# IDENTITY ACCESS MANAGEMENT

Technical Specification 1.0

Mr. RAHUL KUMAR THAI VALAPPIL

## Table of Contents

# 1. Subject description

Identity Access Management(IAM) System is an administrative application that deals with manipulation and management of individual identities, their authentication within or across system. The application must perform basic CRUD (Create, Read, Update and Delete) operations on a persistent storage system such as database. The term “User” is used as “the person who will manage the information” and the term “Identity” is used to signify the “information of user in the system”.

# 2. Subject Analysis

## 2.1. Major features

The application must allow the users to maintain and manipulate their identity information through a web interface. Every identity entity being comprised of first name, last name, email address and data birth and these details will persist in server database. Additionally, it should include a layer of security for a login process that will restrict the access from unauthorized users.
Application allows user to perform below functions related to his/her information:

- Configure User account for authentication with purchased license
- Login with valid credential
- Create an Identity
- Search identities from the system
- Update an existing Identity
- Delete an existing Identity

## 2.2. Application Feasibility

The application has to be accessed through a web browser. Depends on the server configure, the application would allow to deploy on a server with full capabilities of being accessible from within the network, or even online.

## 2.3. Data description

The application will use Derby database with tables for both Identity management and user authentication. The Hibernate and Spring frameworks are used to communicate between application business layer and data layer. Following are the data representation corresponds to Identity and User table.

### 2.3.1. Identity

Identity represents following data:

- Id – Automatically generated field within hibernate
- firstName -String
- lastName -String
- email – String
- birthdate - Date

### 2.3.2. User

User represents following data:

- Id - Automatically generated field within hibernate
- Username – String
- Password – using java simplified pooled digester to encrypt the data.

Additionally, license needs to register a user is hard coded in Spring application context xml, so currently additional user creation is not possible.

# 2.4. Expected results

End user should able to perform following functionalities and HTML pages will be loaded with some JavaScript and CSS functionality for the ease of use and aesthetics.

- Configure user authentication
- Perform user authentication
- Create an Identity
- Search Identities
- Update an Identity
- Delete an Identity

The user will have two main HTML pages, one for user authentication and other for performing CRUD operations.

# 2.5. Algorithms study

Application should adopt a simple MVC paradigm with help of Spring framework as business layer and Hibernate session factory as data layer. When user performs actions the view should send data request to the controller. This controller will contact data model in order to produce data requested and send back to the view where it will have visualized in HTML format. To improve security of the system user password data should be encrypted before storing in database.

# 2.6. Scope of the application (limits, evolutions)

## 2.6.1. Limits

- The application will do no more than the basic CRUD operations of Identity and user authentication
- Deletion or update can perform only one identity at a time
- Currently user management option is not exposed to the user layer
- Identity fields and criteria for search are not configurable
- License issuer is not implemented, so currently using a hard coded license which restricts registration of multiple user account in one system

## 2.6.1. Evolutions

- Separate configuration or setting page for managing user accounts
- License issuer module which is responsible for issuing new license and deciding the user access level
- An option to configure Identity fields and criteria for search in administrative level
- Multiple Identities can be deleted or updated at a time

# 3. Conception

## 3.1. Chosen algorithm

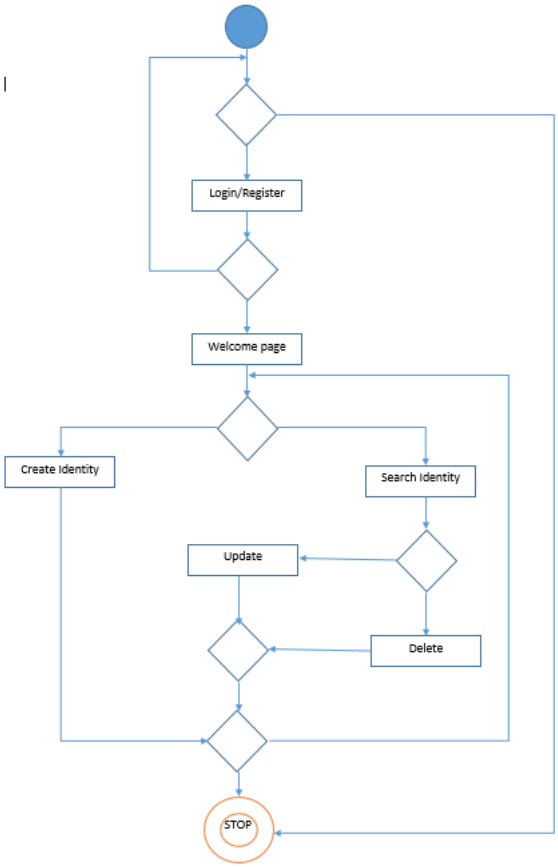
Chosen Java simplified encryption(jasypt) to encrypt user password before storing to database. The standard and simplified way allows to configure the algorithm through spring application context. The jasypt hashing lets to specify the algorithm (and provider) to be used for creating digests, the size of the salt to be applied, the number of times the hash function will be applied (iterations) and the salt generator to be used. Following are the current digest properties.

## 3.2. Data Structures

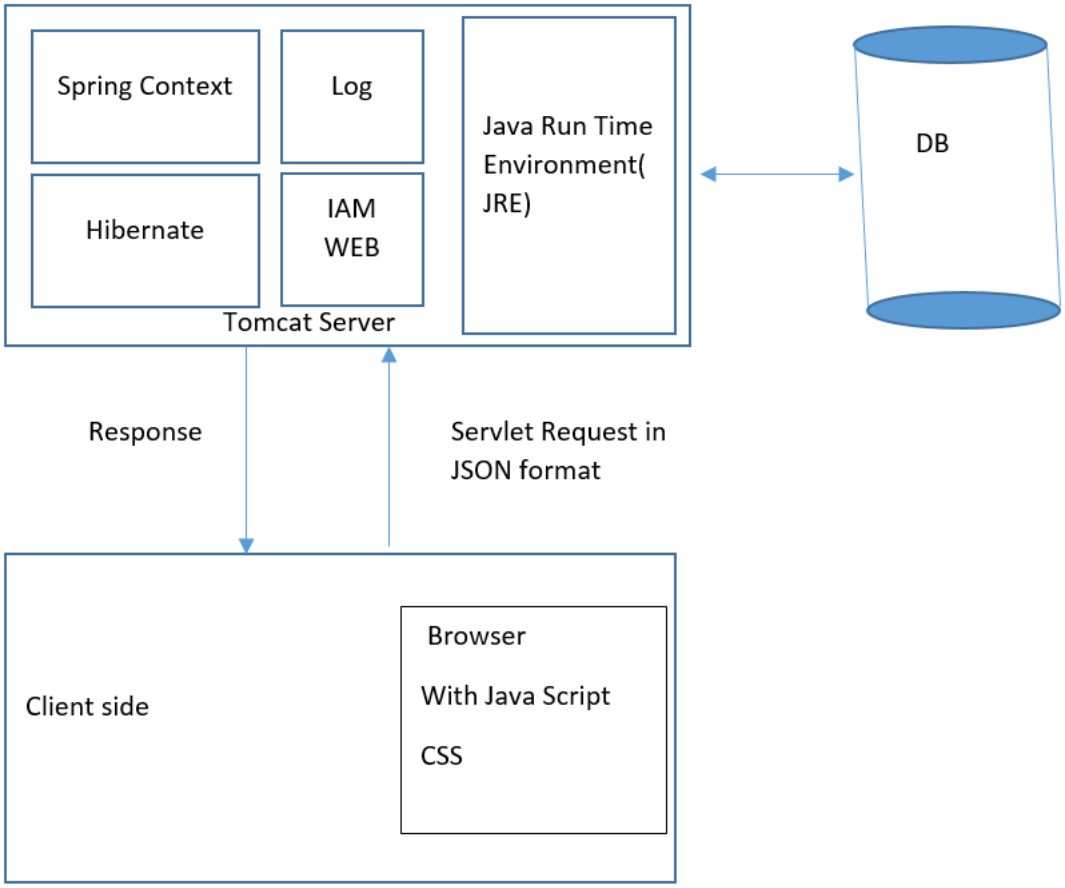
Application mainly Spring Framework, so it provides a comprehensive programming and configuration model for application. Spring allows to execute operation such as database without having to deal with actual database API's through dependency injection and inversion of control. Also application using Data Access Object patterns which allows flexibility in choosing business service level of identity data. Currently application using Hibernate framework for mapping identity object model to derby database.

pool Size - 2
Algorithm - SHA-256
Iterations - 100
salt Generator - org. jasypt. salt. RandomSaltGenerator
salt Size Bytes - 16

## 3.3. Global application flow



# 3.4. Global schema and major features schema

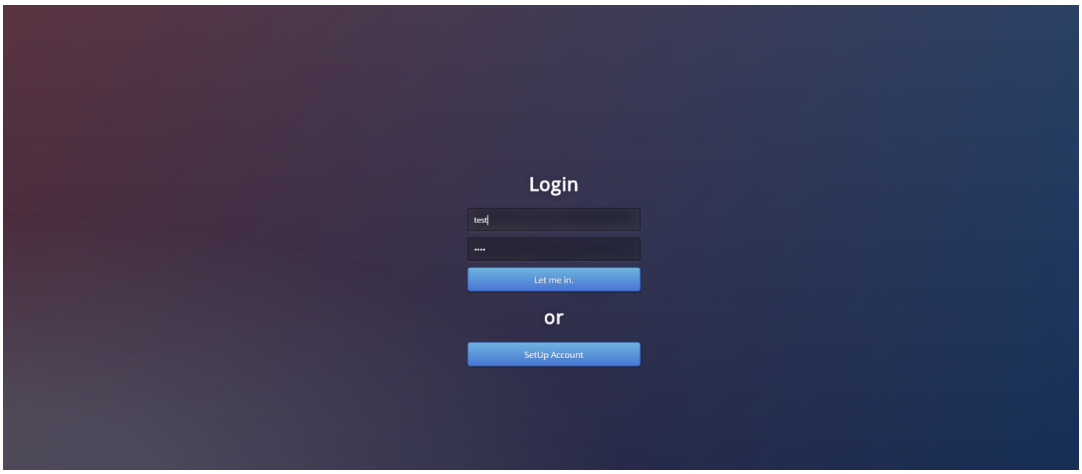


# 4. GUI Operations Description

## 4.1. Login page

Page consist of user login and registration options. Already registered user can enter their username and password for authentication otherwise need to register with valid license by clicking register button.

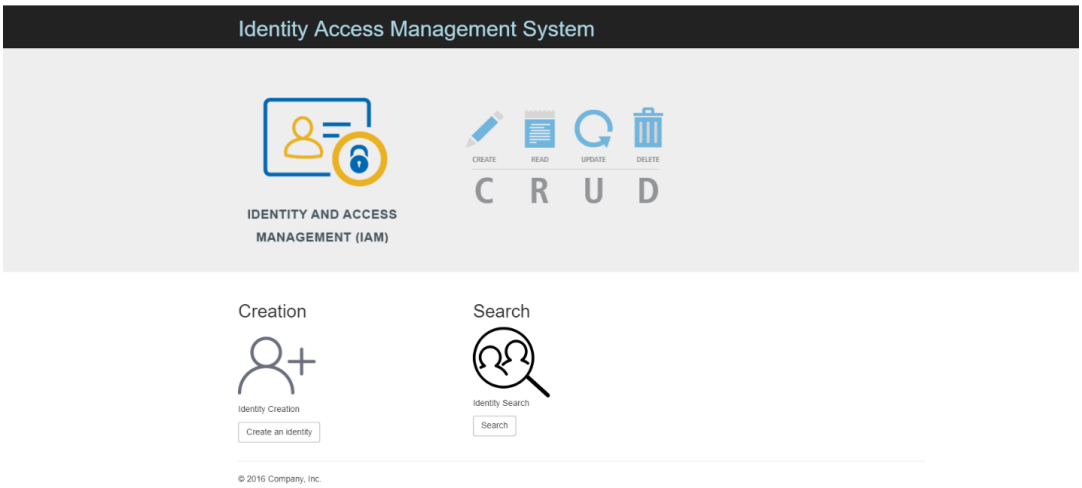
**Figure 1. IAM Login Page**




## 4.2. Home page

The page where user can choose their CRUD operations. Home page will get called once the user successfully authenticates his credential.

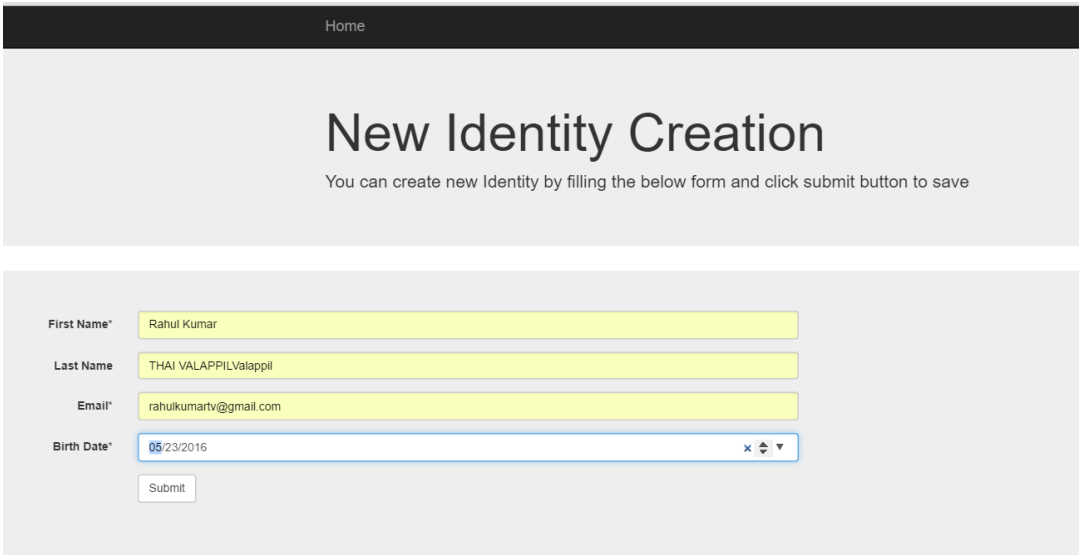
**Figure 2. IAM Welcome Page**




## 4.3. Create Page

The page provides facility to create new identities by entering details. Application will redirect to create page once user clicks on Create option from Home page. The page notify to user if any error happened during the create operation.

**Figure 3. IAM Create Page**




# 3.4. Search Page

The page facilitate user to search identity from their email address and provide an option to update or delete the identities.

**Figure 4. IAM Search Page**

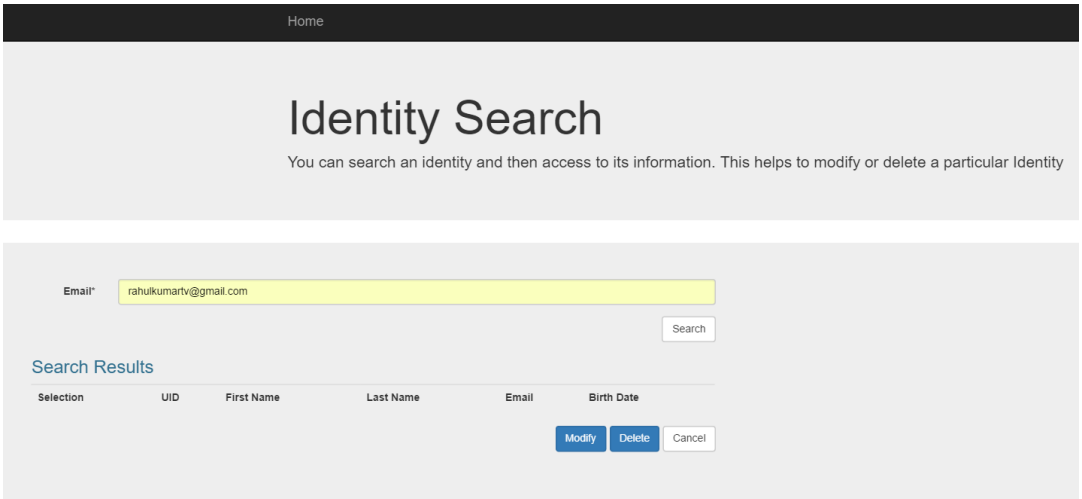


**Figure 5. IAM Update Operation**

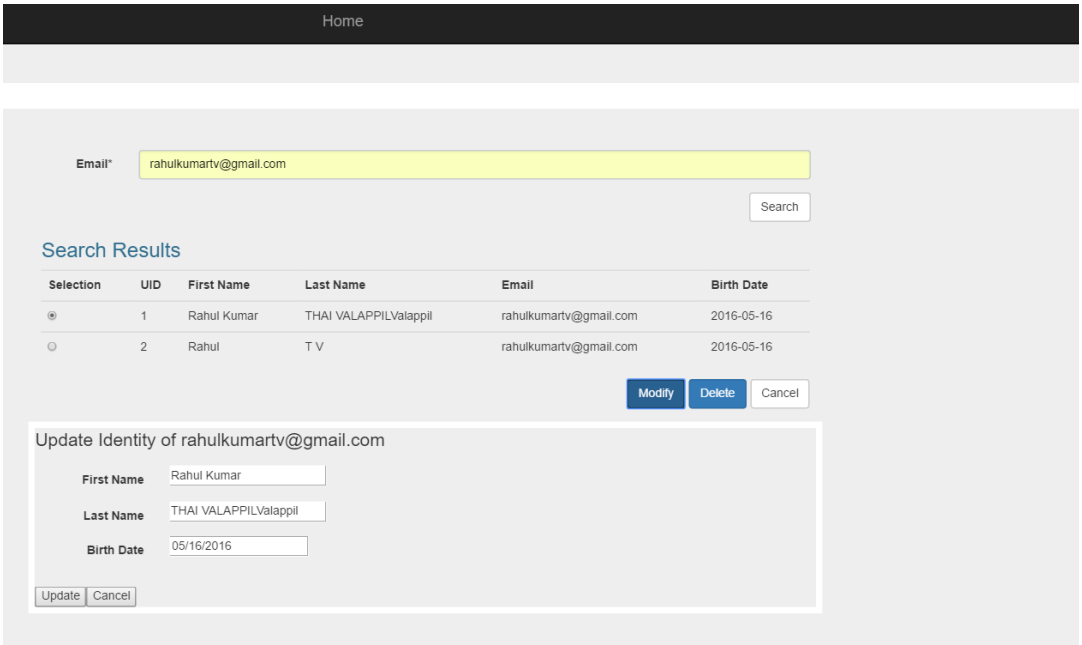
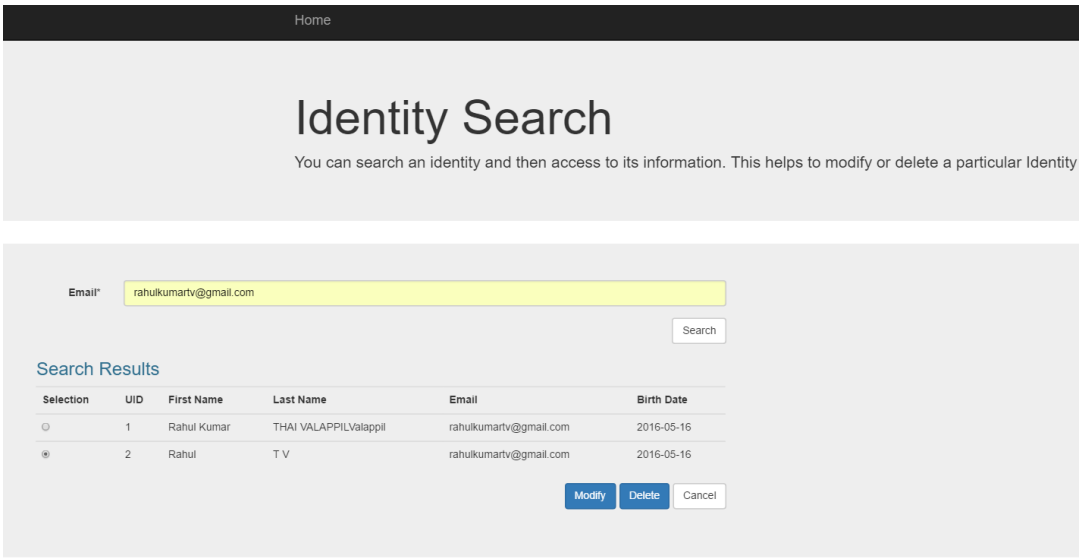



**Figure 6. IAM Delete Operation**



# 5. Configuration instructions

1. Install latest Java run time environment

2. Install Apache tomcat server and start the server

3. User should install the Apache Derby database before doing any operation and start the derby Server by executing startNetworkServer.bat from Derby installation folder. Please visit following links for more details

   https://db.apache.org/derby/papers/DerbyTut/install_software.html

4. Deploy the IAMWEB application using the Tomcat Host Manager

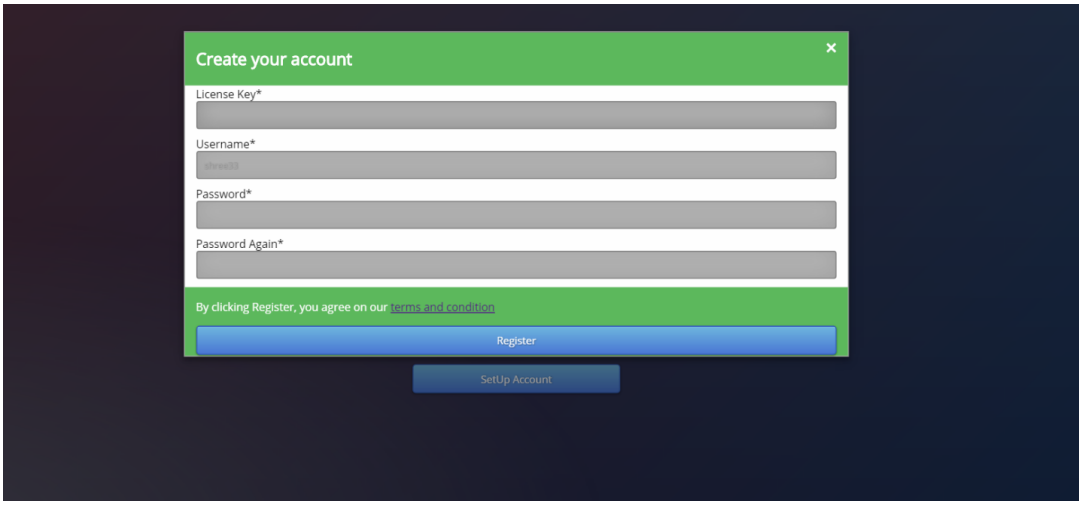
5. Configure log file location in log4j.properties from resource folder

   *log4j.appender.file.File=./log/trace.log*

6. For First time user. register with valid license is mandatory to access the Identity management system

   *License - 23a4a4bd-83a9-4b95-be9a-638bf5fd35f3*

**Figure 7. IAM Configuration**



# 5. Commented Screenshots

### Example 1. Authentication DAO Interface

```
/**
 * @author Rahul Thai Valappil
 * DAO interface responsible for user authentication and registration
 *
 */
public interface AuthDAOInterface {

    /**
     * Add user to the Authentication system with valid license.
     * So the user will have permission make changes to the Identity system
     * @param user
     */
    void addUser( Credentail user);
    /**
     * check whether a user exist for a license
     * @param license
     * @return true if same licensed user already exists other wise false
     */
    boolean licensedUserAlreadyExist( String license);
    /**
     * check the validity of  the user credential
     * @param user
     * @return
     */
    boolean checkUserAuthentication( Credentail user);
    /**
     * Authenticate license
     * @param license
     * @return
     */
    boolean checkValidLicense( String license);
}
```

## Example 2. Identity DAO Interface

```
/**
 * @author Rahul Thai Valappil
 * DAO responsible for handling  Operation such as Identity
 * Creation, Deletion,Updation and Search
 *
 */
public interface IdentityDAOInterface {


    /**
     * Read all the identity details created in the Identity system
     * @return list of identity
     */
    List<Identity> readAll();

    /**
     * search particular identity with email adress
     * @param identity
     * @return
     */
    List<Identity> search(Identity identity);

    /**
     * create an identity to the Identity system
     * @param identity
     */

    /**
     * Update existing identity details
     * @param identity
     */
    void update(Identity identity);

    /**
     *Delete an existing identity from the system
     * @param identity
     */
    void delete(Identity identity);

}
```

## Example 3. Log Manager

```
/**
 * @author Rahul Thai Valappil
 * The class handling logging of all the modules
 */
final public class LogManager {

    /**
     * private constructor
     */
    private LogManager() {
            // Prevent instantiation
        }

    /**
     * log messages or errors to the log file
     * @param message - message to log
     * @param className - information about class from where the log started
     */
    public static void log(final String message, final Class<?> className,final Level level ){
        final Logger loggerObj = Logger.getLogger(className);
        loggerObj.log(level, message);
    }
}
```

## Example 4. I am Web Java Script

```
/**
 * @author Rahul Thai Valappil
 * Java script file to handle the IAM System data request and manipulation
 *
 */
function postServlet(method,url,callback,datapair)
{
    var xhr;
    if( typeof XMLHttpRequest !== 'undefined')
    {
        xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function()
        {
            if(xhr.readyState == 4 && xhr.status == 200)
            {
                if (typeof(callback) !== "undefined" && callback)
                {
                    callback.call(xhr.responseText);
                }

                return;
            }
        }

    }
    else
    {
        var versions = ['MSXML2.XmlHttp.5.0','MSXML2.XmlHttp.4.0','MSXML2.XmlHttp.3.0','MSXML2.XmlHttp.2.0','Microsoft.XmlHttp'];
        for( var i=0,len=versions.length;i<len;i++)
        {
                xhr = new ActiveXObject(versions[i]);
        }
    }

    xhr.open(method, url, true);
    xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8');
    xhr.send(JSON.stringify(datapair));
}
```

## Example 5. User Data Model

```
package fr.rktv.iamcore.datamodel;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 * The class responsible for storing User informations like
 * username and encrypted password
 * user should have a valid license to register
 * @author Rahul Thai Valappil
 * @version 1.0
 */
@Entity
@Table(name="CREDENTAILS")
public class Credentail {

    /**
     * unique id of a User
     */
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int userId;

    /**
     * user name of a User
     */
    @Column(name="USERNAME")
    private String username;

    /**
     * encrypted password of a User
     */
    @Column(name="PASSWORD")
    private String password;

    /**
     * valid license of a User
     */
    @Column(name="License")
    private String license;

    /**
     * default constructor
     */
    public Credentail(){
        // The explicit constructor is here, so that it is possible to provide Javadoc.
    }
```

```java
/**
 * Constructor to initiate user object from username and password
 * @param username - String
 * @param password - String
 */
public Credentail(final String username, final String password)
{
    this.username = username;
    this.password = password;
}

/**
 * @return the configured license corresponds to a user
 */
public String getLicense() {
    return license;
}

/**
 * @param license -string
 * Set valid license for a user
 */
public void setLicense(final String license) {
    this.license = license;
}

/**
 * @return unique id corresponds to a user
 */
public int getUserId() {
    return userId;
}

/**
 * @param uid - int
 * Set unique id to user
 */
public void setUserId(final int uid) {
    this.userId = uid;
}

/**
 * @return user name of a user
 */
public String getUsername() {
    return username;
}

/**
 * @param userName - string
 * set username to user
 */
public void setUsername(final String userName) {
    this.username = userName;
}
```

## Example 6. Identity Data Model

```
package fr.rktv.iamcore.datamodel;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 * Represents the data model of Identity information
 * @author Rahul Thai Valappil
 * @version 1.0
 */
@Entity
@Table(name="IDENTITIES")
public class Identity {

    /**
     * unique id for an identity
     */
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int identId;

    /**
     * first name of an  identity
     */
    @Column(name="IDENTITY_FIRSTNAME")
    private String firstName;

    /**
     * last name of an identity
     */
    @Column(name="IDENTITY_LASTNAME")
    private String lastName;

    /**
     * email of an identity
     */
    @Column(name="IDENTITY_EMAIL")
```

```java
 */
@Column(name="IDENTITY_BIRTHDATE")
@Temporal(TemporalType.DATE)
private Date birthDate;


/**
 * default constructor
 */
public Identity(){
     // The explicit constructor is here, so that it is possible to provide Javadoc.
}

/**
 * constructor to create identity object from its first name, last name and email
 * @param firstName - String
 * @param lastName  - String
 * @param email  - String
 */
public Identity(final String firstName, final String lastName, final String email) {
    super();
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
}

/**
 * converts Identity object to JSON string format
 */
@Override
public String toString() {
    return "Identity [firstName=" + firstName + ", lastName=" + lastName
            + ", email=" + email + ", birthDate=" + birthDate + "]\n";
}

/**
 * @return the firstName of an Identity
 */
public String getFirstName() {
    return firstName;
}

/**
 * @param firstName - string
 * set first name to an identity
 */
public void setFirstName(final String firstName) {
```

```java
/**
 * @return the lastName from Identity
 */
public String getLastName() {
    return lastName;
}

/**
 * @param lastName - string
 * set lastName to identity
 */
public void setLastName(final String lastName) {
    this.lastName = lastName;
}

/**
 * @return the email from the identity
 */
public String getEmail() {
    return email;
}

/**
 * @param email - string
 * set email address to identity
 */
public void setEmail(final String email) {
    this.email = email;
}

/**
 * @return the birthDate from identity object
 */
public Date getBirthDate() {
    return birthDate;
}

/**
 * @param birthDate -Date
 * Set the birth dare of an identity
 */
public void setBirthDate(final Date birthDate) {
    this.birthDate = birthDate;
}

/**
 * @return unique id of an Identity created automatically
 */
```

## Example 7. POST Search Web Servlet

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    final PrintWriter out = response.getWriter();
    try
    {
        StringBuffer buffReq = new StringBuffer();
        String line = null;
        BufferedReader reader = request.getReader();
        while ((line = reader.readLine()) != null)
        {
            buffReq.append(line);
        }
        final Identity idObj = new ObjectMapper().readValue(buffReq.toString(), Identity.class);
        final HttpSession session = request.getSession();
        final Object loginSts = session.getAttribute("logon.isDone");
        final boolean bloginSts = loginSts == null? false:(boolean)loginSts;
        if( bloginSts )
        {
            List<Identity> searchResult;
            searchResult= hibernateDAO.search(idObj);
            JSONArray jsonArray = new JSONArray();
            ObjectMapper mapper = new ObjectMapper();
            for (Identity identity : searchResult) {
                String jsonInString = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(identity);

                jsonArray.put(jsonInString);
            }
            JSONObject json = new JSONObject();
            json.put("SearchResults", jsonArray);
            out.print(json);
        }
        else
        {
            out.write("Error : Unauthorized access. Please login with valid credentail to search Identities");
        }

    }
    catch(Exception exp )
```

## Example 8. Iam Core Unit Test Snippet

```
@Test
public void hibernateDaoSearch() throws IllegalArgumentException, IllegalAccessException{
    Identity identity = new Identity("Prince","Mathew", "Prince@gmail.com");
    dao.write(identity);
    identity = new Identity("Ragesh","T V", "ragesh@gmail.com");
    dao.write(identity);
    final List<Identity> searchResul =dao.search(identity);
    Assert.notEmpty(searchResul);
}
/**
 * Test hibernate Dao Delete
 * @return void
 */
@Test
public void hibernateDaoDelete() throws IllegalArgumentException, IllegalAccessException{
    final Identity identity = new Identity("DeleteUser","Test", "delete@gmail.com");
    dao.write(identity);
    List<Identity> searchResul =dao.search(identity);
    Assert.isTrue( !searchResul.isEmpty());
    identity.setIdentId(searchResul.get(0).getIdentId());
    dao.delete(identity);
    searchResul =dao.search(identity);
    Assert.isTrue(searchResul.isEmpty());
}
@Test
public void logErrorLevelTest(){
    LogManager.log("Test Error level Log", this.getClass(),Level.ERROR);
}
/**
 * Test case checking failure case of licensedUserAlreadyExist method of Authenticate DAO
 * @throws IllegalArgumentException
 * @throws IllegalAccessException
 */
@Test
public void licesUsrAlreadyExistTestFail() throws IllegalArgumentException, IllegalAccessException{
    final Credentail user = new Credentail("testuser","test");
    user.setLicense(license);
    authenticationDAO.addUser(user);
    Assert.isTrue(!authenticationDAO.licensedUserAlreadyExist("23a4a4bd-83a9-4b95-be9a-638bf5fd3534"));
}
```

## Example 9. Web Servlets Unit Test Code Snippet with Mocking

```
@Before
public void setUp() {
    servlet = new Search();
    request = new MockHttpServletRequest();
    request.setContentType("application/json");
    request.setCharacterEncoding("UTF-8");
    session = Mockito.mock(HttpSession.class);
    //session.setAttribute("logon.isDone", true);
    hibernateDAO = Mockito.mock(IdentityHibernateDAO.class);
    request.setSession(session);
    response = new MockHttpServletResponse();
    servlet.setHibernateDAO(hibernateDAO);
}
@Test
public void searchServletPostTestSucc() throws ServletException, IOException, ParseException {
    final String datapair = "{\"firstName\": \"Rahul\" , \"lastName\": \"T V\", \"email\": \"sample@gmail.com\", \"birthDate\": \"1987-05-05\"}";
    request.setContent(datapair.getBytes());
    Identity ident = new Identity("Rahul","T V","sample@gmail.com");
    Mockito.when(session.getAttribute("logon.isDone")).thenReturn(true);
    List<Identity> results = new ArrayList<Identity>();
    results.add(ident);
    Mockito.when(hibernateDAO.search( any(Identity.class))).thenReturn(results);
    servlet.doPost(request, response);
    assertEquals("application/json", response.getContentType());
    assertTrue(!response.getContentAsString().isEmpty());
}
@Test
public void searchServletPostTestFail() throws ServletException, IOException {
    Mockito.when(session.getAttribute("logon.isDone")).thenReturn(true);
    Mockito.ignoreStubs(hibernateDAO);
    servlet.doPost(request, response);
    assertEquals("Error while Searching the Identity Please contact Administator", response.getContentAsString());
}
@Test
public void searchServletPostTestUnAuthAcess() throws ServletException, IOException {
    final String datapair = "{\"firstName\": \"Rahul\" , \"lastName\": \"T V\", \"email\": \"sample@gmail.com\", \"birthDate\": \"1987-05-05\"}";
    request.setContent(datapair.getBytes());
    Mockito.when(session.getAttribute("logon.isDone")).thenReturn(false);
    Mockito.ignoreStubs(hibernateDAO);
    servlet.doPost(request, response);
```

# 6. Bibliography

1. http://thomas-broussard.fr/work/java/courses/index.xhtml
2. https://db.apache.org/derby/papers/
3. https://docs.oracle.com/javase/tutorial/
4. http://www.jasypt.org/encrypting-configuration.html
5. https://dzone.com/refcardz/mockito
6. http://logging.apache.org/log4j/2.x/
7. http://spring.io/ and http://hibernate.org/