

Technical Specifications

User Identity Management

Fundamental – Spring Semester 2018

Sandeep Shivajirao

July 2018

**Java Project with Derby DB**

* 1. **Purpose of this document**

This documents provide a brief description of all the technical aspects which were involved in the entire development phase of this project.

* 1. **Use of this document**

This document can be used by technical associates to understand, design and enhance this project.

* 1. **Overview**

The document helps in understanding the general coding thought-process applied for the development along with the

**1 INTRODUCTION**

**1.1 Purpose**

The purpose of this project is to use the Identity Management Tool to store and manage the data of the users on a DB and have restrictions to the users to view the data.

**1.2 Scope**

Eclipse - Oxygen.3a Release (4.7.3a)

Derby DB – 10.14.2.0

Java SE Development Kit 8

Java SE Runtime Environment

**1.3 Definitions, Acronyms and Abbreviations**

UIM – User Identity Management

DAO – Data Access Object

**1.4 Overview**

The Identity Management tool uses the database to store the data of users and manage them accordingly. The Derby DB is used for storing and managing the data. The user who needs to access the data should login to the User Identity Management tool (via Eclipse). The CRUD operation can be performed using the User Identity Management tool.

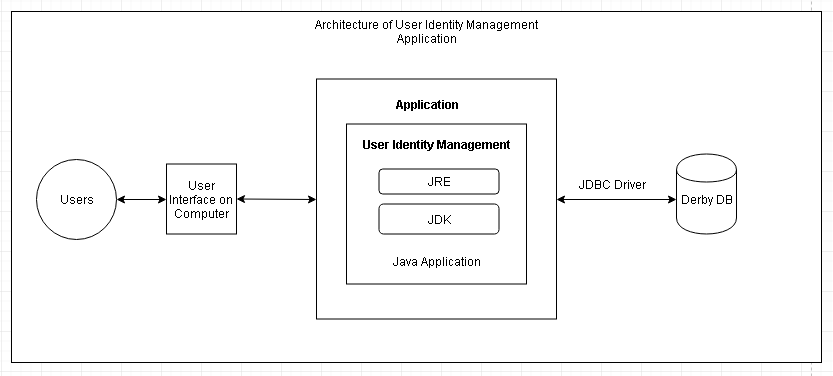
**2 SYSTEM OVERVIEW**

The User Identity Management systems is mainly used to collect the data of user and store the data in a database or file system. The benefits of using such an architecture is to have reduce costs and increase efficiency in storing data. Traditional storage systems such as excel or other traditional tools would not have high accuracy and be fast in all features of CRUD operation.

**2.1 System Characteristics**

The Eclipse console is used as a platform to display and accept inputs from the user. The user can perform CRUD & search operation on the data in Derby database. The tool accepts logon as admin to perform all the functions on user data on database. Once logged on, user will be given options for activity to perform.

**2.2 System Architecture**



The entire architecture of the application has 2-way communication with most of the features covered for User Identity Management. The Java application which contains the Java code resides on the computer which also provides the User Interface to the user.

The derby database used here is mainly to store data and retrieve it whenever required. This functionality is available with the help of the JDBC driver which acts as an intermediate for the Java application and the database.

Java application plays the most important role in the User Management Tool as it contains the Java source code which is compiled into bytecode whenever the javac compiler is used. To run the User Management program, the bytecode is converted using the just-in-time compiler. The output is machine code which is then fed to the memory and executed.

The user whenever uses the tool has the options to perform CRUD operations based on requirements. This operation is executed by Java code and data of identities of several users are stored or fetched from the Derby database.

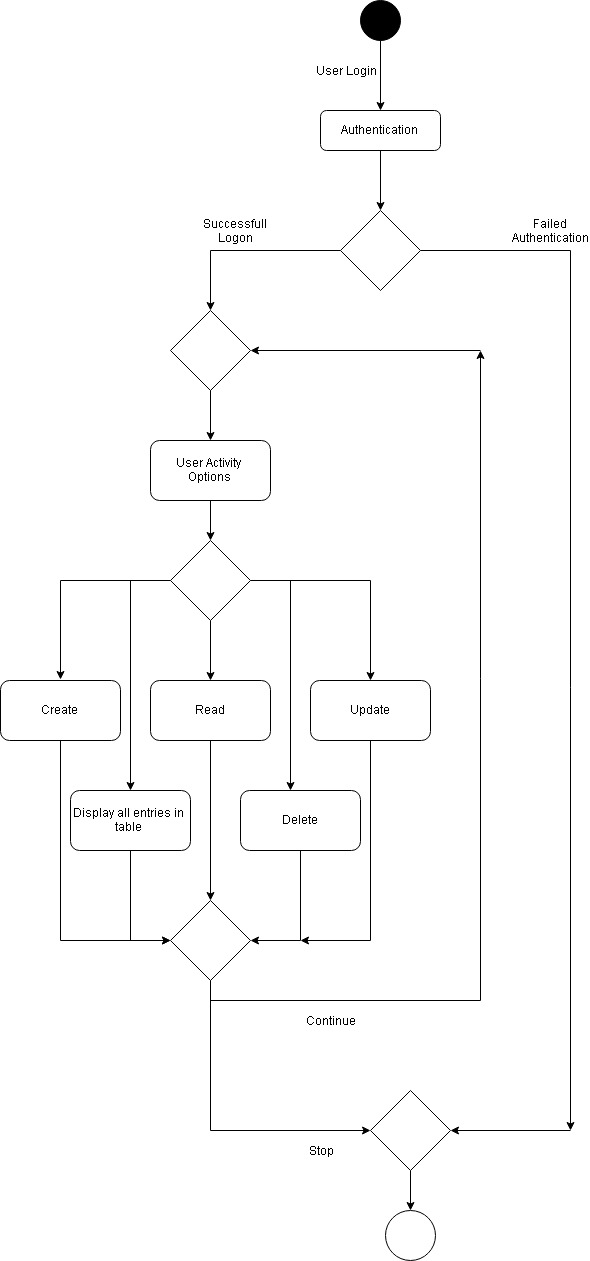
**3. SYSTEM CONTEXT:**

The system used for the User Identity Management will be connected externally to the Derby Database. There exists an option to connect to other data stores such as XML or file system to store the data, however, this document demonstrates the connection with the Derby DB. As a future enhancement, we could also have the XML or file system as a backup store in case of any DB failure.

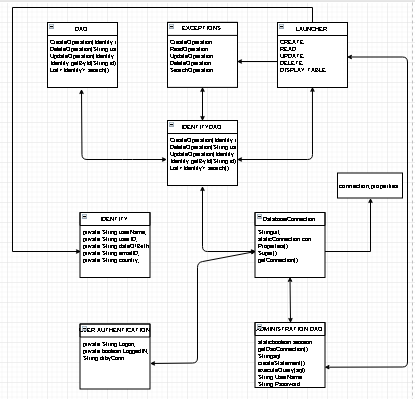
**4. SYSTEM DESIGN:**

The system which has been developed has mainly 5 main activities to be performed by the user during program execution, they are Create, Read, Update, Delete and Display entire table which is available. Prior to this, user needs to logon to the system with the user ID and password provided for the user.

A top-down approach of the functioning of the application

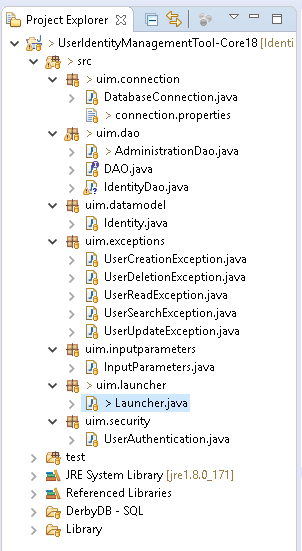
****

The UML diagram below provides an overview of all the objects defined for the project.



**4.1 DESIGN METHODOLOGY**

The below screenshot of the Project Explorer shows the entire list of objects involved in this Java Project. The design of all the classes and interface segregated based on the functionality of the code.



Folders relevant to Derby and Libraries would be described in a separate section below.

**5. CODE DEFINITION:**

The entire program code for the application has been written in Java considering all programming design standards. To create the initial tables and insert data into the database, SQL commands were used as an initial setup.

The code utilized to connect to the database, uses the connection properties from the .txt file to connect to Derby DB.

The code created for this application uses programming standards which involves creating of packages, classes, interface, test packages, SQL files, Reference Libraries and Library for derby jar file.

**5.1 CODE DESIGN OVERVIEW:**

The code developed for the tool contains several folders, such as mentioned below

* **src**

The src folder contains the source code of the java program

* **test**

The test folder contains certain code which is used for testing the functioning of the application before the application is to be used on the production environment.

* **JRE System Libraries**

Contains mainly the jar files required for Java application.

* **Reference Libraries**

Contains mainly the jar files required for the Derby DB

* **Derby DB - SQL**

Contains the .sql files which are used for creating the table and inserting data at the initial stages.

* **Library**

Contains the derbyclient.jar file which holds the DB for connecting to the Java application.

**5.2 PACKAGES INVOLVED:**

The packages created for the application are mainly based on the functionality of the code to be written. This is a direct implementation based on the classes defined in the UML diagram. The packages have a naming convention of uim.\* as this would be a much easier way to demonstrate a package and its functionality.

All the packages contain classes or interface in src folder, but the properties for db connection has been maintained in the .txt folder. Below are a list of the packages available in the src folder.

* uim.connection
* uim.dao
* uim.datamodel
* uim.exceptions
* uim.inputparameters
* uim.launcher
* uim.security

**5.3 CLASSES AND INTERFACES INVOLVED:**

Classes and Interfaces developed are applied based on the functionality of the code to be defined. These classes and interfaces, also contain few variables and methods utilized which is mentioned below along with their respective packages:

* uim.connection

**DatabaseConnection.java:**

This class is created mainly for database connection. Here the code is written to fetch the properties from the .txt file such as the url, userId and password to connect the database. Also, the code to open and close the database is developed here.

Variables and Methods used in this class:

*String url;*

*static Connection con*

*Properties()*

*Super()*

*getConnection()*

* uim.dao

**AdministrationDao.java**

This class is created to store the credentials of the users who need to login. These details are stored in a separate table - Administration.

Variables and Methods used in this class:

*static boolean session*

*getDaoConnection()*

*String sql*

*createStatement()*

*executeQuery(sql)*

**DAO.java (Interface)**

The DAO interface is mainly used for CRUD operation related functions and also to define the throws exceptions for each of the operations. This interface is used further by several other classes as ‘implements’.

Variables and Methods used in this class:

*public void CreateOperation(Identity entity)*

*public void DeleteOperation(String userId)*

*public void UpdateOperation(Identity entity)*

*public Identity getById(String id)*

*public List<Identity> search()*

**IdentityDao.java**

The IdentityDao implements DAO which allows the definition of the code to be written. Each of the operations such as Create, Read, Update and Deleted are defined here.

Variables and Methods used in this class:

*public void CreateOperation(Identity entity)*

*public void DeleteOperation(String userId)*

*public void UpdateOperation(Identity entity)*

*public Identity getById(String id)*

*public List<Identity> search()*

* uim.datamodel

**Identity.java**

The data model package contains the class Identity.java. This class contains all the user inputs which is required to be stored in the database. Here, the inputs which are requested from the user will be captured and stored from the get and set code.

Variables and Methods used in this class:

*private String userName;*

*private String userID;*

*private String dateOfBirth;*

*private String emailID;*

*private String country;*

* uim.exceptions

This package contains several exception classes which are classified based on the functionality.

UserCreationException

UserDeletionException

UserReadException

UserSearchExceotion

UserUpdateException

* uim.inputparameters

This class contains all the user inputs which is required to be stored in the database. Here, the inputs which are requested from the user will be captured and stored from the get and set code.

Variables and Methods used in this class:

*private String userName;*

*private String userID;*

*private String dateOfBirth;*

*private String emailID;*

*private String country;*

* uim.launcher

**Launcher.java**

This package contains the Launcher.java class which contains the main method. Here, the launcher also contains the UI texts which needs to be displayed to the user.

Variables and Methods used in this class:

String User\_Option;

String userChoice;

Identity old

String userID

* uim.security

**UserAuthentication.java**

The security package contains UserAuthentication.java class which is mainly for validation of the user’s logon credentials. This validation would be done with the Authentication table created in the Derby DB.

Variables and Methods used in this class:

private String Logon;

private boolean LoggedIN;

String drbyConn

**5.4 Properties file**

The properties file in the uim.connection package is used to provide the input parameters such as link, user ID and password to connect to the database.

****

**5.5 CODE DESCRIPTION**

The DAO is the interface which is the backbone for this project. It defines the CRUD operation which is required to be performed and is being implemented by IdentityDAO.

**public** **class** IdentityDao **implements** DAO

The input parameters or Identity class has all the required variables to be stored such as mentioned below

**private** String UserName;

**private** String UserID;

**private** String DateofBirth;

**private** String EmailID;

**private** String Country;

The user authentication class checks data from the table Administrator to verify & authenticate the user.

**6. CONNECTING TO DERBY**

**6.1 Overview**

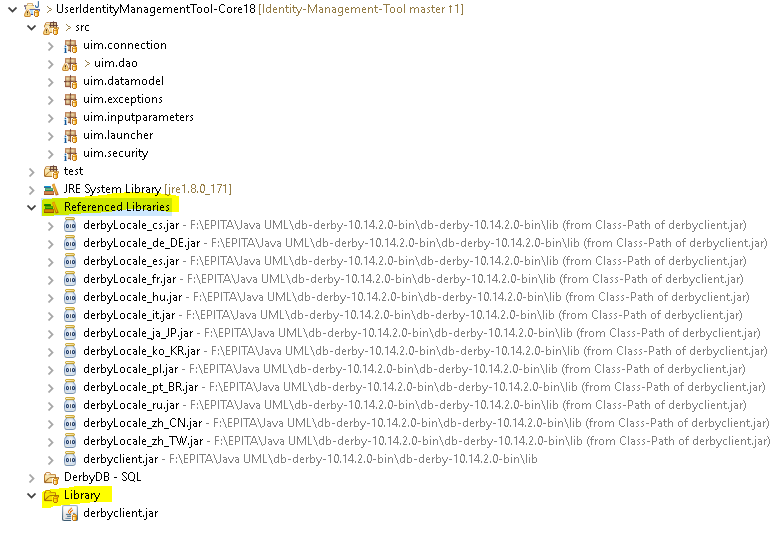
Connecting the Derby to database would require few steps as mentioned below:

Step1:

Download the Derby DB and run the startNetworkServer.

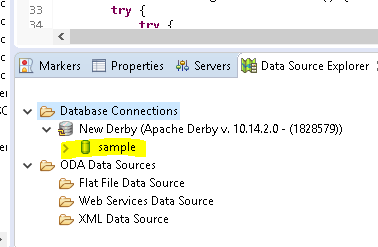
Step2:

Add the jar file into Library folder and also another folder (Referenced Libraries) to build path to the derbyclient.jar.



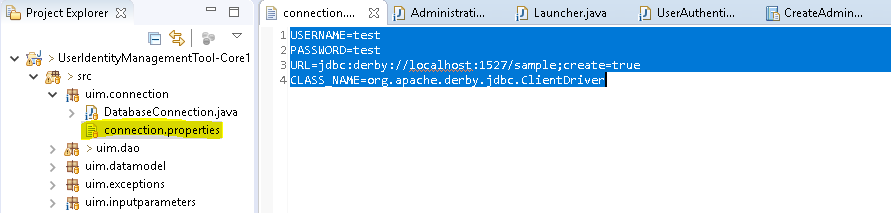
Step3:

Under Data Source Explorer click on New for Database connection and give source path, then the user and password for the database connection. Then click on test connection and then, apply and close.

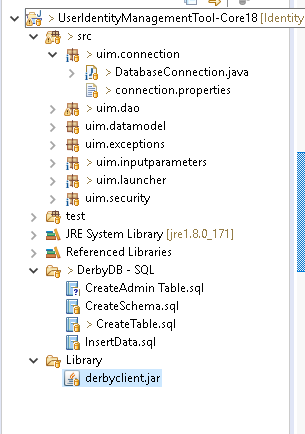


Step 4:

Provide the same user and password details in the connection.properties.txt file and give below details



**6.2 Implementation**



The connection.properties contain the property setting required for connecting to the user.

**Tables:**

For this application 2 tables have been created which is as mentioned below:

1. IDENTITIES:

The identities table contains the data identities of the users where we can perform the CRUD operation.

1. ADMINISTRATION:

The administration table contains the data of the users to be able to logon to the system.

*Note: default user ID and password as maintained in the table is admin with password admin123*

**Connection:**

To connect to the Derby DB we need to change the properties such as userID, password, url and Driver properties in the properties file. To allow compatibility with other databases such as MySQL, Derby, etc.. we need to just change the driver properties and url in the connection.properties file.

USERNAME=test

PASSWORD=test

URL=jdbc:derby://localhost:1527/sample;create=true

CLASS\_NAME=org.apache.derby.jdbc.ClientDriver

By calling this block of code we have made the developer to configure the details in properties file which would be fetched from connection.properties file.

input = **new** FileInputStream(System.*getProperty*("user.dir") + "/src/uim/connection/connection.properties");

prop.load(input);

After the code redirects to the connection.properties.txt file,

*con*=DriverManager.*getConnection*(

prop.getProperty("URL"),prop.getProperty("USERNAME"),prop.getProperty("PASSWORD"));

would fetch details as mentioned above. Once activity is performed, the connection is closed.

**INITIAL SETUP OF TABLES:**

To set up the 2 tables – Identities and Administration below mentioned code was used.

**CREATE** **TABLE** IDENTITIES (

userName **VARCHAR**(30) **NOT** **NULL**,

userID **varchar**(30) **NOT** **NULL**,

dateOfBirth **varchar**(30) **NOT** **NULL**,

emailID **varchar**(30) **NOT** **NULL**,

country **varchar**(30) **NOT** **NULL**);

**CREATE** **TABLE** ADMINISTRATION (

adminID **VARCHAR**(30) **NOT** **NULL**,

pass **VARCHAR**(30) **NOT** **NULL**);

Also, it is advisable to create a schema before we create the tables inorder to have the data in a particular location and not the default location.

**DB OPERATION CODE:**

For every activity of CRUD a corresponding SQL command is executed to update the database table.

**Create User:**

String query = "Insert into Identities (userName, userID, dateOfBirth, emailID, country) values" + " (?, ?, ?, ?, ?)";

**Reading a User:**

String sql = "Select \* from Identities where userID = '"+id+"'";

**Update a User:**

String query = "Update Identities SET dateOfBirth = ?, emailID = ?, country = ? where userName = ? and userID = ?"

**Delete a User:**

String sql = "delete from Identities where userID ='"+userId+"'";

**Display Entire Table:**

String sql = "Select \* from Identities";

For each of the operations performed, the exceptions are configured based on requirement.

**AUTHENTICATION CHECK**

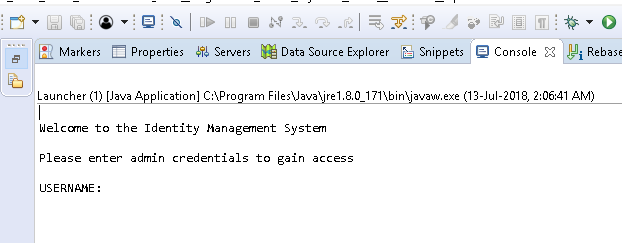
The authentication is checked with the Table Administration for entries provided by the user.

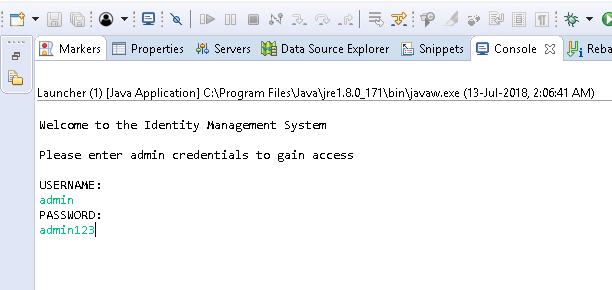
String sql = "Select \* from Administration where adminID = '"+userName + "' and pass = '" + password + "'";

The list of User Ids and password which needs to login should be maintained in this table.

**7. TESTING EVIDENCE:**

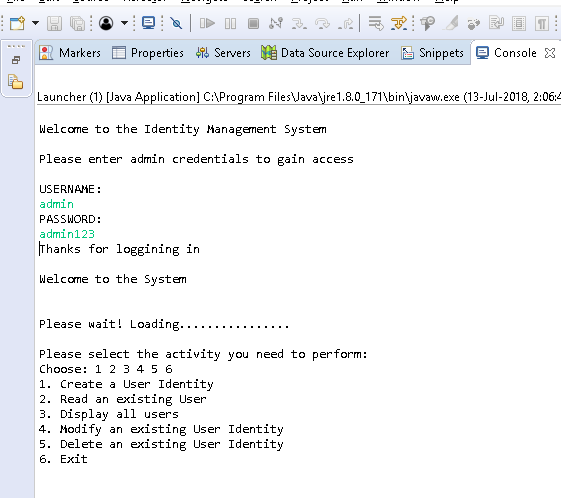
**Logon Screen**





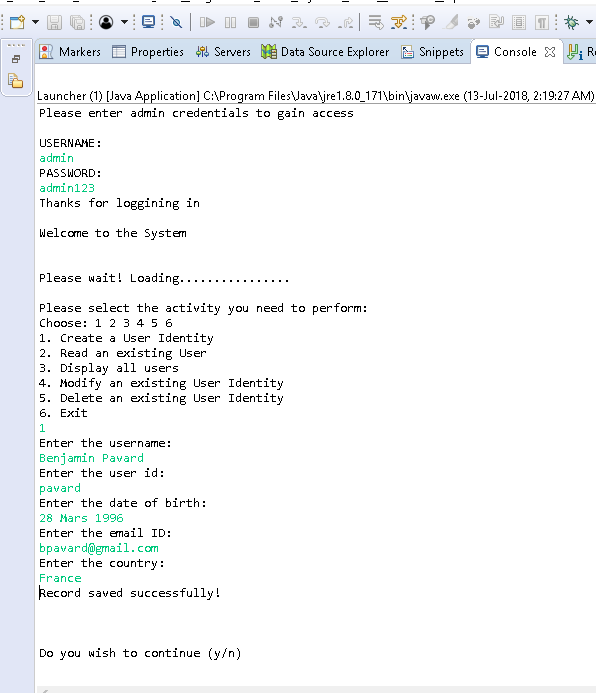
The user name and password need to be entered once requested.

**Initial Screen after logon:**

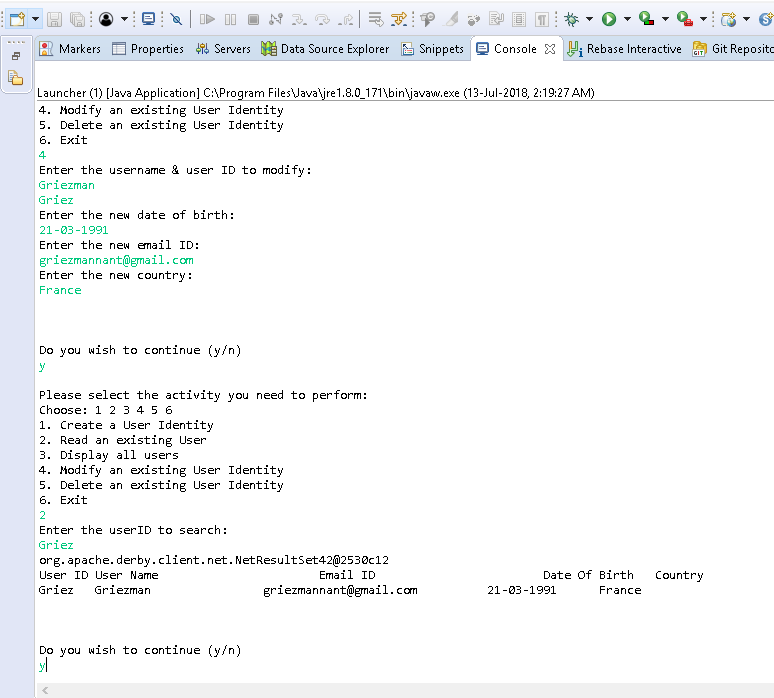


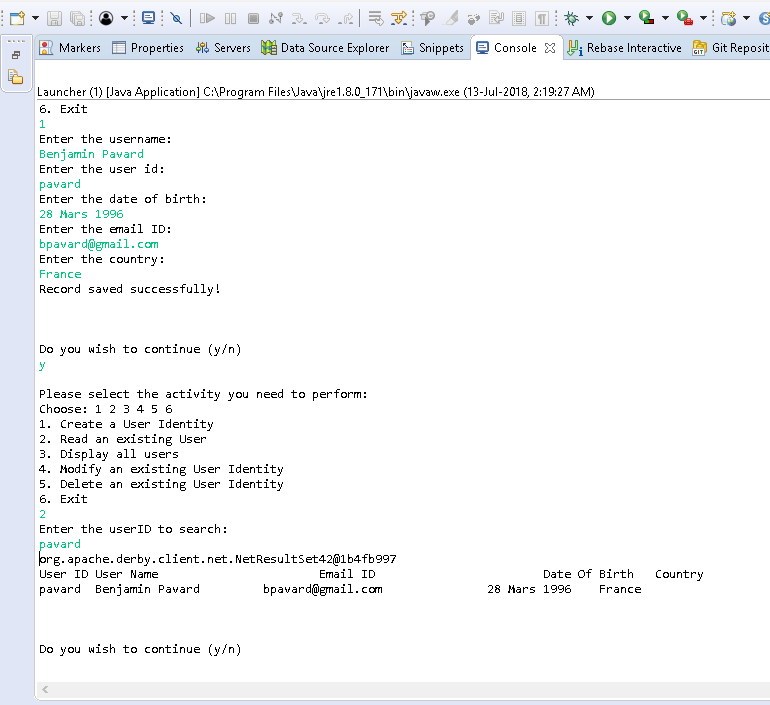
The user will be able to see the menu options available in the application.

1. **Create User in the table**

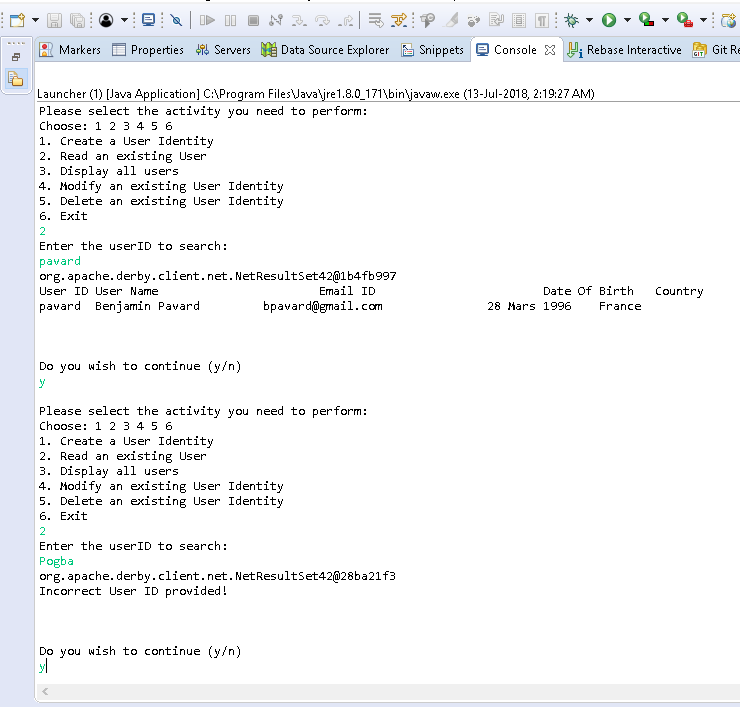


**2. Read exiting user from Table**

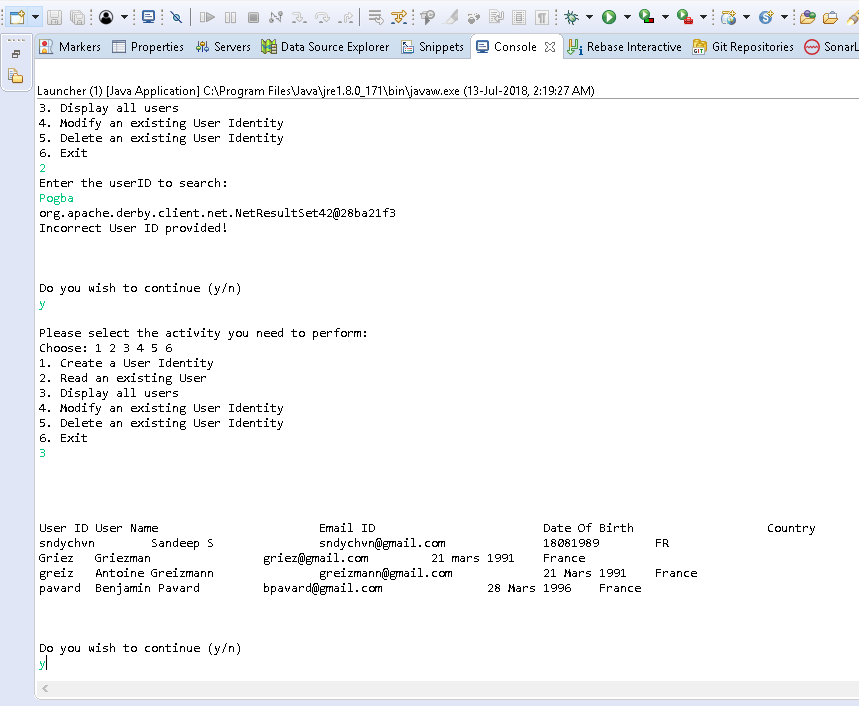




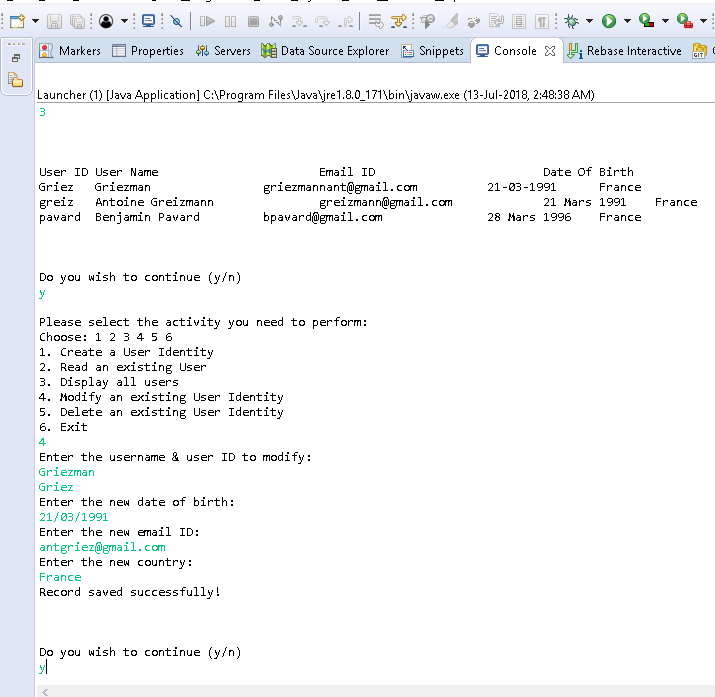
**Incorrect User ID provided to search…….**



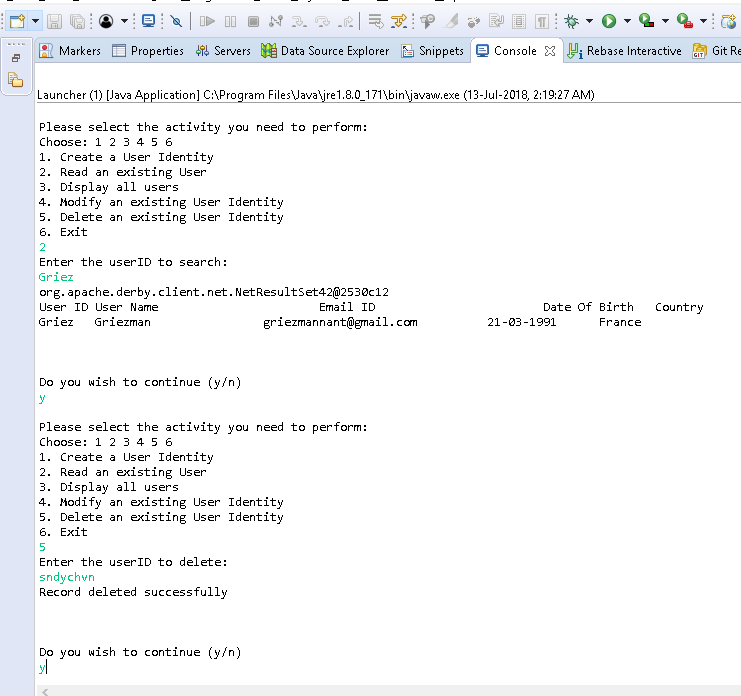
**3. Read all User Ids in Database:**



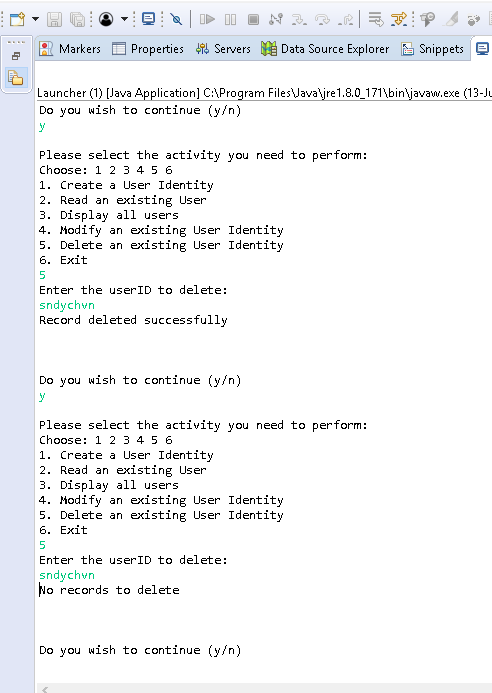
**4. Modify User ID**



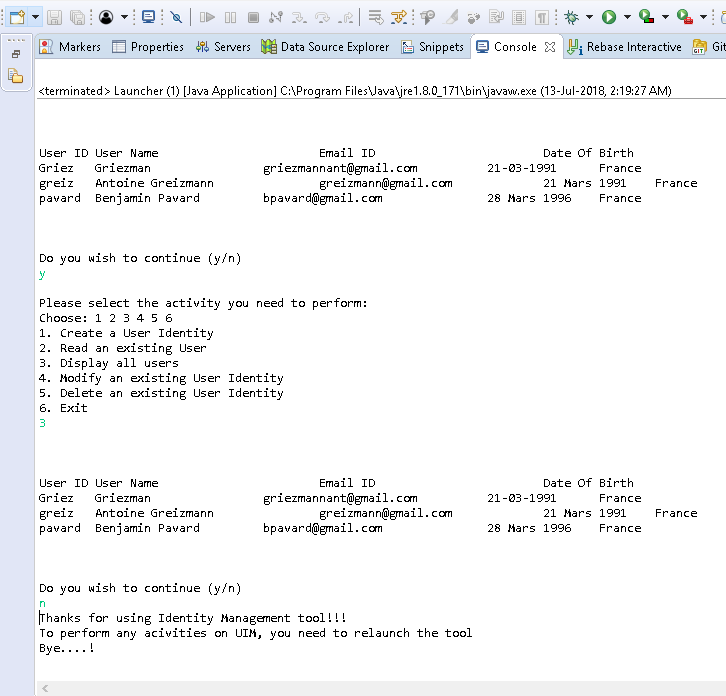
**5. Delete Record from Table**



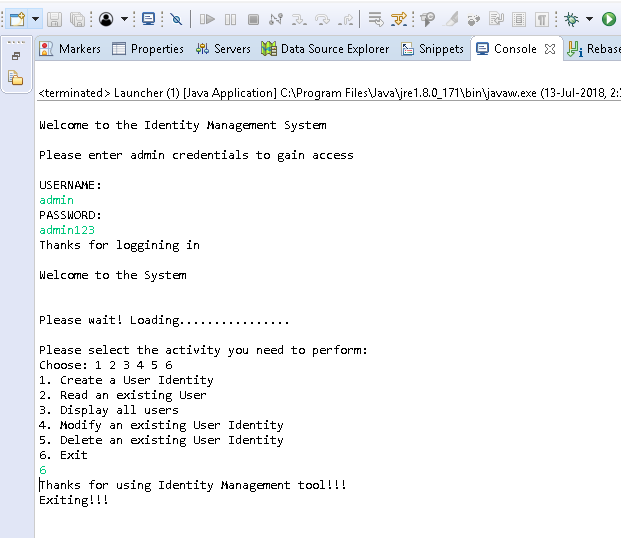
**Incorrect user ID to delete**



**Do not wish to continue…….**



**6. Exit option**



**User does not have access**

