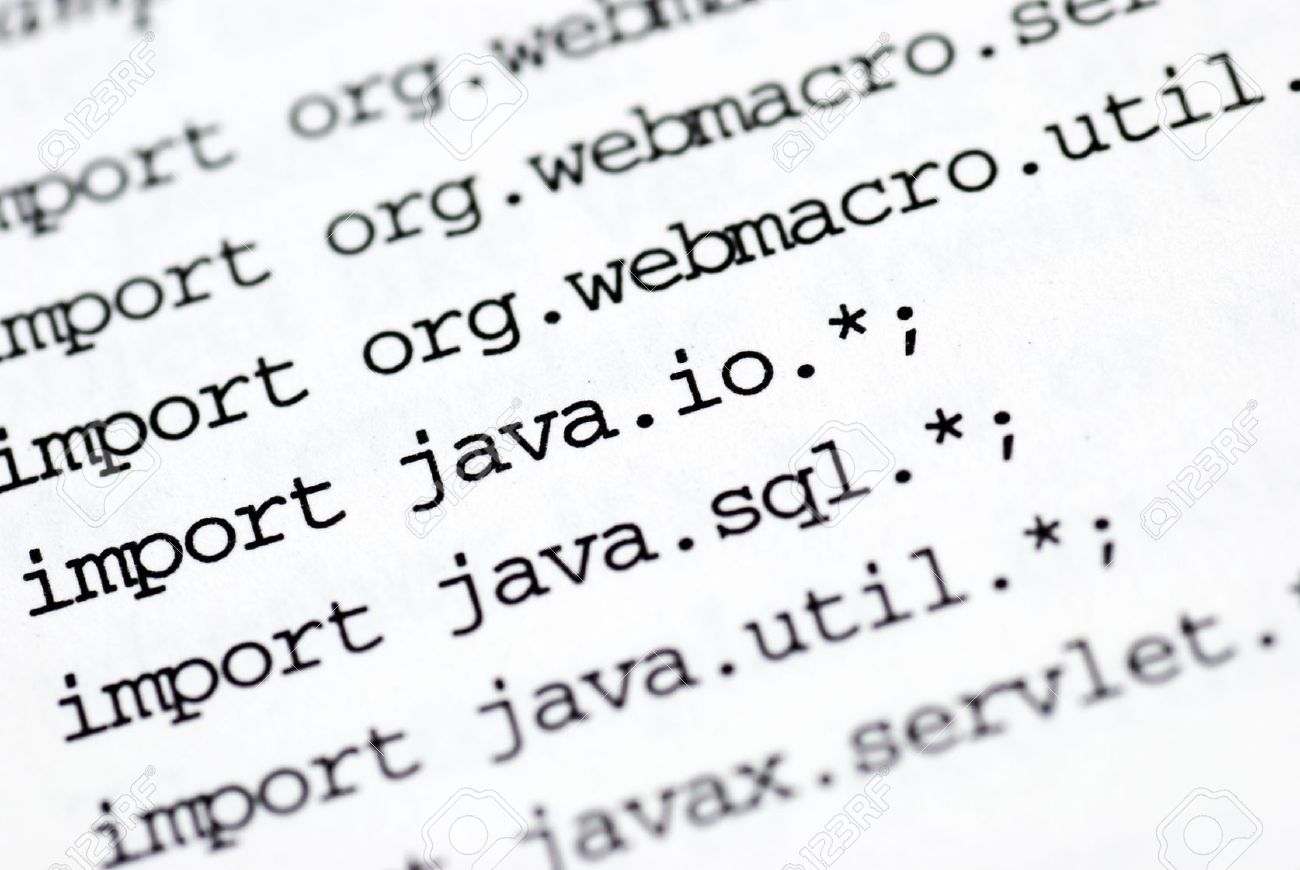
# **Advanced Java Project**



*Rohit Chauhan*

# IAM Application

Contents

[Subject description 4](#_Toc484621437)

[Subject analysis 4](#_Toc484621438)

[Major features 4](#_Toc484621439)

[Frameworks 4](#_Toc484621440)

[Application Feasibility 5](#_Toc484621441)

[Data description 5](#_Toc484621442)

[Expected results 6](#_Toc484621443)

[Scope of the application 6](#_Toc484621444)

[Conception 6](#_Toc484621445)

[Data structures 6](#_Toc484621446)

[Identities 7](#_Toc484621447)

[IAMUser 7](#_Toc484621448)

[Address 7](#_Toc484621449)

[Global application flow 7](#_Toc484621450)

[Create an identity 11](#_Toc484621451)

[Search Screen 12](#_Toc484621452)

[Identity update 12](#_Toc484621453)

[Delete an Identity 13](#_Toc484621454)

[Identity Details 14](#_Toc484621455)

[Success 15](#_Toc484621456)

[Global schema 15](#_Toc484621457)

[GUI description 16](#_Toc484621458)

[Login Page 16](#_Toc484621459)

[Welcome page 17](#_Toc484621460)

[Create Identity Page 17](#_Toc484621461)

[Search Page 18](#_Toc484621462)

[Identity Details Page 18](#_Toc484621463)

[Identity Edit Page 19](#_Toc484621464)

[Configuration instructions 19](#_Toc484621465)

[Prerequisites 19](#_Toc484621466)

[Get project from GitHub 19](#_Toc484621467)

[Importing Maven projects 20](#_Toc484621468)

[Database Connection Configuration 21](#_Toc484621469)

[Data base configuration and schema creation 24](#_Toc484621470)

[Running the Web app 27](#_Toc484621471)

# Project description

The Identity Management project is the project to be presented in the Advanced Java class by all the students as a part of their evaluation of this semester.

Identity Management software is a web as well as different other java framework (Spring, Maven, Hibernate) based application. The main goal is to manage users of an Information System. The project was started alongside the Java class for the purpose of learning and then was handed over to the students for taking it forward.

# Project analysis

## Main Features

The IAM Project has 4 major features.

* Authentication: A User should be authenticated before performing any actions or managing the Identities.
* Manage Identities: the user should be able to
  + Create,
  + Update,
  + Search,
  + Delete Identities from the application.

The Identities contains the identity data as well as the user data and the address is linked to each identities as application data. We use Derby as the Database engine for managing the entities.

### Frameworks

Maven

Apache Maven is a framework which helps manage the software project. It based on the concept of a project object model (POM). Maven can manage a project's build, reporting and documentation from a central piece of information. Maven helps handle jar dependencies easily through the pom.xml file.

Junit

It is used for the testing purpose. It has a lot of special notations and tags to handle the testing in a program. Some examples are @Before, @BeforeClass, @After and @AfterClass allows us to initialize values and terminate sessions or open database statements. This framework allows us to develop and test code at the same time and the best part is it supports or can easily be integrated with Maven.

Log4j2

A logging tool which helps us trace the control inside the code. It supports 5 levels of logs: error, warning, info, debug and trace. To use it we need to create the object of the LOGGER class and then this object is used at the moment of the logging. It also has a threshold that indicates from which level the logs are being printed.

Spring

A framework that allows us to configure and implement the IoC(Inversion if Control) in an easy way by providing us a list of predefined instance providers. There are different Annotations which are used to identify the instances. To use spring a configuration file has to be written, which will contain all the classes as beans which needs to be used in the program.

Hibernate

An ORM(Object Relational Mapping) framework which helps in data persistence as it applies to Relational Databases (via JDBC) . The most important part of hibernate is the SessionFactory, which helps us connect to the specific database engine using the database parameters which we have define. It also helps in the creation of the database instance and schema with the authentication level defined. Since the database engine is configured in a separate properties file it makes the database engine dynamic and hence increases the level of isolation as to which engine we are using, so it is easy to do changes.

## Application Feasibility

The application feasibility is defined at the early phases of the semester wherein we defined the entire structure of the IAM application. A downgraded version of the same software was presented in the Harmonization semester which lead to the feasibility report of the project being positive, hence the need came to enhance the already exisiting project using latest frameworks and both the levels of java i.e J2SE and J2EE.

This application not only helps us to understand the nitty gritty of the different frameworks and their integration but also serves the purpose of managing the identity that allow us to clearly see the process of managing entities.

## Data description

Under the Data description, there are two different types of entities which forms the major crust of the application and which needs to be mentioned: Identities and Address.

The project requirements as defined at the early phase stated the need of Identities which encompasses the user and login details, and the Address which can link to all the identity records.

Both, Address and Identities are a part of relational database.

The tables for these entities are automatically generated with the help of Hibernate.

## Expected results

There is the UI login interface which helps the user to interact with the application.

The user using the application should be able to perform all the CRUD operations successfully. The main goal of the application is to be fully functional and to fulfil the basic needs of the user which makes it complaint with the defined specifications.

## Scope of the application

As stated above, the application is limited to manage the Identities and the Address. This is the no doubt the basic version of the application. There would be more exciting and additional features in the future versions.

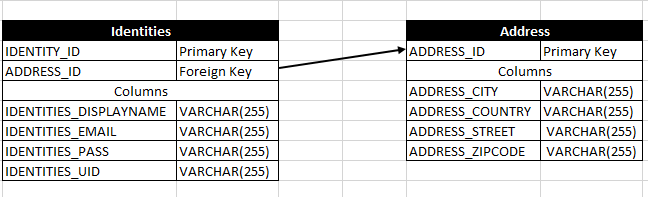
The Identities are linked to the database and has no way to select a file as storage mode for Identities. Hence regretfully the user must have a Database engine software on his computer/laptop.

The user management is not a part of the scope for this application and can be taken in the next version of the software a change request has already been passed for the same.

# Conception

## Data structures

As mentioned Hibernate is the framework which manages the data model - database relationship. Hence, we have defined a data model package where all our tables are described as classes, once the application builds Hibernate with the help of the predefined tags found in the data model classes, creates the respective tables in the database. Here is an entity relationship diagram of our three tables.



### Identities

Identities are the entity which will be managed precisely, 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:

* ID: It’s the user id a unique id auto-generated and auto-incremented as a Primary Key in the Identity table.
* UID: User Identification, it can be the value of any of his identification cards (for students it could be the roll number).
* Display Name: Represented as a string. It’s the name of the user,
* Email: represented as a string. It’s the email address of the user.
* PASS: represented as a string. It’s the login password for the user.

Users are not supposed to be managed by the application. A user will have an email id which will be his/her login and a password, both are stored as string.

### Address

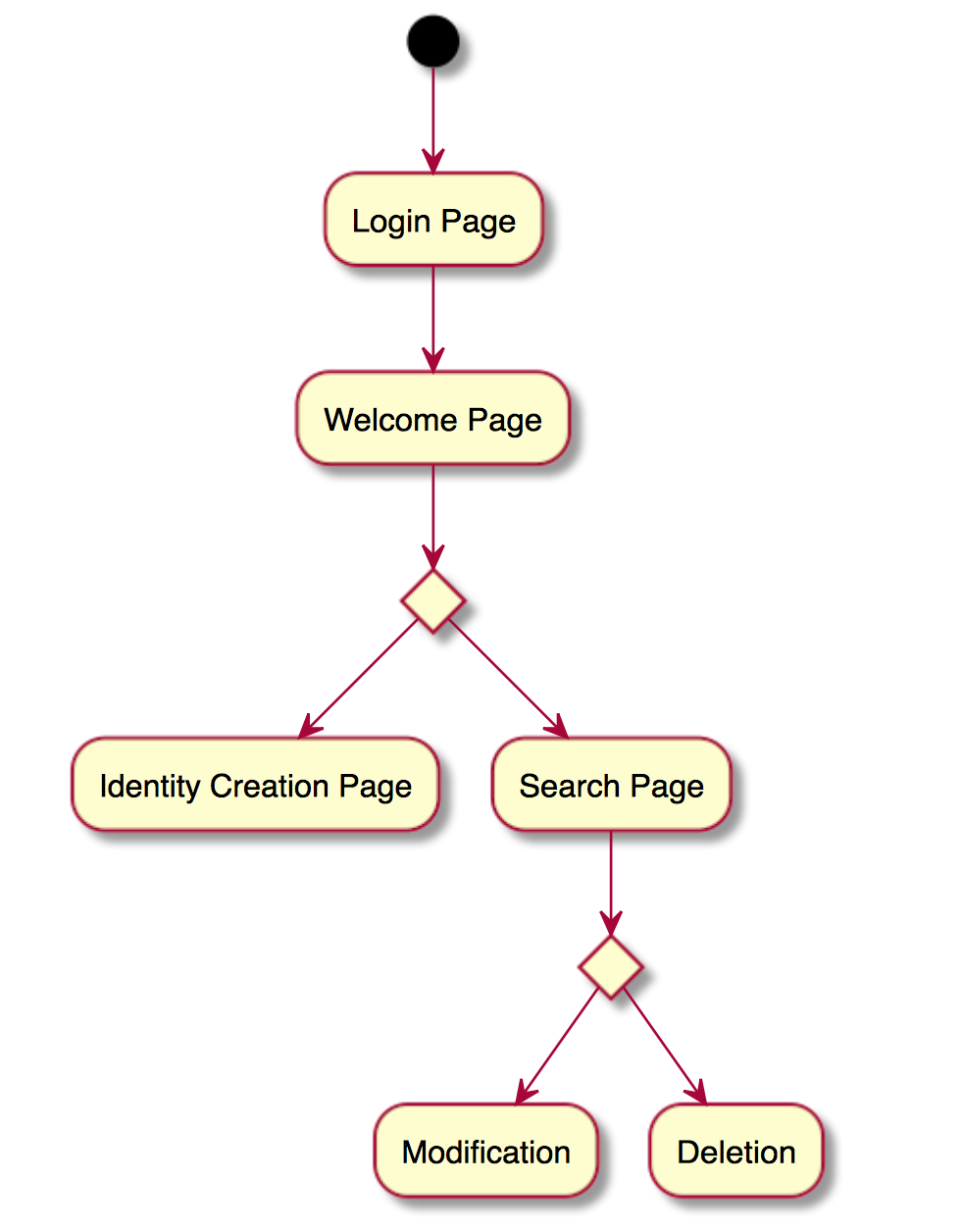
Address table has its own id as primary key and as with the user table, we have a foreign key to identity table, however, in this case the relations ship is one to one or none.

* ID: It’s the address id a unique id auto-generated and auto-incremented as a Primary Key in the Address table.
* Street: Represented as a string. It’s the street where the user resides.
* City: Represented as a string. It’s the city where the user resides.
* Country: represented as a string. It’s the country where the user resides.
* Zip code: represented as a string. Due to the mention of alphabets in the zip code in some countries around the world

## 

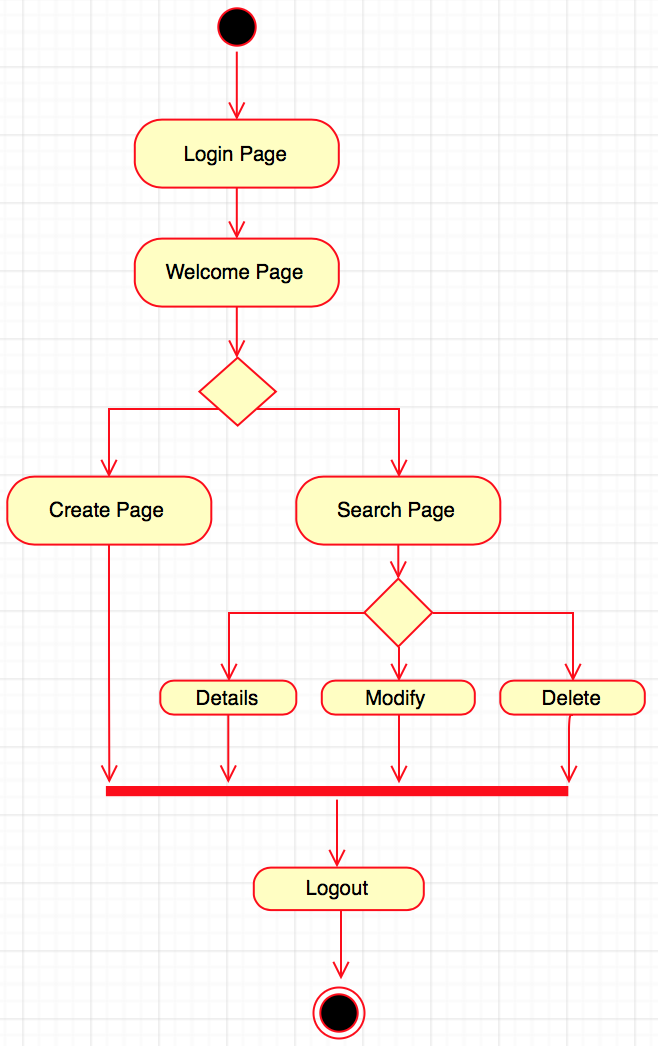
## Global application flow

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



The implementation was carried out with respect to this structure, though a new screen was added at the Modification / Deletion level, the new screen is a Search and Details screen that will allows us to see the addresses a user may have.

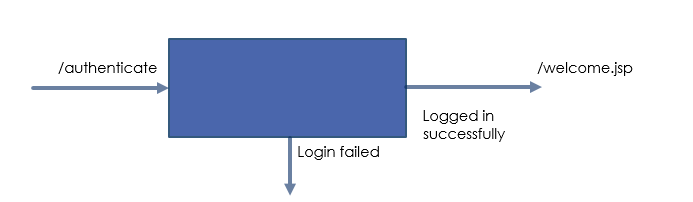
Also, a logout functionality was added on the search screen. The following is the modified application flow.



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

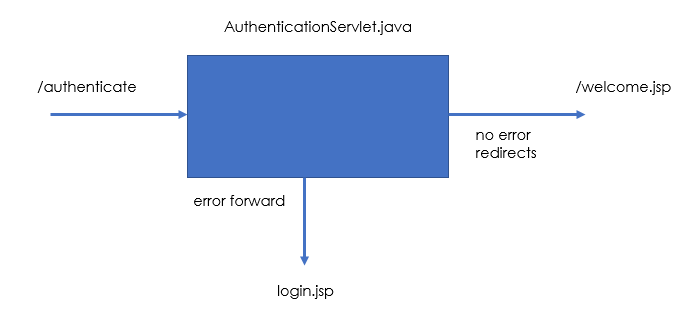


The following diagram illustrates how the servlet implementation was done for the rendering of the login screen as entry point of our application.



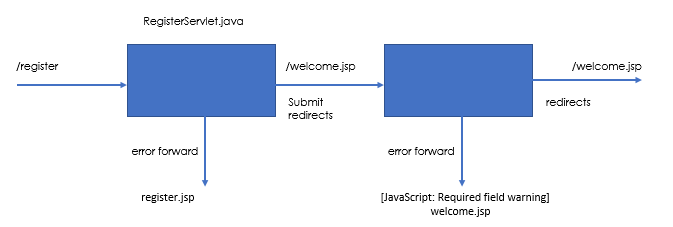
In order to determine if a user has logged we are storing the sessions of the user, if we can retrieve it from the session it proves the user logged in previously and we redirect him directly to the welcome page.

When the user is on the login.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 with those stored in the database, in case of failure the user would stay on the login.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.

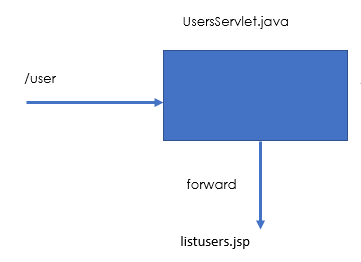


The first step is to call the register JSP file which will create the register form on the screen. Once it is loaded and we have entered the date, then we submit the form, we can either have an error on the form, or encounter an error while saving the form which will forward us to different pages. If everything goes well, we will be forwarded to a welcome JSP that, welcomes the user with the username he/she choose while registering.

### Search Screen

The search screen is the central part of the application since from it we can perform the different activities like 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 won’t do a search as a check has been placed on it. Otherwise, according to the name (partial or full) it will filter the results and display it.

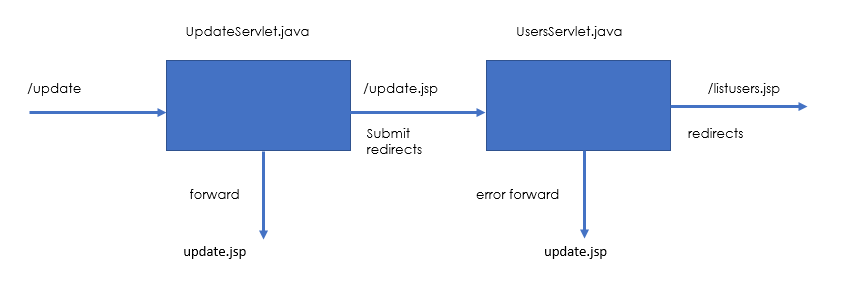
The servlet architecture is presented on the following image.



### Identity update

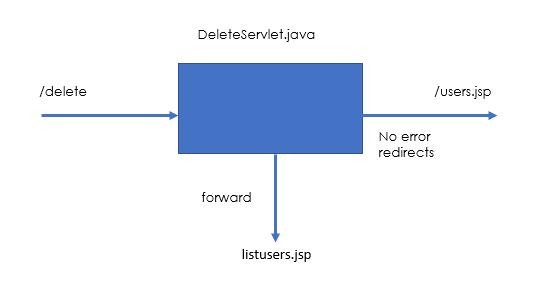
The update identity process involves two servlet classes. One to look for the selected record and pass it to the JSP page which in turn will display all the records and the user will be asked to updated the records as per his/her choice, and a second one to process the submission of the form. The diagram bellow illustrates this process.

The /update has as parameter the Id of the identity we are going to edit. so, the UpdateServlet.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

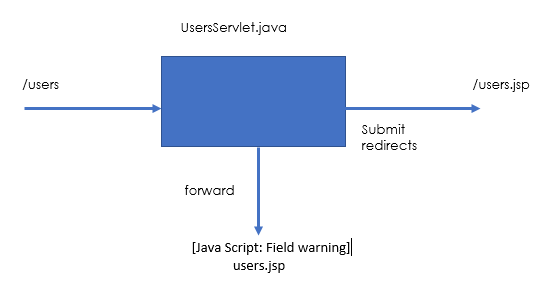
Delete Identity will be a straight forward process compared to create or update, mainly because we do not need to pre-load a form screen in the middle of the process. This process again, has as parameter the Id from the Identities and address\_id from the Address table to be deleted. The servlet is in charge to look for the identity on the database and delete it using the DeleteServlet service. Due to the foreign key constraint, the record in the Identities table will be deleted afterwards the record in the Address table will be deleted If the operation succeeds we will be redirected to the SearchServlet with the success message, else an error will be displayed.



### Identity Details

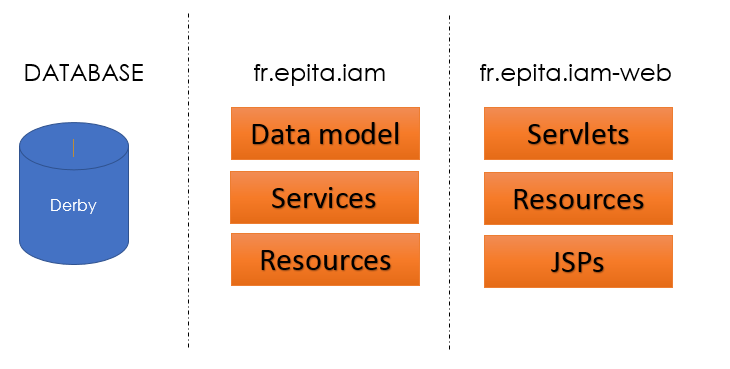
Additional features- The presence of an address object with the identities lead to the development of the search page which will display address as well as identity properties to the user.

Unlike the update and delete cases, we pass the name parameter (partial or full) to the servlet to fetch one or multiple results. The servlet will then retrieve it from the database and load the appropriate details so they can be retrieved by the jsp view.



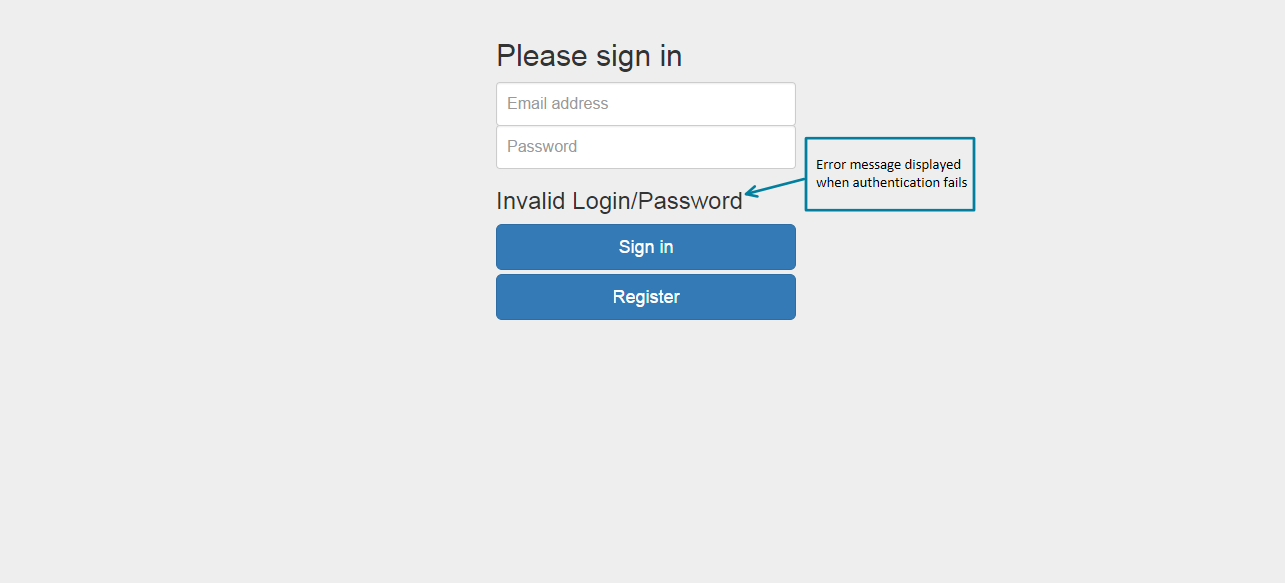
## Global schema

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.

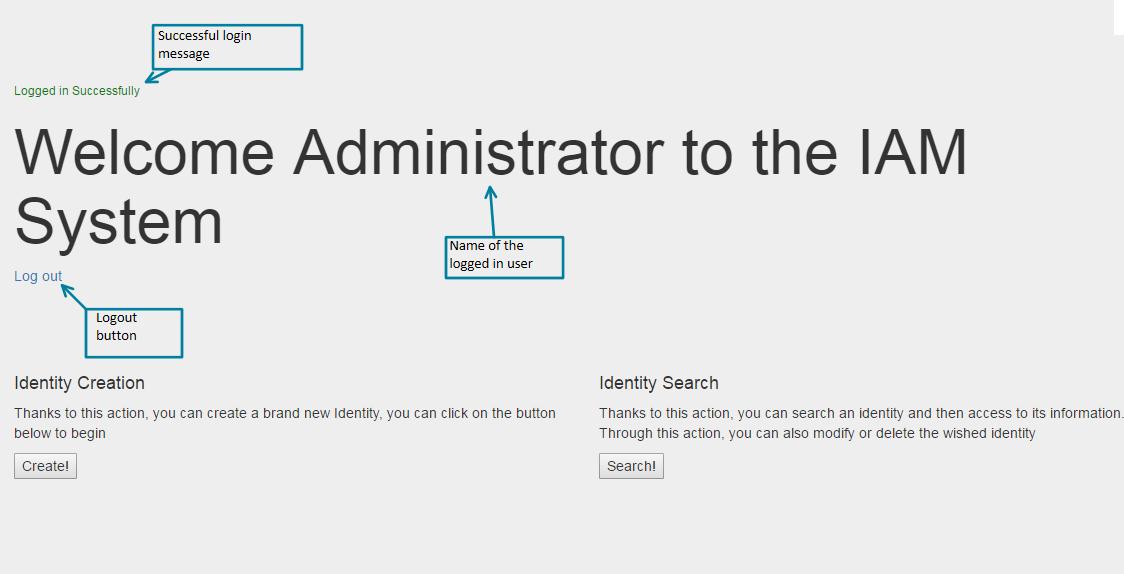


# GUI description

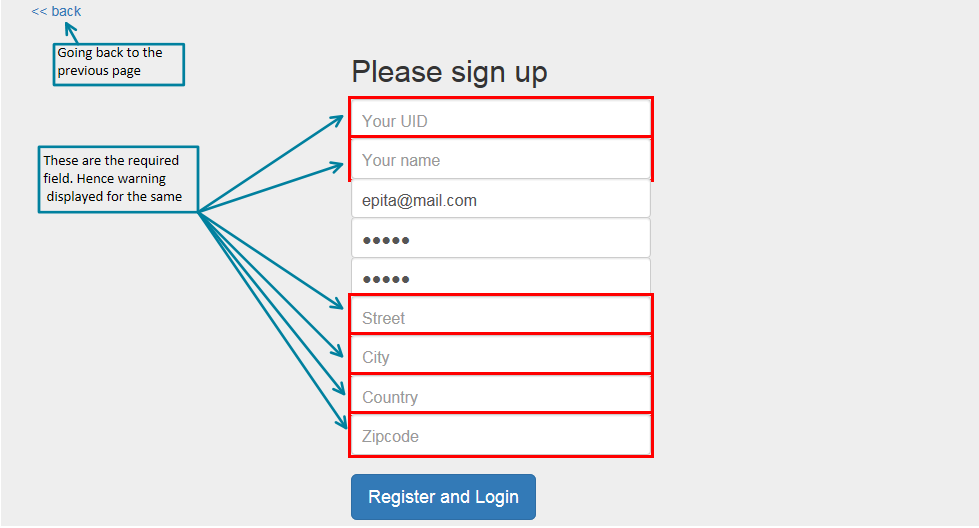
## Login Page



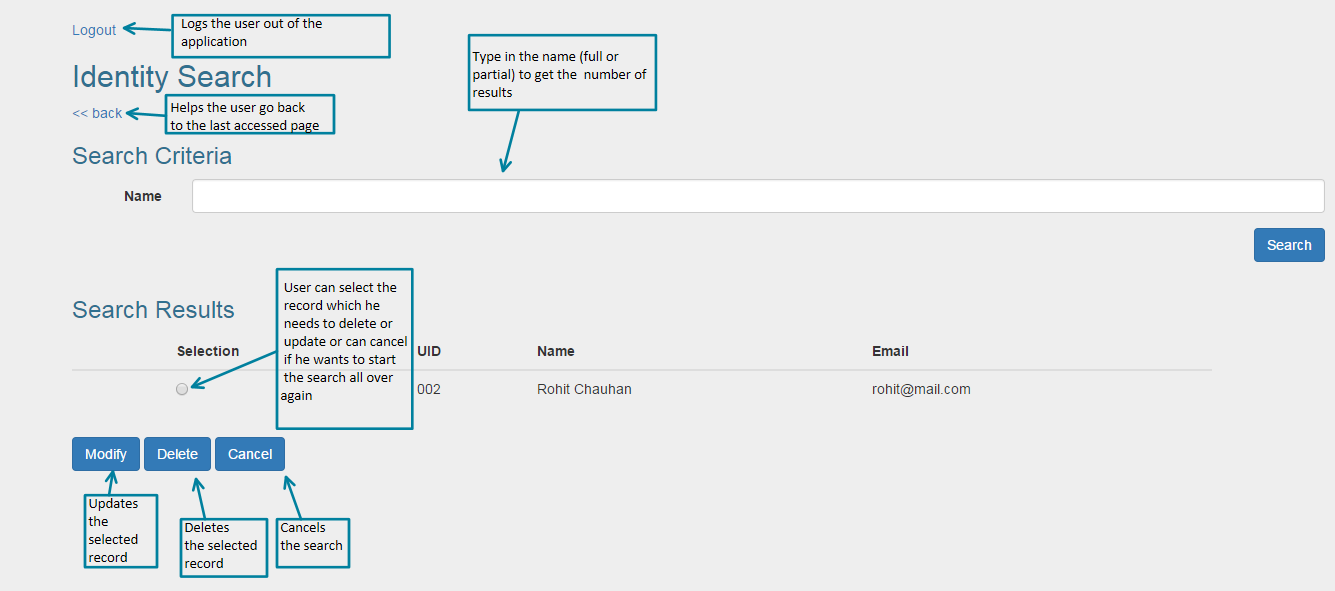
## Welcome page



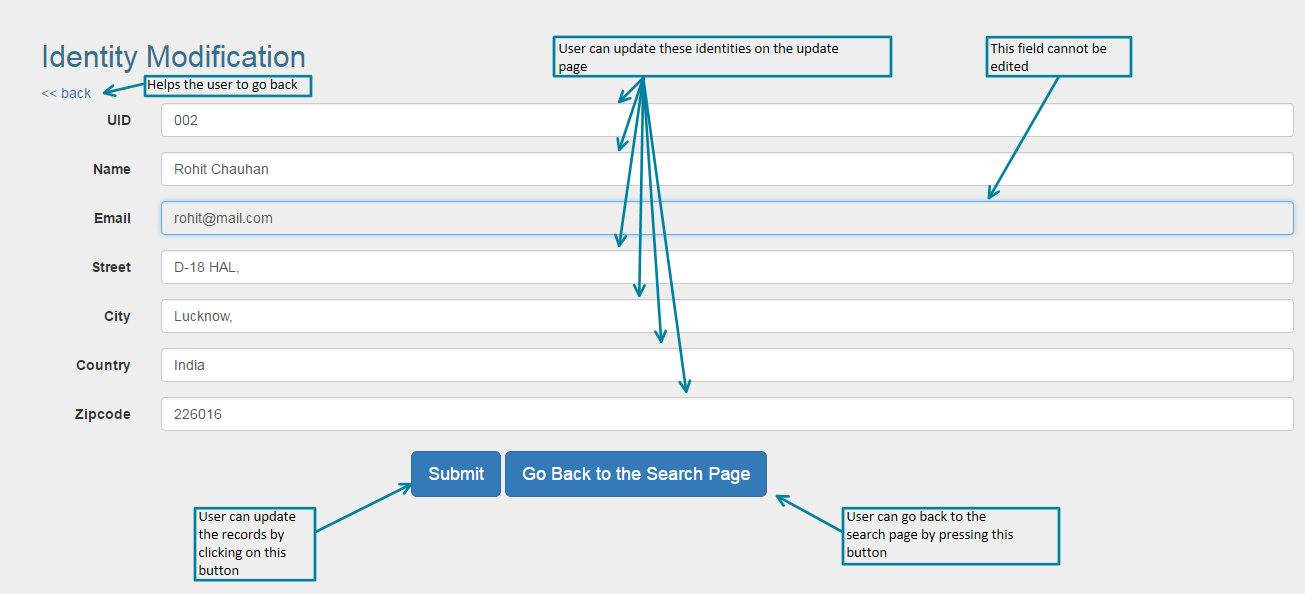
## Create Identity Page



## Search Page



## Identity Details and 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

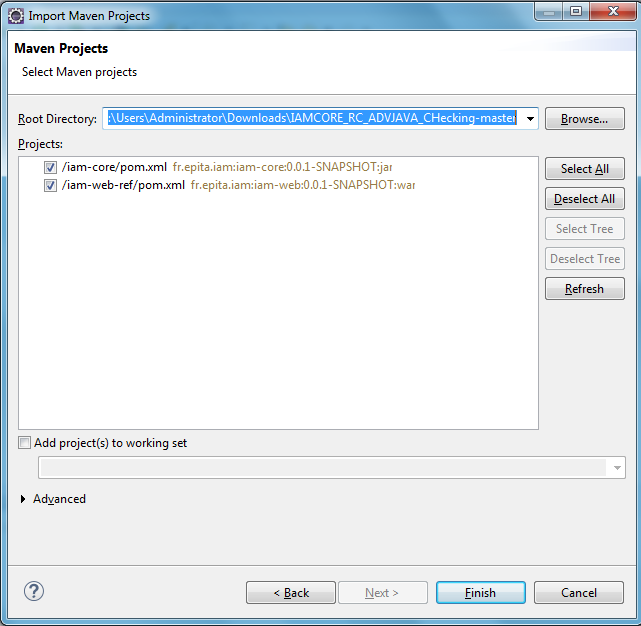
1. Open Eclipse, and go: File -> Import
2. Select Git -> Projects from Git click Next
3. Select Clone URL click Next
4. Paste the following URL: https://github.com/rohit19j91/IAMCORE\_RC\_ADVJAVA.git click Next
5. Make sure master branch is selected and click Next.
6. Select the destination Directory 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. 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.

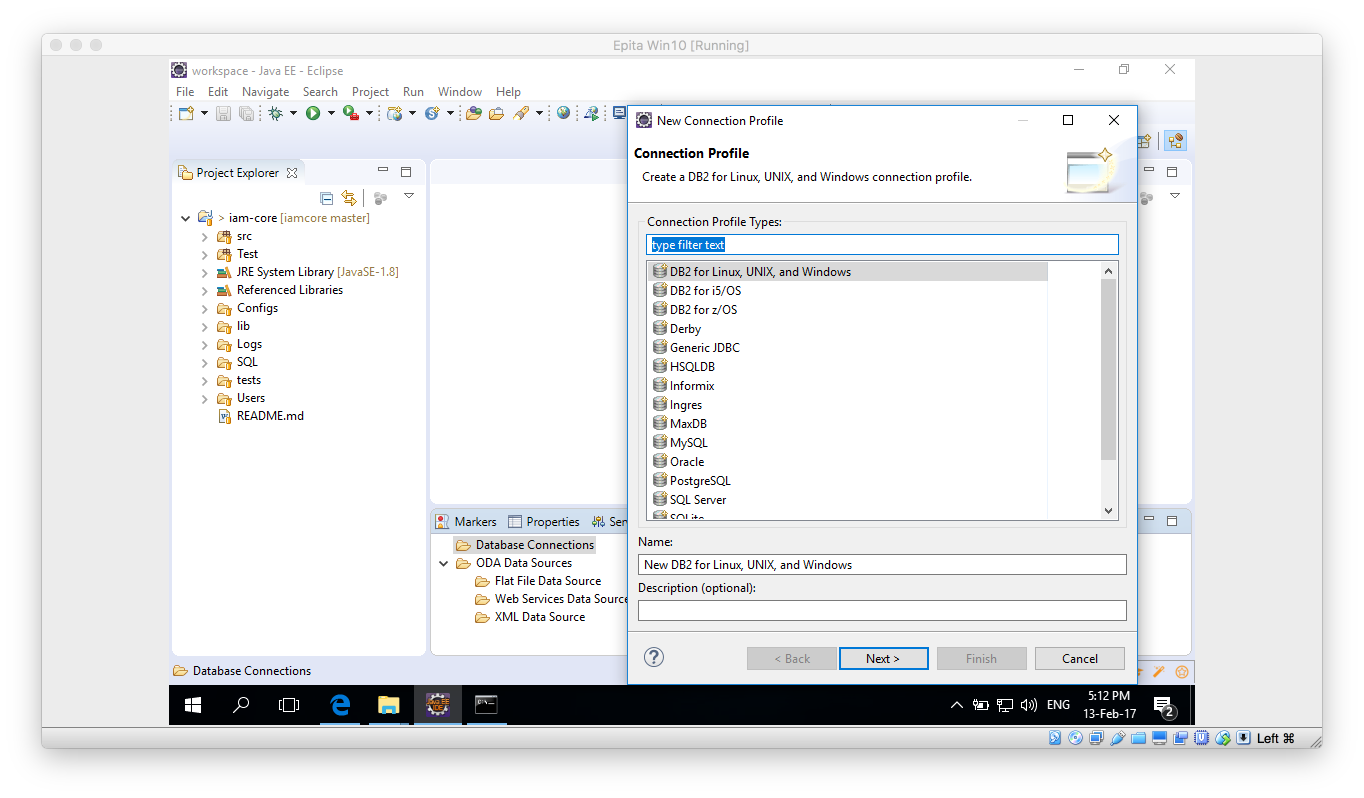
1. The result should look like this:



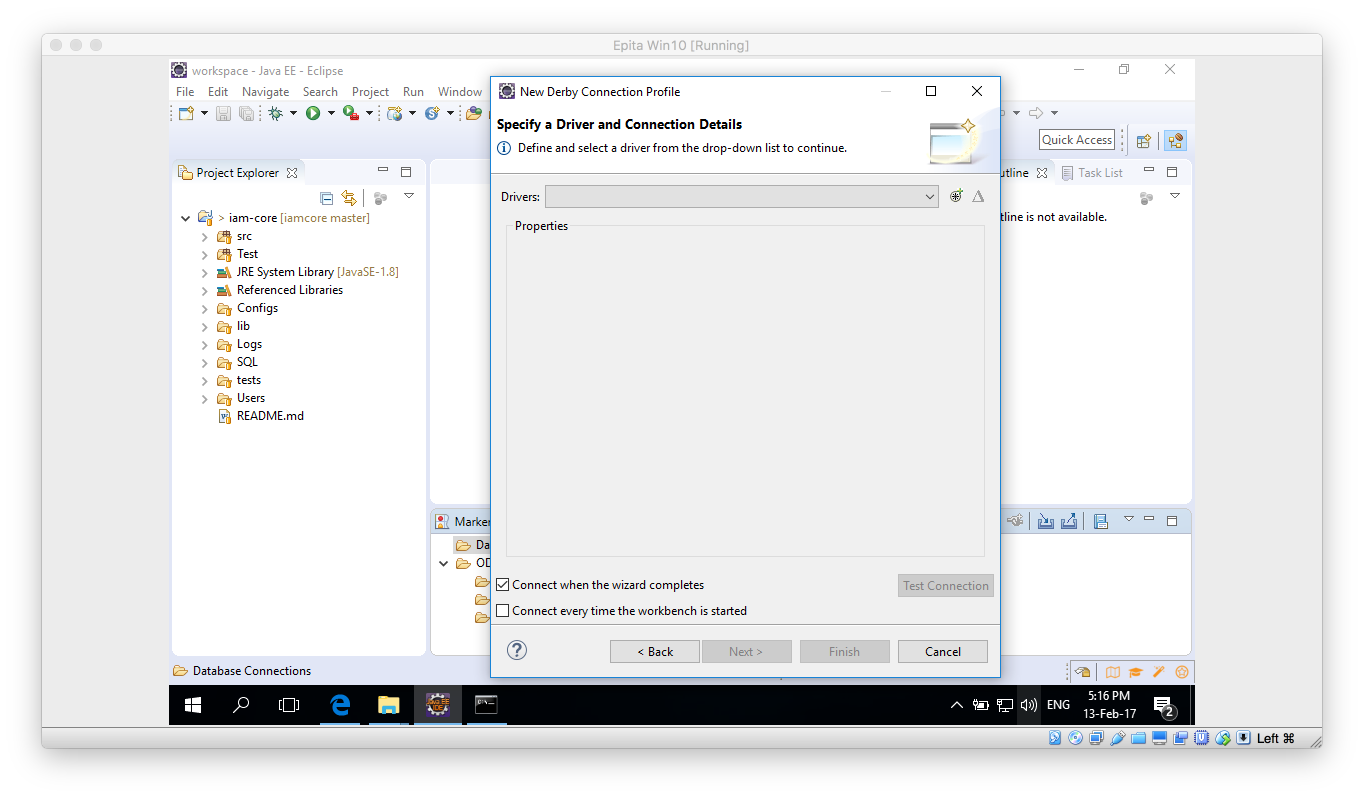
1. Click Finish.
2. Right Click on each project and go to Maven > Update Projects... It helps in updating the dependencies and IoCs.

## Database Connection Configuration

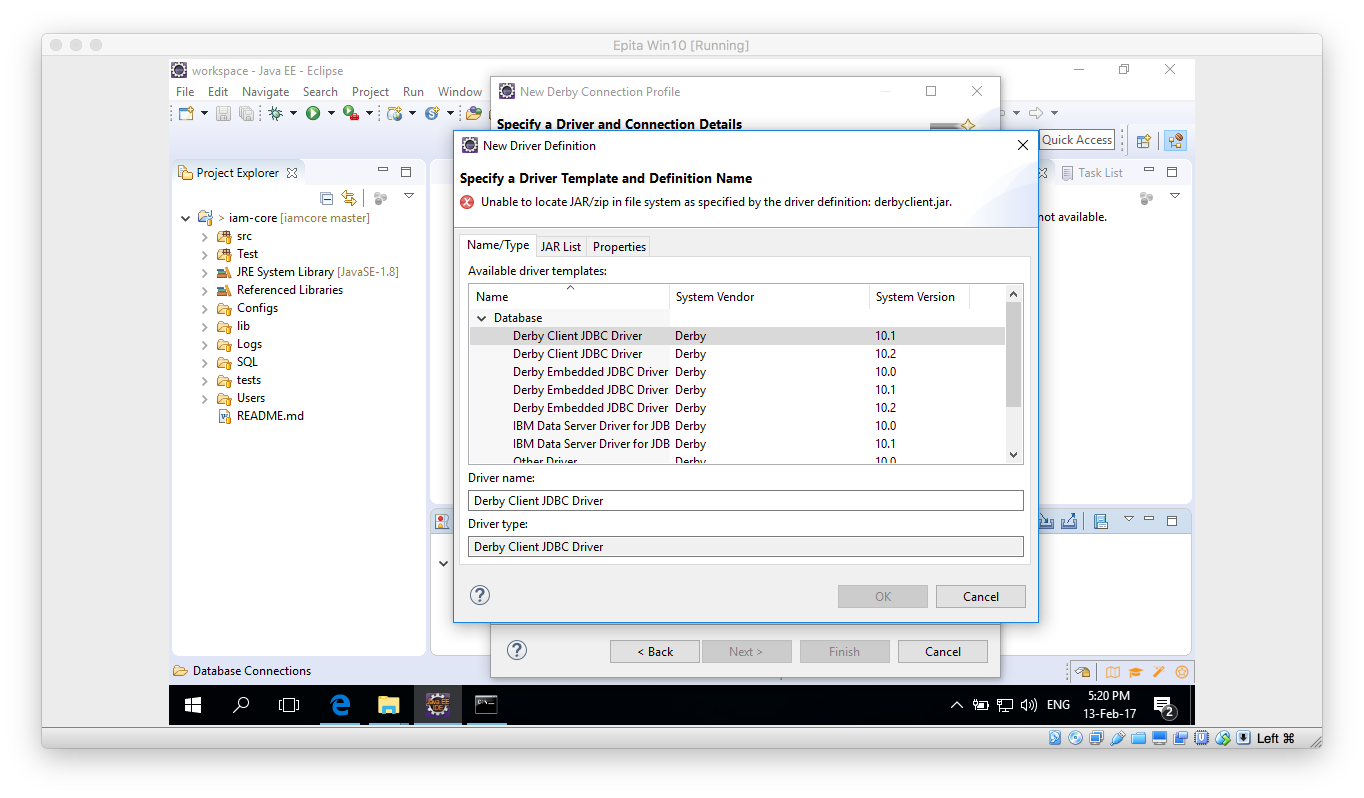
1. Start your Derby engine running the startNetworkServer file.
2. Open the Data Source Explorer window on Eclipse.
3. Right Click Database connections, select New.



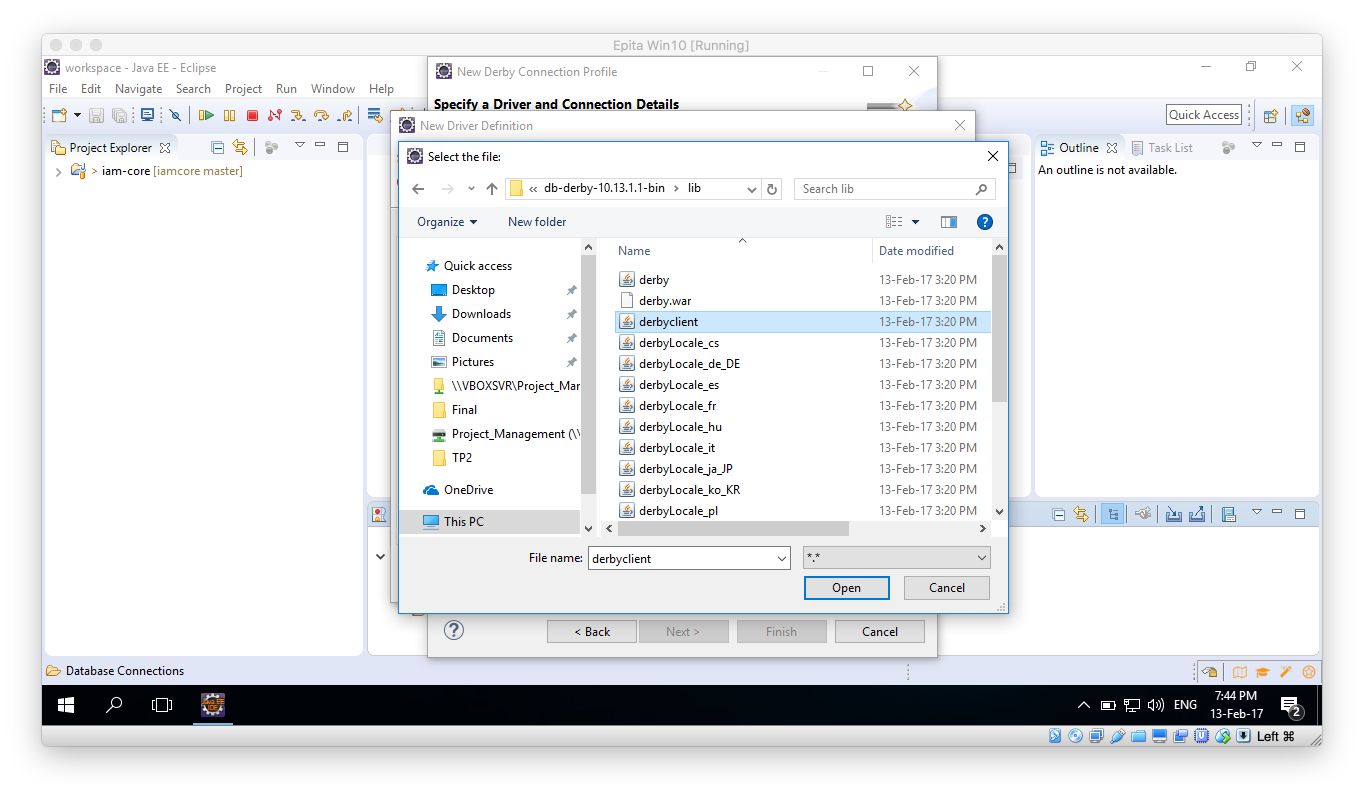
1. In the contextual window select Derby. Click Next.
2. In the new window click on the “New Driver definition button”



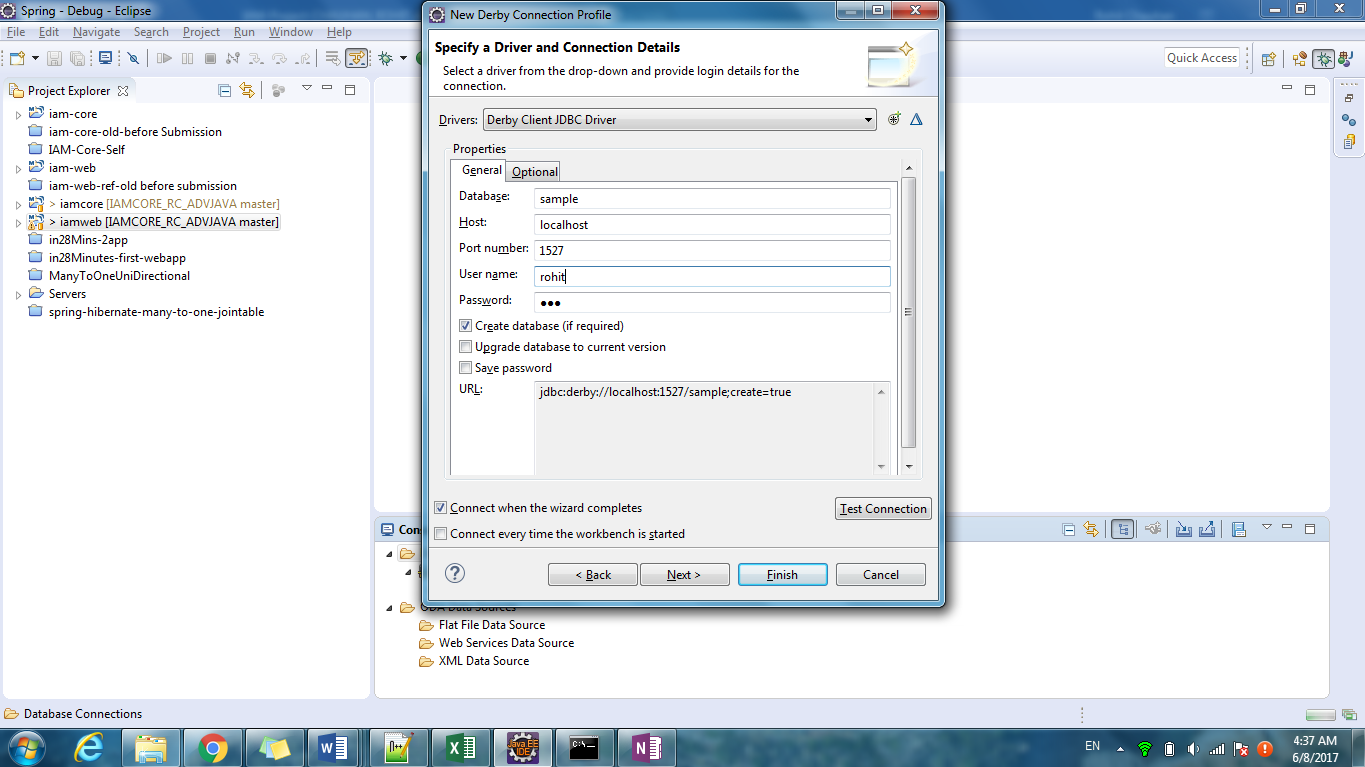
1. Select any Definiton on the Name/Type tab and then move to the JAR List tab.



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.



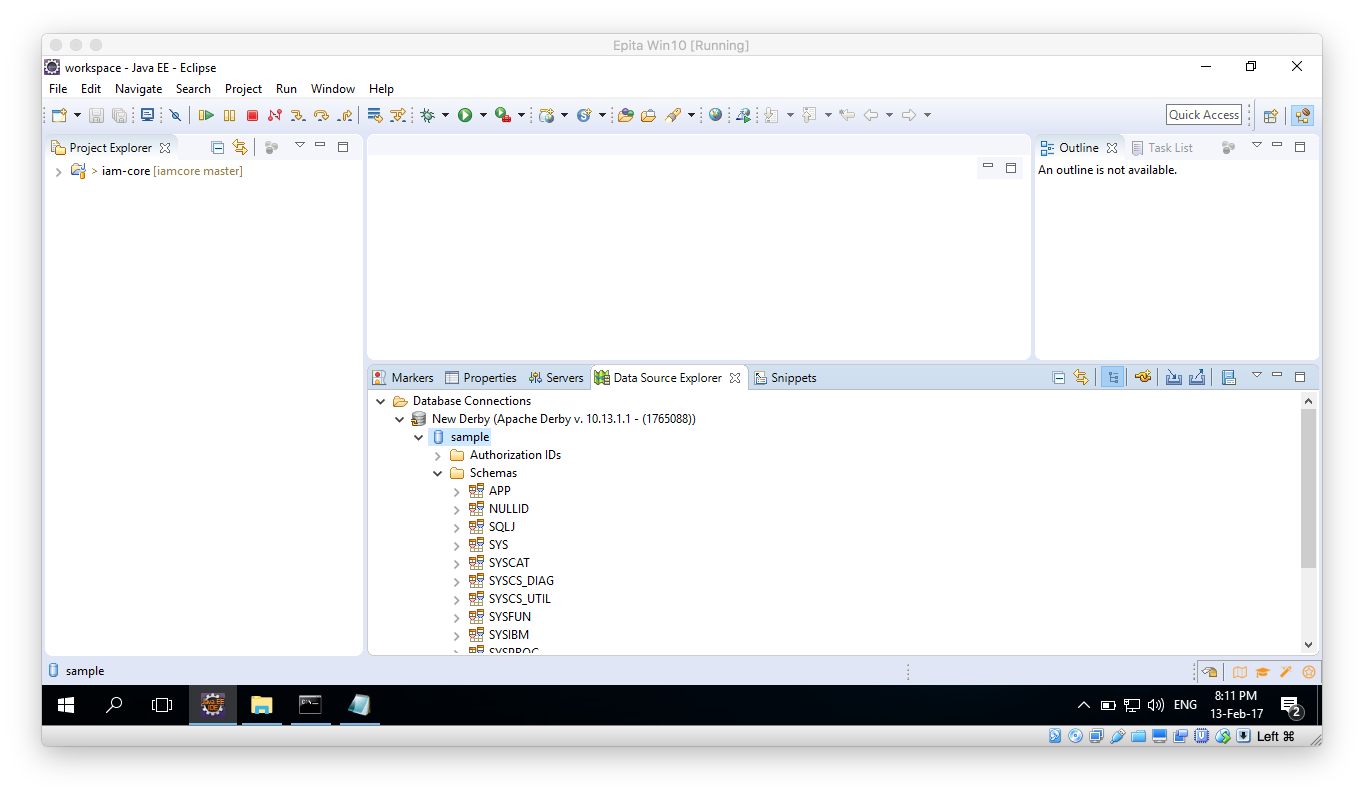
1. Click Open, then the “New Derby Connection Profile” window should be updated. Please take note of the URL and change the user and password if desired. Remember those because we are going to need them later on.



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

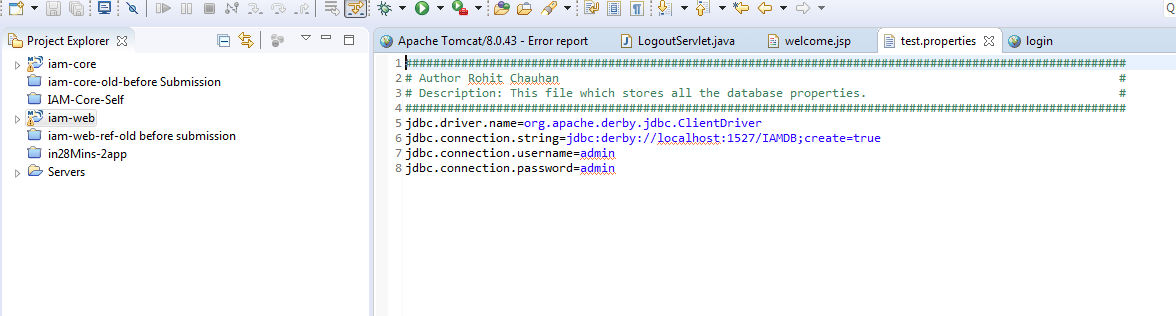
## Data base configuration and schema creation

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 test.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/IAMDB;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 Scrapbook window and execute.
2. Right click anywhere on the Scrapbook window and select “Execute All”.
3. Execution should be successful:

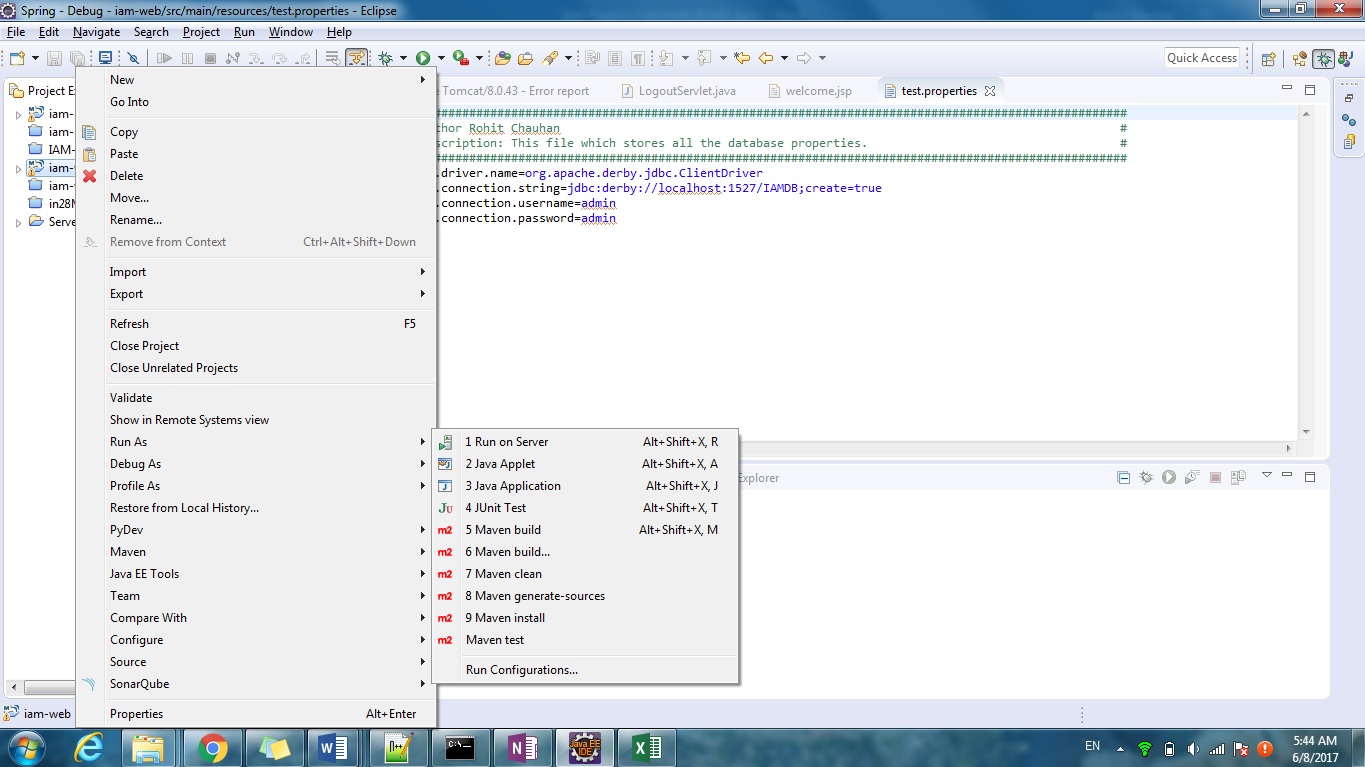


1. Save this file and close it. Now, do the same changes to the file found on the iamweb project under: src/main/resources save it and close it.
2. The Database setup is all set and ready to go.

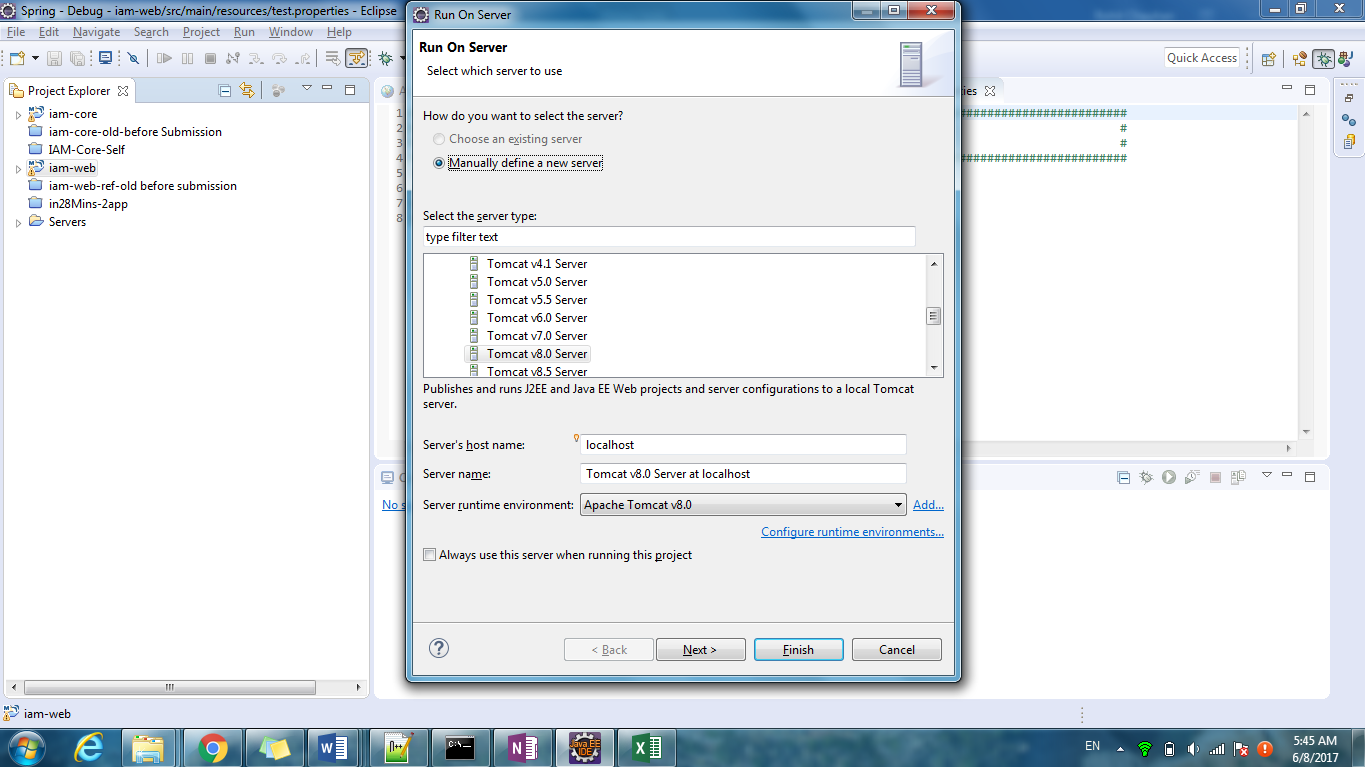
## Running the Web app

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

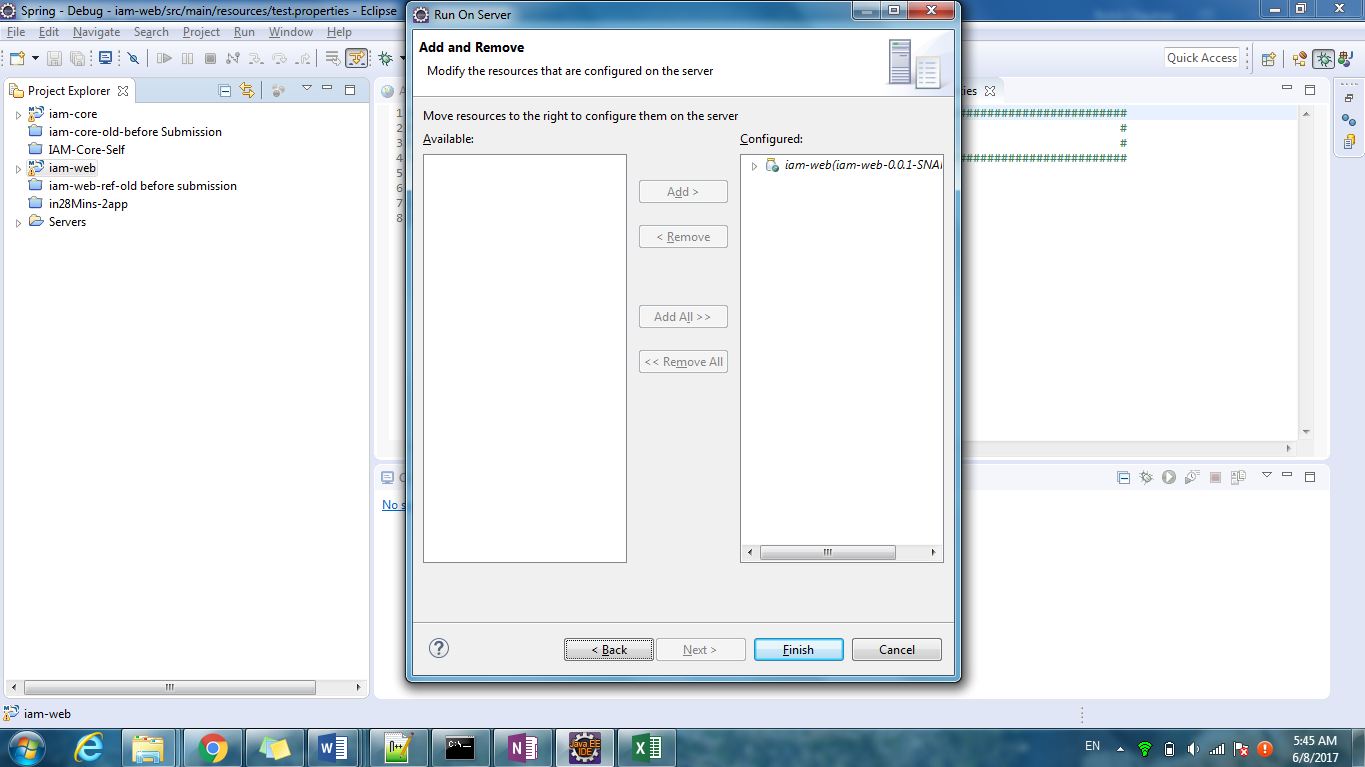
1. In the project explorer find the iamweb project and right click on it, selevt: Run as -> Run on server.



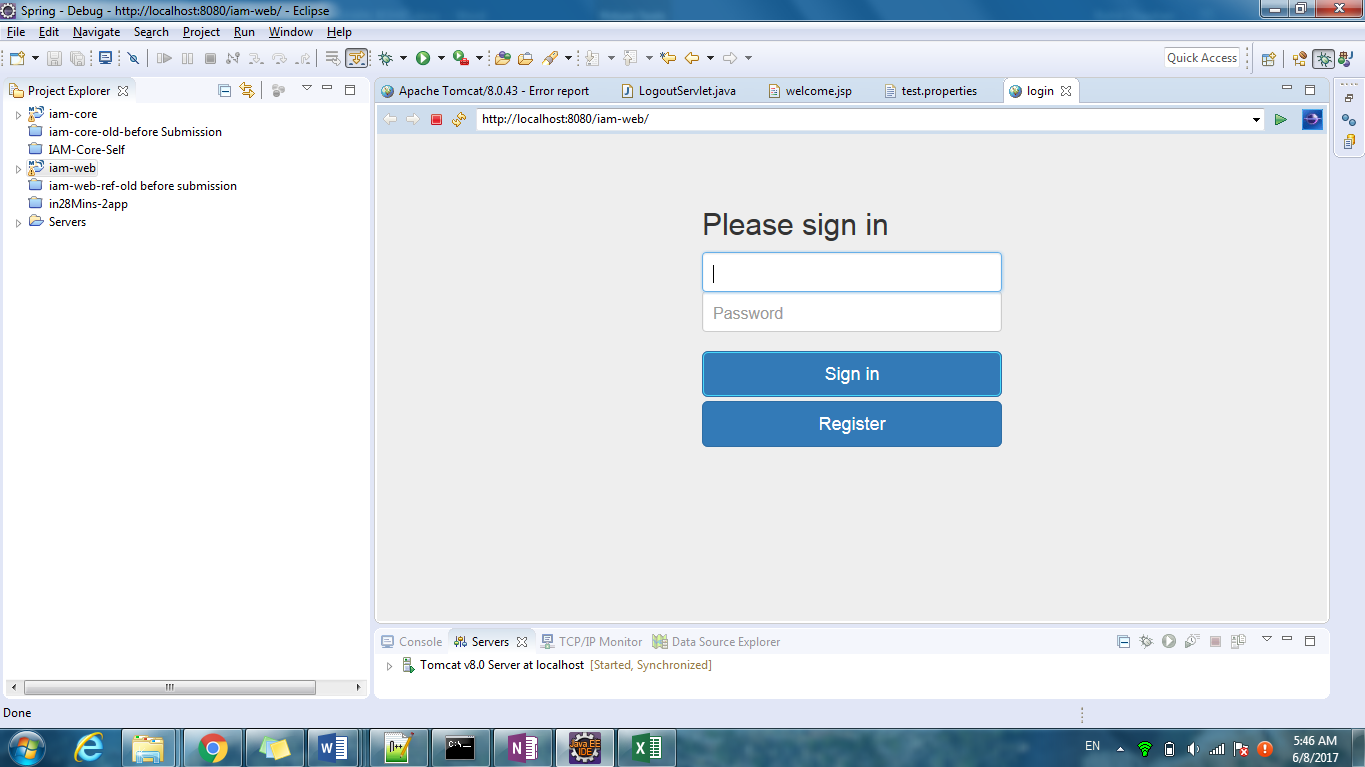
1. If you haven’t already used the Tomcat v.8.0 before, you should have to configure it now.



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 Database Creation is admin/admin.

For a better experience, it is recommended to explore the IAM application on a browser such as Firefox, Google Chrome or Safari, not the eclipse integrated browser.