

Developer Manual

Project Team:

Abhishek Sarkar 11010101

B Sriharsha 11010110

K S V A Ravi Teja 11010126

Moon Sushant 11010141

Sagar Patel 11010161

Shreyas Basarge 11010177

Employee Management System

Overview:

The software contains all the modules extensively. The functions of various modules are shown along with their code snippets in this manual.

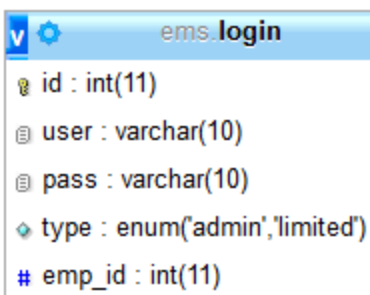
Security Issues: Special care was taken during the execution of the Sql queries so that **SQL injection** cannot be done. The usage of prepared statements helps to reduce the possibility of SQL injection and hacking the database.

Database View:

The database contains the following tables along with their columns.

Login table:

Login table contains the username , password , type of user i.e., admin or supplementary user.











The image shows a screenshot of a database table definition for 'ems.login'. The table has five columns: 'id' (int(11)), 'user' (varchar(10)), 'pass' (varchar(10)), 'type' (enum('admin','limited')), and 'emp_id' (int(11)).

Column	Definition
id	int(11)
user	varchar(10)
pass	varchar(10)
type	enum('admin','limited')
emp_id	int(11)

Employee table:

This table holds typical the employee general details like name, picture, DOB, designation i.e., professor or security guard or staff etc. and also the information if the employee is suspended/removed or not.

ems.employee	
	id : int(11)
	title : varchar(10)
	name : varchar(50)
	picture : mediumblob
	dob : date
	mobile : varchar(15)
	role : enum('prof','security','staff')
	salary : int(11)
	start_date : date
	leaves : int(11)
	__active : tinyint(1)
	__update : timestamp

This table has information regarding the details of professors like his department, courses undertaken, ongoing projects, his qualifications etc.

A separate table is designed exclusively to hold information of professor's research interests.

ems_prof_details
id : int(11)
dept : varchar(50)
creds : varchar(100)
course : varchar(6)
courses : varchar(100)
projects : varchar(200)
webmail : varchar(20)
bach : varchar(50)
mast : varchar(50)
phd : varchar(50)

ems_prof_research_interests
id : int(11)
prof_id : int(11)
field : varchar(50)

This table has the information regarding the feedback given by students such as rating, course.

ems_prof_feedback
id : int(11)
fid : varchar(20)
course : varchar(6)
course_rating : int(11)
prof_rating : int(11)
filled : tinyint(4)

The security details table has the information like his working days, working slots and his working place. Also the staff_details table has the information of technical staff members like his department, location, his responsibility and his category.

ems.security_details	
id	int(11)
emp_id	int(11)
day	varchar(20)
start_time	time
end_time	time
place	varchar(20)

ems.staff_details	
id	int(11)
category	varchar(50)
dept	varchar(50)
location	varchar(50)
respon	varchar(50)

Connection to database:

The connection is set by **openDB()** function of the **Auth** class. The connection to the database initially is set by the following code snippet.

Make sure that the username of the database server is **emsadmin** and the password is **ptls** or modify accordingly in the **constr** string.

```
static void openDB(){
    if (IO::File::Exists(Application::StartupPath + "\\server")){
        constr = String::Format("Server={0};Database=ems;Uid=emsadmin;pwd=ptls",
            IO::File::ReadAllText(Application::StartupPath + "\\server"));
    } else
        constr = "Server=localhost;Database=ems;Uid=emsadmin;pwd=ptls";

    con = gcnew MySqlConnection(constr);
    try {
        con->Open();
    } catch (Exception^ ex){
    }
}
```

The initial build of database just requires the presence of a database of the name **"ems"** in the server. The following code snippet takes the Sql file **"firstrun"** from

the folder in which the application exists and executes the SQL queries and creates the tables of the database.

```
//first run setup
if (IO::File::Exists(Application::StartupPath + "\\firstrun")){
    String^ cstri = IO::File::ReadAllText(Application::StartupPath + "\\firstrun");
    MySqlCommand^ cm = gcnew MySqlCommand(cstri, Auth::con);
    cm->ExecuteNonQuery();
    IO::File::Move(Application::StartupPath + "\\firstrun", Application::StartupPath + "\\firstrun"
}
```

Login :

Initial login screen allows any user to login into the software. The authentication is done in the **login()** function of the **Auth** class . The function returns **true** if the password and username are correct.

The following code snippet approves the authentication for the admin. If the **type equals admin** then the login is approved.

```

static bool login(String^ user, String^ pass){
    pass = encrypt(pass);
    String^ query = "SELECT * FROM `login` WHERE user=@user AND pass=@pass";
    MySqlCommand^ cmd = gcnew MySqlCommand(query, con);
    cmd->Prepare();
    cmd->Parameters->AddWithValue("@user", user);
    cmd->Parameters->AddWithValue("@pass", pass);
    MySqlDataReader^ result = cmd->ExecuteReader();

    if (result->Read()){

        Auth::user = result->GetString("user");
        Auth::emp_id = result->GetInt32("emp_id");
        //MessageBox::Show(user);
        isLoggedIn = true;
        String^ type = result->GetString("type");
        if (type->Equals("admin")){
            isAdmin = true;
        } else {
            isAdmin = false;
        }
    }
}

```

Add Module:

Adding a new user:

A new user can be added by the admin by filling in the username and password and the following function is called after pressing the save button.(Refer User Manual for usage)

```

System::Void btnSaveUP_Click(System::Object^ sender, System::EventArgs^ e) {
    int x;
    if (!Int32::TryParse(lblId->Text, x)) return;
    MySqlCommand^ cmd = gcnew MySqlCommand(String::Format("UPDATE login set
        user=@user, pass=@pass WHERE emp_id={0}", x), Auth::con);

    cmd->Prepare();
    cmd->Parameters->AddWithValue("@user", txtNewUsername->Text);
    cmd->Parameters->AddWithValue("@pass", txtNewPassword->Text);
    cmd->ExecuteNonQuery();
    showMsg("New credentials set.");
}

```

Adding Employee details:

Employee has to be added by first selecting the type of employee from the menu option **“Add”** ,then fill the details and press the **save button** which calls the event **OnClick** for the save button which extensively checks all the fields as in the code snippet below.

```
if(this->txtName->Text->Equals(""))
{
    errorblank += " Name must be entered";
    errorblank += " \n";
    err= true;
}
if(this->txtDob->Text->Equals(""))
{
    errorblank += " Date of Birth must be entered";
    errorblank += " \n";
    err= true;
}
if(this->txtStartDate->Text->Equals(""))
{
    errorblank += " Starting Date must be entered";
    errorblank += " \n";
    err= true;
}
if(this->txtRole->Text->Equals(""))
{
    errorblank += " Role must be entered";
    errorblank += " \n";
    err= true;
}
```

1. **Professor:**

Professor is added by first selecting from the tool strip menu Add->Professor and then filling all the records. The professor details are added into the database using the save_employee function which is called if no errors are found.


```

if (txtRole->Text->Equals("prof")){
    ProfDetails^ pd = gcnew ProfDetails();
    pd->dept = this->txtProfDept->Text;
    //this->txtProfCreds->Text = pd->creds;
    pd->creds = "";
    pd->course = this->txtProfCourse->Text;
    pd->courses = this->txtProfCoursesTaught->Text;
    pd->projects = this->txtProfProjects->Text;
    pd->webmail = this->txtProfWebmail->Text;
    pd->bach = this->txtbach->Text;
    pd->mast = this->txtmast->Text;
    pd->phd = this->txtphd->Text;

    List<String^>^ ri = gcnew List<String^>();
    for (int i=0; i<lstProfRi->Items->Count; i++){
        ri->Add(lstProfRi->Items[i]->ToString());
    }
    if (!isNew) e.putProf(pd, ri); else e.putNewProf(pd, ri);
}

```

2. Security:

Security is added by first selecting from the tool strip menu

Add-> Security and then filling all the records. The Security details are added into the database using the save_employee function which is called if no errors are found.

```

else if(txtRole->Text->Equals("security")){

    List<SecurityDetails^>^ st = gcnew List<SecurityDetails^>();
    for (int i=0;i<lstSecurityTimings->Items->Count;i++)
    {
        SecurityDetails^ s = gcnew SecurityDetails();
        array<String^>^ l = lstSecurityTimings->Items[i]->ToString()->Split('\t');
        s->day = l[0];
        s->place = l[3];
        //s->start_time = l[2];
        //s->end_time l[3];
        DateTime dt1 = Convert::ToDateTime(l[1]);
        s->start_time = dt1;
        DateTime dt2 = Convert::ToDateTime(l[2]);
        s->end_time = dt2;

        st->Add(s);
    }
}

```

3. Staff:

Staff is added by first selecting from the tool strip menu Add-> Staff and then filling all the records. The Staff details are added into the database using the save_employee function which is called if no errors are found.

```
else if (txtRole->Text->Equals("staff")){
    StaffDetails^ sd = gcnew StaffDetails();
    sd->category = this->txtStaffCategory->Text;
    sd->respon = this->txtStaffResponsibility->Text;
    sd->location = this->txtStaffLocation->Text;
    sd->dept = this->txtStaffDepartment->Text;
    if (!isNew) e.putStaff(sd); else e.putNewStaff(sd);
}
```

Search Module:

The search module is implemented in the class “**search**”. Various functions for the different combo boxes are implemented in this class.

- The **FillEmployeeCombo** function is called to fill the different employees in the employee combo box using the following snippet.

```
MySQLCommand^ cmd = gcnew MySQLCommand(String::Format("SELECT * FROM `employee` GROUP BY `role`"), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();

Employee_comboBox->Items->Clear();

while (result->Read()){
    String^ Employee;
    Employee=result->GetString("role");
    Employee_comboBox->Items->Add(Employee);
}
```

1. Professor Search:

- i. The **searchby** combo box for professor is filled using the following code snippet.

```

Searchby_comboBox->Items->Clear();
FillCombo_Prof(comboBox1);
Searchby_comboBox->Items->Add("Dept");
Searchby_comboBox->Items->Add("Research");
Searchby_comboBox->Items->Add("Webmail");
Searchby_comboBox->Items->Add("Projects");

```

- ii. The Dept combo Box is filled using the following code snippet.

```

MySQLCommand^ cmd = gcnew MySqlCommand(String::Format("SELECT employee.name, prof_details.dept
FROM employee INNER JOIN prof_details ON employee.id=prof_details.id WHERE prof_details.dept='{0}'",Dept), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();
comboBox1->Items->Clear();
while (result->Read()){
    String^ ProfName;
    ProfName=result->GetString("name");
    comboBox1->Items->Add(ProfName);
}

```

- iii. The Research combo Box is filled using the following code snippet.

```

MySQLCommand^ cmd = gcnew MySqlCommand(String::Format("SELECT * FROM `prof_research_interests` GROUP BY `field`"), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();

Research_comboBox->Items->Clear();

while (result->Read()){
    String^ field;
    field=result->GetString("field");
    Research_comboBox->Items->Add(field);
}

```

- iv. The Webmail combo Box is filled using the following code snippet.

```

MySQLCommand^ cmd = gcnew MySqlCommand(String::Format("SELECT * FROM `prof_details` GROUP BY `webmail`"), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();

webmail->Items->Clear();

while (result->Read()){
    String^ webmail_id;
    webmail_id=result->GetString("webmail");
    webmail->Items->Add(webmail_id);
}

```

- v. The Projects combo Box is filled using the following code snippet.

```

MySQLCommand^ cmd = gcnew MySQLCommand(String::Format("SELECT * FROM `prof_details` GROUP BY `projects`"), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();

projects->Items->Clear();

while (result->Read()){
    String^ project_name;
    project_name=result->GetString("projects");
    projects->Items->Add(project_name);
}

```

- vi. The namecombo for any of the above searches is filled by calling the corresponding fillnamecombo function with the string obtained from the above search as the parameter. The SQL query joins the 2 tables employee and the corresponding search query and chooses the name of the employee and displays in the combobox.

Eg.: Fill name combobox for professor with search parameter as projects:

```

MySQLCommand^ cmd = gcnew MySQLCommand(String::Format("SELECT employee.name, prof_details.projects
FROM employee INNER JOIN prof_details
ON employee.id=prof_details.id WHERE prof_details.projects='{0}'",project_name), Auth::con);
MySQLDataReader^ result = cmd->ExecuteReader();
comboBox1->Items->Clear();
while (result->Read()){
    String^ ProfName;
    ProfName=result->GetString("name");
    comboBox1->Items->Add(ProfName);
}

```

2. Security Guard Search :

- The following code snippet fills what search filter are made available, which is available in **FillSearchComboforSecurity** which is called when employee type is selected as guard.
- The **FillCombo_place** function is called when place is selected as filter and

it fills with all available places of duty, the `FillCombo_place` combobox.

- The following code snippet fills the Result combobox .This is present in the **FillNameComboByPlace** function which is called when certain place(of duty) is selected.
- The **FillNameComboByTimeandDay** function is called when 'time and date' is selected as filter and it prompts user to select workink time.

Delete Module:

The delete button deletes the records for the corresponding employee .The delete module is implemented in such a way that the admin can undelete the employee back again (Refer Undelete module). The **_active attribute** for the corresponding employee is **set to zero** and not completely deleted from the database. **After 24 hours**, the employee details will be **completely deleted** from the database.

```
if (Int32::TryParse(lblId->Text, x) && !isNew){  
    MySqlCommand^ cmd = gcnew MySqlCommand(String::Format("UPDATE employee SET __active=0  
                                                             WHERE id={0}", x), Auth::con);  
    cmd->ExecuteNonQuery();  
    showMsg("Record deleted.");  
}
```

Undelete Module:

The undelete module sets back the **_active** attribute to 1 and the row

becomes active again. This can be done before 24 hours after which the data will be completely deleted.

```
System::Void btnUndelete_Click(System::Object^ sender, System::EventArgs^ e) {  
  
    if (lstDeleted->SelectedIndex != -1){  
        int x = Convert::ToInt32(lstDeletedId->Items[lstDeleted->SelectedIndex]);  
        MySqlCommand^ cmd = gcnew MySqlCommand(String::Format("UPDATE employee SET __active=1  
                        WHERE id={0}", x), Auth::con);  
        cmd->ExecuteNonQuery();  
        showMsg("Record restored");  
    }  
}
```

Backup and Restore:

On selecting Backup option, first a file dialog box appears to fetch path of storage and this is achieved by this following snippet:

```
if (!Auth::isAdmin) return;  
  
SaveFileDialog^ ofd = gcnew SaveFileDialog();  
ofd->Filter = "EMS Backup files (*.emsbkp)|*.emsbkp";
```

then a process is made to start which runs appropriate command to create a backup which is achieved by this snippet:

```
ProcessStartInfo^ startInfo = gcnew ProcessStartInfo();  
startInfo->FileName = "cmd.exe";  
startInfo->Arguments = "/c mysqldump.exe -u emsadmin -ppts ems > \"\" + ofd->FileName + "\"\"";  
Process::Start(startInfo);
```

Explanation of code:

/c mysqldump.exe -u emsadmin -ppts ems > \"\" + ofd->FileName + "\"\"

emsadmin=user name of database sever ;

ppts=password of server

The above command transfers the data from mysql database to the filename using the redirection operator '>' .

and also for restoring database from a backup file a process is started which executes required cmd command.

```
ProcessStartInfo^ startInfo = gcnew ProcessStartInfo();  
startInfo->FileName = "cmd.exe";  
startInfo->Arguments = "/c mysql.exe -u emsadmin -pptls ems < \"\" + ofd->FileName + "\"\"";  
Process::Start(startInfo);
```

Explanation of code:

/c mysqldump.exe -u emsadmin -pptls ems < \"\" + ofd->FileName + "\""

emsadmin=user name of database sever ;

pptls=password of server

The above command transfers the data from the backup file "Filename " to the Mysql database using the redirection operator '<' .

Feedback Module:

The **ProfFeedback** class contains the required functionalities of this module.

- The following code snippet generates the passwords .This is present in the **generateFeedbackIds** function which is called when the **Generate** button is clicked.

```

for (int i=0; i<numOfStudents; i++){
    fid = String::Format("{0}{1}", course, r->Next(100000000));
    cmd->Parameters["@fid"]->Value = fid;
    cmd->ExecuteNonQuery();
    fids->Add(fid);
}

```

- The **showFeedback function** is called when the Show Feedback button is clicked.

This function generates a message box showing the feedback.

```

static bool showFeedback(String^ course){
    if (!Auth::isAdmin){
        return false;
    }

    MySqlCommand^ cmd = gcnew MySqlCommand(String::Format("SELECT COUNT(*) AS numF, AVG(`prof_rating`)
        AS avgProfRat, AVG(`course_rating`) AS avgCourseRat FROM `prof_feedback`
        WHERE `prof_feedback`.`filled` != 0", course), Auth::con);
    MySqlDataReader^ result = cmd->ExecuteReader();
    result->Read();
    MessageBox::Show(
        String::Format(
            "Number of Students: {0} \n Average Prof Rating: {1} \n Average Course Rating: {2}",
            result->GetInt32("numF"), result->GetInt32("avgProfRat"), result->GetInt32("avgCourseRat")
        )
    );
    result->Close();
    return true;
}

```

Bugs Occurred and Fixed:

- **Problem:** During the filling of the name combo box after selecting a research field, multiple instances of the same name were occurring due to multiple researches for the same professor.

Solution: This problem can be resolved by using the “GROUP BY” query to select distinct names.

