**ENROLLMENT QUEUING MANAGEMENT SYSTEM**

****

**In Partial Fulfillment of the requirements in**

**CS105L: Data Structures and Algorithms**

**Presented by:**

**GIMRIL N. LOZARITA, BSCpE 2nd**

**RALPH ROGER A. ROSALES, BSCpE 2nd**

**CALVIN L. LEE, BSCpE 2nd**

**Presented to:**

**PROF. EMMY GRACE REQUILLO**

**JULY 4, 2024**

**I. Introduction**

**A. Problem Scenario**

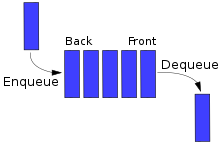
At the start of each semester, Mapua Malayan Colleges Mindanao has a big challenge: coordinating thousands of students' enrollments. The existing manual queuing method frequently results in long wait times, confusion, and dissatisfaction for students. The inefficiency of this method is evident when students pack the registration office, producing disorderly queues that flow throughout the halls and even outside the building. Because of the lack of a systematic queue system, some students are forced to wait for hours before being advised to return the next day due to the large number of applications. This chaotic environment not only impacts students, but also places enormous strain on administrative personnel, who struggle to manage the large number of enrollments properly.

Furthermore, the manual approach does not give real-time updates on queue status, leaving students unsure about their wait times and progress. This ambiguity causes frequent queries and interruptions, which further slows down the enrolling process. The lack of a systematic method to prioritizing different types of students, such as freshmen, transferees, and those who need advising, exacerbates the situation. A more effective and user-friendly queue management system is clearly required to simplify the process, minimize wait times, and enhance overall satisfaction for both students and staff. Implementing a technical solution that can manage real-time updates, prioritize different sorts of students, and give a clear, ordered structure is critical for tackling these difficulties and improving the enrolling process at Mapua Malayan Colleges Mindanao.

**B. Objectives**

The primary objective of this project is to design and implement an efficient enrollment queuing management system for Mapua Malayan Colleges Mindanao to streamline the enrollment process, reduce wait times, and enhance the overall experience for students and administrative staff. Below are the specific objectives:

1. **Queuing**. This system enables students to select either the enrollment or advising option, allowing them to be added to the respective queuing line efficiently.
2. **Real-Time Display.** It provides real-time updates on queue status, ensuring that students are kept informed throughout the process.
3. **Admin Control.** This system offers an enhanced method for managing the queuing process for the administration. Through various control features, it focuses on the operations and ensures a more organized and efficient system.
4. **Notifications.** This system offers a notification feature for students who opt to receive updates via SMS or email. When their ticket number approaches the currently served number, they will receive a notification.

** C. Project Definition**

*Figure 1. Queue Linear Data Structure Operation (FIFO)*

This project involves the development of an enrollment queuing management system for Mapua Malayan Colleges Mindanao utilizing a Queue Linear Data Structure Operation, specifically adhering to the First In, First Out (FIFO) principle. This system ensures that students are served in the exact order of their arrival, thus maintaining fairness and efficiency in the enrollment process. By implementing a queue data structure, the system can accurately manage and update the sequence of students, providing real-time status updates and minimizing confusion and wait times. The FIFO approach is crucial for handling the large volume of students during peak enrollment periods, ensuring that the process is efficient, and that each student receives timely attention based on their position in the queue.

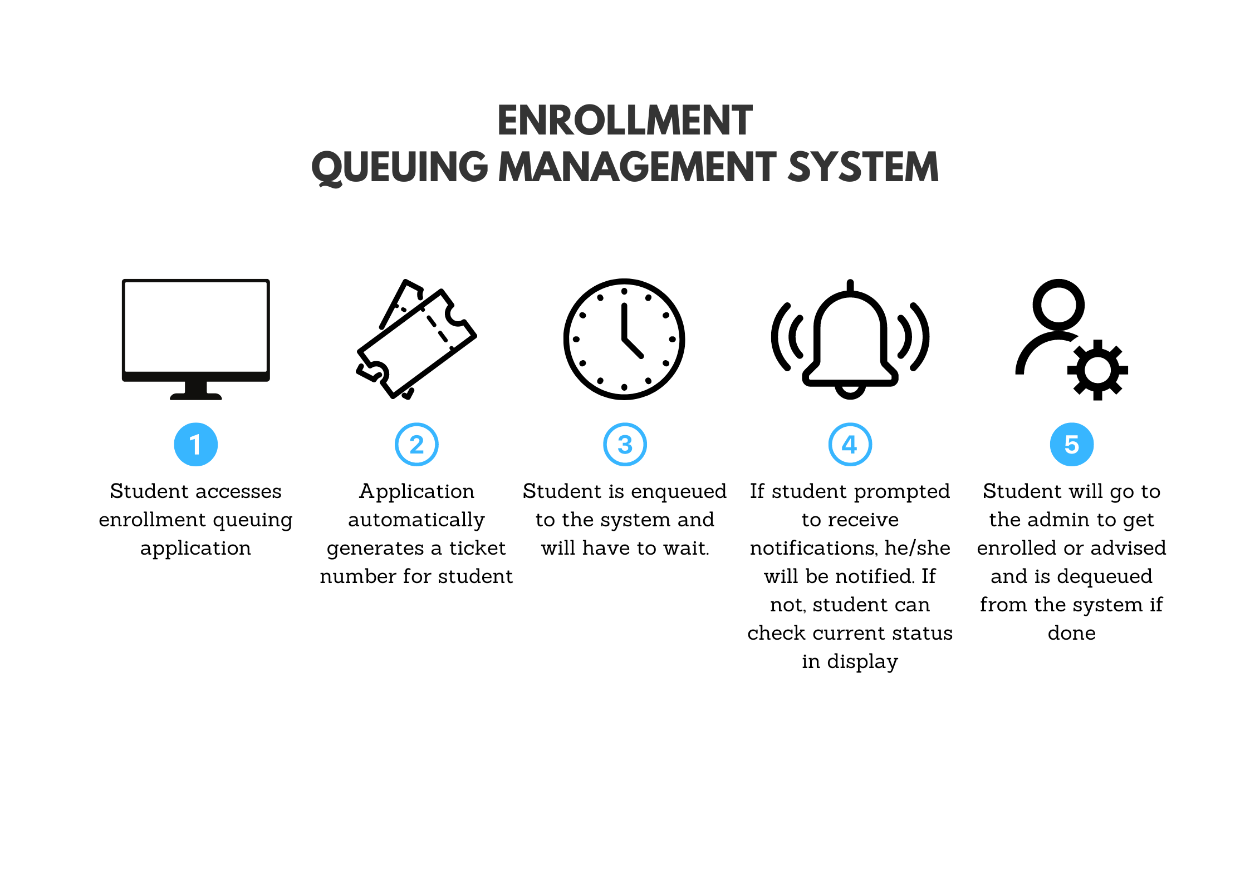
**D. Time and Space Complexity**

A black rectangular sign with white text

Description automatically generatedThe enrollment queuing management system utilizes a Queue Linear Data Structure (FIFO) characterized by efficient time complexity of O(1) for enqueueing and dequeuing operations, ensuring constant-time performance regardless of queue size. This efficiency is essential during high-volume enrollment periods, facilitating quick management of student arrivals and departures without delays. Operations like checking queue status or accessing the front element also maintain O(1) time complexity, enhancing system responsiveness. In terms of space complexity, the system operates linearly at O(n), where n represents the number of students in the queue, ensuring scalability with minimal memory overhead. This design approach optimizes memory usage by efficiently managing memory allocation, guaranteeing effective performance and scalability to meet the demands of the enrollment process at Mapua Malayan Colleges Mindanao.

*Figure 2. Queue Time Complexity in big O notation*

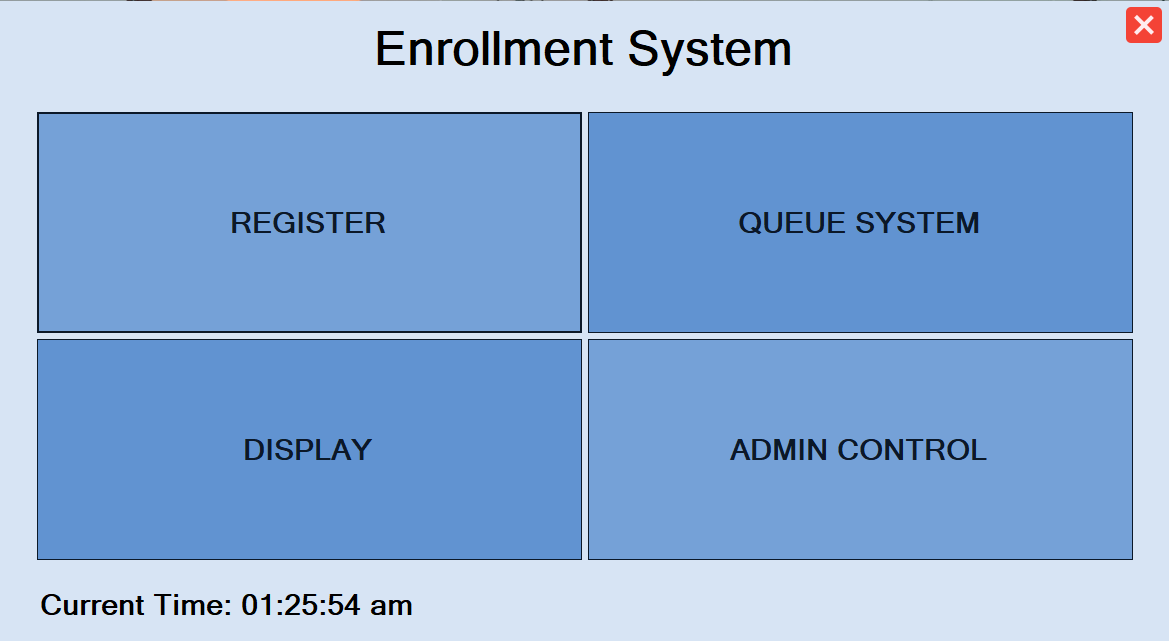
**II. Project/System Prototyping**



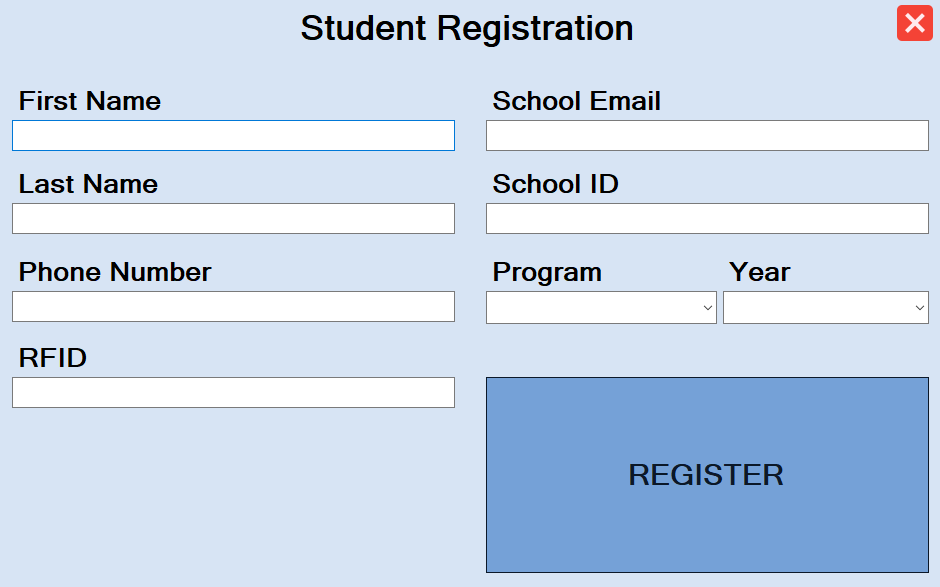
*Figure 3. Enrollment Queuing Management System Flowchart*

The flowchart outlines the operational sequence of the enrollment queuing application designed to enhance the enrollment process at Mapua Malayan Colleges Mindanao. It begins with the student accessing the application, initiating the process by generating a unique ticket number automatically. This step ensures each student is assigned a sequential identifier, adhering to the First In, First Out (FIFO) principle of queue management. Once queued, the student enters a waiting phase, during which the application offers the option to receive notifications about queue progress. If chosen, the student receives updates on their enrollment status, minimizing the need for continuous manual inquiries.

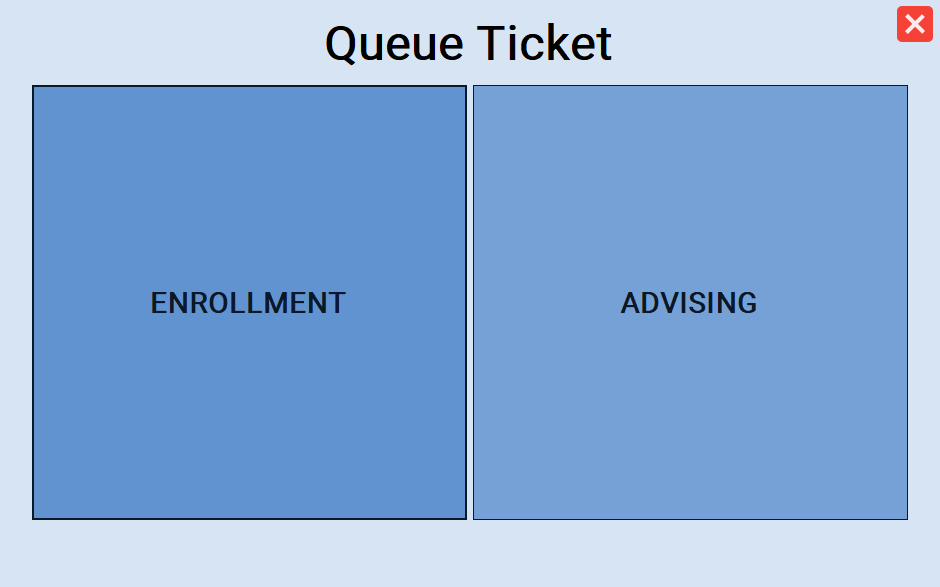
Alternatively, if notifications are declined, students can independently monitor their position in the queue via a display feature within the application. This self-service aspect empowers students with real-time information, reducing uncertainty and congestion around administrative offices. When it's the student's turn to proceed with enrollment or advising, they present themselves to the administrative staff. Upon completion of these processes, such as course enrollment or academic advising, the student is dequeued from the system. This step signifies the removal of their ticket from the queue, ensuring the queue remains accurate and efficiently managed for subsequent students.

**III. Sample Input/Output**

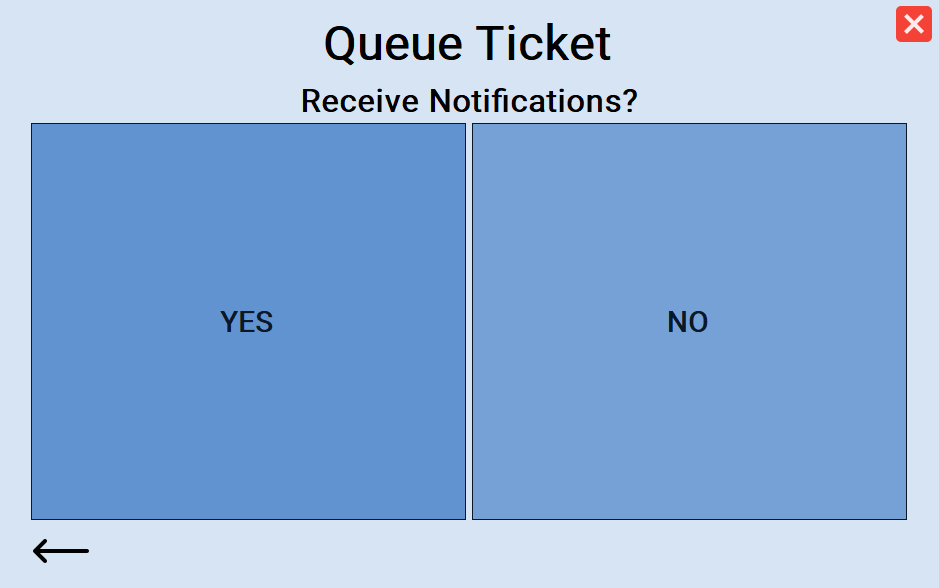
*Figure 4. Enrollment Queuing Management System Main Screen*

There are four distinct buttons corresponding to different functions within the application interface. The "Register" button facilitates student registration, enabling the creation of accounts stored in the database. The "Queue System" button serves as the primary interface for managing the queuing system, allowing students to enter and monitor their queue status efficiently. The "Display" button opens a dedicated screen that displays unit statuses alongside their respective ticket numbers being handled. Lastly, the "Admin Control" button provides administrative access to functionalities such as fetching the first-in ticket number and dequeuing students once their transactions are completed. This structured interface design ensures clear navigation and operational control tailored for both students and administrative users.

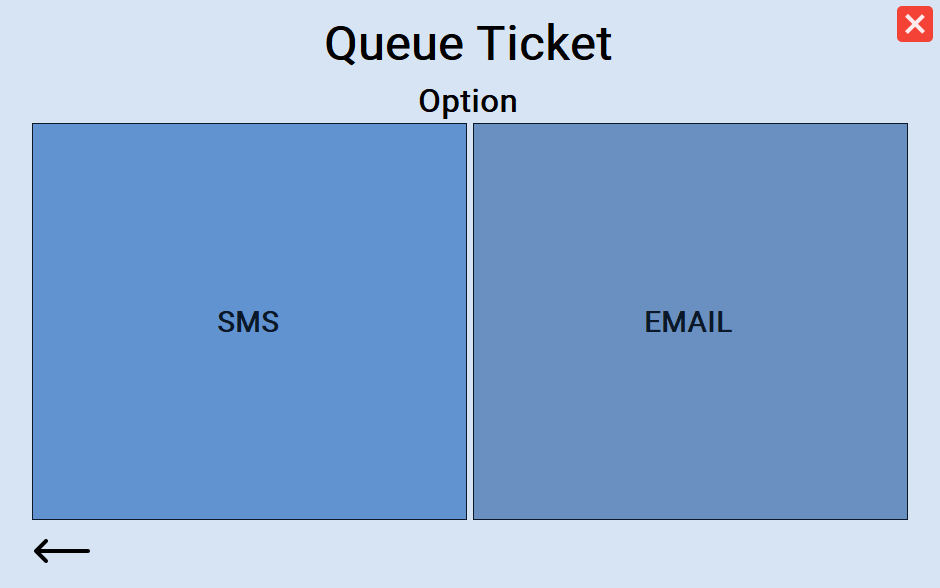
*Figure 5. Enrollment Queuing Management System Student Registration*

**

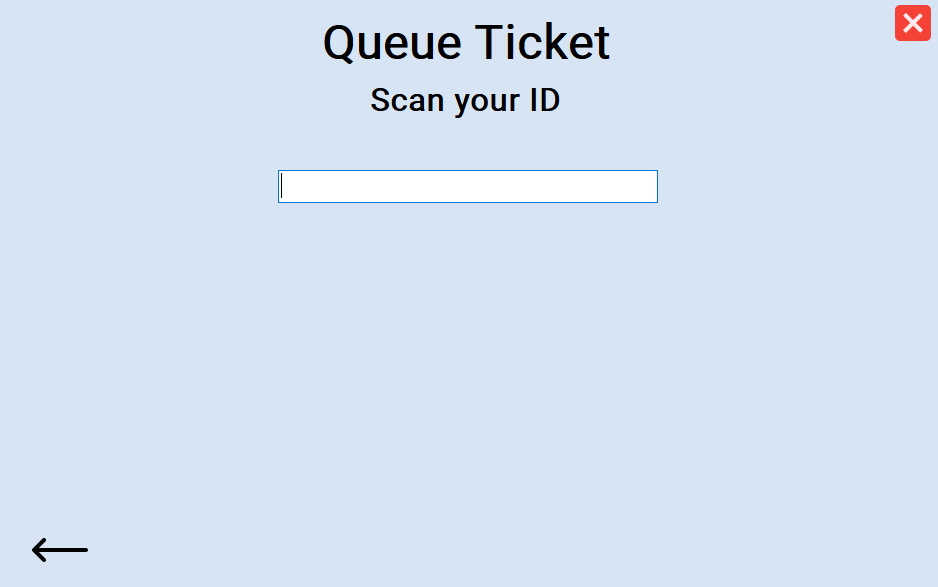
*Figure 6. Enrollment Queuing Management System Queue Ticket*

**

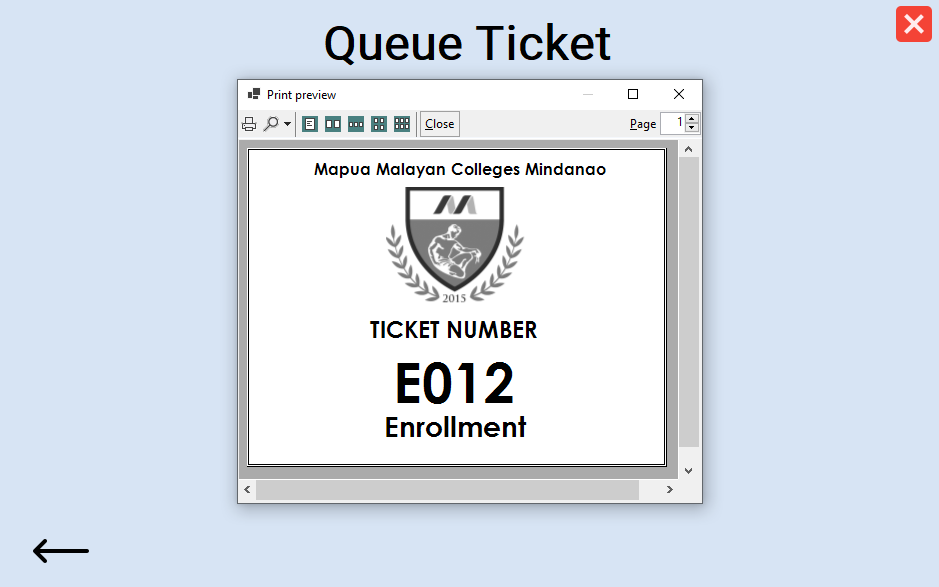
*Figure 7. Enrollment Queuing Management System Notification Prompt*

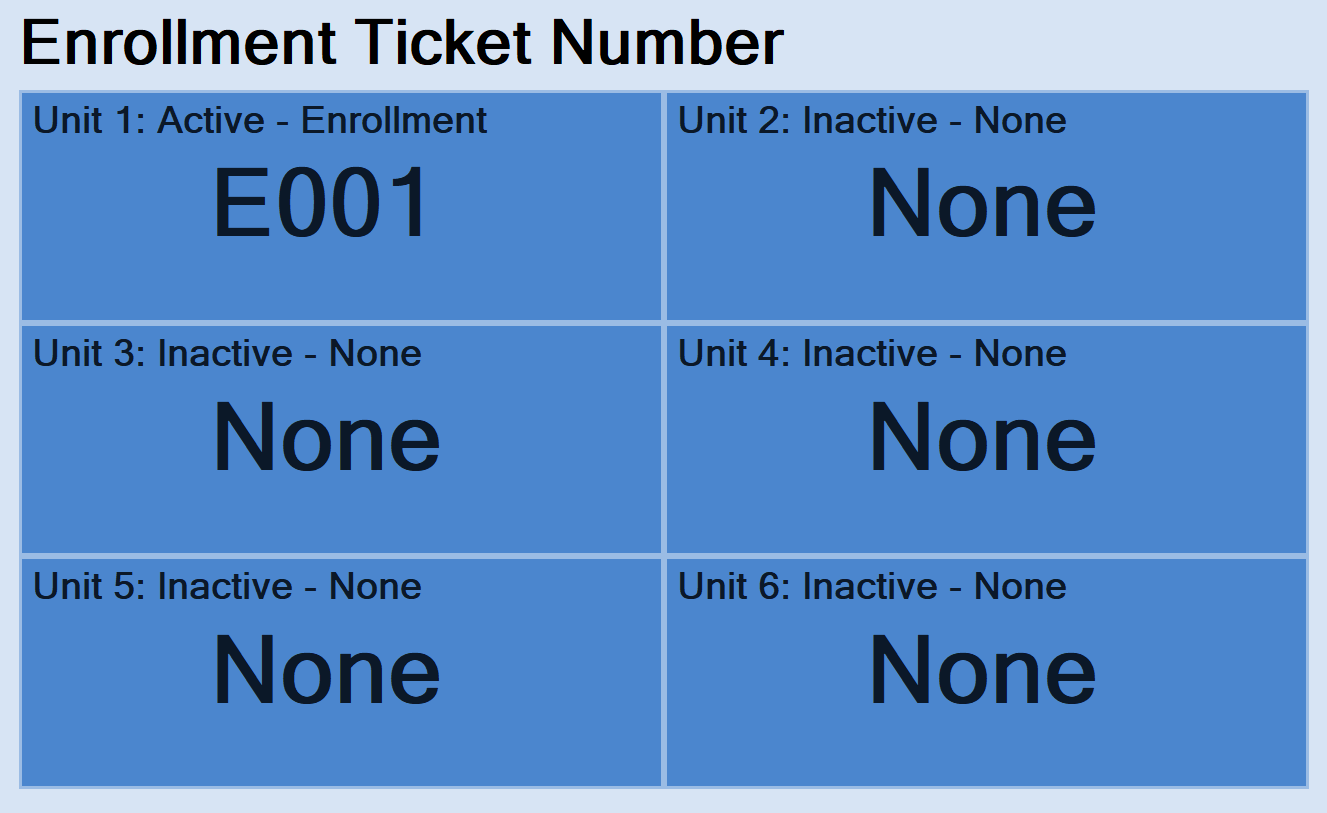


*Figure 8. Enrollment Queuing Management System Option (Yes Prompt)*

**

*Figure 9. Enrollment Queuing Management System RFID Scanner*



*Figure 10. Enrollment Queuing Management System Ticket Generator*

*Figure 11. Enrollment Queuing Management System Display*

A screenshot of a cellphone

Description automatically generated

*Figure 12. Enrollment Queuing Management System Admin Unit*

A blue squares with black text

Description automatically generated

*Figure 13. Enrollment Queuing Management System Admin Type*

A screenshot of a application

Description automatically generated

*Figure 14. Enrollment Queuing Management System Admin Control*

**IV. Source Code**

***Main.cs***

using Enrollment\_System.Database;

using Enrollment\_System.Notifications;

using Enrollment\_System.Entity\_Class;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Security.Cryptography.X509Certificates;

using System.Drawing.Text;

using Microsoft.EntityFrameworkCore.Query.Internal;

namespace Enrollment\_System.Screens

{

public partial class Main : Form

{

private static Main mainInstance;

private string currentStudentEmail1 = "";

private string currentStudentEmail2 = "";

private string currentStudentNumber = "";

public Main()

{

InitializeComponent();

mainInstance = this;

timer1.Start();

}

public static Main GetMainInstance()

{

if (mainInstance == null || mainInstance.IsDisposed)

{

mainInstance = new Main();

}

return mainInstance;

}

private void timer1\_Tick(object sender, EventArgs e)

{

DateTime dateTime = DateTime.Now;

txtTime.Text = dateTime.ToString("hh:mm:ss tt");

advisingtNotif();

enrollmentNotif();

}

private void enrollmentNotif()

{

var CheckTicket = DBHelper.Instance.CheckEnrollmentTickets();

string studentNotif = DBHelper.Instance.getStudentNotif(CheckTicket);

var notifStatus = DBHelper.Instance.getNotifStatus(CheckTicket);

if (CheckTicket != null && notifStatus == "None")

{

if (studentNotif == "Email")

{

string studentEmail = DBHelper.Instance.getStudentEmail(CheckTicket);

if (studentEmail != currentStudentEmail1)

{

Accounts account = new Accounts

{

email = studentEmail

};

currentStudentEmail1 = studentEmail;

EmailSender.SendEmailAsync(account);

MessageBox.Show(CheckTicket);

int updateNotifStatus = DBHelper.Instance.updateNotifStatus(CheckTicket);

if (updateNotifStatus < 0)

{

MessageBox.Show("Error updating ticket notifications", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

}

}

if (studentNotif == "SMS")

{

string studentNumber = DBHelper.Instance.getStudentNumber(CheckTicket);

string modified = studentNumber.Substring(1);

string finalNumber = "+63" + modified;

if (finalNumber != currentStudentNumber)

{

Accounts account = new Accounts

{

phoneNumber = finalNumber

};

currentStudentNumber = finalNumber;

TextSender.SendTextAsync(account);

}

}

}

}

private void advisingtNotif()

{

var CheckTicket = DBHelper.Instance.CheckAdvisingTickets();

string studentNotif = DBHelper.Instance.getStudentNotif(CheckTicket);

var notifStatus = DBHelper.Instance.getNotifStatus(CheckTicket);

if (CheckTicket != null && notifStatus == "None")

{

if (studentNotif == "Email")

{

string studentEmail = DBHelper.Instance.getStudentEmail(CheckTicket);

if (studentEmail != currentStudentEmail2)

{

Accounts account = new Accounts

{

email = studentEmail

};

currentStudentEmail2 = studentEmail;

EmailSender.SendEmailAsync(account);

MessageBox.Show(CheckTicket);

int updateNotifStatus = DBHelper.Instance.updateNotifStatus(CheckTicket);

if (updateNotifStatus < 0)

{

MessageBox.Show("Error updating ticket notifications", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

}

}

if (studentNotif == "SMS")

{

string studentNumber = DBHelper.Instance.getStudentNumber(CheckTicket);

string modified = studentNumber.Substring(1);

string finalNumber = "+63" + modified;

if (finalNumber != currentStudentNumber)

{

Accounts account = new Accounts

{

phoneNumber = finalNumber

};

currentStudentNumber = finalNumber;

TextSender.SendTextAsync(account);

}

}

}

}

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

{

Register.GetRegisterInstance().Show();

this.Hide();

}

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

{

Queue.GetQueueInstance().Show();

this.Hide();

}

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

{

Admin.GetAdminInstance().Show();

this.Hide();

}

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

{

Display displayForm = Display.GetDisplayInstance();

displayForm.Show();

}

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

{

Environment.Exit(0);

}

}

}

***Register.cs***

using Enrollment\_System.Database;

using Enrollment\_System.Entity\_Class;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Globalization;

using System.Linq;

using System.Net.Mail;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Enrollment\_System.Screens

{

public partial class Register : Form

{

private static Register registerInstance;

private string[] programLists =

{

"BSA", "BSE", "BSMA", "BSREM", "BSTM",

"BACOMM", "BMMA",

"BSCS", "BSEMC", "BSIS",

"BSAr", "BSChE", "BSCE", "BSCpE", "BSEE", "BSECE", "BSIE", "BSME",

"BSBio", "BSPsy", "BSPh", "BSPT"

};

private string[] yearLists =

{

"1st Year", "2nd Year", "3rd Year", "4th Year", "5th Year"

};

public Register()

{

InitializeComponent();

registerInstance = this;

this.KeyDown += tbRFID\_KeyDown;

InitializeComboBox();

}

public static Register GetRegisterInstance()

{

if (registerInstance == null || registerInstance.IsDisposed)

{

registerInstance = new Register();

}

return registerInstance;

}

private void Register\_Load(object sender, EventArgs e)

{

cbProgram.Click += new EventHandler(ComboBox\_ShowDropdown);

cbProgram.Enter += new EventHandler(ComboBox\_ShowDropdown);

cbYear.Click += new EventHandler(ComboBox\_ShowDropdown);

cbYear.Enter += new EventHandler(ComboBox\_ShowDropdown);

}

private void InitializeComboBox()

{

cbYear.Items.AddRange(yearLists);

cbProgram.Items.AddRange(programLists);

cbProgram.TextUpdate += cbProgram\_TextUpdate;

}

public void clearInputs()

{

tbFirstName.Clear();

tbLastName.Clear();

tbEmail.Clear();

tbID.Clear();

tbPhoneNum.Clear();

cbProgram.SelectedIndex = -1;

cbYear.SelectedIndex = -1;

tbRFID.Clear();

}

private void registration()

{

var \_firstName = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(tbFirstName.Text.ToLower());

var \_lastName = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(tbLastName.Text.ToLower());

var \_schoolEmail = tbEmail.Text;

var \_schoolID = tbID.Text;

var \_phoneNum = tbPhoneNum.Text;

var \_program = cbProgram.Text;

var \_year = cbYear.Text;

var \_RFID = tbRFID.Text;

if (String.IsNullOrEmpty(\_firstName) || String.IsNullOrEmpty(\_lastName) || String.IsNullOrEmpty(\_schoolEmail) ||

String.IsNullOrEmpty(\_schoolID) || String.IsNullOrEmpty(\_phoneNum) || String.IsNullOrEmpty(\_program) ||

String.IsNullOrEmpty(\_year) || String.IsNullOrEmpty(\_RFID))

{

MessageBox.Show("Please input all requirements.");

return;

}

if (!\_schoolEmail.EndsWith("@mcm.edu.ph"))

{

MessageBox.Show("School Email must end with @mcm.edu.ph");

return;

}

if ((!\_phoneNum.StartsWith("09") && !\_phoneNum.StartsWith("08")) || \_phoneNum.Length != 11)

{

MessageBox.Show("Phone number must start with 09 or 08 and be 11 digits long.");

return;

}

var newAccount = new Accounts

{

firstName = \_firstName,

lastName = \_lastName,

email = \_schoolEmail,

schoolID = \_schoolID,

phoneNumber = \_phoneNum,

program = \_program,

year = \_year,

RFID = \_RFID

};

var isAccountFound = DBHelper.Instance.GetUserID(newAccount.RFID);

if (isAccountFound > 0)

{

MessageBox.Show("Student already registered.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

clearInputs();

return;

}

var insertedAccount = DBHelper.Instance.CreateUser(newAccount);

if (insertedAccount == null)

{

MessageBox.Show("Student failed to create account.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

clearInputs();

return;

}

MessageBox.Show("Registration Successful.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

clearInputs();

Main.GetMainInstance().Show();

this.Hide();

}

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

{

registration();

}

private void tbRFID\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

registration();

}

}

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

{

Main.GetMainInstance().Show();

this.Hide();

}

private void cbProgram\_TextUpdate(object sender, EventArgs e)

{

string userInput = cbProgram.Text;

string[] filteredList = programLists

.Where(program => program.IndexOf(userInput, StringComparison.OrdinalIgnoreCase) >= 0)

.ToArray();

UpdateComboBox(filteredList, userInput);

}

private void UpdateComboBox(string[] filteredList, string userInput)

{

cbProgram.Items.Clear();

cbProgram.Items.AddRange(filteredList);

cbProgram.DroppedDown = true;

cbProgram.Select(cbProgram.Text.Length, 0);

Cursor.Current = Cursors.Default;

}

private void ComboBox\_ShowDropdown(object sender, EventArgs e)

{

ComboBox? comboBox = sender as ComboBox;

if (comboBox != null )

{

comboBox.DroppedDown = true;

}

}

}

}

***Display.cs***

using Enrollment\_System.Database;

using Enrollment\_System.Entity\_Class;

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Enrollment\_System.Screens

{

public partial class Display : Form

{

private static Display displayInstance;

private float originalWidth;

private float originalHeight;

public Display()

{

InitializeComponent();

displayInstance = this;

timer1.Start();

originalWidth = this.Width;

originalHeight = this.Height;

}

public static Display GetDisplayInstance()

{

if (displayInstance == null || displayInstance.IsDisposed)

{

displayInstance = new Display();

}

return displayInstance;

}

private void timer1\_Tick(object sender, EventArgs e)

{

var units = new string[] { "unit\_1", "unit\_2", "unit\_3", "unit\_4", "unit\_5", "unit\_6" };

var labels = new Label[] { lblUnit1, lblUnit2, lblUnit3, lblUnit4, lblUnit5, lblUnit6 };

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

{

var unitStatus = DBHelper.Instance.getSpecificUnitStatus(units[i]);

labels[i].Text = $"Unit {i + 1}: {unitStatus}";

}

var ticketLabels = new Label[] { lblNum1, lblNum2, lblNum3, lblNum4, lblNum5, lblNum6 };

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

{

var ticketNum = DBHelper.Instance.getCurrentNumber(units[i]);

ticketLabels[i].Text = ticketNum;

}

}

private void Display\_FormClosed(object sender, FormClosedEventArgs e)

{

Main.GetMainInstance().Show();

this.Hide();

}

private void Display\_Resize(object sender, EventArgs e)

{

if (this.WindowState != FormWindowState.Minimized)

{

float scaleX = (float)this.Width / originalWidth;

float scaleY = (float)this.Height / originalHeight;

foreach (Control control in this.Controls)

{

control.Scale(new SizeF(scaleX, scaleY));

}

originalWidth = this.Width;

originalHeight = this.Height;

}

}

}

}

***Queue.cs***

using Enrollment\_System.Database;

using Enrollment\_System.Entity\_Class;

using System.Drawing.Printing;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Enrollment\_System.Screens

{

public partial class Queue : Form

{

private static Queue queueInstance;

private string \_studentType;

private string \_notifOptions = "None";

private bool \_EnrollmentNotif;

private bool \_AdvisingNotif;

private int xpos = 500;

private int ypos = 380;

public Queue()

{

InitializeComponent();

timer1.Start();

}

private void startingPanel()

{

pnlOption.Visible = true;

pnlNotif.Visible = false;

pnlMail.Visible = false;

pnlRFID.Visible = false;

}

private void Queue\_Load(object sender, EventArgs e)

{

startingPanel();

}

public static Queue GetQueueInstance()

{

if (queueInstance == null || queueInstance.IsDisposed)

{

queueInstance = new Queue();

}

return queueInstance;

}

private void timer1\_Tick(object sender, EventArgs e)

{

int enrollNum = DBHelper.Instance.QueueRowCount("Enrollment");

int adviseNum = DBHelper.Instance.QueueRowCount("Advising");

\_EnrollmentNotif = enrollNum < 10 ? false : true;

\_AdvisingNotif = adviseNum < 10 ? false : true;

}

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

{

Main.GetMainInstance().Show();

this.Hide();

startingPanel();

}

private void SetVisibility(Control controlToShow, params Control[] controlsToHide)

{

foreach (var control in controlsToHide)

{

control.Visible = false;

}

controlToShow.Visible = true;

}

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

{

if (\_EnrollmentNotif)

{

SetVisibility(pnlNotif, pnlOption, pnlMail);

}

else

{

SetVisibility(pnlRFID, pnlOption);

btnBack3.Visible = false;

tbScan.Focus();

}

\_studentType = "Enrollment";

}

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

{

if (\_AdvisingNotif)

{

SetVisibility(pnlNotif, pnlOption, pnlMail);

}

else

{

SetVisibility(pnlRFID, pnlOption);

btnBack3.Visible = false;

tbScan.Focus();

}

\_studentType = "Advising";

}

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

{

SetVisibility(pnlMail, pnlNotif);

}

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

{

SetVisibility(pnlRFID, pnlNotif);

\_notifOptions = "None";

tbScan.Focus();

}

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

{

SetVisibility(pnlOption, pnlMail, pnlNotif);

}

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

{

SetVisibility(pnlRFID, pnlMail);

\_notifOptions = "SMS";

btnBack3.Visible = true;

tbScan.Focus();

}

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

{

SetVisibility(pnlRFID, pnlMail);

\_notifOptions = "Email";

btnBack3.Visible = true;

tbScan.Focus();

}

private void btnBack2\_Click(object sender, EventArgs e)

{

SetVisibility(pnlNotif, pnlMail);

}

private void btnBack3\_Click(object sender, EventArgs e)

{

SetVisibility(pnlNotif, pnlRFID);

}

private void tbScan\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

var \_RFID = tbScan.Text.Trim();

if (String.IsNullOrEmpty(\_RFID))

{

MessageBox.Show("Please input all requirements");

return;

}

DateTime datetime = DateTime.Now;

var isAccountFound = DBHelper.Instance.GetUserID(\_RFID);

if (isAccountFound < 0)

{

MessageBox.Show("Student not found. Please Try Again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

tbScan.Clear();

return;

}

var GetStudentID = DBHelper.Instance.getStudentID(\_RFID);

var isTicketFound = DBHelper.Instance.isTicketFound(GetStudentID);

if (isTicketFound > 0)

{

MessageBox.Show("Student already generated a ticket.", "Ticket already found", MessageBoxButtons.OK, MessageBoxIcon.Warning);

tbScan.Clear();

return;

}

var newTicket = new Tickets

{

studentID = $"{GetStudentID}",

studentType = \_studentType,

dateTime = datetime.ToString("MM/dd/yyyy hh:mm:ss tt"),

notifOptions = \_notifOptions,

};

var insertTicket = DBHelper.Instance.CreateTicket(newTicket);

if (insertTicket == null)

{

MessageBox.Show("Ticket failed to create.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

tbScan.Clear();

return;

}

printDocument1.DefaultPageSettings.PaperSize = new PaperSize("pprnm", xpos, ypos);

using (PrintPreviewDialog printPreviewDialog1 = new PrintPreviewDialog())

{

printPreviewDialog1.ClientSize = new Size(250, 300);

printPreviewDialog1.StartPosition = FormStartPosition.CenterScreen;

printPreviewDialog1.Document = printDocument1;

if (printPreviewDialog1.ShowDialog() == DialogResult.OK)

{

printDocument1.Print();

}

}

tbScan.Clear();

pnlOption.Visible = true;

pnlNotif.Visible = false;

pnlMail.Visible = false;

pnlRFID.Visible = false;

Main.GetMainInstance().Show();

this.Hide();

}

}

private void printDocument1\_PrintPage(object sender, PrintPageEventArgs e)

{

e.Graphics.DrawString(

"Mapua Malayan Colleges Mindanao",

new Font("Century Gothic", 14, FontStyle.Bold),

Brushes.Black,

new Point(xpos / 2 - 174, 12)

);

string logoPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Media", "Logo.png");

Image logo = Image.FromFile(logoPath);

e.Graphics.DrawImage(logo, new Rectangle(xpos / 2 - 85, 45, 167, 140));

e.Graphics.DrawString(

"TICKET NUMBER",

new Font("Century Gothic", 20, FontStyle.Bold),

Brushes.Black,

new Point(xpos / 2 - 108, 200)

);

var maxTicketNum = DBHelper.Instance.getMaxNumber(\_studentType);

Font fontTicket = new Font("Century Gothic", 48, FontStyle.Bold);

SizeF textSize = e.Graphics.MeasureString(maxTicketNum, fontTicket);

e.Graphics.DrawString(

maxTicketNum,

fontTicket,

Brushes.Black,

new PointF((e.PageBounds.Width / 2) - (textSize.Width / 2), 240)

);

Font fontType = new Font("Century Gothic", 24, FontStyle.Bold);

SizeF typeSize = e.Graphics.MeasureString(\_studentType, fontType);

e.Graphics.DrawString(

\_studentType,

fontType,

Brushes.Black,

new PointF((e.PageBounds.Width / 2) - (typeSize.Width / 2), 315)

);

}

}

}

***Admin.cs***

using Enrollment\_System.Database;

using Enrollment\_System.Entity\_Class;

namespace Enrollment\_System.Screens

{

public partial class Admin : Form

{

private static Admin adminInstance;

private int currentUnit;

public Admin()

{

InitializeComponent();

timer1.Start();

}

private void Admin\_Load(object sender, EventArgs e)

{

pnlUnits.Visible = true;

pnlOption.Visible = false;

pnlCtrl.Visible = false;

}

public static Admin GetAdminInstance()

{

if (adminInstance == null || adminInstance.IsDisposed)

{

adminInstance = new Admin();

}

return adminInstance;

}

private void Admin\_FormClosed(object sender, FormClosedEventArgs e)

{

if (currentUnit > 0)

{

updateUnitStatus($"unit\_{currentUnit}", "Inactive", "None");

}

pnlUnits.Visible = true;

pnlOption.Visible = false;

pnlCtrl.Visible = false;

Main.GetMainInstance().Show();

this.Hide();

}

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

{

lblOption.Text = "Enrollment";

pnlOption.Visible = false;

pnlCtrl.Visible = true;

btnNxtEnroll.Enabled = true;

btnNxtAdvise.Enabled = false;

updateUnitStatus($"unit\_{currentUnit}", "Active", "Enrollment");

}

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

{

lblOption.Text = "Advising";

pnlOption.Visible = false;

pnlCtrl.Visible = true;

btnNxtEnroll.Enabled = true;

btnNxtAdvise.Enabled = true;

updateUnitStatus($"unit\_{currentUnit}", "Active", "Advising");

}

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

{

if (lblOption.Text == "Enrollment")

{

lblOption.Text = "Advising";

btnNxtEnroll.Enabled = true;

btnNxtAdvise.Enabled = true;

updateUnitStatus($"unit\_{currentUnit}", "Active", "Advising");

}

else if (lblOption.Text == "Advising")

{

lblOption.Text = "Enrollment";

btnNxtEnroll.Enabled = true;

btnNxtAdvise.Enabled = false;

updateUnitStatus($"unit\_{currentUnit}", "Active", "Enrollment");

}

}

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

{

updateUnitStatus($"unit\_{currentUnit}", "Inactive", "None");

pnlOption.Visible = false;

pnlCtrl.Visible = false;

pnlUnits.Visible = true;

Main.GetMainInstance().Show();

this.Hide();

}

private void timer1\_Tick(object sender, EventArgs e)

{

string tbEnrollment = DBHelper.Instance.getTicketNumber("Enrollment");

dspEnroll.Text = tbEnrollment;

string tbAdvising = DBHelper.Instance.getTicketNumber("Advising");

dspAdvise.Text = tbAdvising;

LoadUnitStatus();

var currentNumber = DBHelper.Instance.getCurrentNumber($"unit\_{currentUnit}");

currentUnitNum.Text = currentNumber;

var studentName = DBHelper.Instance.getStudentName(currentNumber);

lblStudentID.Text = $"Name: {studentName}";

if (lblOption.Text == "Advising" && currentUnitNum.Text.StartsWith("A"))

{

btnSwitch.Enabled = false;

} else {

btnSwitch.Enabled = true;

}

}

public void updateUnitStatus(string unit, string status, string type)

{

var newUpdate = new Units();

newUpdate.currentUnit = unit;

newUpdate.currentStatus = status;

newUpdate.currentType = type;

int updateUnitStatus = DBHelper.Instance.updateUnit(newUpdate);

if (updateUnitStatus < 0)

{

MessageBox.Show("Error choosing unit.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

private void transferUI()

{

pnlUnits.Visible = false;

pnlOption.Visible = true;

}

private void btnUnit1\_Click(object sender, EventArgs e)

{

currentUnit = 1;

transferUI();

}

private void btnUnit2\_Click(object sender, EventArgs e)

{

currentUnit = 2;

transferUI();

}

private void btnUnit3\_Click(object sender, EventArgs e)

{

currentUnit = 3;

transferUI();

}

private void btnUnit4\_Click(object sender, EventArgs e)

{

currentUnit = 4;

transferUI();

}

private void btnUnit5\_Click(object sender, EventArgs e)

{

currentUnit = 5;

transferUI();

}

private void btnUnit6\_Click(object sender, EventArgs e)

{

currentUnit = 6;

transferUI();

}

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

{

updateUnitStatus($"unit\_{currentUnit}", "Inactive", "None");

pnlOption.Visible = false;

pnlUnits.Visible = true;

}

private void LoadUnitStatus()

{

var unitsStatus = DBHelper.Instance.getUnitStatus();

btnUnit1.Enabled = unitsStatus["unit\_1"] != "Active";

btnUnit2.Enabled = unitsStatus["unit\_2"] != "Active";

btnUnit3.Enabled = unitsStatus["unit\_3"] != "Active";

btnUnit4.Enabled = unitsStatus["unit\_4"] != "Active";

btnUnit5.Enabled = unitsStatus["unit\_5"] != "Active";

btnUnit6.Enabled = unitsStatus["unit\_6"] != "Active";

}

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

{

string adviseText = dspAdvise.Text.Trim();

if ((lblOption.Text == "Advising" && adviseText != "None") || currentUnitNum.Text.StartsWith("A"))

{

MessageBox.Show("Advising Queue still available. Prioritize Advising Queue.", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

HandleNextButtonClick("Enrollment", dspEnroll.Text);

}

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

{

if (currentUnitNum.Text.StartsWith("E"))

{

HandleNextButtonClick("Enrollment", dspAdvise.Text);

return;

}

HandleNextButtonClick("Advising", dspAdvise.Text);

}

private void HandleNextButtonClick(string type, string displayText)

{

if (!string.IsNullOrEmpty(currentUnitNum.Text) && currentUnitNum.Text != "None")

{

DBHelper.Instance.deleteMinTicketNumber(type, currentUnitNum.Text);

}

var updateUnit = new Units

{

currentUnit = $"unit\_{currentUnit}",

currentNum = displayText

};

int updateUnitNum = DBHelper.Instance.updateUnitNum(updateUnit);

if (updateUnitNum < 0)

{

MessageBox.Show("Error choosing unit.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

var currentNumber = DBHelper.Instance.getCurrentNumber($"unit\_{currentUnit}");

currentUnitNum.Text = currentNumber;

var update = new Tickets

{

status = "Serving",

unit = $"unit\_{currentUnit}",

ticketNum = displayText

};

var updatedTicket = DBHelper.Instance.updateTicket(update);

if (updatedTicket < 0)

{

MessageBox.Show("Queue Empty.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

if (updatedTicket == 0)

{

MessageBox.Show("Error updating ticket.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

***Accounts.cs***

namespace Enrollment\_System.Entity\_Class

{

public class Accounts

{

public int ID { get; set; }

public string firstName { get; set; }

public string lastName { get; set; }

public string email { get; set; }

public string phoneNumber { get; set; }

public string program { get; set; }

public string year { get; set; }

public string RFID { get; set; }

public string schoolID { get; set; }

}

}

***Units.cs***

namespace Enrollment\_System.Entity\_Class

{

public class Units

{

public string currentUnit { get; set; }

public string currentStatus { get; set; }

public string currentType { get; set; }

public string currentNum { get; set; }

}

}

***Tickets.cs***

namespace Enrollment\_System.Entity\_Class

{

public class Tickets

{

public string ticketNum { get; set; }

public string studentID { get; set; }

public string studentType { get; set; }

public string dateTime { get; set; }

public string notifOptions { get; set; }

public string status { get; set; }

public string unit { get; set; }

}

}

***EmailSender.cs***

using Enrollment\_System.Entity\_Class;

using System;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Enrollment\_System.Notifications

{

public class EmailSender

{

public static async Task SendEmailAsync(Accounts account)

{

try

{

using (MailMessage mail = new MailMessage())

{

mail.From = new MailAddress("c4rboncopier@gmail.com");

mail.To.Add($"{account.email}");

mail.Subject = "Enrollment Queue";

mail.Body = "Your ticket number is approaching. Please proceed to the Enrollment Room.";

mail.IsBodyHtml = true;

using (SmtpClient smtpClient = new SmtpClient("smtp.gmail.com", 587))

{

smtpClient.Credentials = new NetworkCredential("c4rboncopier@gmail.com", "msoy yidw ujuy ftor");

smtpClient.EnableSsl = true;

await smtpClient.SendMailAsync(mail);

}

}

Console.WriteLine("Email sent successfully.");

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

}

}

***TextSender.cs***

using Enrollment\_System.Entity\_Class;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Twilio;

using Twilio.Rest.Api.V2010.Account;

using Twilio.Types;

namespace Enrollment\_System.Notifications

{

// THIS CODE WILL WORK IF TWILIO ACCOUNT IS VERIFIED

public class TextSender

{

public static async Task SendTextAsync(Accounts account)

{

var accountSid = "AC18df8b30742cc5a5572bd0438dc6d4d5";

var authToken = "37e4c1a1799a4f5cf1d376d83112617a";

if (string.IsNullOrEmpty(accountSid) || string.IsNullOrEmpty(authToken))

{

MessageBox.Show("ERROR");

}

TwilioClient.Init(accountSid, authToken);

var messageOptions = new CreateMessageOptions(new PhoneNumber(account.phoneNumber))

{

From = new PhoneNumber("+16189258466"),

Body = "Your ticket number is approaching. Please proceed to the Enrollment Room."

};

try

{

var message = await MessageResource.CreateAsync(messageOptions);

Console.WriteLine(message.Body);

}

catch (Exception ex)

{

MessageBox.Show("ERROR: " + ex.Message);

}

}

}

}

***DBHelper.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Sockets;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using Enrollment\_System.Entity\_Class;

using Microsoft.Data.Sqlite;

using static Microsoft.EntityFrameworkCore.DbLoggerCategory.Database;

namespace Enrollment\_System.Database

{

public sealed class DBHelper

{

private static DBHelper? \_instance = null;

private readonly SqliteConnection? \_connection = null;

public const string DB\_NAME = "EnrollmentSystem.db";

private DBHelper()

{

\_connection = new SqliteConnection("Data Source=" + DB\_NAME);

\_connection.Open();

}

public static DBHelper Instance

{

get

{

if (\_instance == null)

{

\_instance = new DBHelper();

}

return \_instance;

}

}

public Accounts? CreateUser(Accounts accounts)

{

var createUserCommand = \_connection?.CreateCommand();

createUserCommand!.CommandText = $@"

INSERT INTO Students (firstName, lastName, schoolEmail, phoneNumber, program, year, RFID, schoolID)

VALUES ('{accounts.firstName}', '{accounts.lastName}', '{accounts.email}',

'{accounts.phoneNumber}', '{accounts.program}', '{accounts.year}', '{accounts.RFID}', '{accounts.schoolID}')";

try

{

var result = createUserCommand.ExecuteNonQuery();

if (result < 0)

{

return null;

}

accounts.ID = GetUserID(accounts.RFID);

}

catch (Exception)

{

accounts.ID = GetUserID(accounts.RFID);

}

return accounts;

}

public int GetUserID(string RFID)

{

var selectCommand = \_connection?.CreateCommand();

selectCommand!.CommandText = $@"SELECT ID FROM Students WHERE RFID = '{RFID}'";

var reader = selectCommand.ExecuteReader();

if (!reader.Read())

{

return -1;

}

int userId = reader.GetInt32(0);

return userId;

}

public string? getStudentID(string RFID)

{

if (\_connection == null)

{

return null;

}

string? studentID = null;

try

{

using (var selectCommand = \_connection.CreateCommand())

{

selectCommand.CommandText = @"SELECT schoolID FROM Students WHERE RFID = @RFID";

selectCommand.Parameters.AddWithValue("@RFID", RFID);

var result = selectCommand.ExecuteScalar();

if (result != null)

{

studentID = result.ToString();

}

}

}

catch (Exception)

{

// Handle database errors

studentID = null;

}

return studentID;

}

public int? isTicketFound(string studentID)

{

try

{

using (var selectCommand = \_connection.CreateCommand())

{

selectCommand.CommandText = @"SELECT ID FROM Queue WHERE Student = @studentID";

selectCommand.Parameters.AddWithValue("@studentID", studentID);

var reader = selectCommand.ExecuteReader();

if (!reader.Read())

{

reader.Close();

return -1;

}

int ID = reader.GetInt32(0);

reader.Close();

return ID;

}

}

catch(Exception ex)

{

MessageBox.Show("ERROR: " + ex.Message);

return -1;

}

}

public Tickets? CreateTicket(Tickets tickets)

{

var createTicketCommand = \_connection?.CreateCommand();

string prefix = tickets.studentType == "Enrollment" ? "E" : "A";

var getMaxTicketCommand = \_connection?.CreateCommand();

getMaxTicketCommand!.CommandText = $@"

SELECT MAX(ticketNum) FROM Queue WHERE Type = '{tickets.studentType}' AND ticketNum LIKE '{prefix}%'";

string newTicketNum;

try

{

var maxTicketNum = getMaxTicketCommand.ExecuteScalar()?.ToString();

if (!string.IsNullOrEmpty(maxTicketNum))

{

int currentMaxNum = int.Parse(maxTicketNum.Substring(1));

newTicketNum = $"{prefix}{(currentMaxNum + 1).ToString("D3")}";

}

else

{

newTicketNum = $"{prefix}001";

}

}

catch (Exception)

{

newTicketNum = $"{prefix}001";

}

tickets.ticketNum = newTicketNum;

createTicketCommand!.CommandText = $@"

INSERT INTO Queue (ticketNum, Student, Type, dateTime, Notif, NotifSent, Status, Unit)

VALUES ('{tickets.ticketNum}', '{tickets.studentID}', '{tickets.studentType}',

'{tickets.dateTime}', '{tickets.notifOptions}', 'None', 'Waiting', 'None')";

try

{

var result = createTicketCommand.ExecuteNonQuery();

if (result < 0)

{

return null;

}

}

catch (Exception)

{

return null;

}

return tickets;

}

public string getTicketNumber(string type)

{

string prefix = type == "Enrollment" ? "E" : "A";

try

{

using (var getMaxTicketCommand = \_connection?.CreateCommand())

{

if (getMaxTicketCommand == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

getMaxTicketCommand.CommandText = @"

SELECT MIN(ticketNum) FROM Queue

WHERE Type = @type

AND ticketNum LIKE @prefix

AND Status = 'Waiting'";

getMaxTicketCommand.Parameters.AddWithValue("@type", type);

getMaxTicketCommand.Parameters.AddWithValue("@prefix", $"{prefix}%");

var maxTicketNum = getMaxTicketCommand.ExecuteScalar()?.ToString();

if (string.IsNullOrEmpty(maxTicketNum))

{

return "None";

}

return maxTicketNum;

}

}

catch (Exception)

{

return "ERROR";

}

}

public string getMaxNumber(string type)

{

string prefix = type == "Enrollment" ? "E" : "A";

try

{

using (var getMaxTicketCommand = \_connection?.CreateCommand())

{

if (getMaxTicketCommand == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

getMaxTicketCommand.CommandText = @"

SELECT MAX(ticketNum) FROM Queue

WHERE Type = @type

AND ticketNum LIKE @prefix

AND Status = 'Waiting'";

getMaxTicketCommand.Parameters.AddWithValue("@type", type);

getMaxTicketCommand.Parameters.AddWithValue("@prefix", $"{prefix}%");

var maxTicketNum = getMaxTicketCommand.ExecuteScalar()?.ToString();

if (string.IsNullOrEmpty(maxTicketNum))

{

return "None";

}

return maxTicketNum;

}

}

catch (Exception)

{

return "ERROR";

}

}

public string getCurrentNumber(string unitNum)

{

try

{

if (\_connection == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

using (var getCurrentNumberCommand = \_connection.CreateCommand())

{

getCurrentNumberCommand.CommandText = @"

SELECT currentNum FROM Units

WHERE Unit = @unit";

getCurrentNumberCommand.Parameters.AddWithValue("@unit", unitNum);

var currentNum = getCurrentNumberCommand.ExecuteScalar()?.ToString();

if (string.IsNullOrEmpty(currentNum))

{

return "None";

}

return currentNum;

}

}

catch

{

return "ERROR";

}

}

public string getStudentNotif(string ticketNum)

{

try

{

using (var getStudentNotifCommand = \_connection.CreateCommand())

{

getStudentNotifCommand.CommandText = @"

SELECT Notif FROM Queue

WHERE ticketNum = @ticketNum";

getStudentNotifCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

var studentNotif = getStudentNotifCommand.ExecuteScalar()?.ToString();

if (string.IsNullOrEmpty(studentNotif))

{

return "None";

}

return studentNotif;

}

}

catch

{

return "ERROR";

}

}

public string getNotifStatus(string ticketNum)

{

try

{

using (var getNotifStatusCommand = \_connection.CreateCommand())

{

getNotifStatusCommand.CommandText = @"

SELECT NotifSent FROM Queue

WHERE ticketNum = @ticketNum";

getNotifStatusCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

var NotifStatus = getNotifStatusCommand.ExecuteScalar()?.ToString();

if (string.IsNullOrEmpty(NotifStatus))

{

return "ERROR";

}

return NotifStatus;

}

}

catch

{

return "ERROR";

}

}

public string getSpecificUnitStatus(string unit)

{

try

{

if (\_connection == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

using (var getStatusCommand = \_connection.CreateCommand())

{

getStatusCommand.CommandText = @"

SELECT Status, Type FROM Units

WHERE Unit = @unit";

getStatusCommand.Parameters.AddWithValue("@unit", unit);

using (var reader = getStatusCommand.ExecuteReader())

{

if (reader.Read())

{

var currentStatus = reader["Status"].ToString();

var unitType = reader["Type"].ToString();

if (string.IsNullOrEmpty(currentStatus))

{

return "None";

}

return $"{currentStatus} - {unitType}";

}

}

return "None";

}

}

catch (Exception)

{

return "ERROR";

}

}

public string getStudentEmail(string ticketNum)

{

try

{

using (var getStudentEmailCommand = \_connection.CreateCommand())

{

getStudentEmailCommand.CommandText = @"

SELECT s.schoolEmail

FROM Students s

JOIN QUEUE q ON s.schoolID = q.Student

WHERE q.ticketNum = @ticketNum";

getStudentEmailCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

using (var reader = getStudentEmailCommand.ExecuteReader())

{

if (reader.Read())

{

string email = reader["schoolEmail"].ToString();

return email;

}

else

{

return "ERROR";

}

}

}

}

catch (Exception)

{

return "ERROR";

}

}

public string getStudentNumber(string ticketNum)

{

try

{

using (var getStudentEmailCommand = \_connection.CreateCommand())

{

getStudentEmailCommand.CommandText = @"

SELECT s.phoneNumber

FROM Students s

JOIN QUEUE q ON s.schoolID = q.Student

WHERE q.ticketNum = @ticketNum";

getStudentEmailCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

using (var reader = getStudentEmailCommand.ExecuteReader())

{

if (reader.Read())

{

string email = reader["phoneNumber"].ToString();

return email;

}

else

{

return "ERROR";

}

}

}

}

catch (Exception)

{

return "ERROR";

}

}

public string getStudentName(string currentNum)

{

try

{

if (\_connection == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

using (var getStudentNameCommand = \_connection.CreateCommand())

{

getStudentNameCommand.CommandText = @"

SELECT s.firstName, s.lastName

FROM Students s

JOIN Queue q ON s.schoolID = q.Student

WHERE q.ticketNum = @ticketNum";

getStudentNameCommand.Parameters.AddWithValue("@ticketNum", currentNum);

using (var reader = getStudentNameCommand.ExecuteReader())

{

if (reader.Read())

{

string firstName = reader["firstName"].ToString();

string lastName = reader["lastName"].ToString();

return $"{firstName} {lastName}";

}

else

{

return "Student not found";

}

}

}

}

catch (Exception)

{

return "Error retrieving student name";

}

}

public int updateTicket(Tickets tickets)

{

try

{

using (var updateCommand = \_connection?.CreateCommand())

{

if (updateCommand == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

updateCommand.CommandText = @"

UPDATE Queue

SET Status = @status, Unit = @unit

WHERE ticketNum = @ticketnum";

updateCommand.Parameters.AddWithValue("@status", tickets.status);

updateCommand.Parameters.AddWithValue("@unit", tickets.unit);

updateCommand.Parameters.AddWithValue("@ticketnum", tickets.ticketNum);

int result = updateCommand.ExecuteNonQuery();

return result > 0 ? 1 : -1;

}

}

catch

{

return 0;

}

}

public void deleteMinTicketNumber(string type, string ticketNum)

{

try

{

if (\_connection == null)

throw new InvalidOperationException("Database connection is not initialized.");

using (var deleteTicketCommand = \_connection.CreateCommand())

{

deleteTicketCommand.CommandText = @"

DELETE FROM Queue

WHERE Type = @type

AND ticketNum = @ticketNum

AND Status = 'Serving'";

deleteTicketCommand.Parameters.AddWithValue("@type", type);

deleteTicketCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

int rowsAffected = deleteTicketCommand.ExecuteNonQuery();

if (rowsAffected == 0)

{

throw new InvalidOperationException("No rows were deleted. The ticket might not exist or its status is not 'Serving'.");

}

}

}

catch (Exception ex)

{

throw new ApplicationException("Error deleting the minimum ticket number.", ex);

}

}

public int updateUnit(Units units)

{

try

{

using (var updateCommand = \_connection?.CreateCommand())

{

if (updateCommand == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

updateCommand.CommandText = @"

UPDATE Units

SET Status = @status, Type = @type

WHERE Unit = @currentUnit";

updateCommand.Parameters.AddWithValue("@status", units.currentStatus);

updateCommand.Parameters.AddWithValue("@type", units.currentType);

updateCommand.Parameters.AddWithValue("@currentUnit", units.currentUnit);

int result = updateCommand.ExecuteNonQuery();

return result > 0 ? 1 : -1;

}

}

catch

{

return -1;

}

}

public int updateNotifStatus(string ticketNum)

{

try

{

using (var updateCommand = \_connection?.CreateCommand())

{

updateCommand.CommandText = @"

UPDATE Queue

SET NotifSent = @notifUpdate

WHERE ticketNum = @ticketNum";

updateCommand.Parameters.AddWithValue("@notifUpdate", "Sent");

updateCommand.Parameters.AddWithValue("@ticketNum", ticketNum);

int result = updateCommand.ExecuteNonQuery();

return result > 0 ? 1 : -1;

}

}

catch

{

return -1;

}

}

public int updateUnitNum(Units units)

{

try

{

using (var updateCommand = \_connection?.CreateCommand())

{

if (updateCommand == null)

{

throw new InvalidOperationException("Database connection is not initialized.");

}

updateCommand.CommandText = @"

UPDATE Units

SET currentNum = @currentnum

WHERE Unit = @currentUnit";

updateCommand.Parameters.AddWithValue("@currentUnit", units.currentUnit);

updateCommand.Parameters.AddWithValue("@currentnum", units.currentNum);

int result = updateCommand.ExecuteNonQuery();

return result > 0 ? 1 : -1;

}

}

catch

{

return -1;

}

}

public Dictionary<string, string> getUnitStatus()

{

Dictionary<string, string> unitsStatus = new Dictionary<string, string>();

try

{

var getStatusCommand = \_connection?.CreateCommand();

getStatusCommand!.CommandText = $@"

SELECT Unit, Status FROM Units";

var reader = getStatusCommand.ExecuteReader();

while (reader.Read())

{

string unit = reader["Unit"].ToString();

string status = reader["Status"].ToString();

unitsStatus[unit] = status;

}

}

catch (Exception ex)

{

return null;

}

return unitsStatus;

}

public string CheckEnrollmentTickets()

{

try

{

using (var checkTicketCommand = \_connection?.CreateCommand())

{

var smallestEnrollmentTicket = GetSmallestTicketNumberE();

if (smallestEnrollmentTicket.HasValue)

{

var EnrollTicket = smallestEnrollmentTicket + 10;

string enrollTicket = EnrollTicket.Value.ToString("D3");

return $"E{enrollTicket}";

}

return null;

}

}

catch

{

return null;

}

}

public string CheckAdvisingTickets()

{

try

{

using (var checkTicketCommand = \_connection?.CreateCommand())

{

var smallestAdvisingTicket = GetSmallestTicketNumberA();

if (smallestAdvisingTicket.HasValue)

{

var AdviseTicket = smallestAdvisingTicket + 10;

string adviseTicket = AdviseTicket.Value.ToString("D3");

return $"A{adviseTicket}";

}

return null;

}

}

catch

{

return null;

}

}

private int? GetSmallestTicketNumberE()

{

using (var command = \_connection.CreateCommand())

{

command.CommandText = "SELECT MAX(ticketNum) FROM Queue WHERE ticketNum LIKE @ticketType || '%' AND Status = 'Serving'";

command.Parameters.AddWithValue("@ticketType", 'E');

var result = command.ExecuteScalar();

if (result != DBNull.Value && result != null)

{

var ticketNumStr = result.ToString();

if (ticketNumStr.Length > 1)

{

var numericPart = ticketNumStr.Substring(1);

if (int.TryParse(numericPart, out int ticketNum))

{

return ticketNum;

}

}

}

return null;

}

}

private int? GetSmallestTicketNumberA()

{

using (var command = \_connection.CreateCommand())

{

command.CommandText = "SELECT MAX(ticketNum) FROM Queue WHERE ticketNum LIKE @ticketType || '%' AND Status = 'Serving'";

command.Parameters.AddWithValue("@ticketType", 'A');

var result = command.ExecuteScalar();

if (result != DBNull.Value && result != null)

{

var ticketNumStr = result.ToString();

if (ticketNumStr.Length > 1)

{

var numericPart = ticketNumStr.Substring(1);

if (int.TryParse(numericPart, out int ticketNum))

{

return ticketNum;

}

}

}

return null;

}

}

public int QueueRowCount(string type)

{

try

{

using (var command = \_connection.CreateCommand())

{

command.CommandText = "SELECT COUNT(\*) FROM Queue WHERE Type = @type";

command.Parameters.AddWithValue("@type", type);

var result = command.ExecuteScalar();

return result != DBNull.Value && result != null ? Convert.ToInt32(result) : 0;

}

}

catch

{

return 0;

}

}

}

}