

Alumni Management System

Project Report

Semester 6

Software Engineering

B. Tech Computer Science And Engineering

By:

Sarvesh Shroff AP10110010057

&

Aditya Dhir AP17110010001

Index

Title Page Number

1. SRS Document 2
2. DFDs 15
3. Structured Chart 18
4. Use Case Diagram 19
5. Class Diagram 20
6. Object Diagram 21
7. Sequence Diagram 22
8. Collaboration Diagram 23
9. State-Chart Diagram 25
10. Activity Diagram 26
11. Component Diagram 27
12. Deployment Diagram 28
13. Unit testing 29

Software Requirements Specification

Version 1.0

# Alumni Management System

Software Engineering Project,

Student of SRM University AP Amaravati

## Table of Contents

[**Table of Contents** ii](#_30j0zll)

1. [**Document purpose** 1](#_1fob9te)
2. [**Document overview** 1](#_3znysh7)
3. [**General description of the product** 1](#_2et92p0)
   1. [The current situation 1](#_tyjcwt)
   2. [Purpose of the product 1](#_3dy6vkm)
   3. [Product context 1](#_1t3h5sf)
   4. [Benefits 2](#_4d34og8)
   5. [Actors 2](#_2s8eyo1)
      1. [Alumnus 2](#_17dp8vu)
      2. [Secretary Staff 2](#_3rdcrjn)
   6. [System boundary 3](#_26in1rg)
   7. Use cases description 4
      1. **Access Alumni Home Page** 4
      2. **Fill out the Survey** 5
      3. **Create a new entry** 6
      4. **Update an Entry** 7
      5. **Search for an Alumni/E-mail an Alumni** 8
4. **Non-functional requirements** 9
   1. User Interface Requirements 9
   2. Performance Requirements 9
   3. Availability & Reliability 9
   4. Security Requirements 9
5. **Document purpose**

This document is intended to describe accurately the capabilities that the software product “Alumni Database Management System. It should provide to its end-users and also to specify all the non-functional requirements that the application should implement, regarding subjects like: performance, availability, reliability, security, etc.

# Document overview

The remainder of this document is three chapters, the first offering a general description of the software product about the initial situation, the purpose of the project, the context and the benefits of the project.

The second chapter lists the functional requirements that the software product should meet. So, it describes the actors, the system boundary and the use cases.

The final chapter exposes the non-functional requirements of the application, such as: performance, safety, security issues, etc.

# General description of the product

## The current situation

In SRM university AP Amaravati, there is no Alumni Management System. Our aim is to build an alumni management system which will help alumni of SRM university AP Amaravati to register in the database and have access to all the events and much more.

## Purpose of the product

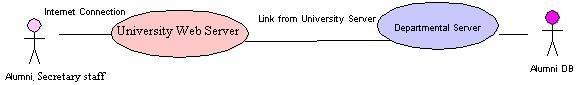
The software product is intended to offer remote access to the Alumni Database for the Alumni of SRM University AP Amaravati. So, the stored information will remain accurate and it will be visible both to the secretary staff and alumni.

## Product context

The software product is designed to run on the departmental server of the faculty of Automatic Control and Computers. It will receive incoming requests from the University

Web Server, it will process them and, finally, it will query the underlying Alumni database and obtain the desired results/information, which will be passed towards the web server.

Users would access the product by using their web browser, so an Internet connection is necessary to access the system.



## Benefits

This software product is supposed to satisfy both the alumni wish for sharing access to the database storing their information and the secretary staff need of storing accurate information regarding the graduates.

# Functional requirements

## Actors

The profiles of all user categories are described here.

### Alumnus

The ***Alumnus*** performs any of the following operations:

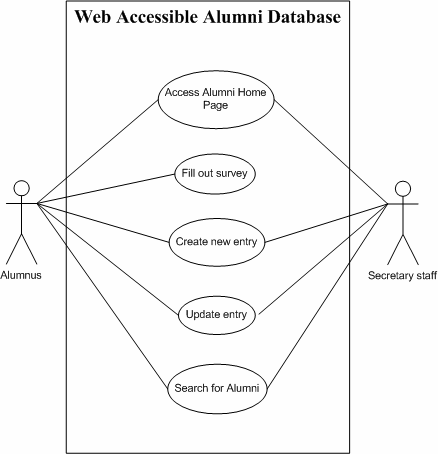
* + - * Access Alumni Home Page
      * Fill out survey
      * Create new entry
      * Update entry
      * Search for Alumni

### Secretary Staff

The ***Secretary Staff*** performs the same operations like the Alumni, except for the *Fill out the survey* operation.

So, we decided to consider ***Alumnus*** to be the main actor of this software system.

## System boundary



* 1. **Use cases description**

### Access Alumni Home Page

|  |  |
| --- | --- |
| **Use Case Name:** | Access Alumni Home Page |
| **Brief Description:** | The Departmental Web Server is waiting on  alumni to connect. |
| **Priority** | Essential |
| **Trigger** | Alumni select the Alumni link on the ACC  home page. |
| **Precondition** | Alumni is connected to the Internet and on  the ACC home page |
| **Basic Path** | 1. University Web Server sends the Alumnus to the Departmental Server. 2. The Departmental Server presents the Alumnus with the Alumni Home Page. |
| **Alternate Path** | N/A |
| **Post condition** | Alumni is on the Alumni Home Page |
| **Exception Path** | If there is a connection failure, the  Departmental Server returns to the wait state |

* + 1. **Fill out the Survey**

|  |  |
| --- | --- |
| **Use Case Name:** | Survey |
| **Brief Description:** | This operation permits alumni to fill out a  survey. |
| **Priority** | Essential |
| **Trigger** | Alumni choose to fill out a survey. |
| **Precondition** | Alumni are connected to the Internet and on  the ACC Alumni Home Page |
| **Basic Path** | 1. The Departmental Server presents the Alumni with a form. 2. Alumni fill in the form and click submit 3. The Departmental Server checks to see if all required fields are not empty. 4. If the required fields are not empty, the Departmental Server creates a new record in then Survey Table of the Alumni Database. 5. If any of the required fields are empty, the Departmental Server returns a message and returns the Alumnus to the Survey form. 6. The Departmental Server returns the Alumnus to the Alumni Home Page |
| **Alternate Path** | N/A |
| **Post condition** | The survey record is created in the Survey  Table of the Alumni Database. |
| **Exception Path** | 1. If the connection is terminated before the form is submitted, the fields are all  cleared and the Departmental Server is returned to the wait state. |

* + 1. **Create a new entry**

|  |  |
| --- | --- |
| **Use Case Name:** | Create a new entry |
| **Brief Description:** | This operation permits alumni to create a  new entry on the Entries page. |
| **Priority** | Essential |
| **Trigger** | Alumni choose to create a new entry on the  Entries page. |
| **Precondition** | Alumni must be connected to the Internet  and on the ACC Entries page. |
| **Basic Path** | 1. Alumni click on add a new entry. 2. The Departmental Server returns a form. 3. Alumni fill in the form and click *submit*. 4. The Departmental Server checks to see if any required field is empty. 5. If any required field is empty, the Departmental Server will send a message and return Alumni to the new entry form page. 6. If no required field is empty, the Departmental Server will create a new record in the Alumni Table in the Alumni Database, and return Alumni to the ACC Alumni Home Page. 7. Alumni may select Cancel. 8. If Alumni select Cancel, the form is cleared and Alumni are returned to the ACC Alumni Home page. |
| **Alternate Path** | N/A |
| **Post condition** | A record is created in the Alumni Table of  the Alumni Database. |
| **Exception Path** | 1. If the connection is terminated before the form is submitted, the fields are cleared and the Departmental Server is returned to the wait state. 2. If the connection is terminated after the form is submitted, but before Alumni are returned to the ACC Alumni Home Page, the record is created in the Alumni Table of the Alumni Database. |

* + 1. **Update an Entry**

|  |  |
| --- | --- |
| **Use Case Name:** | Update an Entry |
| **Brief Description:** | This operation permits alumni to update an existing  entry in the Alumni Database. |
| **Priority** | Essential |
| **Trigger** | Alumni choose to update an existing entry in the  Alumni Database. |
| **Precondition** | Alumni must be connected to the Internet and on the  ACC Entries Page. |
| **Basic Path** | 1. The Alumnus clicks on update an entry link. 2. The Departmental Server returns a form. 3. The Alumnus enters his/her year of graduation. 4. The Departmental Server queries the Alumni Database for that particular year and returns a table of all graduates from that year in a form with radio buttons and requesting their password. 5. If the password does not match, the Departmental Server returns a message and allows the Alumnus to try again. 6. If after 3 tries, the password does not match, the Departmental Server will return a message telling the Alumnus to contact the ACC designated faculty member to receive their password. 7. If the password matches, go to 8. 8. The Departmental Server returns a form with the data for that Alumnus in it and a message to update the data they wish and click submit. 9. The Departmental Server replaces the old data with the new data and returns the Alumnus to the ACC Alumni Home Page. |
| **Alternate Path** | If after three attempts to match the name and password,  the Departmental Server will return a message and block the Alumnus from the update section. |
| **Post condition** | The record in the Alumni Table of the Alumni Database has been updated and the Alumnus is returned to the  ACC Alumni Home Page. |
| **Exception Path** | 1. If the connection is terminated before the form is submitted, the fields are cleared and the Departmental Server is returned to the wait state. 2. If the connection is terminated after the form is submitted, but before the Alumnus is returned to the ACC Alumni Home Page, the record in the Alumni Table of the Alumni Database is updated and the Departmental Server is returned to the wait state |

* + 1. **Search for an Alumni/E-mail an Alumni**

|  |  |
| --- | --- |
| **Use Case Name:** | Search for an Alumni |
| **Brief description:** | This operation permits the Alumnus to search for the information  belonging to other Alumni. |
| **Priority** | If time permits. |
| **Trigger** | The Alumnus chooses to search/e-mail Alumnus. |
| **Precondition** | The Alumnus is connected to the Internet and on the ACC Alumni  Home Page. |
| **Basic Path** | 1. The Departmental Server returns a form. 2. The Alumnus fills in the form and clicks submit. 3. The Departmental Server checks to see if any required fields are empty. 4. If any required fields are empty, the Departmental Server returns a message and the form. 5. If none of the required fields are empty, the Departmental Server queries the Alumni Database for the requested Alumnus’s entry. 6. The Departmental Server returns the non-private information on the requested Alumnus and a message stating if the requested Alumnus will accept e-mails. 7. If the requested Alumnus is not in the Alumni Database, the Departmental Server returns a message and the Alumnus is returned to the ACC Home Page. 8. If the requested Alumnus will accept e-mails, the Alumnus can select *E-mail this Alumnus*. 9. If not, the Alumnus can select *Search for another Alumnus* or return to ACC Alumni Home Page. 10. If the Alumnus chooses to *Search for another Alumnus* go to step 2. 11. If the Alumnus selects return to ACC Alumni Home Page, the Departmental Server returns the Alumnus to the ACC Alumni Home Page. 12. The Departmental Server presents the Alumnus with a form to fill out and a place for the message. 13. The Alumnus selects *send*. 14. The Department Server will forward the e-mail with all necessary information to the requested Alumnus. 15. The Departmental Server returns a message and returns the Alumnus to the ACC Alumni Home Page |
| **Alternate Path** | N/A |
| **Post condition** | The Alumnus receives the information on the requested Alumnus, receives e-mail confirmation message, or he is returned to the ACC  Alumni Home Page |
| **Exception Path** | 1. If the connection is terminated before the information is returned, the Departmental Server is returned to the wait state. 2. If the connection is terminated after the information is returned, the Departmental Server is returned to the wait state |

1. **Non-functional requirements**
   1. **User Interface Requirements**

The user interface of the application must be user-friendly, intuitive and easy to use, implementing the ergonomics standards.

## Performance Requirements

The system shall function in real-time: any operation on the stored information, triggered by the Alumni, shall complete in less than *10 seconds*.

The system shall allow simultaneous use by at least *100 users*, without data corruption.

## Availability & Reliability

The software system could provide automatically generated *backup* (on external hard drives) containing all the stored information at the time the backup is taken. The system shall allow authorized users to restore the data from an existing backup.

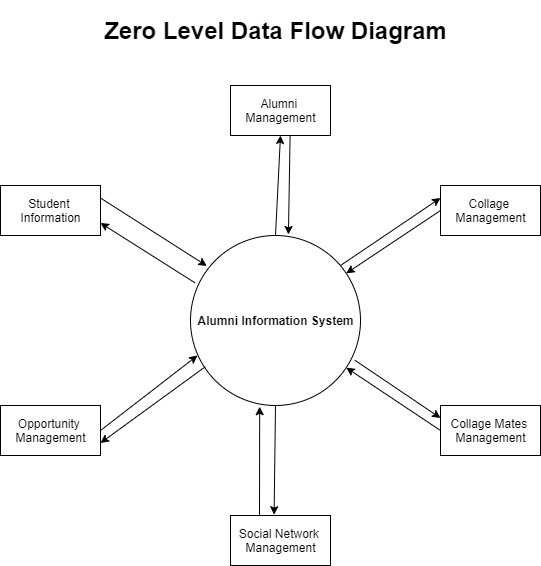
## Security Requirements

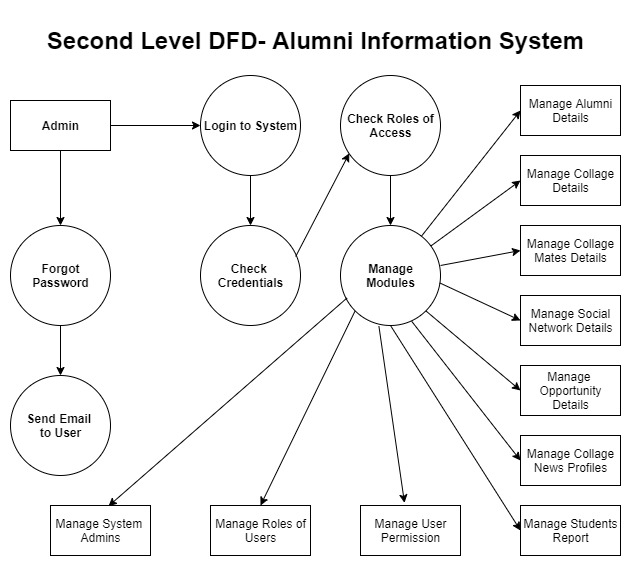
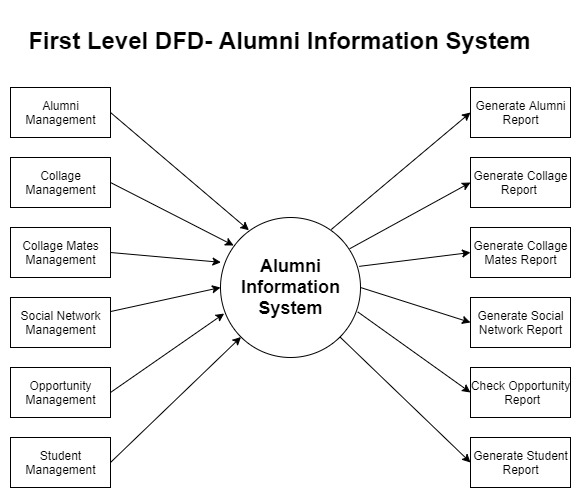
In order to use certain features of the system, users must first authenticate themselves by name and password. The system shall not allow access if the user fails to provide correct log in information.

The system should automatically perform log out if the user has been idle for a specific period (e.g. 30 minutes).

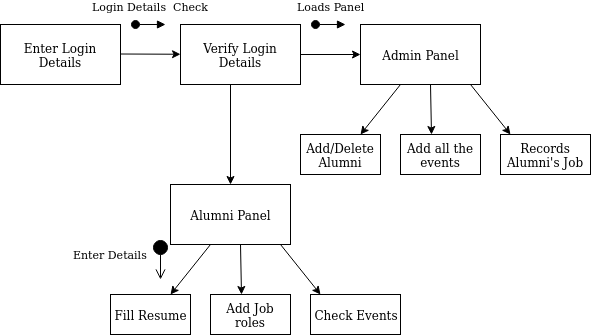
Physical access to the computer(s) storing the Alumni Database shall be restricted to authorized personnel.

**Data Flow Diagram (DFD)**

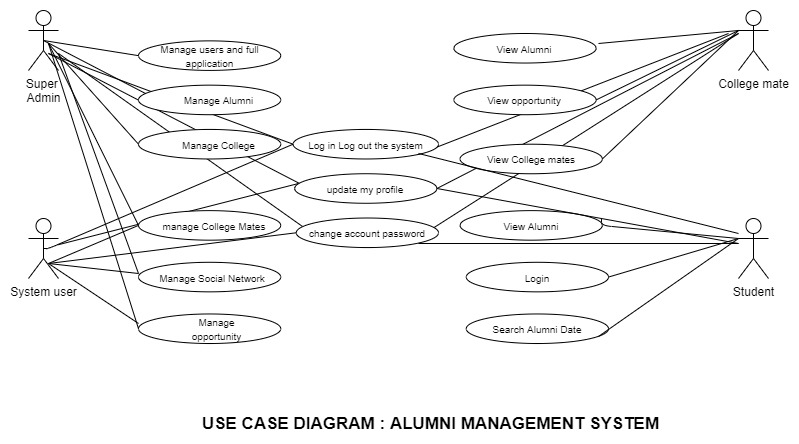
****

****

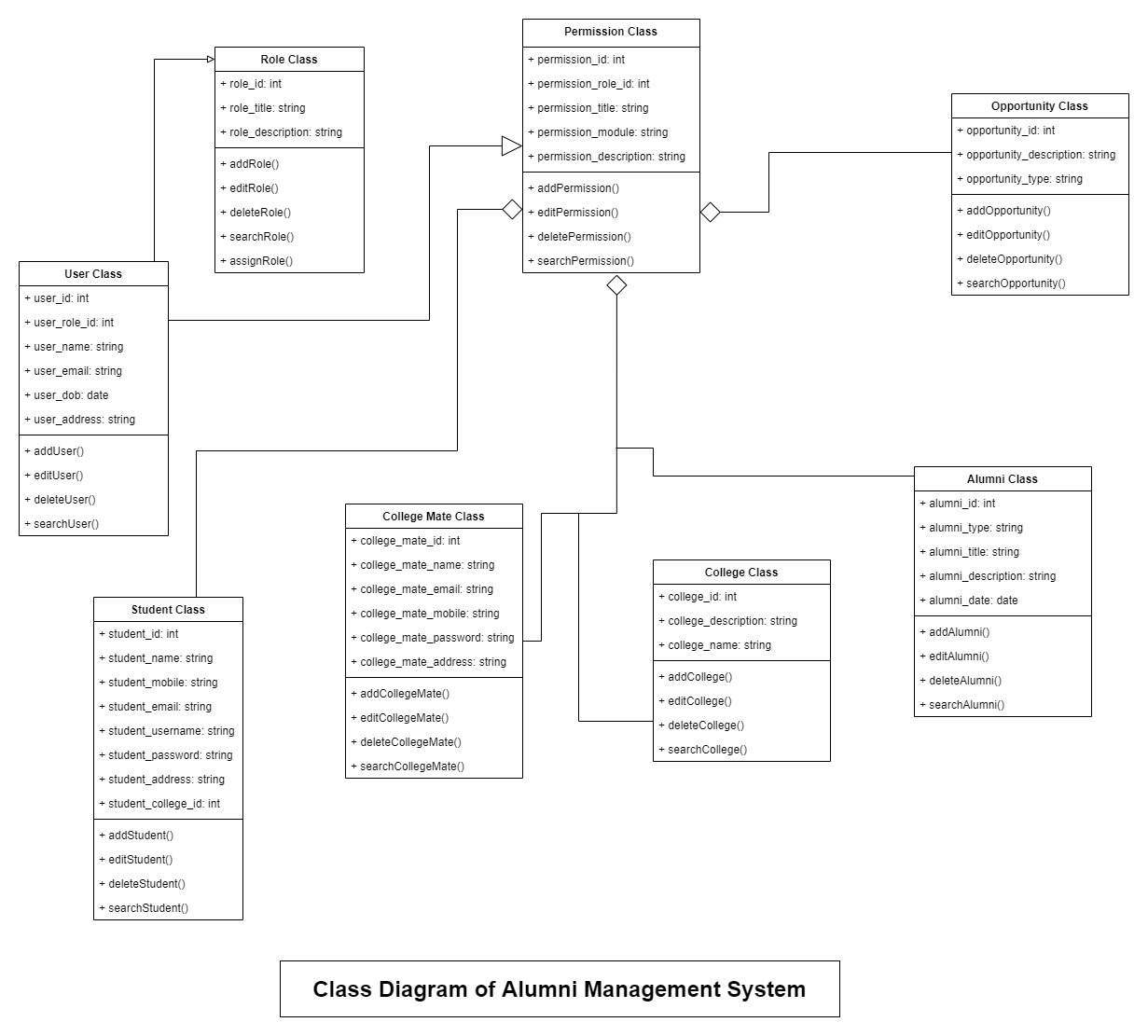
**Structured Chart**

****

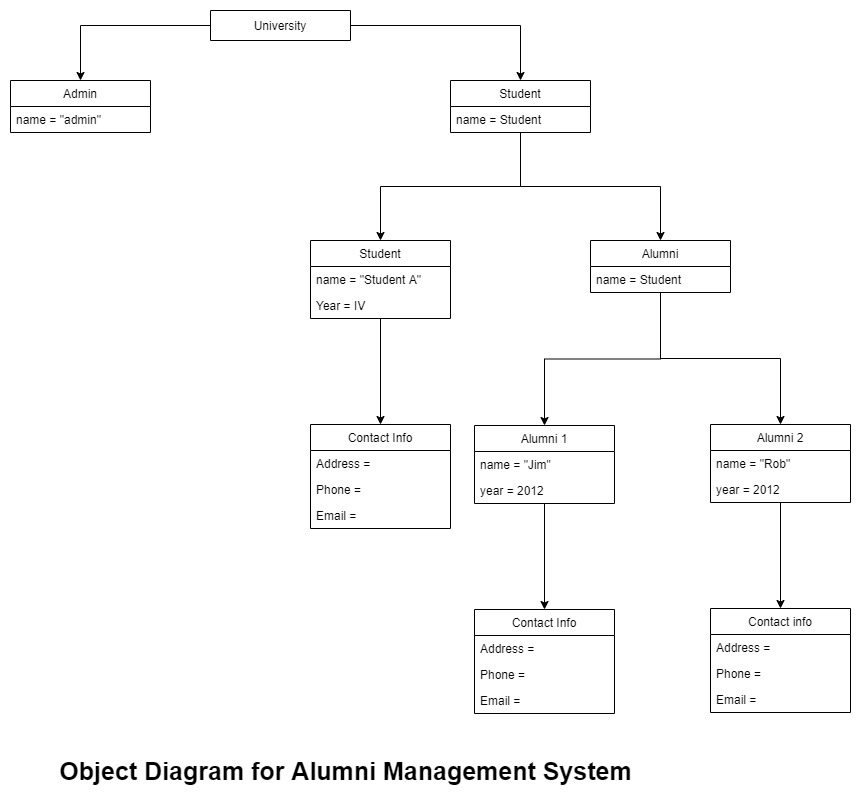
**Use Case Diagram**

****

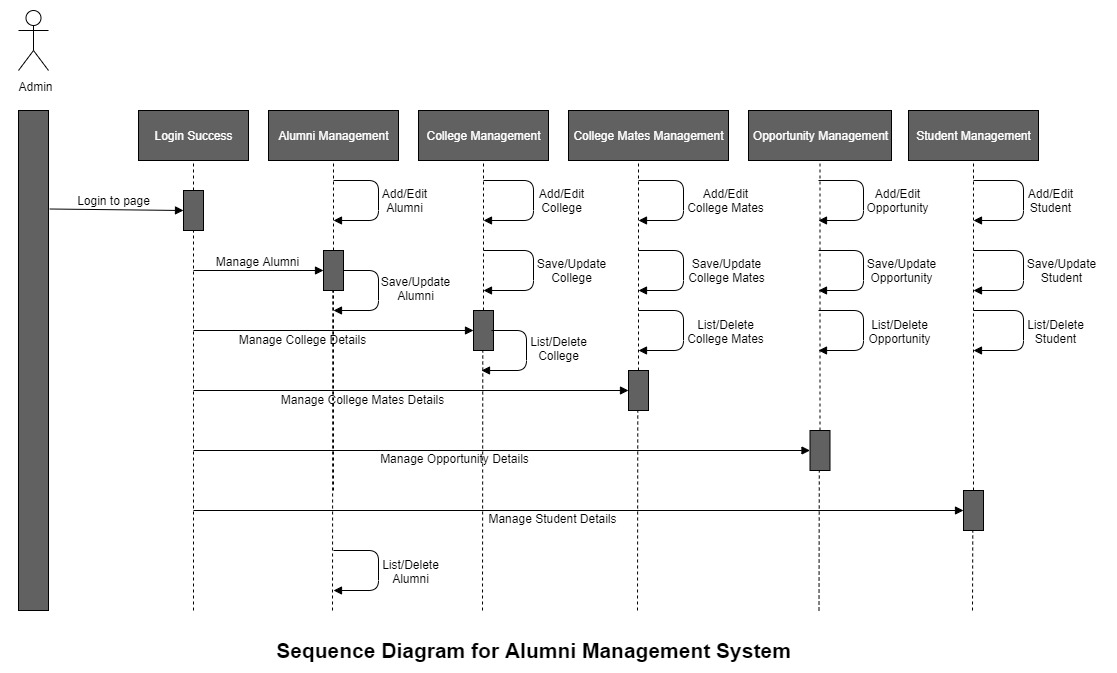
**Class Diagram**

****

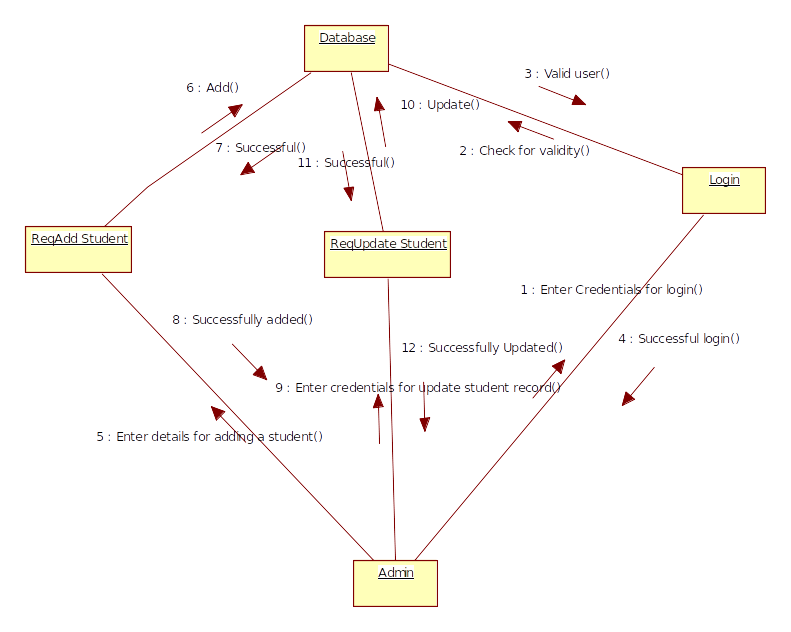
**Object Diagram**

****

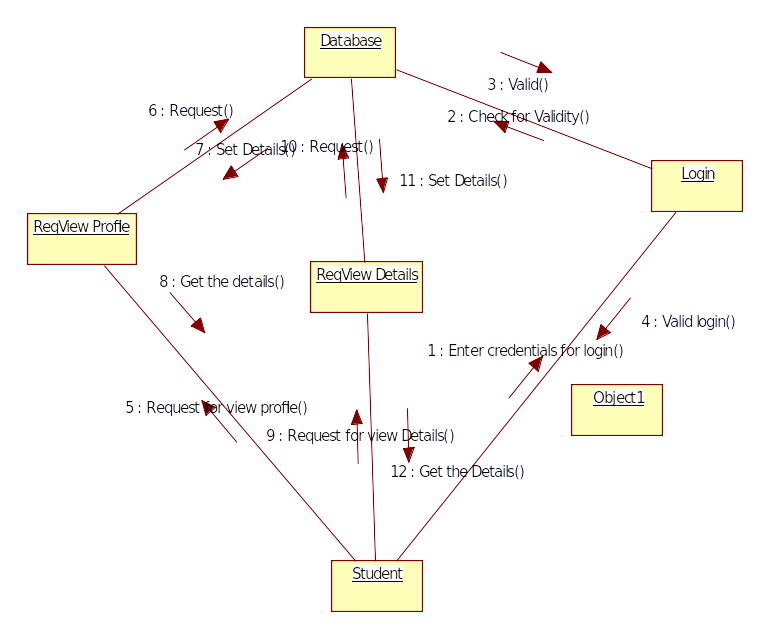
**Sequence Diagram**

****

**Collaboration Diagram**

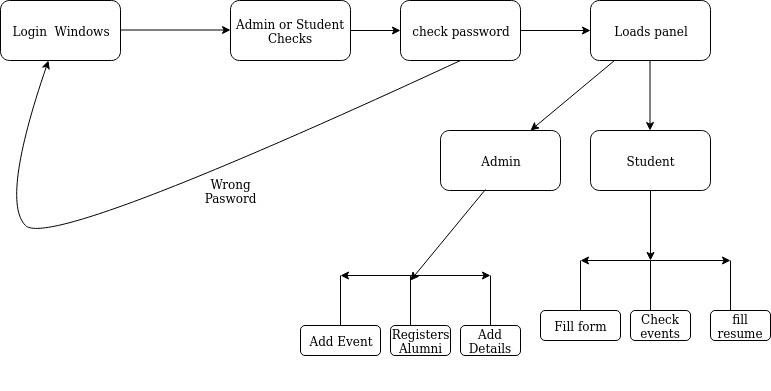
****

**Admin**

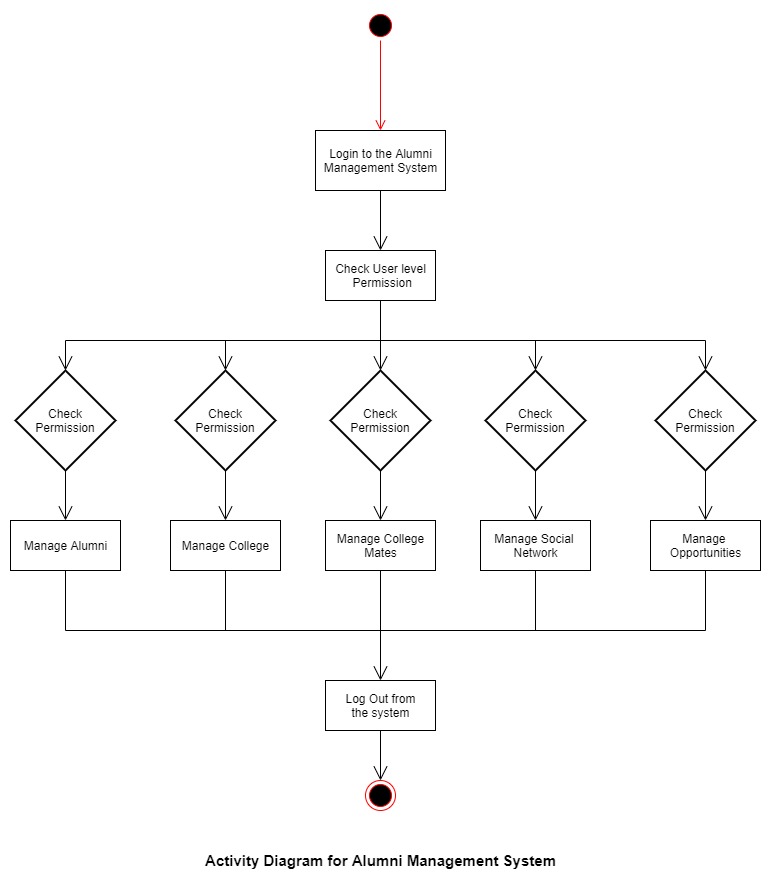
****

**Student**

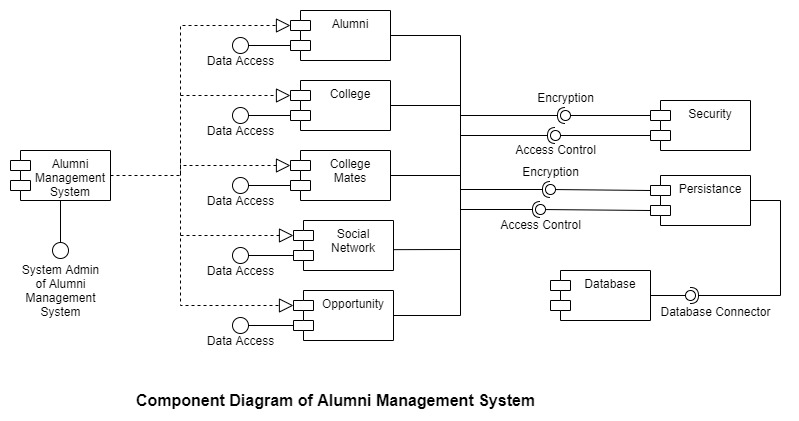
**State Chart Diagram**

****

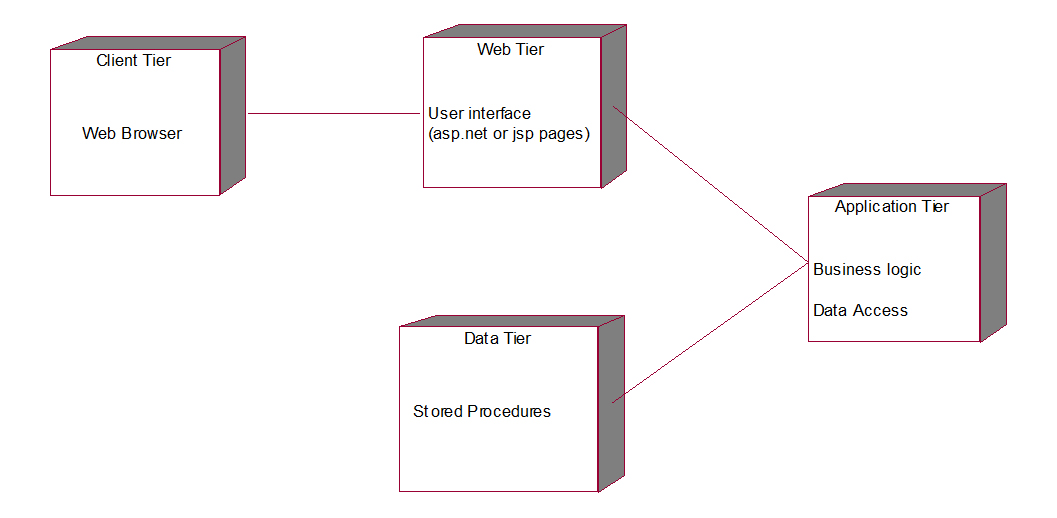
**Activity Diagram**

****

**Component Diagram**

****

**Deployment Diagram.**

****

**10. To perform various testing using the testing tool unit testing, integration testing**

10A. Unit Testing

public class Complex {

int real\_part; int imaginary\_part;

public Complex(int r, int i) {

real\_part=r;

imaginary\_part=i;

}

public Complex() {

real\_part=0;

imaginary\_part=0;

}

public boolean Equal(Complex c) {

boolean result = false;

if ((real\_part==c.get\_r()) && (imaginary\_part==c.get\_i())) result=true;

return result;

}

public Complex Add(Complex c) {

Complex result = new

Complex(c.get\_r()+real\_part,c.get\_i()+imaginary\_part);

return result;

}

public int get\_r() { return real\_part;}

public int get\_i() { return imaginary\_part; }

}

JUnit test cases for the above complex class

package javaapplication4;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Assert;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

public class ComplexTest {

Complex c1;

Complex c2;

public ComplexTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

c1 = new Complex(7,3);

c2 = new Complex(12,6);

}

@After

public void tearDown() {

}

/\*\*

\* Test of Equal method, of class Complex.

\*/

@Test

public void testEqual() {

System.out.println("Equal");

assertTrue(!c2.Equal(c1));

assertTrue(c1.Equal(new Complex(7,3)));

}

/\*\*

\* Test of Add method, of class Complex.

\*/

@Test

public void testAdd() {

System.out.println("Add");

Complex result = c1.Add(new Complex(5,3));

assertEquals(result.get\_r(),c2.get\_r());

assertEquals(result.get\_i(),c2.get\_i());

}

/\*\*

\* Test of get\_r method, of class Complex.

\*/

@Test

public void testGet\_r() {

System.out.println("get\_r");

Complex instance = new Complex(5,10);

int expResult = 5;

int result = instance.get\_r();

assertEquals(expResult, result);

}

/\*\*

\* Test of get\_i method, of class Complex.

\*/

@Test

public void testGet\_i() {

System.out.println("get\_i");

Complex instance = new Complex(5,10);

int expResult = 10;

int result = instance.get\_i();

assertEquals(expResult, result);

}

}

Testsuite: javaapplication4.ComplexTest

Add

Equal

get\_i

get\_r

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.09 sec

------------- Standard Output ---------------

Add

Equal

get\_i

get\_r

------------- ---------------- ---------------

10B. Integration Testing

CalculatorService.java

public interface CalculatorService {

public double add(double input1, double input2);

public double subtract(double input1, double input2);

public double multiply(double input1, double input2);

public double divide(double input1, double input2);

}

MathApplication.java

public class MathApplication {

private CalculatorService calcService;

public void setCalculatorService(CalculatorService calcService){

this.calcService = calcService;

}

public double add(double input1, double input2){

return calcService.add(input1, input2);

}

public double subtract(double input1, double input2){

return calcService.subtract(input1, input2);

}

public double multiply(double input1, double input2){

return calcService.multiply(input1, input2);

}

public double divide(double input1, double input2){

return calcService.divide(input1, input2);

}

}

Integration Test cases

MathApplicationTest.java

import static org.mockito.Mockito.when;

import org.junit.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.runners.MockitoJUnitRunner;

// @RunWith attaches a runner with the test class to initialize the test data

@RunWith(MockitoJUnitRunner.class)

public class MathApplicationTest {

//@InjectMocks annotation is used to create and inject the mock object

@InjectMocks

MathApplication mathApplication = new MathApplication();

//@Mock annotation is used to create the mock object to be injected

@Mock

CalculatorService calcService;

@Test

public void testAdd(){

//add the behavior of calc service to add two numbers

when(calcService.add(10.0,20.0)).thenReturn(30.00);

//test the add functionality

Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);

}

}

output

Testsuite: javaapplication4.MathApplicationTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.416 sec

test:

BUILD SUCCESSFUL (total time: 2 seconds)