**CHAPTER 1**

**INTRODUCTION**

**1.1 Purpose**

This system can be used as an application for the Alumni Information Database to manage the college information and student’s information. The system is an online application that can be accessed throughout the organization and outside customers as well with proper login provided, which will give better service to the customers.

**1.2 Scope**

This system can be used as the Office of Alumni and College Relations seeks to protect the privacy of its alumni and friends, and thus, endeavors to safeguard the use of information in its custody. To that end, the Office of Alumni and College Relations provides constituent information to requestors only under the conditions.

**1.3 Overview**

Overall description consists of background of the entire specific requirement. It also gives explanation about actor and function which is used. It gives explanation about architecture diagram and it also gives what we are assumed and dependencies. It also support specific requirement and also it support functional requirement, supplementary requirement other than actor which is used. It also gives index and appendices. It also gives explanation about any doubt and queries.

Once a student graduates from the institute, his/her professional life or career begins, with higher education playing an important role in establishing himself/herself in the profession. In respect of College, it has been our experience that from the very beginning, the alumni have maintained personal contacts with one another, rather than use the channel of Alumni Association.

The advancements in information technology have certainly helped in creating new resources such as alumni web pages, list servers etc., so as to permit greater interactions between the alumni.

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 Existing System**

The Existing system is a computerized system but which is maintained at individual databases i.e. in excels sheets, it’s a time delay process. And maintaining all the records in Excel sheets is difficult. If they want any record they have to search all the records. It doesn’t provide multiple user accessibility and also doesn’t have different user privileges. So the system is not accessible for all the employees of the organization.

**Limitations in Existing System**

* The current system is not completely complete computerized and manual system in entering students and staff data and handling it.
* There is no centralized database maintenance
* There is no easy access to the particular student’s record
* The student cannot easily navigate through the database

**Advantages over Existing System**

* It is completely automated system in handling the college database
* This system provides centralized database maintenance
* This system provides easy access to the particular students account or his complete details
* This system provides student to easily navigate through the application for more information in a most secure manner.

**2.2 Proposed System**

The Proposed system is a computerized system but which is maintained at Centralized databases i.e. in automated forms it’s a very fast process. And maintaining all the records in online systems database which makes it very easy to access and retrieve data from the database. If they want any record they can easily search all the records. It provides multiple user accessibility and also has different user privileges. So the system is accessible for all the employees of the organization

**CHAPTER 3**

**HARDWARE AND SOFTWARE REQUIREMENT**

**3.1 Software Requirements**

**Server Side Requirements: -**

Operating System : Windows XP/2003 or Linux/Solaris

User Interface : HTML, CSS

Client-side Scripting : JavaScript

Programming Language : Java

Web Applications : JDBC, JNDI, Servlets, JSP

IDE/Workbench : My Eclipse

Database : MySQL

Server Deployment : Tomcat

**Client Side Requirements: -**

Operating System : Any Operating System

Browser : Any Browser (IE, Opera, Mozilla, etc.)

**3.2 Hardware Requirements**

**Server Side Requirements: -**

Processor : Pentium IV or above

Hard Disk : 40GB

RAM : 256MB

**Client Side Requirements: -**

Processor : Intel(R)

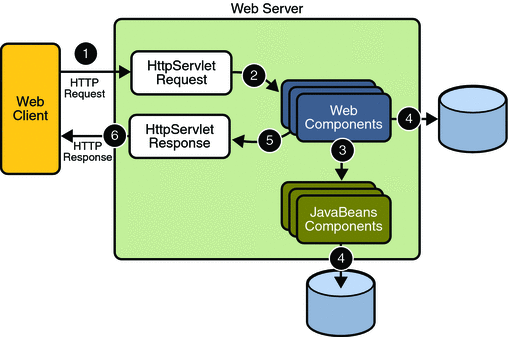
Hard Disk : 20GB

RAM : 128MB

**CHAPTER 4**

**SYSTEM DESIGN**

**4.1 Architecture Diagram**

****

**CHAPTER 5**

**AUTHENTICATION**

**5.1 Functional Description**

* Login to the system through the first page of the application*.*
* Change the password after login to the application.
* See his/her details and change it.
* Help from the system.

**CHAPTER 6**

**GENERAL OPERATIONS**

**6.1 Student**

* User can Register to Alumni
* Students can login to the system
* Update the Profile
* Students can see all student’s information in the alumni
* They can see the events details
* They can also send messages to their friends
* Students also can see the campus interview details

**6.2 Admin Users**

* Has full access to all the modules of this system.
* Responsible for the accounts of all students.
* Update, modify or delete event details
* Update, modify or delete campus interviews details
* Prepares and submits also Daily reports, user reports, event reports, etc.

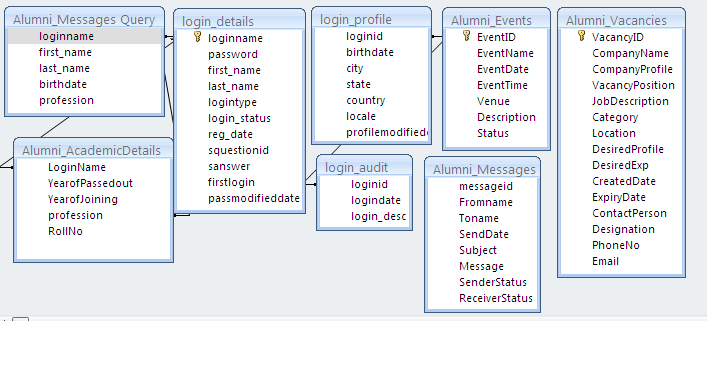
**6.3 Normal Users**

* Has restricted access. i.e., Normal users have access to some of the modules only i.e. user can search the list of all their friends.
* They cannot send messages to their batch mates

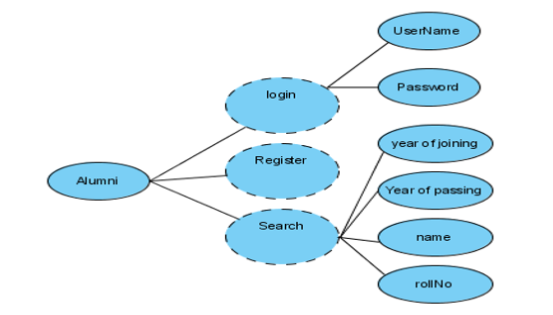
**CHAPTER 7**

**DETAIL DESIGN**

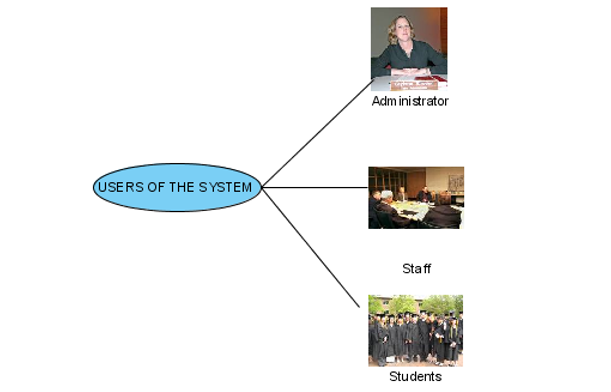
**7.1 E - R Diagram**

****

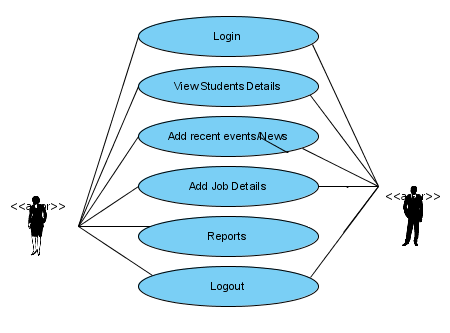
**7.2 Use-case Diagram**



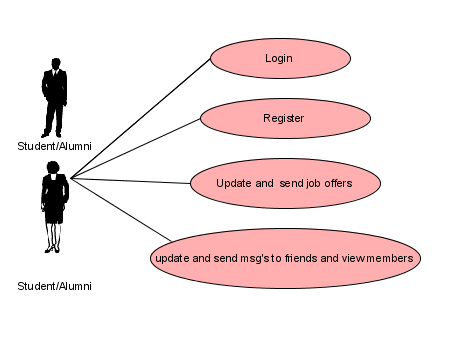
**7.3 users of the system**



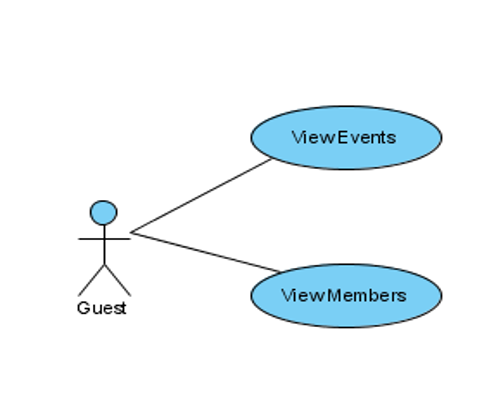
**7.4 Administrator Use Case**

****

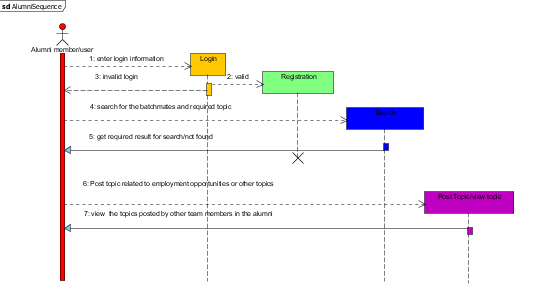
**7.5 Student/Alumni Use Case**



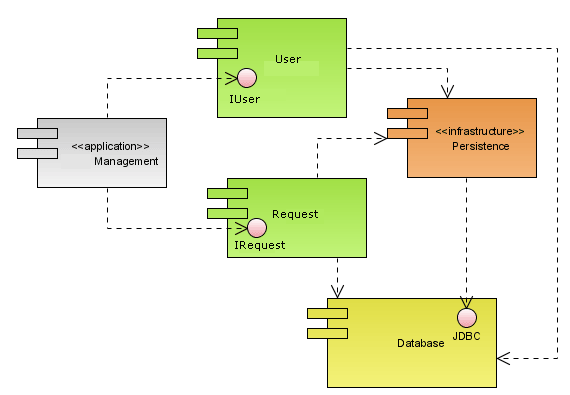
**7.6 Guest Use Case**

****

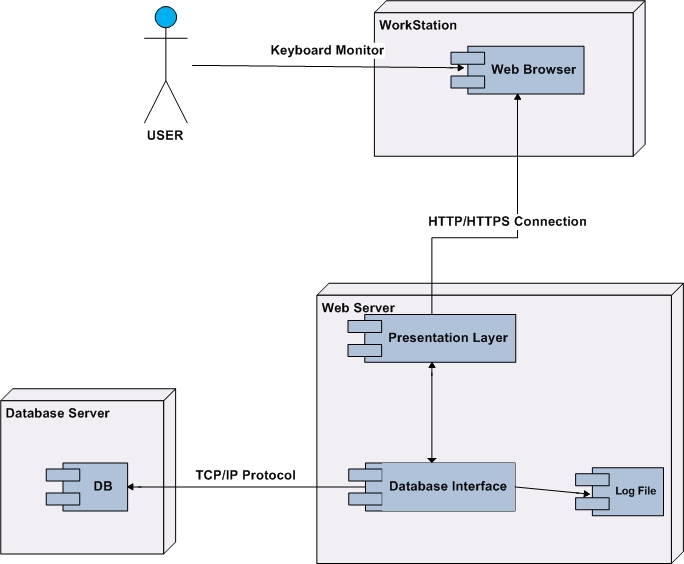
**7.7 Sequence Diagram for Alumni**



**7.8 Component Diagram**

****

**7.9 Deployment Diagram**

****

**CHAPTER 8**

**IMPLEMENTATION**

**8.1 ADMIN: -**

The one who leading or maintaining the Alumni application is Admin. In this Alumni application admin is playing a main role to maintain a system by giving the authorization to alumni student and pursuing who registered online to application by verifying whether online registered user are belongs to college or not.

In this Alumni application admin can view all alumni student who registered online. Admin also can delete the alumni meet, Campus Interview notifications. Admin can delete the events once it’s done.

**/\*\* ADMIN Processes \*/**

public String logIn(LoginTO to) {

String checkLogin = "SELECT password FROM admin WHERE username='"

+ to.getUsername() + "' ";

try {

st = con.createStatement();

rs = st.executeQuery(checkLogin);

if (rs.next()) {

if (rs.getString(1).equals(to.getPassword())) {

status = "SUCCESS";

} else {

status = "sorry password invalid";

}

} else {

status = "sorry username invalid";

}

} catch (SQLException e) {

System.*err*.println(e);

}

return status;

}

public List<Alumnidetails> alumnis() {

List<Alumnidetails> list = new ArrayList<Alumnidetails>();

String query = "select \* from Alumnidetails";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

Alumnidetails to = new Alumnidetails();

to.setAlumniId(rs.getInt(1));

to.setFirstname(rs.getString(2));

to.setUsnNumber(rs.getString(4));

to.setDob(rs.getString(5));

to.setSex(rs.getString(6));

to.setGraduation(rs.getString(7));

to.setBatch(rs.getString(8));

to.setEmailId(rs.getString(9));

to.setPhone(rs.getLong(10));

to.setStatus(rs.getString(14));

to.setCompany(rs.getString(15));

list.add(to);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

}

public void authorizeAlumni(int id) {

try {

con.createStatement().executeUpdate(

"update alumnidetails set status = 'Authorized' where alumni\_id = "

+ id + " ");

} catch (Exception e) {

e.printStackTrace();

}

}

public void adelete(int aid) {

try {

con.createStatement().executeUpdate(

"delete from alumnidetails where alumni\_id = " + aid + " ");

} catch (Exception e) {

e.printStackTrace();

}

}

public List<Studentdetails> students () {

List<Studentdetails> list = new ArrayList<Studentdetails>();

String query = "select \* from Studentdetails";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

Studentdetails to = new Studentdetails();

to.setStudentId(rs.getInt(1));

to.setFirstname(rs.getString(2));

to.setUsnNumber(rs.getString(4));

to.setDob(rs.getString(5));

to.setSex(rs.getString(6));

to.setGraduation(rs.getString(7));

to.setPercentage(rs.getString(8));

to.setBatch(rs.getString(9));

to.setEmailId(rs.getString(10));

to.setPhone(rs.getLong(11));

to.setAddress(rs.getString(13));

to.setStatus(rs.getString(14));

list.add(to);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

}

public void authorizes (int id) {

try {

con.createStatement().executeUpdate(

"update studentdetails set status = 'Authorized' where student\_id = "

+ id + " ");

} catch (Exception e) {

e.printStackTrace();

}

}

public void sdelete(int sid) {

try {

con.createStatement().executeUpdate(

"delete from studentdetails where student\_id = " + sid

+ " ");

} catch (Exception e) {

e.printStackTrace();

}

}

public String normalEvent(NormalEvent to) {

String regQuery = "INSERT INTO normal\_event(event\_name, start\_date, end\_date, reg\_fee, event\_descripction) VALUES(?,?,?,?,?)";

try {

PreparedStatement pst = con.prepareStatement(regQuery);

pst.setString(1, to.getEventName());

pst.setString(2, to.getStartDate());

pst.setString(3, to.getEndDate());

pst.setLong(4, to.getRegFee());

pst.setString(5, to.getEventDescripction());

count = pst.executeUpdate();

if (count > 0) {

status = "Added Successfully";

} else {

status = "Failed to add";

}

} catch (Exception e) {

System.*err*.println(e);

}

return status;

}

public List<NormalEvent> viewnormalevent() {

List<NormalEvent> list = new ArrayList<NormalEvent>();

String query = "select \* from normal\_event";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

NormalEvent to = new NormalEvent();

to.setEventId(rs.getInt(1));

to.setEventName(rs.getString(2));

to.setStartDate(rs.getString(3));

to.setEndDate(rs.getString(4));

to.setRegFee(rs.getInt(5));

to.setEventDescripction(rs.getString(6));

list.add(to);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

}

public void ndelete(int eid) {

try {

con.createStatement().executeUpdate(

"delete from Normal\_Event where event\_id = " + eid + " ");

} catch (Exception e) {

e.printStackTrace();

}

}

public void rdelete(int rid) {

try {

con.createStatement().executeUpdate(

"delete from alumnirecruitment where id = " + rid + " ");

} catch (Exception e) {

e.printStackTrace();

}

}

**8.2 ALUMNI: -**

This alumni application is design to get alumni student know about all event information of college and get participated in the event. The main thing alumni can do in this app is to post the job vacancies to their junior to help to get jobs. This is the one biggest advantage of alumni application.

Alumni user can view his profile and he can edit the information through online portal app.

**/\*\*ALUMNI Processes\*\*/**

public String register(Alumnidetails to) {

String regQuery = "INSERT INTO alumnidetails(firstname, lastname, usn\_number, dob, sex, graduation, batch, email\_id, phone, password, address, current\_location, status, company, designation) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";

try {

PreparedStatement pst = con.prepareStatement(regQuery);

pst.setString(1, to.getFirstname());

pst.setString(2, to.getLastname());

pst.setString(3, to.getUsnNumber());

pst.setString(4, to.getDob());

pst.setString(5, to.getSex());

pst.setString(6, to.getGraduation());

pst.setString(7, to.getBatch());

pst.setString(8, to.getEmailId());

pst.setLong(9, to.getPhone());

pst.setString(10, to.getPassword());

pst.setString(11, to.getAddress());

pst.setString(12, to.getCurrentLocation());

pst.setString(13, "Waiting");

pst.setString(14, to.getCompany());

pst.setString(15, to.getDesignation());

count = pst.executeUpdate();

if (count > 0) {

status = "Alumni Registered Successfully";

} else {

status = "Failed to Register Alumni";

}

} catch (Exception e) {

System.*err*.println(e);

}

return status;

}

public String alumnilogIn(LoginTO to) {

try {

PreparedStatement ps = con

.prepareStatement("SELECT password, status FROM alumnidetails WHERE email\_id = ?");

ps.setString(1, to.getUsername());

ResultSet rs = ps.executeQuery();

if (rs.next()) {

if (to.getPassword().equals(rs.getString(1))) {

if (rs.getString(2).equals("Waiting")) {

status = "Please wait for Admin permission";

} else {

status = "SUCCESS";

}

} else {

status = "password invalid";

}

} else {

status = "username invalid";

}

} catch (SQLException e) {

e.printStackTrace();

}

return status;

}

public Alumnidetails alumniProfile(String email) {

Alumnidetails to = new Alumnidetails();

String query = "select \* from Alumnidetails where email\_id = '" + email

+ "' ";

try {

st = con.createStatement();

rs = st.executeQuery(query);

if (rs.next()) {

to.setAlumniId(rs.getInt(1));

to.setFirstname(rs.getString(2));

to.setLastname(rs.getString(3));

to.setUsnNumber(rs.getString(4));

to.setDob(rs.getString(5));

to.setSex(rs.getString(6));

to.setGraduation(rs.getString(7));

to.setBatch(rs.getString(8));

to.setEmailId(rs.getString(9));

to.setPhone(rs.getLong(10));

to.setAddress(rs.getString(12));

to.setCurrentLocation(rs.getString(13));

to.setStatus(rs.getString(14));

to.setCompany(rs.getString(15));

to.setDesignation(rs.getString(16));

}

} catch (SQLException e) {

System.*err*.println(e);

}

return to;

}

public int updatealumni(Alumnidetails to) {

int i = 0;

try {

String updateQuery = "UPDATE alumnidetails SET firstname='"

+ to.getFirstname() + "', lastname='" + to.getLastname()

+ "', graduation='" + to.getGraduation() + "', email\_id='"

+ to.getEmailId() + "', phone='" + to.getPhone()

+ "', current\_Location='" + to.getCurrentLocation()

+ "', company='" + to.getCompany() + "', designation='"

+ to.getDesignation() + "' WHERE alumni\_id="

+ to.getAlumniId() + "";

con.createStatement().executeUpdate(updateQuery);

i = 1;

} catch (SQLException e) {

System.*err*.println(e);

i = 0;

}

return i;

}

public int alumnirecruitmentevent(AlumniRecruitmentEvent to) {

String regQuery = "INSERT INTO alumnirecruitment(job\_profile, company\_name, interview\_date, salary, job\_description, alumni\_name) VALUES(?,?,?,?,?,?)";

try {

PreparedStatement pst = con.prepareStatement(regQuery);

pst.setString(1, to.getJobProfile());

pst.setString(2, to.getCompanyName());

pst.setString(3, to.getInterviewDate());

pst.setInt(4, to.getSalary());

pst.setString(5, to.getJobDescription());

pst.setString(6, to.getAlumniName());

count = pst.executeUpdate();

} catch (Exception e) {

System.*err*.println(e);

}

return count;

}

public List<AlumniRecruitmentEvent> alumniRecruitmentEvent() {

List<AlumniRecruitmentEvent> list = new ArrayList<AlumniRecruitmentEvent>();

String query = "select \* from alumnirecruitment";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

AlumniRecruitmentEvent event = new AlumniRecruitmentEvent();

event.setId(rs.getInt(1));

event.setJobProfile(rs.getString(2));

event.setCompanyName(rs.getString(3));

event.setInterviewDate(rs.getString(4));

event.setSalary(rs.getInt(5));

event.setJobDescription(rs.getString(6));

event.setAlumniName(rs.getString(7));

list.add(event);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

}

**8.3 STUDENT: -**

Student can get job reference from seniors through alumni application. Students can view their profiles and they can edit the profile. Also students can view and participated for all college events like college fest, Anniversary function, Alumni meet, Campus interviews and interview notifications.

**/\*\* STUDENT Processes\* \*/**

public String studentregister(Studentdetails to) {

String regQuery = "INSERT INTO studentdetails(firstname, lastname, usn\_number, dob, sex, graduation, percentage, batch, email\_id, phone, password, address, status) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)";

try {

PreparedStatement pst = con.prepareStatement(regQuery);

pst.setString(1, to.getFirstname());

pst.setString(2, to.getLastname());

pst.setString(3, to.getUsnNumber());

pst.setString(4, to.getDob());

pst.setString(5, to.getSex());

pst.setString(6, to.getGraduation());

pst.setString(7, to.getPercentage());

pst.setString(8, to.getBatch());

pst.setString(9, to.getEmailId());

pst.setLong(10, to.getPhone());

pst.setString(11, to.getPassword());

pst.setString(12, to.getAddress());

pst.setString(13, "Waiting");

count = pst.executeUpdate();

if (count > 0) {

status = "Student Registered Successfully";

} else {

status = "Failed to Register Student";

}

} catch (Exception e) {

System.*err*.println(e);

}

return status;

}

public String studentlogIn(LoginTO to) {

try {

PreparedStatement ps = con

.prepareStatement("SELECT password, status FROM studentdetails WHERE email\_id = ?");

ps.setString(1, to.getUsername());

ResultSet rs = ps.executeQuery();

if (rs.next()) {

if (to.getPassword().equals(rs.getString(1))) {

if (rs.getString(2).equals("Waiting")) {

status = "Please wait for Admin permission";

} else {

status = "SUCCESS";

}

} else {

status = "password invalid";

}

} else {

status = "username invalid";

}

} catch (SQLException e) {

e.printStackTrace();

}

return status;

}

// done

public int studentlogIn(String username, String password) {

System.*out*.println(username);

int i = 0;

try {

PreparedStatement ps = con

.prepareStatement("SELECT password, status FROM studentdetails WHERE email\_id = ?");

ps.setString(1, username);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

if (password.equals(rs.getString(1))) {

i = 1;

}

if (rs.getString(2).equals("Waiting")) {

i = 2;

}

}

} catch (SQLException e) {

e.printStackTrace();

}

return i;

}

public Studentdetails setudenProfile(String emailId) {

Studentdetails to = new Studentdetails();

String query = "select \* from studentdetails where email\_id = '"

+ emailId + "' ";

try {

st = con.createStatement();

rs = st.executeQuery(query);

if (rs.next()) {

to.setStudentId(rs.getInt(1));

to.setFirstname(rs.getString(2));

to.setLastname(rs.getString(3));

to.setUsnNumber(rs.getString(4));

to.setDob(rs.getString(5));

to.setSex(rs.getString(6));

to.setGraduation(rs.getString(7));

to.setPercentage(rs.getString(8));

to.setBatch(rs.getString(9));

to.setEmailId(rs.getString(10));

to.setPhone(rs.getLong(11));

to.setAddress(rs.getString(13));

to.setStatus(rs.getString(14));

}

} catch (SQLException e) {

System.*err*.println(e);

}

return to;

}

public int updatestudent(Studentdetails to) {

int i = 0;

try {

String updateQuery = "UPDATE studentdetails SET firstname='"

+ to.getFirstname() + "', lastname='" + to.getLastname()

+ "', graduation='" + to.getGraduation()

+ "', percentage='" + to.getPercentage() + "', email\_id='"

+ to.getEmailId() + "', phone='" + to.getPhone()

+ "', address='" + to.getAddress() + "' WHERE student\_id="

+ to.getStudentId() + "";

con.createStatement().executeUpdate(updateQuery);

i = 1;

} catch (SQLException e) {

System.*err*.println(e);

i = 0;

}

return i;

}

public List<AlumniRecruitmentEvent> alumniRecruitmentEvent(String alumni) {

List<AlumniRecruitmentEvent> list = new ArrayList<AlumniRecruitmentEvent>();

String query = "select \* from alumnirecruitment where alumni\_name = '"

+ alumni + "' ";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

AlumniRecruitmentEvent event = new AlumniRecruitmentEvent();

event.setId(rs.getInt(1));

event.setJobProfile(rs.getString(2));

event.setCompanyName(rs.getString(3));

event.setInterviewDate(rs.getString(4));

event.setSalary(rs.getInt(5));

event.setJobDescription(rs.getString(6));

event.setAlumniName(rs.getString(7));

event.setStatus(rs.getString(8));

list.add(event);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

}

public RecruitmentEvent process (int id) {

RecruitmentEvent event = new RecruitmentEvent();

String query = "select \* from alumnirecruitment where id = " + id + " ";

try {

st = con.createStatement();

rs = st.executeQuery(query);

if (rs.next()) {

event.setJobProfile(rs.getString(2));

event.setCompanyName(rs.getString(3));

event.setInterviewDate(DateFormatter.*toUtilDate*(rs.getString(4)));

event.setSalary(rs.getInt(5));

event.setJobDescription(rs.getString(6));

event.setAlumniName(rs.getString(7));

con.createStatement().executeUpdate(

"update alumnirecruitment set status = 'Processed' where id= "

+ id + " ");

}

} catch (SQLException e) {

System.*err*.println(e);

}

return event;

}

public Map<Integer, String> alumniusers(String alumni) {

Map<Integer, String> map = new HashMap<Integer, String>();

String query = "select alumni\_id, firstname, lastname from alumnidetails where email\_id != '"

+ alumni + "' ";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

map.put(rs.getInt(1), rs.getString(2) + " " + rs.getString(3));

}

} catch (SQLException e) {

System.*err*.println(e);

}

return map;

}

public String alumniusers(MessageTO to) {

try {

con.createStatement().executeUpdate(

"insert into message(from\_id ,to\_id, date\_time, message) values("

+ to.getFromId() + ", " + to.getToId() + ", '"

+ to.getDate() + "', '" + to.getMessage() + "') ");

status = "message sent successfully";

} catch (SQLException e) {

System.*err*.println(e);

status = "message sending failed";

}

return status;

}

public List<MessageTO> messages(int alumniId, int id) {

List<MessageTO> list = new ArrayList<MessageTO>();

String query = "select \* from message where from\_id = " + alumniId

+ " and to\_id = " + id + " || from\_id = " + id

+ " and to\_id = " + alumniId + "";

try {

st = con.createStatement();

rs = st.executeQuery(query);

while (rs.next()) {

MessageTO to = new MessageTO();

to.setFromId(rs.getInt(2));

to.setToId(rs.getInt(3));

to.setDate(rs.getString(4));

to.setMessage(rs.getString(5));

if (id == rs.getInt(rs.getInt(2))) {

to.setStatus("user");

} else {

to.setStatus("me");

}

list.add(to);

}

} catch (SQLException e) {

System.*err*.println(e);

}

return list;

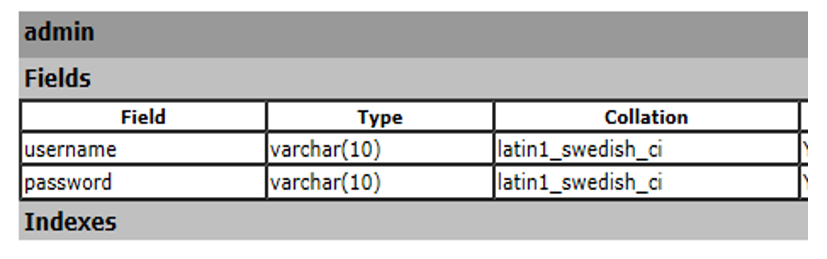
}

}

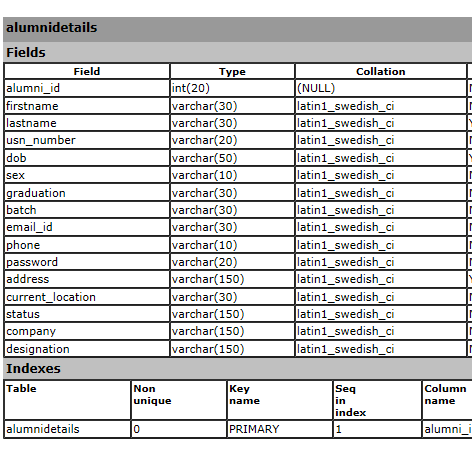
**CHAPTER 9**

**DATA DICTIONARY**

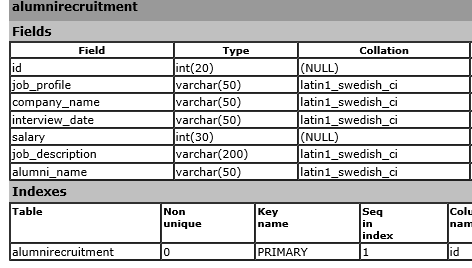
**9.1 ADMIN: -**

****

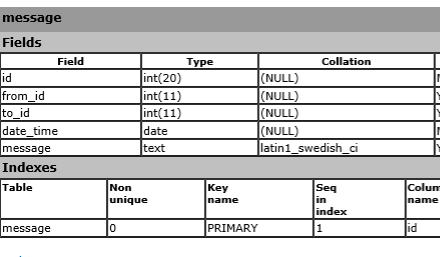
**9.2 ALUMNI DETAILS: -**



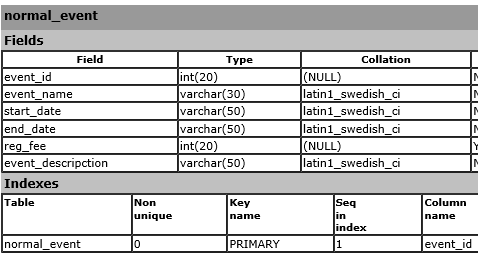
**9.3 ALUMNI RECRUITMENT: -**



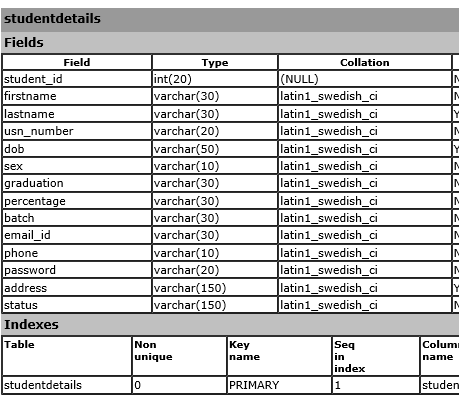
**9.4 MESSAGE: -**



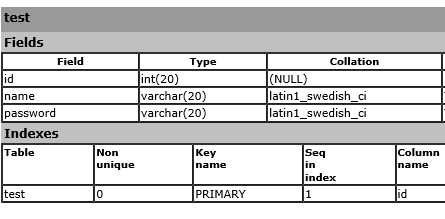
**9.5 NORMAL\_EVENT: -**



**9.6 STUDENT DETAILS: -**



**9.7 TEST: -**



**CHAPTER 10**

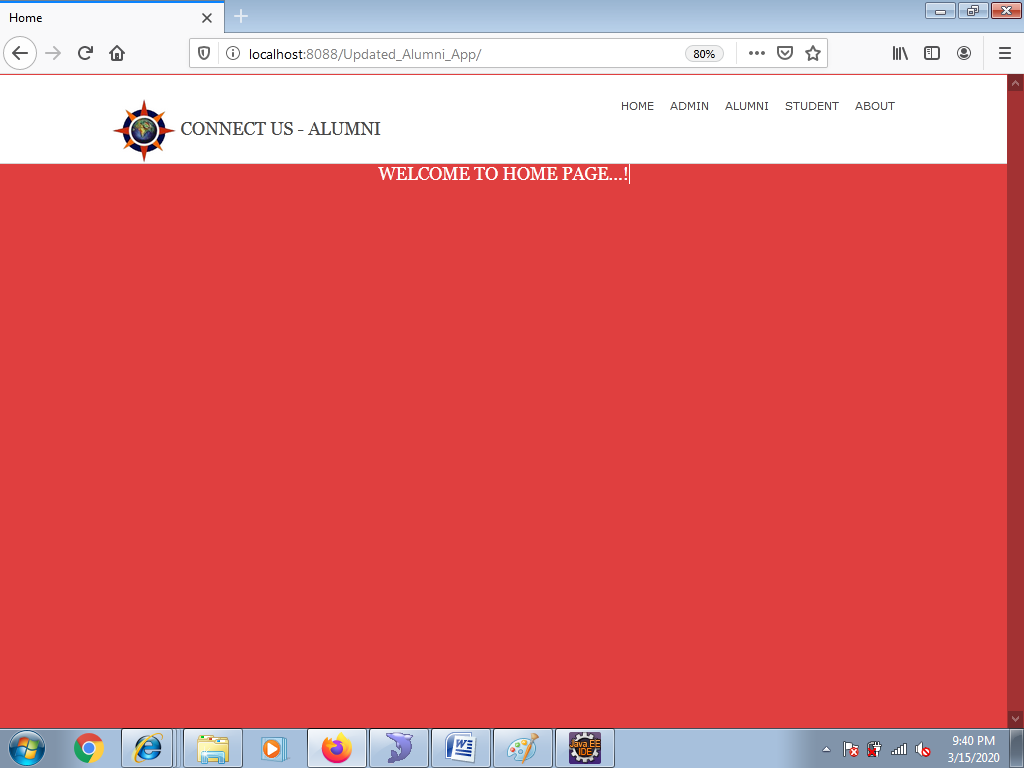
**FUTURE ENHANCEMENTS**

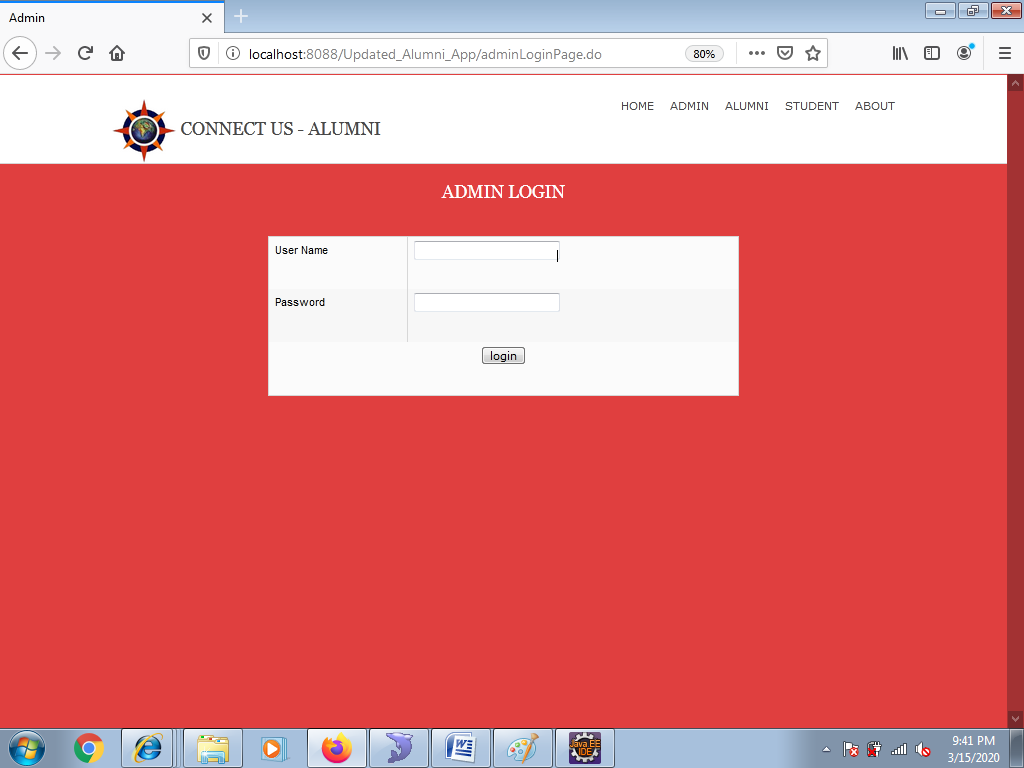
It is not possible to develop a system that makes all the requirements of the user. User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system are:

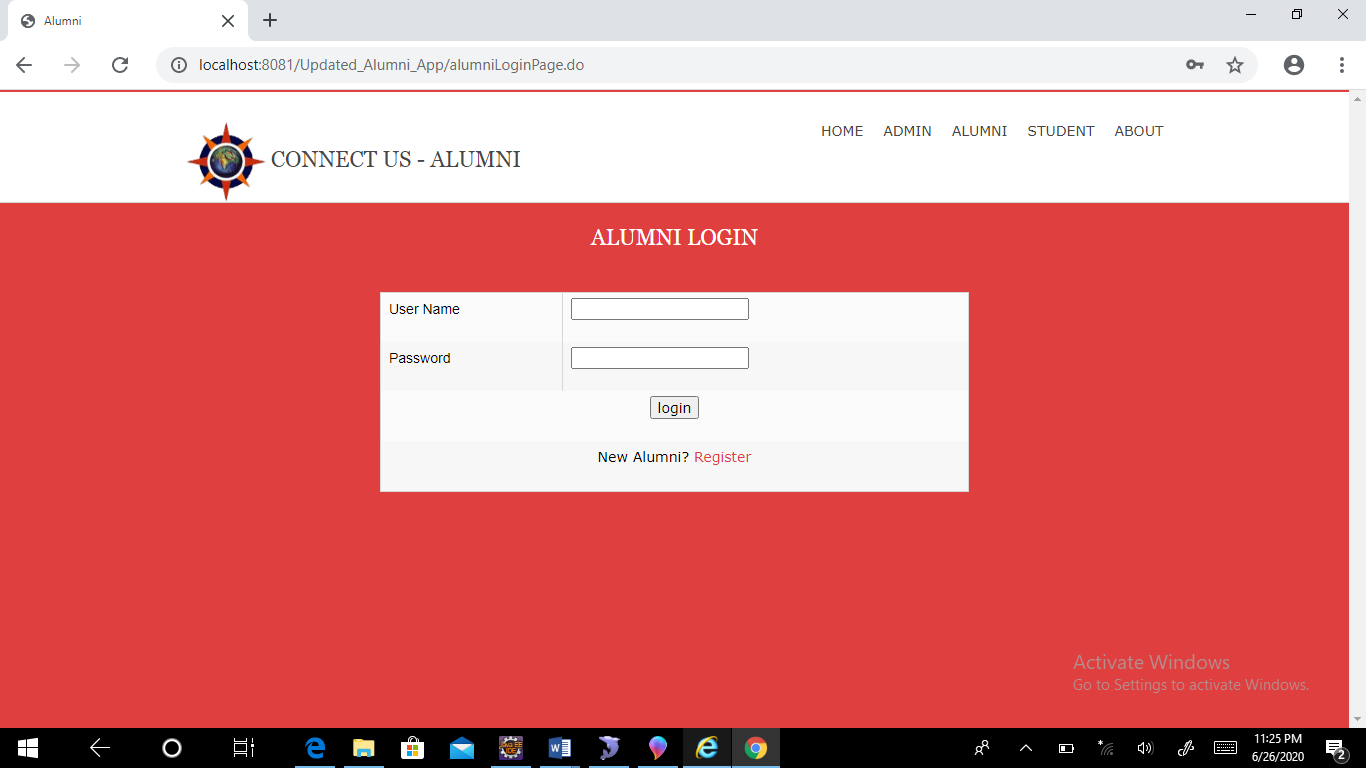
* As the technology emerges, it is possible to upgrade the system and can be adaptable to desired environment.
* Because it is based on object-oriented design, any further changes can be easily adaptable.
* Based on the future security issues, security can be improved using emerging technologies.
* Attendance module can be added
* sub admin module can be added

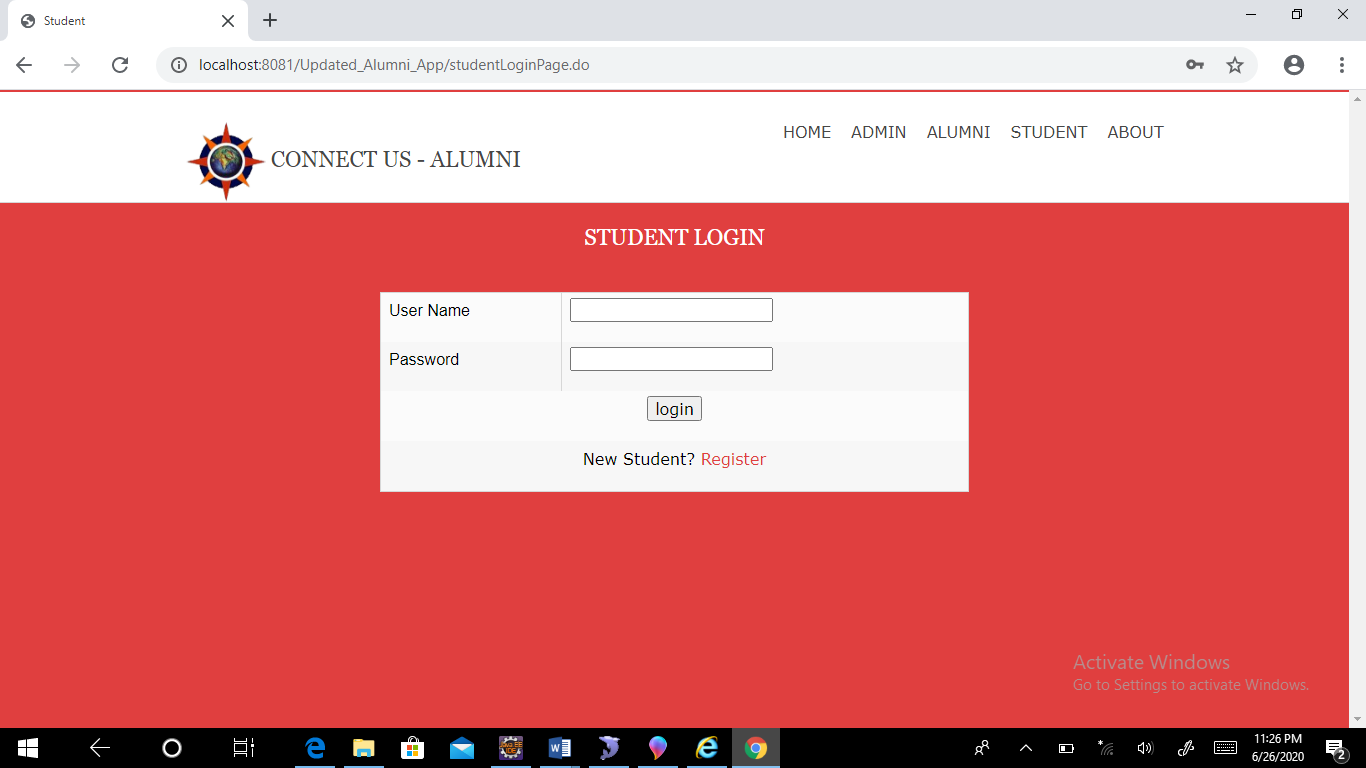
**CHAPTER 11**

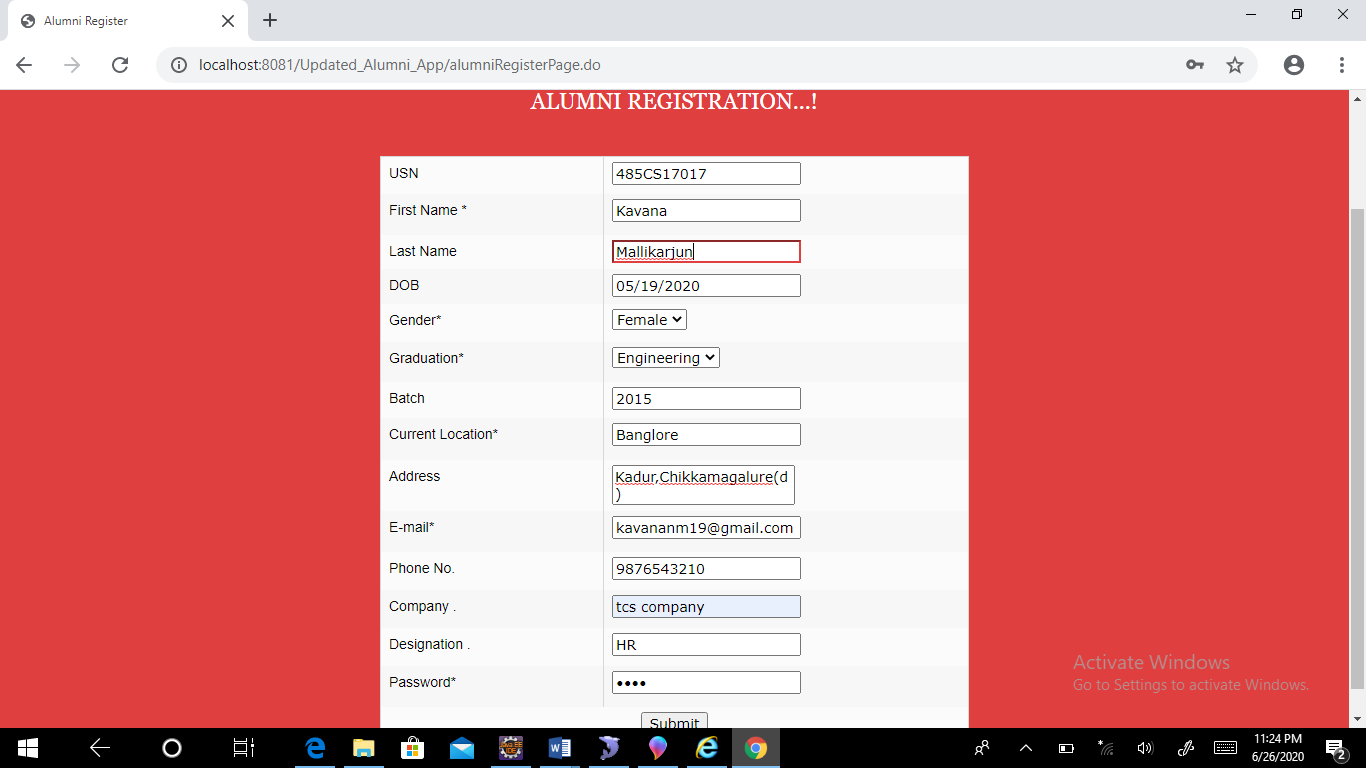
**SNAPSHOTS**

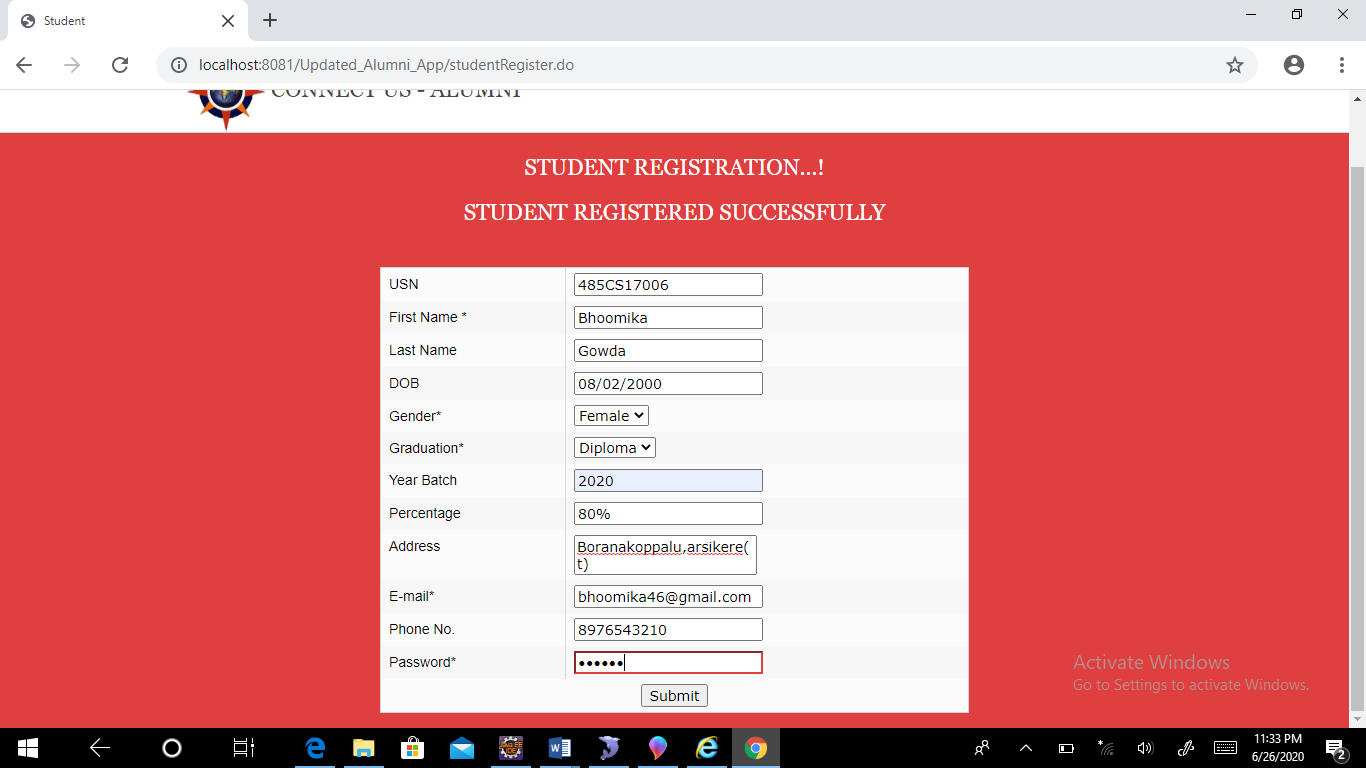


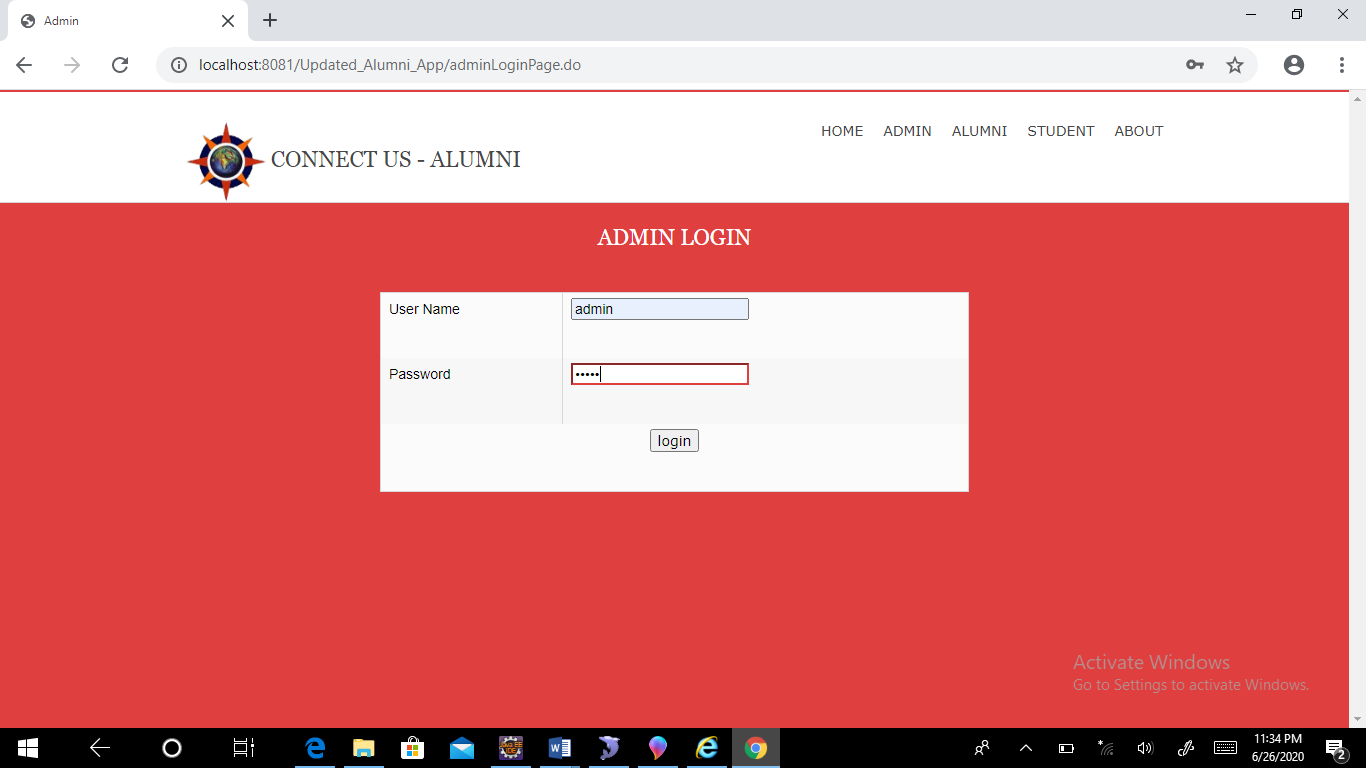


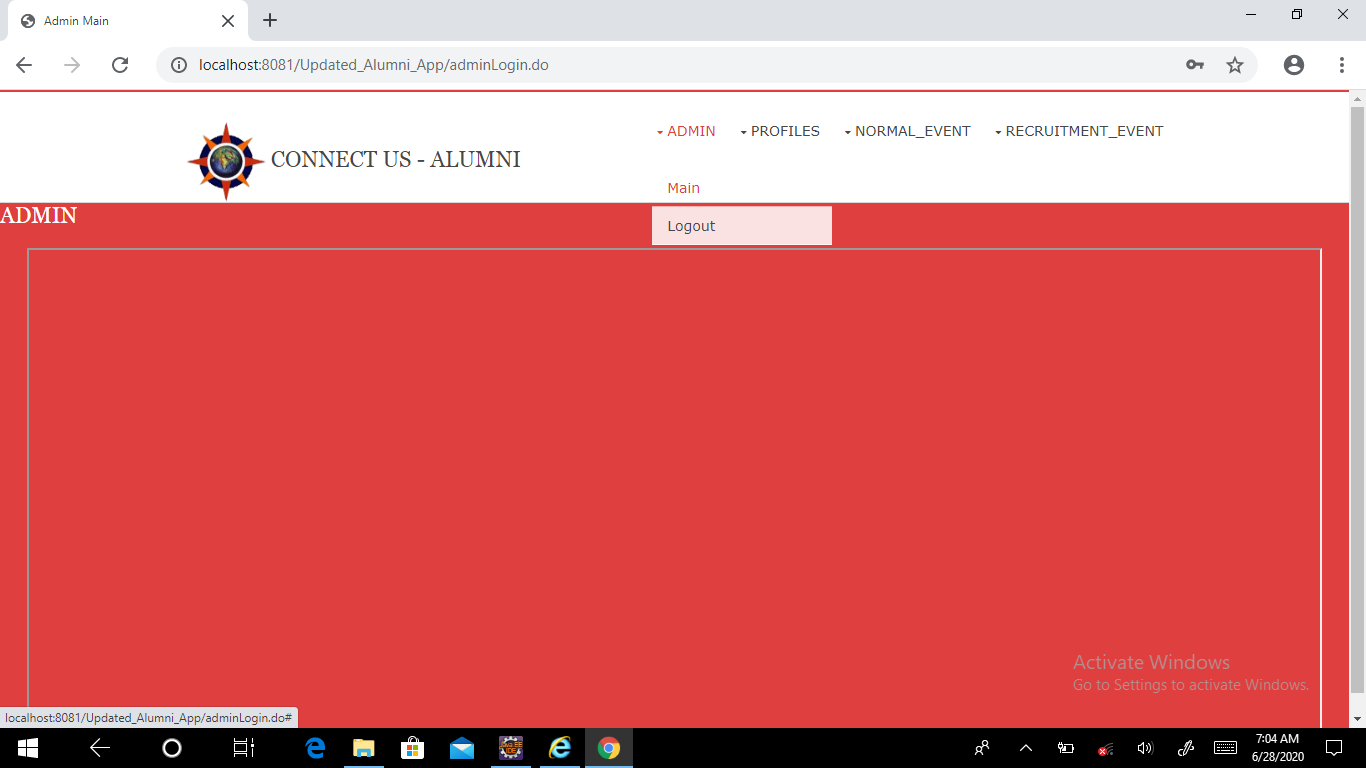


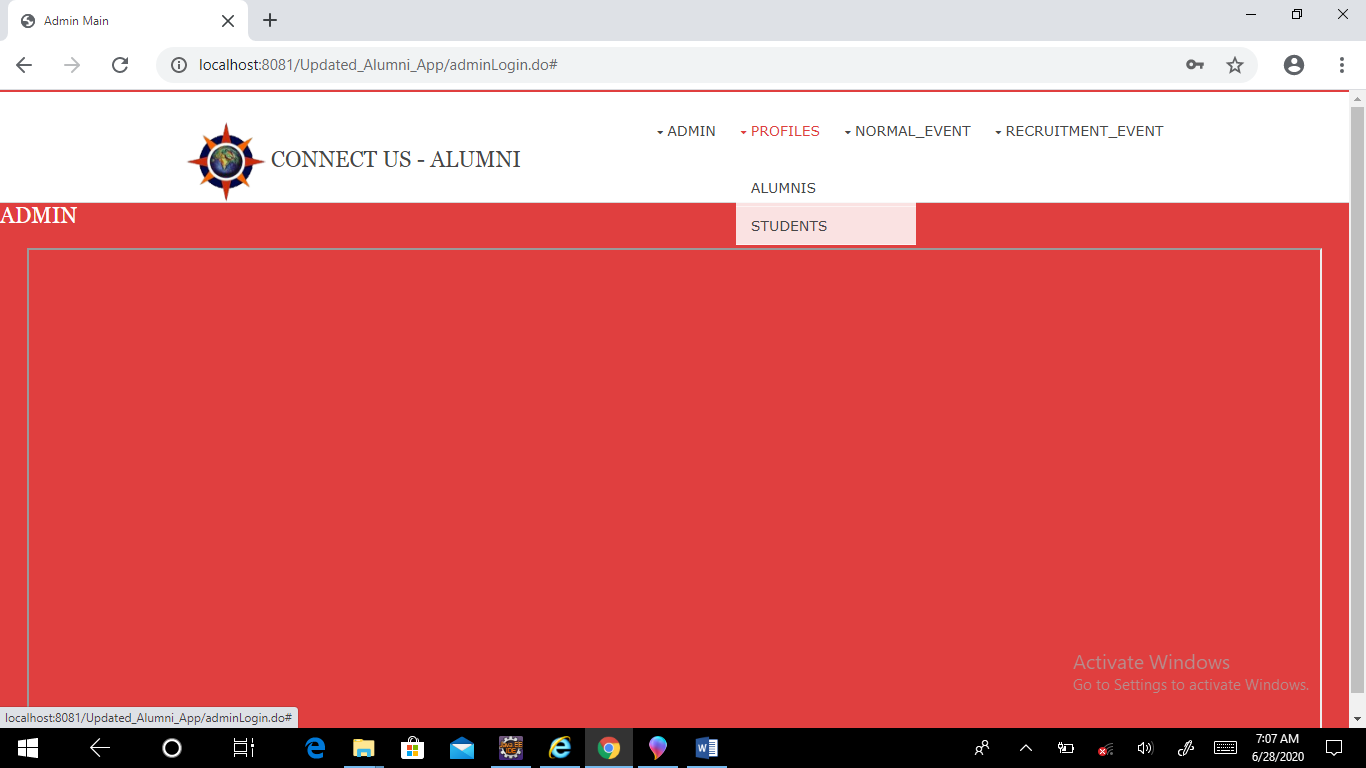


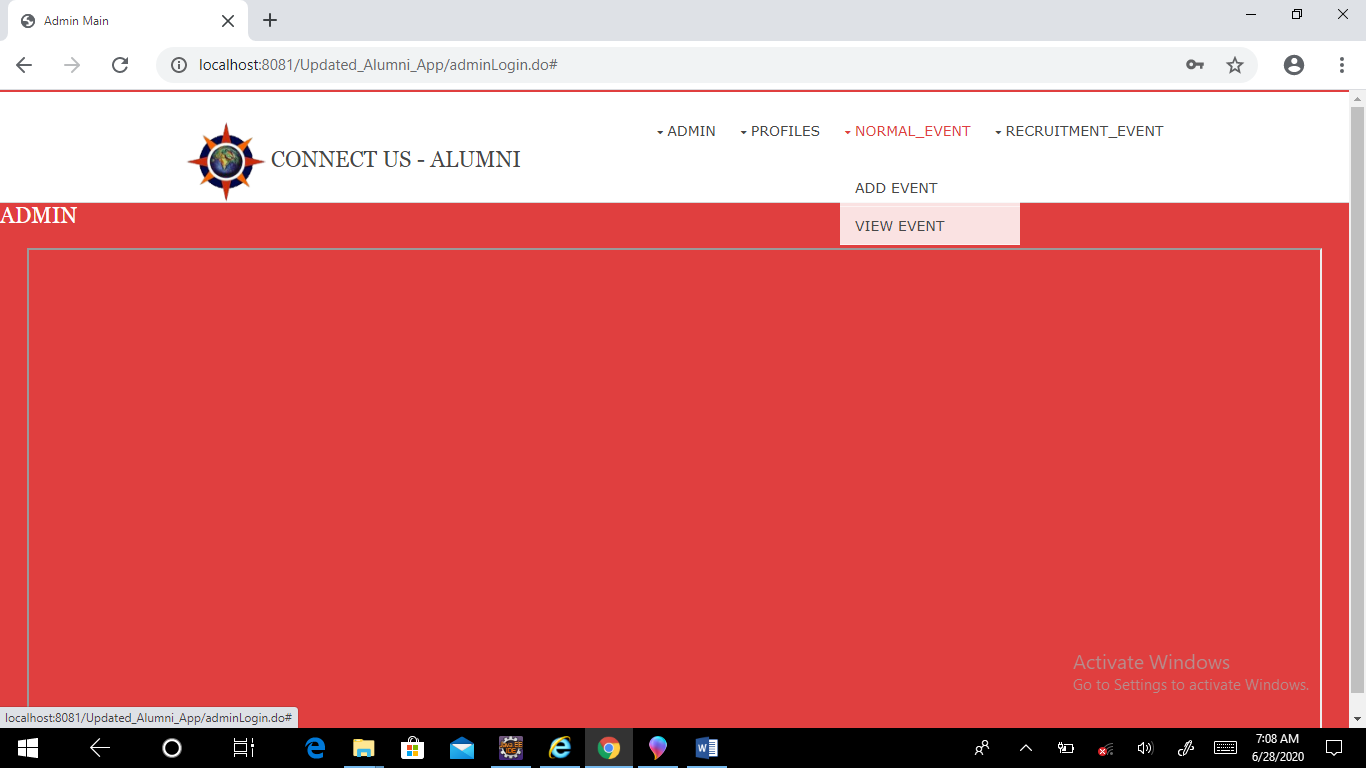


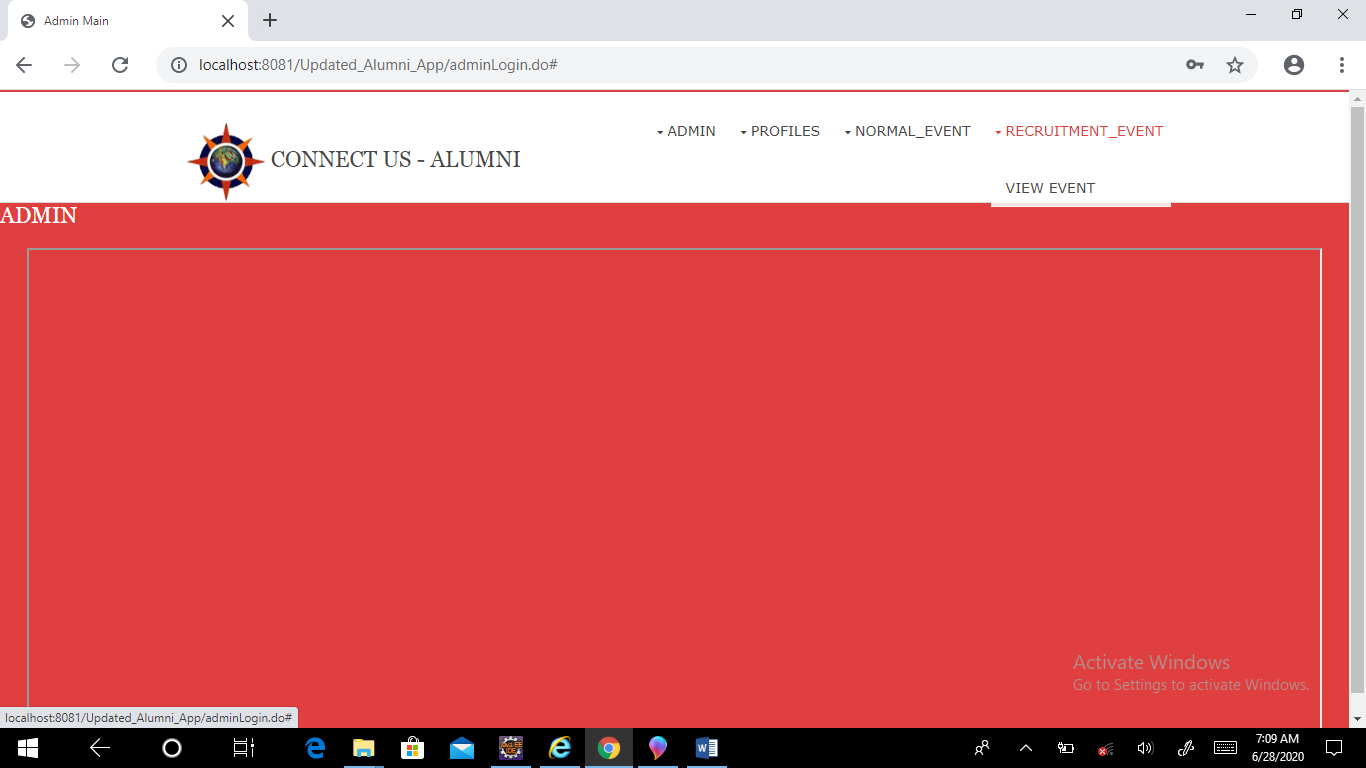


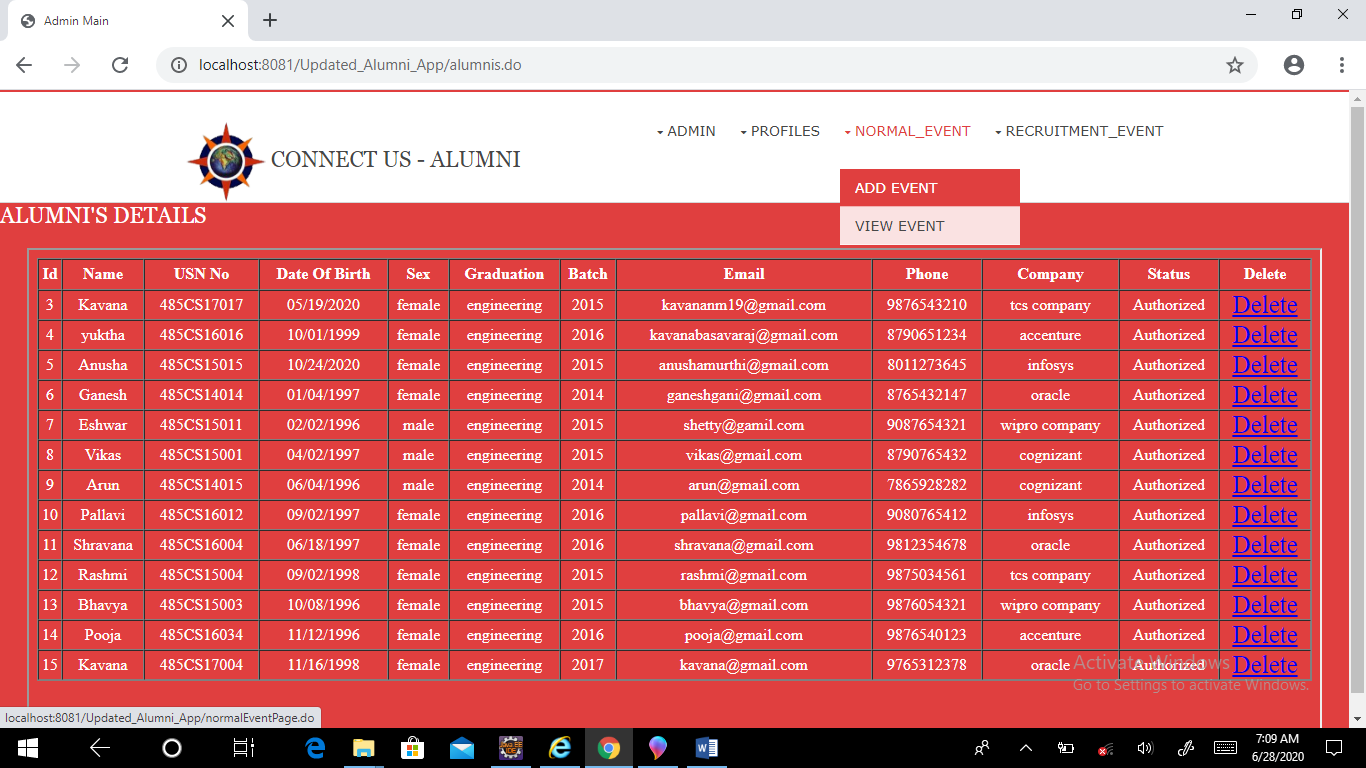


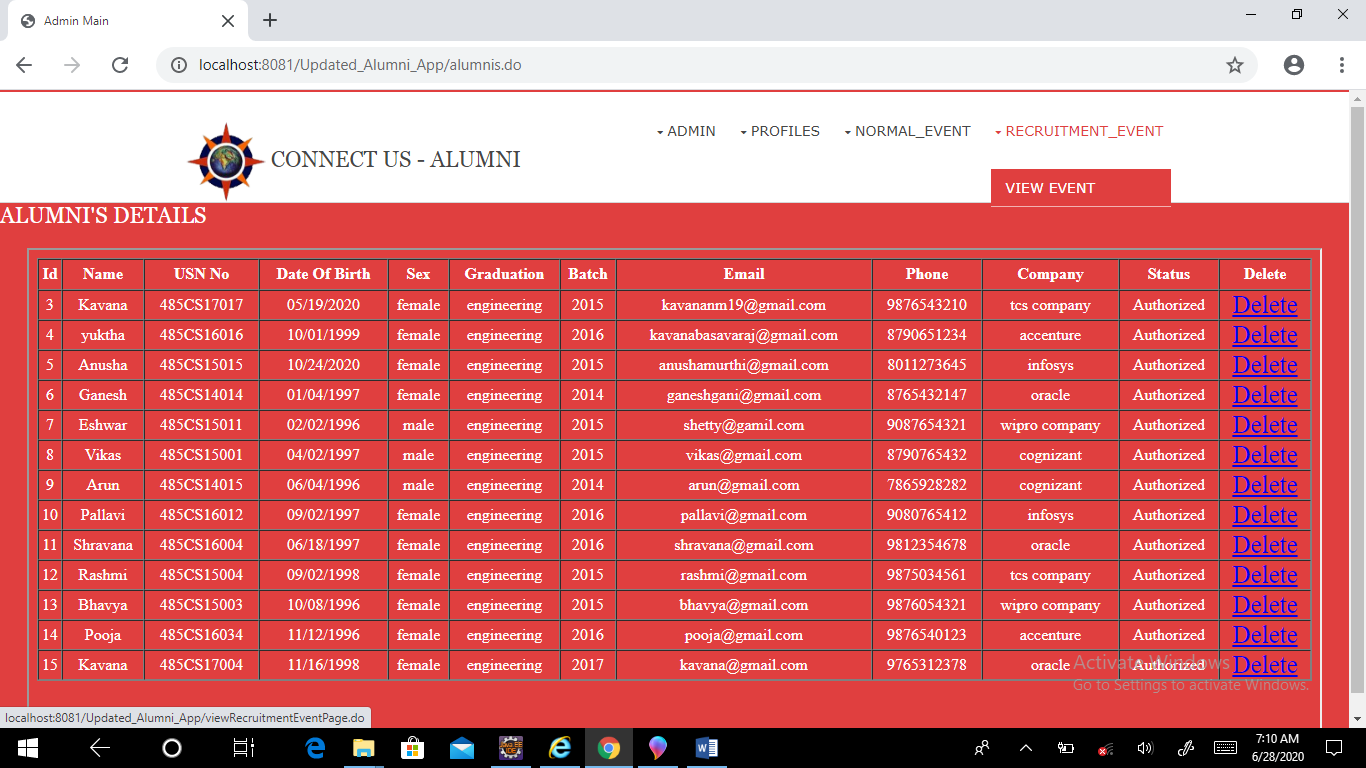


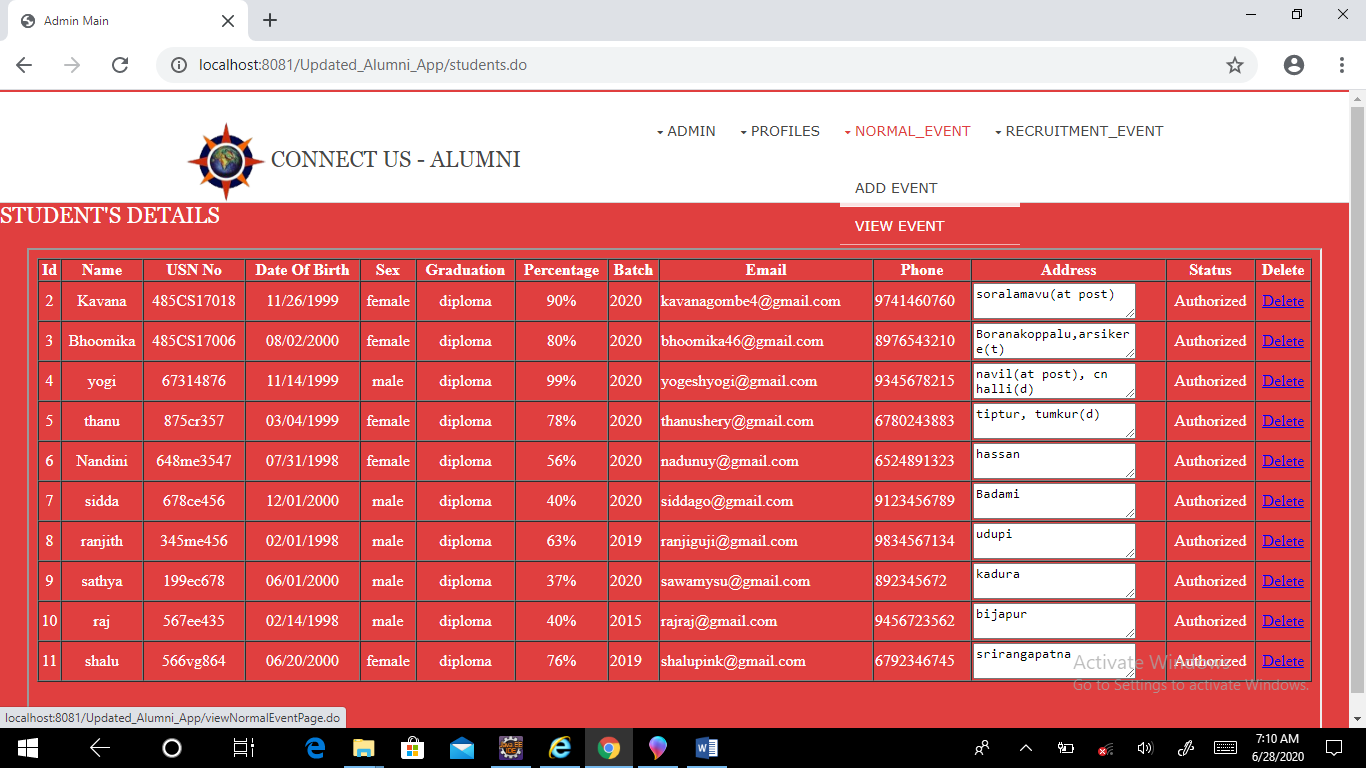


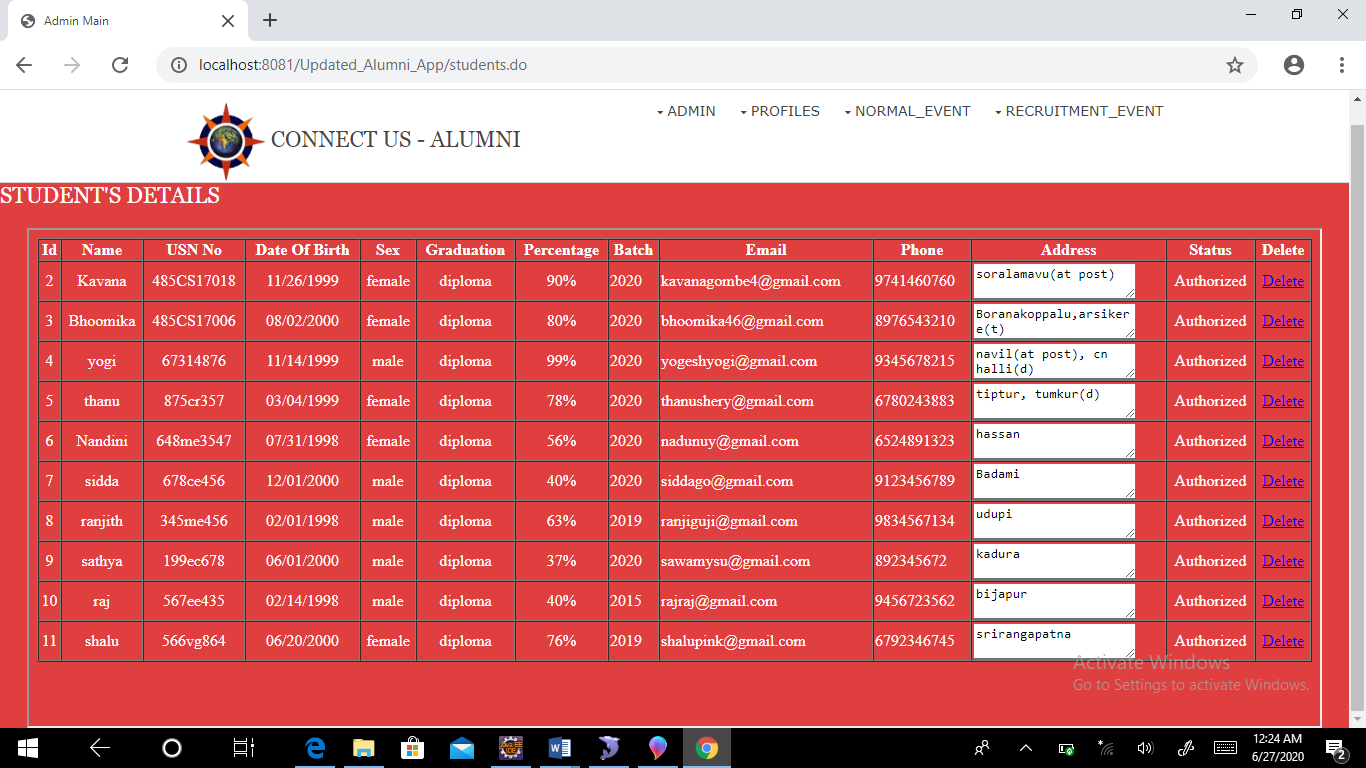


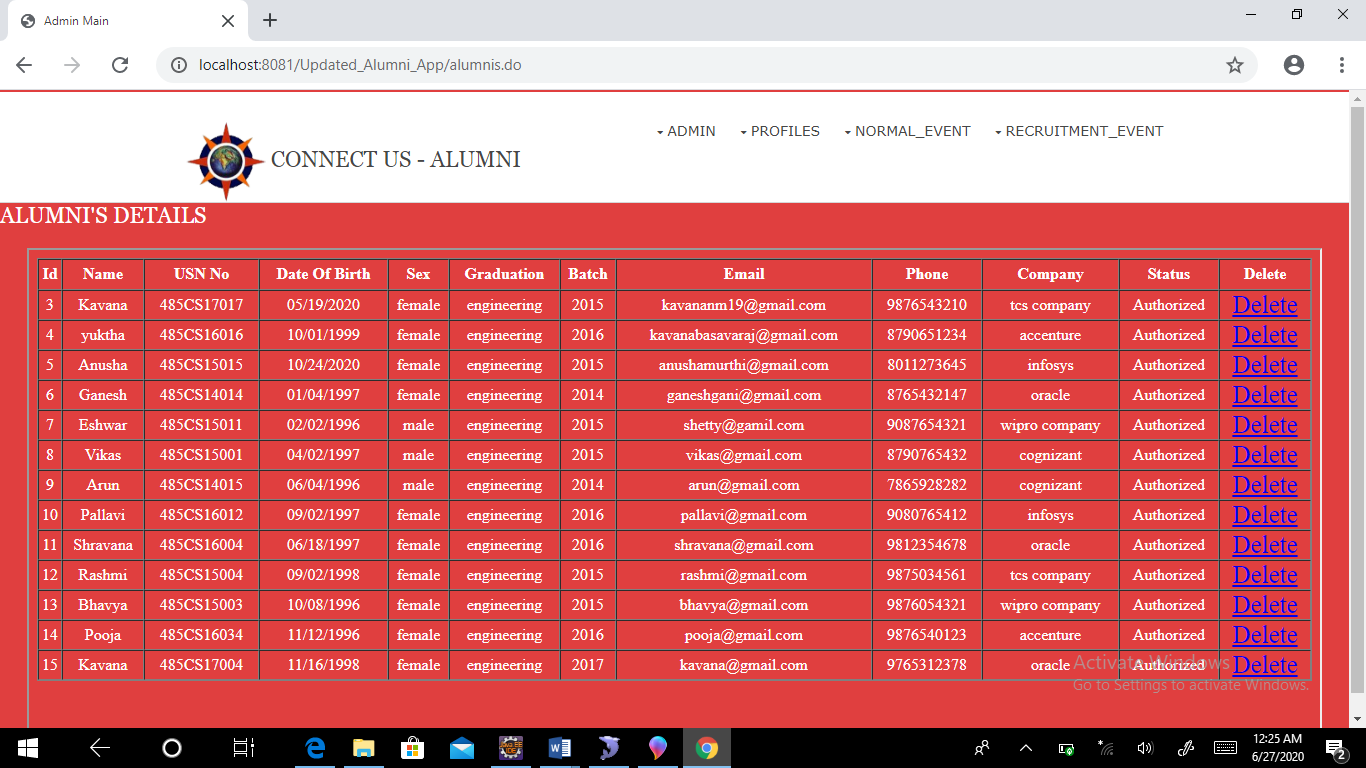


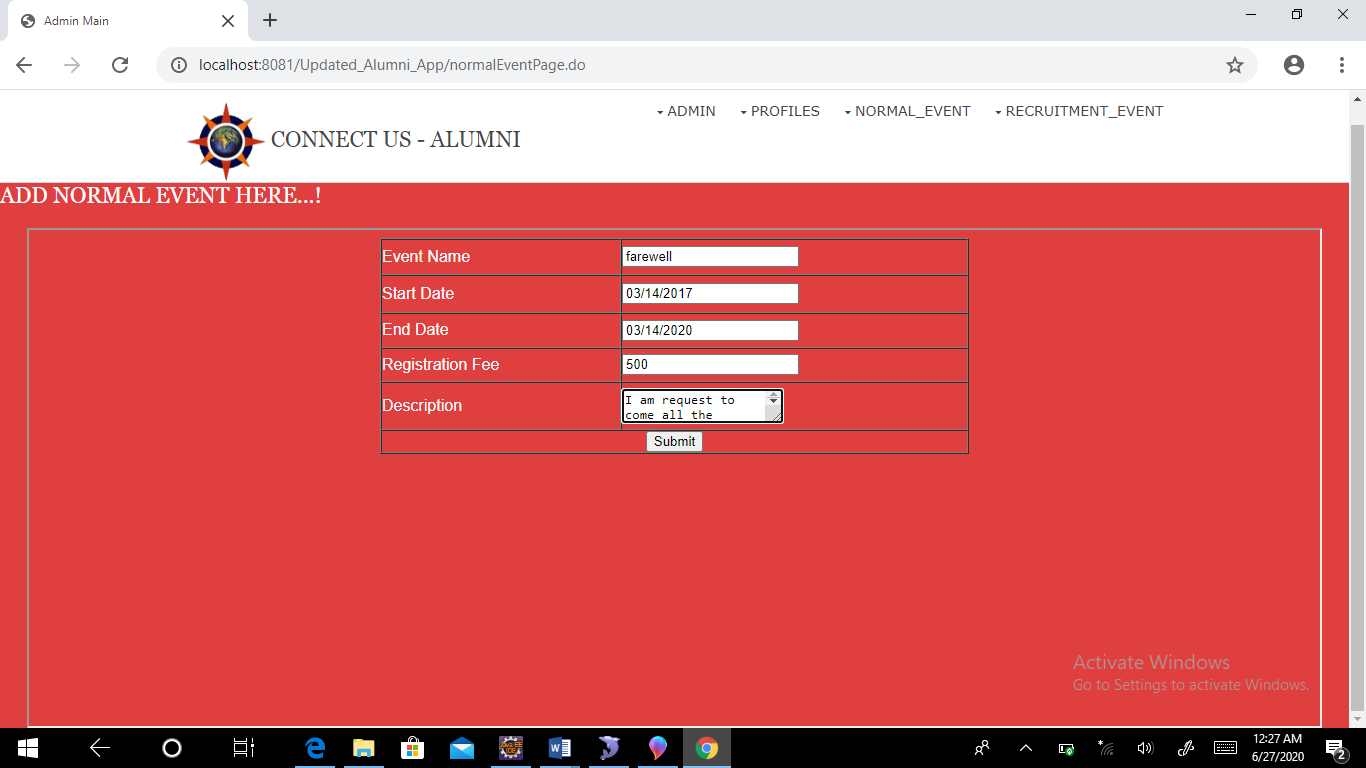














**CHAPTER 12**

**BIBLIOGRAPHY**

* Core JavaTM  2 Volume 2 – Advanced - Cay S. Hortsman
* Pearson Education – Sun Microsystems - Gary Comell
* The Complete Reference-Java - Herbert Schildt

**CHAPTER 13**

**CONCLUSION**

So the Alumni Information Database is mainly used to share the views between the users of the application which is very useful to upgrade the knowledge of everyone. The application is also serve as a useful site to know what is going on in our college and can also know about the various opportunities of the outer world. The application can be further expanded by following the future Enhancements mentioned above.