Task 2

Encrypts or decrypts messages by shifting the alphabet by -25 to 25 characters, depending on what is set in the shift by field.
Allows to choose where to save an encrypted message or what file to open and decrypt a message from.
The encryption/decryption is easily achieved by converting a message's characters to their ASCII codes and performing checks on the arrays containing them.

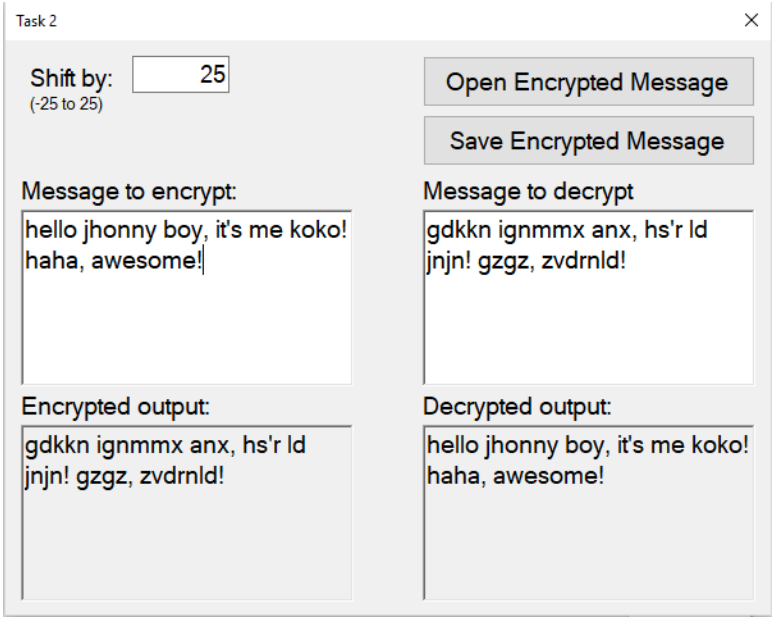
Task 3

Checks if a keyword for today is set in the remote database, if not then a new one must be set, otherwise a login screen appears.
Setting a new keyword is achieved by loading poems from the poems table located on the remote database, then splitting them accordingly in the combobox and listbox fields, from which the user can choose which word to set as daily keyword.
Additional poems can also be added, from the Add Poems form.
After setting the new keyword, every agent on the agents table will be emailed the generated number set and the date it is valid for.
When logging in, keyword and agent initials must match what is stored on the database.
There are also options to delete the keyword of the day or to add a new agent (initials + email required).
Once logged in, messages can be encrypted/decrypted with the daily keyword and each operation is recorded on the remote database.
The combobox allows to load old encryptions/decryptions and to perform new encryptions/decryptions using the old keyword that was loaded from an old message.
There are also options for saving an encrypted message or opening and decrypting one.

Documentation for Task 2

Example:
a) Shifting alphabet by 0 characters, so the encrypted message is the same as input/decrypted:

b) Shifting alphabet by 25 characters:



Logic behind the program:

```
18        globalToolBox global = new globalToolBox();
```

Where globalToolBox is a class located on common.cs, it contains the following variables used by Task 2's Form:

```
public StreamWriter write;
public StreamReader read;
public int max = 122;
public int min = 97;
```

- I chose to store some of the variables and functions which both Task 2 and Task 3 would utilize in a class that could be easily accessed from both forms.

The actual Encryption/Decryption is done here:

```
4 references
public string encryptMessage(string message, bool encryptOrDecrypt = true)
{
    int shiftBy = 0;
    int.TryParse(textBox1.Text, out shiftBy);
    string encrypted = "";
    int charCode = 0;
    char newChar = new char();
    int totalNew = 0;
    foreach (char c in message)
    {
        charCode = (int)c;
        totalNew = charCode;
        if(charCode != 32 && charCode != 39 && charCode != 33 && charCode != 63 && charCode != 46 && charCode != 44)
        {
            totalNew = (encryptOrDecrypt) ? (charCode + shiftBy) : (charCode - shiftBy);
            if (totalNew > global.max)
                totalNew = totalNew - global.max + global.min - 1;

            if (totalNew < global.min)
                totalNew = global.max - global.min + totalNew + 1;
        }
        newChar = (char)totalNew;
        encrypted += newChar.ToString();
    }
    return encrypted;
}
```

The above function is called when:
- Shift By is changed;
- A saved file is opened;
- Upon any input on Encrypt and Decrypt textboxes.

Each rich textbox is "purified" by allowing only lowercase letters and `' ! ? , .` characters. This is done by making use of the following regex, located on common.cs:

```
Regex regex = new Regex("[^a-z '!?.,]");
```

```
1 reference
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
    richTextBox1.Text = global.regex.Replace(richTextBox1.Text.ToLower(), "");
    richTextBox1.Text = richTextBox1.Text.ToLower();
    if (richTextBox1.Text.Length > 0)
    {
        richTextBox2.Text = encryptMessage(richTextBox1.Text);
        richTextBox4.Text = richTextBox2.Text;
        richTextBox1.SelectionStart = richTextBox1.Text.Length;
        button1.Enabled = true;
    }
    else
    {
        richTextBox4.Text = "";
        richTextBox2.Text = "";
        button1.Enabled = false;
    }
}

1 reference
private void richTextBox4_TextChanged(object sender, EventArgs e)
{
    richTextBox4.Text = global.regex.Replace(richTextBox4.Text.ToLower(), "");
    if (richTextBox4.Text.Length > 0)
    {
        richTextBox3.Text = encryptMessage(richTextBox4.Text, false);
        richTextBox1.Text = richTextBox3.Text;
        richTextBox4.SelectionStart = richTextBox4.Text.Length;
    }
    else
    {
        richTextBox1.Text = "";
        richTextBox3.Text = "";
    }
}
```

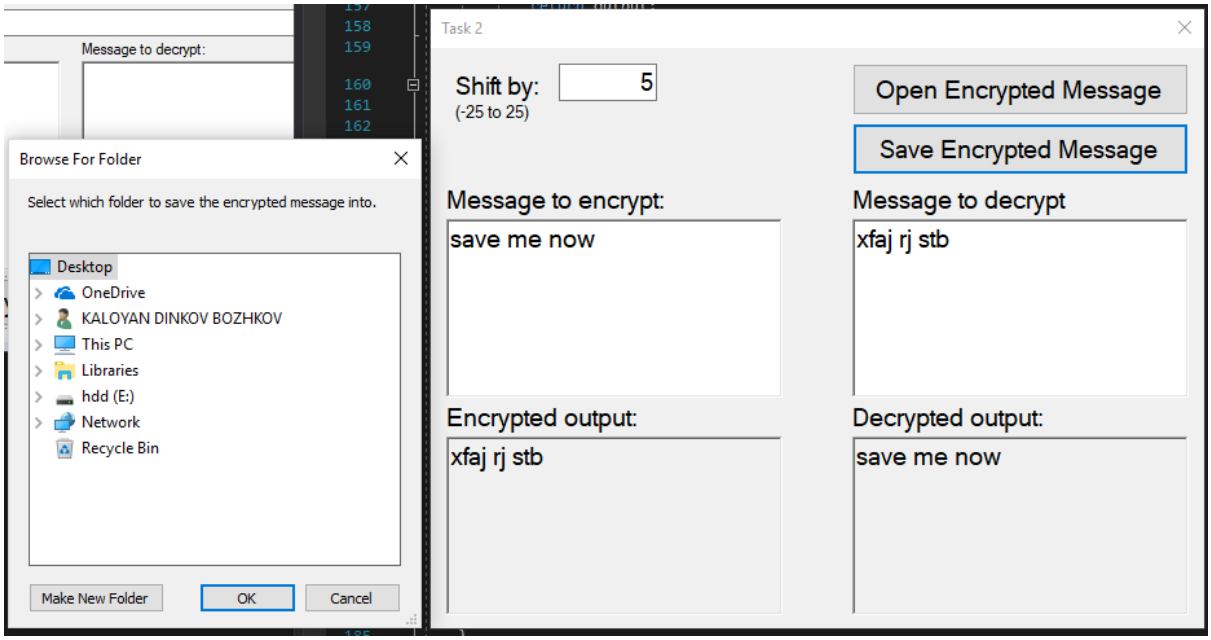
The ShiftBy textbox makes use of the following functions to make sure its input only allows for -25 to 25:

```
1 reference
private void textBox1_TextChanged(object sender, EventArgs e)
{
    textBox1.Text = checkInput(textBox1.Text, true);
    if (textBox1.Text.Length > 0)
    {
        textBox1.SelectionStart = textBox1.Text.Length;

        if(richTextBox1.Text.Length > 0)
        {
            richTextBox2.Text = encryptMessage(richTextBox1.Text);
            richTextBox4.Text = richTextBox2.Text;
            richTextBox3.Text = encryptMessage(richTextBox4.Text, false);
        }
    }
    else
    {
        richTextBox2.Text = richTextBox1.Text;
        richTextBox4.Text = richTextBox3.Text;
    }
}
```

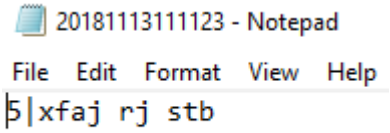
```
1 reference
public static string checkInput(string x, bool canBeNegative)
{
    string output = "";
    char y;
    for (var i = 0; i <= x.Length - 1; i++)
    {
        y = x[i];
        if (IsNumeric(y.ToString()))
            output += y.ToString();
        else if (canBeNegative && y.ToString() == "-" && i == 0)
            output += "-";
    }

    int testOutput = 0;
    if (int.TryParse(output, out testOutput))
    {
        if (testOutput > 25)
            output = "25";
        else if (testOutput < -25)
            output = "-25";
    };
    return output;
}

1 reference
public static bool IsNumeric(string str)
{
    double myNum = 0;
    if (Double.TryParse(str, out myNum))
        return true;

    return false;
}
```

The only difference between the Encrypt and Decrypt process is that when decrypting I am passing **false** as second parameter (named *encryptOrDecrypt*) that is used in the IF statement inside the main loop. When encryptOrDecrypt is false, the ShiftBy number is subtracted from the char code of the letter in the current loop cycle, instead of being added to it.

Save:



Saves the ShiftBy number & Encrypted string to txt file as follows:



Code for save:

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    global.writeToSeelectedFolder(textBox1.Text + "|" + richTextBox2.Text);
}
```

Which references writeToSelectedFolder() of globalToolBox class on common.cs:

```
2 references
public bool writeToSeelectedFolder(string output, bool askForFileName = false)
{
    string path = "";
    DialogResult result;
    DateTime time = DateTime.Now;
    string name = time.ToString("yyyyMMddhhmmss");
    if (!askForFileName)
    {
        FolderBrowserDialog FolderBrowser = new FolderBrowserDialog();
        FolderBrowser.Description = "Select which folder to save the encrypted message into.";
        result = FolderBrowser.ShowDialog();
        path = FolderBrowser.SelectedPath + "/" + name + ".txt";
    }
    else
    {
        SaveFileDialog saveFile = new SaveFileDialog();
        saveFile.Filter = "txt files (*.txt)|*.txt";
        saveFile.FilterIndex = 1;
        saveFile.RestoreDirectory = true;
        saveFile.FileName = name + ".txt";
        result = saveFile.ShowDialog();
        path = Path.GetFullPath(saveFile.FileName);
    }

    if (result == DialogResult.OK)
    {
        string filePathFileName = path;
        try
        {
            write = new StreamWriter(filePathFileName);
            write.WriteLine(output);
            write.Close();
            MessageBox.Show("File saved successfully!");
            return true;
        }
        catch (Exception ex)
        {
            MessageBox.Show("There was an issue when trying to write to " + filePathFileName + "." + ex.ToString());
        }
    }
    return false;
}
```

Opening a saved file:

```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    string[] content = global.loadEncryptedMessageFromFolder();
    if(content != null)
    {
        richTextBox1.Text = "";
        textBox1.Text = content[0];
        richTextBox4.Text = content[1];
    }
}
```

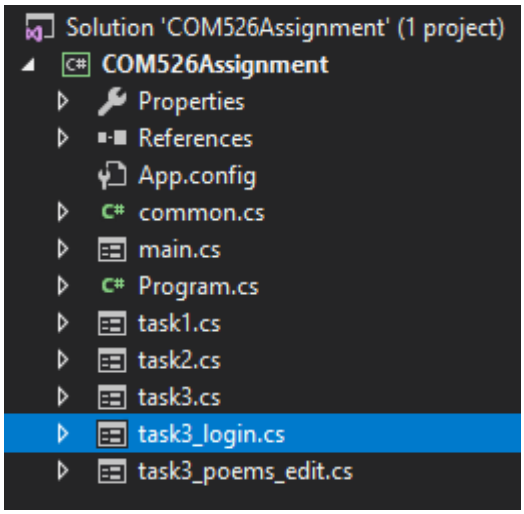
After fetching the encrypted message and ShiftBy number from the saved file, I simply set them into the relative input fields, which triggers the encrypt/decrypt function because of the *on input* event trigger.

The above code references loadEncryptedMessageFromFolder() of globalToolBox class on common.cs:

```
2 references
public string[] loadEncryptedMessageFromFolder()
{
    OpenFileDialog fileBrowserDialog = new OpenFileDialog();
    fileBrowserDialog.Title = "Select which file to load encrypted message from.";
    fileBrowserDialog.Filter = "TXT files|*.txt";
    fileBrowserDialog.InitialDirectory = @"C:\";
    if (fileBrowserDialog.ShowDialog() == DialogResult.OK)
    {
        string filePathFileName = fileBrowserDialog.FileName; // file will always exist when selected from dialog
        try
        {
            read = new StreamReader(filePathFileName);
            string line = read.ReadLine(); // I save as shitfBy|text, so always only 1 line
            if(line.Length > 0 && line.Contains("|"))
            {
                string[] content = line.Split('|');
                read.Close();
                MessageBox.Show("Encrypted message from file '" + filePathFileName + "' was successfully loaded!");
                return content;
            }
            else
            {
                MessageBox.Show("The file you are tyring to open seems unusual!");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("There was an issue when trying to open " + filePathFileName + "." + ex.ToString());
        }
    }
    return null;
}
```

I return an array of strings, where position [0] is the ShiftBy number, and [1] is the encrypted message.
(This, together with everything else on common.cs, is reused on task 3);

Documentation for Task 3



Unlike Task2, Task 3 is made of 3 forms:

- Task3_login, which is the main form that loads.
- Task3_poems, which is a form that is displayed when the Add New Poem button on Task3_login is clicked.
- Task3, which contains the following functionalities: Encrypt/Decrypt, Open, Save and Load/Filter old messages from database.

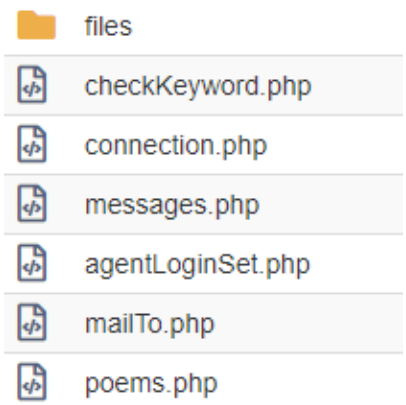
For Task 3 the program is communicating with my database, which is stored on a shared hosting server.
Each operation (encrypt, decrypt, register agent, delete keyword, login, set keyword, send email) connects to different PHP pages which query the database and return, via REST calls, the output to C#.
globalToolBox contains the single function, named connectToDatabase, which handles all the calls to the different PHP pages hosted on my server:

```
public string connectToDatabase(string pageName = "", string query = "", string operationType = "read")
{
    var webClient = new WebClient();
    webClient.Headers.Add("Content-Type", "application/x-www-form-urlencoded");
    webClient.Headers.Add("Accept", "text/xml");
    var parameters = new System.Collections.Specialized.NameValueCollection();
    parameters.Add("query", query);
    parameters.Add("operationType", operationType);

    try{
        byte[] bret = webClient.UploadValues("http://kaloyanbozhkov.com/glyndwr/" + pageName + ".php", "POST", parameters);
        return Encoding.UTF8.GetString(bret);
    }
    catch(Exception ex)
    {
        MessageBox.Show("Oops! There seems to be a problem connecting to the database :(\n\nPlease check your internet connection and try again later.\n
        return null;
    }
}
```

As parameters I pass the PHP's **page name** and the **query** string which is a normal string composed of concatenated strings separated by a special character on which I split into an array in PHP and generate the needed SQL queries for the desired output. The **operationType** parameter allows for fast IF statement checks on the same PHP page.

Explanation of The Server Side

Here are all the PHP files which create and execute the queries server side:



- **connection.php:** sets up the SQL database connection, this file is used in every other PHP file.
- **checkKeyword.php:**
  - If operationType is read: checks the existence of a keyword for today's date;
  - If oprationType is write:
    - sets the new keyword for the day;
  - If operationType is delete:
    - deletes the keyword of the day.

- **fetchMessages.php**: if operationType is read: selects all contents form the messages table and returns the output, depending on the filters used (order by date, select for specific keyword and so on). If operationType is write: inserts a new row into the messages table, with the specified values contained in the query string that was sent as a parameter.
- **agentLoginSet.php**: if operationType is read: checks if the agent initials and daily keyword used for logging in match with the database records, if so then it returns the generated number set representing the daily keyword used to login. If operationType is write: inserts a new agent in the users table (passing only initials and a valid unique email);
- **mailTo.php**: queries the agents table for their email address and initials, sends back to C# a string that contains initials?email+initials?email.. +gmailPassword+emailHTMLContent
  Also sends me an E-mail containing the IP Address from which the C# program executed the REST call. (I am not sending the agents emails from PHP, as I am doing with the IP tracking email, because of two reasons: 1) I wanted to try emailing through C# 2) The shared hosting server I have allows only for the main domain to send emails, and not sub domains (main domain is svetlaestetica.com, subdomain is kaloyanbozhkov.com [where the /glyndwr/ folder containing all PHP files for rest calls is], only the first domain has a mail service).
- **poems.php**: if operationType is read: it outputs all poems in the poems table, in XML format. If operationType is write: it inserts a new row into the poems table.

For example, the checkKeyword.php file is structured like this:

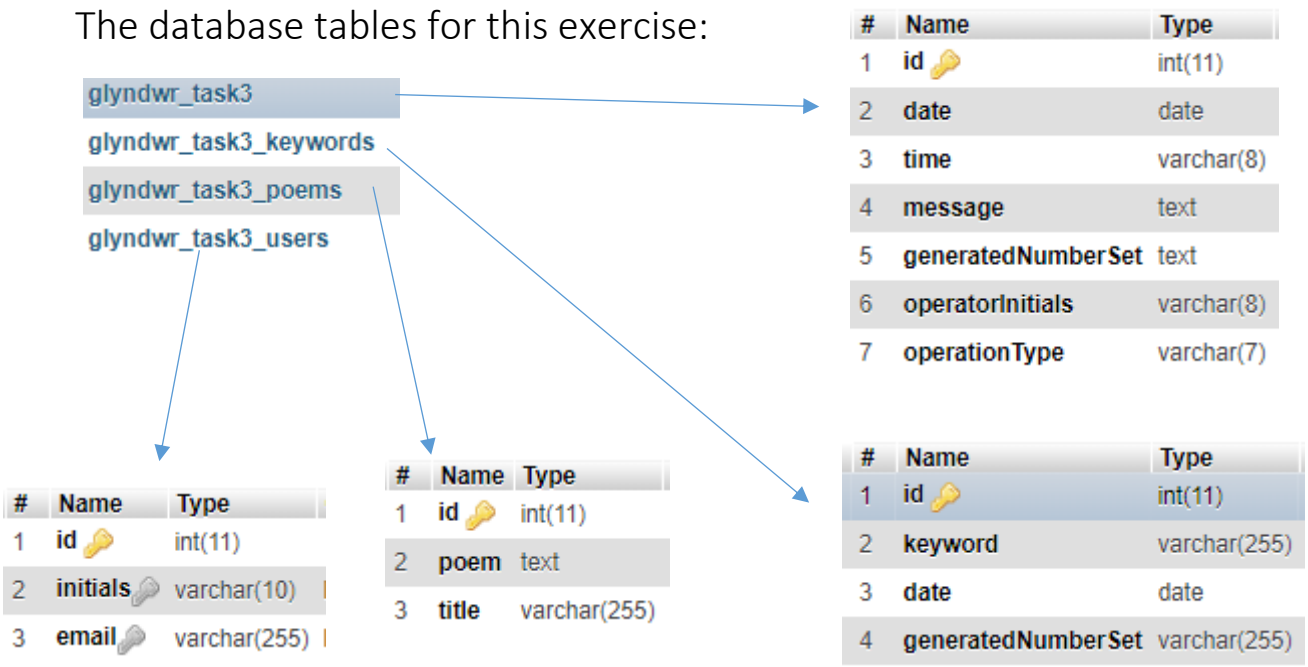
Editing: /ndwr/checkKeyword.php Encoding: utf-8

Keyboard shortcuts

```php
<?php
require("connection.php");
$query = $_POST["query"];
$content = explode("|", $query);
$operationType = $_POST["operationType"];
if($operationType == "read"){
    $query = "SELECT id FROM `glyndwr_task3_keywords` WHERE date = '
    $result = mysqli_query($link, $query);
    $number = mysqli_num_rows($result);
    if ($number != 0) {
        echo("exists");
    }else{
        echo("unknown");
    }
}else{
    if($operationType == "write"){
        $query = "INSERT INTO `glyndwr_task3_keywords` VALUES('', ''
    }elseif($operationType == "delete"){
        $query = "DELETE FROM `glyndwr_task3_keywords` WHERE date =
    }
    if(mysqli_query($link, $query)){
        echo("success");
    }else{
        echo("issue");
    }
}
?>
```

operationType allows for multiple operations on the same PHP file, whilst still being easy to read (Instead of concatenating the operationType to the query string).

The database tables for this exercise:



Step by Step Process & Explanation

Having explained how the most important part of the program works (the backend), here are the processes that fire from start to finish:

1) On start:

```
private void task3_login_Load(object sender, EventArgs e)
{
    if (!updatePoemList())
    {
        this.Close(); //no internet connection/database inaccessible
    }
    else
    {
        string query = DateTime.Now.ToString("yyyy-MM-dd");
        if (global.connectToDatabase("checkKeyword", query) == "exists")
        {
            //keyword exists so proceed to login screen
            panel1.Visible = true;
            resetWindow(180);
        }
        else
        {
            //keyword needs to be created since it does not exist
            MessageBox.Show("Fetching poem list from database, please wait a while!");
            date.Text = "No keyword has been set for today's date (" + DateTime.Now.ToString("yyyy-MM-dd") + ")";
            panel2.Visible = true;
            resetWindow(545);
        }
    }
}
```

task3_login has two panels, first one containing the *login form* and the second containing the *poems selection menu*. If a keyword of the day exists, then the panel containing the login form is shown and the other one is hidden, and vice versa.

I could have separated the two panels into different forms like I did with *task3_poems_edit*, however playing around with form size and visibility of panels seemed like a more interesting approach.

The first thing that happens is updatePoemList() being executed:

a) updatePoemList() is executed:

```
public bool updatePoemList()
{
    string poemListStirng = global.connectToDatabase("poems");
    if(poemListStirng != null)
    {
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(poemListStirng);
        XmlNodeList elem = doc.GetElementsByTagName("poem");
        comboBox1.Items.Clear();
        for (int k = 0; k < elem.Count; k++)
        {
            comboBox1.Items.Add(elem.Item(k).SelectSingleNode("id").InnerText + " - " + elem.Item(k).SelectSingleNode("title").InnerText);
            global.poemsList.Add(elem.Item(k).SelectSingleNode("body").InnerText);
        }
        return true;
    }
    return false;
}
```

Which fetches the XML formatted output from poems.php and loops through each poem element and inserts the inner text of its ID and Title nodes in comboBox1, whereas the poem contents are loaded into the poemsList variable located on common.cs.

```
public List<string> poemsList = new List<string>();
```

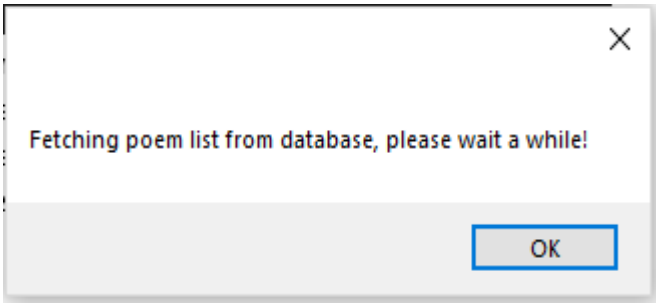
Since the poems' body/content are added into the list of strings in the same order as the poems' ID and Title are added into the combobox1, it is easy to refer to a poem's body/content stored in the list of strings with the selectedIndex value of the combobox.

The below image shows how I generate the XML “file” that is passed on to C#.

```php
<?php
require("connection.php");
header('Content-Type: text/xml');
$type = $_POST["operationType"];
if($type == "read"){
    $div = '<?xml version="1.0" encoding="UTF-8"?><poems>';
    $sql = "SELECT * FROM `glyndwr_task3_poems` WHERE id > 0 ORDER BY id ASC";
    $result = mysqli_query($link, $sql);
    $number = mysqli_num_rows($result);
    if ($number != 0) {
        while ($row=mysqli_fetch_array($result)) {
            $div .= "
            <poem id='".$row["id"]."'>
            <id>".$row["id"]."</id>
            <title>".$row["title"]."</title>
            <body>".$row["poem"]."</body>
            </poem>";
        }
        echo($div."</poems>");
    }else{
        echo("No poems added yet.");
    }
}elseif($type == "write"){
    $query = $_POST["query"];
    $i = explode("|", $query);
    $sql = "INSERT INTO `glyndwr_task3_poems`(`id`, `poem`, `title`) VALUES
    ('','".mysqli_real_escape_string($link, $i[1])."','".
    mysqli_real_escape_string($link, $i[0])."')";
    if (mysqli_query($link, $sql)) {
     echo("done");
    }else{
     echo("issue");
    }
}
?>
```

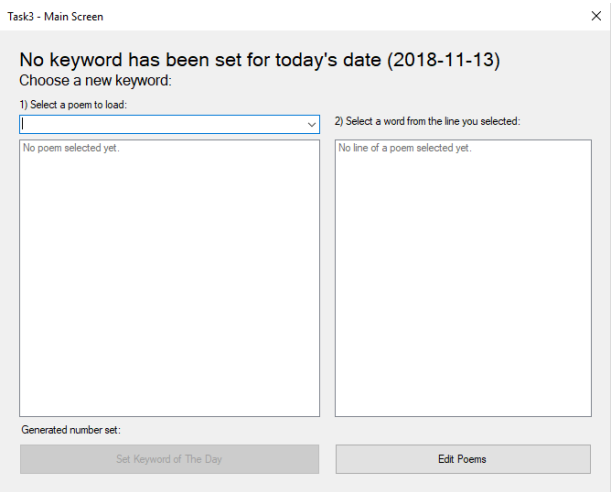
I decided to use XML for this part instead of regular string concatenation, mainly because I wanted to experiment with XML, however in this case using XML is also better since, not having to split on a single character, the poems can contain anything (exclamation marks, question marks, dots, points and so on) and, had I not used XML, I would have had to consider replacing such characters to make sure splitting the string into an array would function as intended.

If the connectToDatabase function manages to connect to the remote database then the return value is true, and the following message shows:

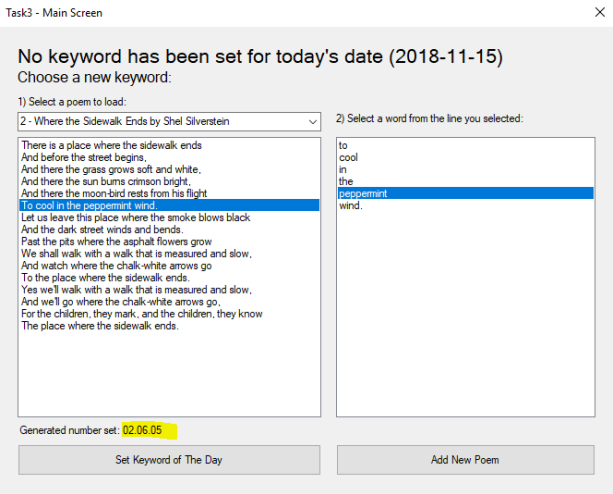


If the connection is impossible, due to internet connection issues or other problems, then a message regarding that issue is shown and the program closes. [I only check and close the program once, assuming that if an agent launches the program and has connected to the database successfully the first time, then he will have connectivity throughout the entire run time.

b) **the panel containing the poems selection menu is shown:**



- The default view if no poem is selected.



Once a poem is selected Generated Number Set becomes n.., once a row is selected the generated number set becomes: n.n., and once a word is selected it becomes: n.n.n

```
14        string[] encodedDate = { " ", " ", " " };
```

The Generated Number Set is saved in the encodedDate variable. (in retrospect, a misleading variable name!)

The encodedDate (which is the generated number set) is gradually generated when the user selects a poem, a row and finally a word. Once a word is selected the Set Keyword of The Day button becomes enabled.

When a poem is selected:

```
1 reference
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    encodedDate[0] = " ";
    encodedDate[1] = " ";
    encodedDate[2] = " ";
    listBox2.Items.Clear();
    listBox2.Items.Add("No line of a poem selected yet.");
    listBox1.Items.Clear();
    if (comboBox1.SelectedIndex > -1)
    {

        listBox1.Enabled = true;
        string[] loadPoemLines = global.poemsList[comboBox1.SelectedIndex].Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);
        for (int k = 0; k < loadPoemLines.Length; k++)
            listBox1.Items.Add(loadPoemLines[k]);

        encodedDate[0] = ((int)(comboBox1.SelectedIndex) + 1).ToString();
        setZeroInEncodedDate(0);
    }
    else
    {
        listBox2.Enabled = false;
        listBox2.Enabled = false;
    }
}
```

When a row of a poem is selected:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{

    if (listBox1.SelectedIndex > -1)
    {
        if (listBox1.SelectedIndex == 0 && listBox1.Items[0].ToString() == "No poem selected yet.")
        {
            listBox1.Enabled = false;
        }
        else
        {
            listBox2.Items.Clear();
            listBox1.Enabled = true;
            listBox2.Enabled = true;
            string[] words = global.regexA.Replace(listBox1.Items[listBox1.SelectedIndex].ToString().ToLower(), "").Split(' ');
            for (int k = 0; k < words.Length; k++)
            {
                listBox2.Items.Add(words[k]);
            }
            encodedDate[1] = ((int)(listBox1.SelectedIndex) + 1).ToString();
            setZeroInEncodedDate(1);
        }
    }
    else
    {
        listBox1.Enabled = false;
        listBox2.Enabled = false;
        listBox2.Items.Clear();
        listBox2.Items.Add("No line of a poem selected yet.");
    }
}
```

Where regexA is located on Common.cs:

```
public Regex regexA = new Regex("[^a-z ]");
```

When a word from a row of a poem is selected:

```
1 reference
private void listBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox2.Enabled == true && listBox2.SelectedIndex > -1)
    {
        button1.Enabled = true;
        encodedDate[2] = ((int)(listBox2.SelectedIndex) + 1).ToString();
        setZeroInEncodedDate(2);
    }
}
```

setZeroInEncodedDate(), shown below, makes sure that poem, row and word counters start with a 0 in case they are less than 10, to comply with the exercise's format. (e.g. "01" instead of "1").
The function also updates the label text.

```
3 references
public void setZeroInEncodedDate(int n)
{
    if (encodedDate[n].Length == 1)
    {
        encodedDate[n] = "0" + encodedDate[n];
    }
    generatedNumberSet.Text = "Generated number set: " + encodedDate[0] + "." + encodedDate[1] + "." + encodedDate[2];
}
```

When the Set Keyword of The Day button is pressed the following code is executed:

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    string generatedNumberSetValue = encodedDate[0] + "." + encodedDate[1] + "." + encodedDate[2];
    string date = DateTime.Now.ToString("yyyy-MM-dd");
    string query = listBox2.Items[listBox2.SelectedIndex] +"|"+ date + "|"+ generatedNumberSetValue;
    string result = global.connectToDatabase("checkKeyword", query, "write");
    if(result == "success")
    {
        MessageBox.Show("Keyword of the day has successfully been set, please wait while sending emails to agents.");
        string response = global.connectToDatabase("mailTo");
        if (response != "no")
        {
            string subject = "Get your freshly generated number set!";
            string body = "";
            string[] agentInfo;
            string[] agents = response.Split('+');
            string htmlMessage = agents[agents.Length - 1];
            for (int k = 0; k < agents.Length - 2; k++)
            {
                agentInfo = agents[k].Split('?');
                body = htmlMessage.Replace("XYZJ", global.formatInitials(agentInfo[0])).Replace("GNSJZW", generatedNumberSetValue).Replace("fkdate", date);
                sendEmail(body, subject, agentInfo[1], agents[agents.Length - 2]);
            }
            MessageBox.Show("Emails have been sent containing the Generated Number Set and expiration date.");
        }
        panel2.Visible = false;
        panel1.Visible = true;
        resetWindow(180);
    }
    else
    {
        MessageBox.Show("There has been an issue setting the keyword of the day.");
    }
}
```

In detail:

```
string generatedNumberSetValue = encodedDate[0] + "." + encodedDate[1] + "." + encodedDate[2];
string date = DateTime.Now.ToString("yyyy-MM-dd");
string query = listBox2.Items[listBox2.SelectedIndex] +"|"+ date + "|"+ generatedNumberSetValue;
string result = global.connectToDatabase("checkKeyword", query, "write");
if(result == "success")
```

Inserts into the keywords table a row containing the keyword (which is in string format, used only for the login authentication process), the date the keyword is valid for and the generated number set which refers to a word in a row in a poem.

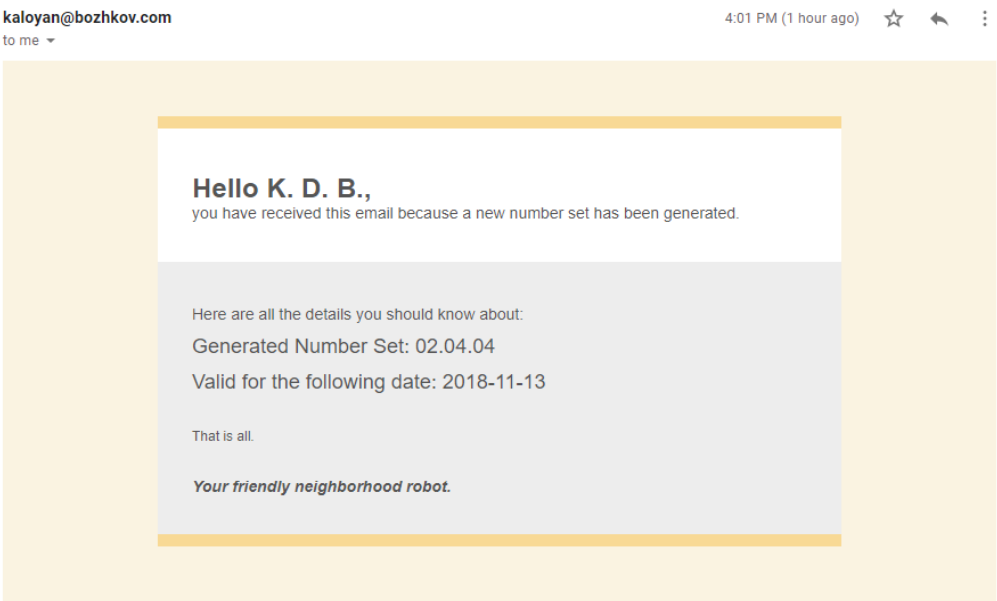
Since “write”, as operationType parameter, is passed to connectToDatabase, the INSERT query is executed.

If the query executed successfully, the output is “success”.

```
<?php
require("connection.php");
$query = $_POST["query"];
$content = explode("|", $query);
$operationType = $_POST["operationType"];
if($operationType == "read"){
    $query = "SELECT id FROM `glyndwr_task3_ke
    $result = mysqli_query($link, $query);
    $number = mysqli_num_rows($result);
    if ($number != 0) {
        echo("exists");
    }else{
        echo("unknown");
    }
}else{
    if($operationType == "write"){
        $query = "INSERT INTO `glyndwr_task3_k
    }elseif($operationType == "delete"){
        $query = "DELETE FROM `glyndwr_task3_k
    }
    if(mysqli_query($link, $query)){
        echo("success");
    }else{
        echo("issue");
    }
}
?>
```

```
if(result == "success")
{
    MessageBox.Show("Keyword of the day has successfully been set, please wait while sending emails to agents.");
    string response = global.connectToDatabase("mailTo");
    if (response != "no")
    {
```

After setting the new daily keyword, every agent in the agents table is going to be emailed the following email:



This is done by first fetching the return value of mailTo.php, which (if successful) is a string that contains initials?email+initials?email.. +gmailPassword+emailHTMLContent. This string is in the response string var.

```
if (response != "no")
{
    string subject = "Get your freshly generated number set!";
    string body = "";
    string[] agentInfo;
    string[] agents = response.Split('+');
    string htmlMessage = agents[agents.Length - 1];
    for (int k = 0; k < agents.Length - 2; k++)
    {
        agentInfo = agents[k].Split('?');
        body = htmlMessage.Replace("XYZJ", global.formatInitials(agentInfo[0])).Replace("GNSJZW", generatedNumberSetValue).Replace("fkdate", date);
        sendEmail(body, subject, agentInfo[1], agents[agents.Length - 2]);
    }
    MessageBox.Show("Emails have been sent containing the Generated Number Set and expiration date.");
}
panel2.Visible = false;
panel1.Visible = true;
resetWindow(180);
```

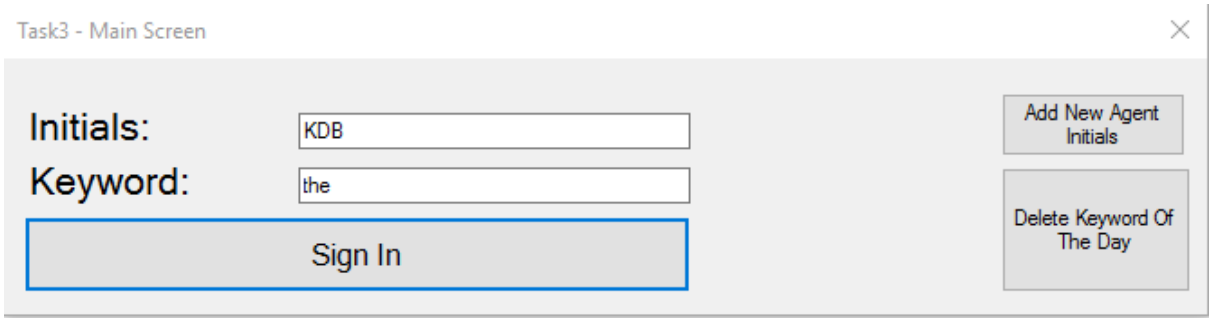
After splitting the response string and getting the initials of each agents and their relative emails, the last thing to do before sending the email is replacing those characters in the body of the email's HTML where the initials, generated number set and the date it is valid for should be placed.

The emails themselves are sent through the sendEmail function, which is shown below:

```
public void sendEmail(string emailBody, string emailSubject, string emailTo, string easyToFindOutSecretButIAmTrackingYourIpIfYouTry)
{//using net.mail
    try
    {
        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.From = new MailAddress("kaloyan@bozhkov.com");
        mail.To.Add(emailTo);
        mail.Subject = emailSubject;
        mail.IsBodyHtml = true;
        mail.Body = emailBody;
        SmtpServer.Port = 587;
        SmtpServer.Credentials = new System.Net.NetworkCredential("kaloyan@bozhkov.com", easyToFindOutSecretButIAmTrackingYourIpIfYouTry);
        SmtpServer.EnableSsl = true;
        SmtpServer.Send(mail);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

Again, every time this function is called, I am being notified that my gmail password was passed to C# through the mailTo.php file.

After sending the emails, a message box shows up saying that everything is complete, then the form resizes and the other panel containing the login layout is displayed (as seen below).

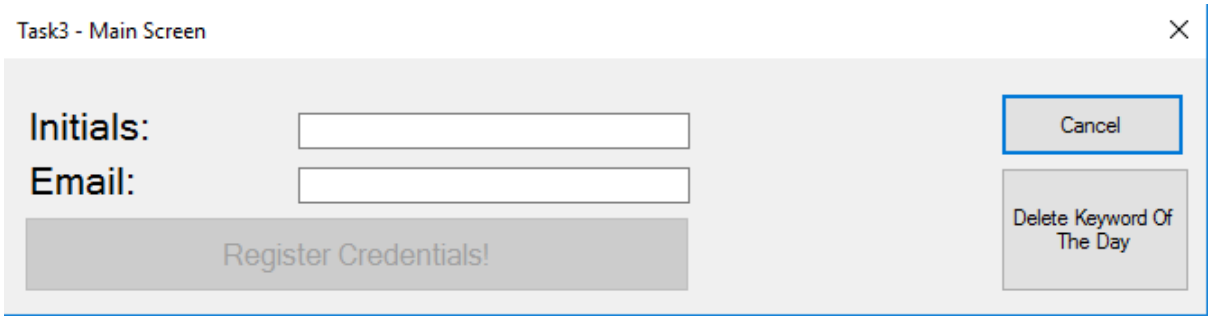


2) Actions on login screen:

- Delete Keyword Of The Day: calls checkKeywords.php, passing delete as operationType parameter, and executing a delete query for today's date.

- Add New Agent: changes button and label text, and sets registrationProcess to true:

```
private void button5_Click(object sender, EventArgs e)
{
    if (!registrationProcess)
    {
        registrationProcess = true;
        button4.Text = "Register Credentials!";
        button5.Text = "Cancel";
        label5.Text = "Email:";
    }
    else
    {
        resetSettings();
    }
    enableBtn();
}
```



When clicking the register credentials button, the second part of the IF statement shown below is executed:

```
private void button4_Click(object sender, EventArgs e)
{
    string initials = textBox1.Text.Replace(".", "").Replace(" ", "").Replace(",", "").ToLower();
    string keyword = textBox2.Text.ToLower();
    if (!registrationProcess)
    {
        string query = initials + ";" + keyword + ";" + DateTime.Now.ToString("yyyy-MM-dd");
        string output = global.connectToDatabase("agentLoginSet", query);
        if (output != "none")
        {
            MessageBox.Show("Logged in!");
            task3 x = new task3();
            x.creds = new credentials();
            x.creds.password = keyword;
            x.creds.initials = initials;
            x.creds.generatedNumberSet = output;
            x.global = new globalToolBox();
            x.global.poemsList = global.poemsList;
            this.Hide();
            x.ShowDialog();
            this.Close();
        }
        else
        {
            MessageBox.Show("Wrong Initials Or Keyword!");
        }
    }
    else
    {
        if(global.connectToDatabase("agentLoginSet", initials+";"+keyword, "write") == "done"){
            MessageBox.Show("Agent was successfully registered!");
            resetSettings();
        }
        else
        {
            MessageBox.Show("Agent could not be registered at the moment, make sure the email is not already in use.");
        }
    }
}
```

If registrationProcess is true then agentLoginSet.php is called and the required parameters are passed in order to build the insert query (Image of the server-side part is shown further below).

The email address is checked with the validatEmail function.

```
public void enableBtn()
{
    if (textBox1.Text.Length > 0 && textBox2.Text.Length > 0 && !registrationProcess)
        button4.Enabled = true;
    else if(!registrationProcess)
        button4.Enabled = false;
    else if (registrationProcess && textBox1.Text.Length > 0 && textBox2.Text.Length > 0 && validateEmail(textBox2.Text))
        button4.Enabled = true;
    else
        button4.Enabled = false;
}
```

```
public bool validateEmail(string email)

{
            if (email.Length > 0 && email.Contains("@") && email.Split('@')[0].Length
> 0 && email.Split('@')[1].Length > 0 && email.Split('@')[1].Contains(".") &&
email.Split('@')[1].Split('.')[0].Length > 0 &&
email.Split('@')[1].Split('.')[1].Length > 0)
                return true;

            return false;
}
```

The enableButton() function is called by both textboxes whenever there is any text change.

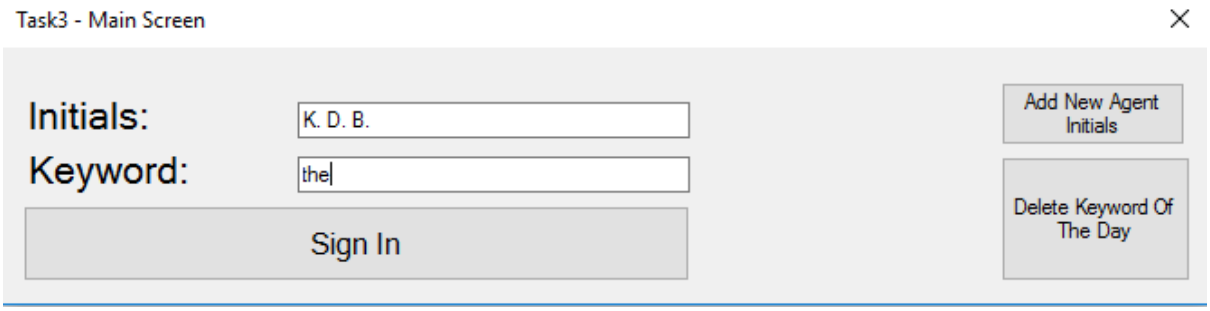
```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    enableBtn();
}

private void textBox2_TextChanged(object sender, EventArgs e)
{
    enableBtn();
}
```

The Add New Agent button also calls said function (as well as resetSettings())

```
private void button5_Click(object sender, EventArgs e)
{
    if (!registrationProcess)
    {
        registrationProcess = true;
        button4.Text = "Register Credentials!";
        button5.Text = "Cancel";
        label5.Text = "Email:";
    }
    else
    {
        resetSettings();
    }
    enableBtn();
}

public void resetSettings()
{
    registrationProcess = false;
    button4.Text = "Sign In";
    button5.Text = "Add New Agent Initials";
    label5.Text = "Keyword:";
    textBox2.Text = "";
}
```

This makes it so that you can toggle between adding a new user and logging in, from the same button.

- Sign in button



When the Sign In button is pressed, the following code is executed:

```
248     private void button4_Click(object sender, EventArgs e)
249     {
250         string initials = textBox1.Text.Replace(".", "").Replace(" ", "").Replace(",", "").ToLower();
251         string keyword = textBox2.Text.ToLower();
252         if (!registrationProcess)
253         {
254             string query = initials + ";" + keyword + ";" + DateTime.Now.ToString("yyyy-MM-dd");
255             string output = global.connectToDatabase("agentLoginSet", query);
256             if (output != "none")
257             {
258                 MessageBox.Show("Logged in!");
259                 task3 x = new task3();
260                 x.creds = new credentials();
261                 x.creds.password = keyword;
262                 x.creds.initials = initials;
263                 x.creds.generatedNumberSet = output;
264                 x.global = new globalToolBox();
265                 x.global.poemsList = global.poemsList;
266                 this.Hide();
267                 x.ShowDialog();
268                 this.Close();
269             }
270             else
271             {
272                 MessageBox.Show("Wrong Initials Or Keyword!");
273             }
274         }
275         else
276         {
277             if(global.connectToDatabase("agentLoginSet", initials+";"+keyword, "write") == "done"){
278                 MessageBox.Show("Agent was successfully registered!");
279                 resetSettings();
280             }
281             else
282             {
283                 MessageBox.Show("Agent could not be registered at the moment, make sure the email is not already in use.");
284             }
285         }
286     }
```

First the query string is built with the initials and keyword inserted, as well as the current date. Following this agentLoginSet.php is called, with read (operationType) and the previously created query string as parameters.

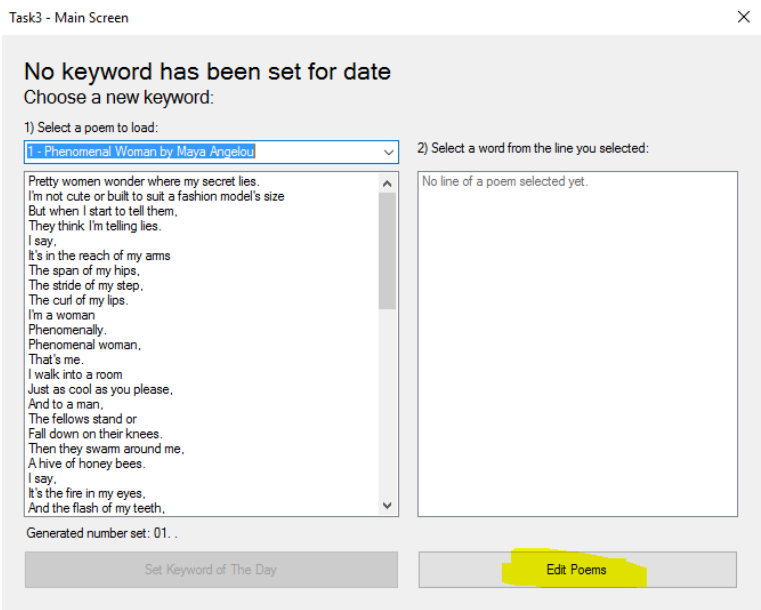
The agentLoginSet.php page's can be seen below:

```php
<?php
require("connection.php");
$query = $_POST["query"];
$i = explode(";", $query);
$operationType = $_POST['operationType'];
if($operationType == "read"){
    $sql = "(SELECT id FROM `glyndwr_task3_users` WHERE initials = '".$i[0]."') UNION (SELECT generatedNumberSet as 'id' FROM
    `glyndwr_task3_keywords` WHERE keyword = '".$i[1]."' and date='".$i[2]."')";
    $result = mysqli_query($link, $sql);
    $number = mysqli_num_rows($result);
    if ($number >= 2) {
            $code = "";
            while ($row=mysqli_fetch_array($result)) {
                $code = $row["id"]; //loops twice on second time sets code to generatedNumberSet
            }
        echo($code);
    }else{
        echo("none");
    }
}else{
    $sql = "INSERT INTO `glyndwr_task3_users`(`id`, `initials`, `email`) VALUES ('','".mysqli_real_escape_string($link,
    $i[0])."','".mysqli_real_escape_string($link, $i[1])."')";
    if (mysqli_query($link, $sql)) {
        echo("done");
    }else{
        echo("issue");
    }
}
?>
```

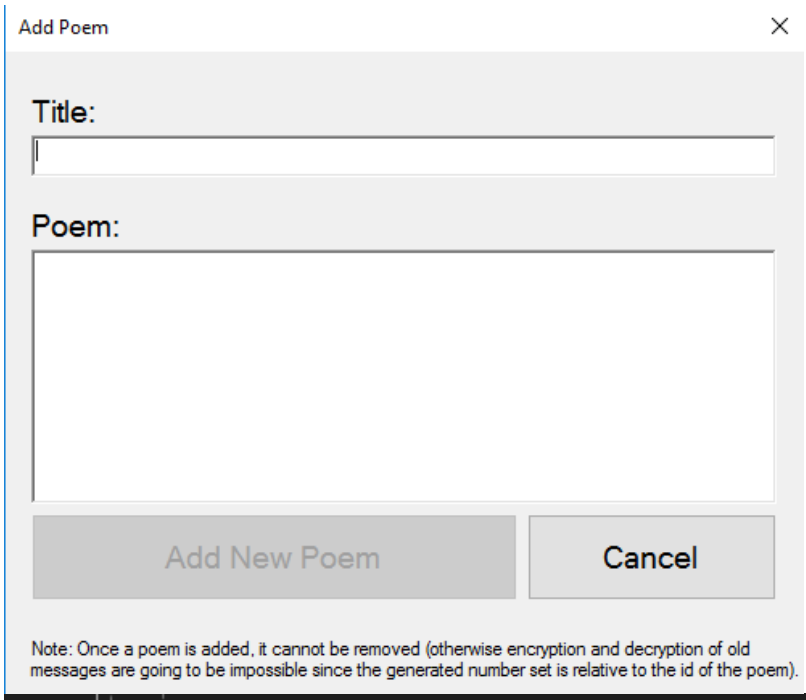
Since the query first selects the ID column from the users (agents) table, and afterwards the ID (which is actually the generatedNumberSet column 'as "id"') from the keywords table, the output will always have as first row the ID from the users table (which was selected with the initials passed in the query string) and the second row will be the generatedNumberSet (which was selected with today's date and keyword.. both passed to PHP through the query string). If there are less than 2 rows then the login was a failure! (initials on users table and date on keywords table are both unique!)

- Add New Poem Button

Before continuing to the main form, there is also the task3_poems_edit form, which is opened through:



And leads to:



Initially I wanted to be able to edit poems as well as removing them, however that would have messed up the generated number set encode/decode logic, since the keyword used for encrypting/decrypting is always taken from the poems.
Because of this, it is only possible to add new poems. On click of Add New Poem, poems.php is called with the required parameters: poem title and poem body/content.

```
private void button1_Click(object sender, EventArgs e)
{
    string poem = richTextBox1.Text + "|" + richTextBox2.Text;
    if (global.connectToDatabase("poems", poem, "write") == "done")
    {
        MessageBox.Show("Poem was added successfully.");
        successfull = true;
        this.Hide();
    }
    else {
        MessageBox.Show("There was an issue adding the poem.");
    }
}
```

After validating the login, Task 3's main form is opened:

```
MessageBox.Show("Logged in!");
task3 x = new task3();
x.creds = new credentials();
x.creds.password = keyword;
x.creds.initials = initials;
x.creds.generatedNumberSet = output;
x.global = new globalToolBox();
x.global.poemsList = global.poemsList;
this.Hide();
x.ShowDialog();
this.Close();
```

Common.cs contains this public class:

```
public class credentials
{
    public string password;
    public string initials;
    public string generatedNumberSet;
}
```

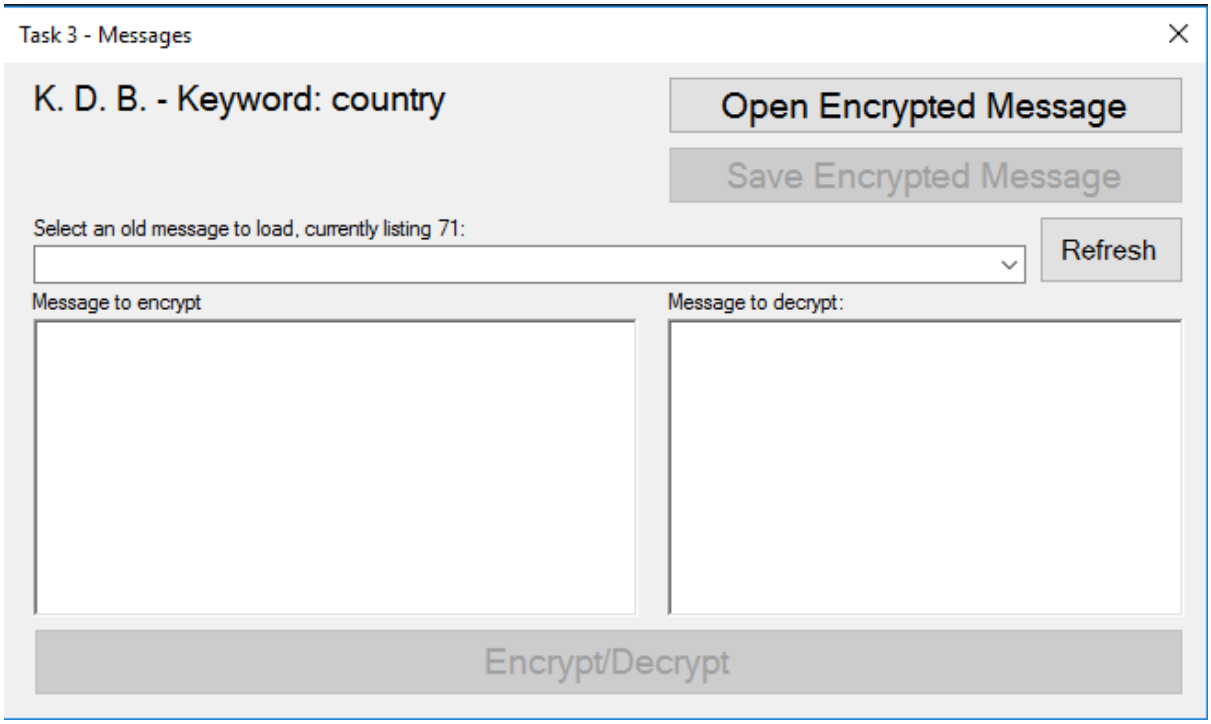
as well as this variable:

```
public List<string> poemsList = new List<string>();
```

Before opening Task 3's main form, I am passing every variable that is needed for the new form to function: the agent's relative credentials, the generated number set and the poemsList.

3) On Main Program Load

After signing in the task3 form is opened:



On form load the following code is executed:

```
55    private void task3_Load(object sender, EventArgs e)
56    {
57        //code load messages in combobox
58        loadOldMessages();
59        global.generateAlphabets(creds.password);
60        label1.Text = global.formatInitials(creds.initials) + " - Keyword: " + creds.password;
61    }
```

Three things happen here:

- The label's text is changed to 'Initials – Keyword'. The initials are formatted from lowercase attached (e.g. kdb) to uppercase formatted (e.g. K. D. B.). This is done through the formatInitials function in the globalToolBox class, located on common.cs:

```
40    public string formatInitials(string initials)
41    {
42        return initials.Aggregate(string.Empty, (c, i) => c + i + ". ").ToUpper().Trim();
43    }
```

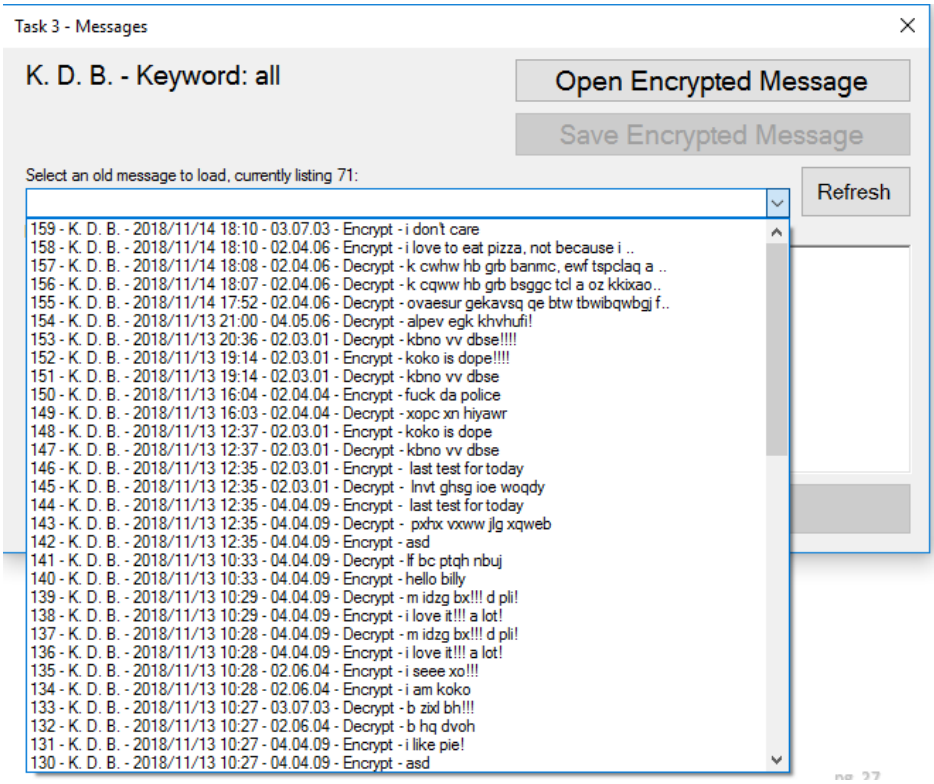
- The generateAlphabets function is called, passing the generated number set as parameter (the GNS value was generated on task3_login form and passed to the current form shortly before opening the task3 form, as shown earlier).

```
23    public int max = 122;
24    public int min = 97;
25    public List<int[]> generatedAlphabets = new List<int[]>();
26
```

```
public void generateAlphabets(string word)
{
    generatedAlphabets.Clear();
    //n of alphabets
    char[] alphabetStartLetter = word.ToCharArray();
    int startCharCode = 0;
    for (int k = 0; k < alphabetStartLetter.Length; k++)
    {
        startCharCode = (int)alphabetStartLetter[k];
        int[] alphabet = new int[26];
        int charCounter = startCharCode;
        for (int j = 0; j <= 25; j++)
        {
            if (charCounter > max)
                charCounter = min;

            alphabet[j] = charCounter;
            charCounter++;
        }
        generatedAlphabets.Add(alphabet);
    }
}
```

The generateAlphabets function essentially takes every word's letter and, for each, creates an array containing ASCII character codes representative of the lowercase alphabet letters. However, instead of starting from 97 (a) to 122(z), the arrays' first element starts from whatever the current letter of the loop cycle is (each of the keyword's characters being looped through).

- The loadOldMessages() function is called, which populates the combobox with items representing previous messages.



This is done with the following code:

```
public class singleMessage
{
    public string message { get; set; }
    public string date { get; set; }
    public string time { get; set; }
    public int id { get; set; }
    public string agentInitials { get; set; }
    public string generatedNumberSet { get; set; }
    public string operationType { get; set; }
}

public static class oldMessages
{
    public static List<singleMessage> messages = new List<singleMessage>();

    public static string getDateTime(int n)
    {
        return messages[n].date + " " + messages[n].time;
    }

}
```

```
public void loadOldMessages(string messagesQuerySettings = "")
{
    string amount = "0";
    string response = global.connectToDatabase("messages", messagesQuerySettings);
    comboBox2.Items.Clear();
    if (response != "none")
    {
        string[] messages = response.Split('|');
        amount = messages.Length.ToString();
        string[] tmp = null;
        int maxWords, id;
        string optionalDots;
        oldMessages.messages.Clear();
        singleMessage msg;
        foreach (string singleMessageDetails in messages)
        {
            tmp = singleMessageDetails.Split(';');
            msg = new singleMessage();
            id = 0;
            int.TryParse(tmp[0], out id);
            msg.id = id;
            msg.agentInitials = global.formatInitials(tmp[5]);
            msg.date = tmp[1].Replace("-", "/");
            msg.time = tmp[2];
            msg.generatedNumberSet = tmp[4];
            msg.operationType = tmp[6];
            msg.message = tmp[3];
            maxWords = msg.message.Length;
            optionalDots = "";
            if (msg.message.Length > 35)
            {
                maxWords = 35;
                optionalDots = "..";
            }
            oldMessages.messages.Add(msg);
            comboBox2.Items.Add(msg.id + " - " + msg.agentInitials + " - " + oldMessages.getDateTime(oldMessages.messages.Count-1) + " - " + msg.genera
        }
        label2.Text = "Select an old message to load, currently listing " + amount + ":";
    }
    else
    {
        string output = (messagesQuerySettings == "") ? "No messages saved yet." : "No messages found for the current search criteria.";
        comboBox2.Items.Add(output);
    }
    label2.Text = "Select an old message to load, currently listing " + amount + ":";
}
```

Where messageQuerySettings is a string indicating the filters to query old messages by (this is explained at the end).

As seen below, messages.php will output the following string:

```php
<?php
require("connection.php");
$operationType = $_POST['operationType'];
$q = $_POST["query"];
if($operationType == "read"){
    $sql = "SELECT * FROM `glyndwr_task3`";
    if(strpos($q, "keyword:") !== false){
        $sql .= " WHERE generatedNumberSet LIKE '%".str_replace("keyword:", "", $q)."%'";
    }elseif(strpos($q, "date:") !== false){
        $sql .= " WHERE date = '".str_replace("date:", "", $q)."'";
    }elseif(strpos($q, "between:") !== false){
        $dates = explode("?", str_replace("between:", "", $q));
         $sql .= " WHERE date >= '".$dates[0]."' AND date <= '".$dates[1]."'";
    }elseif(strpos($q, "agent:") !== false){
         $sql .= " WHERE operatorInitials = '".str_replace("agent:", "", $q)."'";
    }
    $sql .= " ORDER BY id DESC;";
    $string = "";
    $result = mysqli_query($link, $sql);
    $number = mysqli_num_rows($result);
    if ($number > 0) {
            while ($row=mysqli_fetch_array($result)) {
                $string .= $row["id"] . ";".  $row["date"] . ";" . $row["time"] . ";".  $row["message"] . ";".
                $row["generatedNumberSet"] . ";". $row["operatorInitials"] . ";". ucfirst($row["operationType"]) . "|"; //loops
                twice on second time sets code to generatedNumberSet
            }
        echo(substr($string, 0, strlen($string) - 1));
    }else{
        echo("none");
    }
}else{
    $details = explode("|", $q);
    $sql = "INSERT INTO `glyndwr_task3`(`id`, `date`, `time`, `message`, `generatedNumberSet`, `operatorInitials`, `operationType`)
    VALUES ('','".mysqli_real_escape_string($link, $details[0])."','".mysqli_real_escape_string($link,
    $details[1])."','".mysqli_real_escape_string($link, $details[2])."','".mysqli_real_escape_string($link,
    $details[3])."','".mysqli_real_escape_string($link, $details[4])."', '".mysqli_real_escape_string($link, $details[5])."')";
    if (mysqli_query($link, $sql)) {
       echo("added");
    }else{
        echo("issue");
    }
}
?>
```

This string, once returned to C#, is then split into an array of strings, which is then used to generate the *singleMessages*, which are then added to the *messages* list of singleMessages and finally inserted as items in the combobox. Since both are populated at the same time and in the same order, the item at the combobox's selected index will match whatever is in oldMessages.messages[combobox.selectedIndex].

3) On Encrypt/Decrypt

When inserting anything in one of the two rich textboxes, the following code is executed:

```
163         private void richTextBox1_TextChanged(object sender, EventArgs e)
164         {
165             richTextBox1.Text = global.regex.Replace(richTextBox1.Text.ToLower(), "");
166             if (!fromSave)
167                 button1.Text = "Encrypt!";
168             else
169                 button1.Text = "Encrypt" + button1.Text.Substring(7);
170
171             if (richTextBox1.Text.Length > 0)
172                 richTextBox1.SelectionStart = richTextBox1.Text.Length;
173                 richTextBox2.Text = "";
174
175             enableButton();
176         }
177         private void richTextBox2_TextChanged(object sender, EventArgs e)
178         {
179             richTextBox2.Text = global.regex.Replace(richTextBox2.Text.ToLower(), "");
180             if(!fromSave)
181                button1.Text = "Decrypt!";
182             else
183                button1.Text = "Decrypt" + button1.Text.Substring(7);
184
185             if (richTextBox2.Text.Length > 0)
186                 richTextBox2.SelectionStart = richTextBox2.Text.Length;
187             else
188                 button2.Enabled = false;
189                 richTextBox1.Text = "";
190
191             enableButton();
192         }
193         public void enableButton()
194         {
195             button1.Enabled = (richTextBox1.Text.Length > 0 || richTextBox2.Text.Length > 0) ? true : false;
196         }
```

This makes sure that if user starts typing in the Encrypt field, the Decrypt field will reset (and vice versa). Also, the input is checked for the following characters, and everything that does not match those is removed:

```
public Regex regex = new Regex("[^a-z '!?.,]");
```

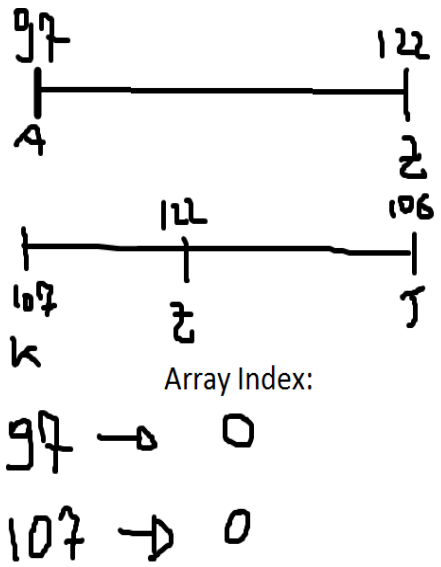
When there is a string in the Encrypt or Decrypt rich textboxes, the Encrypt/Decrypt button becomes enabled and when clicked triggers this code:

```
private void button1_Click(object sender, EventArgs e)
{
    comboBox2.SelectedIndex = -1;
    if (fromSave == true)
    {
        if(MessageBox.Show("Do you want to continue using the saved keyword?", "Continue with old keyword?", MessageBoxButtons.YesNo) != DialogResult.Yes)
        {
            fromSave = false;
            global.generateAlphabets(creds.password);
            button1.Text = (richTextBox2.Text.Length > 0) ? "Encrypt!" : "Decrypt!";
            comboBox2.SelectedIndex = -1;
        }
    }

    string message = "";
    string output = "";
    if (richTextBox1.Text.Length > 0)
    {
        //encrypt
        message = richTextBox1.Text;
        output = encrypt(message);
        MessageBox.Show("Encryption complete! - Clearing Encryption Text Box and Pasting in Decryption Text Box.");
        button2.Enabled = true;
        richTextBox1.Text = "";
        richTextBox2.Text = output;
    }
    else if (richTextBox2.Text.Length > 0)
    {
        //decrypt
        message = richTextBox2.Text;
        output = decrypt(message);
        MessageBox.Show("Decryption complete! - Clearing Dencryption Text Box and Pasting in Encryption Text Box.");
        richTextBox2.Text = "";
        richTextBox1.Text = output;
    }

    updateDatabase(output);
}
```

Depending on which rich textbox has input, I am deducing if the single button is clicked for encrypting or decrypting purposes, based on that the encrypt() and decrypt() functions are called.

```
public string encrypt(string message)
{
    int alphabetCount = 0;
    int alphabetCountMax = global.generatedAlphabets.Count;
    string output = "";
    foreach (char c in message)
    {
        if (c != '?' && c != '!' && c != ',' && c != '.' && c != ' ' && (int)c != 39)
        {
            if (alphabetCount >= alphabetCountMax)
                alphabetCount = 0;

            output += ((char)global.generatedAlphabets[alphabetCount][(25 - (global.max - (int)c))]).ToString();
            alphabetCount++;
        }
        else
        {
            output += c.ToString();
        }
    }
    return output;
}

public string decrypt(string message)
{
    int alphabetCount = 0;
    int alphabetCountMax = global.generatedAlphabets.Count;
    string output = "";
    foreach (char c in message)
    {
        if (c != '?' && c != '!' && c != ',' && c != '.' && c != ' ' && (int)c != 39)
        {
            if (alphabetCount >= alphabetCountMax)
                alphabetCount = 0;

            output += ((char)(global.min + Array.IndexOf(global.generatedAlphabets[alphabetCount], (int)c))).ToString();
            alphabetCount++;
        }
        else
        {
            output += c;
        }
    }
     return output;
}
```

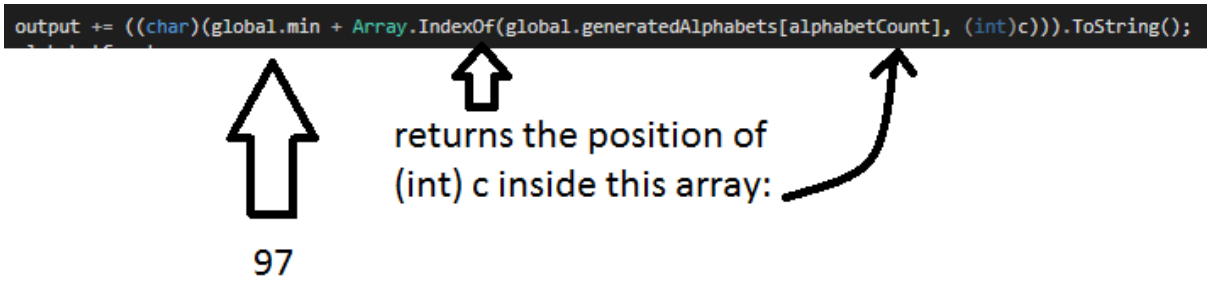
The encrypt method follows this logic:



Since the generated alphabets all have the same number of letters as the normal alphabet, the array containing the letters of the normal alphabet also has the same number of elements as the arrays containing the letters of the generated alphabets.
So
alphabet[0] = a, just as alphabet_from_k[0] = k
and
alphabet[25] = z, just as alphabet_from_k[25] = j

This means that, if z = 122 in ascii code, then doing 25 – (122 – 122) = 25 so alphabet_from_k[25] = j
Just as if b = 98, doing 25 – (122 – 98) = 1 so..
alphabet[1] = b just as alphabet_from_k[1] = l.

The decrypt method follows the same logic as the encrypt method, however the operation is inverted:



For example, 97 + (position of j in alphabet_from_k array) => 97 + 25 = 122 which, once transformed from int to char, gives the decrypted letter for j, which in this example is z since the generatedAlphabet starts from k.

```
if (richTextBox1.Text.Length > 0)
{
    //encrypt
    message = richTextBox1.Text;
    output = encrypt(message);
    MessageBox.Show("Encryption complet
    button2.Enabled = true;
    richTextBox1.Text = "";
    richTextBox2.Text = output;
}
else if (richTextBox2.Text.Length > 0)
{
    //decrypt
    message = richTextBox2.Text;
    output = decrypt(message);
    MessageBox.Show("Decryption complet
    richTextBox2.Text = "";
    richTextBox1.Text = output;
}

updateDatabase(output);
}
```

Once the output of decrypt/encrypt functions is returned, it is inserted into the relative rich textboxes.

Afterwards the updateDatabase function is called (shown below).

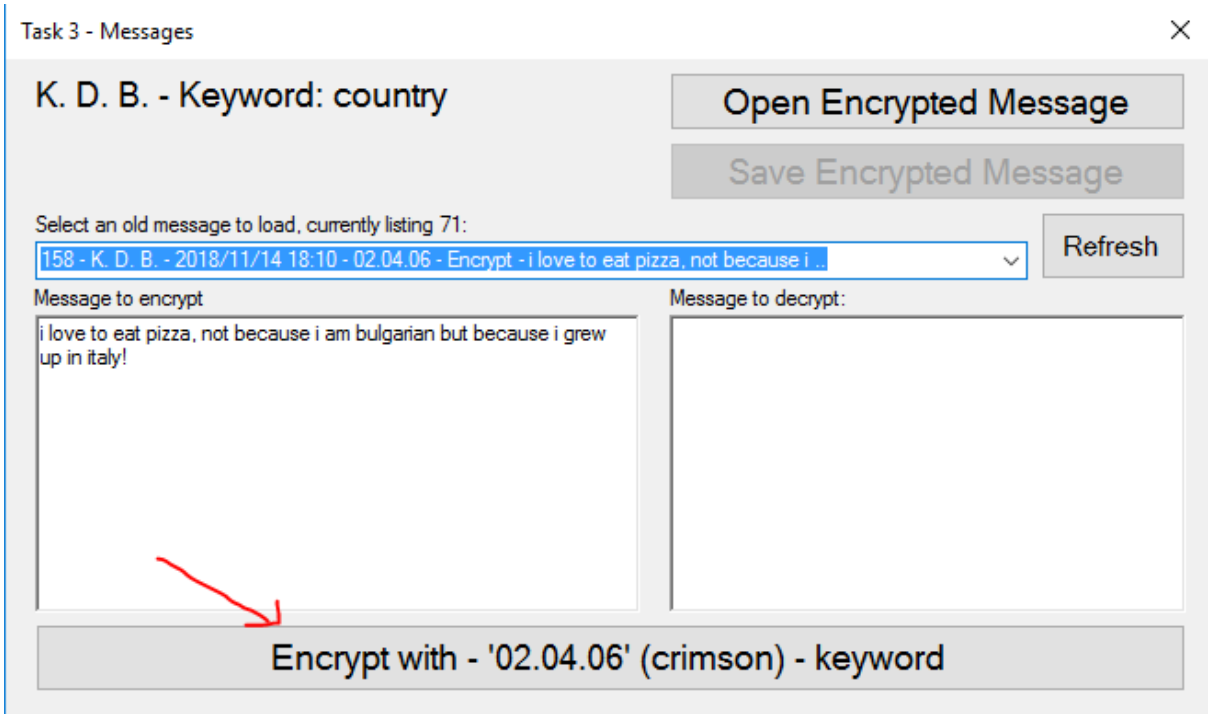
It does a rest call to messages.php with write as operationType parameter, and the query string containing everything required to perform the INSERT sql query (shown further below).

```
public void updateDatabase(string output)
{
    string generatedNumberSet = creds.generatedNumberSet;
    if(fromSave == true)//if save file, set variable to old nset
        generatedNumberSet = button1.Text.Split('\'')[1];

    string lastOperation = (richTextBox2.Text.Length > 0) ? "Decrypt" : "Encrypt";
    string query = DateTime.Now.ToString("yyyy-MM-dd") + "|" + DateTime.Now.ToString("HH:mm") + "|" + output + "|" + generatedNumberSet + "|" +
    if (global.connectToDatabase("messages", query, "write") != "added") {
        MessageBox.Show("Message could not be saved to database.");
    };
}
```

```
}else{
    $q = $_POST["query"];
    $details = explode("|", $q);
    $sql = "INSERT INTO `glyndwr_task3`(`id`, `date`, `time`, `message`, `generatedNumberSet`, `operatorInitials`, `operationType`)
    VALUES ('','".mysqli_real_escape_string($link, $details[0])."','".mysqli_real_escape_string($link,
    $details[1])."','".mysqli_real_escape_string($link, $details[2])."','".mysqli_real_escape_string($link,
    $details[3])."','".mysqli_real_escape_string($link, $details[4])."', '".mysqli_real_escape_string($link, $details[5])."')";
    if (mysqli_query($link, $sql)) {
       echo("added");
    }else{
        echo("issue");
    }
}
?>
```

4) On Old Message Loaded:

When an old message is selected from the combobox, the fromSave variable is set to true, and the Text property of the button1 is changed (from Encrypt/Decrypt to what is shown below).



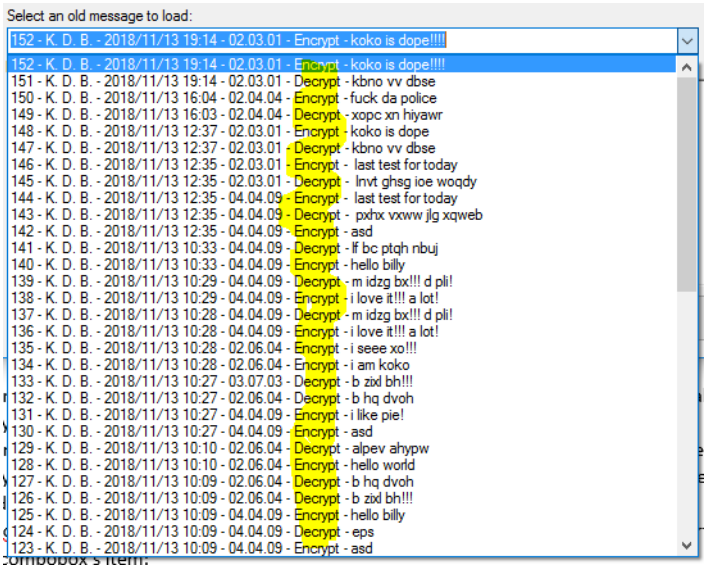
When something is selected from the combobox the following code is executed:

```
private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    int n = comboBox2.SelectedIndex;
    if (n > -1 && comboBox2.Items[n].ToString().Length > 24)
    {
        string[] content = { oldMessages.messages[n].generatedNumberSet, oldMessages.messages[n].message };
        generateForOldMessages(content);
        if (oldMessages.messages[n].operationType.ToLower() == "encrypt")
        {
            richTextBox1.Text = content[1];
            button2.Enabled = false;
        }
        else
        {
            richTextBox2.Text = content[1];
            button2.Enabled = true;
        }
    }
}
```

The IF statement makes sure that the selected Index does not point to nothing and that the selected Item is not “No messages saved yet.”, which is the only item that loadOldMessages adds to the combobox if messages.php returns ‘’none’’ on read.

Afterwards, based on the selected index of the combobox, I set the content string array to content[1] being the generated number set and content[1] the encrypted/decrypted message.

The operationType variable contains either encrypt or decrypt and is taken from this part of the combobox's item:



The generateForOldMessages function, which accepts as parameter the array "content" (which contains the generated number set and encrypted/decrypted message) does the following:

```
public void generateForOldMessages(string[] content) {
    fromSave = true;
    string word = getWordFromPoemList(content[0]);
    global.generateAlphabets(word);
    richTextBox1.Text = "";
    richTextBox2.Text = "";
    button1.Text = button1.Text.Substring(0, 7) + " with - '" + content[0] + "' (" + word + ") - keyword";
}
```

- Sets the variable fromSave to true;
- Sets the variable word, which is the keyword used for encrypting/decrypting, to the string returned by the getWordFromPoemList function:

```
public string getWordFromPoemList(string code)
{
    string[] info = code.Split('.');
    int poem = 0; int line = 0; int wordC = 0;
    if (int.TryParse(info[0], out poem) && int.TryParse(info[1], out line) && int.TryParse(info[2], out wordC)) {
    string[] lines = global.poemsList[poem - 1].Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);
    string[] words = global.regexA.Replace(lines[line - 1].ToLower(), "").Split(' ');
        return words[wordC - 1];
    }
    return "";
}
```

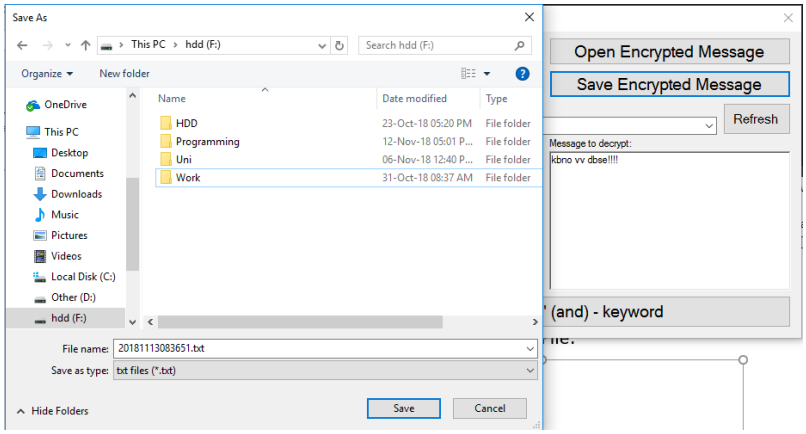
- Alphabets are generated with the word variable (the old keyword used), through the generateAlphabets function.
- The button1 text property is changed to the format shown below:

When button1 is pressed the following code, which was explained previously, is executed:

```
private void button1_Click(object sender, EventArgs e)
{
    comboBox2.SelectedIndex = -1;
    if (fromSave == true)
    {
        if(MessageBox.Show("Do you want to continue using the saved keyword?", "Continue with old keyword?", MessageBoxButtons.YesNo) != DialogResult.Yes)
        {
            fromSave = false;
            global.generateAlphabets(creds.password);
            button1.Text = (richTextBox2.Text.Length > 0) ? "Encrypt!" : "Decrypt!";
            comboBox2.SelectedIndex = -1;
        }
    }

    string message = "";
    string output = "";
    if (richTextBox1.Text.Length > 0)
    {
        //encrypt
        message = richTextBox1.Text;
        output = encrypt(message);
        MessageBox.Show("Encryption complete! - Clearing Encryption Text Box and Pasting in Decryption Text Box.");
        button2.Enabled = true;
        richTextBox1.Text = "";
        richTextBox2.Text = output;
    }
    else if (richTextBox2.Text.Length > 0)
    {
        //decrypt
        message = richTextBox2.Text;
        output = decrypt(message);
        MessageBox.Show("Decryption complete! - Clearing Dencryption Text Box and Pasting in Encryption Text Box.");
        richTextBox2.Text = "";
        richTextBox1.Text = output;
    }

    updateDatabase(output);
}
```

The only new thing here is that in case fromSave is true, then ask the user if they would like to encrypt/decrypt using the old keyword or the current one. If the user wants to encrypt/decrypt using the new keyword, then the generateAlphabets function is called, passing the original daily keyword (which is stored in the creds.password variable [in retrospect, a misleading name!]). This will re-generate the alphabets that the encrypt/decrypt functions base their logic on.

4) On Save Message To File:



The Save Encrypted Message button fires the writeToSeelectedFolder function inside of the globalToolBox class, as shown below:

```
private void button2_Click(object sender, EventArgs e)
{
    string generatedNumber = creds.generatedNumberSet;
    if (fromSave)
        generatedNumber = button1.Text.Split('\'')[1];

    if (global.writeToSeelectedFolder(generatedNumber + "|" + richTextBox2.Text, true))
    {
        richTextBox2.Text = "";
        comboBox2.SelectedIndex = -1;
    }
}
```

This sets the generatedNumber variable to, first, the default generated number set (which was initially set from task3_login, before opening task3. Then, in case fromSave is true, it sets the generatedNumber variable to this part of the button:

Decrypt with - '02.03.01' (and) - keyword

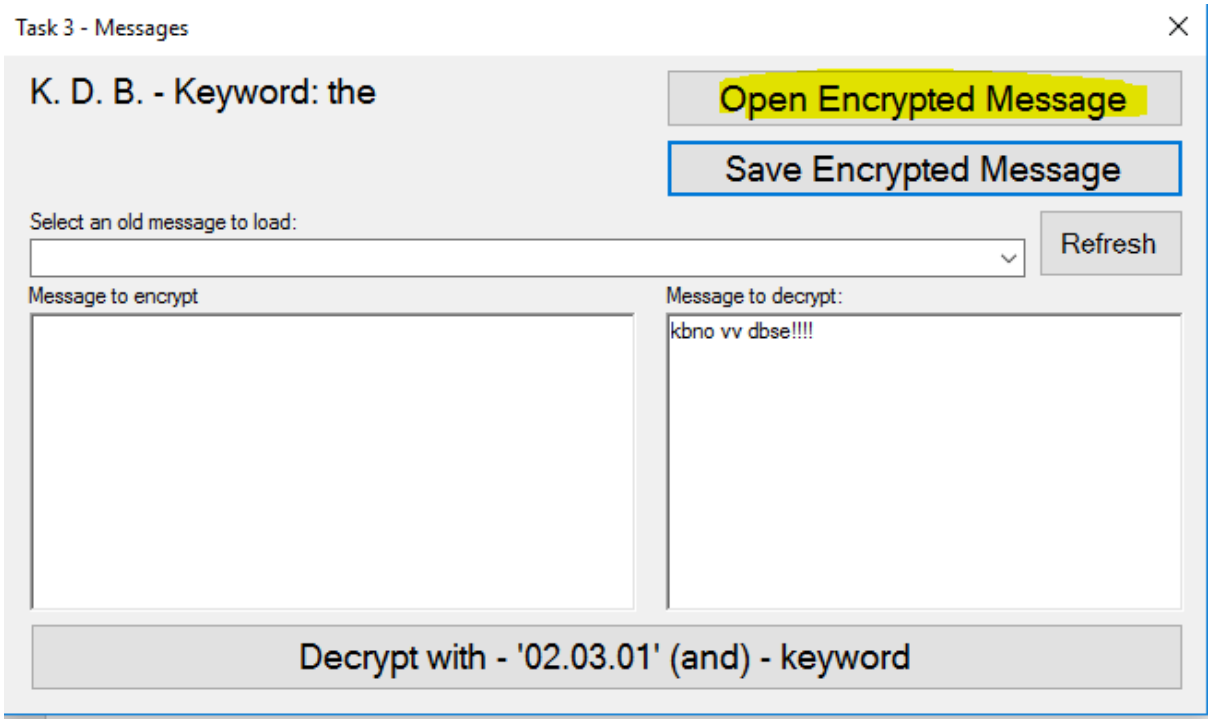
Finally, the writeToSeelectedFolder function is called, passing the generatedNumberSet concatenated to the encrypted/decrypted message.
This function was shown on task2's documentation, since it also makes use of it. This time the second parameter passed is true, meaning that instead of a FolderBrowerDialog, a SaveFileDialog will be shown:

```
public bool writeToSeelectedFolder(string output, bool askForFileName = false)
{
    string path = "";
    DialogResult result;
    DateTime time = DateTime.Now;
    string name = time.ToString("yyyyMMddhhmmss");
    if (!askForFileName)
    {
        FolderBrowserDialog FolderBrowser = new FolderBrowserDialog();
        FolderBrowser.Description = "Select which folder to save the encrypted message into.";
        result = FolderBrowser.ShowDialog();
        path = FolderBrowser.SelectedPath + "/" + name + ".txt";
    }
    else
    {
        SaveFileDialog saveFile = new SaveFileDialog();
        saveFile.Filter = "txt files (*.txt)|*.txt";
        saveFile.FilterIndex = 1;
        saveFile.RestoreDirectory = true;
        saveFile.FileName = name + ".txt";
        result = saveFile.ShowDialog();
        path = Path.GetFullPath(saveFile.FileName);
    }
```

When the save is complete, rich textbox fields are deleted and the combobox's selected item is set to -1.

4) On Open of File:

Clicking this button:



Runs this code:

```
private void button3_Click(object sender, EventArgs e)
{
    richTextBox1.Text = "";

    string[] content = global.loadEncryptedMessageFromFolder();
    if (content != null)
    {
        generateForOldMessages(content);
        richTextBox1.Text = decrypt(content[1]);
        comboBox2.SelectedIndex = -1;
    }
}
```

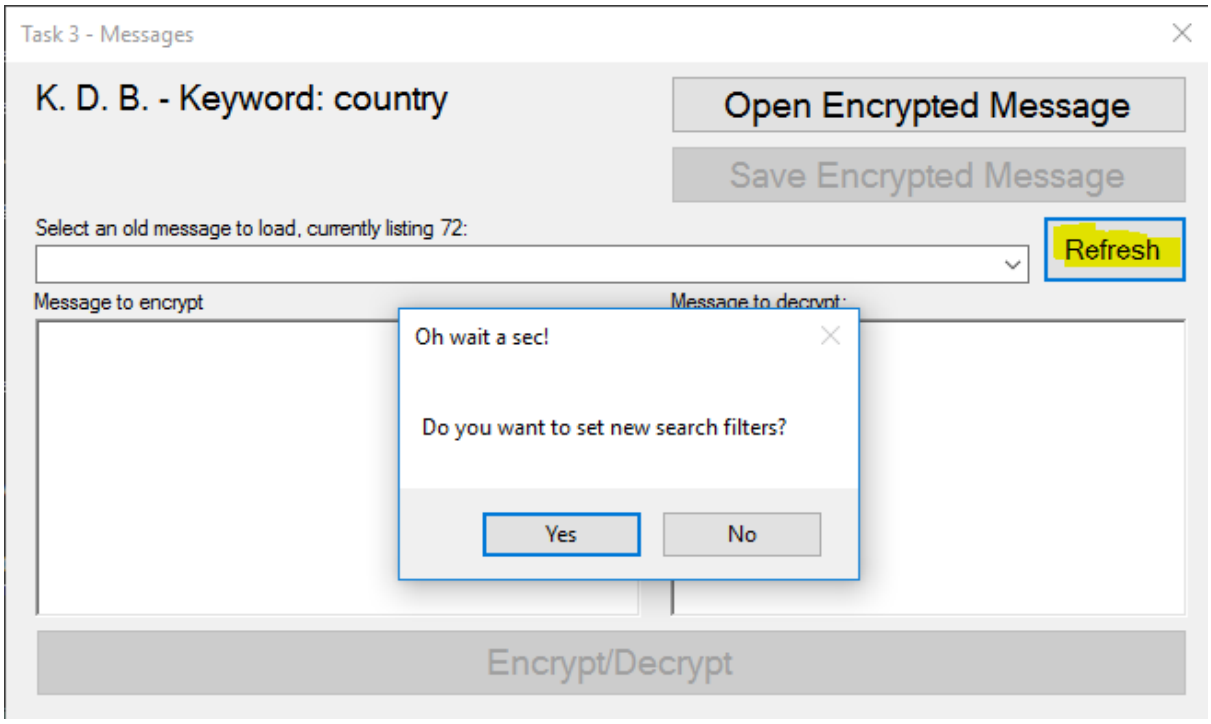
Which empties the rich textboxes and sets content to the output of the loadEncryptedMessagesFromFolder function (which was documented for task 2 as well).

```
112
113     public string[] loadEncryptedMessageFromFolder()
114     {
115         OpenFileDialog fileBrowserDialog = new OpenFileDialog();
116         fileBrowserDialog.Title = "Select which file to load encrypted message from.";
117         fileBrowserDialog.Filter = "TXT files|*.txt";
118         fileBrowserDialog.InitialDirectory = @"C:\";
119         if (fileBrowserDialog.ShowDialog() == DialogResult.OK)
120         {
121             string filePathFileName = fileBrowserDialog.FileName; // file will always exist when selected from dialog
122             try
123             {
124                 read = new StreamReader(filePathFileName);
125                 string line = read.ReadLine(); // I save as shitfBy|text, so always only 1 line
126                 if(line.Length > 0 && line.Contains("|"))
127                 {
128                     string[] content = line.Split('|');
129                     read.Close();
130                     MessageBox.Show("Encrypted message from file '" + filePathFileName + "' was successfully loaded!");
131                     return content;
132                 }
133                 else
134                 {
135                     MessageBox.Show("The file you are tyring to open seems unusual!");
136                 }
137             }
138             catch (Exception ex)
139             {
140                 MessageBox.Show("There was an issue when trying to open " + filePathFileName + "." + ex.ToString());
141             }
142         }
143         return null;
144     }
```

Task 2's save file's content's format was: ShiftBy|Message, whereas Task 3's save file's content's format is GeneratedNumberSet|Message, so the same function can be used for both.

Once the content variable is set to the array returned by the loadEncryptedMessagesFromFolder function,
I check if it is not null, in case of an empty/invalid file opened, and then the following important pieces of code are executed:
- run the generateForOldMessages function, passing the content array variable as parameter. The generateForOldMessages function gets the word represented by the generated number set taken from the save file and stored into content[0] and generates alphabets with it.
- runs the decrypt function, passing as parameter the encrypted message taken from the save file and loaded into content[1].

5) On Refresh Button Click:

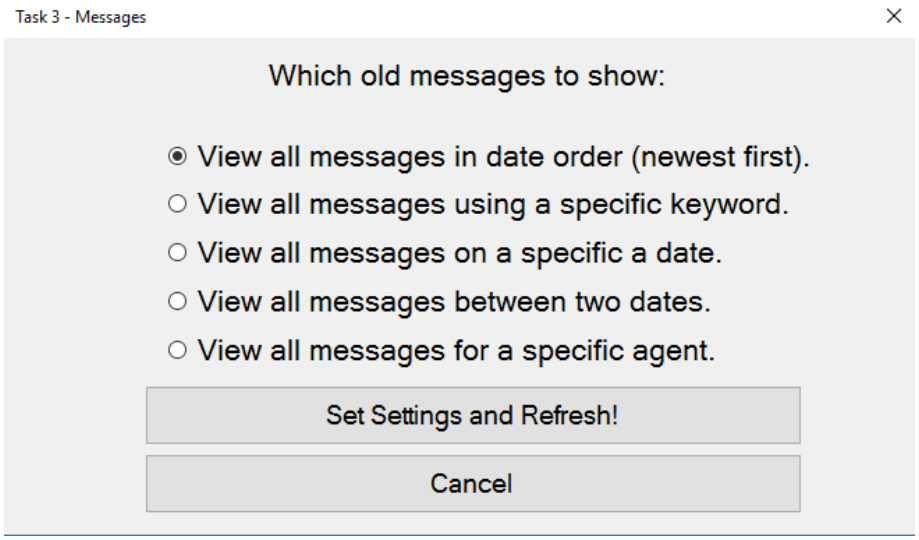


If user presses No, then the following code executes:

```
private void button4_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Do you want to set new search filters?", "Oh wait a sec!", MessageBoxButtons.YesNo) != DialogResult.Yes) {
        MessageBox.Show("Please wait while new messages are fetched");
        loadOldMessages();
        MessageBox.Show("New messages successfully updated!");
    }
    else
    {
        panel1.Visible = true;
    }
}
```

Which just calls loadOldMessages, which was explained earlier.

If the user presses Yes, though, this panel is set to visible:



After selecting a radio button option, and clicking the first button, this code executes:

```
private void button5_Click(object sender, EventArgs e)
{
    string query = "";
    if (radioButton2.Checked)
        query = "keyword:" + textBox1.Text.Trim().ToLower();
    else if (radioButton3.Checked)
        query = "date:" + dateTimePicker1.Value.ToString("yyyy-MM-dd").Trim();
    else if (radioButton4.Checked)
        query = "between:" +  dateTimePicker2.Value.ToString("yyyy-MM-dd").Trim() + "?" + dateTimePicker3.Value.ToString("yyyy-MM-dd").Trim();
    else if (radioButton5.Checked)
        query = "agent:"+ textBox1.Text.Trim().ToLower().Replace(".", "").Replace(".", "").Replace(" ", "");

    MessageBox.Show("Please wait while new messages are fetched");
    loadOldMessages(query);
    MessageBox.Show("New messages successfully updated!");
    panel1.Visible = false;
    resetRefresh();
}
```

Essentially, loadOldMessages is called, and the query string is passed as parameter. Based on the query string passed, on messages.php I modify the SQL select query accordingly.
When loadOldMessages was explained, the server-side code was shown.
After refreshing the combobox's contents, the panel is hidden and the function reserRefresh is ran:

```
public void resetRefresh()
{
    panel1.Visible = false;
    resetProcess = true;
    radioButton1.Checked = true;
    int[] radioButtonPositions = { 63, 97, 132, 167, 202 };
    foreach(Control rb in panel1.Controls)
        if (rb is RadioButton)
        {
            rb.Visible = true;
            rb.Location = new System.Drawing.Point(rb.Location.X, radioButtonPositions[int.Parse(rb.Name.Substring(rb.Name.Length-1))-1]);
        }

    textBox1.Visible = false;
    dateTimePicker2.Visible = false;
    dateTimePicker3.Visible = false;
    textBox1.Text = "";
    label6.Visible = false;
    label7.Visible = false;
    dateTimePicker1.Visible = false;
    button5.Enabled = true;
}
```

Which resets all form elements in the panel to their default state (almost).