Technical solution

using System;

using System.Data;

using System.Windows.Forms;

using BAL; //Include references for BAL and BEL classes.

using BEL;

namespace Physics\_Studio\_2

{

public partial class frmSignup : Form

{

#region instantiation

public Information info = new Information(); // Create a new instance of the Information class.

public Operations opr = new Operations(); // Create a new instance of the Operations class.

#endregion

public frmSignup()

{

InitializeComponent();

}

#region events

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

{

int passLength = txtNewPassword.Text.Length; int userLength = txtNewUsername.Text.Length;

int nameLength = txtNewName.Text.Length;

if (passLength > 6 && userLength > 5 && nameLength > 0)

{

info.Name = txtNewName.Text;

info.userID = txtNewUsername.Text;

info.Password = opr.getHash(txtNewPassword.Text);

int rows = opr.insertStudent(info); // This variable uses the insertStudent() function from the operations class.

// This returns the number of rows affected when passing the query string.

if (rows > 0) // This is used as validation to check the registration was successful.

{

frmLogin loginForm = new frmLogin(); // A new instance of the frmLogin class is created.

bool boolSuccess = teacherConnection(); // Declare a new boolean variable 'boolSuccess' and set call

//theteacherConnection subroutine.

if (boolSuccess == true) // If this the connection is successful, then the overall registration is also successful.

{

MessageBox.Show("Registration successful.", "Report"); // A messagebox showing the confirmation of the

//registration is shown.

Hide(); // The current form is hidden.

loginForm.Show(); // The new form is shown.

}

}

// If the connection is not successful, and boolSuccess is false, the chosen username must already be in use.

else MessageBox.Show("The username you have chosen already exists.", "Error");

}

// If the requirements are not met, display an error message.

else MessageBox.Show("Please check the requirements for the username, name and password.", "Error");

}

private void btnBack\_Click(object sender, EventArgs e) // Closes the current form and reopens the Student Home page.

{

frmLogin frm = new frmLogin();

Hide();

frm.Show();

}

#endregion

private bool teacherConnection() // This checks the validity of the teeacher ID that has been entered by the student.

{

DataTable dtCon = new DataTable();

int teacherRef; // Declare a new integer variable 'teacherRef'.

if (Int32.TryParse(txtConTeacher.Text, out teacherRef)) // The teacherRef variable is set to the text entered in the

{ // txtConTeacher textbox, only if the contents is strictly numerical.

info.teacherRef = teacherRef; // The teacherRef variable in the Information class is set.

dtCon = opr.conStudent(info); // Returns the name of the teacher that the student is connected to.

if (dtCon.Rows.Count > 0) // If the datatable has been populated then the connection is successfull

{

opr.insertConnection(info); // The new connection is input into the database using the insertConnection

// subroutine in the Operations class.

string teacher = dtCon.Rows[0][0].ToString(); // Delcare a new string 'teacher' and set it as the name of the

// teacher found in the dtCon DataTable.

if (teacher.Length > 0) // If the length of this string is more than zero, it means a value has been successfully

{ //returned.

MessageBox.Show("Connected to " + teacher, "Report"); // Display a confirmation message to the user.

return true;

}

// If not, it means the teacher reference was incorrect.

else MessageBox.Show("Incorrect teacher reference", "Error"); return false;

}

}

// If the ID entered was not strictly numerical, display an error message.

else MessageBox.Show("IDs must contain only numerical characters"); return false;

}

}

}

using System;

using System.Data;

using System.Windows.Forms;

using BAL; //Include references for BAL and BEL classes.

using BEL;

namespace Physics\_Studio\_2

{

public partial class frmLogin : Form

{

#region

Information info = new Information(); // Create a new instance of the class 'information'.

Operations opr = new Operations(); // Create a new instance of the class 'operations'.

DataTable dtUserInfo = new DataTable(); // Create a new instance of the class 'DataTable'.

public frmLogin()

{

InitializeComponent();

}

#endregion

#region login

private void Login()

{

dtUserInfo = opr.studentLogin(info); // Returns a DataTable from the ExeReader function in the DbConnection class.

string uType; // Declare a new string variable 'uType' for the type of user.

int switchcase = dtUserInfo.Rows.Count; // Declare a new integer variable 'switchcase' and set it to the number of

//rows in the dtUserInfo DataTable.

#region Case

switch (switchcase)

{

case 0: // If no rows have been added, then the information has not been found in the students table. Hence, the

// teacher table is now checked.

dtUserInfo = opr.teacherLogin(info); // The dtUserInfo DataTable is now set to the DataTable returned by the

// teacherLogin function, which will check the teacher table.

if (dtUserInfo.Rows.Count > 0) // If the dtUserInfo DataTable has now been populated, then the search was

{ // successful.

uType = dtUserInfo.Rows[0][3].ToString().Trim(); // The uType variable is now made equal to the value found in the

// 4th column of the dtUserInfo DataTable.

if (uType == "A") { goto case 3; } // If the user type is "A", then case 3 is run. Hence, the user is an admin.

goto case 2; // If not, then case 2 is run. Hence, the user is a regular teacher.

}

// If the dtUserInfo DataTable has still not been populated, then either the username is incorrect,

// the passsword is incorrect, or both are.

else MessageBox.Show("Your username or password was incorrect.", "Error"); incorrect.

break;

case 1: // If one row has been added to the dtUserInfo DataTable, then the student home is loaded.

frmStudentHome studentForm = new frmStudentHome(info); // Create a new instance of the frmStudentHome

// class, and pass the 'info' object as an argument.

info.teacherRef = Convert.ToInt32(dtUserInfo.Rows[0][2]); // Set the teacherRef variable in the Information class as

// the value that was returned from the database.

Hide(); // Hide the current form.

studentForm.Show(); // Show the new form.

break; // Break from this case.

case 2: // This case loads the teacher home. Note: This case can only be reached via case 0.

frmTeacherHome teacherForm = new frmTeacherHome(info); // Create a new instance of the frmTeacher class, and

// pass the 'info' object as an argument.

Hide(); // Hide the current form.

teacherForm.Show(); // Show the new form.

break; // Break from this case.

case 3: // This case loads the admin home. Note: This case can only be reached via case 0.

frmAdmin adminForm = new frmAdmin(); // Create a new instance of the frmAdmin class, and pass the 'info'

// object as an argument.

Hide(); // Hide the current form.

adminForm.Show(); // Load the new form.

break; // Break from this case.

#endregion

}

}

#endregion

#region events

private void btnSignup\_Click(object sender, EventArgs e) // Hides the current form and shows the 'Sign up' form on the

{ // Click event handler for the btnSignUp button.

frmSignup signupForm = new frmSignup(); // Create a new instance of the frmSignup class.

Hide(); // Hide the current form.

signupForm.Show(); // Show the new form.

}

private void btnLogin\_Click(object sender, EventArgs e) // Calls the Login() subroutine on the Click event handler for

{ // the btnLogin button.

info.userID = txtUsername.Text; // Set the userID property of the Information class using the txtUsername textbox.

info.Password = opr.getHash(txtPassword.Text); // Set the Password property of the Information class using the

// txtPassword textbox and the getHash function in the Operations class.

Login(); // Call the Login() subroutine.

}

#endregion

}

}

using System;

using System.Data;

using System.Windows.Forms;

using BEL;

using BAL;

namespace Physics\_Studio\_2

{

public partial class frmTeacherHome : Form

{

#region instantiation

public Operations opr = new Operations(); // Create a new instance of the Operations class.

public DataTable dtStudents = new DataTable(); // Create a new DataTable.

public Information info = new Information(); // Create a new instance of the Informtation class.

#endregion

public frmTeacherHome(Information infoPassed)

{

InitializeComponent();

dtStudents = opr.getStudents(infoPassed); // Get the students from the database.

info = infoPassed; // Use the Information object passed to get the student information.

}

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

{

dgvStudents.DataSource = dtStudents; // Use dtStudents as the data source for the dgvStudents DataGridView.

}

#region events

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

{

frmLogin frm = new frmLogin(); // Create a new instance of the frmLogin class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

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

{

frmAddClass frm = new frmAddClass(dtStudents, info); // Create a new instance of the frmAddClass class. Pass

// dtStudents and info as arguments.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

#endregion

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

{

frmEditClass frm = new frmEditClass(); // Create a new instance of the frmEditClass class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

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

{

frmViewClasses frm = new frmViewClasses(dtStudents, info); // Create a new instance of the frmViewClass class.

// Pass dtStudents and info as arguments.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

}

}

using System;

using System.Data;

using System.Windows.Forms;

using Newtonsoft.Json;

using BEL;

using BAL;

namespace Physics\_Studio\_2

{

public partial class frmViewClasses : Form

{

Information info = new Information(); // Create a new instance of the Information class.

Operations opr = new Operations(); // Create a new instance of the Operations class.

DataTable dtStudents = new DataTable(); // Create a new DataTable.

public frmViewClasses(DataTable dtPassed, Information infoPassed)

{

InitializeComponent();

dtStudents = dtPassed; // Use the DataTable passed tot get the students.

info = infoPassed; // Use the Information object passed to get the student information.

cbFilterExperiments.SelectedIndex = 0;

}

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

{

dgvStudents.DataSource = dtStudents; // Set the data source for dgvStudents to dtStudents.

getClasses(); // Call the getClasses function.

}

private void cbSelectClass\_SelectedIndexChanged(object sender, EventArgs e)

{

DataTable dtFilter = new DataTable(); // Create a new DataTable.

dtFilter.Columns.Add("StudentID", typeof(string)); // Set the columns for the DataTable.

dtFilter.Columns.Add("Name", typeof(string));

dtFilter.Columns.Add("Class", typeof(string));

foreach (DataRow row in dtStudents.Rows) // For each of the rows in dtStudents/

{

if ((int)row[2] == cbSelectClass.SelectedIndex) // If the class number in the row is

{ // the same as the selected class

dtFilter.ImportRow(row); // Import the current row in the new DataTable dtFilter

}

}

dgvStudents.DataSource = dtFilter; // Set the data source of dgvStudents to this mew DataTable.

}

private void getClasses()

{

for (int i = 1; i <= opr.getClassMaxNumber(info); i++) // Iterates back from the highest class

{ // number currently in the connections table in the database.

cbSelectClass.Items.Add("Class " + i); // Add the each class number as an item to the combobox.

}

cbSelectClass.Text = cbSelectClass.Items[0].ToString(); // Set the text to the first item in the combobox.

}

private void dgvStudents\_CellContentDoubleClick(object sender, DataGridViewCellEventArgs e)

{

string folder = cbFilterExperiments.SelectedItem.ToString(); // Create a new string variable folder. Sets it to the item

// currently selected in the combobox.

string username = dgvStudents.Rows[e.RowIndex].Cells[0].Value.ToString(); // Create a new string username. Sets it to

// the text in the cell that was double clicked.

try

{

string jsonString = System.IO.File.ReadAllText(@"C:\Users\Jacob Winkworth\Documents\Computer Science\" +

folder + @"\" + username + ".text"); // Gets the JSON string

// from the correct file.

DataTable dtResults = JsonConvert.DeserializeObject<DataTable>(jsonString); // Deserializes the JSON string and sets it

// to dtResults.

dgvLoadResults.DataSource = dtResults; // Set the data source for dgvLoadResults to dtResults.

}

catch { MessageBox.Show("This user has not completed the chosen experiment"); } // If the file is not found, it means

// the chose user has not completed the experiment.

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using BAL;

using BEL;

using System.Windows.Forms;

namespace Physics\_Studio\_2

{

public partial class frmAddClass : Form

{

#region instantiation

Information info = new Information(); // Create a new instance of the Information class.

Operations opr = new Operations(); // Create a new instance of the Operations class.

DataTable dtStudents = new DataTable(); // Create a new instance of the DataTable class.

#endregion

public frmAddClass(DataTable dtPassed, Information infoPassed)

{

InitializeComponent();

dtStudents = dtPassed; // Use the DataTable passed to get the students.

info = infoPassed; // Use the Information object passed to get the student information.

}

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

{

dgvStudents.DataSource = dtStudents; // Add the DataTable 'dtStudents' as the new DataSource of the dgvStudents

// DataTable.

txtClassNumber.Text = (opr.getClassMaxNumber(info) + 1).ToString(); // Make the txtClassNumber textbox text equal

// to one more than the maximum class number already found in the connections table.

} // This will ensure the new class number is unique.

private void txtAddStudent\_TextChanged(object sender, EventArgs e) // When the TextChanged event handler is

// triggered for the txtAddStudent textbox, a binary search is run on the entire contents of the DataGridView.

{

List<string> listUserIDs = new List<string>(); // Create a new instance of the List<string> class.

List<string> listNames = new List<string>(); // Create another new instance of the List<string> class.

DataTable dtSearch = new DataTable(); // Create a new instance of the DataTable class.

dtSearch.Columns.Add("StudentID", typeof(string)); // Add a column for the StudentIDs in the dtSearch DataTable.

dtSearch.Columns.Add("Name", typeof(string)); // Add a column for the Names in the dtSearch DataTable.

string input = txtAddStudent.Text; // Declare a new string variable 'input' and make it equal to the text typed into the

// txtAddStudent textbox.

int lenInput = input.Length; // Declare a new integer variable 'lenInput' and make it equal to the length of the 'input'

// variable.

if (lenInput == 0) // If all text is deleted, the datasource for the dgvStudents DataGridView is re-established as the

// global DataTable.

{ // This ensures the Binary Search does not attempt to find an empty string,

dgvStudents.DataSource = dtStudents; // along with displaying all student IDs and names once again.

}

else

{

foreach (DataRow row in dtStudents.Rows) // Populate list with all items from the DataRow that are longer than or

// equal to the length of the input.

{ // This ensures no errors later when searching strings with values out of range.

string item = (string)row[0]; // Declare a new string variable 'item' and set it as the item in the first column of the

// current row.

if (item.Length >= lenInput) { listUserIDs.Add(item); listNames.Add((string)row[1]); }

// If the length of the item is greater than the length of the input, then the item is added to the User IDs list, and the item

// in the second column of the row is added to the Names list

}

int index = BinarySearchRecursive(listUserIDs, input, 0, listUserIDs.Count - 1);

// Declare a new integer variable and use the BinarySearchRecursive function to set it to the index of the last item equal to

// the input.

while (index > -1 && input == listUserIDs[index].ToString().Substring(0, lenInput)) // This while loop iterates back

{ // through the list for each substring of an item equal to the input.

dtSearch.Rows.Add(listUserIDs[index], listNames[index]); // Each row is also added to the user IDs list and the

// Names list.

index--; // The index integer is reduced by 1 each time.

}

dgvStudents.DataSource = dtSearch; // The dgvStudents DataGridView data source is set to the new datatable

// dtSearch.

dgvStudents.Sort(dgvStudents.Columns[0], ListSortDirection.Ascending); // Finally, the dgvStudents DataGridView

// is sorted into alphabetical order.

} // This prevents the need for any other sorting algorithm for the binary search to take place.

public int BinarySearchRecursive(List<string> inputList, string key, int min, int max)

{

if (min > max) { return -1; } // If the minimum index value is greater than the maximum index value, a value of -1 is

// returned. This represents that the 'key' has not been found.

else

{

int length = key.Length; // Declare a new integer variable 'length' and set it equal to the length of the input key.

int mid = (int)Math.Ceiling((double)(min + max) / 2); // Declare a new integer variable 'mid' and set it equal to the

// middle value between the min and max values.

string item = inputList[mid].ToString().Substring(0, length); // Declare a new string variable 'item' and set it to the item

// in the middle of the list passed into the function.

if (key == item) { return mid; } // If the key is equal to this item, then the key has been found and the index for it (mid)

// is returned.

// Otherwise, the key comes before the item alphanumerically, and the function returns itself with the new max value equal to mid - 1.

else if (string.Compare(key, item) > 0) { return BinarySearchRecursive(inputList, key, mid + 1, max);

// Otherwise, the key comes before the item alphanumerically, and the function returns itself with the new max value equal to mid - 1.

else { return BinarySearchRecursive(inputList, key, min, mid - 1); }

}

}

private void btnAdd\_Click(object sender, EventArgs e) // When the Click event handler is triggered for the btnAdd button,

// all selected rows in the dgvStudents DataGridView are added to the dgvAddClass DataGridview,

{ // and also removed from the dgvStudents DataGridView

foreach (DataGridViewRow row in dgvStudents.SelectedRows) // This for loop iterates through all selected rows in the

// dgvStudents DataGridView.

{

dgvAddClass.Rows.Add(row.Cells[0].Value, row.Cells[1].Value); // The current row is added to the dgvAddClass

// DataGridView

dtStudents.Rows.RemoveAt(row.Index); // This row is also removed from the dtStudents DataTable, as to also remove

// it from the dgvStudents DataGridView and ensure the two refect each other.

}

}

private void btnRemove\_Click(object sender, EventArgs e) // When the Click event handler is triggered for the

// btnRemove button, the selected rows in the dgvAddClass DataGridView should be removed and re-added

{ // to the dgvStudents DataGridView.

foreach (DataGridViewRow row in dgvAddClass.SelectedRows) // This for loop iterates through all selected rows in

// the dgvAddClass DataGridView.

{

dtStudents.Rows.Add(row.Cells[0].Value, row.Cells[1].Value); // The row is added to the dtStudents DataTable.

dgvAddClass.Rows.RemoveAt(row.Index); // The row is removed from the dgvAddClass DataGridView.

}

dgvStudents.Sort(dgvStudents.Columns[0], ListSortDirection.Ascending); // The dgvStudents DataGridView is now

// sorted into alphabetical order, to ensure the Binary Search algorithm run successfully.

}

private void btnSave\_Click(object sender, EventArgs e) // When the Click event hander is triggered for the btnSave

// button, the information in the database must be updated.

{

string classNumber = txtClassNumber.Text; // Declare a new string variable 'classNumber' and set it to the value in the

// txtClassNumber textbox.

foreach (DataGridViewRow row in dgvAddClass.Rows) // This for loop iterates through each row of the dgvAddClass

// DataGridView.

{

opr.updateClass(info, classNumber, row.Cells[0].Value.ToString()); // The updateClass subroutine is called using the

// Operations class to update the connections table in the database accordingly.

}

}

private void btnBack\_Click(object sender, EventArgs e) // When the Click event handler is triggered for the btnBack

// button, the current form should hide, and the teacher home should re-open.

{

frmTeacherHome frm = new frmTeacherHome(info); // A new instance of the frmTeacherHome class is created

Hide(); // The current form is hidden

frm.Show(); // The new form is shown

}

// When the RowHeaderMouseDoubleClick event is triggered for the dgvStudents DataGridView, the row that was double clicked should automatically be added to the dgvAddClass DataGridView and removed from the dgvStudents DataGridView.

private void dgvStudents\_RowHeaderMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs e)

{

DataGridViewRow row = dgvStudents.Rows[e.RowIndex]; // Create an instance of the DataGridViewRow class 'row' and

// set it as the row that was double clicked.

dgvAddClass.Rows.Add(row.Cells[0].Value, row.Cells[1].Value); // Add the this row to the dgvAddClass DataGridView

dtStudents.Rows.RemoveAt(e.RowIndex); // Remove this row from the dtStudents DataTable.

}

#endregion

}

}

using System;

using System.Windows.Forms;

using BEL;

using BAL;

namespace Physics\_Studio\_2

{

public partial class frmStudentHome : Form

{

#region instantiation

Operations opr = new Operations(); // Create a new instance of the Operations class.

Information info; // Create an instance of the Information class.

#endregion

public frmStudentHome(Information infoPassed)

{

info = infoPassed; // Use the Information object passed to get the information for the user.

InitializeComponent();

}

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

{

txtYourConnection.Text = opr.conStudent(info).Rows[0][0].ToString(); // Get the name of the connected teacher.

}

#region load experiment

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

{

frmStatWaves frm = new frmStatWaves(info); // Create a new instance of the frmStatwaves class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

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

{

frmDoubleSlit frm = new frmDoubleSlit(info); // Create a new instance of the frmDoubleSlit class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

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

{

frmSnellLaw frm = new frmSnellLaw(info); // Create a new instance of the frmSnellLaw class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

#endregion

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

{

frmLogin frm = new frmLogin(); // Create a new instance of the frmLogin class.

Hide(); // Hide the current form.

frm.Show(); // Show the new form.

}

}

}

using System;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using BEL;

using BAL;

namespace Physics\_Studio\_2

{

public partial class frmStatWaves : Form

{

Operations opr = new Operations();

Information info = new Information();

DataTable dtResults = new DataTable();

int harmonic = 1; double frequency = 40;

public frmStatWaves(Information infoPassed)

{

InitializeComponent();

info = infoPassed;

lblMValue.Text = ((double)tbMass.Value / 10).ToString();

cbHarmonics.SelectedIndex = 0;

checkInvFagainstL.Checked = true;

setColumns();

dgvResults.DataSource = dtResults;

}

private void pnlWave\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

int harmonic = cbHarmonics.SelectedIndex; // Gets the value of the selected harmonic, 1st Harmonic, 2nd harmonic etc.

int coefficient = getCoefficient(harmonic, 715);// Gets the required multiplier for transforming the Sine graph to represent the selected Harmonic.

g.DrawCurve(Pens.White, getPoints(coefficient, 50)); // Draws the positive Sine wave.

g.DrawCurve(Pens.White, getPoints(coefficient + 2, 50)); // Draws the negative Sine wave.

}

private PointF[] getPoints(int coX, int coY) // Function that returns a PointF[] array of x and y coordinates to plot on the panel.

{

double xStep = 3.0 / 12;

double[] xValues = new double[13]; // Declare a new array of doubles that stores the xValues needed to plot the graph

double[] yValues = new double[13]; // Declare a new array of doubles that stores the yValues needed to plot the graph

for (int i = 0; i < 13; i++)

{

xValues[i] = coX \* (i \* xStep); // Calculates the xValues needed to plot the Sine graph. Values are transformed to project the graph onto the panel.

yValues[i] = (coY \* (Math.Sin(xValues[i] \* 2 \* Math.PI))) + 100; // Calculates the yValues needed to plot the Sine graph. Values are transformed to project the graph onto the panel.

}

PointF[] pointArray = xValues.Zip(yValues, (xCoord, yCoord) => (new PointF((float)xCoord, (float)yCoord))).ToArray();

// Uses the LINQ extension to merge and convert the two arrays into a PointF[] array

return pointArray;

}

private int getCoefficient(double h, int value)

{ // A function that uses recursion to calculate the correct coefficient

if (h == 0) { if (value % 2 == 0) { value += 1; } return value; } // needed to transform the Sine wave into the selected harmonic.

value = (int)((h \* value) / (h + 1)); // Uses the sequence Nth term = (h - 1) \* (N-1)th term) / h where h is the harmonic and h > 1

return getCoefficient(h - 1, value);

}

public double getFrequency(double h, double m, double l, double T)

{

double u = 9.5 \* Math.Pow(10, -4);

double waveSpeed = Math.Pow(T / u, 0.5);

double f = (waveSpeed \* (h / (2 \* l)));

return f;

}

private void cbHarmonics\_SelectedIndexChanged(object sender, EventArgs e)

{

harmonic = cbHarmonics.SelectedIndex + 1; // Set the global harmonic variable to the chosen harmonic.

setFrequency(getMass(), getLength()); // Calls the setFrequency subroutine.

pnlWave.Invalidate(); // Triggers all events the the pnlWave panel.

}

private void plotGraph(double xValue, double yValue, int xMax) // Plots the graph.

{

Graph.ChartAreas[0].AxisX.Minimum = 0;

Graph.ChartAreas[0].AxisX.Maximum = xMax;

Graph.Series[0].Points.AddXY(xValue, yValue);

Graph.Series[1].Points.AddXY(xValue, yValue);

}

private void tbFrequency\_Scroll(object sender, EventArgs e)

{

frequency = tbFrequency.Value; // Sets the global frequency variable

lblFValue.Text = frequency.ToString(); // Changes the text of the lblFValue label

}

private void tbMass\_Scroll(object sender, EventArgs e)

{

setFrequency(getMass(), getLength()); // Calls the setFrequency subroutine.

}

private void tbLength\_Scroll(object sender, EventArgs e)

{

setFrequency(getMass(), getLength()); // Call the setFrequency subroutine.

}

private double getMass()

{

double mass = (double)tbMass.Value / 10; // Declares a new double variabke 'mass'.

lblMValue.Text = mass.ToString(); // Changes the text of the lblMValue label

return mass; // Returns the mass.

}

private double getLength()

{

double length = (double)tbLength.Value / 100; // Declares a new double variable 'length'.

lblLValue.Text = length.ToString(); // Changes the text of the lblLValue label

return length; // Returns the length.

}

private void setFrequency(double mass, double length)

{

double tension = 9.81 \* mass; // Declares a new double variable 'tension'.

frequency = (getFrequency(harmonic, mass, length, tension)); // Sets the frequency using the getFrequency function.

tbFrequency.Value = (int)frequency; // Sets the value for the tbFrequency trackbar.

lblFValue.Text = tbFrequency.Value.ToString(); // Sets the text for the lblFValue label.

}

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

{

double invFrequency = 1 / frequency; // Declare new double variables for each piece of data.

double freqSquared = Math.Pow(frequency, 2);

double length = getLength();

double mass = getMass();

double tension = Math.Round(mass \* 9.81, 3);

if (checkInvFagainstL.Checked == true) { plotGraph(length, invFrequency, 1); } // Plots invFrequency against length if checkInvFagainst is selected.

else plotGraph(tension, freqSquared, 10); // Otherwise the tension is plotted against the square of the frequency.

dtResults.Rows.Add(length, mass, (int)frequency, freqSquared, invFrequency, tension);

}

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

{

int indexLast = dgvResults.Rows.Count - 2;

try

{

double dY = Graph.Series[0].Points[indexLast].YValues[0] - Graph.Series[0].Points[0].YValues[0]; // Declare a new double variable 'dY' and set it to the difference between the first and last y coordinates on the graph.

double dX = Graph.Series[0].Points[indexLast].XValue - Graph.Series[0].Points[0].XValue; // Declare a new double variable 'dX' and set it to the difference between the first and last x coordinates on the graph.

double gradient = dY / dX; // Declare a new double variable 'gradient' and set it to the gradient of the graph using the dY and dX variables.

double massperlength; // Declare a new variable massperlength

if (checkInvFagainstL.Checked == true)

{

massperlength = Math.Pow((harmonic \* gradient / 2), 2) \* (getMass() \* 9.81);

} // Calculate the mass per unit length using the gradien if checkInvFagainstL is selected.

else { massperlength = (1 / Math.Pow(2 \* getLength(), 2)) \* (1 / gradient); } // Calculate the mass per unit length using the gradien if checkInvFagainstL is not selected.

txtMassPerLength.Text = massperlength.ToString(); // Set the txtMassPerLength textbox to the value calculated.

}

catch { MessageBox.Show("No data found, please carry out the experiment first.", "Error:"); } // Display this error message if no data is found.

}

private void btnClear\_Click(object sender, EventArgs e) // Clears all trackbars and series from the graph.

{

Graph.Series[0].Points.Clear();

Graph.Series[1].Points.Clear();

dtResults.Rows.Clear();

tbLength.Value = tbLength.Minimum; lblLValue.Text = tbLength.Value.ToString();

tbMass.Value = tbMass.Minimum; lblMValue.Text = tbMass.Value.ToString();

setFrequency(getMass(), getLength());

txtMassPerLength.Clear();

}

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

{

checkInvFagainstL.Checked = true; // Sets checkInvFagainstL.Checked to true.

checkFagainstM.Checked = false; // Sets checkFagainstM.Checked to false;

}

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

{

checkFagainstM.Checked = true; // Sets checkFagainstM.Checked to false;

checkInvFagainstL.Checked = false; // Sets checkInvFagainstL.Checked to false.

}

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

{

string path = @"C:\Users\Jacob Winkworth\Documents\Computer Science\Stattionary Waves\" + info.userID + ".text";

string json = opr.ConvertDataTabletoJSON(dtResults);

try { System.IO.File.WriteAllText(path, json); MessageBox.Show("Your results have been uploaded",

"Upload successful"); }

catch { MessageBox.Show("Please populate the table with results before attempting to upload the data.",

"Upload failed"); }

}

private void setColumns()

{

dtResults.Columns.Add("Length of wire / (m)", typeof(double));

dtResults.Columns.Add("Mass / (kg)", typeof(double));

dtResults.Columns.Add("Frequency / (Hz)", typeof(double));

dtResults.Columns.Add("Frequency squared", typeof(double));

dtResults.Columns.Add("Inverse frequency", typeof(double));

dtResults.Columns.Add("Tension / (N/kg)", typeof(double));

}

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

{

foreach (DataGridViewRow row in dgvResults.SelectedRows) // Deletes the selected rows in the DataGridView control upon the btnDelete\_Click event handler.

{

dtResults.Rows.RemoveAt(row.Index);

}

}

}

}

using System;

using System.Drawing;

using System.Data;

using System.Windows.Forms;

using BEL;

using BAL;

namespace Physics\_Studio\_2

{

public partial class frmDoubleSlit : Form

{

Information info;

Operations opr = new Operations();

DataTable dtResults = new DataTable();

Color graphColour = Color.Red;

private bool doPaint = false; // Boolean doPaint variable prevents painting to the pblScreen panel on startup.

int rVal; int gVal; int bVal;

{

public frmDoubleSlit(Information infoPassed)

{

info = infoPassed;

InitializeComponent();

cbColours.SelectedIndex = 0; // The index of the cbColours combobox is made zero.

addColumns(); // Call the addColumns subroutine

dgvResults.DataSource = dtResults;

}

#region graph

private void plotGraph(double distance, double seperation, double wavelength)

{

double recipDistance = 1 / distance; // The reciprocal of the distance is calculated

Graph.Series[0].Points.Clear(); // The points on the first series of the Graph control are cleared

Graph.Series[0].Color = graphColour; // The graph colour is changed to represent the colour of light used

Graph.ChartAreas[0].AxisX.Maximum = (Convert.ToInt32(lblLambdaValue.Text) \* recipDistance); // The maximum

// of the x axis is made equal to the wavelength mulitplied by the reciprocal of the distance

for (double i = 0; i < 10; i += 0.001) // A For loop iterates through values of i from 0 to 10, with an incrementation of 0.001. These represent the x coordinates for the graph.

{

double yValue = getY(i, distance, wavelength, seperation); // The y value for each value of x is calculated using the getY function.

Graph.Series[0].Points.AddY(yValue); // This y value is added to the graph.

}

}

private double getY(double x, double D, double lambda, double s) // This function uses recursion to output the correct y value for the input of an x value, the distance from the screen, the wavelenght and the slit seperation (see equation in analysis, page 3.

{

double Alpha = (((Math.PI \* s) / lambda) \* Math.Sin(x) \* D); // The value of Alpha is calulated.

double Beta = (((Math.PI \* s / 10) / lambda) \* Math.Sin(x) \* D); // The value of Beta is calculated.

if (x == 0) { return Math.Pow((Math.Cos(Alpha)), 2); } // If x is 0, then return the square of alpha.

else { return getY(0, D, lambda, s) \* Math.Pow(Math.Cos(Alpha), 2) \* Math.Pow(Math.Sin(Beta), 2); }

}

#endregion

#region button events

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

{

double wavelength = (tbWavelength.Value) \* Math.Pow(10, -9); // Declares a new variable 'Wavelength', sets the value from the lblWavelength label and converts from Nanometres to Metres.

double distance = Convert.ToDouble(lblDistance.Text); // Declares a new variable 'Distance', sets the value from the lblDistance label and converts it to an integer.

double seperation = (tbSeperation.Value) \* Math.Pow(10, -4); // Declares a new variable 'Seperation', sets the value from the tbSeperation trackbar and converts it to a double in Metres.

double spacing = Math.Round(getSpacing(wavelength, distance, seperation) \* Math.Pow(10, 3), 2); // Declares a new variable 'Spacing', sets the value from the getSpacing function in the PhysOperations class and converts it to metres

doPaint = true; // Sets the doPaint variable to true.

dtResults.Rows.Add(cbColours.Text, distance, lblSValue.Text, lblLambdaValue.Text, spacing); // Adds the information to the dtResults datatable, which is bound to the dgvResults DataGridView.

pnlScreen.Invalidate(); // Triggers the pnlScreen\_Paint event handler.

plotGraph(distance, seperation, wavelength \* 10); // Call the plotGraph subroutine.

}

private double getSpacing(double λ, double D, double S)

{

double W = (λ \* D) / S;

return W;

}

private void btnBack\_Click(object sender, EventArgs e) // Closes the current form and reopens the Student Home page.

{

Hide();

frmStudentHome frm = new frmStudentHome(info);

frm.Show();

}

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

{

foreach (DataGridViewRow row in dgvResults.SelectedRows) // Deletes the selected rows in the DataGridView.

{

dtResults.Rows.RemoveAt(row.Index);

}

}

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

{

dtResults.Rows.Clear(); // Clears all data from the dgvResults DataGridView.

}

private void btnReset\_Click(object sender, EventArgs e) // Reset the values of the trackbars.

{

tbDistance.Value = tbDistance.Minimum;

tbSeperation.Value = tbSeperation.Minimum;

tbWavelength.Value = tbWavelength.Minimum;

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Windows.Forms;

using BAL;

using BEL;

namespace Physics\_Studio\_2

{

public partial class frmSnellLaw : Form

{

Operations opr = new Operations(); // Create a new instance of the Operations class

Information info = new Information(); // Create a new instance of the Information class

DataTable dtResults = new DataTable(); // Create a new DataTable

double[] indexValues = new double[] { 1, 1.33, 1.66 };

public frmSnellLaw(Information infoPassed)

{

InitializeComponent();

info = infoPassed;

setColumns();

dgvResults.DataSource = dtResults;

cbMedium1.SelectedIndex = 0;

cbMedium2.SelectedIndex = 1;

}

private void pnlExperiment\_Paint(object sender, PaintEventArgs e)

{

float lineHeight = pnlExperiment.Height / 3; // Creates a variable which is equal to the seperation of the on-screen lines

double angleRads = Convert.ToDouble(lblIValue.Text) \* Math.PI / 180; // Converts the angle to radians

double index1 = indexValues[cbMedium1.SelectedIndex];

double index2 = indexValues[cbMedium2.SelectedIndex];

Graphics g = e.Graphics; // Initiates an instance of the graphics class using the PaintEventArgs e

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias; // Set the Graphics smoothing mode to anti alias

Pen blackpen = new Pen(Color.Black); // Initiate a new pen of the colour black

PointF startPoint = new PointF(tbPosition.Value, pnlExperiment.Height); // Creates a start point for line to begin (bottom of the screen)

List<PointF> pointList = getPoints(angleRads, index1, index2, new List<PointF>(), startPoint, lineHeight); // Creates a list of type PointF using the getPoints function

g.DrawRectangle(blackpen, 0, lineHeight, pnlExperiment.Width, lineHeight); // Draw a rectangle on screen which is used as the medium

g.DrawLine(blackpen, startPoint, pointList[0]); // Draws a line from the start point to the first position in pointList

for (int i = 0; i < pointList.Count - 1; i++) // For loop that draws a line from point to point using each item in pointList

{

g.DrawLine(blackpen, pointList[i], pointList[i + 1]); // Draws a line between the current item and the next item in the list

if (i < pointList.Count - 2)

g.DrawLine(Pens.Red, new Point((int)pointList[i].X, (int)pointList[i].Y - 100), new Point((int)pointList[i].X, pnlExperiment.Height)); // Draws the normal line of each point

}

}

private List<PointF> getPoints(double angle, double index1, double index2, List<PointF> pointList, PointF startPoint, double height) // A function that returns a list of points after each refraction.

{

float yStart = startPoint.Y; // Creates a float variable which is the y coordinate of the start posion passed into the function.

double angRefraction = getAngleR(index1, index2, angle); // Creates a double variable which is the angle of refraction using the getAngleR function.

if (yStart >= 0) // If the y coordinate of the start point has not reached the top of the screen.

{

lblRAngle.Text = Math.Abs(Math.Round((angle / (Math.PI / 180)), 2)).ToString(); // Sets display label for the angle of refraction with the angle of refraction in degrees.

float xStart = startPoint.X; // Creates a float variable which is the x coordinate of the start posion passed into the function.

double length = Math.Abs(getLength(angle, height)); // Creates a double variable which is the distance between the current point the and point of incidence.

float xIncident = (float)(xStart + (Math.Sin(angle)) \* length); // Creates a float variable which is the x coordinate of the point incident on the next medium.

float yIncident = (float)(yStart - (Math.Cos(angle)) \* length); // Creates a float variable which is the y coordinate of the point incident on the next medium.

double critAngle = getCritAngle(index1, index2); // Creates a double variable which is the critical angle between the current medium and the next using the getCritAngle function.

if (xIncident > pnlExperiment.Width && yIncident > pnlExperiment.Height / 3) // If the xIncident variable is greater than the width of the panel and the yIncident variable is greater than a third of the panel height.

{ // In other words, if the x coordinate is off the screen and the y coordinate is within the glass block.

height = (xIncident - pnlExperiment.Width - 5) / (float)Math.Tan(angle); // The height variable is altered using trigonometry in order to accomodate for this, so the animation does not go off the screen.

yIncident += (int)height; // This height is added to the yIncident variable.

xIncident = pnlExperiment.Width; angRefraction = -angle; // The xIncident variable is now set to the width of the panel, and the angle of refraction is made to be negative

} // in order to reflect the 'light' back onto the screen.

else height = pnlExperiment.Height / 3; // Otherwise, the height is set to a third of the overall height of the panel.

if (angle > critAngle)

{

angRefraction = Math.PI - angle;

}

PointF pointofIncidence = new PointF(xIncident, yIncident); // A new instance of the pointF class 'pointofIncidence' is created.

pointList.Add(pointofIncidence); // This point is then added to the pointList.

return getPoints(angRefraction, index2, index1, pointList, pointofIncidence, height); // The function now returns itself, with the refractive indexes reversed and the new start point set to the point of incidence.

}

return pointList; // The pointList is returned.

}

private double getAngleR(double ni, double nr, double I) // This function returns the angle of refraction.

{

double angleR; // Declare a new double variable 'angleR'

double sinTheta = ni \* Math.Sin(I) / nr; // Declare a new double variable 'theta', and use the Snell's Law equation to work out the sine of the angle of refraction.

angleR = Math.Asin(sinTheta); // Set the angleR variable to the Arcsine of the sinTheta variable. Hence, working out theta.

return angleR; // Return this angleR.

}

private double getLength(double angle, double height) // This returns the length of the line.

{

double length = height / Math.Cos(angle); // Declare a new double variable and use trigonometry to set it to the length of the hypotenuse when given the height and angle.

return length; // Return this length.

}

private double getCritAngle(double ni, double nr) // This returns the critical angle between the two mediums using the critical angle formula.

{

double critAngle = Math.Asin(nr / ni);

return critAngle;

}

private void tbAngle\_Scroll(object sender, EventArgs e) // When the Scroll event handler is triggered for the tbAngle trackbar, the events for the pnlExperiment panel should be run.

{

pnlExperiment.Invalidate(); // This triggers all events for the pnlExperiments panel to be executed.

lblIValue.Text = ((double)tbAngle.Value / 100).ToString(); // The value for lblIValue is set to a one hundredth of the value of the trackbar.

}

private void tbPosition\_Scroll(object sender, EventArgs e) // When the Scroll event handler is triggered for the tbPosition trackbar, the events for the pnlExperiment panel should be run.

{

pnlExperiment.Invalidate(); // This triggers all events for the pnlExperiments panel to be executed.

}

private void btnPlot\_Click(object sender, EventArgs e) // When the Click event handler for the btnPlot button is triggered,

{ // the data should be added to the dgvResults DataGridView and recorded on the graph.

double angleI = Convert.ToDouble(lblIValue.Text); // Declare a new double variable for the angle of incidence and set it to the contents of the lblIValue label.

double angleR = Convert.ToDouble(lblRAngle.Text); // Declare a new double variable for the angle of refraction and set it to the contents of the lblRefractedAngle label.

double sineI = Math.Round(Math.Sin(angleI \* Math.PI / 180), 5); // Declare a new double variable for the sine of the angle of incidence in radians.

double sineR = Math.Round(Math.Sin(angleR \* Math.PI / 180), 5); // Declare a new double variable for the sine of the angle of refraction in radians.

dtResults.Rows.Add(angleI, angleR, sineI, sineR, Math.Round(sineI / sineR)); // Add this data to the dtResults DataTable, which is the data source for the dgvResults DataGridView.

Graph.ChartAreas[0].AxisX.Maximum = 1; // Set the maximum of the x axis to 1.

Graph.Series[0].Points.AddXY(sineR, sineI); // Add the sineR value as the x coordinate, and the sineI value as the y coordinate, to the series 0 of the graph.

Graph.Series[1].Points.AddXY(sineR, sineI); // Add the sineR value as the x coordinate, and the sineI value as the y coordinate, to the series 1 of the graph.

} // This will display both the points and lines on the graph.

private void btnFindIndex\_Click(object sender, EventArgs e) // When the Click event for the btnFindIndex button is triggered, the refractive index of the material used must be found.

{

int indexLast = dgvResults.Rows.Count - 2; // Declare a new integer variable and set it to the last row populated with data.

try // If dgvResults is not populated with data, then an error message will be shown.

{

double dY = Graph.Series[0].Points[indexLast].YValues[0] - Graph.Series[0].Points[0].YValues[0]; // Declare a new double variable 'dY' and set it to the difference between the first and last y coordinates on the graph.

double dX = Graph.Series[0].Points[indexLast].XValue - Graph.Series[0].Points[0].XValue; // Declare a new double variable 'dX' and set it to the difference between the first and last x coordinates on the graph.

double gradient = Math.Round(dY / dX, 2); // Declare a new double variable 'gradient' and set it to the gradient of the graph using the dY and dX variables.

txtRIndex.Text = gradient.ToString(); // Set the text of the txtRIndex textbox to the gradient variable, as this represents the refractive index of the material.

}

// Display an error message.

catch { MessageBox.Show("No data found, please carry out the experiment first.", "Error:"); }

}

private void btnClear\_Click(object sender, EventArgs e) // When the Click event handler is triggered for the btnClear button, all the data recorded so far should be cleared.

{

dtResults.Rows.Clear(); // Clear the data from the dgvResults DataGridView.

Graph.Series[0].Points.Clear(); // Clear the data from series 0 of the graph.

Graph.Series[1].Points.Clear(); // Clear the data from series 1 of the gaph.

txtRIndex.Clear(); // Clear the text from the txtRIndex textbox.

}

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

{

foreach (DataGridViewRow row in dgvResults.SelectedRows)

{

dtResults.Rows.RemoveAt(row.Index);

}

}

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

{

string path = @"C:\Users\Jacob Winkworth\Documents\Computer Science\Snell's law\" + info.userID + ".text";

string json = opr.ConvertDataTabletoJSON(dtResults);

try { System.IO.File.WriteAllText(path, json); MessageBox.Show("Your results have been uploaded",

Upload successful"); }

catch { MessageBox.Show("Please populate the table with results before attempting to upload the data."); }

}

private void setColumns()

{

dtResults.Columns.Add("Angle incident / º", typeof(double));

dtResults.Columns.Add("Angle refracted / º", typeof(double));

dtResults.Columns.Add("Sin(i)", typeof(double));

dtResults.Columns.Add("Sin(r)", typeof(double));

dtResults.Columns.Add("Sin(i) / Sin(r)", typeof(double));

}

}

}

using System;

using System.Collections.Generic;

using DAL; //Include references for DAL, BEL, System.Data and System.Data.OleDb classes.

using BEL;

using System.Data;

using System.Data.OleDb;

using System.Web.Script.Serialization;

namespace BAL

{

public class Operations

{

public DbConnection db = new DbConnection(); // Create a new instance of the class 'DbInformation'.

public Information info = new Information(); // Create a new instance of the class 'Information'.

#region Inserts

public int insertStudent(Information info) // Create a function to use a query that inserts a new row into the SQL database by calling upon the ExeNonQuery() function.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format

cmd.CommandText = "INSERT INTO tblStudents VALUES (@studID, @Password, @Name)"; // Set the command text to the query string. This string inserts the student ID, their password and their name into the students table.

cmd.Parameters.AddWithValue("@studID", info.userID); // Add the parameter value for the student's ID using the Information class.

cmd.Parameters.AddWithValue("@Password", info.Password); // Add the parameter value for the student's password using the Information class.

cmd.Parameters.AddWithValue("@Name", info.Name); // Add the parameter value for the student's name using the Information class.

return db.ExeNonQuery(cmd); // Use the DbConnection class to pass the command through the ExeNonQuery() function.

}

public int insertTeacher(Information info) // Create a function to use a query that inserts a new row into the SQL database by calling upon the ExeNonQuery() function.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format.

cmd.CommandText = "INSERT INTO tblTeachers VALUES (@teachID, @Password, @Name, @uType)"; // Set the command text to the query string. This string inserts the teacher's ID, their password, their name and their usertype into the teacher table.

cmd.Parameters.AddWithValue("teachID", info.userID); // Add the parameter value for the teacher's ID using the Information class.

cmd.Parameters.AddWithValue("@Password", info.Password); // Add the parameter value for the teacher's password using the Information class.

cmd.Parameters.AddWithValue("@Name", info.Name); // Add the parameter value for the teacher's name using the Information class.

cmd.Parameters.AddWithValue("@uType", "B"); // Add the parameter for the teacher's user type. ("A" is admnin, "B" is for a regular teacher account.

return db.ExeNonQuery(cmd); // Use the DbConnection class to pass the command through the ExeNonQuery() function.

}

public void insertConnection(Information info) // Create a new subroutine that inserts a new connection between a teacher and a student using the ExeNonQuery() function.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format.

cmd.CommandText = "INSERT INTO tblConnections VALUES (@studID, @ref)"; // Set the command text to the query string. This string inserts a student ID and corresponding teacher ID that represetns the connection between the two into the connections table.

cmd.Parameters.AddWithValue("@studID", info.userID); // Add the parameter value for the student ID using the Infomation class.

cmd.Parameters.AddWithValue("@ref", info.teacherRef); // Add the parameter value for the teacher ID using the teacherRef variable in the Information class.

db.ExeNonQuery(cmd); // Use the DbConnection class to pass the command through the ExeNonQuery() function.

}

public void updateClass(Information info, string className, string student) // Create a new subroutine that updates the class of a student using the ExeNonQuery() function.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format.

cmd.CommandText = "UPDATE tblConnections " + // Set the command text to the query string.

"SET class = @classNum” +

“WHERE studentID = @studID AND teacherID = @teachID";

cmd.Parameters.AddWithValue("@studID", student); // Add the parameter value for the student ID using the 'student' parameter.

cmd.Parameters.AddWithValue("@teachID", info.userID);

cmd.Parameters.AddWithValue("@classNum", className); // Add the parameter value for the class number using the 'className' parameter.

db.ExeNonQuery(cmd); // Use the DbConnection class to pass the command through the ExeNonQuery() function.

}

#endregion

#region DataTables

public DataTable studentLogin(Information info) // Create a new function that returns a DataTable populated with the information of the student that has logged in.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format.

// Set the command text to the query string.

cmd.CommandText = "SELECT tblStudents.StudentID, Name, TeacherId, Class from tblStudents” +

“INNER JOIN tblConnections ON tblStudents.StudentID = tblConnections.StudentID " +

"WHERE tblStudents.StudentID = @studID and tblStudents.Password = @Password";

cmd.Parameters.AddWithValue("@studID", info.userID); // Add the parameter value for the student ID using the Information class

cmd.Parameters.AddWithValue("@Password", info.Password); // Add the parameter value for the student's password using the Information class.

return db.ExeReader(cmd); // Use the DbConnection class to pass the command through the ExeReader() function.

}

public DataTable teacherLogin(Information info) // Create a new function that returns a DataTable populated with the information of the teacher that has logged in.

{

OleDbCommand cmd = new OleDbCommand(); // Create a new instance of the OleDBCommand class.

cmd.CommandType = CommandType.Text; // Set the command type to the text format.

cmd.CommandText = "SELECT \* from tblTeachers " + // Set the command text to the query string.

"WHERE TeacherID = @teachID and Password = @Password";

cmd.Parameters.AddWithValue("@teachID", info.userID); // Add the parameter value for the teacher ID using the Information class.

cmd.Parameters.AddWithValue("@Password", info.Password); // Add the paremeter value for the teacher's password using the Information class.

return db.ExeReader(cmd); // Use the DbConnection class to pass the command through the ExeReader() function.

}

public DataTable conStudent(Information info) // Get the name of a teacher connected to a specifc student.

{

OleDbCommand cmd = new OleDbCommand();

cmd.CommandType = CommandType.Text;

cmd.CommandText = "SELECT Name from tblTeachers WHERE TeacherID = @ref";

cmd.Parameters.AddWithValue("@ref", info.teacherRef);

return db.ExeReader(cmd);

}

public DataTable getStudents(Information info) // Get all teachers connected to a specific teacher

{

OleDbCommand cmd = new OleDbCommand();

cmd.CommandType = CommandType.Text;

cmd.CommandText = "SELECT tblStudents.StudentID, tblStudents.Name, tblConnections.Class" +

"FROM tblStudents INNER JOIN tblConnections " +

"ON tblStudents.StudentID = tblConnections.StudentID " +

"WHERE tblConnections.TeacherID = @ID " +

"ORDER BY tblStudents.StudentID ASC";

cmd.Parameters.AddWithValue("@ID", info.userID);

return db.ExeReader(cmd);

}

#endregion

public int getClassMaxNumber(Information info) // Gets the maximum class number from the connections table.

{

OleDbCommand cmd = new OleDbCommand();

cmd.CommandType = CommandType.Text;

cmd.CommandText = "SELECT Class FROM tblConnections WHERE TeacherID = @ID ORDER BY Class DESC";

cmd.Parameters.AddWithValue("@ID", info.userID);

return Convert.ToInt32(db.ExeReader(cmd).Rows[0][0]);

}

public DataTable getClasses(int classNum) // Gets all students from a specific class.

{

OleDbCommand cmd = new OleDbCommand();

cmd.CommandType = CommandType.Text;

cmd.CommandText = "SELECT StudentID, Name FROM tblStudents” +

“INNER JOIN tblConnections ON tblStudents.StudentID = tblConnections.StudentID” +

“WHERE Class = @Class ORDER BY StudentID ASC";

cmd.Parameters.AddWithValue("@Class", classNum);

return db.ExeReader(cmd);

}

public string getHash(string hashValue) // Gets the hash value for a password.

{

string hashCode = String.Format("{0:X}", hashValue.GetHashCode());

return hashCode;

}

public string ConvertDataTabletoJSON(DataTable dt) // Converts a given DataTable to JSON format.

{

JavaScriptSerializer serializer = new JavaScriptSerializer(); // Create a new instance of the JavaScriptSerializer class

List<Dictionary<string, object>> listRows = new List<Dictionary<string, object>>(); // Create a list of Dictionaries

Dictionary<string, object> row; // Create an instance of a Dictionary

foreach (DataRow dr in dt.Rows) // For each row in the given DataTable

{

row = new Dictionary<string, object>(); // Set the instance of the dictionary.

foreach (DataColumn col in dt.Columns) // For each column in the DataTable.

{

row.Add(col.ColumnName, dr[col]); // Add the column name and the corresponding data to the row Dictionary.

}

listRows.Add(row); // Add this Dictionary to the list.

}

return serializer.Serialize(listRows); // Serialize the list and return it.

}

}

namespace BEL

{

public class Information

{

#region variables

public string userID { get; set; } // Create variables for the student information.

public string Name { get; set; }

public string Password { get; set; }

public int teacherRef { get; set; }

public string userType { get; set; }

#endregion

}

}

using System.Data;

using System.Data.OleDb; // Include references for System.Data.OleDb namespaces.

namespace DAL

{

public class DbConnection

{

// Add a new SqlConnection using the OleDbConecction class

public OleDbConnection conn = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Jacob Winkworth\Documents\Project\Physics Studio Database.accdb");

#region connection

public OleDbConnection getcon() //Create a function to open the connection

{

conn.Open();

return conn;

}

public int ExeNonQuery(OleDbCommand cmd) // Create an ExecuteNonQuery function for inserts, updates, etc.

{

cmd.Connection = getcon(); // Use the getcon() function to open the SQL connection.

int rowsaffected = -1; // If the ExecuteNonQuery() is unsuccessful and the given values are not in the table then this will remain at -1. In this case, the final messagebox will not display.

try { rowsaffected = cmd.ExecuteNonQuery(); } //Takes the cmd parameter and runs an ExecuteNonQuery.

catch { conn.Close(); return rowsaffected; }

conn.Close(); //Close the connection

return rowsaffected; // Returns the number of rows in which an action has been made.

} // Used later in the program for checking a registration is successful.

public object exeScalar(OleDbCommand cmd)

{

cmd.Connection = getcon(); // Open the connection.

object obj = -1; // Create a new object and set it's default to -1.

obj = cmd.ExecuteScalar(); // This represents the number of rows affected by passing the query.

conn.Close(); // Close the connection.

return obj; // Return the object.

}

public DataTable ExeReader(OleDbCommand cmd) //Create a function which returns a DataTable based upon the parameter cmd. Used when logging in etc.

{

cmd.Connection = getcon(); // Use the getcon() function to open the SQL connection.

OleDbDataReader sdr; // Create a new instance of the OleDbDatReader class.

DataTable dt = new DataTable(); // Create a new DataTable.

sdr = cmd.ExecuteReader();

dt.Load(sdr); // Load the Reader into the DataTable.

conn.Close(); // Close the connection.

return dt; //Return the DataTable.

}

#endregion

}

}