**Requirements Analysis Document**

Employee Scheduling System

CSCI 4711 Software Engineering

Fall 2016

Augusta University

Augusta, GA

Date: 7 December 2016

Version 5

Team Members

Matt Tennis

Chris Gonsalves

Connor Williams

Ryan Mahoney

**Abstract**

This document contains the requirements, analysis and design artifacts for the Employee Scheduling System (ESS) software system. ESS is a personnel scheduling system that facilitates the employee submission and subsequent supervisor approval or denial of time off requests.

The rest of this document is structured as follows: Chapter 1 contains the introduction. This chapter presents a brief description of the system. Chapter 2 outlines the functional requirements of the system. In addition, Chapter 2 contains use case diagrams and use case descriptions for all use cases involved in ESS. Chapter 3 illustrates key GUI screen mockups for the Employee Scheduling System. Chapter 4 contains graphical representations of class design and object interactions. Chapter 5 depicts a decomposition of the ESS subsystems, while Chapter 6 is the appendix.

**Table of Contents**

1. Introduction.................................................................................4

1.1 Scope of System........................................................................4

1.2 Overview of Document...........................................................4

2 Requirements of the System......................................................6

2.1 Functional Requirements......................................................6

2.2 Use Case Diagram....................................................................7

2.3 Use Case Descriptions.............................................................8

2.4 Requirements Analysis...........................................................16

3 User Interface Mockups..........................................................25

3.1 Login..........................................................................................25

3.2 InvalidLogin.............................................................................25

3.3 Logout.......................................................................................26

3.4 TimeOffRequest.......................................................................26

3.5 RequestResponse.....................................................................27

3.6 SupervisorMenu......................................................................27

4 Object Design..............................................................................28

4.1 Object Interaction..................................................................28

4.2 Detailed Class Design............................................................31

4.3 State chart Diagrams.............................................................32

5 System Design.............................................................................35

5.1 Subsystem Decomposition......................................................35

6 Appendix.......................................................................................36

6.1 Appendix A – Source Code......................................................47

1. **Introduction**

**1.1 Scope of System**

The Employee Scheduling System (ESS) is a system used to provide simple and efficient means for an employee to request time off and for appointed supervisors to administrate, approve, or deny those requests. ESS has an internal database with authorized users and their password hashes. Employees can submit requests for time off, which are stored in the database. Supervisors are then able to see the contents of the time off requests, the employee that initiated it, and the reason for the request. Once the Supervisor responds to a request, it is removed from the Supervisor’s queue and the database. The system includes secure login, logout functionality in addition to the primary scheduling applications.

**1.2 Overview of Document**

The rest of the document is structured as follows: Chapter 2 outlines the functional requirements of the system, then the use case diagram. Individual detailed use case descriptions are then listed. Chapter 3 depicts several individual user interface mockups. Chapter 4 displays the class design and object interactions. Chapter 5 shows the ESS subsystem decomposition. Finally, chapter 6 is an appendix of initial implementation of some entity objects.

1. **REQUIREMENTS OF SYSTEM**

**2.1 Functional Requirements**

* **Login –** All users, **Employees** and **Supervisors**, must supply valid login credentials (EmployeeID and password) to be authorized to access and use the system. Upon doing so, the user will have created a session with ESS, where a user can modify database contents through normal usage. Valid login will direct the user to his or her appropriate activity based on the user’s class.
  + **InvalidLogin** – Handle invalid credentials, out-of-scope characters, and exploitation attempts. Returns control to user after job.
* **Logoff –** All users must have clear and immediate access to a Logoff button in order to gracefully and securely close the connection with ESS. Resources allocated to a user session must be terminated in an orderly fashion as to eliminate potential software bugs. Every form or interface must have a clearly marked Logoff button.
* **TimeOffRequest – Employees** must be able to supply a time off request in the Time Off Request form. Employee will select dates via the calendar GUI. Radio buttons enable the **Employee** to indicate the reason (and weight) of his or her request. The user can then submit or logout from that form. Then the system sends time off request to the database.
* **RequestResponse – Supervisors** must be able to view the time off requests that have been submitted in a scroll box queue. The queue will have highlighted regions that correspond to the reason (or weight) supplied by the user’s time off request. The **Supervisor** can then approve, deny, or logout from this window. Approvals and denials modify database contents and update the queue, while logout will terminate the session gracefully.
  + **ApproveRequest** – Approves Employee request and updates relevant database data.
  + **DenyRequest** – Denies Employee request and updates relevant database data

**2.2 Use Case Diagram**

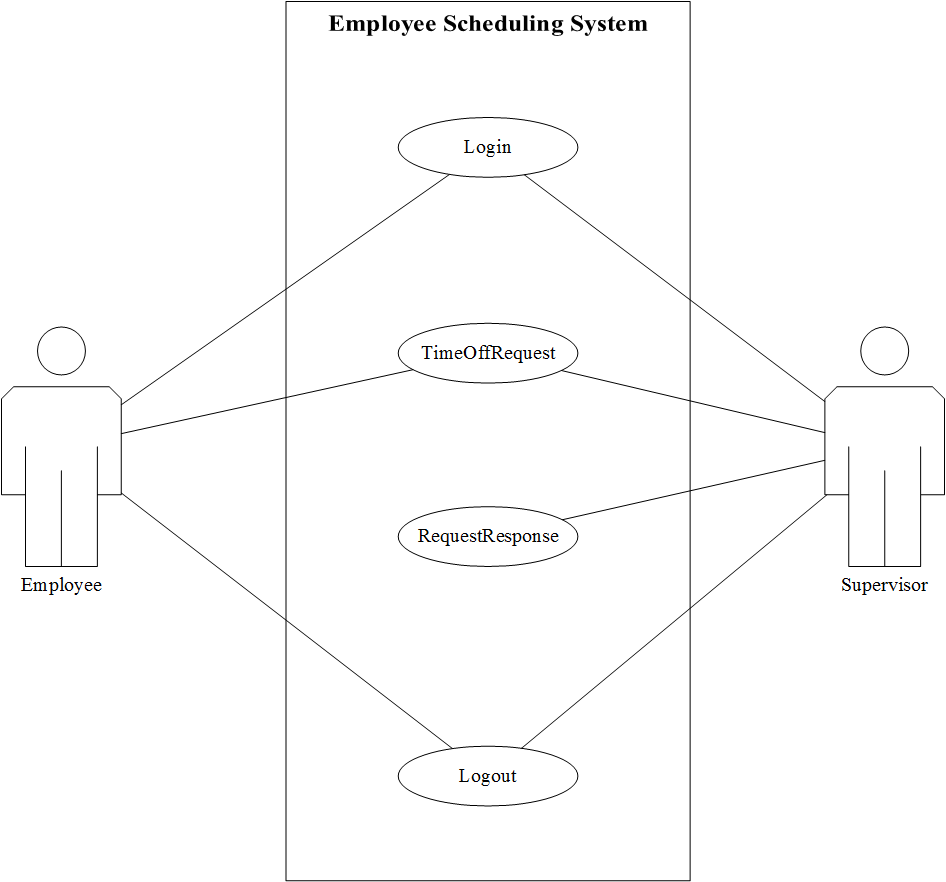


Figure 2.1 Use Case Diagram for ESS

**2.3 Use Case Descriptions**

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating actors* | Initiated by Employee or Supervisor |
| *Flow of events* | 1. Employee enters their user ID in User ID field and Password in Password field. 2. **ESS responds by authenticating the entered user ID and Password**. **The ESS will distinguish if User is Employee or Supervisor via SQL query. The Time Off Request interface will open for Employee and the Supervisor Menu interface will open for Supervisor.** |
| *Entry condition* |  |
| *Exit condition* | User ID and password are authenticated via SQL query. |
| *Security requirements* | The password must be hashed at all times. The dialogue boxes that handle username and password must be shielded against code execution and SQL injections. Password policy must be used to eliminate malicious input. Windows shortcut-key exploits must be disabled to avoid accessing a shell or forcing an exploit. Only <ENTER> will be recognized for acknowledgement of the message/dialogue box. |

Figure 2.2: Login: Valid Login

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating actors* | Initiated by Employeeor Supervisor**.** |
| *Flow of events* | 1. User supplies invalid credentials to the login interface. 2. **System handles the input, returning a user-specific error in a pop-up message/dialog box. A dialogue box pops up to alert User of invalid login.** 3. The user must acknowledge the button in the dialog/box in order to proceed. 4. 4. **System returns the user to the login page, where the user is then able to try to enter valid credentials once more.** |
| *Entry condition* |  |
| *Exit condition* | The user acknowledges the invalid entry. |
| *Security requirements* | The password must be hashed at all times. The dialogue boxes that handle username and password must be shielded against code execution and SQL injections. Password policy must be used to eliminate malicious input. Windows shortcut-key exploits must be disabled to avoid accessing a shell or forcing an exploit. Only <ENTER> will be recognized for acknowledgement of the message/dialogue box. |

Figure 2.3: Login: Invalid Login

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating actors* | Initiated by User |
| *Flow of events* | 1. User presses the logout button on the Time Off Request menu. 2. **System closes Time Off Request menu and returns user to the login screen.** |
| *Entry condition* | User is logged in to the ESS system. |
| *Exit condition* | User is logged out and returned to the login interface. |
| *Security requirements* | Resources allocated to the session must be terminated properly to ensure there are no bugs in the software. |

Figure 2.4: Logout: TORMenu

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating actors* | Initiated by Supervisor |
| *Flow of events* | 1. Supervisor presses the logout button on Request Response menu. 2. **System closes Response Request menu and returns user to the login screen.** |
| *Entry condition* | Supervisor is logged in to the ESS system. |
| *Exit condition* | Supervisor is logged out and returned to the login interface. |
| *Security requirements* | Resources allocated to the session must be terminated properly to ensure there are no bugs in the software. |

Figure 2.5: Logout: RRMenu

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating actors* | Initiated by Supervisor |
| *Flow of events* | 1. Supervisor presses the logout button on Supervisor menu. 2. **System closes Response Request menu and returns user to the login menu.** |
| *Entry condition* | Supervisor is logged in to the ESS system. |
| *Exit condition* | Supervisor is logged out and returned to the login menu. |
| *Security requirements* | Resources allocated to the session must be terminated properly to ensure there are no bugs in the software. |

Figure 2.6: Logout: Supervisor

|  |  |
| --- | --- |
| *Use case name* | RequestResponse |
| *Participating actors* | Initiated by Supervisor |
| *Flow of events* | 1. Supervisor selects the appropriate request from Time Off Request queue on the Request Response interface and clicks the Approve button. 2. **ESS updates Time Off Request status field in database with "Approved".** |
| *Entry condition* | The Supervisor selects Time Off Request Response button from Supervisor Menu Form. |
| *Exit condition* | Time Off Request status field in database with "Approved". |
| *Security requirements* | All responses are tracked by User ID ensuring that no unauthorized individuals are able to surreptitiously gain access to a request. |

Figure 2.7: RequestResponse: Approve

|  |  |
| --- | --- |
| *Use case name* | RequestResponse |
| *Participating actors* | Initiated by Supervisor |
| *Flow of events* | 1. Supervisor selects the appropriate request from Time Off Request queue on the Request Response interface and clicks the Deny button. 2. **ESS updates Time Off Request status field in database with "Denied".** |
| *Entry condition* | The Supervisor selects Time Off Request Response button from Supervisor Menu Form. |
| *Exit condition* | Time Off Request status field in database with "Denied". |
| *Security requirements* | All responses are tracked by User ID ensuring that no unauthorized individuals are able to surreptitiously gain access to a request. |

Figure 2.8: RequestResponse: Deny

|  |  |
| --- | --- |
| *Use case name* | TimeOffRequest |
| *Participating actors* | Initiated by Supervisor or Employee |
| *Flow of events* | 1. The Employee or Supervisor select a date, time and a reason per request. 2. **ESS receives the form and pushes following fields to the database tables: Employee/Supervisor name, request date, request time, and request reason.** |
| *Entry condition* | The Supervisor selects Request Time Off button from Supervisor Menu Form. |
| *Exit condition* | The employee’s time off request is reflected in the appropriate employee and supervisor queues. |
| *Security requirements* |  |

Figure 2.9: TimeOffRequest

**2.4 Requirements Analysis**

This page intentionally left blank

Figure 2.10: EmployeeLogin (non-supervisor) sequence

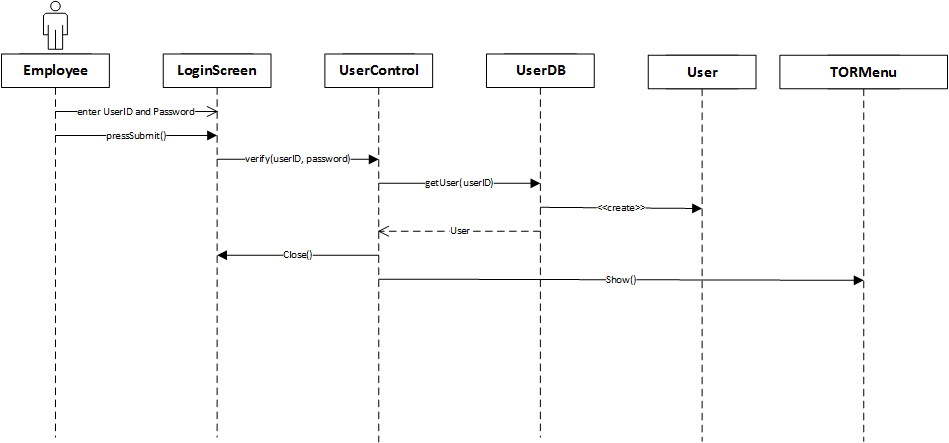


Figure 2.10: Supervisor Login (supervisor) Sequence

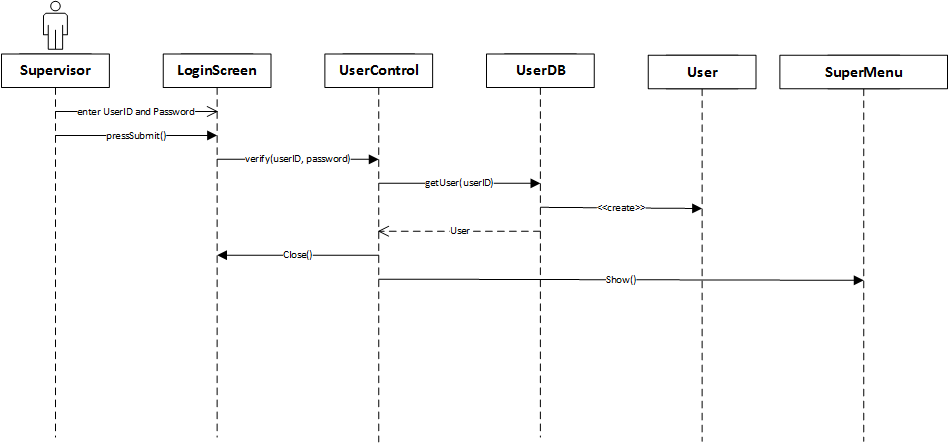


Figure 2.11: InvalidLogin sequence

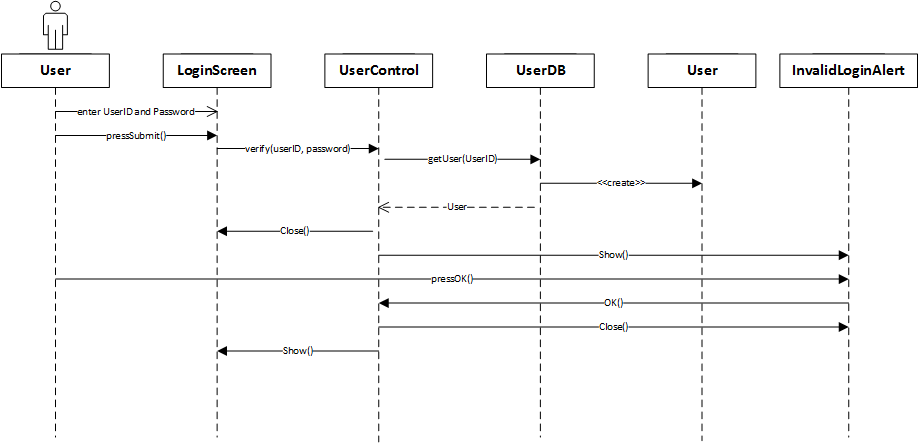


Figure 2.12: Logout: TORMenu sequence

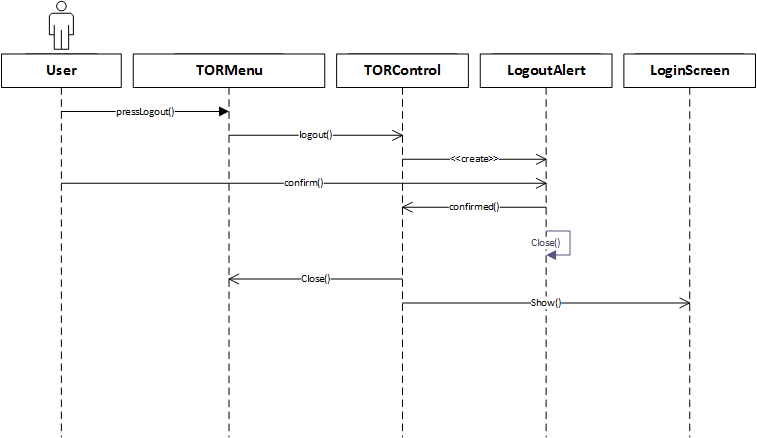


Figure 2.13: RRLogout sequence

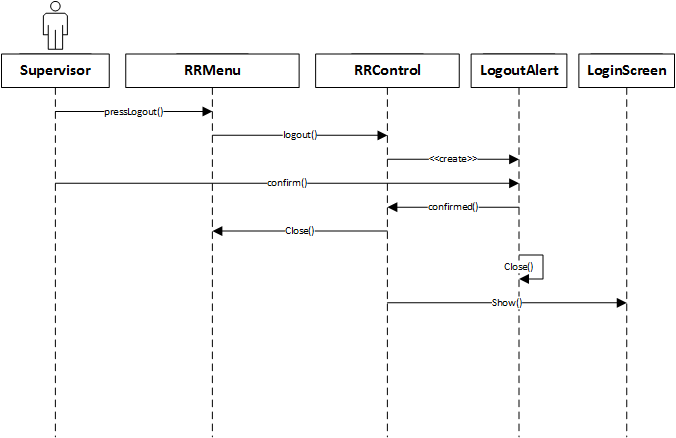


Figure 2.14: TOR(Time Off Request) Sequence

Figure 2.14: TOR sequence

Figure 2.14: TOR(Time Off Request) Sequence



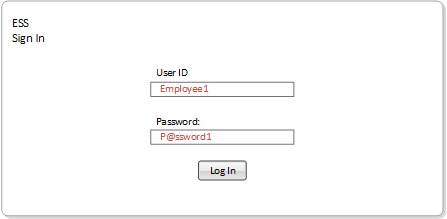
Figure 2.15: RRApprove sequence



Figure 2.16: RRDeny sequence

Figure 2.20: TimeOffResponseApprove sequence

1. **USER INTERFACE MOCKUPS**

 **3.1 Login**

**3.2 InvalidLogin**



**3.3 Logout**



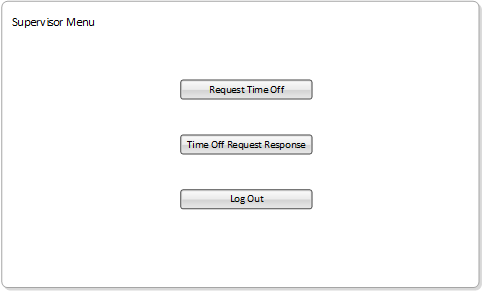
**3.4 TimeOffRequest**



**3.5 RequestResponse**



**3.6 SupervisorMenu**



1. **OBJECT DESIGN**

**4.1 Object Interaction**

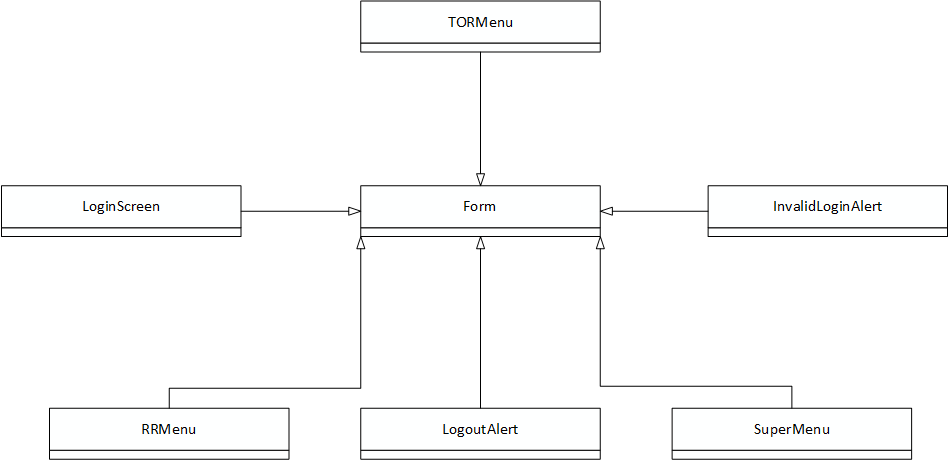


Figure 4.1: Class Diagram: Boundary

Figure 4.2: Class Diagram: Control

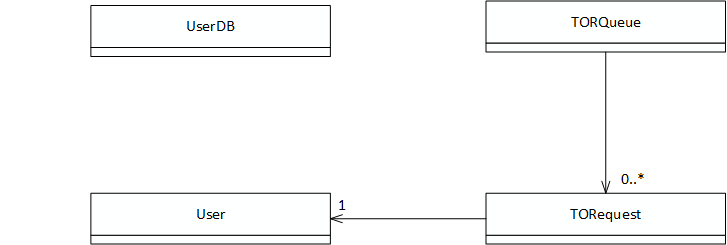
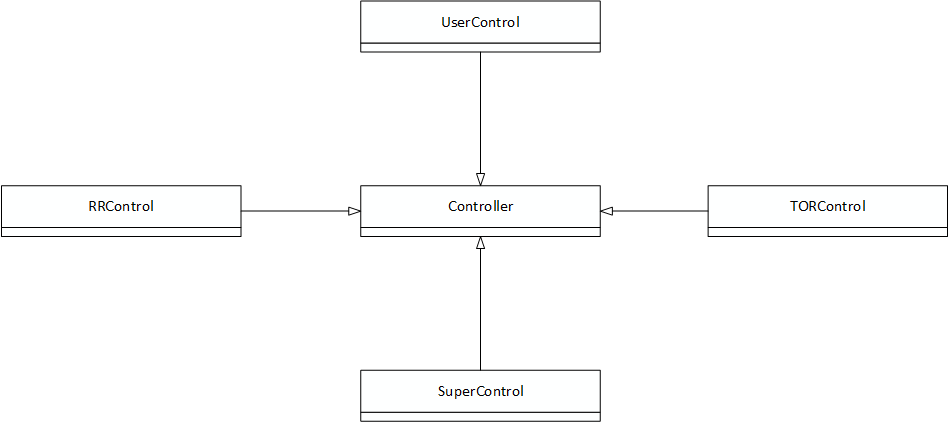


Figure 4.3: Class Diagram: Entity

Figure 4.4: Class Diagram

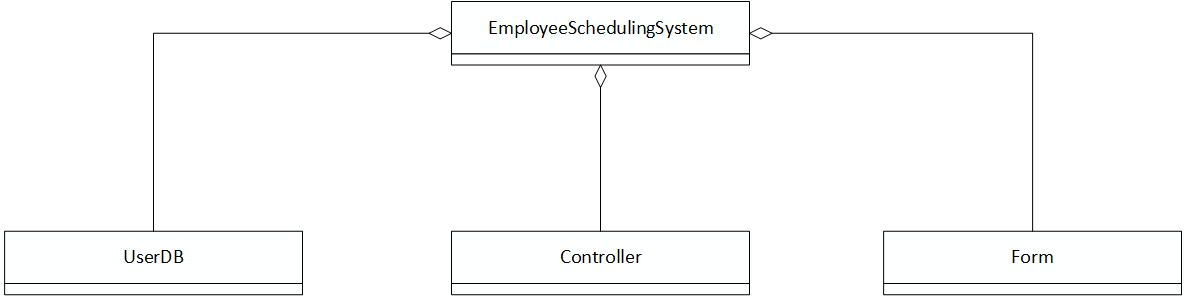
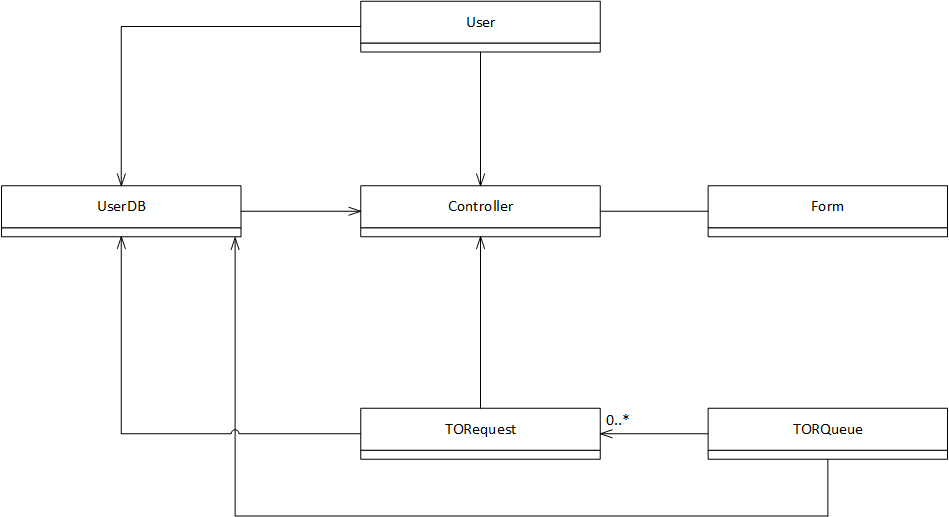


Figure 4.5: Class Diagram

**4.2 Detailed Class Design**

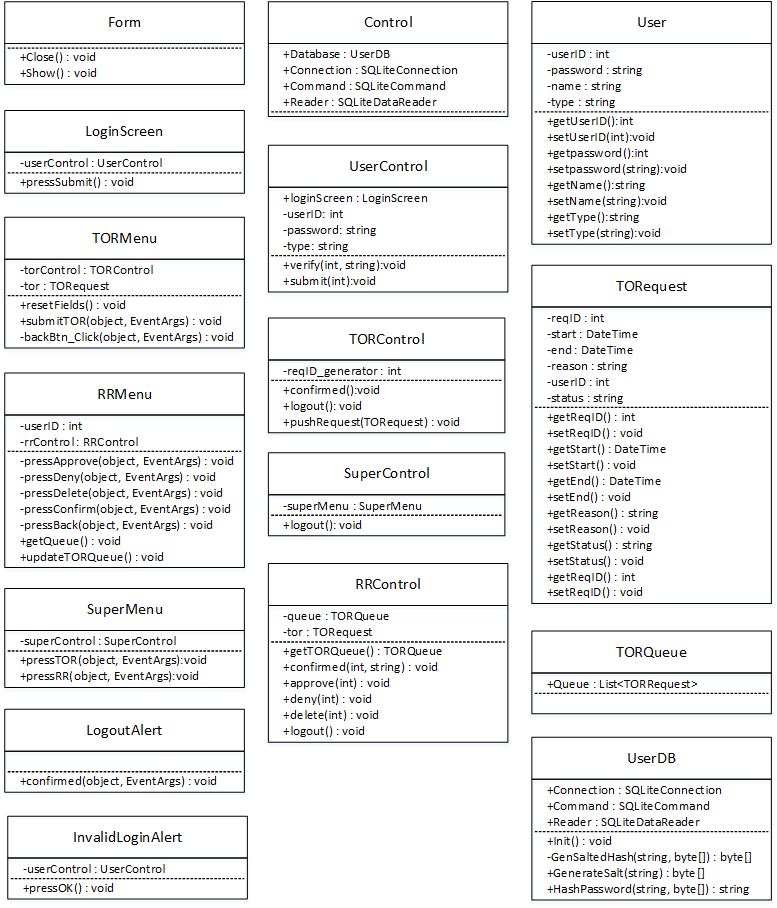
****

Figure 4.6: Class Diagram

Figure 4.6: Class Diagram

**4.3 Statechart Diagrams**

****

Figure 4.8: RRControl

****

Figure 4.10: TORControl

****

Figure 4.11: UserControl

1. **SYSTEM DESIGN**

**5.1 Subsystem Decomposition**

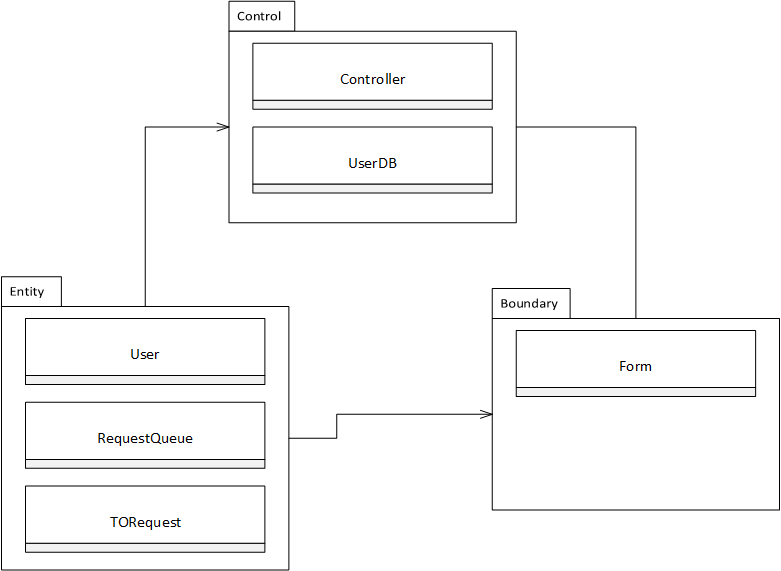


Figure 5.1: Subsystem decomposition

1. **APPENDIX**

**6.1 Appendix A – Source Code**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EmployeeSchedulingSystem.Entity

{

public class TORequest

{

private int reqID;

private DateTime start;

private DateTime end;

private string reason;

private int UserID;

private string status;

public void setUserID(int u)

{

UserID = u;

}

public int getUserID()

{

return UserID;

}

public int getReqID()

{

return this.reqID;

}

public void setReqID(int value)

{

this.reqID = value;

}

public DateTime getStart()

{

return this.start;

}

public void setStart(DateTime value)

{

this.start = value;

}

public DateTime getEnd()

{

return this.end;

}

public void setEnd(DateTime value)

{

this.end = value;

}

public string getReason()

{

return this.reason;

}

public void setReason(string value)

{

this.reason = value;

}

public string getStatus()

{

return this.status;

}

public void setStatus(string value)

{

this.status = value;

}

}

}

Figure 6.1: TORequest Source Code (Entity 1)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SQLite;

using System.ComponentModel.DataAnnotations;

using System.Security.Cryptography;

namespace EmployeeSchedulingSystem.Entity

{

public class UserDB

{

public SQLiteConnection Connection;

public SQLiteCommand Command;

public SQLiteDataReader Reader;

public UserDB() { Init(); }

public UserDB(SQLiteConnection conn, SQLiteCommand cmd)

{

try

{

Connection = conn ?? new SQLiteConnection("Data Source=database.db;Version=3;New=True;Compress=True;");

Command = cmd ?? new SQLiteCommand();

}

catch (Exception e)

{

throw new Exception(e.Message);

}

}

// initialize database for demonstration

public void Init()

{

string hashedPass;

string userSalt;

string pass = "Pa$$w0rd";

Connection = new SQLiteConnection("Data Source=database.db;Version=3;New=True;Compress=True;");

Connection.Open();

Command = Connection.CreateCommand();

// Dropping tables from previous run

Command.CommandText = "DROP TABLE Users";

Command.ExecuteNonQuery();

Command.CommandText = "DROP TABLE TORequests";

Command.ExecuteNonQuery();

// Create Users table

Command.CommandText = @"CREATE TABLE IF NOT EXISTS Users(

Id INTEGER PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Password VARCHAR(255) NOT NULL,

Type VARCHAR(32) NOT NULL,

Salt VARCHAR(255) NOT NULL

)";

Command.ExecuteNonQuery();

// Create TORequests table

Command.CommandText = @"CREATE TABLE IF NOT EXISTS TORequests (

Id INTEGER PRIMARY KEY,

StartDate VARCHAR(32) NOT NULL,

EndDate VARCHAR(32) NOT NULL,

Reason VARCHAR(100) NOT NULL,

UserID INTEGER NOT NULL,

Status VARCHAR(9) NOT NULL,

FOREIGN KEY(UserID) REFERENCES Users(Id)

)";

Command.ExecuteNonQuery();

// Insert demonstration data into Users

// Supervisor Test 1

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 0);

Command.Parameters.AddWithValue("@name", "Supervisor 01");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Supervisor");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

// Employee Test 1

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 1);

Command.Parameters.AddWithValue("@name", "Employee 01");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Employee");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

// Matt

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 1111);

Command.Parameters.AddWithValue("@name", "Matt");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Supervisor");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

// Chris

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 2222);

Command.Parameters.AddWithValue("@name", "Chris");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Employee");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

// Ryan

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 3333);

Command.Parameters.AddWithValue("@name", "Ryan");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Supervisor");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

// Connor

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 4444);

Command.Parameters.AddWithValue("@name", "Connor");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Employee");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

//pass = "Pa$$w0rd";

userSalt = Convert.ToBase64String(GenerateSalt(pass));

hashedPass = HashPassword(pass, Convert.FromBase64String(userSalt));

Command.CommandText = @"INSERT INTO Users (Id, Name, Password, Type, Salt) VALUES (@id, @name, @pass, @type, @salt)";

Command.Parameters.AddWithValue("@id", 5555);

Command.Parameters.AddWithValue("@name", "HashTester");

Command.Parameters.AddWithValue("@pass", hashedPass);

Command.Parameters.AddWithValue("@type", "Employee");

Command.Parameters.AddWithValue("@salt", userSalt);

Command.ExecuteNonQuery();

//Connection.Close();

}

public string HashPassword(string plaintext, byte[] salt)

{

byte[] saltedPass = GenSaltedHash(plaintext, salt);

return Convert.ToBase64String(saltedPass);

}

public byte[] GenerateSalt(string plaintext)

{

byte[] password\_bytes = Encoding.UTF8.GetBytes(plaintext);

byte[] salt = new byte[32];

byte[] pass\_and\_salt = new byte[password\_bytes.Length + salt.Length];

using (RNGCryptoServiceProvider RNG = new RNGCryptoServiceProvider())

{

RNG.GetNonZeroBytes(salt);

}

return salt;

}

private byte[] GenSaltedHash(string plaintext, byte[] salt)

{

HashAlgorithm hashAlgo = new SHA256Managed();

byte[] password\_bytes = Encoding.UTF8.GetBytes(plaintext);

byte[] pass\_and\_salt = new byte[password\_bytes.Length + salt.Length];

// for loops populate byte array of pass + salt

for (int i = 0; i < salt.Length; i++)

{

pass\_and\_salt[i] = salt[i];

}

for (int i = 0; i < password\_bytes.Length; i++)

{

pass\_and\_salt[password\_bytes.Length + i] = password\_bytes[i];

}

return hashAlgo.ComputeHash(pass\_and\_salt);

}

}

}

Figure 6.2: UserDB Source Code (Entity 2)

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data;

using System.Data.SQLite;

using EmployeeSchedulingSystem.Controller;

using EmployeeSchedulingSystem.Entity;

namespace EmployeeSchedulingSystem.Boundary

{

public partial class TORMenu : Form

{

TORequest tor = new TORequest();

TORControl torControl;

private int userID;

SQLiteConnection Connection;

SQLiteCommand Command;

SQLiteDataReader Reader;

public TORMenu(TORControl tControl, int userID)

{

InitializeComponent();

dateTimePicker1.Format = DateTimePickerFormat.Time;

dateTimePicker1.ShowUpDown = true;

dateTimePicker2.Format = DateTimePickerFormat.Time;

dateTimePicker2.ShowUpDown = true;

this.userID = userID;

torControl = tControl;

// Local database properties

Connection = torControl.Database.Connection;

Command = torControl.Database.Command;

// populate existing requests if exist

string query = "SELECT \* FROM TORequests WHERE UserID = '" + userID + "'";

Command.CommandText = query;

SQLiteDataAdapter dataAdapter = new SQLiteDataAdapter(Command);

DataTable dt = new DataTable();

dataAdapter.Fill(dt);

dataGridView1.ReadOnly = true;

dataGridView1.DataSource = dt;

}

private void logout(object sender, EventArgs e)

{

this.Close();

torControl.logout();

//LogoutAlert logoutAlert = new LogoutAlert();

//logoutAlert.Show();

}

public void submitTOR(object sender, EventArgs e)

{

tor.setUserID(userID);

tor.setStart(dateTimePicker1.Value);

tor.setEnd(dateTimePicker2.Value);

tor.setStatus("Pending");

if (radioButton1.Checked == true)

tor.setReason("Vacation");

else if (radioButton2.Checked == true)

tor.setReason("Personal");

else if (radioButton3.Checked == true)

tor.setReason("Emergency");

else

MessageBox.Show("please select a reason for time off.");

torControl.confirmed(tor);

Command = Connection.CreateCommand(); // may be unnecessary

string query = "SELECT \* FROM TORequests WHERE UserID = '" + userID + "'";

Command.CommandText = query;

SQLiteDataAdapter dataAdapter = new SQLiteDataAdapter(Command);

DataTable dt = new DataTable();

dataAdapter.Fill(dt);

dataGridView1.ReadOnly = true;

dataGridView1.DataSource = dt;

torControl.pushRequest(tor);

}

private void backBtn\_Click(object sender, EventArgs e)

{

string type = "";

string query = "SELECT Type FROM Users WHERE ID = @userID";

Command.Parameters.AddWithValue("@userID", userID);

Command.CommandText = query;

Command.ExecuteNonQuery();

using (SQLiteDataReader Reader = Command.ExecuteReader())

{

if(!Reader.HasRows)

{

// error

}

else

{

while (Reader.Read())

{

type = Reader.GetString(0);

}

if (type == "Supervisor")

{

SuperControl superControl = new SuperControl(torControl.Database, userID);

this.Close();

}

else if (type == "Employee")

{

torControl.logout();

this.Close();

}

else

{

InvalidLoginAlert alert = new InvalidLoginAlert();

alert.Show();

}

}

}

}

}

}

Figure 6.3: TORMenu Source Code (Boundary)

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EmployeeSchedulingSystem.Boundary;

using EmployeeSchedulingSystem.Entity;

namespace EmployeeSchedulingSystem.Controller

{

public class TORControl : Controller

{

// overload Controller Database prop

private new UserDB Database;

int reqID\_generator = 0;

// default constructor

public TORControl()

{

Connection = new SQLiteConnection("Data Source=database.db;Version=3;New=True;Compress=True;");

//Connection.Open();

Command = Connection.CreateCommand();

}

public TORControl(UserDB db) : base(db)

{

Database = db; // ?? new UserDB();

Connection = Database.Connection;

Command = Database.Command;

}

public void confirmed(TORequest TOR)

{

TORequest tor = TOR;

tor.setReqID(reqID\_generator);

reqID\_generator++; // increment TORequest primary key generator

pushRequest(tor);

}

public void pushRequest(TORequest t)

{

// Insert data into TORequests table

//string query = "INSERT INTO TORequests (Id, StartDate, EndDate, Reason, UserID, Status) VALUES ('"+ tor.getReqID() +"','"+ tor.getStart()+"','"+tor.getEnd() +"','"+tor.getReason() +"','"+tor.getUserID() +"','"+tor.getStatus() +"')";

try

{

string query = "INSERT INTO TORequests (Id, StartDate, EndDate, Reason, UserID, Status) VALUES (@reqID, @start, @end, @reason, @id, @status)";

// SQL Injection Prevention

Command.Parameters.AddWithValue("@reqID", t.getReqID());

Command.Parameters.AddWithValue("@start", t.getStart());

Command.Parameters.AddWithValue("@end", t.getEnd());

Command.Parameters.AddWithValue("@reason", t.getReason());

Command.Parameters.AddWithValue("@id", t.getUserID());

Command.Parameters.AddWithValue("@status", t.getStatus());

Command.CommandText = query;

Command.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public void logout()

{

LogoutAlert \_logout = new LogoutAlert();

\_logout.Show();

}

}

}

Figure 6.4: TORControl Source Code (Controller)

**Revision History:**

Version #, Section #: Item Modified

Version 2, All Sections: Corrected formatting

Version 2, Section 1.1, Overview of System: Added system overview

Version 2, Section 2.1, Functional Requirement: Logoff

Version 2, Section 2.4, Analysis Requirements: Added sequence diagrams

Version 3, Section 2.2, Use Case Diagram: Updated Use Case Diagram

Version 3, Section 2.3, Use Case Descriptions: Edited multiple Use Case Descriptions

Version 3, Section 2.4, Analysis Requirements: Edited multiple sequence diagrams

Version 3, Section 4.1, Object Relationship: Added Object Interaction Diagrams

Version 3, Section 4.2, Detailed Class Design: Added Class Design Diagrams

Version 4, Section 2.3, Use Case Descriptions: Edited multiple Use Case Descriptions

Version 4, Section 4.3, State Chart Diagrams: Added Object Design Diagrams

Version 4, Section 5.1, Subsystem Decomposition: Added System Design Diagrams

Version 4, Section 6.1, Appendix: Source Code

Version 5, Section 2.4, Analysis Requirements: Adjusted Sequence Diagrams

Version 5, Section 4.1, Object Relationship: Fixed version discrepancies

Version 5, Section 4.2, Detailed Class Diagrams: Updated Class Diagram to Code

Version 5, Section 4.3, State Chart Diagrams: Improved State Chart Diagrams

Version 5, Section 5.1, Subsystem Decomposition: Fixed subsystem associations

Version 5, Section 6.1, Appendix: Added two Entity, one Boundary, and two Control diagrams