NORTHWEST NAZARENE UNIVERSITY

Wire Manager: A Financial Wire Transfer Management Solution

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Edgar Madrigal Sosa
2021

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Edgar Madrigal Sosa
2021

Wire Manager: A Financial Wire Transfer Management Solution

Author: Edgar M Sosa
Edgar M Sosa

Approved: Barry Myers
Barry Myers, Ph.D., Professor of Computer Science, Department of Mathematics and Computer Science, Faculty Advisor

Approved: Cesar Sosa
Cesar Sosa, B.S., P.A. Student
Second Reader

Approved: Barry Myers
Barry L. Myers, Ph.D., Chair,
Department of Mathematics & Computer Science

ABSTRACT

Wire Manager: A Financial Wire Transfer Management Solution.
SOSA, EDGAR (Department of Mathematics and Computer Science), MYERS, DR. BARRY (Department of Mathematics & Computer Science).

Large financial institutions transfer millions of dollars to and from other financial institutions on a regular basis. This same process occurs on a smaller scale in between individuals. Particularly, individuals sending money abroad to other countries. Given that currency is being sent from one country to another, this process is monitored by state and federal laws that aim to stop potential criminal activity. These state and federal laws add mandatory government regulations for financial institutions that wire money. The Wire Manager program implements simple but efficient tools to help combat money laundering and prepare for company audits. Money laundering prevention can be challenging for small businesses that do not have enterprise-class software to combat this style of criminal activity. Wire Manager facilitates money-laundering prevention by providing a single software system where wire transaction data can be stored and analyzed by the application's built-in tools. Wire Manager's tools include multiple parameter filter searching, background process analysis of potential money laundering cases (based off of user-provided data), a flagging system that aids employees in identifying potential criminal activity, and persistent profile note-keeping used to communicate information about potential threats within an organization.

Acknowledgements

I would like to give special thanks to the two most influential mentors in my life; my mom and dad (Laura Sosa and Gerardo Sosa). Had it not been for them, I would not be the person I am today. I would also like to give thanks to my younger brothers (Omar Sosa and Cesar Sosa) for helping me through some of my most difficult hardships. I also want to thank Dr. Barry Myers, Dr. Dale Hamilton, and Dr. Kevin McCarty for being life shaping educators and mentors in my journey towards a computer science degree.

Table of Contents

Table of Figures

Introduction

The Wire Manager program originated the acknowledgment that small financial entities require a means to monitor money being wired abroad. Although larger businesses follow the same stipulations, smaller companies sometimes struggle to keep up with federal and state regulations placed on wiring money abroad due to the lack of resources and tools available to them. Wire Manager provides tools which make it easier to fulfill federal and state law regulations placed on smalls business which wire money abroad.

Background

The use and transfer of currency is incredibly important in today's society. In order to create new business ventures, one must have the means to do so. In order to consume essentials (such as food, water, entertainment, etc.…), one must have the means to do so. In order to do anything significant in entrepreneurship, one must have the means to do so. Here "means" deals with financial currency. With the rise of globalization, came the emphasis on national currencies. This allowed for the transfer of goods on a larger scale. Through the nature of governments came the need to regulate imports and exports, especially when observing trades in national currency.

By regulating currency, governments are able to more accurately understand how and why currency is being traded. In order to regulate currency, laws are placed at the federal and state levels to ensure each transaction is accounted for. This is the case with large financial institutions and also with smaller financial institutions. Through the use of computers and software, it became easier to analyze the trading of financial currency (the process of transferring financial currency, electronically, is also known as a wire

transfer). The regulation of wire transfers is increasingly more important when the transfer is abroad. This allows for governments to analyze where their money is going and for what reasons.

For larger institutions there are more resources available to purchase and create the software necessary to ensure that all state and federal laws are being followed. For smaller institutions this can be a bit more difficult as there are less resources available to do the same thing. Software has become an important tool when it comes to monitoring wire transactions (data), and by having the means to obtain this type of software it is much easier to ensure one is abiding the corresponding rules. Federal and state laws require that information be provided to the government dependent on the amount of money particular individuals are sending abroad. These same laws may also require denying a wire transfer when certain financial thresholds are met.

Project Overview

The purpose of Wire Manger is to aid smaller financial institutions in analyzing the corresponding wire transaction data, in order to help them make informed decisions when needing to deny services to a client or provide necessary information to governmental authorities. Wire Manager is able to help with this process by incorporating tools which allow for:

- Multiple parameter filter searching
- Background process analysis of potential money laundering cases
- Flagging system aiding in identifying potential criminal activity
- Persistent profile note-keeping used to communicate intraorganizational information

Body

As with any significant project, there is a process. A project requires brainstorming, research, trial and error, implementation, and revision. Wire Manager was built through these steps (sometimes going backward instead of forwards). The contents within the "Body" section elaborate on this process regarding Wire Manager. The "Body" entails the difficulties and successful implementations of this project.

Product Requirements

An essential step in the software development process is understanding the problem and creating an effective solution. With this project, a problem was identified, and a resolution was built via communication with the corresponding client. The requirements for Wire Manager were determined by active communication with the client. The client needs were to have a more organized and straightforward manner of monitoring the individual wire transaction data on a customer per customer basis. In more detail, the client's needs are as follows:

- A method of quickly searching customer data to determine if they've surpassed a $3000 limit within the last 30 days
- The ability to add notes to a customer's name and flag them if there is a potential for criminal activity
- A manner in which a more advanced search can be conducted on the wire transaction data in order to determine accumulated amounts sent and received according to user-specified dates

Meeting with the client was incredibly important in understanding how to design and implement the end product (from a computer science perspective). After meeting with the

client, there came the time to create the requirements necessary to implement the client's needs. These were the requirements created after understanding the client's needs:

- The ability to add wire transaction data to a database
- A method of adding notes to the name of either a sender or receiver participating in a wire transaction
- The option of flagging a customer who may be of potential for criminal activity
- A manner of quickly searching through wire transaction data to determine the accumulated amount of an individual given a particular date range
- A quick method of viewing the wire transaction data for an individual within the last 30 days
    - Determining if the customer associated with the wire transaction data has surpassed the $3000 thirty-day limit
    - Identifying if a flag is associated with the customer in question
    - Viewing the notes attached (if any) with the customer in question

Technical Project Decisions

Much consideration went into what programming language to use for the Wire Manger application. Considering the client wanted the application to be local and not web-based, all potential web-development implementations were disregarded. Two programming languages considered for a local desktop application were Java and C#. Java was considered for its ability to work on multiple operating system environments, ease of use when building a graphical user interface, and for its strong principles in object-oriented design. C# was considered for its ease of use with Windows operating

system graphical user interfaces and object-oriented design principles. C# was chosen as the programming language of use as the user wanted the application to be developed for the Windows operating system. It was specified that the application would not be installed on any other platform. After researching C# and the tools it provides for front-end and back-end development, a decision was made that this language would be a strong choice for developing Wire Manager.

Having decided on the programming language, next came the choice of which database technology to use. MySQL was seriously considered for its ease of use and widespread resources. However, SQL Server became the final choice as it was highly integrated with Visual Studio (the integrated development environment to be used) and Microsoft (the creators of Windows) products. SQL Server showed significant promise in being able to scale up if ever needed. This was important from a developer standpoint.

Initial Database Implementation

As mentioned earlier (under the Technical Project Decisions section), the technology of choice for the database is SQL Server. The database was important for storing the wire transaction data and allowing the application to retrieve the necessary information from it. Database experimentation and maintenance were made possible by using Microsoft SQL Server Management Studio. The following are the different database tables created and their corresponding data fields:

Wires table (used for wire transaction data)

- wire_transaction_id (PK, bigint, not null)
- amount (money, not null)
- wire_number (varchar(100), not null)

- date (date, not null)
- agency (varchar(100), not null)
- sender_name (varchar(100), not null)
- receiver_name (varchar(100), not null)

Notes table

- note_id (PK, bigint, not null)
- date (date, not null)
- client_name (varchar(100), not null)
- note (varchar(1000), not null)

Flags table

- flag_id (PK, bigint, not null)
- client_name (varchar(100), not null)
- flag_status (bit, not null)

The initial database diagram for Wire Manager can be seen in Figure 1 below.

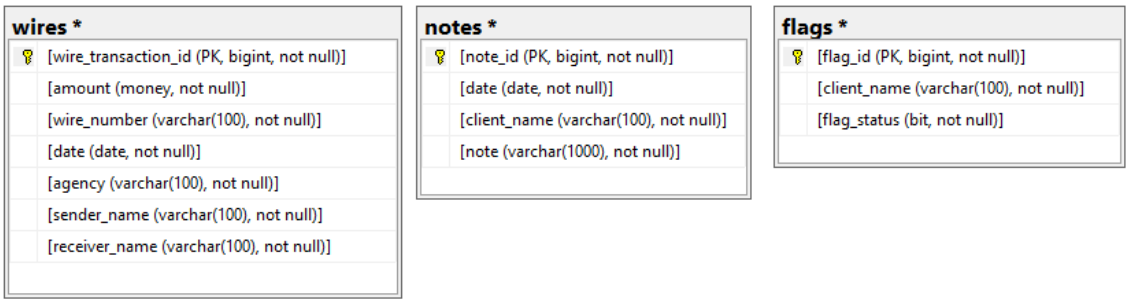

Figure 1: Database Diagram Tables

Initial Database Deficiencies

The purpose of the initial database design was to allow for quick data import into the application. The client requesting this application currently uses Microsoft Excel to store all of their wire transaction data. With minimal data standardization, it was easy to quickly import thousands of existing records into the database without creating a separate data import application using a sophisticated parser.

While functional, there are some database deficiencies presented in the Figure 1 Database Diagram. The initial database did not take advantage of a database's relational capabilities. Having not taken advantage of a database's relational capabilities led to redundancy in data. Data redundancy is a problem for the Wire Manager application because it could later lead to extra storage allocation for the overall application when it's not necessary. Furthermore, as the data set continues to grow, the overall performance of this application could be affected because of the length of time it takes to retrieve the required information. The initial database diagram does not best support a scalable application and requires a different design that would allow it to be more scalable for future development. With this in mind, it is best to approach a data import technique (for records already stored in Microsoft Excel) with a parsing method so a different database design can be used.

New Database Design

A new database schematic was designed to meet the deficiencies of the current database diagram. While not yet implemented, this design will be used in the future as development on Wire Manager continues. This diagram can be seen in the figure below:

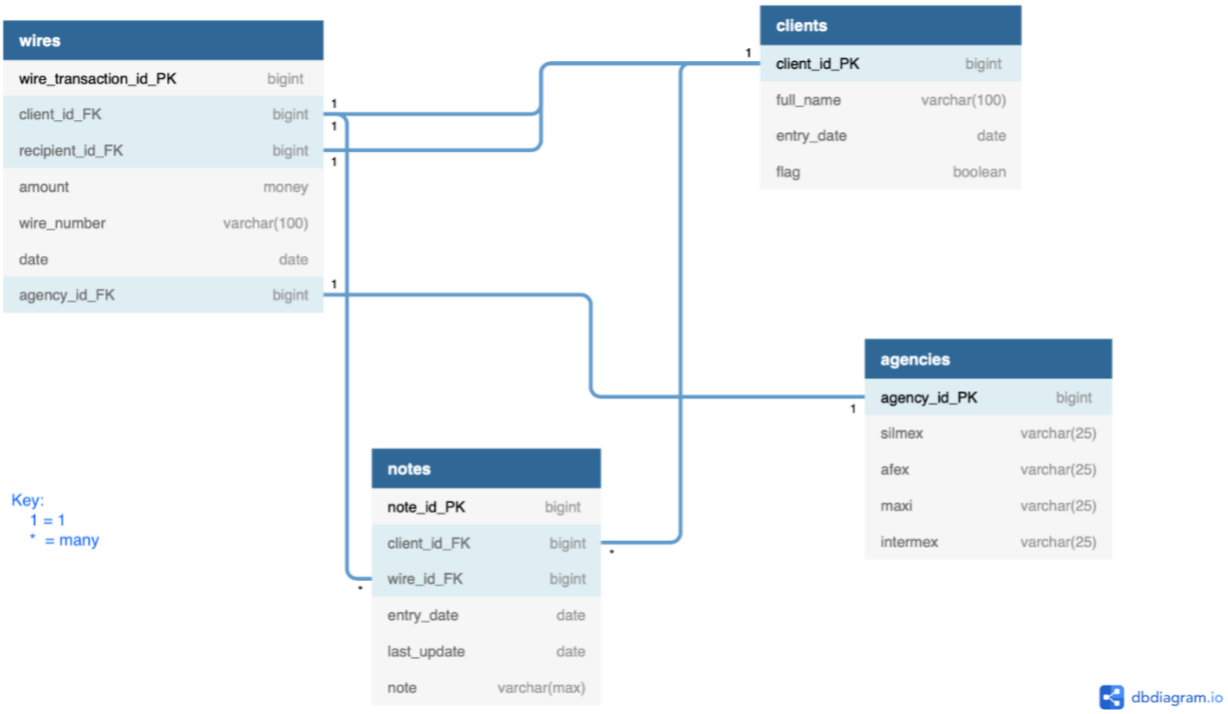

Figure 2: New Database Design

The entity-relationship diagram above more appropriately uses relationships to address the scalability and redundancy issues seen in the original database diagram. There are 1-to-1 relationships between the wires table to the clients table and the wires table to the agencies table. There are also appropriate 1-to-many relationships between the wires table and the clients table to the notes table. These relationships significantly reduce the amount of information needing to be held in the database. There are also some extra fields added, which allow more bookkeeping on the different notes and clients added.

Add Wire User Interface and Implementation

A key part of Wire Manager is to add wire transaction data. The user interface for this functionality can be seen in Figure 3 below.

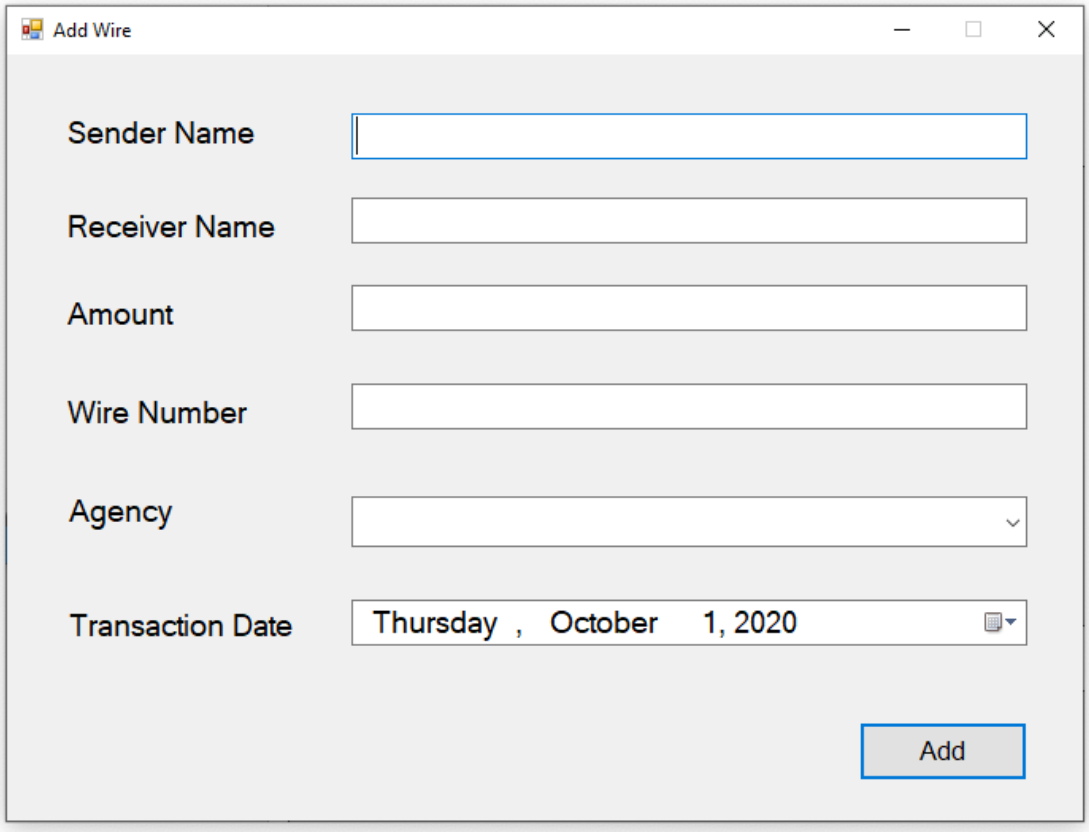

Figure 3: Add Wire User Interface

It is essential to note the "Agency" field dropdown. There are multiple agencies that the client for this project uses to fulfill wire transactions. The dropdown displays a list of the different agencies available. At the moment, the four different agencies were manually programmed into the dropdown as they are the only agencies used by the user. The new database diagram shows how this will be managed once it is implemented. Instead of inputing a new string into the table each time, the agencies table will be referenced in order to make better use of data storage. After a user adds the information, the Add

button is selected, and the program adds the information to the database (so long as all fields are valid). An error or success message is then displayed to the user to verify if the information was successfully added to the database or not. Within the wires table, a new transaction id is created using the auto-increment capability in SQL server.

Add Note/Flag User Interface and Implementation

Adding notes and/or flagging a customer is done through the Add Note window. This window can be seen in Figure 4 below.

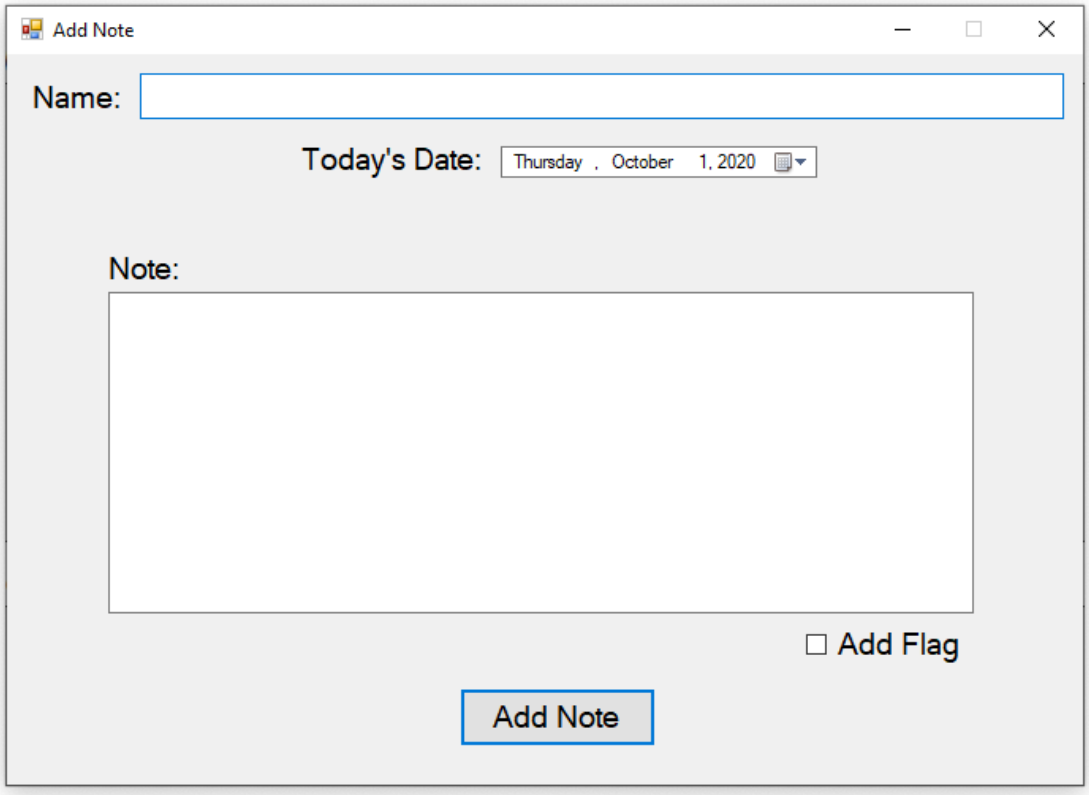

Figure 4: Add Note/Flag User Interface

This form allows a user (in this case, an employee) to add a customer's name (a customer coming to the store wishing to perform a transaction) to the name text box. A note can then be added to the corresponding customer. A date for the note can then be selected, and the note is added to the corresponding text box. An add flag checkbox is provided if a

user (employee) would like to add a flag to the customer in question. After add note is selected, the application adds the corresponding information to the database. If adding the information to the database was successful, a success message is displayed. Otherwise, an error message is shown to the user.

Advanced Search User Interface and Implementation

A slightly more sophisticated search is performed through the advanced search window. This figure can be seen in Figure 5 below.

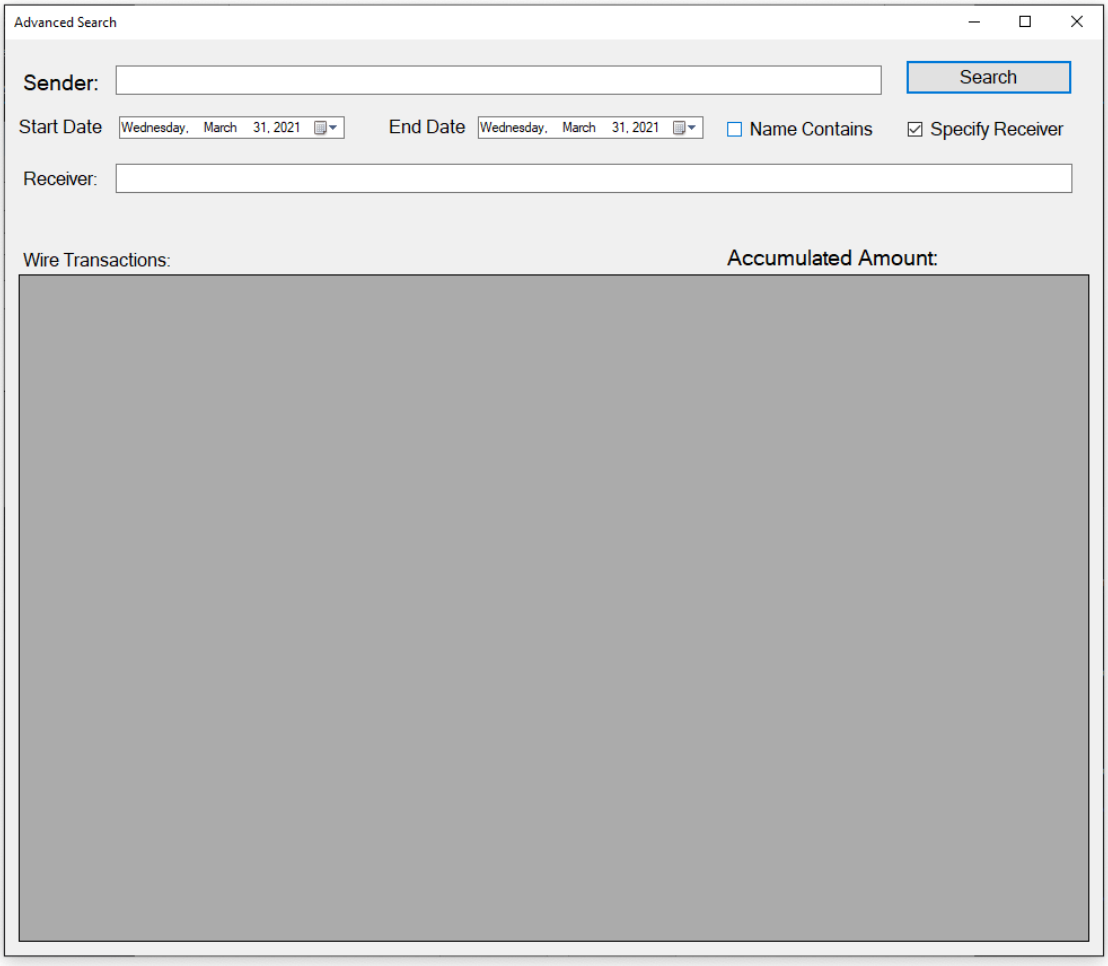

Figure 5: Advanced Search User Interface

The advanced search adds the ability to specify a date range, sender, and receiver. There are two check boxes which allow for extra functionality. These two are shown as:

- Name Contains
- Specify Receiver

Without selecting any of the checkboxes, a search is done on the given date range querying the database for the given “Sender” name. The wire transaction details are then displayed along with the accumulated amount. An example of this can be seen in Figure 6 below.

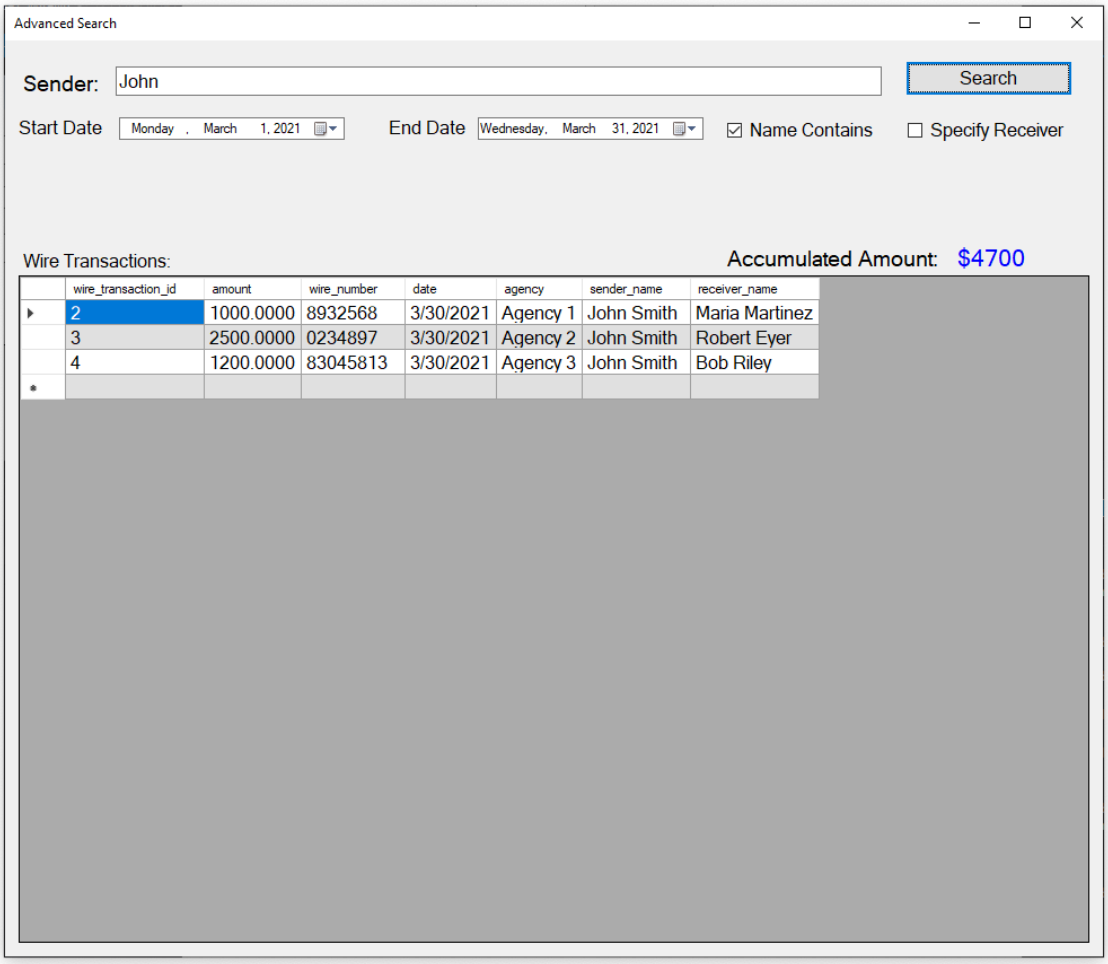

Figure 6: Advanced Search Example

Elaborating on the “Name Contains” and the “Specify Receiver” option … The “Name Contains” option allows for a database search without an exact match of the given string. Without the “Name Contains” option selected, the given name in the Sender textbox will

have to exactly match a name inside of the database in order for anything to be presented to the user. The “Specify Receiver” option allows for a user to search wire transaction data for either a given sender name sending money to a given receiver or just a receiver. In such scenario the “name contains” option can also be selected.

Home User Interface

The previous interfaces (Add Wire, Add Note, and Advanced Search) are all options which can be selected from the Home window. The Home window can be seen in the Figure 7 below.

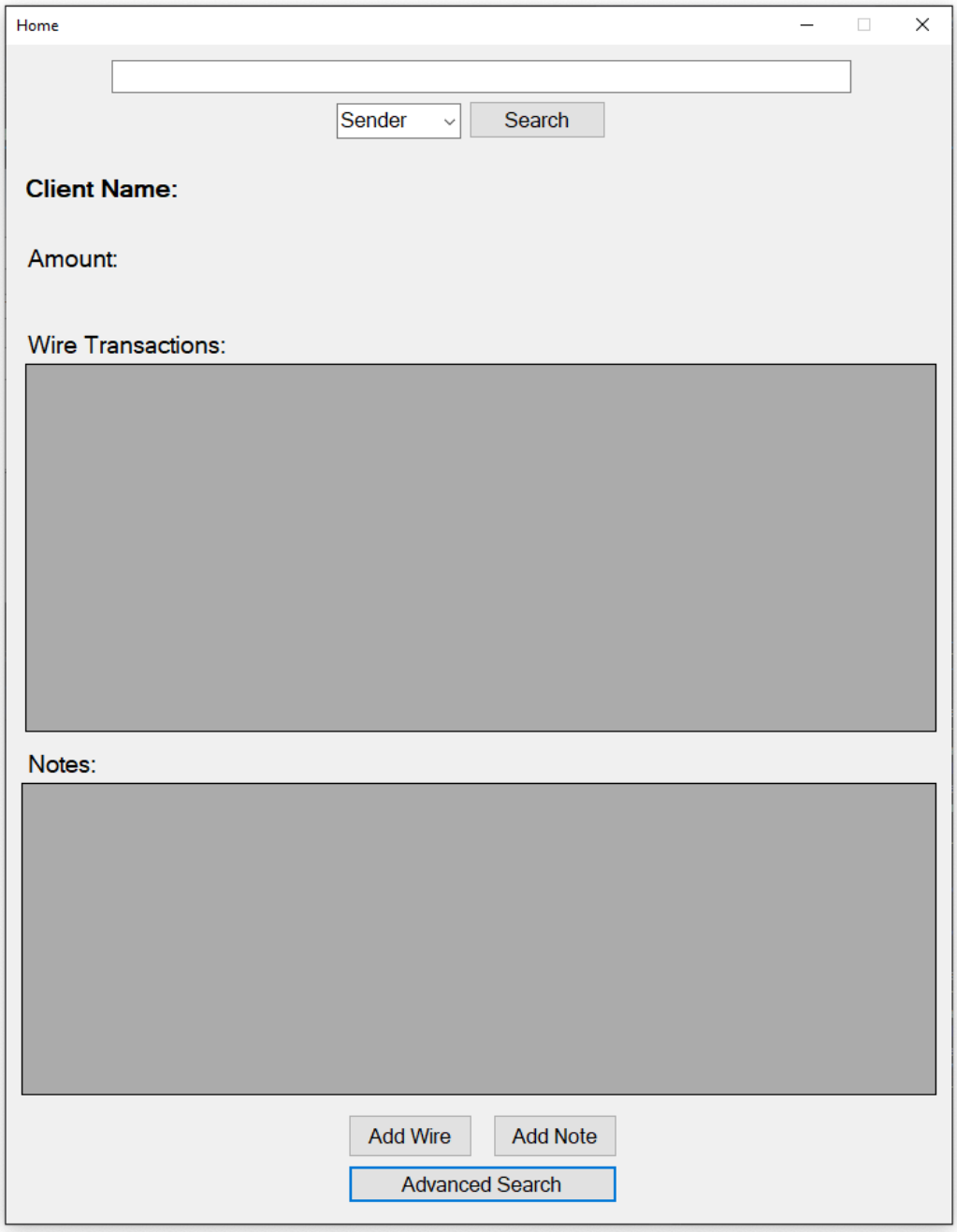

Figure 7: Home Window

The navigation between windows is made possible through the use of Windows Forms with .NET Framework 4.7.2. When a button to open a new window is selected the Home form remains open (as the parent form) and the corresponding child form is opened. Once a child form is terminated, the resources for the child form are purged to preserve memory.

The Home form allows for a quick generic search where either a Sender or Receiver is specified. When the search button is selected, the application queries the Wires table, Notes table, and Flags table. It then displays the corresponding information in the appropriate format. This generic search also identifies if the individual in question has sent or received an accumulated amount of over $3000 within the last 30 days. Figure 8 below shows an example of this.

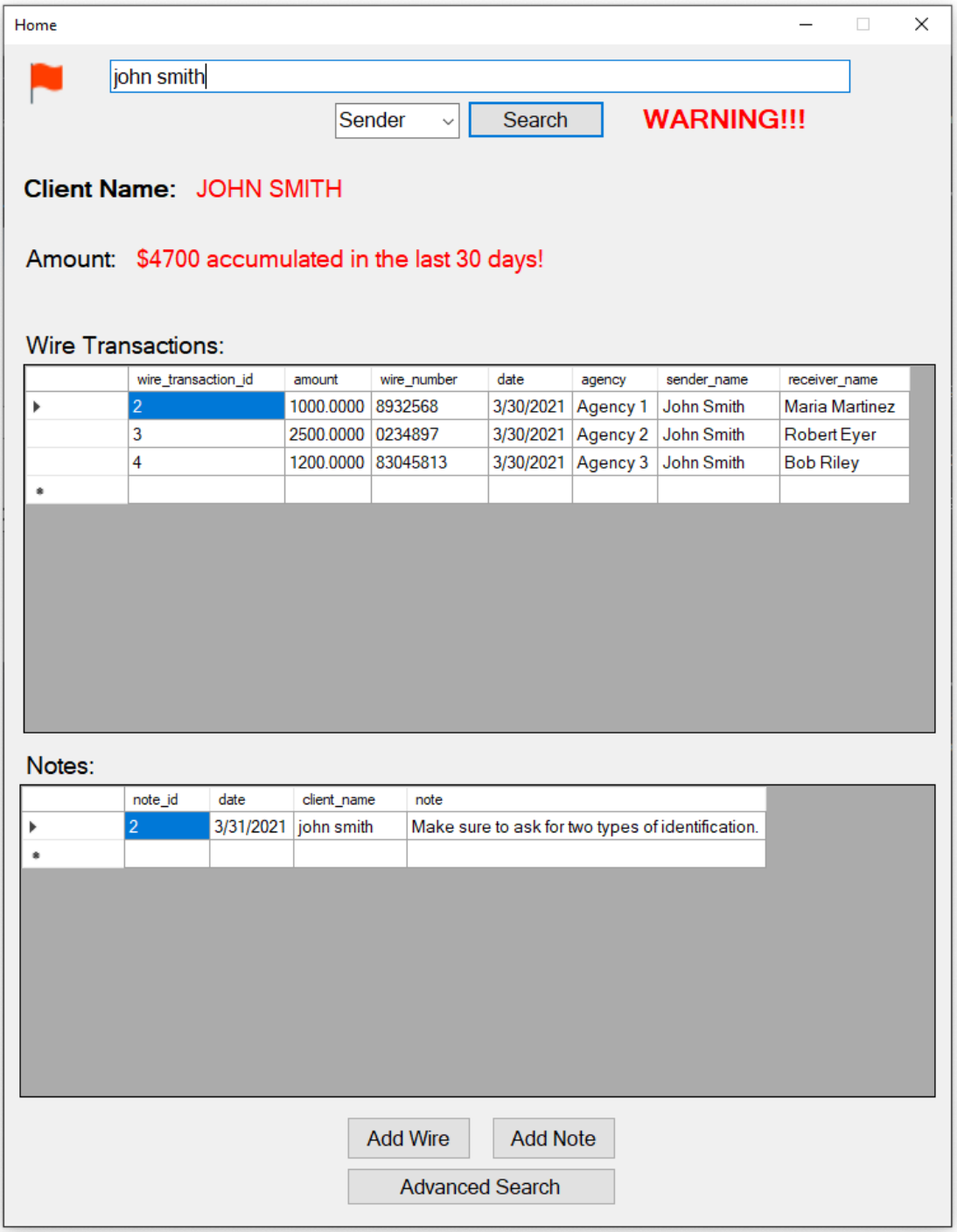

Figure 8: Generic Search Example

As seen in Figure 8 (above), the notes associated with John Smith and the wire transaction data in question are listed. Furthermore, John Smith has been flagged and the

flag can be seen in the upper left-hand corner along with a warning because John has surpassed the $3000 threshold within the last 30 days.

Challenges

While exciting, the development of Wire Manger was no easy task. Multiple hurdles were faced during the development process. Initially the Wire Manger program was a daunting task because of the unfamiliarity with a user interface-based application, the challenge of connecting the database with the application, and designing the user interface. Visual Studio, C#, .NET, and Windows Forms were technologies that I had not previously worked with. Working with these technologies required much experimentation and research. One of the more difficult aspects of the user interface was understanding how to appropriately spin off new forms without having memory issues. It was also necessary to specifically tell the compiler in which order to compile the forms. Navigating Visual Studio to understand how to do this was a challenge.

Appropriately connecting the database to Wire Manager was also a significant challenge. This required significant research and experimentation. Once connected it was also difficult to correctly create SQL objects within the source code and have them communicate with the database. The errors that came from the SQL objects were the most difficult to debug because Visual Studio did not give descriptive enough error message because the errors were mainly coming from the database.

Additionally, the user interface experience required back and forth communication with the client in order to design it in a way that they most preferred. This required active communication and lots of re-working of the user interface experience. The user interface design began with hand drawn sketches and eventually

took its current form. All these challenges led to a stronger understanding in the development process and how to more quickly pick up new skills and technologies.

Future Work

While Wire Manager prototype 1 includes the majority of features required by the client, there are still a few tasks that must be completed and re-worked. Prototype 2 is now in the works and expands to the functionality presented in prototype 1. Prototype 2 will include a more dynamic and modern user interface. The initial design of Prototype 2 can be seen in Figure 9 below.

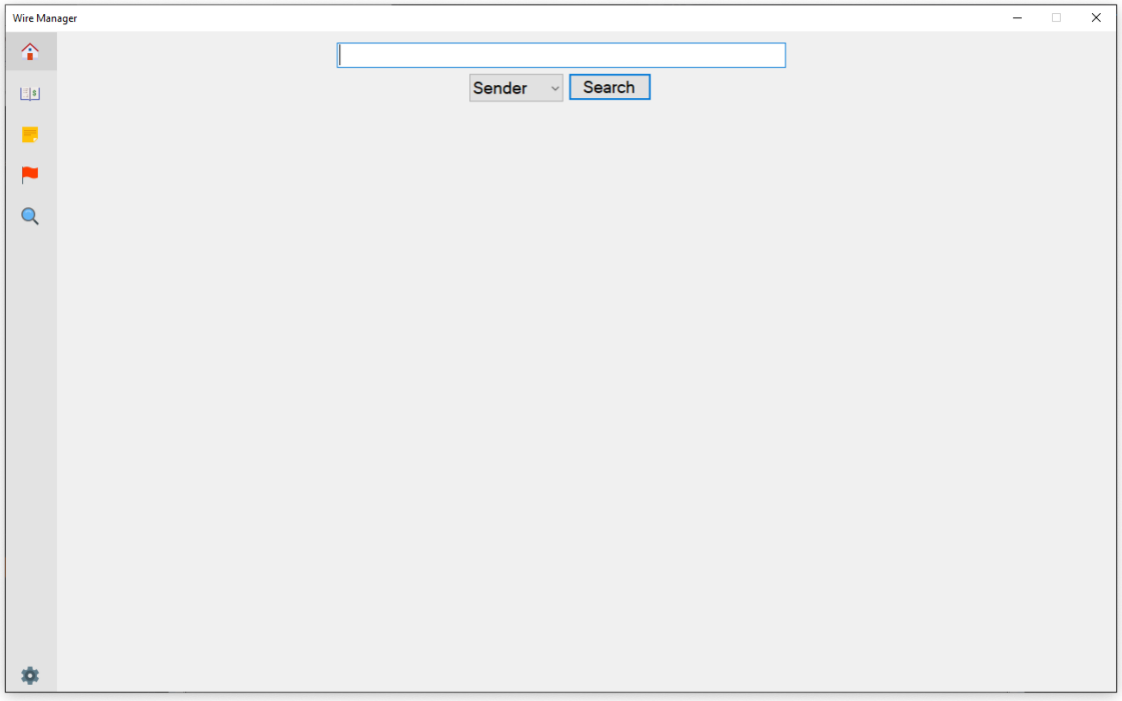

Figure 9: Prototype 2 User Interface

Prototype 2 is expected to be the first official beta release implementation to be used by the client. The functionality for this version will include more advanced search functionality, the ability for an administrator to mutate information within the database through the application interface, the option to export data to a CSV file, and the newly

improved database design. Additionally, with Prototype 2, there will be functionality to access the database from different locations instead of only locally.

Conclusion

Wire Manager allows for higher accountability within small businesses which perform financial wire transactions abroad. This program aids in data entry and data search in an easy-to-use format. While there is still work to be done for the final product to be ready for distribution, Wire Manager includes the foundational tools requested by my client. Wire Manager aims at simplifying the process of meeting the federal and state laws put in place when participating in wiring money abroad. The initial set of tools includes multiple parameter filter searching, background process analysis of potential money laundering cases (based on user-provided data), a flagging system that aids employees in identifying possible criminal activity, and persistent profile note-keeping used to communicate information about potential threats within an organization.

References

The following is a list of websites which were referenced for help with my project:

- https://www.youtube.com/watch?v=6Wwo3Dmd_KY
- https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlconnection?view=netframework-4.8
- https://stackoverflow.com/questions/18113278/populate-a-datagridview-with-sql-query-results
- https://csharp-station.com/Tutorial/AdoDotNet/Lesson06
- https://forums.asp.net/t/1989299.aspx?Get+Single+Value+From+SQL+Database+C+
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/obtaining-a-single-value-from-a-database
- https://www.microsoft.com/design/fluent/#/
- https://docs.devexpress.com/WindowsForms/114555/controls-and-libraries/navigation-controls/navigation-pane
- https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/create-a-multipane-user-interface-with-wf-using-the-designer
- https://docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-run-desktop-app-converter
- https://www.telerik.com/blogs/tips-for-publishing-winforms-wpf-apps-to-microsoft-store
- https://stackoverflow.com/questions/665129/use-of-sqlparameter-in-sql-like-clause-not-working/665157
- https://stackoverflow.com/questions/7510646/like-vs-contains-on-sql-server
- https://www.google.com/search?q=how+to+add+buttons+to+all+data+grid+view+rows+in+c%23&client=firefox-b-1-d&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjt3MKn6eTtAhUGqZ4KHTasBz8Q_AUoAnoECA0QBA&biw=1529&bih=809#imgrc=fhESVyuz-Ra_4M
- https://stackoverflow.com/questions/37026196/datagridview-how-to-set-the-currency-format-for-a-single-column-only
- https://www.essentialsql.com/sql-reference/sql-select-as/
- https://www.zentut.com/sql-tutorial/sql-delete/
- https://stackoverflow.com/questions/3554658/how-to-use-a-class-from-one-c-sharp-project-with-another-c-sharp-project
- https://www.youtube.com/watch?v=V9r-Gp3uNCE
- https://www.youtube.com/watch?v=MQ8t2bHtGlg
- https://www.hostingadvice.com/how-to/best-free-database-hosting/
- https://www.c-sharpcorner.com/UploadFile/c56872/connecting-C-Sharp-application-to-ms-access-database/
- https://www.mikesdotnetting.com/article/280/solved-the-microsoft-ace-oledb-12-0-provider-is-not-registered-on-the-local-machine
- https://answers.microsoft.com/en-us/msoffice/forum/msoffice_install-mso_win10/office-16-click-to-run-extensibility-component-64/e79ee5bd-f119-4808-9bb2-289dd815b76a
- https://docs.microsoft.com/en-us/visualstudio/data-tools/connect-to-data-in-an-access-database-windows-forms?view=vs-2019
- https://social.msdn.microsoft.com/Forums/en-US/19d1ea3b-26be-4ef7-8395-5abc8fcf2024/access-unrecognized-database-format?forum=visualstudiogeneral
- https://social.msdn.microsoft.com/Forums/en-US/1d5c04c7-157f-4955-a14b-41d912d50a64/how-to-fix-error-quotthe-microsoftaceoledb120-provider-is-not-registered-on-the-local?forum=vstsdb
- https://stackoverflow.com/questions/39020305/microsoft-ace-oledb-12-0-not-recognized-in-windows-10
- https://answers.microsoft.com/en-us/msoffice/forum/msoffice_excel-mso_win10-mso_2016/microsoftaceoledb120/7a8d1df5-057b-4d4f-9a50-6d0ff6d85e6d

Appendix A – TableCreationQuery.sql (Initial Database)

```
-- database creation --
create database wire_manager;
use wire_manager;


-- table creation --
create table wires
(
    wire_transaction_id bigint identity(1,1) primary key,
    amount money not null,
    wire_number varchar(100) not null,
    date date not null,
    agency varchar(100) not null,
    sender_name varchar(100) not null,
    receiver_name varchar(100) not null,
);

select * from notes;
create table notes
 (
   note_id bigint identity(1,1) primary key,
   date date not null,
   client_name varchar(100) not null,
   note varchar(1000) not null,
 );
```

```
create table flags
(
   flag_id bigint identity(1,1) primary key,
   client_name varchar(100) not null,
   flag_status bit not null,
);
```

Appendix B – HomeForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Configuration;
using AddNote_Prototype1_WireManager;
using AddWire_Prototype1_WireManager;
using AdvancedSearch_Prototype1_WireManager;


namespace Home_Prototype1_WireManager
{
   public partial class HomeForm : Form
   {
      public HomeForm()
      {
         InitializeComponent();
         comboBox.Text = "Sender"; // sets combo box text to sender
      }

      private void searchButton_Click(object sender, EventArgs e)
      {
         warningLabel.Visible = false; // changes warning visibility to false
         flagPictureBox.Visible = false; // changes flag picture visibility to false
         genericDatabaseSearch();
         checkFlag();
         populateDataGrid();
```

```
    populateWireData();

}

/// <summary>
/// Populates data grid for notes.
/// </summary>
private void populateDataGrid()
{
    SqlParameter param = new SqlParameter("@client_name", nameTextBox.Text);
    using (var connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
    using (SqlCommand cmd = new SqlCommand("select * from notes where client_name = @client_name", connection))
    {
        cmd.Parameters.Add(param); // adds parameter to SQL command


        using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
        using (DataSet dataSet = new DataSet())
        {
            dataAdapter.Fill(dataSet);
            dataGridView.DataSource = dataSet.Tables[0];
            dataGridView.AutoResizeColumns();

        }

        dataGridView.AutoResizeColumns();
    }
}

/// <summary>
/// Populates wire data using generic $3500 accumulated amount and 30-day time period.
/// </summary>
private void populateWireData()
{
    SqlParameter name; // declaration of "name" SqlParameter
    SqlParameter todayDateParam; // today's date parameter
    SqlParameter startDateParam; // start date paramater
    double amount; // declaration of amount variable

    // functionallity for sender
    if (comboBox.Text.Equals("Sender"))
    {
        DateTime today = DateTime.Today; // todays date
        DateTime startDate = DateTime.Today.AddDays(-30); // the date thirty days ago
        name = new SqlParameter("@sender", nameTextBox.Text); // initializes name parameter
        todayDateParam = new SqlParameter("@todayDate", today.ToShortDateString()); // initializes today's date parameter
        startDateParam = new SqlParameter("@startDate", startDate.ToShortDateString()); // initializes start date parameter
```

```
    using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
    using (SqlCommand cmd = new SqlCommand("select * from wires where date between @startDate and @todayDate and sender_name=@sender", connection))
    {
        cmd.Parameters.Add(name); // adds name parameter
        cmd.Parameters.Add(todayDateParam); // adds today's date parameter
        cmd.Parameters.Add(startDateParam); // adds start date parameter
        connection.Open(); // opens connection
        amount = Convert.ToDouble(cmd.ExecuteScalar());

        using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
        using (DataSet dataSet = new DataSet())
        {
            dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query

            wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
            wireDataGridView.AutoResizeColumns(); // auto resizes columns

        }
    }
}

// functionallity for receiver
if (comboBox.Text.Equals("Receiver"))
{
    DateTime today = DateTime.Today; // todays date
    DateTime startDate = DateTime.Today.AddDays(-30); // the date thirty days ago
    name = new SqlParameter("@receiver", nameTextBox.Text); // initializes name parameter
    todayDateParam = new SqlParameter("@todayDate", today.ToShortDateString()); // initializes today's date parameter
    startDateParam = new SqlParameter("@startDate", startDate.ToShortDateString()); // initializes start date parameter
    using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
    using (SqlCommand cmd = new SqlCommand("select * from wires where date between @startDate and @todayDate and receiver_name=@receiver", connection))
    {
        cmd.Parameters.Add(name); // adds name parameter
        cmd.Parameters.Add(todayDateParam); // adds today's date parameter
        cmd.Parameters.Add(startDateParam); // adds start date parameter
        connection.Open(); // opens connection
        amount = Convert.ToDouble(cmd.ExecuteScalar());

        using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
        using (DataSet dataSet = new DataSet())
        {
            dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query

            wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
            wireDataGridView.AutoResizeColumns(); // auto resizes columns

        }
    }
```

```
    }
}

/// <summary>
/// A generic database search which looks to see if a client has surpassed the $3500
/// over a 30-day period.
/// </summary>
private void genericDatabaseSearch()
{
    SqlParameter name; // declaration of "name" SqlParameter
    double amount = 0; // declaration of amount variable

    if (comboBox.Text.Equals("Sender"))
    {
        name = new SqlParameter("@sender", nameTextBox.Text);
        using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
        using (SqlCommand cmd = new SqlCommand("select sum(amount) from wires where sender_name = @sender", connection))
        {
            cmd.Parameters.Add(name); // adds name parameter
            connection.Open(); // opens connection

            try
            {
                amount = Convert.ToDouble(cmd.ExecuteScalar());
            }
            catch (Exception) { } // handles exception where return on database query is null

            if (amount > 3500)
            {
                clientNameLabel.ForeColor = Color.Red; // sets client name label to red
                amountLabel.ForeColor = Color.Red; // sets amount label to red
                warningLabel.Visible = true; // changes warning label visibility to true

                clientNameLabel.Text = nameTextBox.Text.ToUpper(); // sets text to upper case
            }
            else
            {
                clientNameLabel.ForeColor = Color.Green; // sets client name label to red
                amountLabel.ForeColor = Color.Green; // sets amount label to green

                clientNameLabel.Text = nameTextBox.Text.ToUpper(); // sets text to upper case
            }

            clientNameLabel.Visible = true; // make client name label visibile
            amountLabel.Text = "$" + Convert.ToString(amount) + " accumulated in the last 30 days!"; // sets amount label text
            amountLabel.Visible = true; // makes amount label visible
        }
```

```
    }
    else
    {
        name = new SqlParameter("@receiver", nameTextBox.Text);
        using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
        using (SqlCommand cmd = new SqlCommand("select sum(Amount) from wires where receiver_name = @receiver", connection))
        {
            cmd.Parameters.Add(name);
            connection.Open();

            try
            {
                amount = Convert.ToDouble(cmd.ExecuteScalar());
            }
            catch
           (Exception)
            { }

            if (amount > 3500)
            {
                clientNameLabel.ForeColor = Color.Red; // sets client name label to red
                amountLabel.ForeColor = Color.Red; // sets amount label to red
                warningLabel.Visible = true; // changes warning label visibility to true

                clientNameLabel.Text = nameTextBox.Text.ToUpper(); // sets text to upper case
            }
            else
            {
                clientNameLabel.ForeColor = Color.Green; // sets client name label to red
                amountLabel.ForeColor = Color.Green; // sets amount label to green

                clientNameLabel.Text = nameTextBox.Text.ToUpper(); // sets text to upper case
            }

            clientNameLabel.Visible = true; // make client name label visibile
            amountLabel.Text = "$" + Convert.ToString(amount) + " accumulated in the last 30 days!"; // sets amount label text
            amountLabel.Visible = true; // makes amount label visible
        }
    }
}

/// <summary>
/// When button is clicked add note form is displayed.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void addNoteButton_Click(object sender, EventArgs e)
```

```
{
    clearForm();
    using (AddNoteForm form = new AddNoteForm())
    {
        form.ShowDialog();
    }
}

/// <summary>
/// Check if flag exists and if so flag is set to be visibile.
/// </summary>
private void checkFlag()
{
    SqlParameter name = new SqlParameter("@client_name", nameTextBox.Text); // name parameter

    // pinging the database and executing a command
    using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
    using (SqlCommand cmd = new SqlCommand("select client_name from flags where client_name = @client_name", connection))
    {
        string bit = null; // initializes bit integer to 0
        cmd.Parameters.Add(name); // adds paramater to command
        connection.Open(); // opens connection
        bit = (string)cmd.ExecuteScalar();

        if (bit != null)
        {
            flagPictureBox.Visible = true;
        }

    }

}

/// <summary>
/// Loads add wire form.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void addWireButton_Click(object sender, EventArgs e)
{
    clearForm();
    using (AddWireForm form = new AddWireForm())
    {
        form.ShowDialog();
    }
}

/// <summary>
```

```
        /// Function used to clear form.
        /// </summary>
        private void clearForm()
        {
            nameTextBox.Clear();
            clientNameLabel.Visible = false;
            amountLabel.Visible = false;
            warningLabel.Visible = false;
            flagPictureBox.Visible = false;
            wireDataGridView.DataSource = null;
            dataGridView.DataSource = null;
        }

        private void advancedSearchButton_Click(object sender, EventArgs e)
        {
            clearForm();
            using (AdvancedSearchForm form = new AdvancedSearchForm())
            {
                form.ShowDialog();
            }
        }
    }
}
```

Appendix C – AdvancedSearchForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AdvancedSearch_Prototype1_WireManager
{
    public partial class AdvancedSearchForm : Form
    {
        public AdvancedSearchForm()
        {
            InitializeComponent();
        }

        private void searchButton_Click(object sender, EventArgs e)
        {
            // generic search
            if (!nameContainsCheckBox.Checked & !specifyReceiverCheckBox.Checked)
            {
                populateDataGridView_Generic(); // populates data grid view
                findAmount_Generic(); // finds amount
            }
            // name contains search
            if (nameContainsCheckBox.Checked && !specifyReceiverCheckBox.Checked)
            {
                populateDataGridView_Contains(); // populates data grid view
                findAmount_Contains(); // finds amount
            }

            // specify search
            if (!nameContainsCheckBox.Checked && specifyReceiverCheckBox.Checked)
            {
                populateDataGridView_Sepcify();
                findAmount_Specify();
            }

            // specify and contains search
            if (nameContainsCheckBox.Checked && specifyReceiverCheckBox.Checked)
```

```
    {
        populateDataGridView_ContainsAndSpecify();
        findAmount_ContainsAndSpecify();
    }
}

private void specifyReceiverCheckBox_CheckedChanged(object sender, EventArgs e)
{
    // if box is checked set, receiver label and text box to true
    if (specifyReceiverCheckBox.Checked)
    {
        receiverLabel.Visible = true; // changes visibility for receiver label to true
        receiverTextBox.Visible = true; // changes visibility for receiver text box to true
    }

    // if box is unchecked, set receiver label and text box to false
    if (!specifyReceiverCheckBox.Checked)
    {
        receiverLabel.Visible = false; // changes visibility for receiver label to false
        receiverTextBox.Visible = false; // changes visibility for receiver label to true
    }
}

private void populateDataGridView_Generic()
{
    SqlParameter senderNameParam = new SqlParameter("@senderName", senderTextBox.Text); // declares sender name parameter
    SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
    SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

    using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
    using (SqlCommand cmd = new SqlCommand("select * from wires where date between @beginDate and @endDate and sender_name=@senderName", connection))
    {
        cmd.Parameters.Add(senderNameParam); // adds sender name parameter
        cmd.Parameters.Add(beginDateParam); // add start date parameter
        cmd.Parameters.Add(endDateParam); // adds end date parameter
        connection.Open(); // opens conenctions

        using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
        using (DataSet dataSet = new DataSet())
        {
            dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query

            wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
            wireDataGridView.AutoResizeColumns(); // auto resizes columns

        }
    }
}
```

```
        private void findAmount_Generic()
        {
            SqlParameter senderNameParam = new SqlParameter("@senderName", senderTextBox.Text); // declares sender name parameter
            SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
            SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("select sum(amount) from wires where date between @beginDate and @endDate and sender_name=@senderName",
connection))
            {
                cmd.Parameters.Add(senderNameParam); // adds sender name parameter
                cmd.Parameters.Add(beginDateParam); // add start date parameter
                cmd.Parameters.Add(endDateParam); // adds end date parameter
                connection.Open(); // opens conenctions

                double amount = 0; // declares amount value

                try
                {
                    amount = Convert.ToDouble(cmd.ExecuteScalar());
                }
                catch (Exception) { } // handles exception where return on database query is null

                amountLabel.Text = "$" + Convert.ToString(amount); // converts amount to string
                amountLabel.ForeColor = Color.Blue;
                amountLabel.Visible = true; // sets amount visibility to true
            }
        }

        private void populateDataGridView_Contains()
        {
            SqlParameter senderNameParam = new SqlParameter("@senderName","%" + senderTextBox.Text + "%"); // declares sender name parameter
            SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
            SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("select * from wires where date between @beginDate and @endDate and sender_name like @senderName", connection))
            {
                cmd.Parameters.Add(senderNameParam); // adds sender name parameter
                cmd.Parameters.Add(beginDateParam); // add start date parameter
                cmd.Parameters.Add(endDateParam); // adds end date parameter
                connection.Open(); // opens conenctions

                using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
                using (DataSet dataSet = new DataSet())
                {
                    dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query
```

```
                wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
                wireDataGridView.AutoResizeColumns(); // auto resizes columns

            }
        }
    }

    private void findAmount_Contains()
    {
        SqlParameter senderNameParam = new SqlParameter("@senderName", "%" + senderTextBox.Text + "%"); // declares sender name parameter
        SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
        SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

        using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
        using (SqlCommand cmd = new SqlCommand("select sum(amount) from wires where date between @beginDate and @endDate and sender_name like @senderName",
connection))
        {
            cmd.Parameters.Add(senderNameParam); // adds sender name parameter
            cmd.Parameters.Add(beginDateParam); // add start date parameter
            cmd.Parameters.Add(endDateParam); // adds end date parameter
            connection.Open(); // opens conenctions

            double amount = 0; // declares amount value

            try
            {
                amount = Convert.ToDouble(cmd.ExecuteScalar());
            }
            catch (Exception) { } // handles exception where return on database query is null

            amountLabel.Text = "$" + Convert.ToString(amount); // converts amount to string
            amountLabel.ForeColor = Color.Blue;
            amountLabel.Visible = true; // sets amount visibility to true
        }
    }

    private void populateDataGridView_Sepcify()
    {
        SqlParameter senderNameParam = new SqlParameter("@senderName", senderTextBox.Text); // declares sender name parameter
        SqlParameter receiverNameParam = new SqlParameter("@receiverName", receiverTextBox.Text); // declares receiver name parameter
        SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
        SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

        using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
        using (SqlCommand cmd = new SqlCommand("select * from wires where date between @beginDate and @endDate and sender_name=@senderName and
receiver_name=@receiverName", connection))
        {
```

```
            cmd.Parameters.Add(senderNameParam); // adds sender name parameter
            cmd.Parameters.Add(receiverNameParam); // adds receiver name parameter
            cmd.Parameters.Add(beginDateParam); // add start date parameter
            cmd.Parameters.Add(endDateParam); // adds end date parameter
            connection.Open(); // opens conenctions

            using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
            using (DataSet dataSet = new DataSet())
            {
                dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query

                wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
                wireDataGridView.AutoResizeColumns(); // auto resizes columns

            }
        }
    }

    private void findAmount_Specify()
    {
        SqlParameter senderNameParam = new SqlParameter("@senderName", senderTextBox.Text); // declares sender name parameter
        SqlParameter receiverNameParam = new SqlParameter("@receiverName", receiverTextBox.Text); // declares receiver name parameter
        SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
        SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

        using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
        using (SqlCommand cmd = new SqlCommand("select sum(amount) from wires where date between @beginDate and @endDate and sender_name=@senderName and
receiver_name=@receiverName", connection))
        {
            cmd.Parameters.Add(senderNameParam); // adds sender name parameter
            cmd.Parameters.Add(receiverNameParam); // adds receiver name parameter
            cmd.Parameters.Add(beginDateParam); // add start date parameter
            cmd.Parameters.Add(endDateParam); // adds end date parameter
            connection.Open(); // opens conenctions

            double amount = 0; // declares amount value

            try
            {
                amount = Convert.ToDouble(cmd.ExecuteScalar());
            }
            catch (Exception) { } // handles exception where return on database query is null

            amountLabel.Text = "$" + Convert.ToString(amount); // converts amount to string
            amountLabel.ForeColor = Color.Blue;
            amountLabel.Visible = true; // sets amount visibility to true
        }
    }
```

```
        private void populateDataGridView_ContainsAndSpecify()
        {
            SqlParameter senderNameParam = new SqlParameter("@senderName", "%" + senderTextBox.Text + "%"); // declares sender name parameter
            SqlParameter receiverNameParam = new SqlParameter("@receiverName", "%" + receiverTextBox.Text + "%"); // declares receiver name parameter
            SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
            SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("select * from wires where date between @beginDate and @endDate and sender_name like @senderName and
receiver_name like @receiverName", connection))
            {
                cmd.Parameters.Add(senderNameParam); // adds sender name parameter
                cmd.Parameters.Add(receiverNameParam); // adds receiver name parameter
                cmd.Parameters.Add(beginDateParam); // add start date parameter
                cmd.Parameters.Add(endDateParam); // adds end date parameter
                connection.Open(); // opens conenctions

                using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))
                using (DataSet dataSet = new DataSet())
                {
                    dataAdapter.Fill(dataSet); // fills dataAdapter with SQL query

                    wireDataGridView.DataSource = dataSet.Tables[0]; // populates data grid view
                    wireDataGridView.AutoResizeColumns(); // auto resizes columns

                }
            }
        }

        private void findAmount_ContainsAndSpecify()
        {
            SqlParameter senderNameParam = new SqlParameter("@senderName", "%" + senderTextBox.Text + "%"); // declares sender name parameter
            SqlParameter receiverNameParam = new SqlParameter("@receiverName", "%" + receiverTextBox.Text + "%"); // declares receiver name parameter
            SqlParameter beginDateParam = new SqlParameter("@beginDate", startDateTimePicker.Value.ToShortDateString()); // declares begin date param
            SqlParameter endDateParam = new SqlParameter("@endDate", endDateTimePicker.Value.ToShortDateString()); // declares end date param

            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("select sum(amount) from wires where date between @beginDate and @endDate and sender_name like @senderName
and receiver_name like @receiverName", connection))
            {
                cmd.Parameters.Add(senderNameParam); // adds sender name parameter
                cmd.Parameters.Add(receiverNameParam); // adds receiver name parameter
                cmd.Parameters.Add(beginDateParam); // add start date parameter
                cmd.Parameters.Add(endDateParam); // adds end date parameter
                connection.Open(); // opens conenctions

                double amount = 0; // declares amount value
```

```
            try
            {
                amount = Convert.ToDouble(cmd.ExecuteScalar());
            }
            catch (Exception) { } // handles exception where return on database query is null

            amountLabel.Text = "$" + Convert.ToString(amount); // converts amount to string
            amountLabel.ForeColor = Color.Blue;
            amountLabel.Visible = true; // sets amount visibility to true
        }
    }
  }
}
```

Appendix D – AddWireForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AddWire_Prototype1_WireManager
{
    public partial class AddWireForm : Form
    {
        public AddWireForm()
        {
            InitializeComponent();
            agencyComboBox.Text = "SILMEX"; // sets text for agency label
        }

        private void addButton_Click(object sender, EventArgs e)
        {
            addWireToDB();

            // Clears corresponding text box fields.
            senderTextBox.Clear();
            receiverTextBox.Clear();
            amountTextBox.Clear();
            wireNumTextBox.Clear();

        }

        private void addWireToDB()
        {
            CultureInfo cultureInfo = new CultureInfo("en-US"); // creates english CultureInfo object

            SqlParameter amountParam = new SqlParameter("@amount", Convert.ToDecimal(amountTextBox.Text, cultureInfo)); // amount SQL parameter
            SqlParameter senderParam = new SqlParameter("@sender_name", senderTextBox.Text); // sender_name SQL parameter
            SqlParameter receiverParam = new SqlParameter("@receiver_name", receiverTextBox.Text); // receiver_name SQL parameter
            SqlParameter wireNumParam = new SqlParameter("@wire_number", wireNumTextBox.Text); // wire_number SQL parameter
            SqlParameter agencyIDParam = new SqlParameter("@agency", agencyComboBox.Text); // agency ID SQL parameter
            SqlParameter dateParam = new SqlParameter("@date", dateTimePicker.Value.ToShortDateString()); // date SQL parameter
```

```
            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("insert into wires(amount, wire_number, date, agency, sender_name, receiver_name) values (@amount, @wire_number,
@date, @agency, @sender_name, @receiver_name)", connection))
            {
                cmd.Parameters.Add(amountParam); // adds amount parameter to sql command
                cmd.Parameters.Add(senderParam); // adds sender parameter to sql command
                cmd.Parameters.Add(receiverParam); // adds receiver parameter to sql command
                cmd.Parameters.Add(wireNumParam); // adds wire number parameter to sql command
                cmd.Parameters.Add(agencyIDParam); // adds agency id parameter to sql command
                cmd.Parameters.Add(dateParam); // adds date parameter to sql command

                connection.Open(); // opens SQL connection
                cmd.ExecuteNonQuery(); // executes query
            }

            successLabel.Text = senderTextBox.Text.ToUpper() + " added successfully!"; // success message
            successLabel.Visible = true;
        }


    }
}
```

Appendix E – AddNoteForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AddNote_Prototype1_WireManager
{
    public partial class AddNoteForm : Form
    {
        public AddNoteForm()
        {
            InitializeComponent();
        }

        private void addNoteToDB()
        {
            SqlParameter dateParam = new SqlParameter("@date", dateTimePicker.Value.ToShortDateString());
            SqlParameter nameParam = new SqlParameter("@client_name", nameTextBox.Text);
            SqlParameter noteParam = new SqlParameter("@note", noteTextBox.Text);


            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("insert into notes(date, client_name, note) values (@date, @client_name, @note)", connection))
            {
                cmd.Parameters.Add(dateParam);
                cmd.Parameters.Add(nameParam);
                cmd.Parameters.Add(noteParam);

                connection.Open();
                cmd.ExecuteNonQuery();
            }
        }

        private void flagLogic()
        {
            if (flagCheckBox.Checked)
```

```
        {
            SqlParameter name = new SqlParameter("@client_name", nameTextBox.Text);
            SqlParameter newName = new SqlParameter("@client_name", nameTextBox.Text);
            SqlParameter flag = new SqlParameter("@flag_status", 1);




            using (SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DB"].ConnectionString))
            using (SqlCommand cmd = new SqlCommand("select client_name from flags where client_name = @client_name", connection))
            {
                string dbName = null;
                cmd.Parameters.Add(name);
                connection.Open();
                dbName = (string)cmd.ExecuteScalar();

                if (dbName == null)
                {
                    using (SqlCommand flagCmd = new SqlCommand("insert into flags(client_name, flag_status) values (@client_name, @flag_status)", connection))
                    {

                        flagCmd.Parameters.Add(newName);
                        flagCmd.Parameters.Add(flag);



                        flagCmd.ExecuteNonQuery();
                    }
                }


            }
        }


    private void addNoteButton_Click(object sender, EventArgs e)
    {
        addNoteToDB();
        flagLogic();
        this.Dispose();
    }

}
```