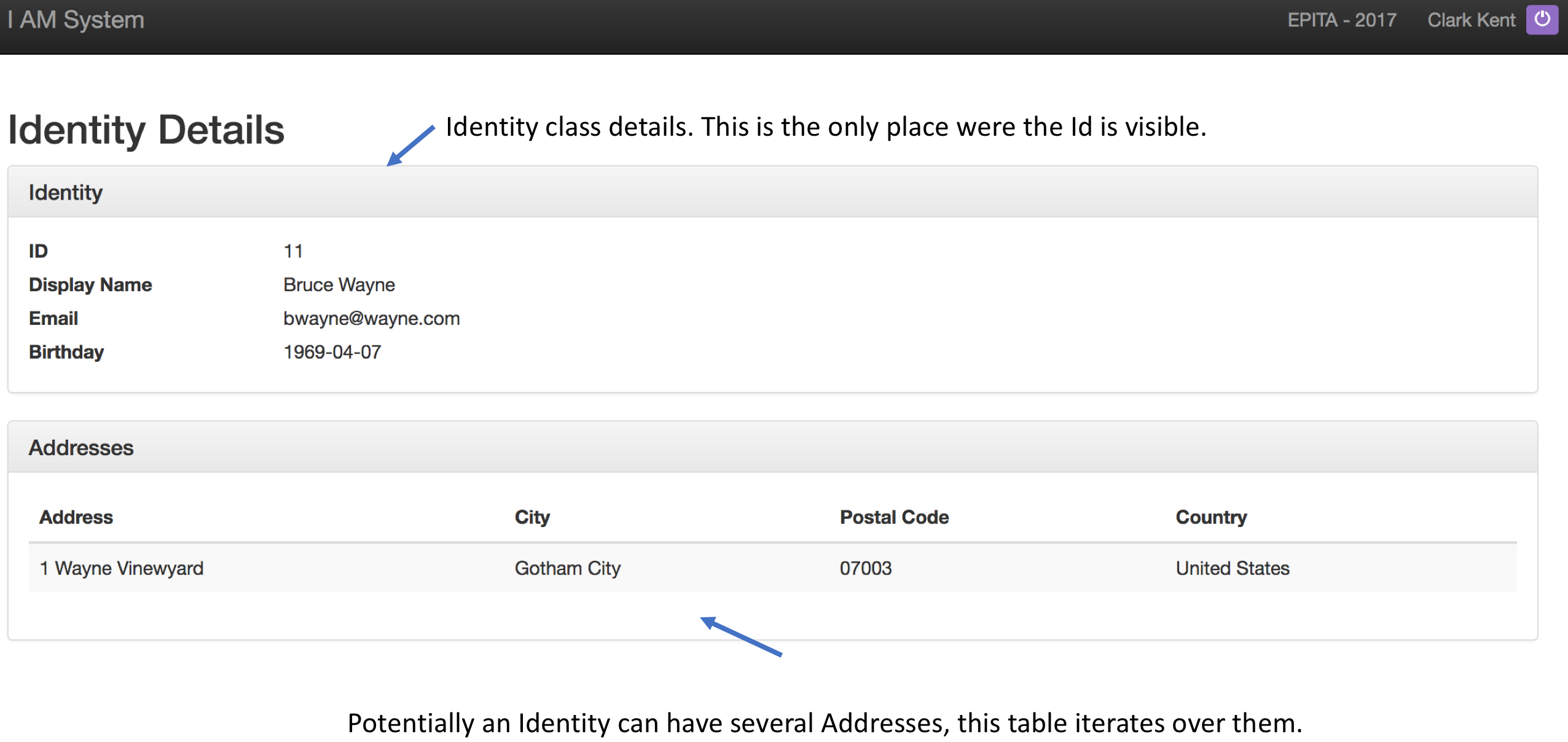
## Create Identity Page



## Search Page



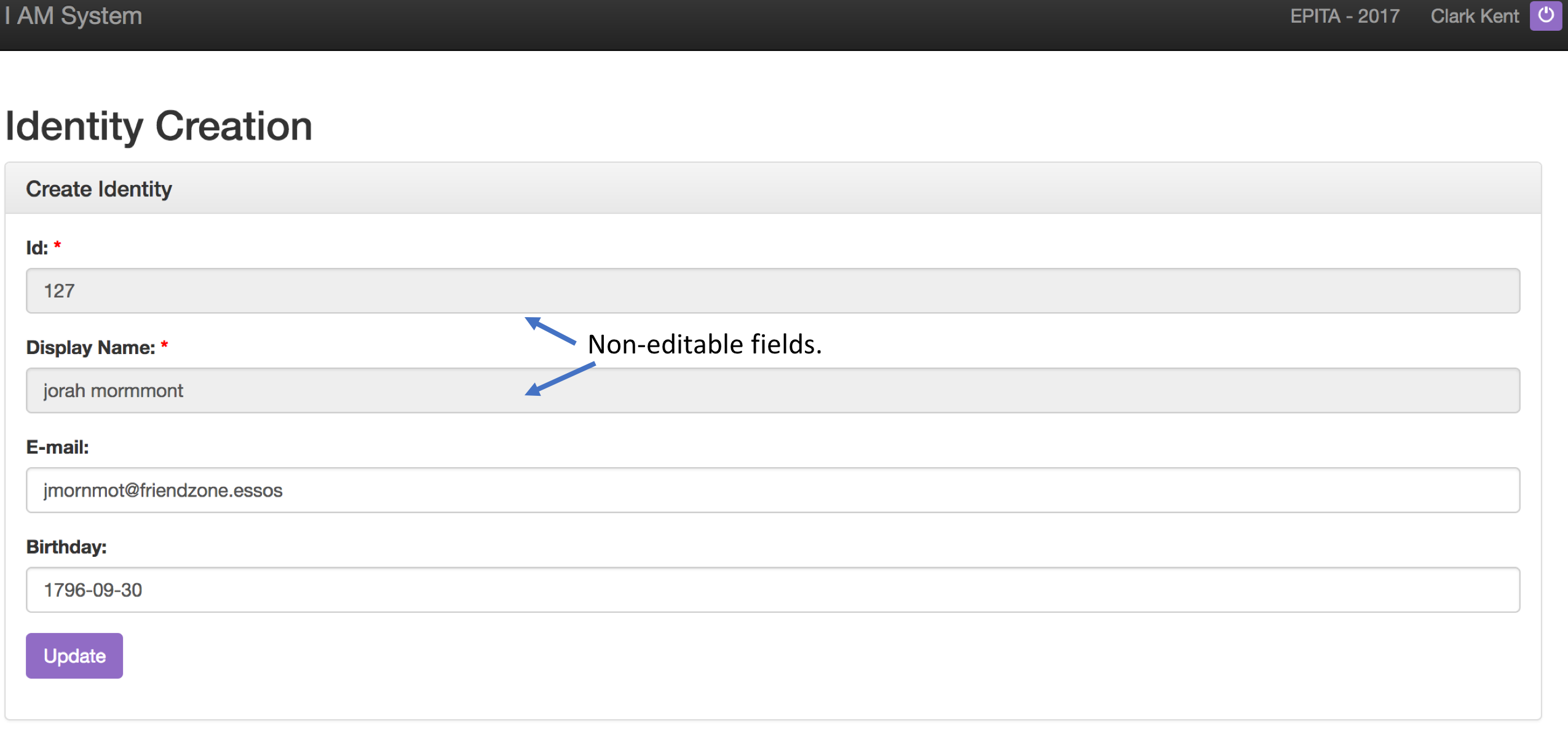
## Identity Details Page



An identity without address:



## Identity Edit Page



# Configuration instructions

## Prerequisites

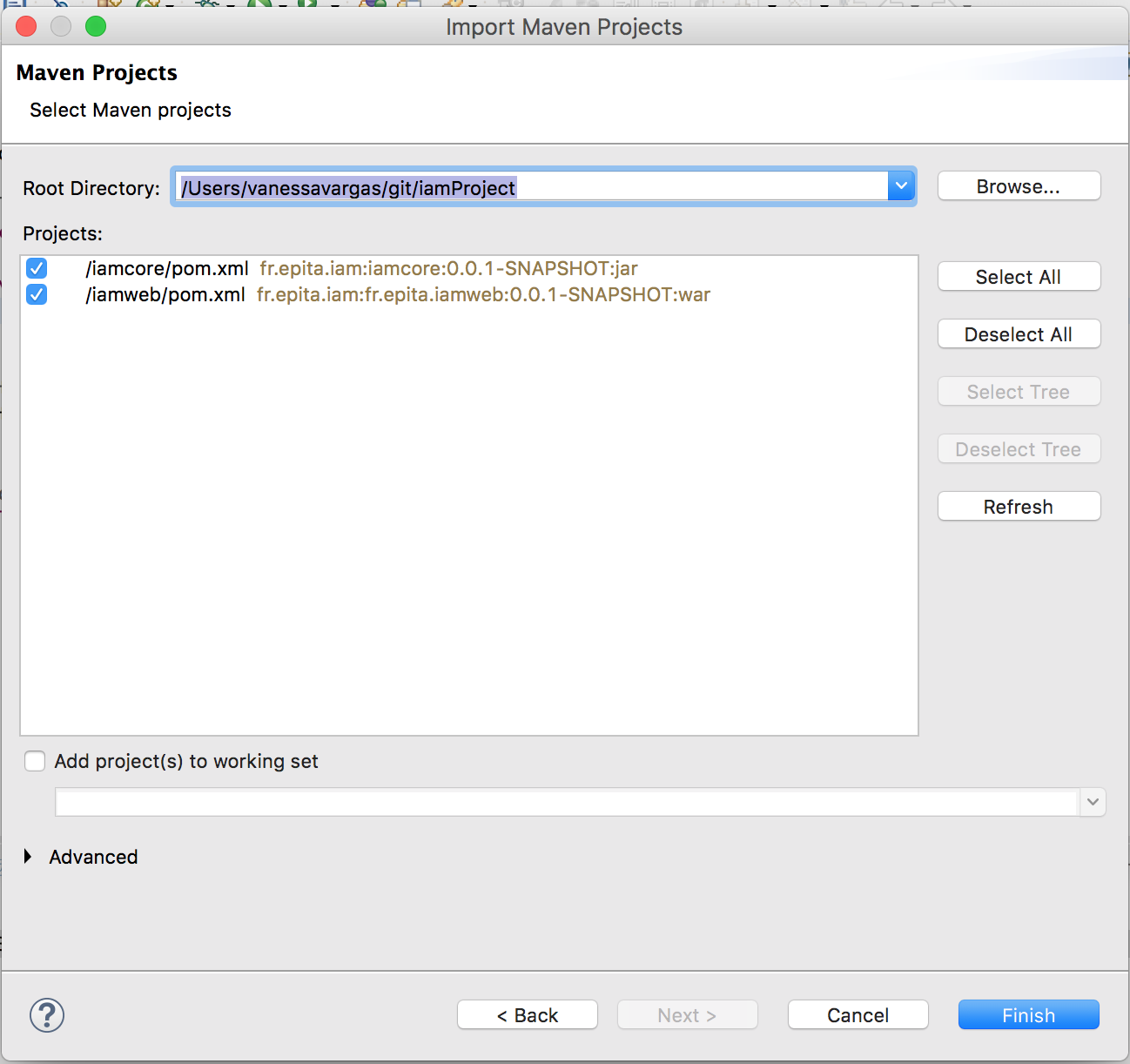
Please make sure you have installed the Java JDK, Eclipse Neon Java EE, Derby 10.13 and Tomcat v. 8.0.

## Get project from GitHub

1. Open Eclipse, and go: File -> Import
2. Select Git -> Projects from Git click Next
3. Select Clone URI click Next
4. Paste the following URI: <https://github.com/vanevar/iamProject.git> click Next
5. Make sure master branch is selected and click Next.
6. Select the destination Directory for the project and click next. Take note of the destination directory.
7. At this point the project has been downloaded to your disk, since it is a Maven project we will not want to use the Eclipse project wizard. So click the cancel button now.

## Importing Maven projects

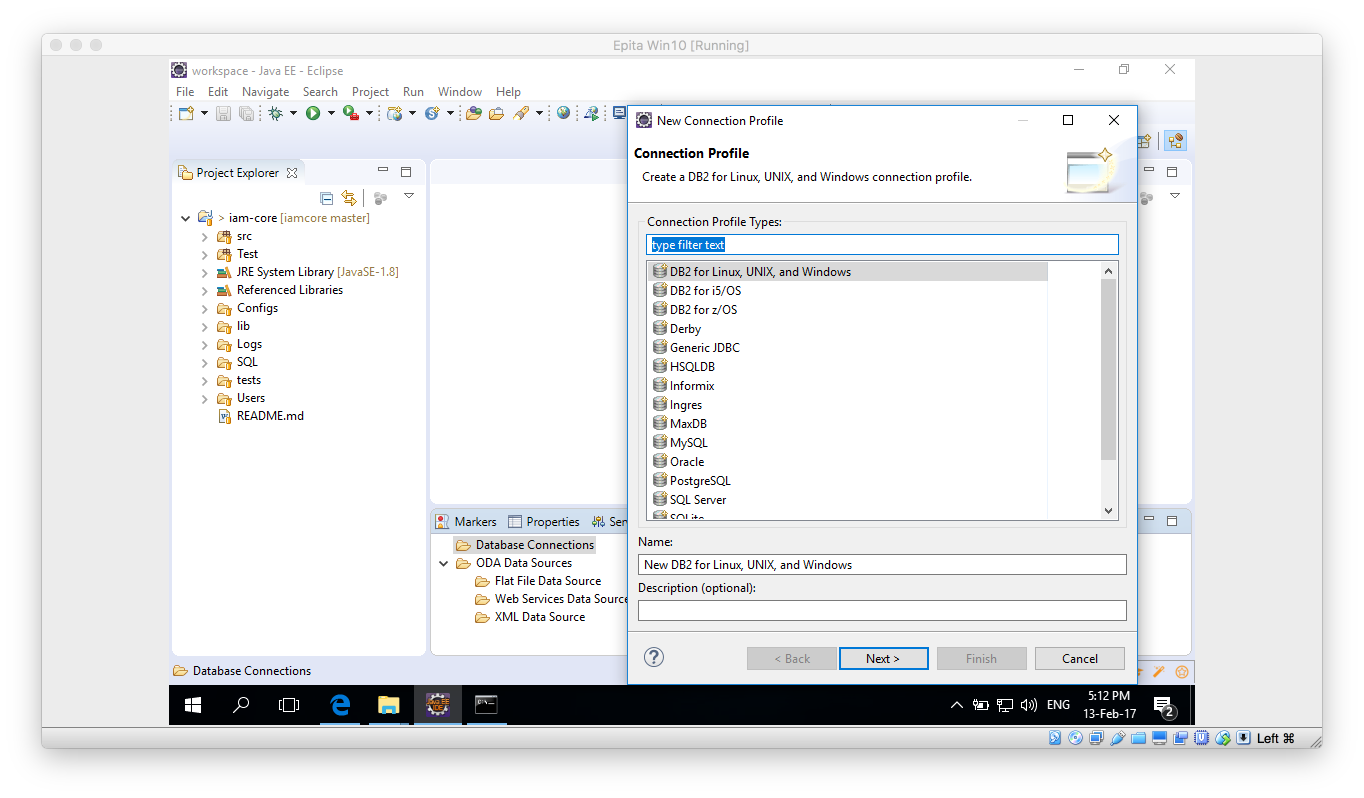
1. In Eclipse go: File -> Import
2. Select Maven -> Existing Maven Projects and click next
3. Browse to the directory where the files were downloaded from Github and select the folder that contains both the core and web projects.
4. The result should look like this:



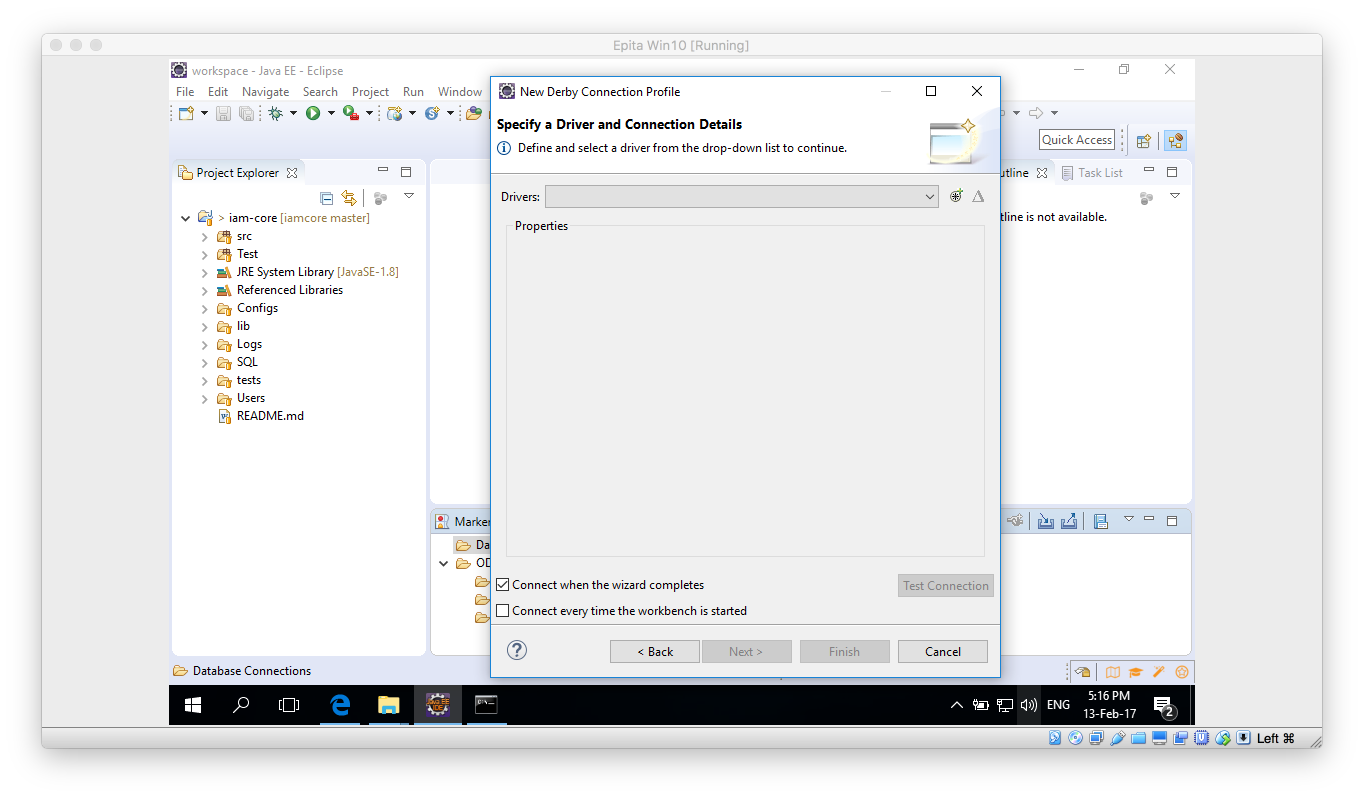
1. Click Finish.

## Database Connection Configuration

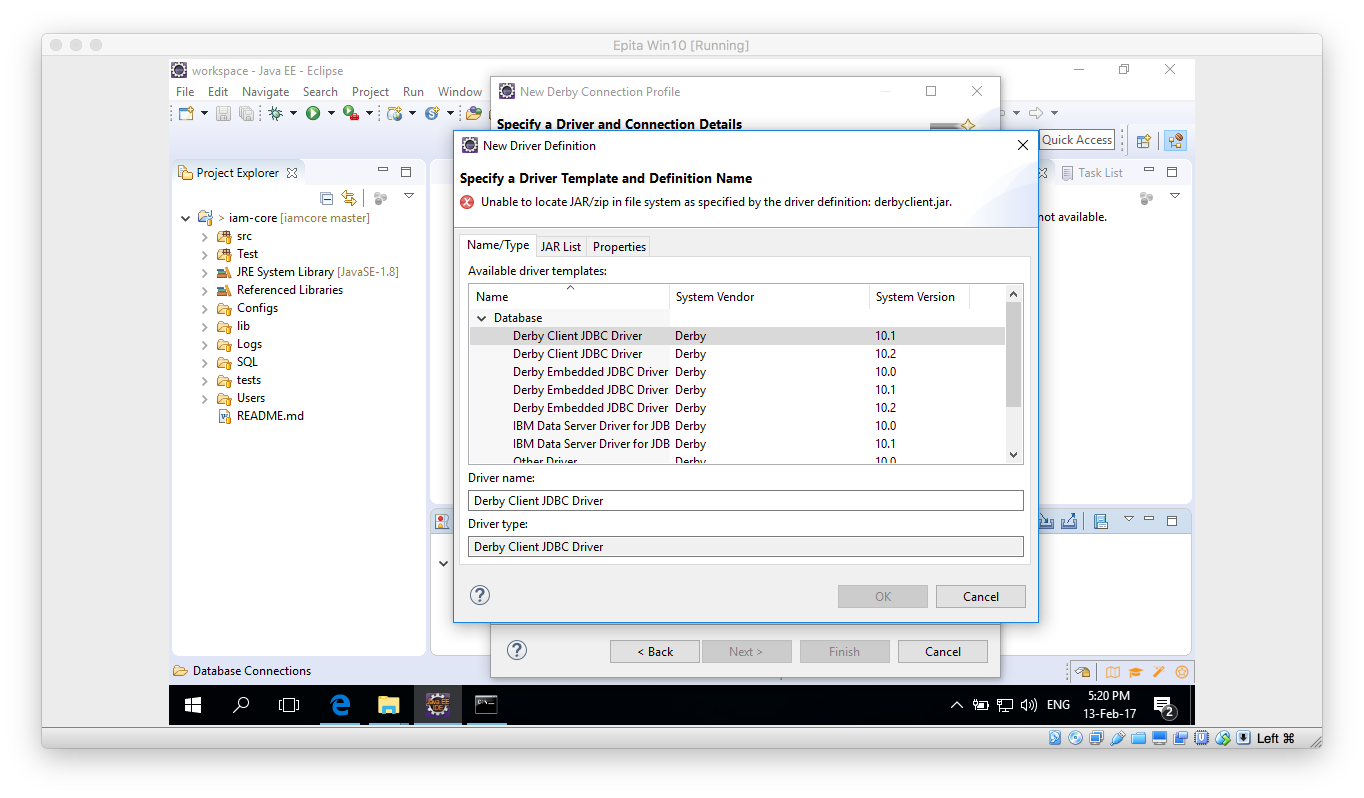
1. Start your Derby engine running the startNetworkServer file.
2. Open the Data Source Explorer window on Eclipse.
3. Right Click Database connections, select New.



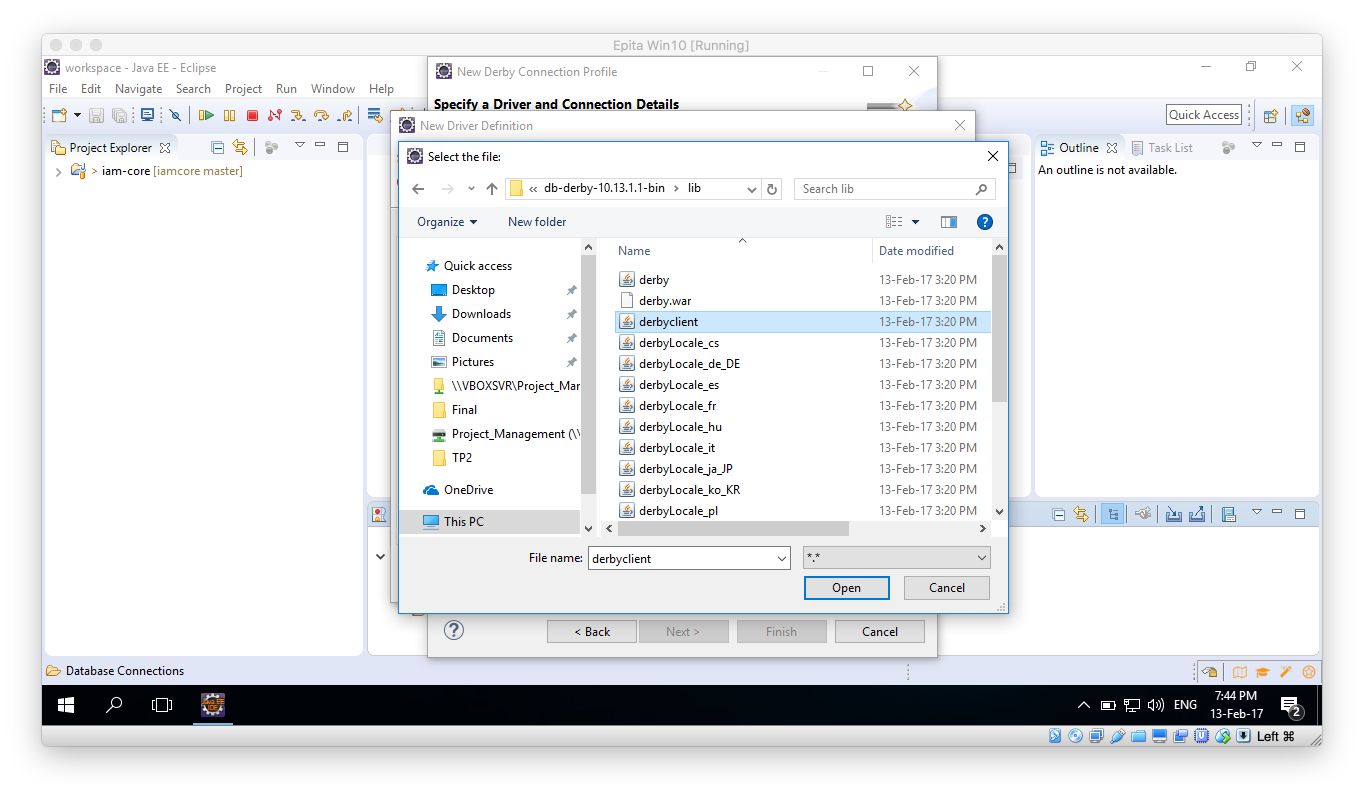
1. In the contextual window select Derby. Click Next.
2. In the new window click on the “New Driver definition button”



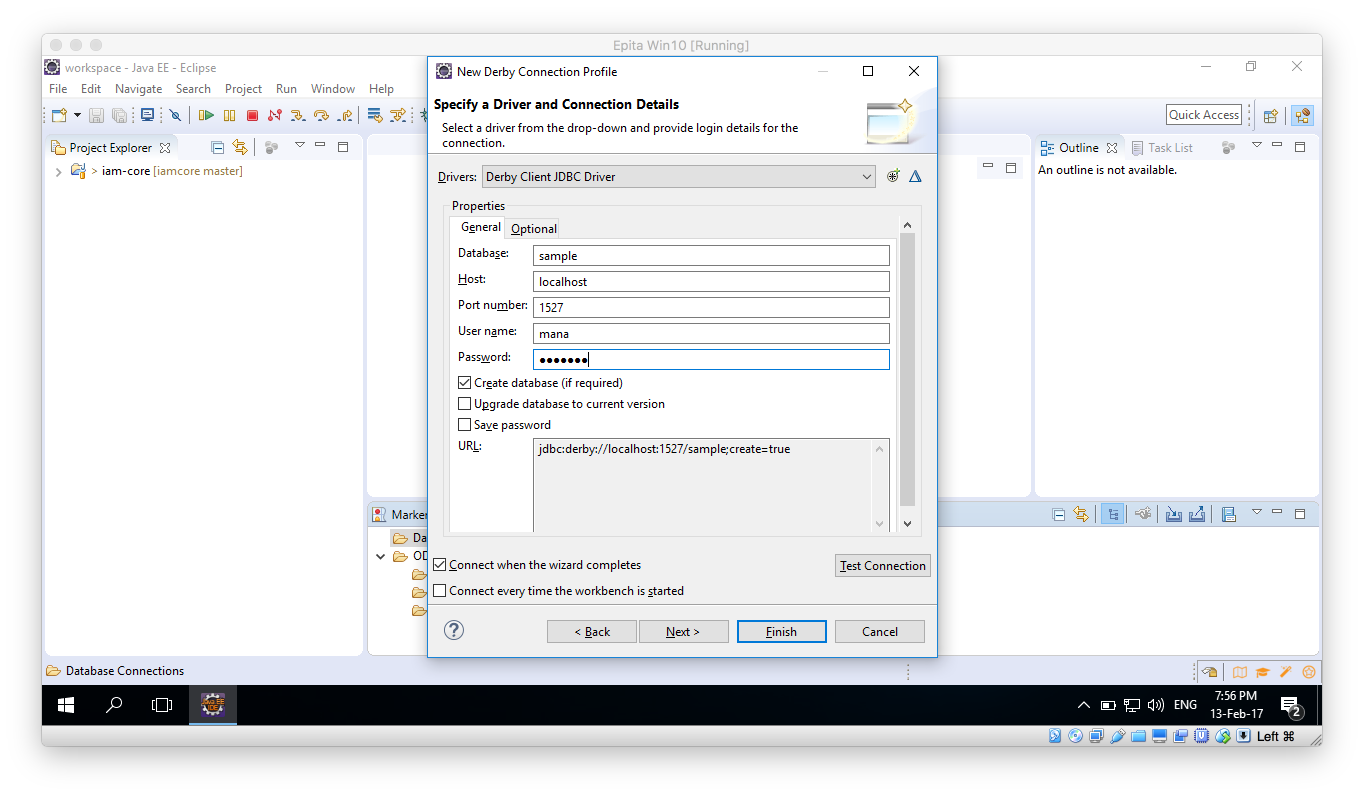
1. Select any Definiton on the Name/Type tab and then move to the JAR List tab.



1. On the JAR List tab click on Clear all button, so no files are listed. Then click on the Add and navigate to the lib folder inside your Derby database. Once there select the derbyclient.jar file.



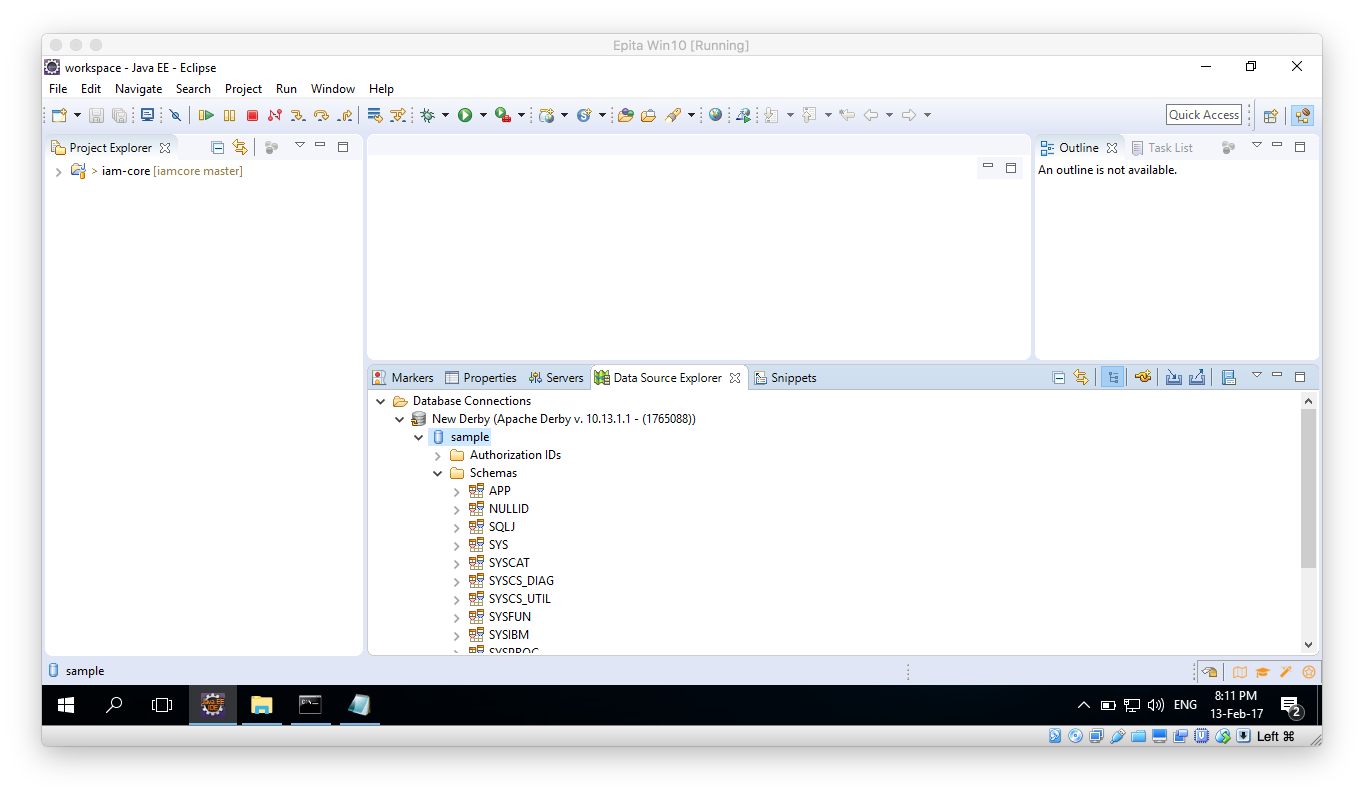
1. Click Open, then the “New Derby Connection Profile” window should be updated. Please take note of the URL and change the user and password if desired. Remember those because we are going to need them later on.



1. Make sure the Derby Database is running an click on Test Connection button, it should succeed.
2. Click on Finish button.

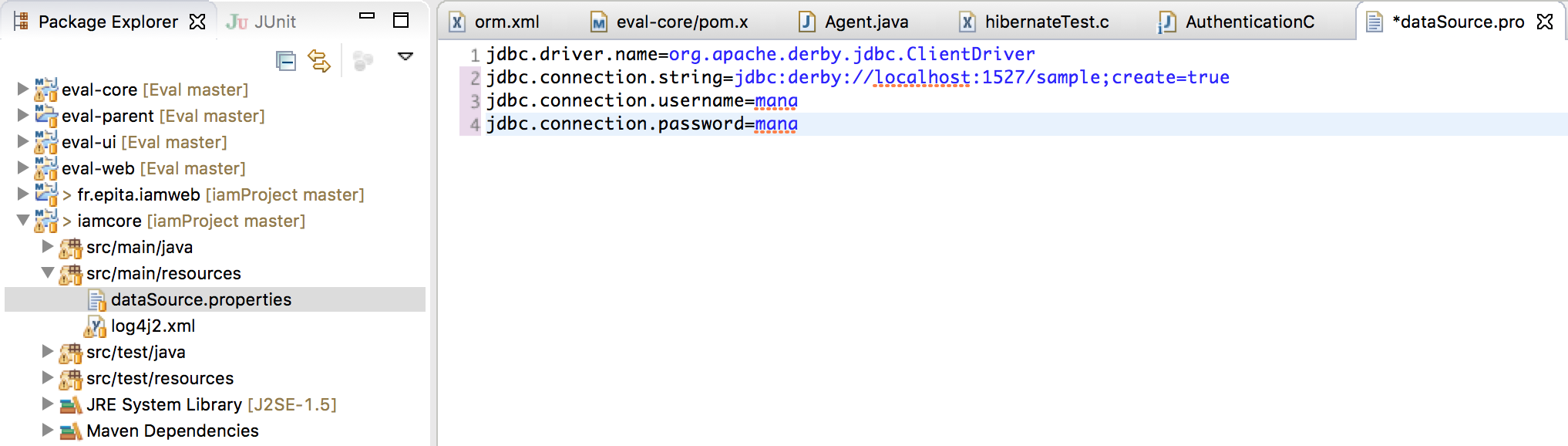
## Data base configuration and schema creation

1. Once the above process is done, within the Data Source Explorer window you should be able to expand all the way to Schemas:

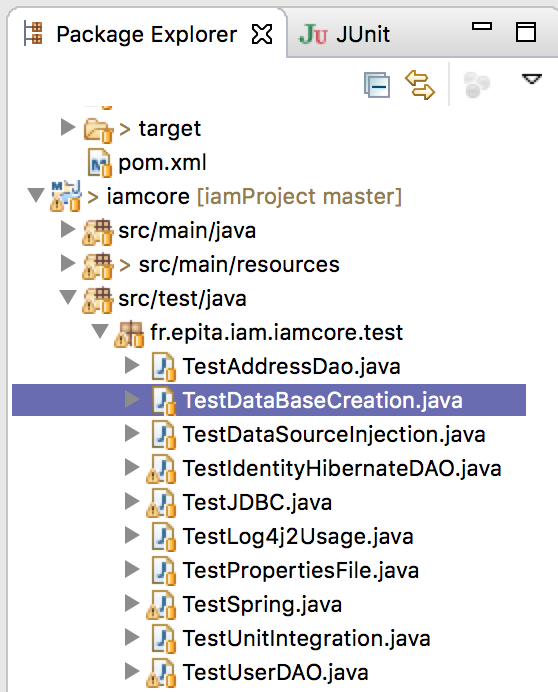


1. Now, you will need to update 2 files. On the Project Explorer window first look into the iamcore project and find a dataSource.properties file under the /src/main/resources path. Update this file in the following manner:
   1. Set the jdbc.connection.string to the one given on the previous step. In this case: jdbc:derby://localhost:1527/sample;create=true
   2. Update the user and password credentials to the ones you just used on the previous step.

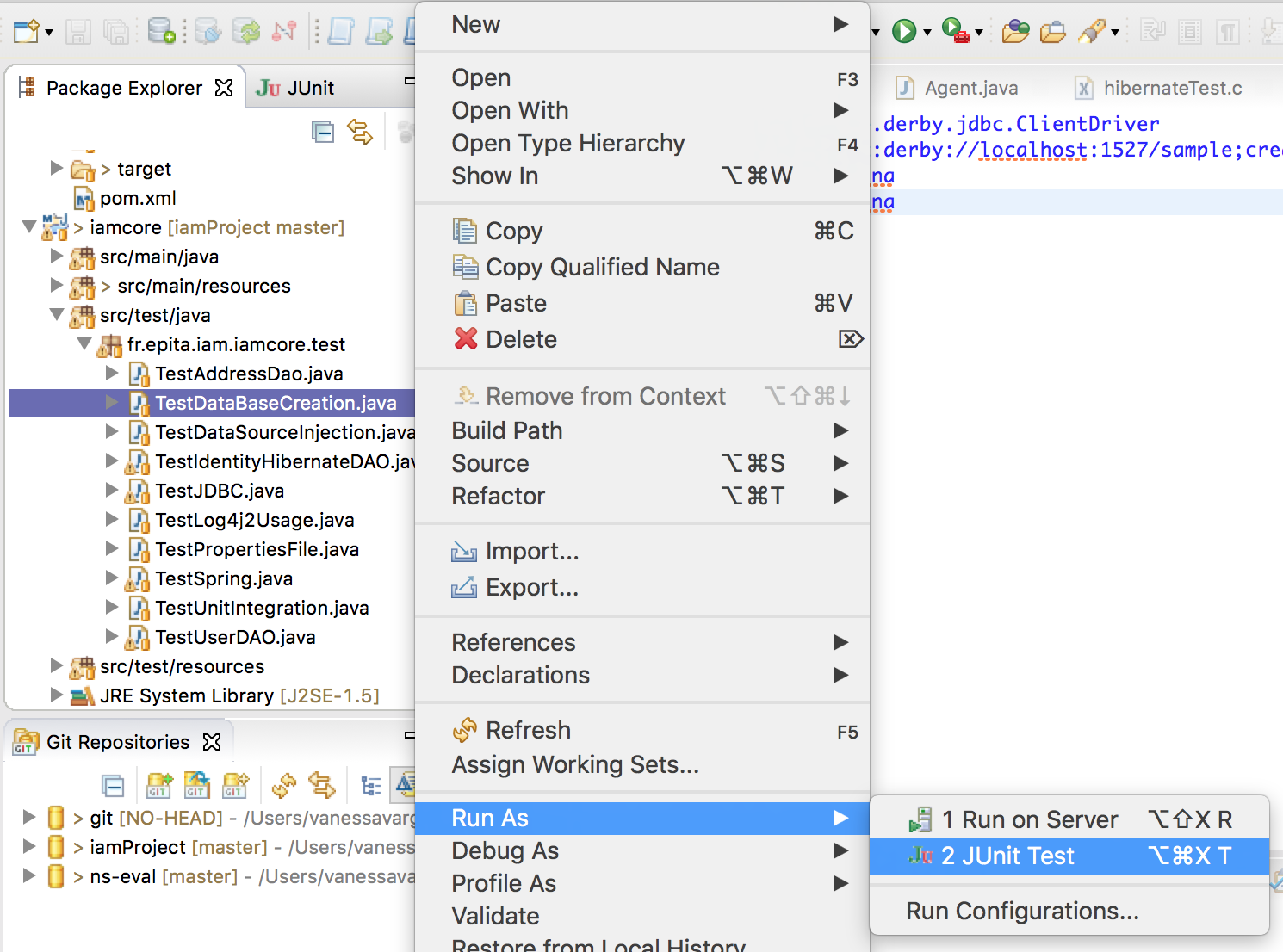
1. Once the new window opens, in the Project Explorer navigate: iam-core/SQL/IDENTITY table creation and open the file. Copy the file contents to the Scarpbook window and execute.
2. Right click anywhere on the Scrapbook window and select “Execute All”.
3. Execution should be successful:



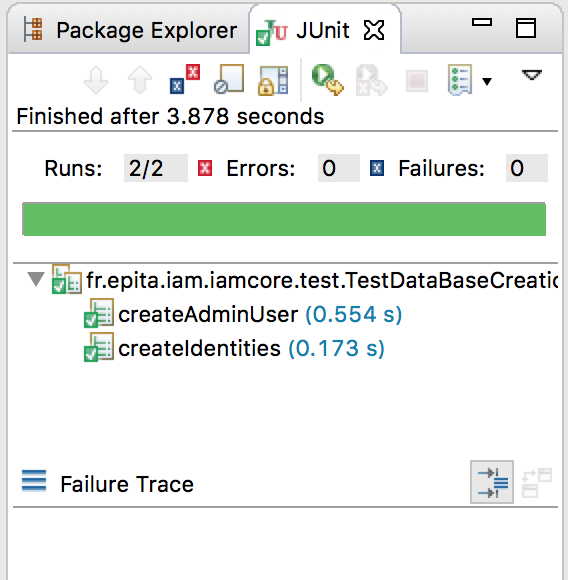
1. Save this file and close it. Now, do the same changes to the file found on the iamweb project under: src/main/resources save it and close it.
2. Because we have used Hibernate as an ORM, for the database to be created we will only need to run a test. It can be found on the iamcore project. In the Project Explorer go to iamcore -> src/test/java -> fr.epita.iam.iamcore.test package and find the TestDataBaseCreation.java



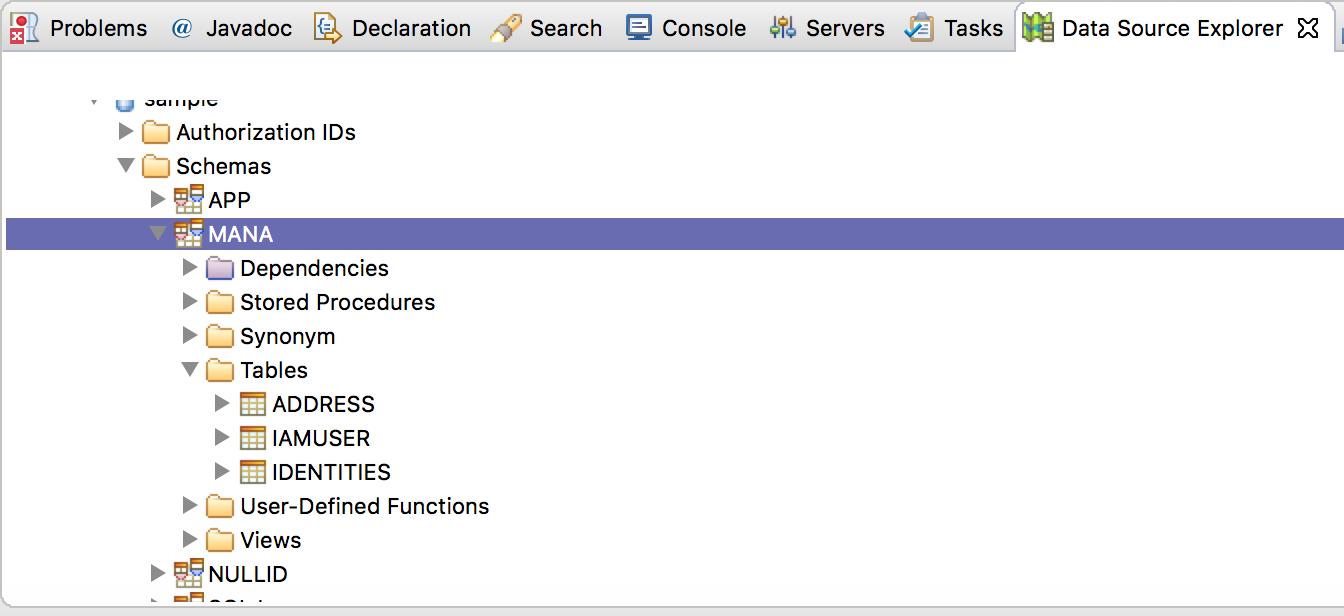
1. Right click the file and choose Run as Junit test



1. If everything goes good you should see a result like this:



1. This means your data base is created and ready to use. You can confirm this by looking on the data source explorer. Just remember to right click and refresh or even, disconnect and reconnect again.



If even after disconnecting and coming back you still not see the DB we recommend to try with a SQL Scrapbook and run the following command:

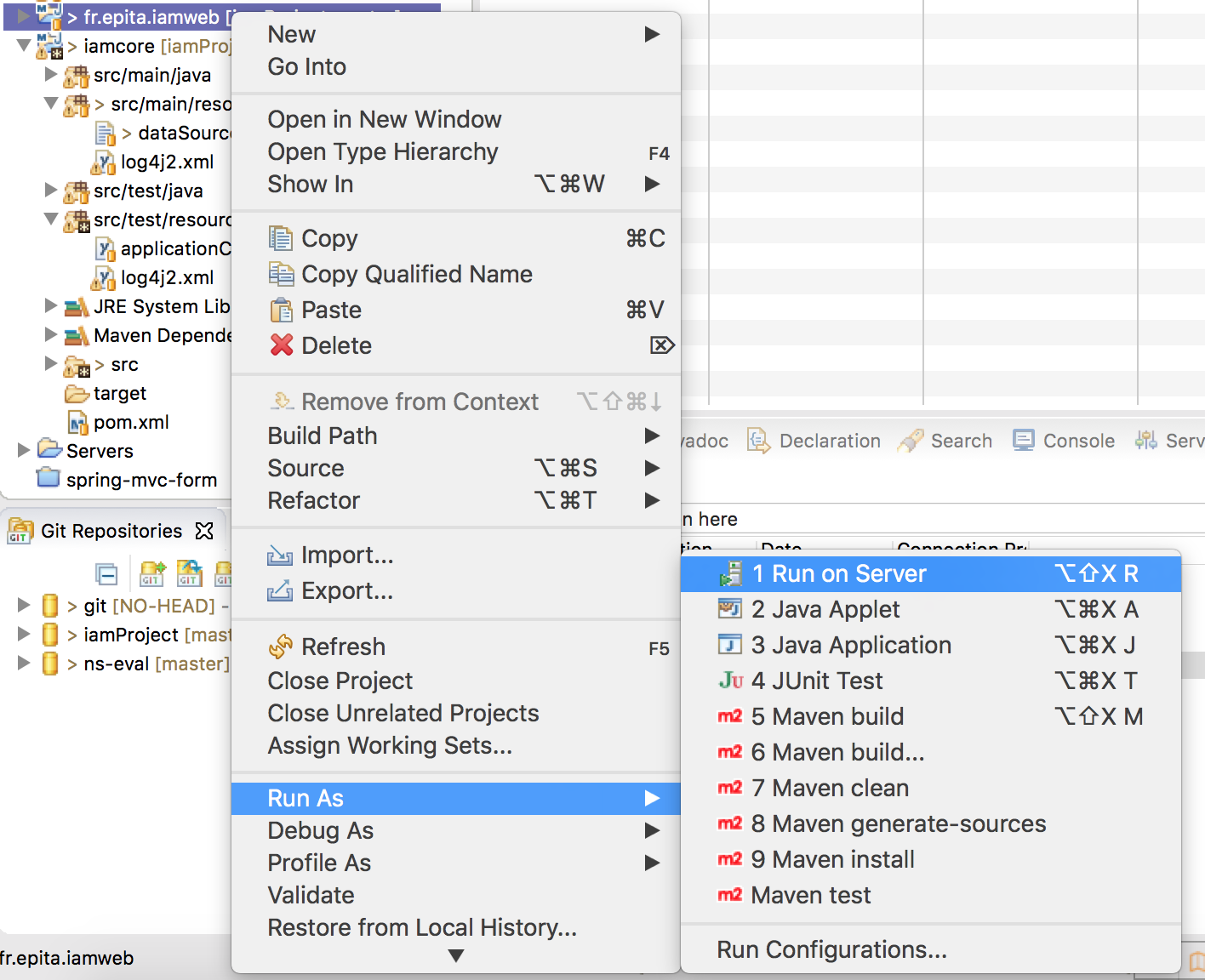
SELECT \* FROM MANA.Identities

If that yields no results please contact me.

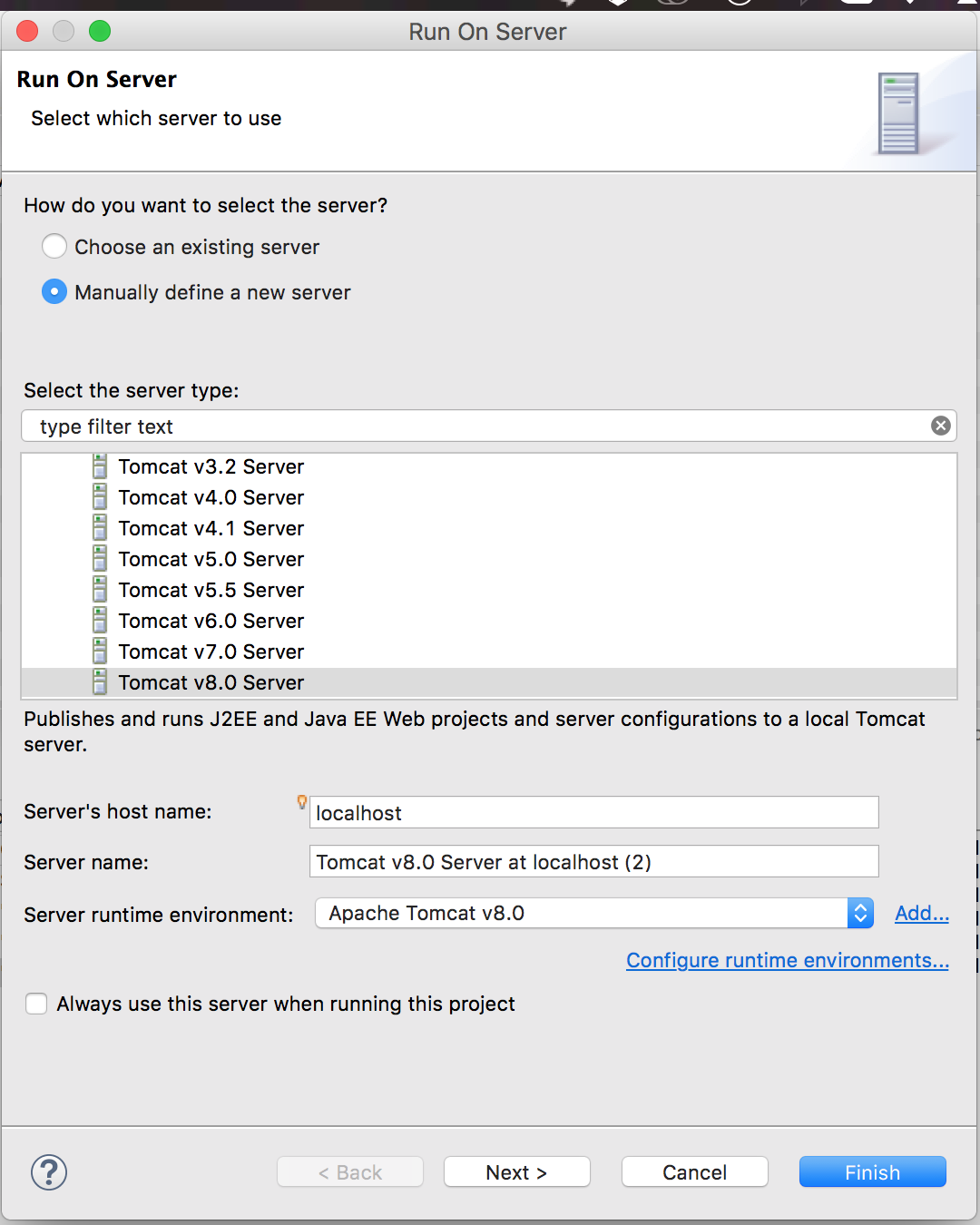
## Running the Web app

Now you should be able to run the IAM-web project as a web application.

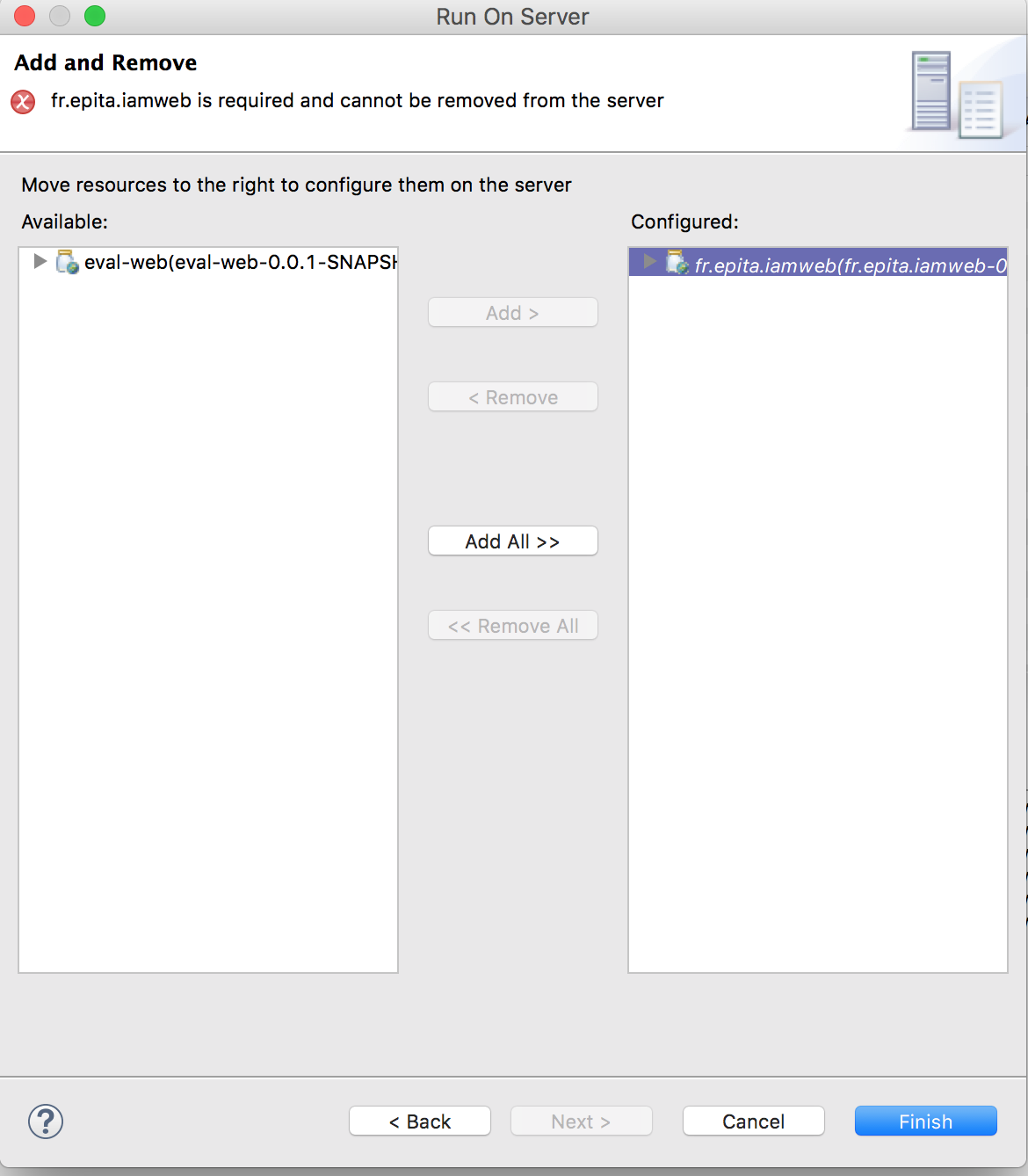
1. In the project explorer find the iamweb project and right click on it, selevt: Run as -> Run on server.



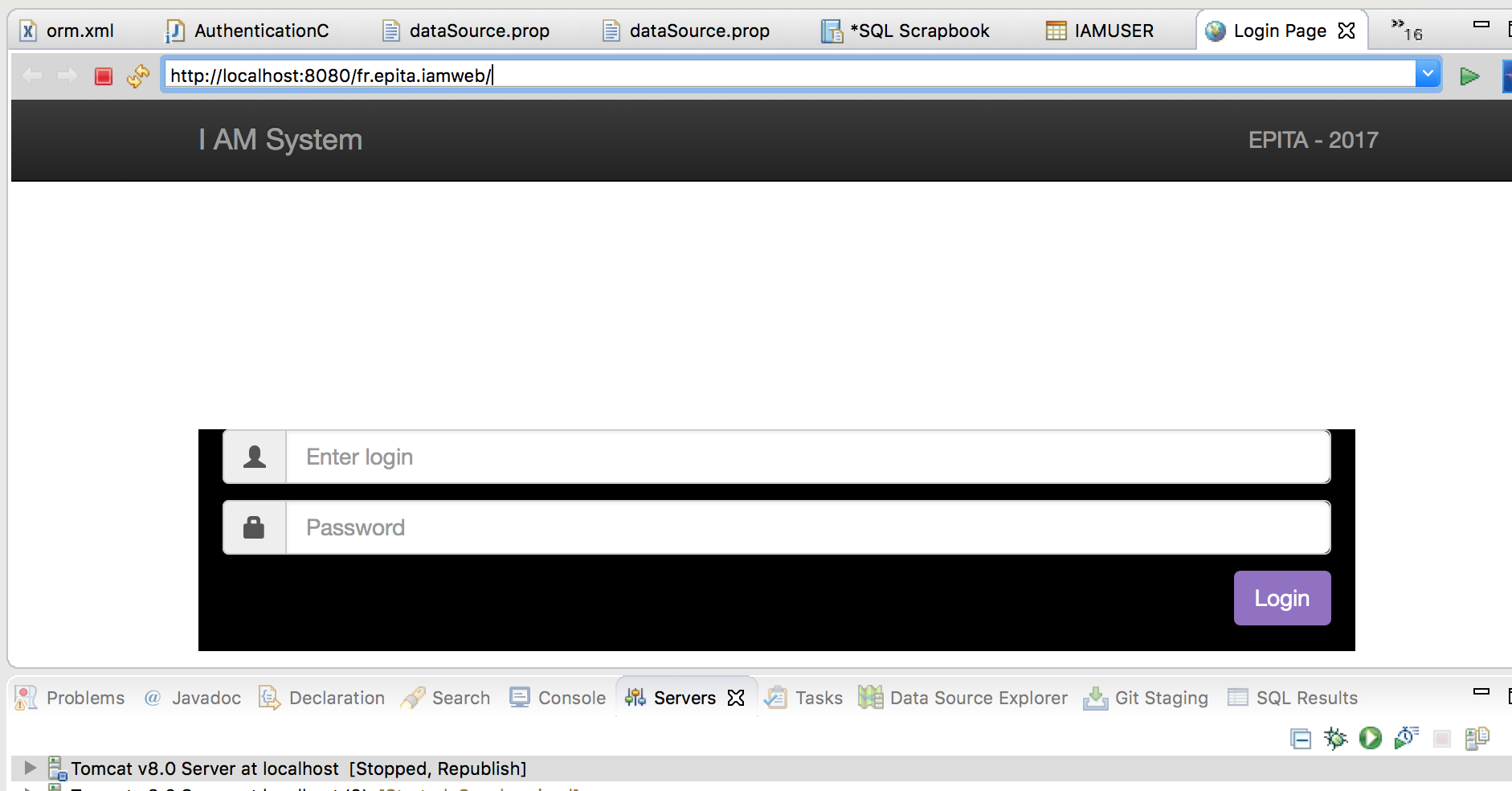
1. If you haven’t already used the Tomcat v.8.0 before, you should have to configure it now.



1. Make sure the iamweb project is under the configured list



1. Click finish.
2. It may take some time for the server to be configured and come up, but in the end eclipse should open a window like the following:



The default login and password created when running the test for DatabaseCreation is admin/admin.

For a better experience we recommend to explore the iam system on a browser such as Firefox , Google Chrome or Safari, not the eclipse integrated browser.