What's It Worth?

Senior Project System Manual

CONTENTS

OVERVIEW AND BACKGROUND

In this document, I will be describing the initial research, analysis, and design of the “*What’s It Worth*?” desktop application that is the basis of my senior project. The primary goal of "*What's It Worth?*" is to develop an application that allows users to calculate the market price of the materials in their jewelry. To fully understand the nature of the project, I will explain the problem that needs to be addressed, my motivation to create a solution that resolves this problem, the background on previous solutions and the industry as a whole, and the solution itself, all within this section.

Problem/Issue

A lot of people have jewelry that they know nothing about. Whether it was inherited, gifted to them, or purchased, most people do not know how much their item is actually worth. This often leads to people not truly understanding/appreciating their pieces of fine jewelry or getting taken advantage of when they need to buy or sell it.

Motivation

The motivation behind "What's It Worth?" is rooted in the need to provide a practical solution for individuals looking to determine the market value of their jewelry. In today's jewelry market, the value of precious metals and gemstones can fluctuate significantly. This constant change in pricing can be used by some immoral businesses to misinform the general population to increase their profit margins. As someone who has worked in this industry for many years, I often get customers coming into my store and asking how much their jewelry is worth. Frequently, they are highly misinformed about their possessions as the first few businesses that they went to evaluate their items severely undervalued the pieces of jewelry to purchase them for dirt cheap. On the other end, I have also encountered uninformed people who were taken advantage of by big-name corporations that grossly overprice pieces of fine jewelry as they target individuals who do not know the industry standard of pricing and assume they will be receiving a fair deal. To counteract this, I am creating an application that aims

to empower users with the ability to calculate the current market price of their jewelry accurately and view changes to that price over time, which can help them make informed decisions about their valuable possessions that could be done at home and not require that much effort.

Background

The problem of people not knowing the value of their jewelry is a longstanding one. Historically, finding out the value of your jewelry requires an appraisal that required taking the item to a professional gemologist or jeweler which was time-consuming and often required paying a fee for the appraisal. This was a problem for a lot of those people as they did not need paperwork and a detailed description about the piece of jewelry itself, but just a price estimation that can be generated based on the most current material prices. Another issue with this method is not knowing this estimated value prior to purchase. For example, if an average person walks into a jewelry store to purchase an engagement ring, they will have no frame of reference to if the price of the ring is justifiable based on the contents of the ring. So, the only way to get the true value of the ring would be to take the ring to an appraiser after it is purchased, but by that time, the damage would have already been done. Luckily, the prices of these materials have been made accessible for the common folk through the use of the internet, but there is not a cost-free “one stop shop” solution that produces an estimation for a piece of jewelry in its entirety.

In 2005, *WashingtonDiamond.com* attempted to address the issue of diamond pricing through the use of their “Diamond Price Calculator”. This online tool allows the user to input the details about their diamond and calculates the estimated value of that stone based on current pricing trends. As seen in Figure 1 below, the user inputs the shape, carat, color, and clarity of the diamond and interacts with “Calculate” button to automatically calculate an estimation of the stone.

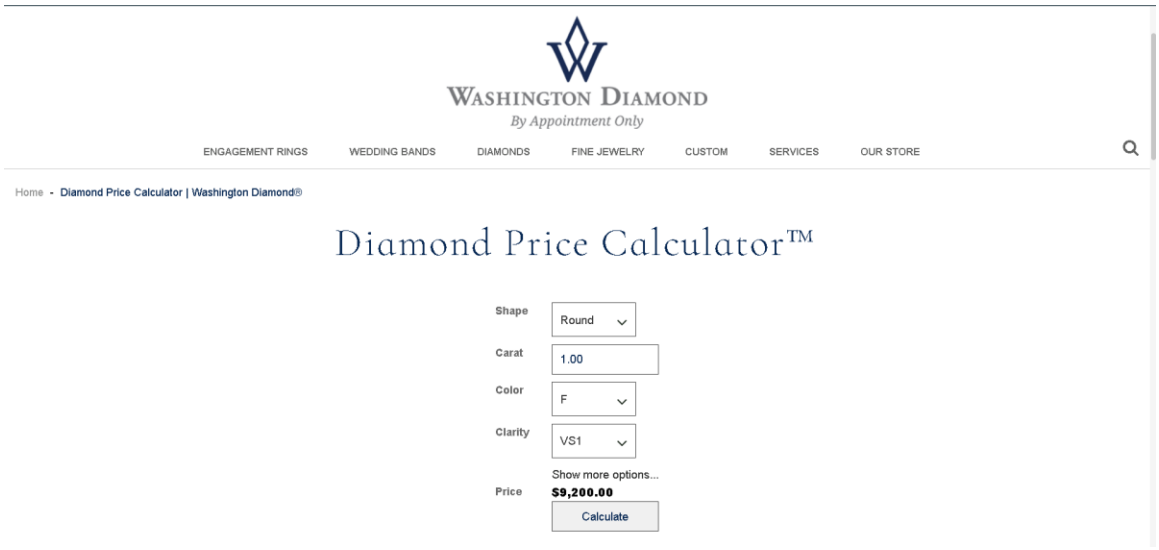


Figure 1: washingtondiamond.com/diamond-price-calculator

However, this solution has its limitations as firstly, it relies on general pricing data rather than a global standard for pricing such as Rapaport or Idex (discussed below). Another issue with this solution is that it is only a calculation for loose diamonds, meaning that the cost of the base material is not considered or calculated. Lastly, there is no option to save the price and information of the stone within the website where the user can view the trend of pricing for their diamond.

In terms of global pricing for diamonds, the industry standard is the “Rapaport Diamond Report” which is a weekly report of diamonds based on the New York cash asking price pictured in Figure 2 down below.

בס"ד

RAPAPORT DIAMOND REPORT

Tel: 877-987-3400 ◆ www.RAPAPORT.com ◆ info@RAPAPORT.com R

April 12, 2024 : Volume 47 No. 15: NEW YORK HIGH CASH ASKING PRICES : Page 2

Round Brilliant Cut Natural Diamonds, GIA Grading Standards per "Rapaport Specification A3" in hundreds of US$ per carat.

We grade SI3 as a split SI2/I1 clarity. Price changes are in **Bold**, higher prices underlined, lower prices in italics.

Rapaport welcomes price information and comments. Please email us at prices@Diamonds.Net.

0.95-0.99 may trade at 5% to 10% premiums over 0.90 — 1.25 to 1.49 Ct. may trade at 5% to 10% premiums over 4/4 prices.

	RAPAPORT : (.90 - .99 CT.) : 04/12/24											ROUNDS	RAPAPORT : (1.00 - 1.49 CT.) : 04/12/24											
	IF	VVS1	VVS2	VS1	VS2	SI1	SI2	SI3	I1	I2	I3		IF	VVS1	VVS2	VS1	VS2	SI1	SI2	SI3	I1	I2	I3	
D	122	108	89	71	60	52	42	35	32	22	15	D	***202***	***153***	129	108	87	***68***	***55***	49	41	25	16	D
E	109	99	83	65	56	48	38	33	30	21	14	E	***150***	***132***	116	99	80	***63***	***51***	46	38	24	15	E
F	100	91	77	61	53	45	35	32	28	20	13	F	***132***	***119***	107	92	75	***59***	***48***	43	36	23	14	F
G	***82***	76	65	56	49	41	33	30	27	19	12	G	***103***	***97***	***90***	82	70	***55***	***45***	40	34	22	13	G
H	***67***	***62***	***56***	51	46	38	31	28	26	18	12	H	***82***	***78***	***74***	70	64	***50***	***42***	37	32	21	13	H
I	***58***	***54***	***49***	***45***	41	34	29	26	24	17	11	I	***68***	***64***	***61***	58	52	45	38	35	30	20	12	I
J	***49***	***46***	***42***	***39***	***35***	31	27	24	21	16	10	J	***57***	***53***	***49***	***46***	43	38	34	32	28	19	12	J
K	***42***	***39***	***36***	***33***	***30***	28	25	21	19	15	9	K	***48***	***45***	***42***	***39***	37	33	30	28	26	18	11	K
L	***35***	***33***	***30***	28	26	24	22	20	18	13	8	L	***40***	***37***	***35***	33	31	28	26	24	23	17	10	L
M	***30***	***28***	***26***	***24***	23	21	19	17	15	11	7	M	***34***	***32***	***31***	30	28	25	23	21	20	16	10	M

Figure 2: Rapaport Diamond Report for Round cut diamonds on April 12, 2024

There is a chart for each size (Carat weight) range, which is broken down further by the color and clarity of the stone. The numbers shown are the average prices divided by 100. Although these reports are very detailed, they are mainly used by jewelry retailers and licensed appraisers as they are somewhat complicated/tedious to decipher and access to them is expensive as a 1-year membership can cost as much as $310 a year.

However, although Rapaport is trusted for an average of the market for the month or week, it does not show the diamonds currently on the market. On top of this, Rapaport does not have an API for quick data retrieval and updating. This is why I have decided to use IDEX for my application. IDEX is a marketplace and diamond market tracker that wholesalers use for loose diamonds. IDEX also has an API which scans the current diamonds on the market with the specified requirements and gets the highest, lowest, and average prices on the market. Although a basic membership is free, a business license is required to create an account.

For precious metals, the pricing is simpler than that of diamonds. The most well-known and trusted source for a standardized cost for precious metals is *Kitco.com* which is a precious metals news source and live spot price analyzer based on the New York ask price which is shown in Figure 3 down below. *Kitco.com* is unique in that it not only shows the most accurate day price but displays different graphs and charts that show the history and trends of the price. Lastly, it is especially convenient as it not only displays the gold and silver prices, but platinum and palladium as well, which have become more popular in recent years.

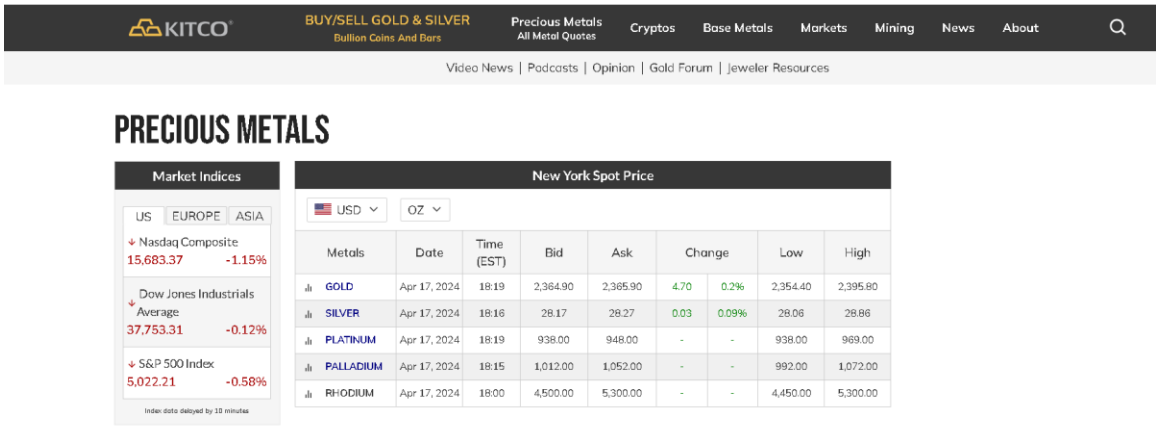


Figure 3: kitco.com/price/precious-metals

Although *Kitco.com* is very accurate, it is not easily usable in the calculation of most jewelry. The price is recorded as a price per troy ounce which is different and more obscure than the commonly used price per ounce. On top of this, the standard of weighing precious metals in the common retail environments is the gram. Another complication is that the price is listed based on complete purity (Example: 24 Karat gold), rather than the more common purities seen in fine jewelry (Example: 10 or 14 Karat gold). Although these prices can be calculated by the individual, it is tedious for the average person to have to recalculate the price using all these factors every day.

My personal background starts with the creation of my family's jewelry store. Adorn Jewelry is a jewelry store that was opened in 2009 that was founded by my father and his 2 brothers at Ingram Park Mall in San Antonio, TX. Only a year later, they opened their second location within Rolling Oaks Mall. Even as a young boy, I was always around my dad and learned what it takes to run a jewelry business. I officially started working at the store during the summers when I turned 15, and by the time I turned 18, I was made the primary manager at one of the stores. Seeking to deepen my expertise, I pursued a certification with the Gemological Institute of America which enabled me to identify and appraise luxury goods properly. This background fuels my passion for developing a solution that will not only benefit my family business and our customers, but jewelry businesses globally. I believe that with my comprehensive understanding of the industry and this opportunity that I have been given, I am committed to addressing the challenges of the problem at hand.

Solution

The basis of this project lies within the realm of jewelry valuation, an area where the value of jewelry is tied to the prices of precious materials which can change rapidly. The problem arises when consumers are taken advantage of for often lacking access to up-to-date market information and the convenient tools necessary to determine the value of their jewelry themselves. To address this issue, it is essential to create a tool that can provide real-time valuations, considering the current market conditions, while still being easy to understand and access.

My solution, the "What's It Worth?" desktop application, will allow users to input detailed information about their jewelry, whether it is just a precious metal or a combination of both metal and diamonds. If it is a combination of both precious metals and diamonds, the detailed information would include jewelry description (Ex: Ring), precious metal type (Ex: Gold), precious metal purity (Ex: 14 Karat), total weight of the piece of fine jewelry (Ex: 8.2 Grams), and a photo of the item along with diamond characteristics for each diamond such as diamond weight (Ex: 1.02 Carat), diamond shape (Ex: Pear), diamond color (Ex: F), and diamond clarity (Ex: VS1). The application will then use this data, along with real-time pricing information from Kitko.com and IDEX, to calculate the current market value of the item and store it within a database to allow the user to view changes over time.

Key features of the application will include (Discussed in detail in next section):

- *User account management (create, edit, delete)*
- *Jewelry item management (add, delete, compare)*
- *Detailed jewelry item information input fields*
- *Reports on the progress of the market value of jewelry items*
- *Live pricing data integration for precious metals and gemstones*
- *Valuation calculation engine to determine item worth*
- *Informational resources and links related to jewelry and family store*
- *Secure data storage and user privacy measures*

By providing this comprehensive and easy-to-use tool, "What's It Worth?" will enable users to better understand the value of their jewelry. This can help them make more informed decisions when purchasing, selling, insuring, or passing down their valuable possessions.

Benefits

Although the "What's It Worth?" application can benefit a wide range of users, it offers the same easy-to-use interface for all of them. Some of the different users that could directly benefit from the application are:

- *Individual Jewelry Owners:* People who have inherited, been gifted, or purchased jewelry can use the application to determine the current market value of their items. This empowers them to make informed decisions about selling, insuring, or passing down their jewelry.

- *Potential Purchasers:* People who are looking to purchase a piece of jewelry can use the application to determine what they are able to purchase based on their price range. This allows them to make an informed purchase at a fair price.
- *Previous Metal Investors:* People who continually purchase jewelry as an investment can use the application to track the value of their jewelry over time, helping them make wise investment decisions and better understand the market.
- *Retail Workers:* Employees at shops that offer the purchasing of precious metals and diamonds can leverage the application to provide more accurate appraisals for their customers, building trust and reputation.
- *Local Companies:* Smaller jewelry stores and pawn shops can benefit from the application adding the items they have purchased to view the cost of all their products as a whole to have an accurate valuation of the merchandise.

By addressing the problem of jewelry undervaluation, "What's It Worth?" can help users be more informed about the financial aspect of their items and serves as a helpful tool to keep track of their high-end luxury goods.

ANALYSIS

Within this section, I will go over the analysis, requirements and key functionalities that will allow the program to fulfill the solution stated in the previous section. Within the analysis subsection, I will describe how I used use cases I have previously created to assist me in the solution approach. In the requirements subsection, I will include the hardware and software requirements for the user and developer. In the functionality's subsection, I describe all the necessary functionalities my project will have. To fully break down and analyze the solution before designing, I looked at the necessary functions that my program must have, along with a system in which it is easy for the user to navigate and understand.

Use Cases

My approach to the problem involved creating use cases to allow me to conceptualize what the user needs to do and how that use case will be fulfilled. This then led me to create the functionalities list necessary for the development of the project. The program will need to handle many different interactions with the user. Although there were many use cases, I am going to highlight the main operations that the user would need to do as these gave me the most functionalities and requirements. Users will be able to:

- *Log in:* If a user has already previously created an account, their email and password would have already been stored in the system. So, the registered email and the password they have chosen previously will allow them to access their account and all the functionalities the program has to offer.
- *Create new account:* In the case of a new user, there will be an option to create a new account. If chosen, the user would need to input their name, email (which will be used as a username when they log in), and a password (which will have to be input twice to verify that is the password that they want). Upon doing this, the user will be redirected back to the login page where they can log in and access their account.
- *View list of items:* From the homepage, the user will have the option to click a button that will take them to a list of items that they have added to

their account. From this page, the user will be able to view the total price of all their items, along with an option to select a specific item which will pull up more information about that item. The user will also have the option to filter/search through their items based on price or material.

- *Add new item:* If this button is clicked from within the homepage, the system will redirect them to a page that will allow the user to input information specific to that item. Once complete, the user will press the save button in which a price will automatically be generated, and the item will be saved to the users list of items to be viewed later.
- *Delete item:* From within the list of items page, the user can select a specific item and have the option to delete it by pressing a button.
- *Compare item:* From within the list of items page, the user can select a specific item and have the option to compare it to another item within the list by pressing a button.
- *Generate total report:* If this button is clicked from within the homepage, the system will save a file that shows the difference in the total price of all the users in that moment from what it was when the user initially added the item.

Although the methods used for solving these use cases differed from this original hypothesis, all of these use cases were solved and enhanced for a better user experience.

Functionalities

This section details the requirements of “What’s It Worth?”. The details listed below will include the hardware and software requirements for the user and the developer. Currently, the requirements for both the user and developer will be the exact same as the only way to access the database is to use Visual Studio which is a developer tool.

The program requires a computer system with the following minimum specifications to use:

Hardware

- Memory: 4 GB RAM
- Storage: 4GB available storage

- Display Resolution: 1280x720 or higher
- Internet Connection
- .NET Framework

Software

- Operating System: Windows 10/11
- Database: Microsoft SQL Server
- Visual Studio 2022

Functionalities

For “What’s It Worth?” to become a user-friendly and interactive interface, it will need multiple functionalities to be able to satisfy the use cases above and become an easily accessible application. To achieve this, the following major functional requirements have been identified below, including what the users are able to do within the system and what the system itself needs to be able to do to function properly. For the system to be fully operational, all these functionalities need to be integrated with one another as they will be accessing the same data within the database. These functions listed are the foundation for the application as I would like there be constant updates adding new functionalities. As this project is based on a solution for the everyday person, the only role will be the user themselves. Every user will be able to do the following actions and activities:

- Create an account
    - *Input User_Name, User_Email, and User_Pass*
- Edit an account
    - *Edit User_Name, User_Email, and User_Pass*
- Delete an account
- Sign in to account
- Logout of account
- View list of items
    - *View total price for all items*
    - *Select specific item*
        - *View latest price of an item*
        - *View item information*
- Compare items
- Add jewelry item

    - *Input Metal_Type, Metal_Purity, and Total_Weight*
    - *Add Diamond*
        - Input *Diamond_Carat, Diamond_Clarity, Diamond_Color, and Diamond_Cut*
    - *Input Photo*
    - *Input Description*
- Delete jewelry item
- Generate total report
- View Install and Project Information
    - Select link to market diamond price API
    - Select link to stock market gold price
    - Select link to family store

On top of these functionalities that the system needs to offer the user, there are system functionalities that the “What’s It Worth?” system needs to perform either automatically or when an event is triggered that the user might not even need to see. In general, these are the functionalities that the system needs to do:

- *Store user login and account information in database*
- *Encryption of sensitive data*
- *Store jewelry item information in database*
- *Generate unique User_ID for each user*
- *Generate unique Item_ID for each item*
- *Calculate price for each item of jewelry*
- *Update metal prices at 12AM UTC everyday*
- *Update price of user items on login of user*
- *Calculate summed price of all items for each user*

DESIGN

In this section, I will be going over the initial design in a more technical sense along will the tools/environment that I will be using to develop my application. I will be implementing techniques and methods that I have learned throughout my education here in St. Mary's, along with couple of solutions that I have researched on my own, to try to integrate them into my application. For example, I will be taking what I learned from my "Files and Database" class with Dr. Barsoum for the general framework of the application, along with the techniques necessary to make a great user experience from my "User Interface Design" class with Dr. Aktunc, and lastly, use what I learned from the "Intro To System Analysis and Design" class that I took with Dr. Fink, to design the initial protype and ensure that there is a plan.

Environments

Within this environment subsection, I will be describing what tools and environments made the development of the "What's It Worth?" desktop application possible. I will also explain why these tools are the best option when it comes to this type of application. Although I will be using a lot of tools that I have been accustomed to previously, I will also integrate some new elements that I believe will take my application to the next level.

Tools (Design)

- *Mockflow*
- *Draw.io*
- *GanttProject*

Tools (Development)

- *Microsoft Visual Studio 2022*
- *Visual Studio Code*
- *.NET Windows Forms*
- *SQL Server Management Studio*
- *GitHub Pages*
- *IDEX API*

- *NuGet Packages*

Languages

- *C#*
- *SQL*
- *XML*
- *CSS*
- *HTML*

Platform

- *Hardware: Desktop*
- *Operating System: Windows 10/11*

The first thing first set of tools I will need to use are for the designing and prototyping of this project. *Mockflow* is a very easy to use website that allows for the creation of detailed mockups and wireframes that will allow me to get input about the design of the pages prior to developing them. I have used *Mockflow* in my classes previously to make wireframes for websites and mobile applications. Next is *Draw.io*, which is an online flowchart and diagram maker. I have used *Draw.io* in multiple classes to create ERD's, Activity Diagrams, and many other structures/diagrams. *GanttProject* is a tool in which Gantt charts can be developed.

The next set of tools will be the development tools which are necessary for the actual creation and deployment of the application. I will be using *Microsoft Visual Studio 22* as my primary IDE where I will write my code and link my database. I have used *Microsoft Visual Studio 22* in previous classes and found that it integrates seamlessly with *Windows Forms* and *SQL Server Management Studio*. For the actual interface, I will be using .NET *Windows Forms* to create the screens that the user will see and interact with. Within *Microsoft Visual Studio 22*, I have found that there are many templates for *Windows Forms* that are already integrated with the IDE so creating elements and coding them to do certain things will be more efficient. Also within Visual studio, I am using NuGet packages for a lot of the plugins that I have such as ReportViewer. For the database, I will be using *SQL Server Management Studio* as I have experience with creating complex data structures and linking them to my IDE already, and it can easily export its data to XML which makes transfer of data easier. Lastly, I

will be deploying my resources page of my application through *GitHub Pages*. I will be using the *IDEX* which integrates with C# retrieve the prices of the diamonds and I will scan the most current metal prices in real-time directly from Kitco.com.

For languages, I will primarily be using *C#* in my IDE to write the code for the functionalities of the elements within the *Windows Forms*, calculate the cost of the items based on the user input, and create objects to store information about the users and their items. To get this information, I will use *SQL* commands within my *C#* code to get, edit, add, or delete information from the database all within my IDE. I will use CSS and HTML to code the resources website. XML is used for API information retrieval.

Lastly, as my program will be written on a Windows computer and my database is on a Windows run server, the platform necessary is a computer with Windows 10 or 11 for now. This is because the application will be saved as an .exe file in the future which is primarily used by Windows computers.

Data Structures

In this subsection, I describe all the necessary database information that my project will need to have to be able to securely store information. As stated previously, I will be using *SQL Server Management Studio* to create the local database with all the necessary tables. In Figure 4 down below, I have created a database schema with some pre-input data that will be used for testing, however, most of the actual data will come from the users when deployed.

Metal Prices

	Metal_Type	Metal_Description	Price_per_Ounce	Data_Updated
	1	Gold	2649.1000	2024-12-01
	2	Silver	30.5900	2024-12-01
	3	Platinum	944.0000	2024-12-01

User

	UserID	User_email	User_name	User_pass	Num_of_Items	Total_Price	Date_Created
	1	bebadinia@yahoo.com ...	Ben Ebadinia ...	y8KtTP5SSGuV4wiLOg1YpA==	5	8648.6100	2024-05-20
	2	johndoe@gmail.com	John Doe ...	ij8Maq96UUDhVGSyUDBDVg==	0	0.0000	2024-06-22
	3	alegra2001@gmail.com	Alegra Ramos ...	IyaeGWIihxBJvTMEjfMLFg==	0	0.0000	2024-09-05
	4	JohnDoe@test	John	uDcl2nq2lXhfzF+4KeUIWA==	2	1144.4700	2024-10-08

Item

	ItemID	UserID	Metal_Type	Metal_Purity	Total_Weight	Price_of_Metal	Price_of_Diamonds	Description	Current_Price	Date_Added	Original_Price	Photo_Link	Date_Updated
	10001A	1	1	14	10.5	521.7300	NULL	14 Yellow Gold Chain 10.5G ...	521.7300	2023-06-05	412.8600	C:\Users\beb...	2024-12-01
	10002B	1	3	NULL	5.4	163.9100	2082.0000	Platinum Engagement Ring 5.4G w...	2245.9100	2023-12-31	2743.6400	NULL	2024-12-01
	10003B	4	1	14	4.0	198.2400	943.0000	14K Gold: 4G with Diamonds ...	1141.2400	2024-10-08	1141.2400	C:\Users\beb...	2024-10-08
	10004A	4	2	NULL	3.2	3.2300	NULL	Silver; 3.2G	3.2300	2024-10-08	3.2300	NULL	2024-10-08
	10007A	1	1	14	15.0	745.3300	NULL	14K Yellow Gold Rope Chain 15G ...	745.3300	2024-10-24	770.2300	NULL	2024-12-01
	10008A	1	1	10	11.5	408.1500	NULL	10K Yellow Gold Bracelet 11.5G ...	408.1500	2022-01-17	282.9500	NULL	2024-12-01
	10009A	1	1	18	74.0	4727.4900	NULL	18K Yellow Gold Bangel Set 74G ...	4727.4900	2021-04-12	3110.4900	NULL	2024-12-01

Diamond

	Diamond_ID	ItemID	Diamond_Carat	Diamond_Color	Diamond_Clarity	Diamond_Cut	Price_of_Diamond
	100001	10002B	0.50	F	VS2	Round	1074.0000
	10002D	10002B	0.50	G	VS2	Round	1008.0000
	10003D	10003B	0.25	G	VS2	Pear	943.0000
	10004D	10006B	0.25	F	VS1	Round	864.0000

Figure 4: Database Schema in SQL Server Management Studio

The first table is *Metal Prices*, which has a set key, the “Metal_Type” attribute, which is an integer that is used to identify the type of metal from the *Item* table. Then the “Metal_Description” attribute states what the “Metal_Type” is referring to. The “Price_per_Ounce” attribute stores the cost of that specific metal type that is obtained from Kitco.com. Lastly, the “Data_Updated” attribute is used to see when the last time the price was updated.

The second table is the *User* which stores user data that is necessary for the user to log in and access their account. The primary key is the “UserID” which is auto-incremented every time a new account is created. The “User_email” and “User_pass” are input from the user and are important for the log-in functionality. The “User_name” attribute is just the first and last name of the user to personalize the home screen. The “Num_of_Items” attribute sums the number of articles within the *Item* that have the same “UserID” as the user. Similar to the last attribute, the “Total_Price” attribute sums the “Current_Price” of all of the items who have the same “UserID” as the user. Lastly, the “Date_Created” attribute automatically stores the creation date of the account.

The third table is the Item table which stored all of the data about each piece of jewelry. The first attribute, “ItemID”, is the primary key and auto-generated every time a new item gets added to someone’s account. The “ItemID” could either end with the letter A, which means that it does not contain any diamonds, or letter B, which means it does contain either 1 or more diamonds. The “UserID” attribute refers to the “UserID” in the *User* table which is necessary to connect the item to the user. The “Metal_Type” attribute refers to the “Metal_Type” in the *Metal Prices* tables which is responsible for identifying the metal type along with the “Price_per_Ounce” of that metal type. The “Metal_Purity” attribute stores the purity of the metal type to allow for calculations of the “Price_of_Metal”. This can be *NULL* as only gold can have purity. The same thing applies for the “Total_Weight” attribute which store the total weight of the item, and is used to calculate the “Price_of_Metal”. The “Price_of_Metal” attribute stores the price of just the precious metal calculated from the “Total_Weight” along with the “Metal_Purity” and the “Price_per_Ounce” in the *Metal Price* table. The “Price_of_Diamonds” is the sum of all of the diamonds in the *Diamond* table that have the same “ItemID” as the item. Next is the “Description” attribute which is input from the user and auto generated. Then, the “Original_Price” attribute stores the market value of the item in that moment, which is just the sum of the “Price_of_Diamonds” (if any) and the “Price_of_Metal”. This will be a fixed attribute, along with the “Date_Added” attribute which just stores the date when the item was added. The “Current_Price” attribute stores the sum of the diamonds and gold just as the “Original_Price” does, however, this price gets updated on the daily. By using the “Original_Price”, “Current_Price”, and “Date_Added”, we are able to view the change in price since the original date of creation. Next is the “Photo_Link” attribute which is the local path for the image of the item. Lastly, the “Date_Updated” attribute allows us to see when the itemn was last updated.

The last table is the *Diamond* table, which stores the data about the diamonds. The “Diamond_ID” is the primary key and auto generated every time a new diamond gets added to an item. The second attribute “ItemID” is responsible for connecting the diamond to its respective item. The third attribute, “Diamond_Carat” refers to the weight of the diamond in Carats. Next are the “Diamond_Color” and “Diamond_Clarity” attributes which are the color and clarity of the diamond and are essential to calculating the price of each diamond. The

sixth attribute is the “Diamond_Cut” attribute which stores the diamond shape and allows the system to know in which price chart to look. Lastly, The ““Price_of_Diamond” attribute is calculated from the IDEX API based on the ”Diamond_Carat”, “Diamond_Color”, and “Diamond_Clarity”.

For a visual representation of the connections between the tables within the database, I have created a relational diagram, also in SQL Server Management Studio which can be seen in Figure 5 below. This diagram allows us to see the links between the data structures described in a previous sub section. As seen from the figure, all the tables are connected to each other in one way or another. Starting with *User*, we are able to see that this table does not have any foreign key but has a primary key attribute of “UserID”, which is used as a foreign key to the *Item* table. The *Item* table has a primary key attribute of “ItemID”, which in turn is used by the *Diamond* database as a foreign key. The *Item* table also receives another foreign key, the “Metal_Type” attribute from the *Metal Prices* table.

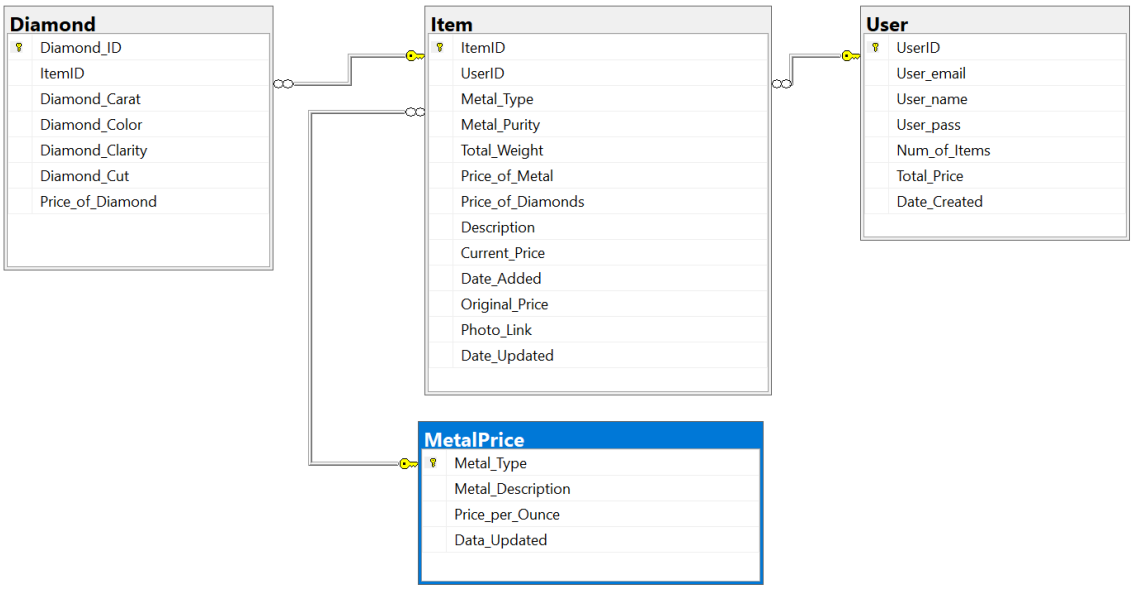


Figure 5: Relational Diagram of Databases in SQL Server Management Studio

Software Modules

In this software module sub section, I will break down the components I would need to write to have the application working properly. I will be connecting the

functionalities stated in the “Analysis” section previously to the actual modules that are necessary for the application. There will also be seven pages that will highlight the interactions and relationships of these functionalities. The pages are the “Log-In Page”, “Create Account Page”, “Home Page”, “Create Item Page”, “Item Information Page”, “Useful Links Page”, and the “Account Information Page”.

Log-In Page:

- *Email and Password textboxes:* These will allow the user to input their email address and password if they have already created an account.
- *Log-In:* If the email address and password match the already existing email and password data in the database, when the user clicks this button, they will be redirected to their “Home Page”.
- *Create an account:* By pressing this button, the user would be redirected to the “Create Account Page”.

Create Account Page:

- *Email and Password textboxes:* These will allow the user to input their name, email address, and the password twice to reduce the risk of an incorrect password.
- *Create Account:* Once this button is clicked, a UserID will be generated, and all the user information will be stored in the database. They will then be redirected back to the home page where they can access their account by logging in.

Home Page:

- *Display list of items:* All the items that are linked to the user’s account will be displayed in a list of data cells with the item description, the most current price of the item, and the change in price percentage.
- *Display total cost of items:* The sum of the current prices of all the user’s items will be displayed.
- *Display total number of items:* The sum of all the items for that user will be displayed.
- *Generate total report:* Once this button is clicked, a report will be created and saved on the user’s computer that displays the list of items with the date added, item description and the most current price of the item. Along

with this, the report will display the current date, total number of items, sum of the original prices for all the items, sum of the current prices for all the items, the difference between these two values, and the growth or decline of their investments as a whole shown as a percentage.

- *Create item:* Once this button is clicked, the user is redirected to the “Add Item Page” where they can input information to add another item to their account.
- *Select an item:* The user can select a specific item from the list and press this button which will redirect them to the “Item Information Page” of that specific item.
- *Compare items:* Once the user interacts with this button, they will be redirected to the “Compare Items Page” where they are able to select two items to compare.
- *Resources button:* If the user clicks this button, they will be redirected to the “Useful Links Page”.
- *Account information:* If the user clicks this button, they will be redirected to the “Account Information Page”.
- *Sign-Out:* If the user clicks this button, then the user will be redirected back to the “Log-In Page”.

Create Item Page:

- *Metal Type and Metal Purity drop-down menu:* The user will select the metal type and metal purity from the drop-down menus for the item which is necessary for the price calculation.
- *Item Total Weight numerical spinner:* The user will input the total weight for the item for the item which is necessary for the price calculation.
- *Add Diamond:* When user clicks this button, a Diamond Cut, Diamond Carat, Diamond Color, and Diamond Clarity drop-down menus become visible allowing users to add either one or multiple diamonds to the item.
- *Add Photo:* Once this button is pressed, the user will be able to select a JPEG from their computer which will be displayed alongside that item.
- *Add Item:* Once this button is pressed, the item is added with its calculated total to the database and the item list for that user.

Item Information Page:

- *Display Item information:* Once the item is selected from the “Home Page” and the “Select Item” button is clicked, detailed information about that item will be displayed.
- *Display Item Photo:* Once the item is selected from the “Home Page”, the item photo will be displayed alongside the item information.
- *Delete Item:* If this button is clicked, the item will be removed from the database and the user’s item list.

Compare Items Page:

- *Display Items:* The user will be prompted to select two items from a drop-down menu which will display detailed information and photos for both items side by side.
- *Compare Items:* If this button is pressed, information detailing the differences in the item’s components (Such as precious metal weight) current cost, and growth of investment will be displayed.

Useful Links Page:

- *Display links to resources about precious metals:* Information about precious metals and links to Kitco.com will be available for the user.
- *Display links to resources about diamond information:* Information about diamonds and links to information about IDEX will be available for the user.
- *Display links to resources about family store:* Information about the location and the mission statement of my family store will be available for the user.

Account Information Page:

- *Display User information:* User information such as name, email, and the total number of items linked to that account is displayed.
- *Edit User information button:* If this button is clicked, then the information in the labels will be accessible for the user to edit.
- *Delete User account button:* If this button is clicked, then the account will be removed from the database along with all of the items linked to that account.

System Structure

Within this system structure subsection, I will be connecting all the parts and software modules in the previous sub section by showing their relationships and dependencies through the use of a system flowchart (Figure 6 below) created in *Draw.io*. This diagram will create a visual representation of flow from one page to another along with how the functionalities will “get along”. This is important as all the functionalities need to be designed with these system hierarchies and order of operations in mind.

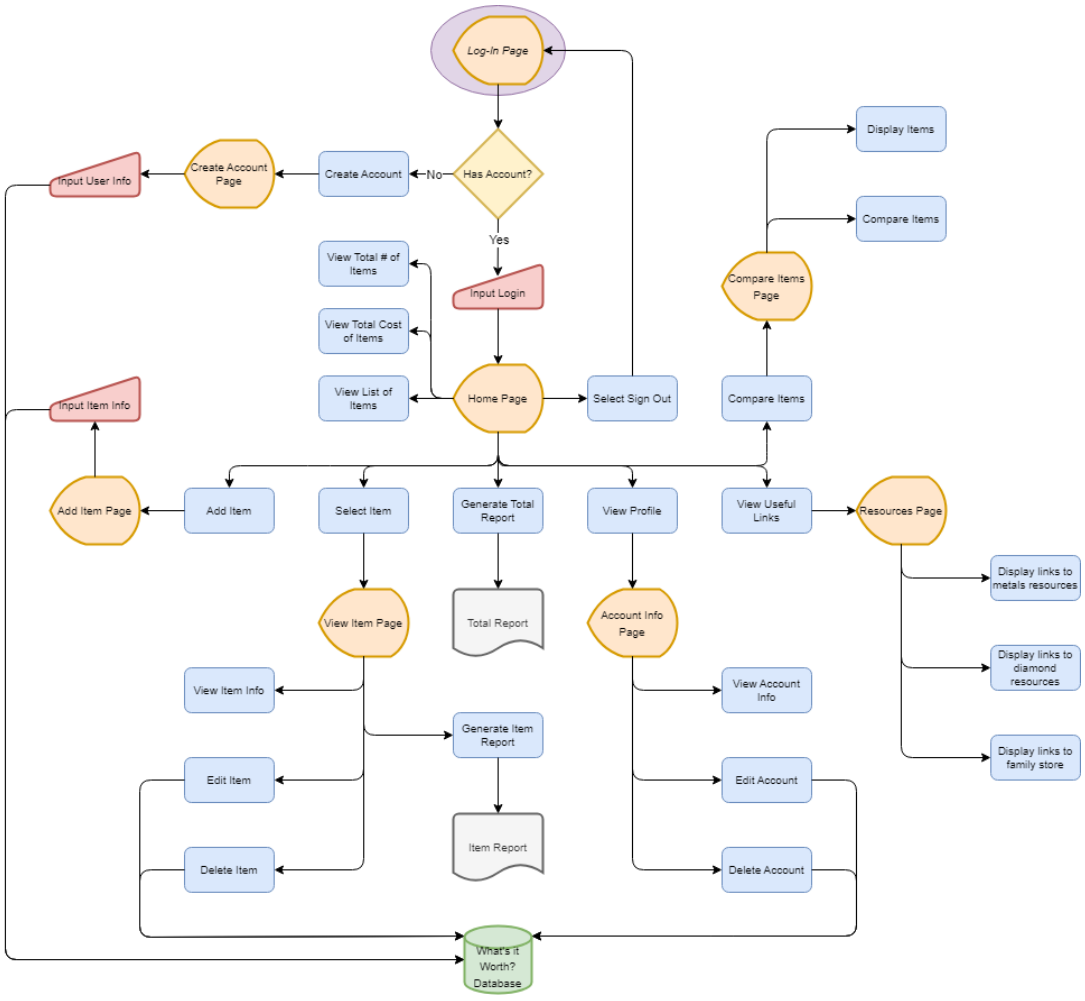
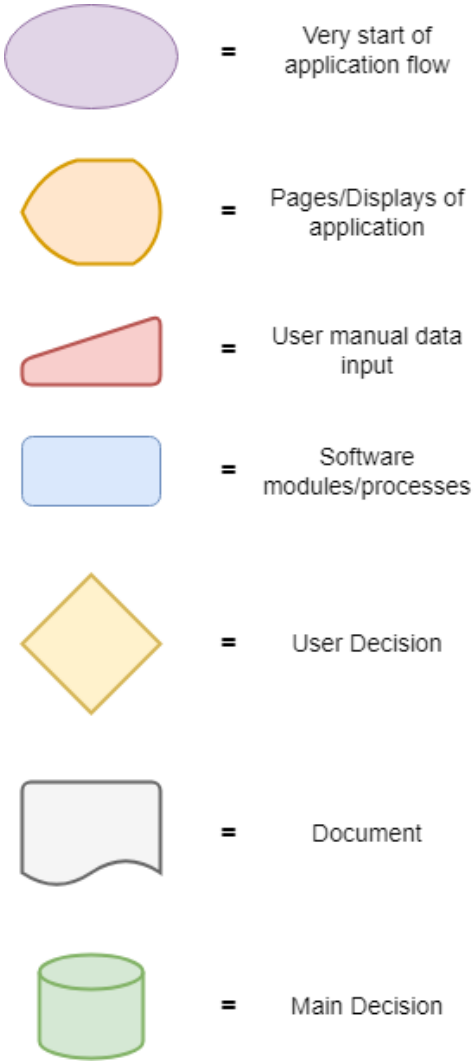


Figure 6: System Flowchart

For ease of use, I have created a key for all the shapes and colors within the chart to differentiate the different components that are interacting with one another:



Starting at the very top, the “What’s It Worth?” application begins at the Log-In page. This is the very first thing that the user is greeted with. Then the user must decide, do they already have an account or not? If they do not, then the user selects “Create Account”, which redirects them to the Create Account Page. The user would then be prompted to input information which generates an account and gets stored in the database. The user would then be redirected back to the start page where they would input their login information. If the login information matches the data within the database, the user’s Home Page would appear. This is the center of the application as the main functionalities are accessible through this page. The Home Page will display a list of all the user’s items and their most current pricing automatically. The user then has an option to select “Add Item”, “Select Item”, “View Profile”, “Resources”, or “Sign Out”. If the user chooses to select “Add Item”, then the system opens the Add Item Page which prompts the user to fill out information necessary to calculating the price of an item. Once the user has filled out all the necessary fields, the user then presses “Save Changes” in which the item is saved to the database, and the user is redirected back to the Home Page where they will be able to view the new item in the list of all items. If

the user decides to “Select Item” from the Home Page, the system opens a View Item Page for that specific item and displays all the detailed information about that item. The user then has the option “Edit Item” or “Delete Item”, which changes the information within the database and displays the updated data back on the Home Page. Similarly, the View Profile option from the Home Page in that it opens a separate page, the View Profile Page, and displays all relevant information about the account. The user would then also be given the option to either “Edit Account”, in which account data is updated and saved in the database, or “Delete Account” in which all records of the user’s account along deleted along with all the items that were linked to the account. If the user selects the “Resources” option, they will be redirected to the Useful Links Page which displays all additional information about precious metals, diamonds, and my family’s jewelry store. Lastly from the home page, the user can select the “Sign Out” option in which the user will be safely redirected back to the Login Page.

Interfaces/Scenes/Screens

Lastly, within this Interfaces subsection, I will be showing mockups and wireframes for the application to get an idea of how users can react with the system. The wireframes seen below were created in *MockFlow* and allow for the visualization of the screens that the user would see. The screens were created with the idea that the application will be used on a computer that has access to the internet.

The first screen is the Log-In page which can be seen in Figure 7 down below. This screen is the first thing that the user will see once the application is opened and allows them to access their account using an email and password that they have created prior. The other function that the user can do from this page is to create an account.

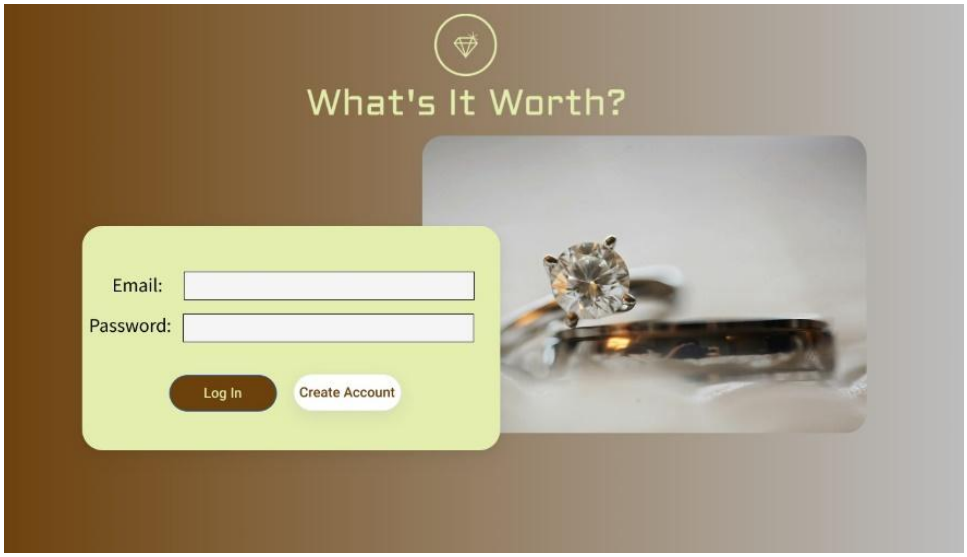


Figure 7: Log-In Page

The second screen is the Create Account page (Figure 8 below) which can only be accessed from the Log-In page. This screen displays necessary information that the user will need to fill out to create an account. Once the changes are saved, this screen will close, and the Log-In page will reopen.

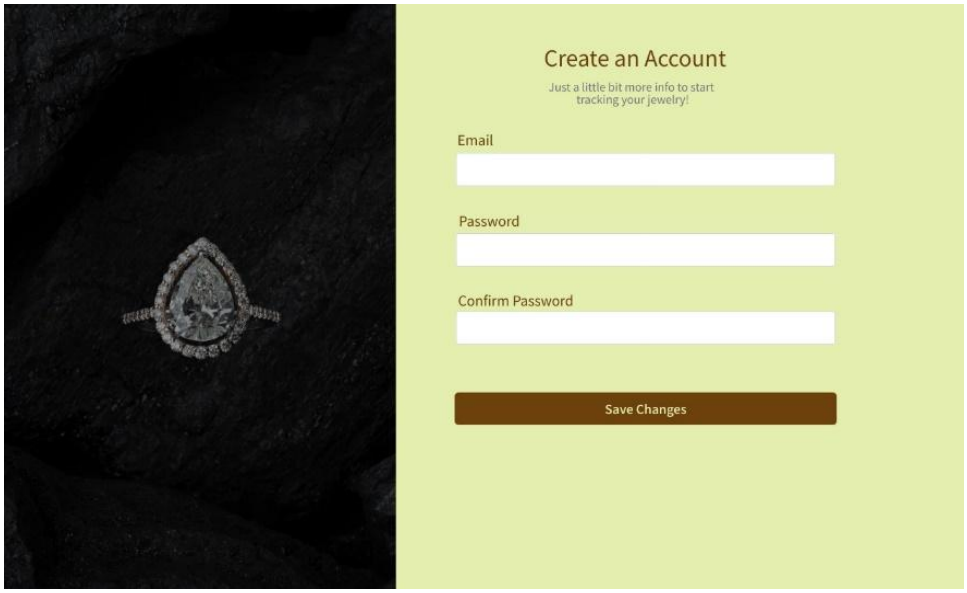


Figure 8: Create Account Page

Once the user successfully logs into their account, they will see the third screen, which is the Home page (Figure 9 below). This screen displays all the main functionalities that the user can access along with a list of all the user's items that they have created previously with a brief description, current market value, and the price change. The application also displays the user's name, sum of all the

current prices, total number of items, and gives the user the ability to sign out. The user will also have the option to Generate Total Report which creates and saves a summary report of all the items with all of the pertinent data listed in the previous sub-section.

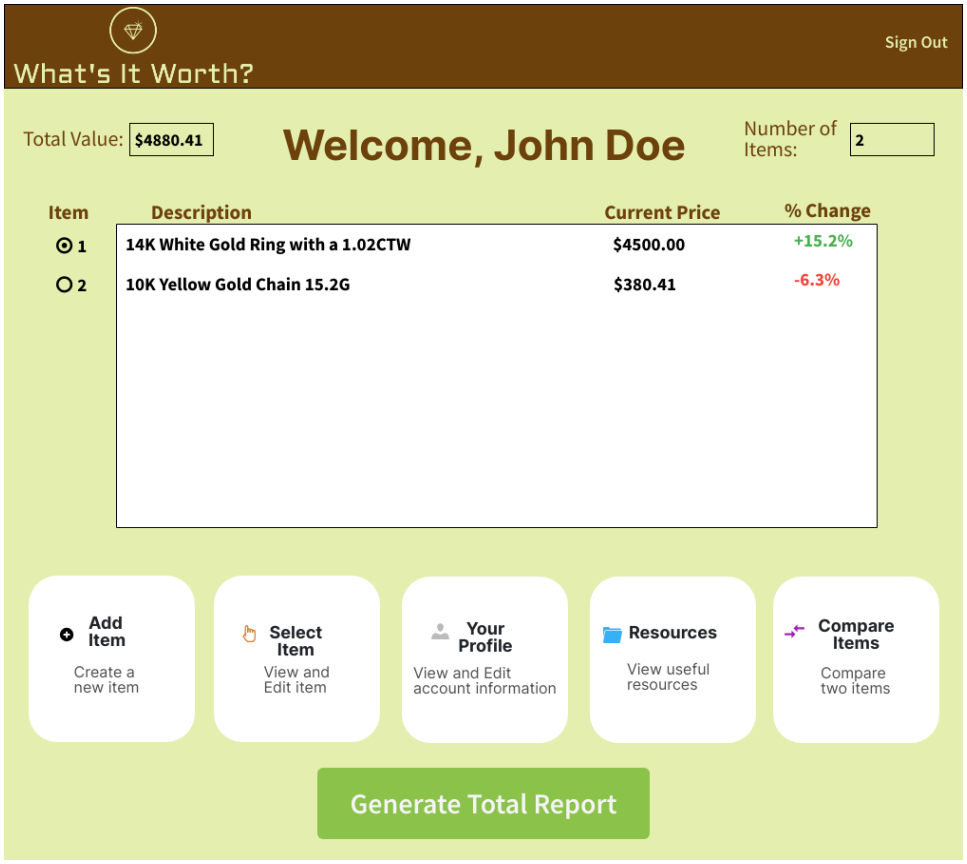


Figure 9: Home Page

From the Home page, the user can access the Add Item page which is the main functionality of this application (Figure 10 below). This screen displays necessary information that the user will need to fill out to create and add an item to their account. The user will need to select the Metal Type and Metal Purity from the drop-down menu and input the Total Weight in the accompanying text field. Then the user has the option to add a diamond which opens a subsection where the user inputs the diamond information. Lastly, the user will also have an option to add a photo of their item by selecting a local image on their computer. Once all the fields are filled out and the user is satisfied, the user will select Save Changes and this screen will close and the Home page will reopen with the updated item within the list.

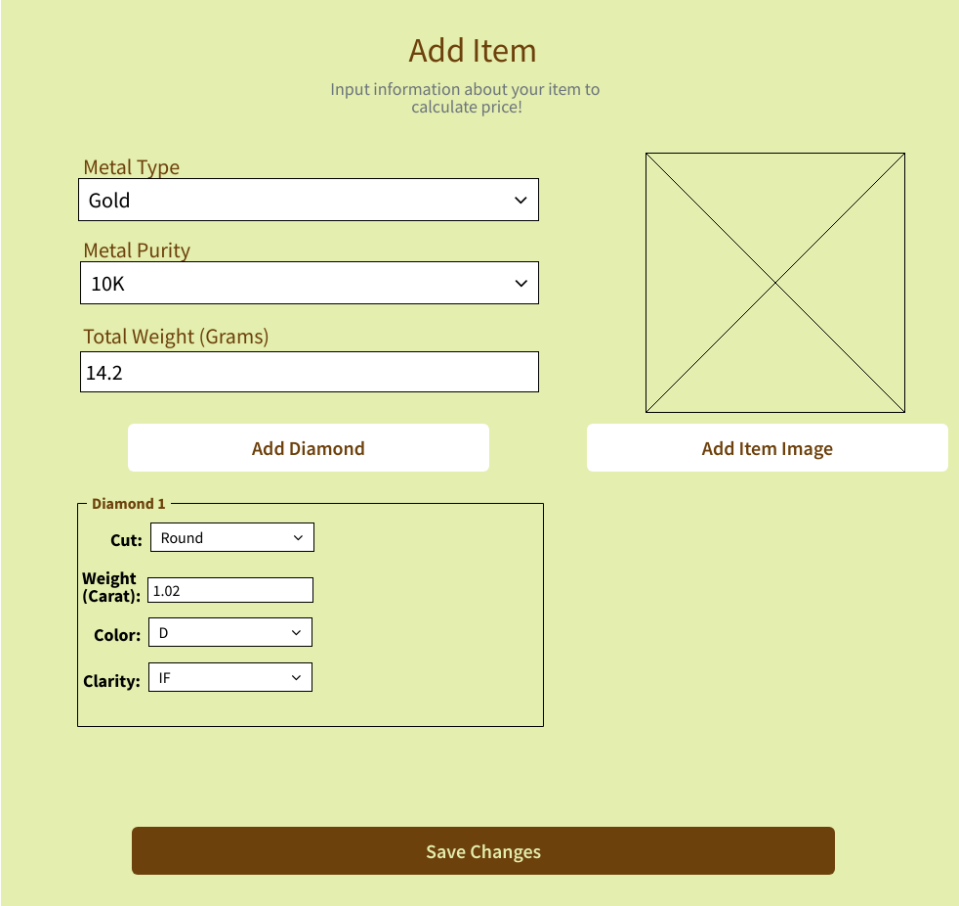


Figure 10: Add Item Page

From the Home page, the user can also access the View Item page which can be displayed once the user selects an item from the list from the radio buttons and selecting the “Selecting Item” button (Figure 11 below). This screen displays all the information about the item the user has selected. The item number is displayed on the top left side with an option for the user to switch the item with a press of the button. On the left side of the screen, the application will display the physical attribute of the item along with the photo of the item on the right side. On the bottom right side, the application displays all relevant pricing information on a detailed level. The user then has the option to either edit the item information where the text fields will become accessible to the user, go back to the Home page which closes the screen, delete the item which removes the item from the list on the Home page, and generate item report which will be created and saved on the user’s computer with all of the pertinent data listed in the previous sub-section.

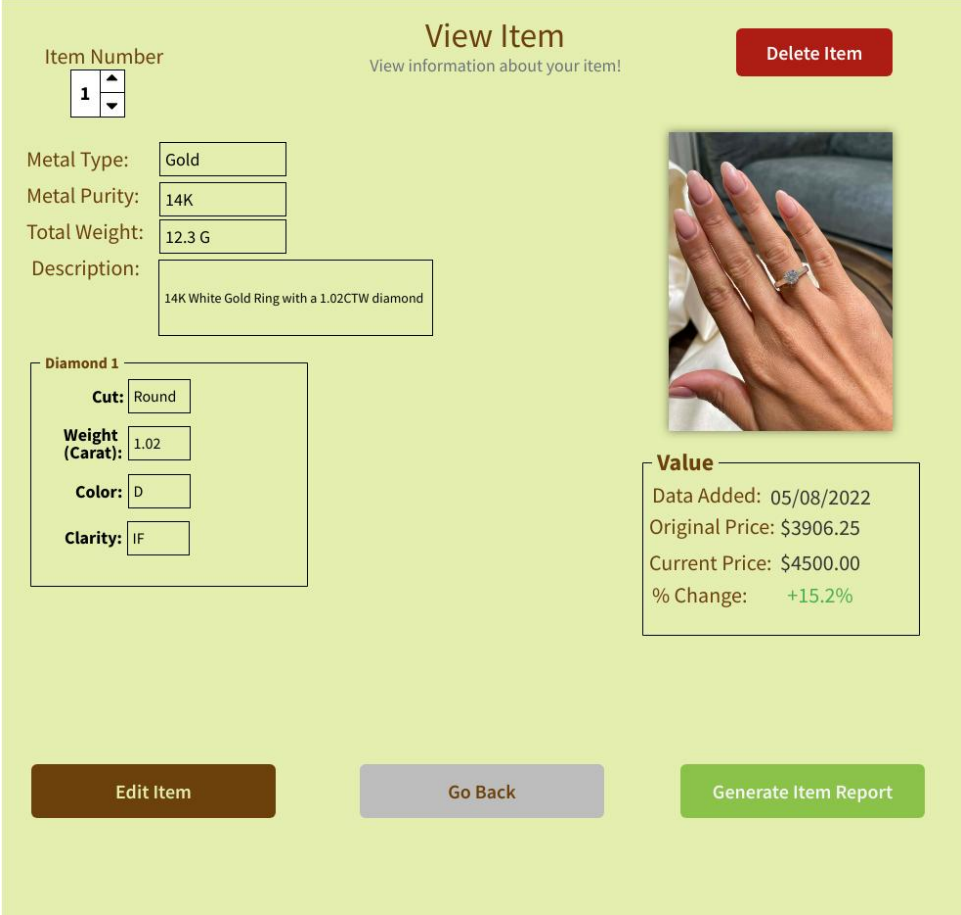


Figure 11: View Item Page

The Account Information page (Figure 12 below) can also only be accessed from the Home page. This screen displays all the information about the user's account. The user then has the option to either edit their account information (including password) where the text fields will become accessible to the user, go back to the Home page which closes the screen, and to delete the account which removes the account and its items from the database.

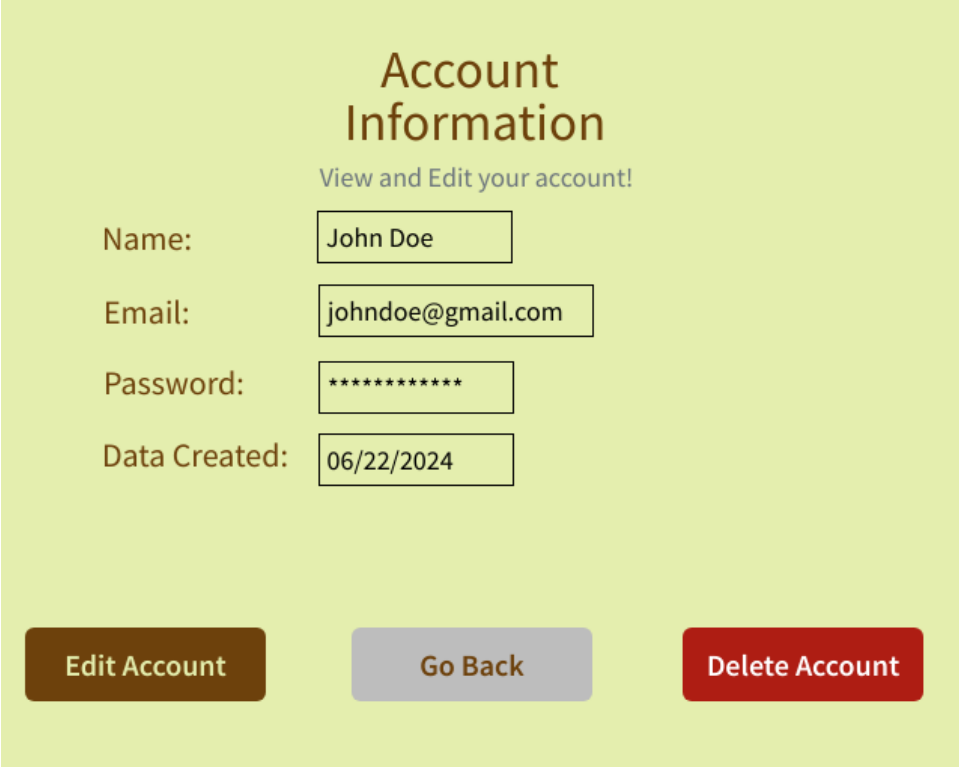


Figure 12: Account Information Page

Lastly, the Compare Items page (Figure 13 below), which can be accessed from the Home page, allows the user to select two items and view detailed item information along with the photo side by side. Once the user makes their selection and presses the Calculate button on the bottom of the screen, the application calculates and displays the change in price of each item and the number of days this change took place. The user also has the option to either to go back to the Home page which closes the screen.

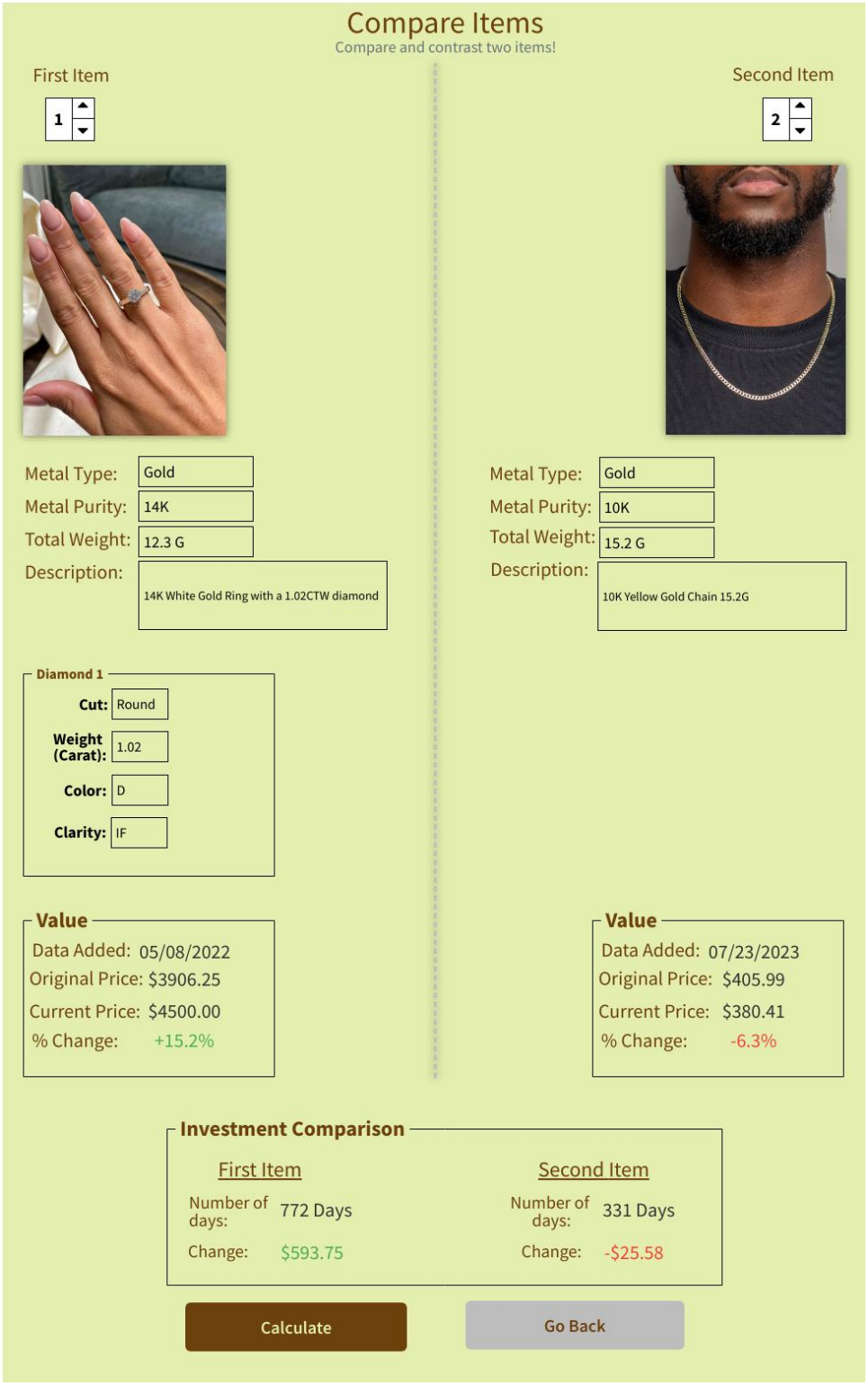


Figure 13: Compare Items Page

INSTALLATION

This section goes over how to install and use the “What’s It Worth?” application. The guide below details how to install, set up and run the application for users and developers wanting access to the application. As the users and developer requirements are the same, so is the installation currently.

This guide outlines the steps a user/developer needs to take to install and set up the application on their machine. It includes how to get the files, restore the database, and run the application. The steps are:

1. Set up Visual Studio.
    a. *Go to https://visualstudio.microsoft.com/*
    b. *Click on “Download Visual Studio” Button*
    c. *Find and open the installer*
    d. *Follow the installation steps*
2. Set up .NET Framework.
    a. *Go to https://dotnet.microsoft.com/en-us/download/dotnet-framework*
    b. *Click on latest version*
    c. *Find and open the installer*
    d. *Follow the installation steps*
3. Set up SQL database.
    a. *Go to https://learn.microsoft.com/en-us/sql/database-engine/install-windows/install-sql-server?view=sql-server-ver15*
    b. *Follow instructions to download Microsoft SQL Server*
    c. *Go to https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16*
    d. *Follow instructions to download SQL Server Management Studio*
4. Copy the “What’s It Worth?” files.
    a. *Download .zip file*
    b. *Extract files into new folder that you would like to use as your workplace on Visual Studio*
5. Import database.
    a. *Open SSMS and connect to local server*
    b. *Right click on ‘Databases’ in Object Explorer and select ‘Import Data-tier Application”*

    c. *Using the import wizard, select the WhatsItWorth.bacpac file from your workplace project folder and import database*

6. Update the connection string.
    a. *Open the App.config file in your preferred text editor*
    b. *Replace placeholders such as ServerName and DatabaseName with your actual server name (often localhost for a local machine) and the database name you restored.*

Example syntax of a connection string to update:

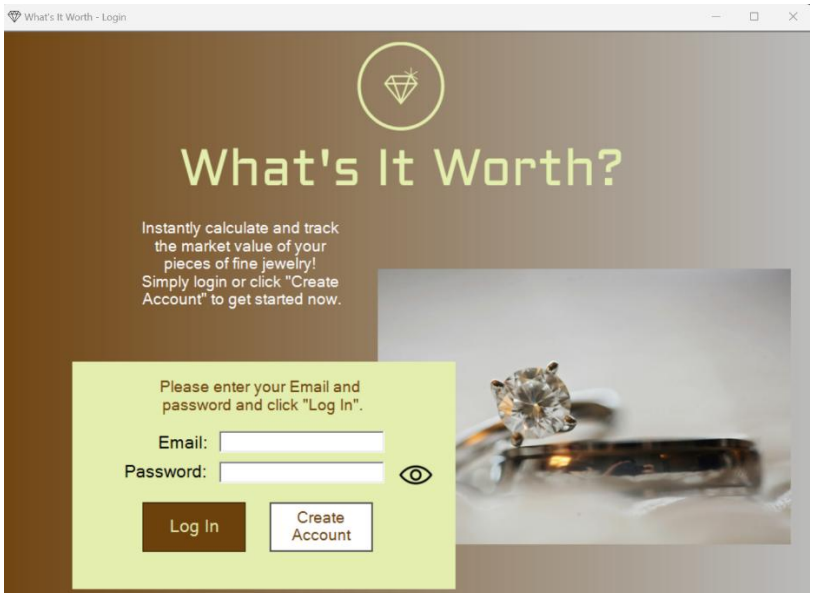
```
<connectionStrings>
<add name="WhatsItWorth.Properties.Settings.WhatsItWorthConnectionString"
connectionString="Data Source=YOURSERVERNAME;Initial
Catalog=WhatsItWorth;Integrated
Security=True;Encrypt=True;TrustServerCertificate=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

7. Build the application.
    a. *Open the WhatsItWorth.sln file in Visual Studio*
    b. *Import the project workplace file with the source code into Visual Studio.*
    c. *Build the application.*
8. Run the application
    a. *Press “Run” in top of Visual Studio*

SAMPLE SESSIONS

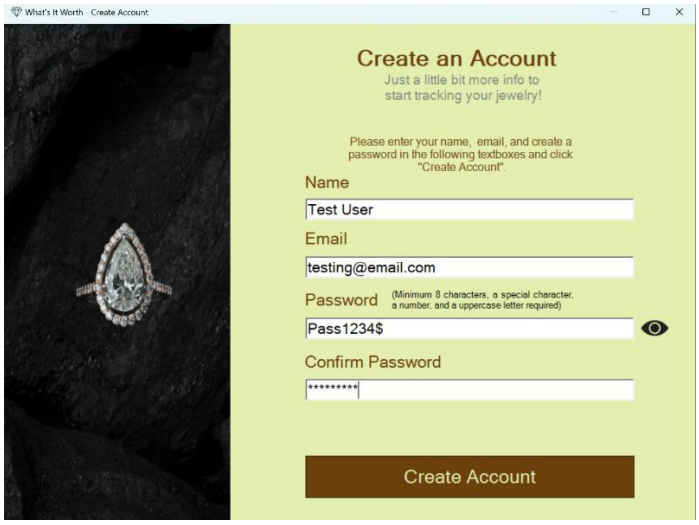
This section covers a sample first-time users' session for the "What's It Worth?" system. For each step or interaction, there will be the name of the step, screenshot of the screen underneath it, along with a description of each below the photo. This walkthrough will show a new user creating an account and adding an item to their account.

1. Starting the Application



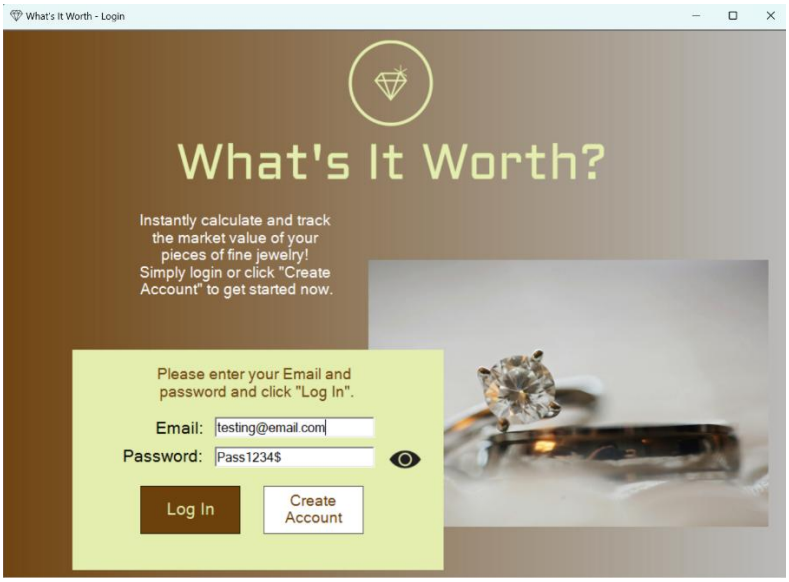
The login page welcomes users with the "What's It Worth?" logo, an overview of the application, a username and password input boxes, and two buttons that say, "Log In" and "Create Account". The background has a photo of a diamond ring to set the theme to a classier application. Here, will press the "Create Account" button.

2. *Creating an Account*



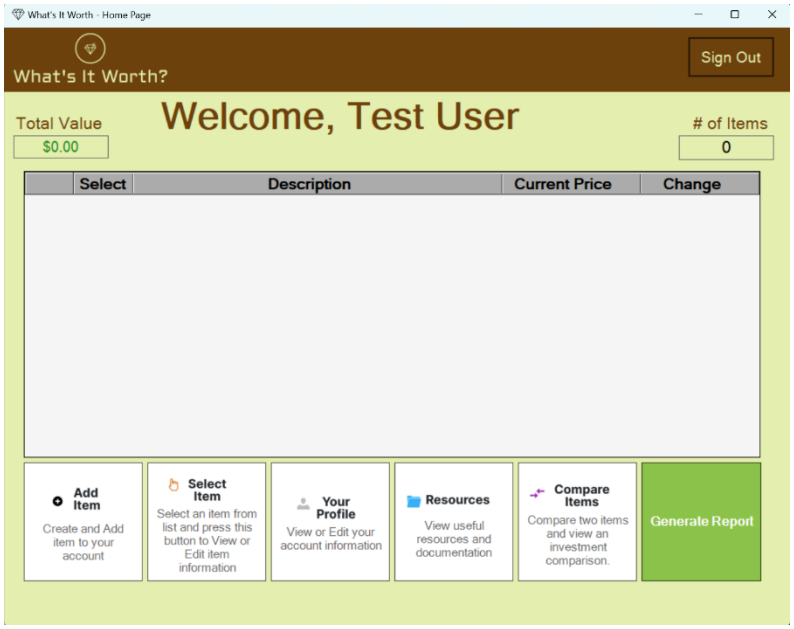
After selecting “Create Account”, users see the “Create the Account” screen displaying 4 textboxes for users to input their Name, Email, Password, and Password again. These are all validated in real time and a user cannot create an account unless there are no errors. Here we see that our name is “Test User”, email is ‘testing@email.com”, and password is “Pass1234$” which meets the minimum requirements. From here, we select create account.

3. *Logging in*



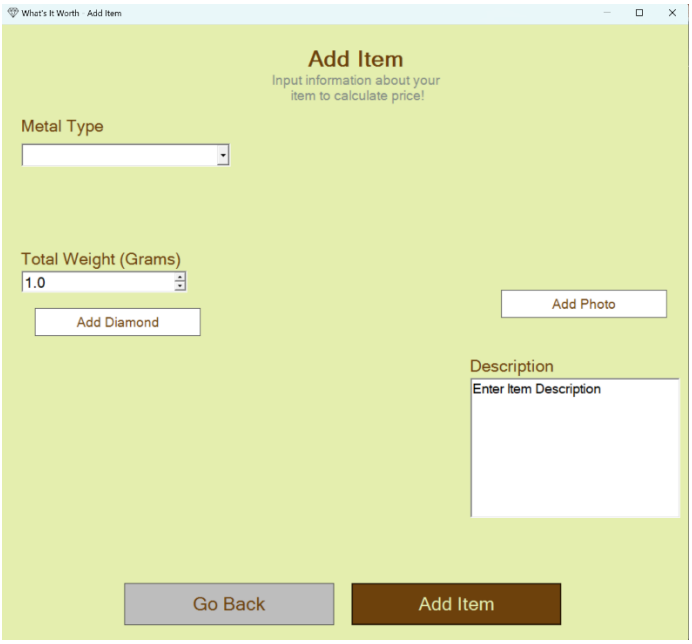
Users then enter the email and password from the account created previously. From here, we will click the Log In button and the system will verify the credentials.

4. Home Page



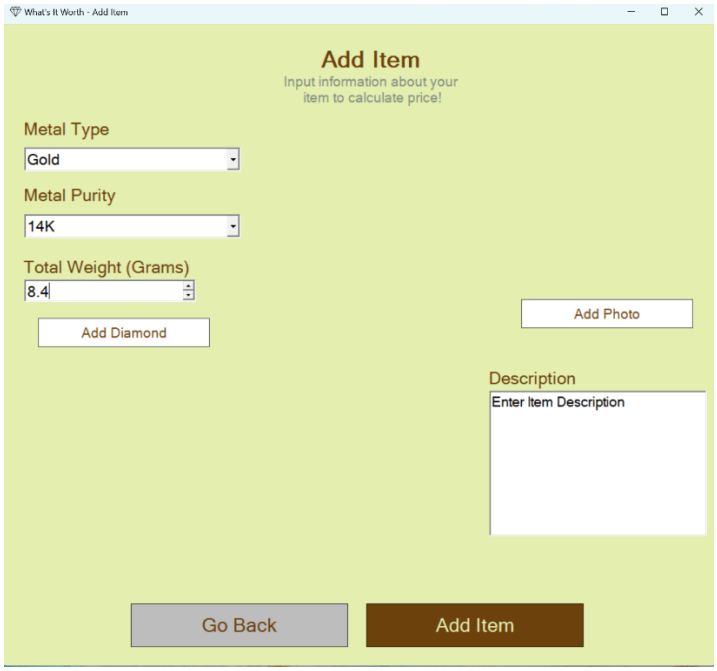
If credentials are verified, the user gets redirected to their own customized Home Page. This is the main page for the application where most of the functionalities are accessed. We can see a banner in the top of the screen with the logo on the left side and a Sign Out button on the right. Underneath that, we can see a welcome message for the user using the name that they set when creating their account. To the left of that, we can see the “Total Value” box that shows the summation of the current prices of all their items. On the right side, we can see the number of items the user has created. Here, they are both 0 as this is a new account. In the middle of the screen, we can see a table with all of the details about the item such as description, current price, and change. The is also a check box in the select column to the left of the details. We can see here that the table is empty as the account does not has any items yet. On the bottom of the screen, we can see an “Add Item”, “Select Item”, “Your Profile”, “Resources”, “Compare Items”, and “Generate Items” buttons. We can now press the “Add Item” button.

5. *Creating Item*



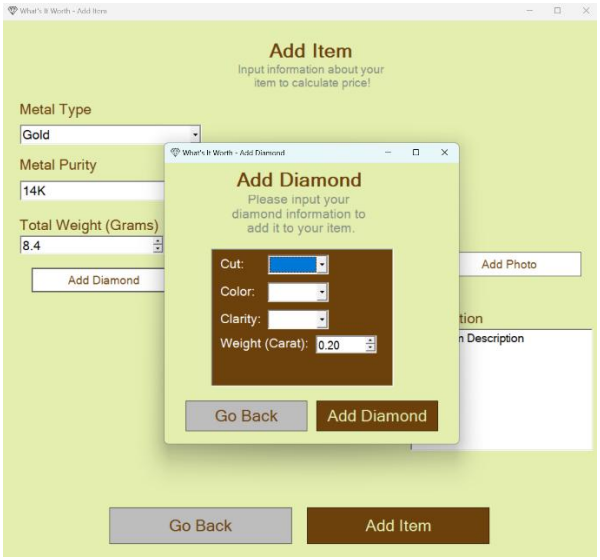
If the “Add Item” button is clicked from the Home Page, the Add Item page will be displayed. This page will display “Metal Type”, “Total Weight” combo-boxes, “Description” textbox, an “Add Diamond” button, and an “Add Photo” button. From here, we add the metal type and total weight of the item.

6. *Inputting Metal Information*



We chose gold for the metal type which displays an extra “Metal Purity” combo-box where we selected 14K. We also input the total weight as 8.4 Grams. From here, lets click the “Add Diamond” button.

7. *Adding a Diamond*

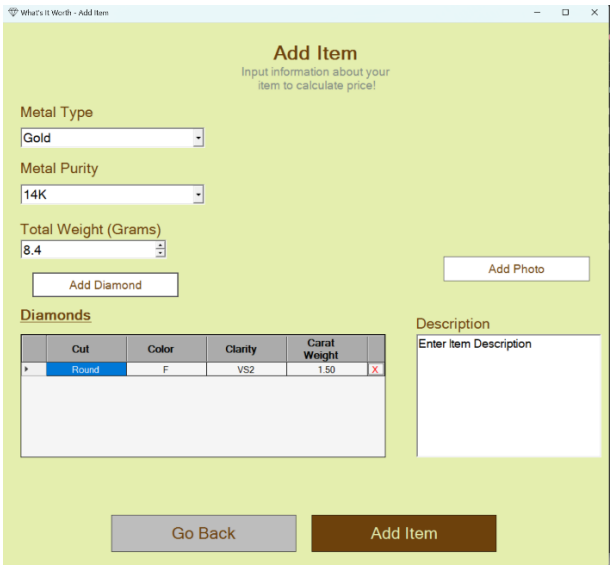


If the “Add Diamond” button is clicked from the Add Item page, the Add Diamond page will be displayed. From here the user needs to select/input the “Cut”, “Color”, “Clarity”, and “Weight”. There is also a “Go Back” and “Add Diamond” button at the bottom of the screen.

8. *Inputting Diamond (Not Found)*

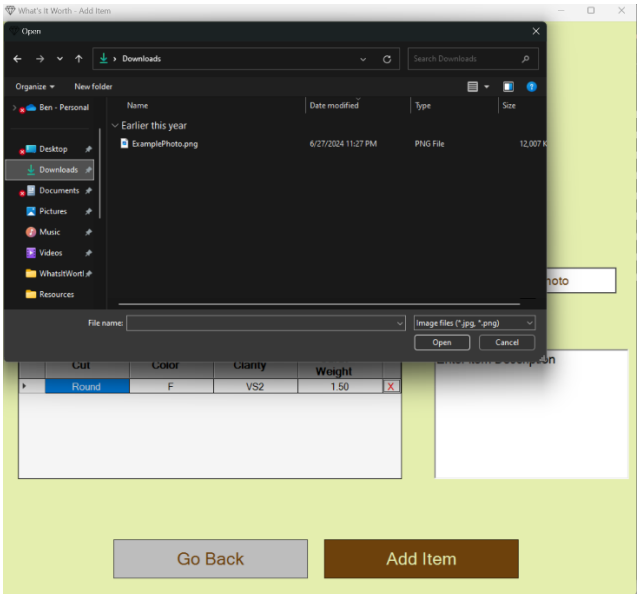
We can select Oval for the diamond “Cut”, D for the diamond “Color”, Clarity for the diamond “Clarity”, and input 8.5 for the diamond “Weight”. After inputting, we click on “Add Diamond”. However, this displays “Diamond Not Found”. This is because that the diamond requested is not currently on the IDEX marketplace, so an accurate price is not able to be calculated based on the characteristics. So we can readjust the diamond and press “Add Diamond” again.

9. Added Diamond



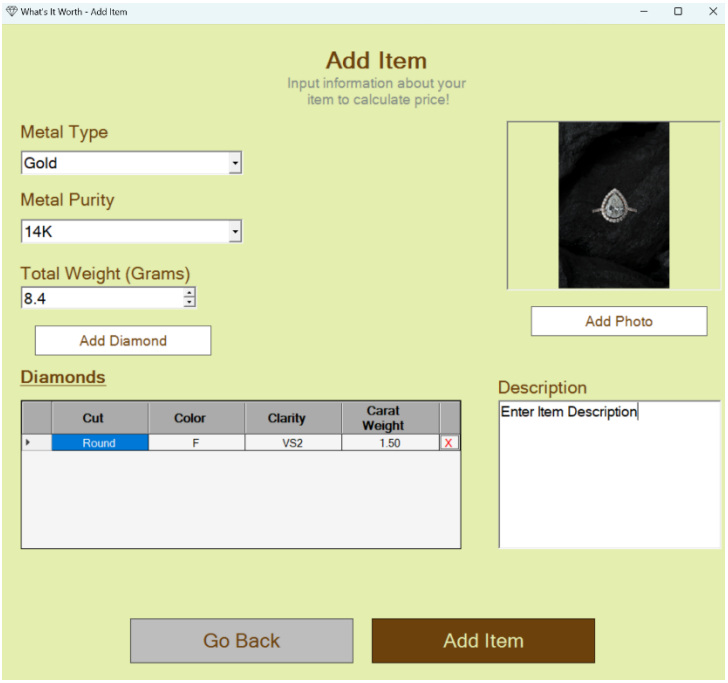
If your diamond is valid and price is obtained, then the Add Diamond page will close and the Add Item page will show a table with the added diamonds’ details. You can add multiple diamonds to an item by pressing “Add Diamond” and remove a diamond from the table by clicking the small ‘x’ in the corresponding row. We then click on the “Add Photo” button.

10. Adding Photo



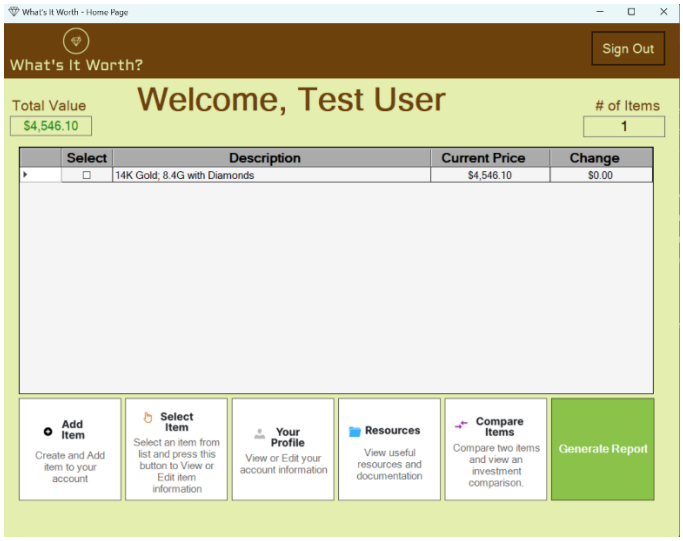
If the “Add Photo” button is clicked from the Add Item page, a local folder browser will be displayed. This will allow you to select a .png or .jpg image to attach to the item. Let’s select ‘ExamplePhoto.png’ and click ‘Open’.

11. Adding Item



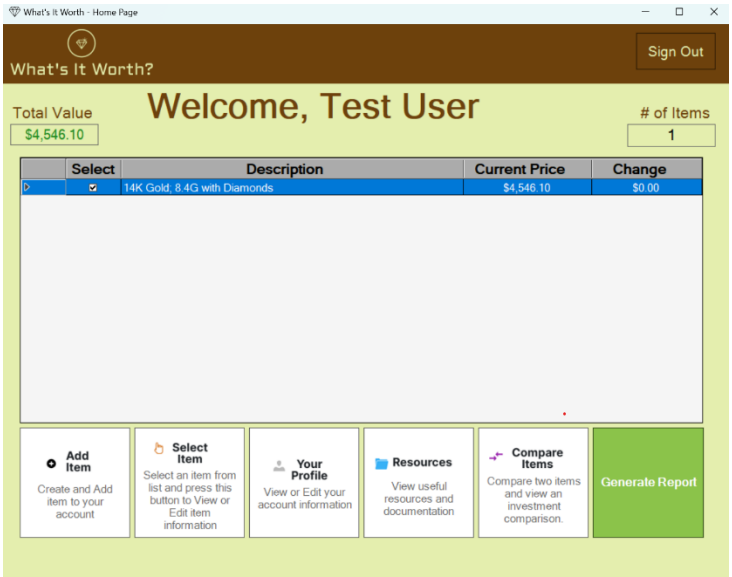
Now we can see all our item information along with the photo of the item. The user is also able to add a description, however, one will be autogenerated if no description is input. We then click the “Add Item” button.

12. Viewing New Item



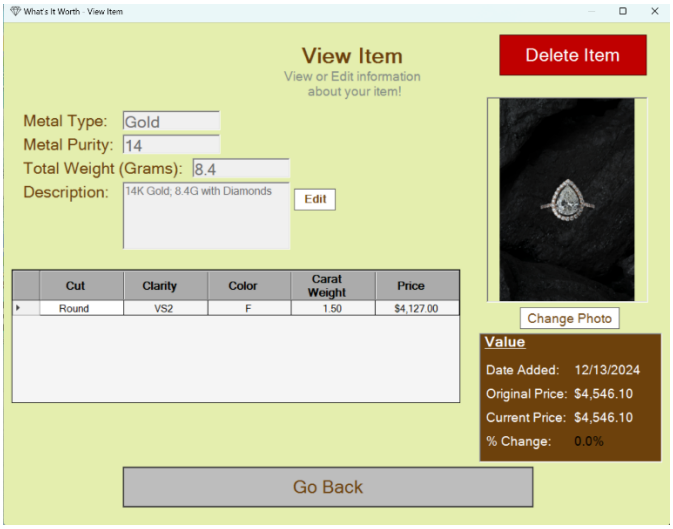
Once the item is added, the Home Page is displayed with the updated information. We can see that the “Total Value” had changed to $4546.10, and that the “# of Items” has been changed to 1. In the middle, we can see an overview of the item.

13. Selecting Item



We select the item using the checkbox in the “Select” column in the corresponding row for the item we want to view. From here, we press the “Select Item” button.

14. Viewing Item



After selecting the item, the View Item page appears with the “Metal Type”, “Metal Purity” (If applicable), “Total Weight”, “Description”, diamond table (If applicable), and photo that we set when we created the item. We can also see a panel in the bottom right corner with “Value” information such as “Date Added”, “Original Price”, “Current Price”, and “% Change”. There is also a “Delete Item” button, that if clicked, deletes the item from the user’s account. Lastly, there is a “Go Back” button on the bottom of the screen that takes the user bag to the Home Page. purpose of this section is to break down and organize all the tasks required to develop the “What’s It Worth? Application. This is essential to the successful completion of my application in not just the required time, but enough time in case something does not go according to plan. Below, I have broken down the application into 36 necessary tasks. As my time frame is from March 1st, 2024 to December 1st, 2024, I have plenty of time to complete all the development of the code and the testing required. I predict the task that will take up the most amount of time is coding the price calculation algorithm since there are so many variables and differences between the pricing of diamonds.

TROUBLESHOOTING

This section goes over the troubleshooting that can still occur during a user walkthrough or installation of “What’s It Worth?” application. The table below gives the symptoms, causes, and solutions to these issues/bugs.

Symptom	Cause	Solution
SQL connection failure.	Invalid connection string.	Check the connection string to the database.
Not able to log in.	Database issue.	Check database in SSMS and SQL connection string.
Error generating report.	RevortViewer package is not installed.	Close application, rebuild solution, and restart application.
Black screen on startup.	Display resolution mismatch or graphics driver issue.	Ensure that device meets project requirement.
Error message: "File not found".	Missing or corrupted source code.	Verify source code files in the original solution match source code in local solution.
Home Page table not showing data.	Invalid connection string.	Check the connection string to the database.

REFERENCES

Aldridge, Kristin A. “Inside the World of Gem and Jewelry Appraising.” *Gemological Institute Of America*, 11 July 2014, www.gia.edu/gia-news-research-jewelry-appraising-student-panel.

“Azure SQL Database – Managed Cloud Database Service:” *Microsoft Azure*, azure.microsoft.com/en-us/products/azure-sql/database. Accessed 8 May 2024.

“The Best Free Stock Photos, Royalty Free Images & Videos Shared by Creators.” *Pexels*, www.pexels.com/. Accessed 8 May 2024.

“The Best Free Stock Photos, Royalty Free Images & Videos Shared by Creators.” *Pexels*, www.pexels.com/. Accessed 8 May 2024.

“Blog.” *Connoisseurs Jewelry Cleaner*, 17 Nov. 2021, connoisseurs.com/blog/.

“Bring Every Team Together under One Roof.” *Atlassian*, www.atlassian.com/software/jira. Accessed 8 May 2024.

“Diamond Price Calculator.” *Washington Diamond*, www.washingtondiamond.com/diamond-price-calculator. Accessed 8 May 2024.

“Download Visual Studio Tools - Install Free for Windows, Mac, Linux.” *Visual Studio*, visualstudio.microsoft.com/downloads/. Accessed 8 May 2024.

Elmasri, Ramez, and Shamkant B. Navathe. *Fundamentals of Database Systems*. 7th ed., Addison-Wesley, 2015.

“Free Flowchart Maker and Diagrams Online.” *Draw.Io*, draw.io/. Accessed 8 May 2024.

“Free Logo Maker - over 20 Million Logos Made.” *LOGO.Com*, logo.com/. Accessed 8 May 2024.

“Free Project Management Tool.” *GanttProject*, www.ganttproject.biz/. Accessed 8 May 2024.

Garrett, Jesse James. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders, 2011.

Johnson, Jeff. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. 2nd ed., Morgan Kaufmann, 2014.

“Let’s Build from Here.” *GitHub*, github.com/. Accessed 8 May 2024.

"Precious Metals." *KITCO*, www.kitco.com/price/precious-metals. Accessed 8 May 2024.

"Rapaport Price Lists." *Diamonds.Net*, www.diamonds.net/Prices/RapaportPriceLists.aspx. Accessed 8 May 2024.

"Real-Time Precious Metal Rates and Currency Conversion API." *MetalpriceAPI*, metalpriceapi.com/. Accessed 8 May 2024.

Rubin, Kenneth S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. 1st ed., Addison-Wesley, 2013.

Satzinger, John W., et al. *Systems Analysis and Design in a Changing World*. 5th ed., Course Technology, 2009.

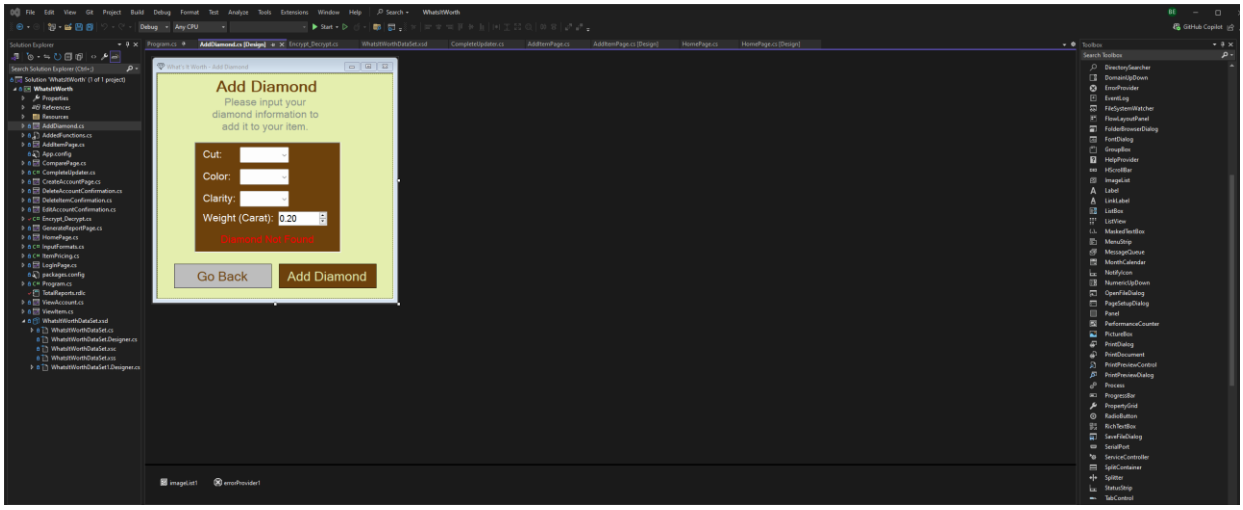
"SQL Server Downloads." *Microsoft*, www.microsoft.com/en-us/sql-server/sql-server-downloads. Accessed 8 May 2024.

"Wireframe Tools, Prototyping Tools, UI Mockups, UX Suite, Remote Designing." *MockFlow*, wireframepro.mockflow.com/. Accessed 8 May 2024.

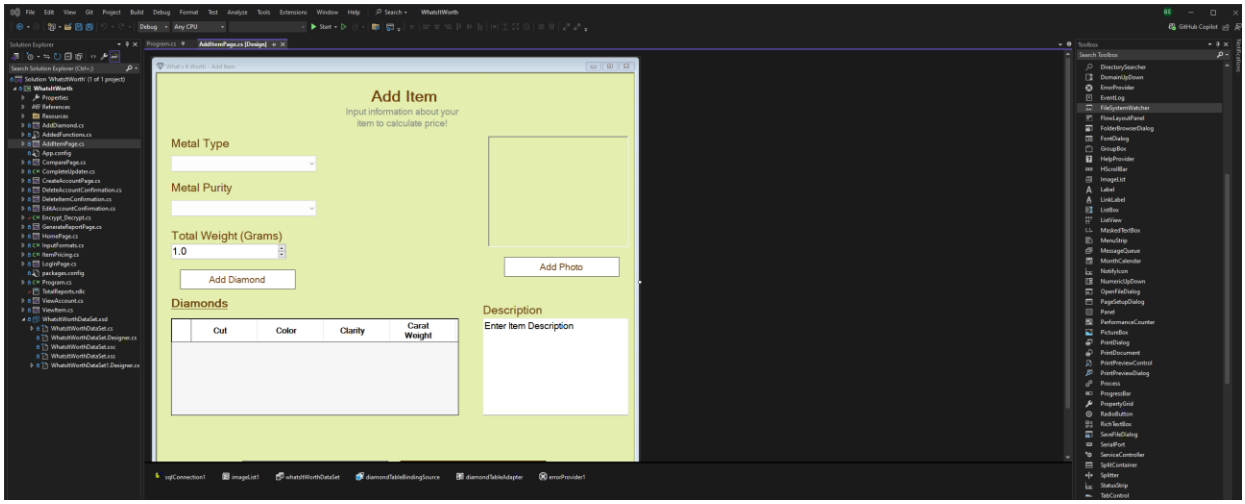
SOURCE CODE FILE LISTINGS

Interfaces Screens

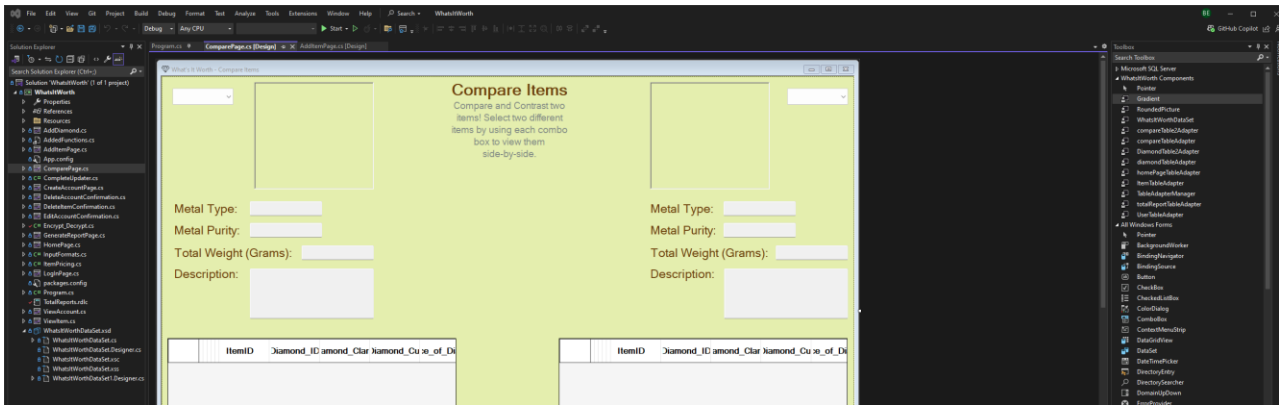
AddDiamond.cs



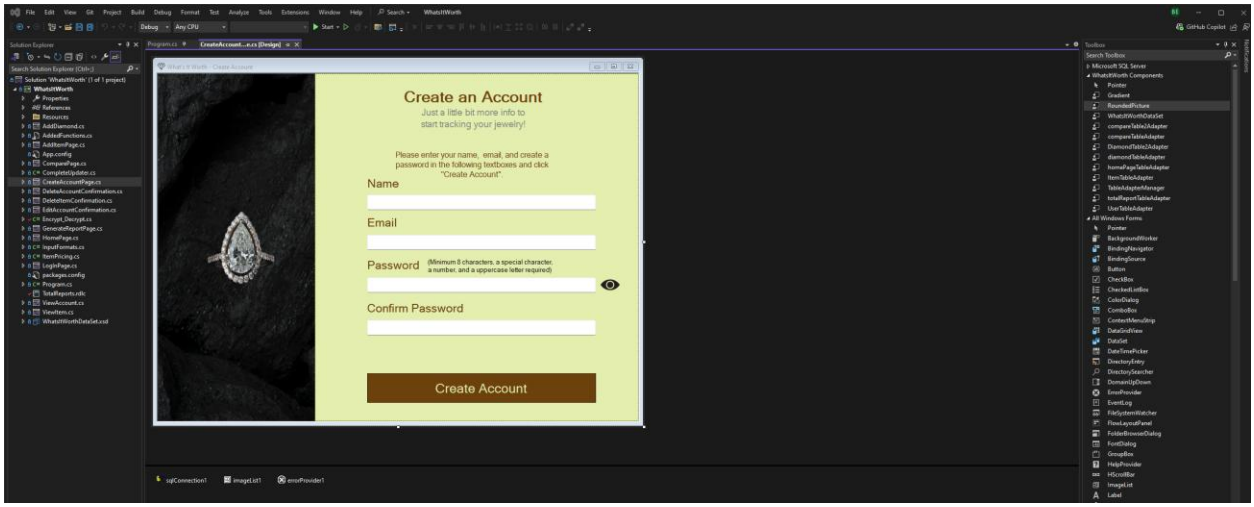
AddItemPage.cs



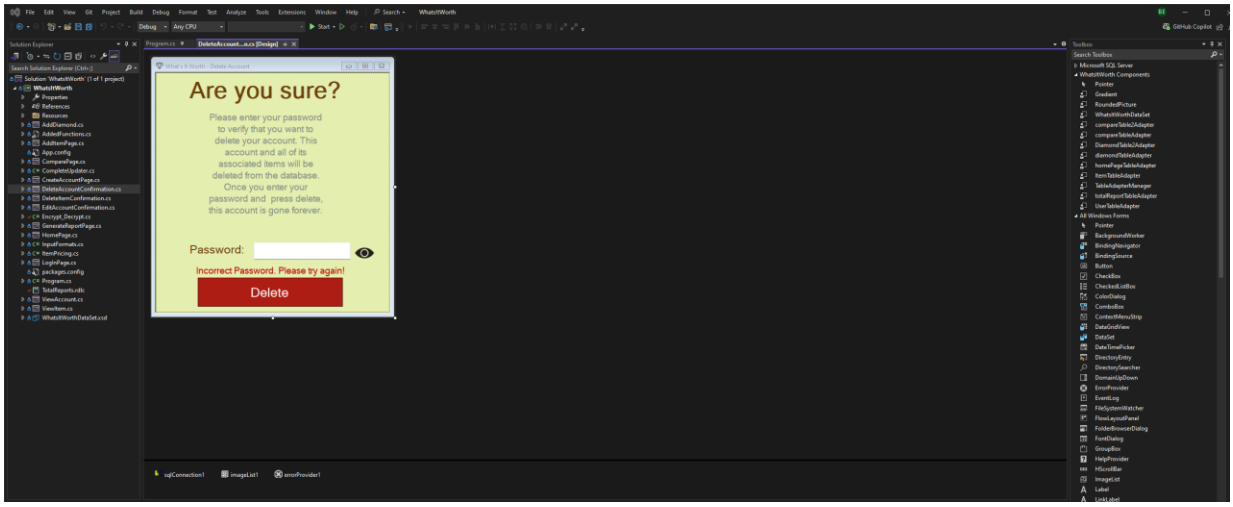
ComparePage.cs



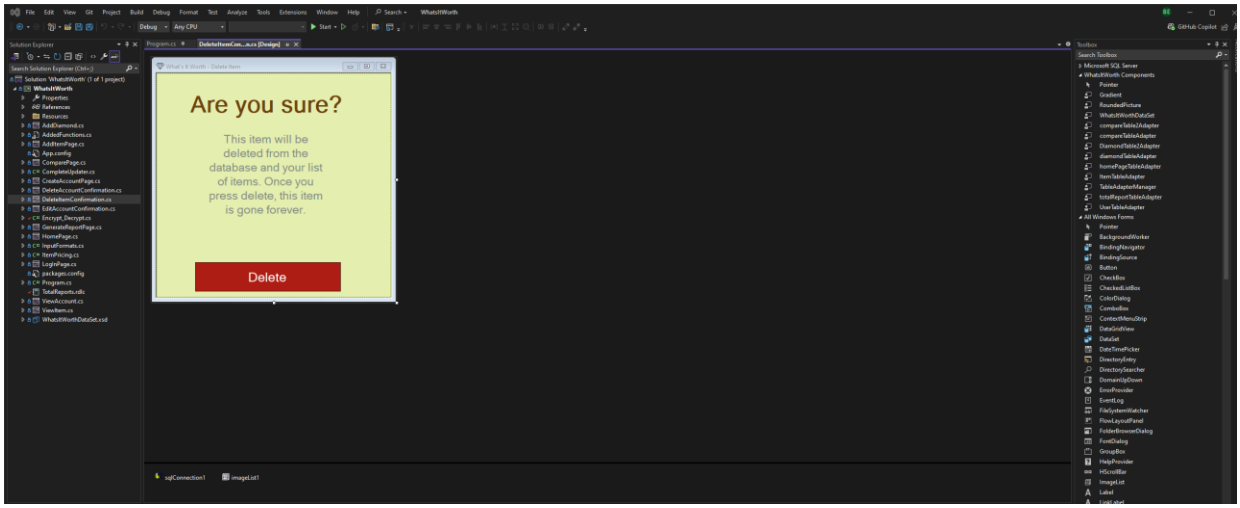
CreateAccountPage.cs



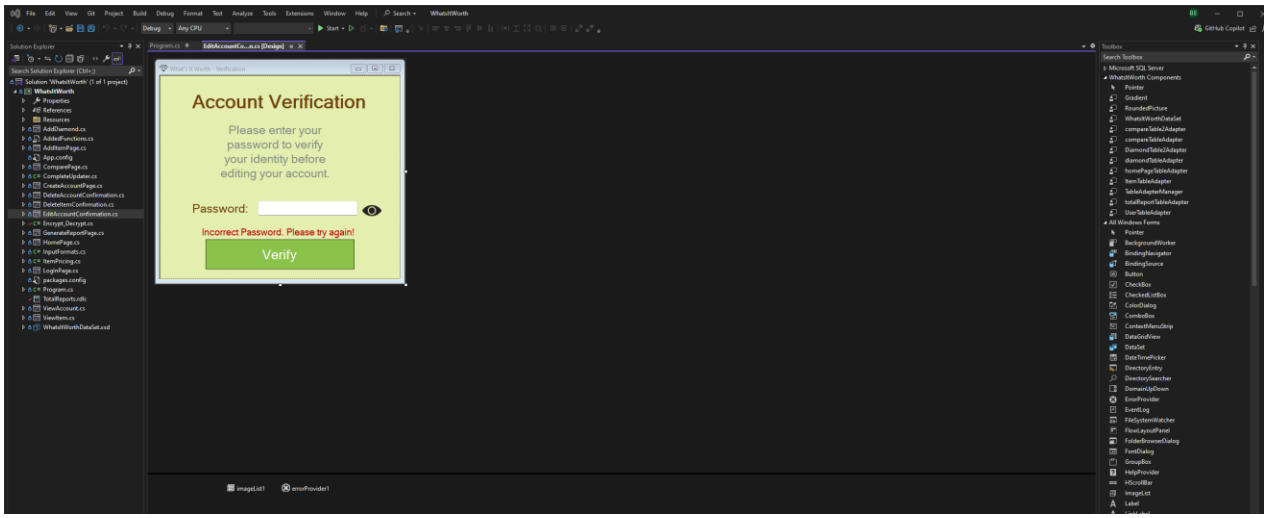
DeleteAccountConfirmation.cs



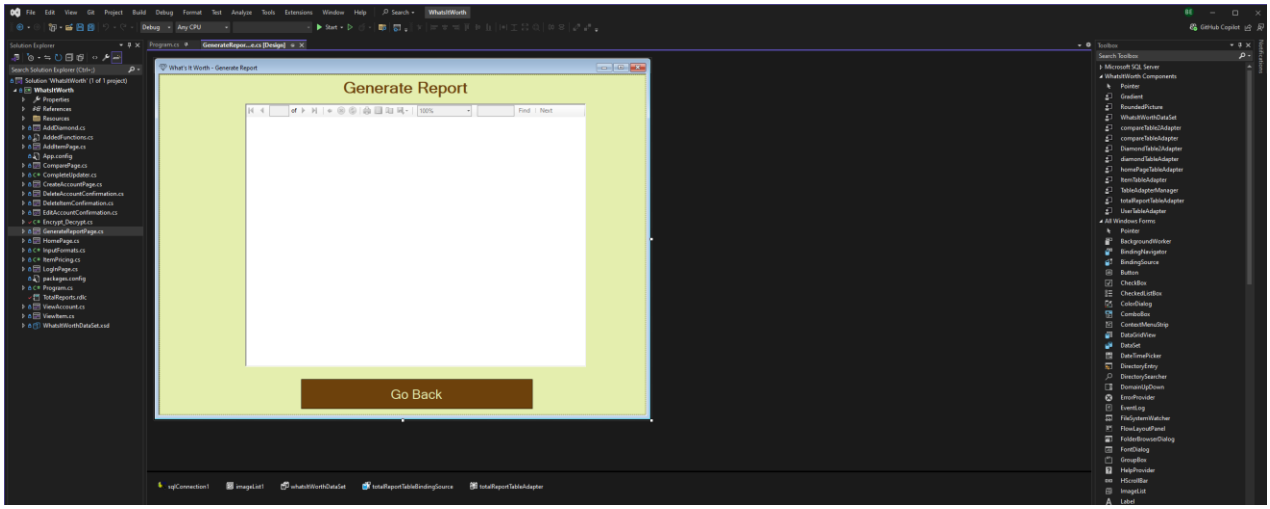
DeleteItemConfirmation.cs



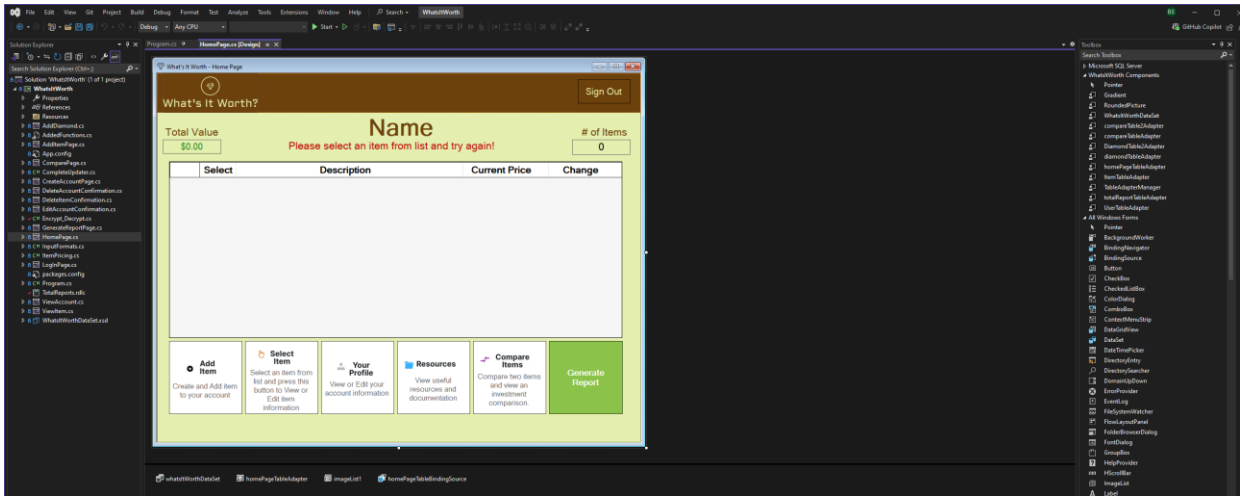
EditAccountConfirmation.cs



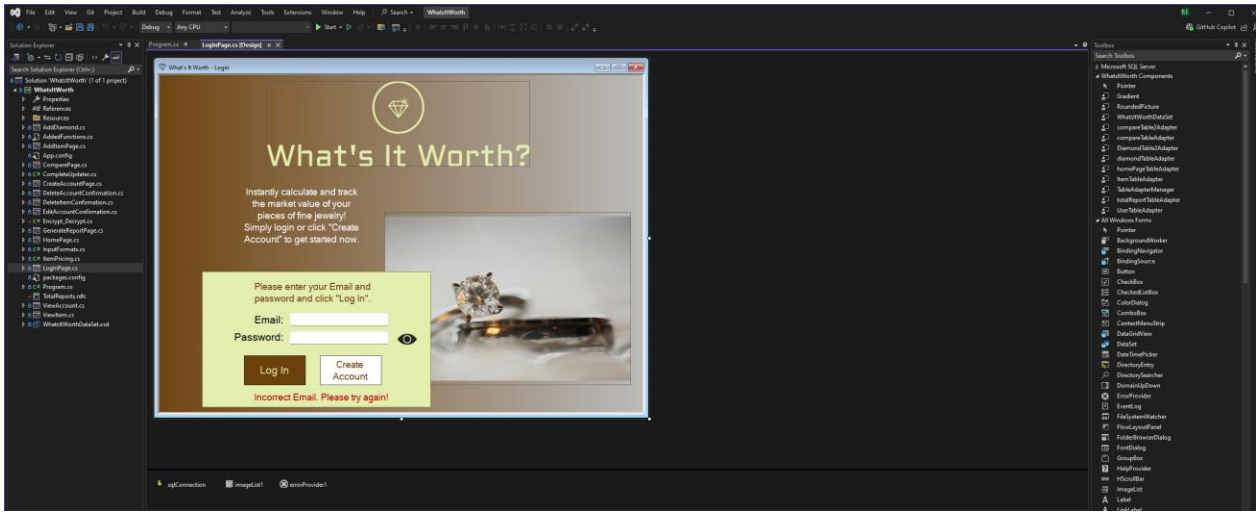
GenerateReportPage.cs



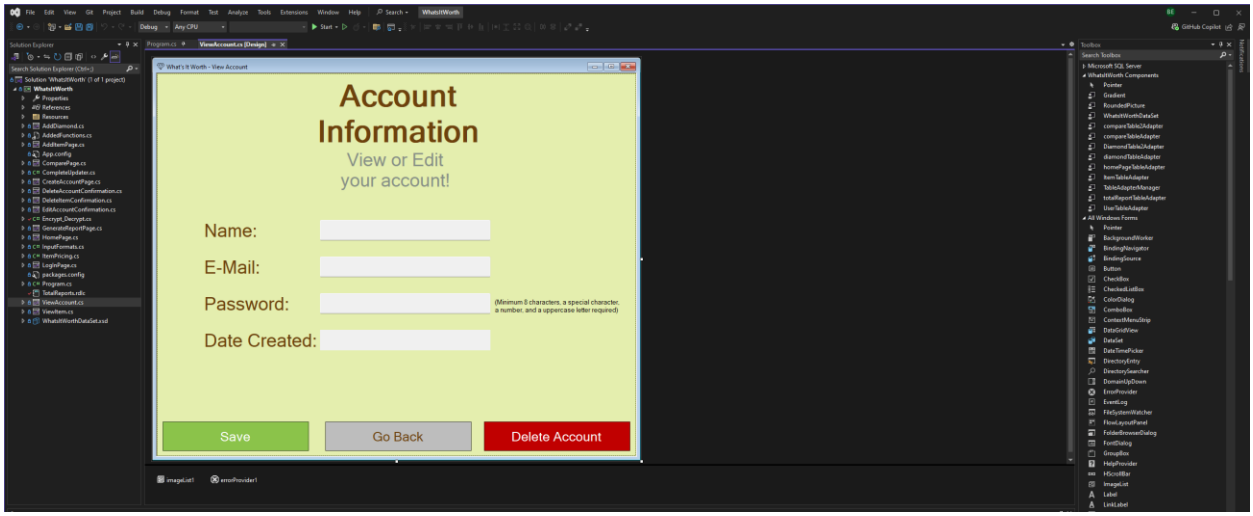
HomePage.cs



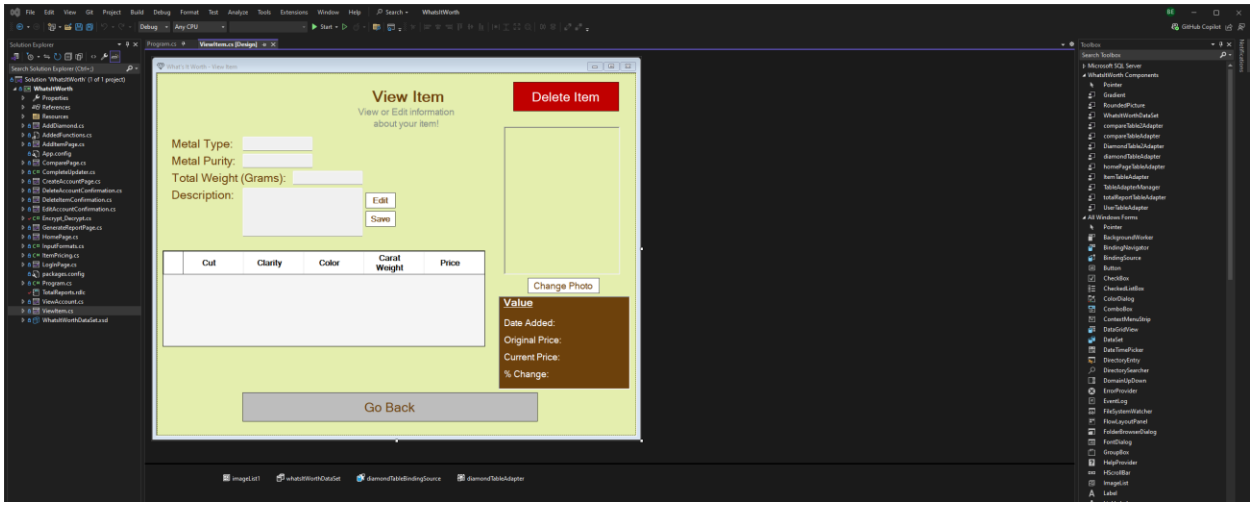
LoginPage.cs



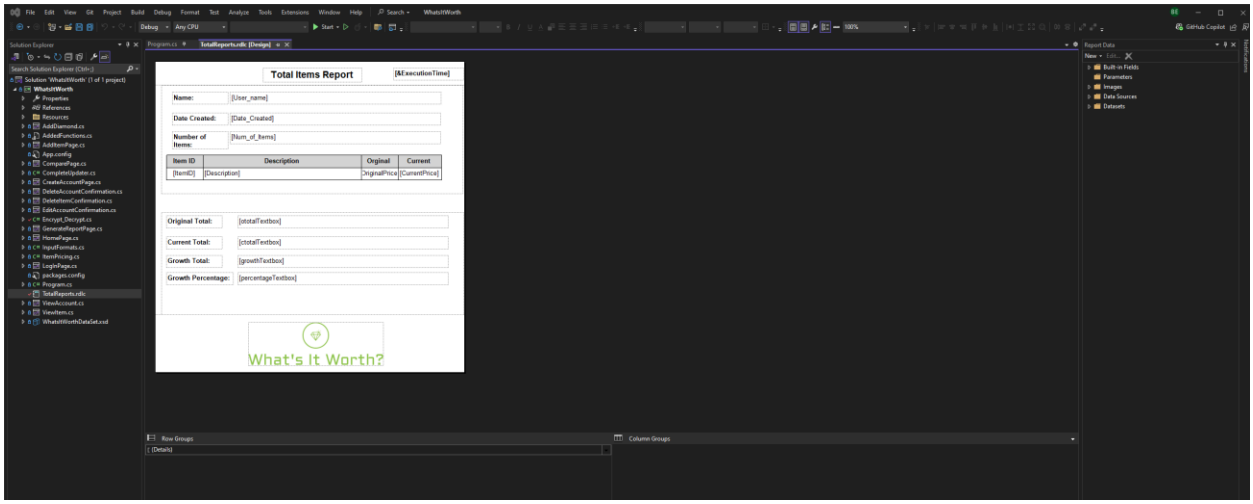
ViewAccount.cs



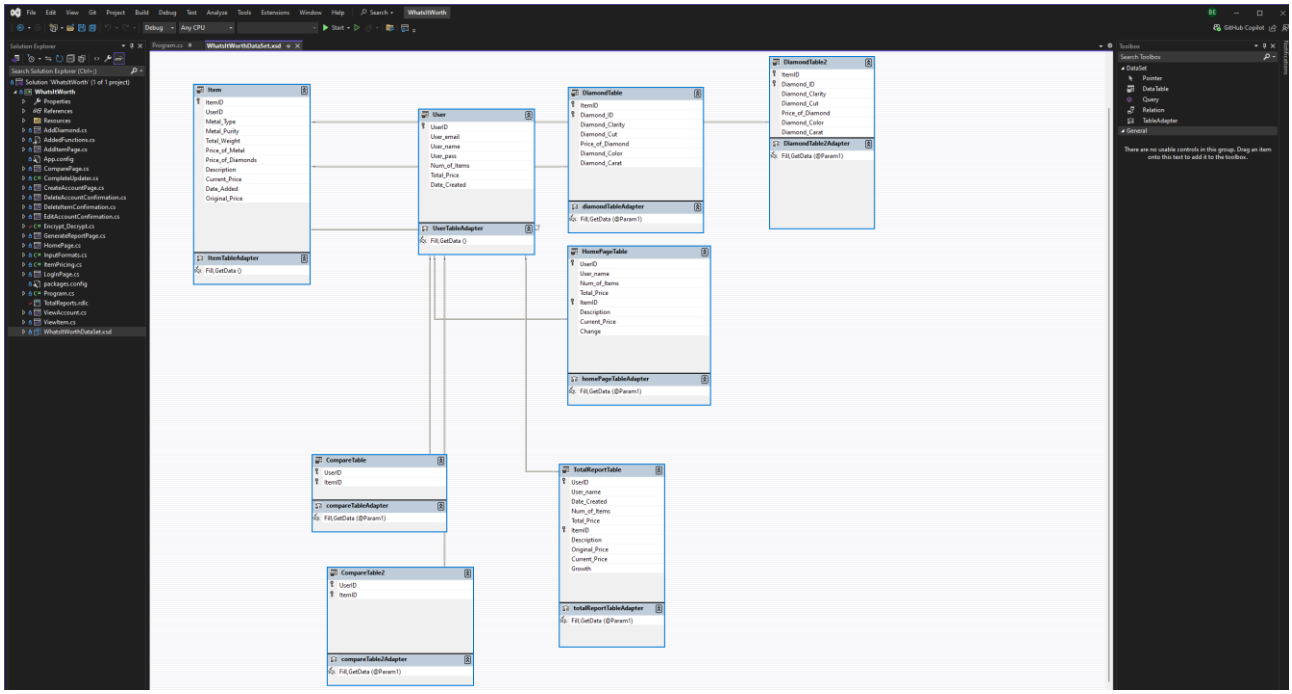
ViewItem.cs



TotalReports.rdlc



WhatsItWorthDataSet.xsd



Source Code

AddDiamond.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class AddDiamond : Form
    {
        public String DiamondString;
        private bool cutValid, colorValid, clarityValid, certificationValid;


        public ItemPricing newItem;


        public AddDiamond()
        {
            InitializeComponent();
```

```
        }

        private void AddDiamondPage_Load(object sender, EventArgs e)
        {

        }


        private void addDiamondButton_Click(object sender, EventArgs e)
        {
            if (cutValid && colorValid && clarityValid)
            {

               try
                {
                    if (newItem.getDiamondPrice(cutComboBox.Text,
colorComboBox.Text, clarityComboBox.Text, weightSpinner.Text) == true)
                    {
                        DiamondString = cutComboBox.Text + ";" +
colorComboBox.Text + ";" + clarityComboBox.Text + ";" + weightSpinner.Text;
                        this.DialogResult = DialogResult.Yes;
                    }
                    else
                    {
                        incorrectLabel.Visible = true;
                    }
                }
                catch
                {
                    incorrectLabel.Visible = true;
                }

            }
            else
            {
                CancelEventArgs ex = new CancelEventArgs();
                cutComboBox_Validating(sender, ex);
                colorComboBox_Validating(sender, ex);
                clarityComboBox_Validating(sender, ex);

            }
        }

        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }


        private void cutComboBox_Validating(object sender, CancelEventArgs e)
        {
            if (cutComboBox.Text == string.Empty)
            {
                errorProvider1.SetError(cutComboBox, "Please Select Cut");
                cutValid = false;
```

```
            }
            else
            {
                errorProvider1.SetError(cutComboBox, "");
                cutValid = true;
            }
        }

        private void colorComboBox_Validating(object sender, CancelEventArgs
e)
        {
            if (colorComboBox.Text == string.Empty)
            {
                errorProvider1.SetError(colorComboBox, "Please Select Color");
                colorValid = false;
            }
            else
            {
                errorProvider1.SetError(colorComboBox, "");
                colorValid = true;
            }
        }

        private void clarityComboBox_Validating(object sender, CancelEventArgs
e)
        {
            if (clarityComboBox.Text == string.Empty)
            {
                errorProvider1.SetError(clarityComboBox, "Please Select
Clarity");
                clarityValid = false;
            }
            else
            {
                errorProvider1.SetError(clarityComboBox, "");
                clarityValid = true;
            }
        }

    }
}
```

AddedFunctions.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace WhatsItWorth
{
    public class Gradient : Panel
    {
        public Color LeftColor { get; set; }

        public Color RightColor { get; set; }
        public float Angle { get; set; }

        protected override void OnPaint(PaintEventArgs e)
        {
            LinearGradientBrush brush =
                new LinearGradientBrush(this.ClientRectangle, this.LeftColor,
this.RightColor, this.Angle);
            Graphics g = e.Graphics;
            g.FillRectangle(brush, this.ClientRectangle);
            base.OnPaint(e);
        }
    }

    public class RoundedPicture : PictureBox
    {

        protected override void OnPaint(PaintEventArgs e)
        {
            GraphicsPath g = new GraphicsPath();
            g.AddEllipse(0, 0, ClientSize.Width, ClientSize.Height);
            this.Region = new System.Drawing.Region(g);
            base.OnPaint(e);
        }
    }
}
```

AddItemPage.cs:

```
using System;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class AddItemPage : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;

        private bool typeValid, purityValid;
```

```
        private string itemIdentification;
        //List<string> combinedDiamonds = new List<String>();
        //Dictionary<string, string> diamondList = new Dictionary<string,
string>();
        //diamondList.Add(addDiamondForm.DiamondString + "");
        private string newestID = "";
        private string tempItemID = "";
        private string photoPath = "";

        ItemPricing newItem = new ItemPricing();


        public AddItemPage(int userID)
        {
            InitializeComponent();
            userIdentification = userID;
        }

        private void AddItemPage_Load(object sender, EventArgs e)
        {
            getNewItemID();
            getMaxDiamondID();

            newItem.DBConnection = DBConnection;
            newItem.userIdentification = userIdentification;

        }

        private void addItemButton_Click(object sender, EventArgs e)
        {
              if (typeValid && purityValid)
              {

                  addItemInfo();

              //   if (viewList.Rows.Count > 0)
                // {
                  //   addDiamonds();
                // }
                //this.DialogResult = DialogResult.Yes;
              }
              else
              {
                    CancelEventArgs ex = new CancelEventArgs();
                    typeComboBox_Validating(sender, ex);
                    purityComboBox_Validating(sender, ex);
              }
        }

        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }

        private void addDiamondButton_Click(object sender, EventArgs e)
```

```
        {
            AddDiamond addDiamondForm = new AddDiamond();
            addDiamondForm.newItem = newItem;
            addDiamondForm.ShowDialog();

            if (addDiamondForm.DialogResult == DialogResult.Yes)
            {
                diamondLabel.Visible = true;
                viewList.Visible = true;

                string[] rowData = addDiamondForm.DiamondString.Split(';');
                viewList.Rows.Add(rowData);
            }
        }



        private void typeComboBox_SelectedValueChanged(object sender,
EventArgs e)
        {
            if (String.Equals(typeComboBox.Text, "Gold"))
            {
                purityLabel.Visible = true;
                purityComboBox.Visible = true;
            }
            else
            {
                purityLabel.Visible = false;
                purityComboBox.Visible = false;
            }
        }

        private void photoButton_Click(object sender, EventArgs e)
        {
            Image File;

            OpenFileDialog dialog = new OpenFileDialog();

            dialog.Filter = "Image files (*.jpg, *.png) | *.jpg; *.png";

            if (dialog.ShowDialog() == DialogResult.OK)
            {
                File = Image.FromFile(dialog.FileName);
                pictureBox.Image = File;
                pictureBox.Visible = true;

                photoPath = dialog.FileName.Trim();
            }
        }

        private void viewList_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            var columnIndex = 4;
```

```
    if (e.ColumnIndex == columnIndex)
    {
        // If the user presses this button, then delete row
        viewList.Rows.RemoveAt(e.RowIndex);
    }

    if (!(viewList.Rows.Count > 0))
    {
        diamondLabel.Visible = false;
        viewList.Visible = false;
    }
}

private void getMaxDiamondID()
{
    try
    {
        int maxNum = 0;

        SqlCommand cmdGetMaxDiamondID = DBConnection.CreateCommand();

        cmdGetMaxDiamondID.CommandText = @"SELECT Diamond_ID
                                           FROM Diamond";

        SqlDataReader rdr = cmdGetMaxDiamondID.ExecuteReader();


        while (rdr.Read())
        {
            String maxString = rdr[0].ToString();
            maxString = maxString.Remove(maxString.Length - 1);

            if (int.Parse(maxString) > maxNum)
            {
                maxNum = int.Parse(maxString);
            }
        }

        rdr.Close();

        String newString = (maxNum + 1).ToString() + 'D';

        newestID = newString;
        Console.WriteLine(newestID);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void getNewItemID()
{
    try
    {
```

```
            int maxNum = 0;

            SqlCommand getNewItemID = DBConnection.CreateCommand();

            getNewItemID.CommandText = @"SELECT ItemID
                                         FROM Item";

            SqlDataReader rdr = getNewItemID.ExecuteReader();

            if (rdr.Read())
            {
                while (rdr.Read())
                {
                    String maxString = rdr[0].ToString();
                    maxString = maxString.Remove(maxString.Length - 1);

                    if (int.Parse(maxString) > maxNum)
                    {
                        maxNum = int.Parse(maxString);
                    }

                }
            }
            else
            {
                maxNum = 0;
            }

            rdr.Close();

            tempItemID = (maxNum + 1).ToString();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }



    private void addItemInfo()
    {
        int itemMetal;
        int goldPurity = 0;

        if (viewList.Rows.Count > 0)
        {
            tempItemID = tempItemID + 'B';
            Console.WriteLine(tempItemID);
        }
        else
        {
            tempItemID = tempItemID + 'A';
            Console.WriteLine(tempItemID);
        }
```

```
            if (String.Equals(typeComboBox.Text, "Gold"))
            {
                itemMetal = 1;
                string purityString = purityComboBox.Text;
                purityString = purityString.Remove(purityString.Length - 1);
                goldPurity = Int32.Parse(purityString);
                Console.WriteLine(goldPurity);
            }
            else if (String.Equals(typeComboBox.Text, "Silver"))
            {
                itemMetal = 2;
            }
            else
            {
                itemMetal = 3;
            }

            if (String.Equals(descriptionTextBox.Text, "Enter Item
Description"))
            {
                descriptionTextBox.Text = "";
            }

            if (viewList.Rows.Count > 0)
            {
                newItem.createItem(tempItemID, itemMetal, goldPurity,
Decimal.ToDouble(weightSpinner.Value), descriptionTextBox.Text.Trim(),
photoPath, true);
            }
            else
            {
                newItem.createItem(tempItemID, itemMetal, goldPurity,
Decimal.ToDouble(weightSpinner.Value), descriptionTextBox.Text.Trim(),
photoPath, false);
            }

            // Add Each Diamond
            if (viewList.Rows.Count > 0)
            {
                string tempCert;

                foreach (DataGridViewRow row in viewList.Rows)
                {
                    newItem.createDiamond(newestID, tempItemID,
row.Cells[0].Value.ToString().Trim(),
char.Parse(row.Cells[1].Value.ToString()),
row.Cells[2].Value.ToString().Trim(),
double.Parse(row.Cells[3].Value.ToString()));
                    incrementID();
                }
            }
            else
            {
                diamondLabel.Visible = false;
```

```
                viewList.Visible = false;
            }


            //Updating Item and User
            newItem.updateNewItem(tempItemID);

            this.DialogResult = DialogResult.Yes;
        }

        private void incrementID()
        {
            int maxNum = 0;
            string maxString = newestID.Remove(newestID.Length - 1);
            maxNum = int.Parse(maxString);
            newestID = (maxNum + 1).ToString() + 'D';
        }

        private void purityComboBox_Validating(object sender, CancelEventArgs
e)
        {
            if (purityLabel.Visible == true)
            {
                if (purityComboBox.Text == string.Empty)
                {
                    errorProvider1.SetError(purityComboBox, "Please Select
Metal Purity");
                    purityValid = false;
                }
                else
                {
                    errorProvider1.SetError(purityComboBox, "");
                    purityValid = true;
                }
            }
        }

        private void typeComboBox_Validating(object sender, CancelEventArgs e)
        {
            purityValid = true;

            if (typeComboBox.Text == string.Empty)
            {
                errorProvider1.SetError(typeComboBox, "Please Select Metal
Type");
                typeValid = false;
            }
            else
            {
                errorProvider1.SetError(typeComboBox, "");
                typeValid = true;
            }
        }
    }
}
```

App.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add
name="WhatsItWorth.Properties.Settings.WhatsItWorthConnectionString"
connectionString="Data Source=DESKTOP-CTR7V1Q;Initial
Catalog=WhatsItWorth;Integrated
Security=True;Encrypt=True;TrustServerCertificate=True"
providerName="System.Data.SqlClient" />
        <add
name="WhatsItWorth.Properties.Settings.WhatsItWorthConnectionStringV1"
connectionString="Data Source=172.31.11.178,1433;Initial
Catalog=WhatsItWorth;Persist Security Info=True;User
ID=sa;Password=Benyamin1234$;TrustServerCertificate=True"
providerName="System.Data.SqlClient" />
        <add
name="WhatsItWorth.Properties.Settings.WhatsItWorthConnectionStringLaptop"
connectionString="Data Source=MSI;Initial Catalog=WhatsItWorth;Integrated
Security=True;Encrypt=False;TrustServerCertificate=True"
providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
    </startup>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe"
publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.SqlServer.Types"
publicKeyToken="89845dcd8080cc91" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-16.0.0.0" newVersion="16.0.0.0"
/>
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Text.Json"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-8.0.0.5" newVersion="8.0.0.5" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

ComparePage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using WhatsItWorth.WhatsItWorthDataSetTableAdapters;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Window;

namespace WhatsItWorth
{
    public partial class ComparePage : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;
        public ComparePage(int userID)
        {
            InitializeComponent();
            userIdentification = userID;
        }

        private void ComparePage_Load(object sender, EventArgs e)
        {

            compareTableAdapter firstItem = new compareTableAdapter();
            // TODO: This line of code loads data into the
'whatsItWorthDataSet.Item' table. You can move, or remove it, as needed.

this.compareTableAdapter.Fill(this.whatsItWorthDataSet.CompareTable,
userIdentification);

this.compareTable2Adapter.Fill(this.whatsItWorthDataSet.CompareTable2,
userIdentification);
        }
        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs
e)
        {
            try
            {
                SqlCommand cmdLoadInfo = DBConnection.CreateCommand();

                cmdLoadInfo.CommandText = @"SELECT Metal_Type, Metal_Purity,
Total_Weight, Description, Date_Added, Original_Price, Current_Price,
Price_of_Diamonds, Photo_Link
                                            FROM Item
                                            WHERE ItemID = @ii";
                cmdLoadInfo.Parameters.AddWithValue("@ii", comboBox1.Text);
```

```
                SqlDataReader rdr = cmdLoadInfo.ExecuteReader();

                if (rdr.Read())
                {

                    if (int.Parse(rdr[0].ToString()) == 1)
                    {
                        metalTextBox1.Text = "Gold";
                        purityLabel1.Visible = true;
                        purityTextBox1.Visible = true;
                        purityTextBox1.Text = rdr[1].ToString();
                    }
                    else if (int.Parse(rdr[0].ToString()) == 2)
                    {
                        metalTextBox1.Text = "Silver";
                    }
                    else if (int.Parse(rdr[0].ToString()) == 3)
                    {
                        metalTextBox1.Text = "Platinum";
                    }

                    totalTextBox1.Text = rdr[2].ToString();
                    descriptionTextBox1.Text = rdr[3].ToString();

                    dateTextBox1.Text = String.Format("{0:d}", rdr[4]);
                    originalTextBox1.Text = String.Format("{0:C}", rdr[5]);
                    currentTextBox1.Text = String.Format("{0:C}", rdr[6]);

                    double originalPrice = double.Parse(rdr[5].ToString());
                    double currentPrice = double.Parse(rdr[6].ToString());
                    double change = ((currentPrice - originalPrice) /
originalPrice) * 100;

                    if (currentPrice > originalPrice)
                    {
                        changeTextBox1.ForeColor = Color.LawnGreen;
                    }
                    else if (currentPrice < originalPrice)
                    {
                        changeTextBox1.ForeColor = Color.Red;
                    }
                    else
                    {
                        changeTextBox1.ForeColor = Color.Black;
                    }

                    changeTextBox1.Text = (change / 100).ToString("P1");

                    if (rdr[7].ToString() != "")
                    {
                        viewList1.Visible = true;
                    }
                    else
                    {
```

```
                        viewList1.Visible = false;
                        viewList1.Refresh();
                    }

                    if (rdr[8].ToString() != "")
                    {
                        try
                        {
                            itemImageBox1.Image = new
Bitmap(rdr[8].ToString().Trim());
                            itemImageBox1.Visible = true;
                        }
                        catch (Exception exception)
                        {
                            Console.WriteLine(exception.Message);
                        }

                    }
                    else
                    {
                        itemImageBox1.Visible = false;
                        itemImageBox1.Refresh();
                    }

                }


this.diamondTableAdapter.Fill(this.whatsItWorthDataSet.DiamondTable,
comboBox1.Text);


                    rdr.Close();

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

            private void comboBox2_SelectedIndexChanged(object sender, EventArgs
e)
            {
                try
                {
                    SqlCommand cmdLoadInfo = DBConnection.CreateCommand();

                    cmdLoadInfo.CommandText = @"SELECT Metal_Type, Metal_Purity,
Total_Weight, Description, Date_Added, Original_Price, Current_Price,
Price_of_Diamonds, Photo_Link
                                                FROM Item
                                                WHERE ItemID = @ii";
                    cmdLoadInfo.Parameters.AddWithValue("@ii", comboBox2.Text);

                    SqlDataReader rdr = cmdLoadInfo.ExecuteReader();
```

```
                if (rdr.Read())
                {

                    if (int.Parse(rdr[0].ToString()) == 1)
                    {
                        metalTextBox2.Text = "Gold";
                        purityLabel2.Visible = true;
                        purityTextBox2.Visible = true;
                        purityTextBox2.Text = rdr[1].ToString();
                    }
                    else if (int.Parse(rdr[0].ToString()) == 2)
                    {
                        metalTextBox2.Text = "Silver";
                    }
                    else if (int.Parse(rdr[0].ToString()) == 3)
                    {
                        metalTextBox2.Text = "Platinum";
                    }

                    totalTextBox2.Text = rdr[2].ToString();
                    descriptionTextBox2.Text = rdr[3].ToString();

                    dateTextBox2.Text = String.Format("{0:d}", rdr[4]);
                    originalTextBox2.Text = String.Format("{0:C}", rdr[5]);
                    currentTextBox2.Text = String.Format("{0:C}", rdr[6]);

                    double originalPrice = double.Parse(rdr[5].ToString());
                    double currentPrice = double.Parse(rdr[6].ToString());
                    double change = ((currentPrice - originalPrice) /
originalPrice) * 100;

                    if (currentPrice > originalPrice)
                    {
                        changeTextBox2.ForeColor = Color.LawnGreen;
                    }
                    else if (currentPrice < originalPrice)
                    {
                        changeTextBox2.ForeColor = Color.Red;
                    }
                    else
                    {
                        changeTextBox2.ForeColor = Color.Black;
                    }

                    changeTextBox2.Text = (change / 100).ToString("P1");

                    if (rdr[7].ToString() != "")
                    {
                        viewList2.Visible = true;
                    }
                    else
                    {
                        viewList2.Visible = false;
                        viewList2.Refresh();
```

```
                    }

                    if (rdr[8].ToString() != "")
                    {
                        try
                        {
                            itemImageBox2.Image = new
Bitmap(rdr[8].ToString().Trim());
                            itemImageBox2.Visible = true;
                        }
                        catch (Exception exception)
                        {
                            Console.WriteLine(exception.Message);
                        }

                    }
                    else
                    {
                        itemImageBox2.Visible = false;
                        itemImageBox2.Refresh();
                    }

                }


this.diamondTable2Adapter.Fill(this.whatsItWorthDataSet.DiamondTable2,
comboBox2.Text);


                rdr.Close();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

CompleteUpdater.cs:

```
using System;
using System.Net;
using System.Net.Http;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using HtmlAgilityPack;
using Newtonsoft.Json.Linq;
```

```
namespace WhatsItWorth
{
    class completeUpdater
    {
        public SqlConnection DBConnection;


        public void checkDateFunction()
        {
            try
            {
                SqlCommand cmdGetDate = DBConnection.CreateCommand();

                cmdGetDate.CommandText = @"SELECT Data_Updated, Metal_Type
                                           FROM MetalPrice";

                SqlDataReader rdr = cmdGetDate.ExecuteReader();

                List<int> metalUpdateInts = new List<int>();

                while (rdr.Read())
                {
                    DateTime dt = DateTime.Parse(rdr[0].ToString());

                    if (DateTime.UtcNow.Date != dt.Date)
                    {
                        Console.WriteLine("Update: " + rdr[1].ToString());
                        metalUpdateInts.Add((int)rdr[1]);
                    }
                    else
                    {
                        Console.WriteLine("Do Nothing: " + rdr[1].ToString());
                    }
                }

                rdr.Close();

                if (metalUpdateInts.Count > 0)
                {
                    updateDateFunction(metalUpdateInts);
                }
                else
                {
                    Console.WriteLine("Everything Remains Same");
                }

                metalUpdateInts.Clear();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void updateDateFunction(List<int> metalType)
```

```
        {

            foreach (int metalTypeInt in metalType)
            {

                try
                {
                    SqlCommand cmdUpdateDate = DBConnection.CreateCommand();
                    cmdUpdateDate.CommandText = @"UPDATE MetalPrice
                                                  SET Data_Updated = @NewDate
                                                  WHERE Metal_Type = @mt";
                    cmdUpdateDate.Parameters.AddWithValue("@NewDate",
DateTime.UtcNow.Date.ToString("yyyy-MM-dd"));
                    cmdUpdateDate.Parameters.AddWithValue("@mt",
metalTypeInt);

                    cmdUpdateDate.ExecuteNonQuery();

                    if (metalTypeInt == 1)
                    {
                        updateGoldPrice();
                    }
                    else if (metalTypeInt == 2)
                    {
                        updateSilverPrice();
                    }
                    else if (metalTypeInt == 3)
                    {
                        updatePlatinumPrice();
                    }

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

        }
        private void updateGoldPrice()
        {
            try
            {
                // initialize the HAP HTTP client
                var web = new HtmlWeb();

                // connect to target page
                var kitcoPath =
web.Load("https://www.kitco.com/price/precious-metals");
                var goldName =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[2]/div/ul/li[1]/div/span[1]/a").First().InnerText;
                var goldPrice =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[3]/div/div[3]/ul/li[1]/div/span[4]").First().InnerText;
```

```
//*[@id=\"__next\"]/main/div[1]/div[2]/div[3]/div/div[3]/ul/li[1]/div/span[4]
                SqlCommand cmdUpdateGold = DBConnection.CreateCommand();
                cmdUpdateGold.CommandText = @"UPDATE MetalPrice
                                              SET Price_per_Ounce = @NewPrice
                                                  WHERE Metal_Type = 1";
                cmdUpdateGold.Parameters.AddWithValue("@NewPrice",
Double.Parse(goldPrice));
                cmdUpdateGold.ExecuteNonQuery();

                Console.WriteLine("Updated: ");
                Console.WriteLine("{");
                Console.WriteLine("\t" + goldName + ": " +
String.Format("{0:C}", Double.Parse(goldPrice)));
                Console.WriteLine("\tPrice per gram: " +
String.Format("{0:C}", Double.Parse(goldPrice) / 31.1));
                Console.WriteLine("}");
                Console.WriteLine();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void updateSilverPrice()
        {
            try
            {
                // initialize the HAP HTTP client
                var web = new HtmlWeb();

                // connect to target page
                var kitcoPath =
web.Load("https://www.kitco.com/price/precious-metals");
                var silverName =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[2]/div/ul/li[2]/div/span[1]/a").First().InnerText;
                var silverPrice =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[2]/div/ul/li[2]/div/span[2]").First().InnerText;

                SqlCommand cmdUpdateSilver = DBConnection.CreateCommand();
                cmdUpdateSilver.CommandText = @"UPDATE MetalPrice
                                              SET Price_per_Ounce = @NewPrice
                                              WHERE Metal_Type = 2";
                cmdUpdateSilver.Parameters.AddWithValue("@NewPrice",
Double.Parse(silverPrice));
                cmdUpdateSilver.ExecuteNonQuery();


                Console.WriteLine("Updated: ");
                Console.WriteLine("{");
```

```
                Console.WriteLine("\t" + silverName + ": " +
String.Format("{0:C}", Double.Parse(silverPrice)));
                Console.WriteLine("\tPrice per gram: " +
String.Format("{0:C}", Double.Parse(silverPrice) / 31.1));
                Console.WriteLine("}");
                Console.WriteLine();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void updatePlatinumPrice()
        {
            try
            {
                // initialize the HAP HTTP client
                var web = new HtmlWeb();

                // connect to target page
                var kitcoPath =
web.Load("https://www.kitco.com/price/precious-metals");
                var platinumName =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[2]/div/ul/li[3]/div/span[1]/a").First().InnerText;
                var platinumPrice =
kitcoPath.DocumentNode.SelectNodes("//*[@id=\"__next\"]/main/div[1]/div[2]/div
[2]/div/ul/li[3]/div/span[2]").First().InnerText;

                SqlCommand cmdUpdatePlatinum = DBConnection.CreateCommand();
                cmdUpdatePlatinum.CommandText = @"UPDATE MetalPrice
                                                  SET Price_per_Ounce =
@NewPrice
                                                  WHERE Metal_Type = 3";
                cmdUpdatePlatinum.Parameters.AddWithValue("@NewPrice",
Double.Parse(platinumPrice));
                cmdUpdatePlatinum.ExecuteNonQuery();

                Console.WriteLine("Updated: ");
                Console.WriteLine("{");
                Console.WriteLine("\t" + platinumName + ": " +
String.Format("{0:C}", Double.Parse(platinumPrice)));
                Console.WriteLine("\tPrice per gram: " +
String.Format("{0:C}", Double.Parse(platinumPrice) / 31.1));
                Console.WriteLine("}");
                Console.WriteLine();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }
```

```
    }
}
```

CreateAccountPage.cs:

```
using System;
using System.ComponentModel;
using System.Data.SqlClient; //to use SQLCommand
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class CreateAccountPage : Form
    {
        public SqlConnection DBConnection;
        public int newUserID;
        private bool nameValid, emailValid, passValid, confirmValid;

        public CreateAccountPage()
        {
            InitializeComponent();
        }

        private void CreateAccountPage_Load(object sender, EventArgs e)
        {
            try
            {
                SqlCommand cmdGetLatestID = DBConnection.CreateCommand();

                cmdGetLatestID.CommandText = @"SELECT MAX(userID)
                                               FROM [User]";

                SqlDataReader rdr = cmdGetLatestID.ExecuteReader();

                if (rdr.Read())
                {
                    newUserID = (int)rdr[0] + 1;

                }

                rdr.Close();

                Console.WriteLine(newUserID);

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

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

```
            if (nameValid && emailValid && passValid && confirmValid)
            {
                try
                {
                    SqlCommand cmdAddUser = DBConnection.CreateCommand();
                    cmdAddUser.CommandText = @"INSERT INTO [User]
                                                VALUES (@nuid, @uemail, @uname,
@upass, 0, 0.00, @date)";
                    cmdAddUser.Parameters.AddWithValue("@nuid", newUserID);
                    cmdAddUser.Parameters.AddWithValue("@uemail",
emailTextBox.Text);
                    cmdAddUser.Parameters.AddWithValue("@uname",
nameTextBox.Text);
                    cmdAddUser.Parameters.AddWithValue("@upass",
Encrypt_Decrypt.EncryptString(passwordTextBox.Text));
                    cmdAddUser.Parameters.AddWithValue("@date",
DateTime.UtcNow.Date.ToString("yyyy-MM-dd"));


                    cmdAddUser.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

                this.DialogResult = DialogResult.OK;
            }
        }

        private void visibleButton_Click(object sender, EventArgs e)
        {
            visibleButton2.Visible = false;
            passwordTextBox.UseSystemPasswordChar = false;
            unvisibleButton2.Visible = true;
        }

        private void unvisibleButton_Click(object sender, EventArgs e)
        {
            unvisibleButton2.Visible = false;
            passwordTextBox.UseSystemPasswordChar = true;
            visibleButton2.Visible = true;
        }

        private void nameTextBox_Validating(object sender, CancelEventArgs e)
        {
            if (nameTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(nameTextBox, "Please Enter Name");
                nameValid = false;
            }
            else
            {
                errorProvider1.SetError(nameTextBox, "");
                nameValid = true;
```

```
            }
        }

        private void emailTextBox_Validating(object sender, CancelEventArgs e)
        {
            if (emailTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(emailTextBox, "Please Enter User
Email");
                emailValid = false;
            }
            else if (InputFormats.EmailValidation(emailTextBox.Text) != true)
            {
                errorProvider1.SetError(emailTextBox, "Email format is not
valid");
                emailValid = false;
            }
            else
            {
                errorProvider1.SetError(emailTextBox, "");
                emailValid = true;
            }
        }

        private void passwordTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (passwordTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(passwordTextBox, "Please Enter
Password");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 1)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 upper case letter");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 2)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 lower case letter");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 3)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 special character");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 4)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 number");
```

```
                passValid = false;
            }
            else if (passwordTextBox.Text.Length < 8)
            {
                errorProvider1.SetError(passwordTextBox, "Password must have
at least 8 characters");
                passValid = false;
            }
            else
            {
                errorProvider1.SetError(passwordTextBox, "");
                passValid = true;
            }
        }

        private void confirmTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (confirmTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(confirmTextBox, "Please Confirm
Password");
                confirmValid = false;
            }
            else if (!String.Equals(passwordTextBox.Text,
confirmTextBox.Text))
            {
                errorProvider1.SetError(confirmTextBox, "Password does not
match");
                confirmValid = false;
            }
            else
            {
                errorProvider1.SetError(confirmTextBox, "");
                confirmValid = true;
            }
        }

    }
}
```

DeleteAccountConfirmation.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
```

```
{
    public partial class DeleteAccountConfirmation : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;
        private bool passValid;

        public DeleteAccountConfirmation(int userID)
        {
            InitializeComponent();
            userIdentification = userID;
        }

        private void CreateAccountPage_Load(object sender, EventArgs e)
        {

        }

        private void deleteButton_Click(object sender, EventArgs e)
        {
            SqlCommand cmdLoadPass = DBConnection.CreateCommand();

            cmdLoadPass.CommandText = @"SELECT User_pass
                                        FROM [User]
                                        WHERE UserID = @ui";
            cmdLoadPass.Parameters.AddWithValue("@ui", userIdentification);

            SqlDataReader rdr = cmdLoadPass.ExecuteReader();

            if (rdr.Read())
            {

                if
(String.Equals(Encrypt_Decrypt.DecryptString(rdr[0].ToString().Trim()),
passwordTextBox.Text))
                {
                    this.DialogResult = DialogResult.Yes;
                }
                else
                {
                    incorrectLabel.Visible = true;
                }
            }

            rdr.Close();
        }

        private void visibleButton_Click(object sender, EventArgs e)
        {
            visibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = false;
            unvisibleButton.Visible = true;
        }

        private void unvisibleButton_Click(object sender, EventArgs e)
```

```
        {
            unvisibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = true;
            visibleButton.Visible = true;
        }

        private void passwordTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (passwordTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(passwordTextBox, "Please Enter
Password");
                passValid = false;
            }
            else
            {
                errorProvider1.SetError(passwordTextBox, "");
                passValid = true;
            }
        }
    }
}
```

DeleteItemConfirmation.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class DeleteItemConfirmation : Form
    {
        public DeleteItemConfirmation()
        {
            InitializeComponent();
        }

        private void CreateAccountPage_Load(object sender, EventArgs e)
        {

        }

        private void deleteButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Yes;
        }
```

```
    }
}
```

EditAccountConfirmation.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class EditAccountConfirmation : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;
        private bool passValid;

        public EditAccountConfirmation(int userID)
        {
            InitializeComponent();
            userIdentification = userID;
        }

        private void EditAccountConfirmation_Load(object sender, EventArgs e)
        {

        }

        private void verifyButton_Click(object sender, EventArgs e)
        {
            SqlCommand cmdLoadPass = DBConnection.CreateCommand();

            cmdLoadPass.CommandText = @"SELECT User_pass
                                        FROM [User]
                                        WHERE UserID = @ui";
            cmdLoadPass.Parameters.AddWithValue("@ui", userIdentification);

            SqlDataReader rdr = cmdLoadPass.ExecuteReader();

            if (rdr.Read())
            {

                if
(String.Equals(Encrypt_Decrypt.DecryptString(rdr[0].ToString().Trim()),
passwordTextBox.Text))
                {
                    this.DialogResult = DialogResult.Yes;
```

```
                }
                else
                {
                    incorrectLabel.Visible = true;
                }
            }

            rdr.Close();
        }

        private void visibleButton_Click(object sender, EventArgs e)
        {
            visibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = false;
            unvisibleButton.Visible = true;
        }

        private void unvisibleButton_Click(object sender, EventArgs e)
        {
            unvisibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = true;
            visibleButton.Visible = true;
        }

        private void passwordTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (passwordTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(passwordTextBox, "Please Enter
Password");
                passValid = false;
            }
            else
            {
                errorProvider1.SetError(passwordTextBox, "");
                passValid = true;
            }
        }
    }
}
```

Encrypt_Decrypt.cs:

```
using System;

using System.IO;
using System.Security.Cryptography;
using System.Text;

namespace WhatsItWorth
{
    class Encrypt_Decrypt
    {
        public static string EncryptString(string plainText)
```

```
        {
            //StreamReader fileReader = new
StreamReader(@"C:\Users\Ben\OneDrive\School Work\Senior
Project\WhatsItWorth\Key.txt");  //Desktop
            StreamReader fileReader = new
StreamReader(@"..\Resources\Key.txt");  //Laptop
            String key = fileReader.ReadLine();

            byte[] iv = new byte[16];
            byte[] array;

            using (Aes aes = Aes.Create())
            {
                aes.Key = Encoding.UTF8.GetBytes(key);
                aes.IV = iv;

                ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key,
aes.IV);

                using (MemoryStream memoryStream = new MemoryStream())
                {
                    using (CryptoStream cryptoStream = new
CryptoStream((Stream)memoryStream, encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter streamWriter = new
StreamWriter((Stream)cryptoStream))
                        {
                            streamWriter.Write(plainText);
                        }

                        array = memoryStream.ToArray();
                    }
                }
            }

            return Convert.ToBase64String(array);
        }

        public static string DecryptString(string cipherText)
        {
            //StreamReader fileReader = new
StreamReader(@"C:\Users\Ben\OneDrive\School Work\Senior
Project\WhatsItWorth\Key.txt"); //Desktop
            StreamReader fileReader = new
StreamReader(@"..\Resources\Key.txt"); //Laptop
            String key = fileReader.ReadLine();

            byte[] iv = new byte[16];
            byte[] buffer = Convert.FromBase64String(cipherText);

            using (Aes aes = Aes.Create())
            {
                aes.Key = Encoding.UTF8.GetBytes(key);
                aes.IV = iv;
```

```
                ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key,
aes.IV);

                using (MemoryStream memoryStream = new MemoryStream(buffer))
                {
                    using (CryptoStream cryptoStream = new
CryptoStream((Stream)memoryStream, decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader streamReader = new
StreamReader((Stream)cryptoStream))
                        {
                            return streamReader.ReadToEnd();
                        }
                    }
                }
            }
        }
    }
}
```

GenerateReportPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class GenerateReportPage : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;

        public GenerateReportPage(int userID)
        {
            InitializeComponent();
            userIdentification = userID;
        }

        private void GenerateReportPage_Load(object sender, EventArgs e)
        {

this.totalReportTableAdapter.Fill(this.whatsItWorthDataSet.TotalReportTable,
userIdentification);

            this.reportViewer1.RefreshReport();
        }
```

```
        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }
    }
}
```

HomePage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
using System.Deployment.Application; //to use SQLCommand
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class HomePage : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;

        public HomePage(int userID)
        {
            InitializeComponent();
            userIdentification = userID;

        }

        private void HomePage_Load(object sender, EventArgs e)
        {
            try
            {
                //MessageBox.Show("Connection String: " +
DBConnection.ConnectionString, "Debug Info");

                SqlCommand cmdUpdateUser = DBConnection.CreateCommand();
                cmdUpdateUser.CommandText = @"UPDATE [User]
                                              SET Num_of_Items = (SELECT
COUNT(ItemID)

FROM Item

WHERE UserID = @ui)
                                              WHERE UserID = @ui";
```

```
                cmdUpdateUser.Parameters.AddWithValue("@ui",
userIdentification);
                cmdUpdateUser.ExecuteNonQuery();


                SqlCommand cmdLoadInfo = DBConnection.CreateCommand();

                /* For Updating Sum and Count (Just change with Update into
database)
                   @"SELECT User_name, SUM(Current_Price), COUNT(ItemID)
                   FROM [User] AS ui, Item AS it
                   WHERE ui.userID = @ui AND it.userID = @ui
                   GROUP BY User_name";
                 */

                cmdLoadInfo.CommandText = @"SELECT User_name, Total_Price,
Num_of_Items
                                            FROM [User]
                                            WHERE userID = @ui";
                cmdLoadInfo.Parameters.AddWithValue("@ui",
userIdentification);

                SqlDataReader rdr = cmdLoadInfo.ExecuteReader();

                if (rdr.Read())
                {
                    nameTextBox.Text = "Welcome, " + rdr[0].ToString();
                    totalValueTextBox.Text = String.Format("{0:C}", rdr[1]);
                    numItemsTextBox.Text = rdr[2].ToString();

                }

                rdr.Close();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            // TODO: This line of code loads data into the
'whatsItWorthDataSet.HomePageTable' table. You can move, or remove it, as
needed.

//this.homePageTableAdapter.Fill(this.whatsItWorthDataSet.HomePageTable,
userIdentification);

            try
            {

this.homePageTableAdapter.Fill(this.whatsItWorthDataSet.HomePageTable,
userIdentification);
            }
            catch (Exception ex)
            {
```

```
                MessageBox.Show("Error filling HomePageTable: " + ex.Message);
            }


            int columnIndex = 3;
            foreach (DataGridViewRow row in tableList.Rows)
            {
                double quantity;

                if (double.TryParse(row.Cells[columnIndex].Value.ToString(),
out quantity))
                {
                    if (quantity < 0)
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Red;
                    }
                    else if (quantity > 0)
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Green;
                    }
                    else
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Black;
                    }
                }
            }

        }


        private void addItemButton_Click(object sender, EventArgs e)
        {
            this.Hide();

            AddItemPage addForm = new AddItemPage(userIdentification);
            addForm.DBConnection = DBConnection;
            addForm.ShowDialog();

          //  if (addForm.DialogResult == DialogResult.Yes)
            //{
                this.Close();
                HomePage homeForm = new HomePage(userIdentification);
                homeForm.DBConnection = DBConnection;
                homeForm.ShowDialog();
           // }
           // else
            //{
              //  this.Show();
            //}
        }

        private void selectItemButton_Click(object sender, EventArgs e)
```

```
        {
            incorrectLabel.Visible = false;
            var columnIndex = 0;

            foreach (DataGridViewRow row in tableList.Rows)
            {

                var isChecked =
Convert.ToBoolean(row.Cells[columnIndex].Value);

                if (isChecked == true)
                {
                    incorrectLabel.Visible = false;
                    this.Hide();

                    ViewItem viewItemForm = new
ViewItem(tableList.Rows[row.Index].Cells[4].Value.ToString());
                    viewItemForm.DBConnection = DBConnection;
                    viewItemForm.ShowDialog();

                    //if (viewItemForm.DialogResult == DialogResult.Yes)
                    //{
                        this.Close();
                        HomePage homeForm = new HomePage(userIdentification);
                        homeForm.DBConnection = DBConnection;
                        homeForm.ShowDialog();
                    //}
                   // else
                   // {
                     //   this.Show();
                    //}

                    break;
                }
                else
                {
                    incorrectLabel.Visible = true;
                }
            }

        }

        private void yourProfileButton_Click(object sender, EventArgs e)
        {
            this.Hide();

            ViewAccount userAccountForm = new ViewAccount(userIdentification);
            userAccountForm.DBConnection = DBConnection;
            userAccountForm.ShowDialog();

            if (userAccountForm.DialogResult == DialogResult.Yes)
            {
                this.DialogResult = DialogResult.Yes;
            }
            else
```

```
            {
                this.Close();
                HomePage homeForm = new HomePage(userIdentification);
                homeForm.DBConnection = DBConnection;
                homeForm.ShowDialog();
            }
        }

        private void resourcesButton_Click(object sender, EventArgs e)
        {
            try
            {
                // Create ProcessStartInfo with the URL
                ProcessStartInfo psi = new ProcessStartInfo
                {
                    FileName = "https://bebadinia.github.io",
                    UseShellExecute = true // Required for launching URLs in
default browser
                };

                // Start the process
                Process.Start(psi);
            }
            catch (System.ComponentModel.Win32Exception noBrowser)
            {
                MessageBox.Show("Could not find a web browser to open the
resources page. Please check your default browser settings.",
                    "Browser Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"An error occurred while trying to open the
resources page: {ex.Message}",
                    "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
        }


        private void compareItemsButton_Click(object sender, EventArgs e)
        {
            this.Hide();

            ComparePage compareForm = new ComparePage(userIdentification);
            compareForm.DBConnection = DBConnection;
            compareForm.ShowDialog();

            this.Show();
        }

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

```
            this.DialogResult = DialogResult.OK;
        }

        private void generateReportButton_Click(object sender, EventArgs e)
        {
            this.Hide();

            GenerateReportPage reportForm = new
GenerateReportPage(userIdentification);
            reportForm.DBConnection = DBConnection;
            reportForm.ShowDialog();


            this.Close();
            HomePage homeForm = new HomePage(userIdentification);
            homeForm.DBConnection = DBConnection;
            homeForm.ShowDialog();
            //this.Show();
        }

        private void tableList_ValueChanged(object sender,
DataGridViewCellEventArgs e)
        {
            var columnIndex = 0;
            if (e.ColumnIndex == columnIndex)
            {
                // If the user checked this box, then uncheck all the other
rows
                var isChecked = true;
//(bool)tableList.Rows[e.RowIndex].Cells[e.ColumnIndex].Value;

                if (isChecked)
                {
                    foreach (DataGridViewRow row in tableList.Rows)
                    {
                        if (row.Index != e.RowIndex)
                        {
                            row.Cells[columnIndex].Value = !isChecked;
                        }
                    }
                }
            }
        }

        private void tableList_DataBindingComplete(object sender,
DataGridViewBindingCompleteEventArgs e)
        {
            tableList.ClearSelection();
        }

        private void tableList_Sorted(object sender, EventArgs e)
        {
            int columnIndex = 3;
            foreach (DataGridViewRow row in tableList.Rows)
            {
```

```
                double quantity;

                if (double.TryParse(row.Cells[columnIndex].Value.ToString(),
out quantity))
                {
                    if (quantity < 0)
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Red;
                    }
                    else if (quantity > 0)
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Green;
                    }
                    else
                    {
                        row.Cells[columnIndex].Style.ForeColor =
System.Drawing.Color.Black;
                    }
                }
            }
        }
    }
}
```

InputFormats.cs:

```
using System;

using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Text.RegularExpressions;
using System.Net.Mail;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.ListView;

namespace WhatsItWorth
{
    class InputFormats
    {
        public static bool UsingRegex(string stringValue)
        {
            var pattern = @"^[0-9]+$";
            var regex = new Regex(pattern);
            return regex.IsMatch(stringValue);
        }


        public static bool EmailValidation(string emailString)
        {
            // ^[^@\s]+@[^@\s]+\.(com|net|org|gov)$
            try
            {
```

```
                var tryMail = new System.Net.Mail.MailAddress(emailString);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public static int PassValidation(string passString)
        {
            //string lowercase = "qwertyuiopasdfghjklzxcvbnm";
            //string uppercase = "QWERTYUIOPASDFGHJKLZXCVBNM";
            //string digits = "0123456789";
            //string specialChars = "!@#$%^&*()_+-=[]|{};:<>?,.";
            //string minLength = 8

            if(!passString.Any(char.IsUpper))
            {
                return 1;
            }
            else if(!passString.Any(char.IsLower))
            {
                return 2;
            }
            else if (!passString.Any(ch => !Char.IsLetterOrDigit(ch)))
            {
                return 3;
            }
            else if (!passString.Any(char.IsDigit))
            {
                return 4;
            }
            else
            {
                return 0;
            }
        }
    }
}
```

ItemPricing.cs:

```
using System;
using System.Net;
using System.Net.Http;
using System.Collections.Generic;
using System.Data.Common;
using System.Data.SqlClient;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
using HtmlAgilityPack;
using Newtonsoft.Json.Linq;
using RestSharp;
using System.Xml.Serialization;
using System.IO;

namespace WhatsItWorth
{
    [XmlRoot(ElementName = "pr")]
    public class DecryptPrice // Decrypting XML and price
    {

        [XmlElement(ElementName = "min")]
        public int Min { get; set; }

    }

    public class ItemPricing
    {
        public SqlConnection DBConnection;
        public int userIdentification;
        private double currentGold, currentSilver, currentPlatinum;
        Dictionary<string, double> itemList = new Dictionary<string,
double>();

        public void checkDate() // Public call which checks the date
        {
            bool updateItemsOption = false;
            try
            {
                //userIdentification = userID;

                SqlCommand cmdGetItemDate = DBConnection.CreateCommand();  //
Getting the item update date

                cmdGetItemDate.CommandText = @"SELECT Date_Updated
                                               FROM Item
                                               WHERE UserID = @ui";
                cmdGetItemDate.Parameters.AddWithValue("@ui",
userIdentification);;

                SqlDataReader rdr = cmdGetItemDate.ExecuteReader();

                while (rdr.Read())
                {
                    if (rdr[0].ToString() != "")
                    {
                        DateTime dt = DateTime.Parse(rdr[0].ToString());
                        if ((DateTime.UtcNow.Date != dt.Date))
                        {


                            Console.WriteLine("Update Items");
                            updateItemsOption = true;
                        }
```

```
                    else
                    {
                        Console.WriteLine("Do Nothing");
                    }

                }
                else
                {
                    Console.WriteLine("Update Items");
                    updateItemsOption = true;
                }


            }

            rdr.Close();

            if (updateItemsOption)
            {
                updateItems(true);
            }
            else
            {
                Console.WriteLine("Everything Remains Same");
            }

        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void updateItems(bool x) // Public call which updates rest of
pricing for item
    {
        try
        {
            SqlCommand cmdGetMetalInfo = DBConnection.CreateCommand();  //
Getting the metal type and the market price from database to store in local
variables

            cmdGetMetalInfo.CommandText = @"SELECT Metal_Type,
Price_per_Ounce
                                            FROM MetalPrice";

            SqlDataReader rdr = cmdGetMetalInfo.ExecuteReader();

            while (rdr.Read())
            {
                if ((int)rdr[0] == 1)
                {
                    currentGold = Double.Parse(rdr[1].ToString());
                }
                else if ((int)rdr[0] == 2)
```

```
                    {
                        currentSilver = Double.Parse(rdr[1].ToString());
                    }
                    else if ((int)rdr[0] == 3)
                    {
                        currentPlatinum = Double.Parse(rdr[1].ToString());
                    }

                }

                rdr.Close();

                if (x)
                {
                    updateMetalPrice(); // Call to update gold price
                    updateDiamondPrice(); // Call to update diamond price
                    updateTotalPrice(); // Call to update total price
                }

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }



        //Updating Metal Pricing
        private void updateMetalPrice() // Getting and Updating the metal
prices for each user item
        {
            try
            {

                SqlCommand cmdGetItemlInfo = DBConnection.CreateCommand(); //
Selecting all users items and finding the metal type

                cmdGetItemlInfo.CommandText = @"SELECT ItemID, Metal_Type,
Metal_Purity, Total_Weight, Price_of_Diamonds
                                                FROM Item
                                                WHERE UserID = @ui";
                cmdGetItemlInfo.Parameters.AddWithValue("@ui",
userIdentification);

                SqlDataReader rdr = cmdGetItemlInfo.ExecuteReader();

                while (rdr.Read())
                {
                    //Console.WriteLine(String.Format("{0:C}",
calculateMetalPrice((int)rdr[1], rdr[2].ToString(),
Double.Parse(rdr[3].ToString()))));
                    itemList.Add(rdr[0].ToString().Trim(),
calculateMetalPrice((int)rdr[1], rdr[2].ToString().Trim(),
Double.Parse(rdr[3].ToString())));
```

```
                }

                rdr.Close();


                foreach (KeyValuePair<string, double> s in itemList)
                {
                    Console.WriteLine("ID = {0} Metal Value = {1}", s.Key,
s.Value);

                    SqlCommand cmdUpdateMetalPrice =
DBConnection.CreateCommand(); // Updating metal price for item in Item
database based on price calculation from calculateMetalPrice

                    cmdUpdateMetalPrice.CommandText = @"UPDATE Item
                                                        SET Price_of_Metal =
@NewPrice
                                                        WHERE ItemID = @ii";
                    cmdUpdateMetalPrice.Parameters.AddWithValue("@NewPrice",
s.Value);
                    cmdUpdateMetalPrice.Parameters.AddWithValue("@ii", s.Key);
                    cmdUpdateMetalPrice.ExecuteNonQuery();
                }

                Console.WriteLine();
                itemList.Clear();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private double calculateMetalPrice(int type, string purity, double
weight) // Calculating and returning the metal price for each item
        {
            double metalPrice = 0;

            Console.WriteLine(currentGold);

            if (type == 1)
            {
                metalPrice = (((currentGold / 31.1) / 24) *
Int32.Parse(purity)) * weight;
            }
            else if (type == 2)
            {
                metalPrice = (currentSilver / 31.1) * weight;
            }
            else if (type == 3)
            {
                metalPrice = (currentPlatinum / 31.1) * weight;
            }
```

```
            Console.WriteLine(Math.Round(metalPrice, 2));

            return Math.Round(metalPrice, 2);
        }



        //Updating Diamond Pricing
        private void updateDiamondPrice() // Getting and Updating the diamond
prices for each user item
        {
            try
            {
                SqlCommand cmdGetDiamondlInfo = DBConnection.CreateCommand();
// Selecting all item diamonds information depending on user

                cmdGetDiamondlInfo.CommandText = @"SELECT D.Diamond_ID,
Diamond_Carat, Diamond_Color, Diamond_Clarity, Diamond_Cut
                                                   FROM Diamond AS D INNER
JOIN Item AS I ON D.ItemID = I.ItemID
                                                   WHERE UserID = @ui";
                cmdGetDiamondlInfo.Parameters.AddWithValue("@ui",
userIdentification);

                SqlDataReader rdr = cmdGetDiamondlInfo.ExecuteReader();

                while (rdr.Read())
                {
                    itemList.Add(rdr[0].ToString().Trim(),
calculateDiamondPrice(rdr[1].ToString().Trim(), rdr[2].ToString().Trim(),
rdr[3].ToString().Trim(), rdr[4].ToString().Trim()));

                }

                rdr.Close();

                foreach (KeyValuePair<string, double> s in itemList)
                {
                    Console.WriteLine("Diamond ID = {0} Diamond Value = {1}",
s.Key, s.Value);

                    if (s.Value != 0.0)
                    {
                        SqlCommand cmdUpdateDiamondPrice =
DBConnection.CreateCommand(); // Updating individual diamond price for each
diamond in Diamond database

                        cmdUpdateDiamondPrice.CommandText = @"UPDATE Diamond
                                                              SET Price_of_Diamond
= @NewPrice
                                                              WHERE Diamond_ID =
@di";

cmdUpdateDiamondPrice.Parameters.AddWithValue("@NewPrice", s.Value);
```

```
                        cmdUpdateDiamondPrice.Parameters.AddWithValue("@di",
s.Key);
                        cmdUpdateDiamondPrice.ExecuteNonQuery();
                    }
                    else
                    {
                        Console.WriteLine("Damond ID {0} did not update
value", s.Key);
                    }
                }

                Console.WriteLine();
                itemList.Clear();

                SqlCommand cmdUpdateDiamondTotal =
DBConnection.CreateCommand(); // Updating total diamond prices for all
diamonds in Item database

                cmdUpdateDiamondTotal.CommandText = @"UPDATE Item
                                                      SET Price_of_Diamonds =
(SELECT SUM(Price_of_Diamond)

FROM Diamond

WHERE Diamond.ItemID = Item.ItemID)
                                                      WHERE UserID = @ui";
                cmdUpdateDiamondTotal.Parameters.AddWithValue("@ui",
userIdentification);
                cmdUpdateDiamondTotal.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private double calculateDiamondPrice(string carat, string color,
string clarity, string cut) // Using API to calculate and return individual
diamond price
        {
            try
            {
                var options = new
RestClientOptions($"http://www.idexonline.com/DPService.asp?SID=4wp7go123jqtkd
yd5f2e&cut={cut}&carat={carat}&color={color}&clarity={clarity}");
                RestClient client = new RestClient(options);
                RestRequest request = new RestRequest("", Method.Get);
                RestResponse response = client.Execute(request);
                //Console.WriteLine(response.Content);

                XmlSerializer serializer = new
XmlSerializer(typeof(DecryptPrice));
                using (StringReader reader = new
StringReader(response.Content))
                {
```

```
                    var test = (DecryptPrice)serializer.Deserialize(reader);
                    return test.Min;
                }
            }
            catch (Exception ex)
            {
                return 0.0;
            }
        }



        //Updating Total Pricing
        private void updateTotalPrice() // Getting and Updating the total
price for item using the updated metal price and diamond price for each user
item
        {
            SqlCommand cmdCalculateItemPrices = DBConnection.CreateCommand();
// Selecting both prices

            cmdCalculateItemPrices.CommandText = @"SELECT ItemID,
Price_of_Metal, Price_of_Diamonds
                                                   FROM Item
                                                   WHERE UserID = @ui";
            cmdCalculateItemPrices.Parameters.AddWithValue("@ui",
userIdentification);

            SqlDataReader rdr = cmdCalculateItemPrices.ExecuteReader();

            while (rdr.Read())
            {
                if (rdr[2].ToString() != "")
                {
                    itemList.Add(rdr[0].ToString().Trim(),
(Double.Parse(rdr[1].ToString()) + Double.Parse(rdr[2].ToString()))); //
Adding both prices if item has diamonds
                }
                else
                {
                    itemList.Add(rdr[0].ToString().Trim(),
Double.Parse(rdr[1].ToString())); // Adding only metal price if item has no
diamonds
                }

            }

            rdr.Close();


            foreach (KeyValuePair<string, double> s in itemList)
            {
                Console.WriteLine("ID = {0} Total Price = {1}", s.Key,
s.Value);
```

```
                SqlCommand cmdUpdateTotalPrice = DBConnection.CreateCommand();
// Updating item databases total price

                cmdUpdateTotalPrice.CommandText = @"UPDATE Item
                                                    SET Current_Price =
@NewPrice, Date_Updated = @NewDate
                                                    WHERE ItemID = @ii";
                cmdUpdateTotalPrice.Parameters.AddWithValue("@NewPrice",
s.Value);
                cmdUpdateTotalPrice.Parameters.AddWithValue("@NewDate",
DateTime.UtcNow.Date.ToString("yyyy-MM-dd"));
                cmdUpdateTotalPrice.Parameters.AddWithValue("@ii", s.Key);
                cmdUpdateTotalPrice.ExecuteNonQuery();

            }

            itemList.Clear();

            SqlCommand cmdUpdateUserTotal = DBConnection.CreateCommand(); //
Updating total item prices for a user in User database

            cmdUpdateUserTotal.CommandText = @"UPDATE [User]
                                               SET Total_Price = (SELECT
SUM(Current_Price)

FROM Item

WHERE UserID = @ui)
                                               WHERE UserID = @ui";
            cmdUpdateUserTotal.Parameters.AddWithValue("@ui",
userIdentification);
            cmdUpdateUserTotal.ExecuteNonQuery();

        }


        public void createItem(string tempItemID,int itemMetal, int
goldPurity, double totalWeight, string description, string photoPath, bool
hasDiamonds)
        {
            updateItems(false);

            try
            {
                SqlCommand cmdAddItem = DBConnection.CreateCommand();
                cmdAddItem.CommandText = @"INSERT INTO Item (ItemID, UserID,
Metal_Type, Metal_Purity, Total_Weight, Price_of_Metal, Description,
Current_Price, Date_Added, Original_Price, Photo_Link)
                                           VALUES (@itemID, @userID,
@metalType, @metalPurity, @totalWeight, @priceMetal, @description,
@currentPrice, @date, @originalPrice, @photo)";

                cmdAddItem.Parameters.AddWithValue("@itemID", tempItemID);
                cmdAddItem.Parameters.AddWithValue("@userID",
userIdentification);
```

```
                cmdAddItem.Parameters.AddWithValue("@metalType", itemMetal);

                if (itemMetal == 1)
                {
                    cmdAddItem.Parameters.AddWithValue("@metalPurity",
goldPurity);
                }
                else
                {
                    cmdAddItem.Parameters.AddWithValue("@metalPurity",
DBNull.Value);
                }

                cmdAddItem.Parameters.AddWithValue("@totalWeight",
totalWeight);
                cmdAddItem.Parameters.AddWithValue("@priceMetal",
calculateMetalPrice(itemMetal, goldPurity.ToString(), totalWeight));

                if (!String.Equals(description, ""))
                {
                    cmdAddItem.Parameters.AddWithValue("@description",
description);
                }
                else
                {
                    string tempDescription;

                    if (itemMetal == 1)
                    {
                        if (hasDiamonds)
                        {
                            tempDescription = goldPurity.ToString() + "K Gold;
" + totalWeight.ToString() + "G with Diamonds";
                        }
                        else
                        {
                            tempDescription = goldPurity.ToString() + "K Gold;
" + totalWeight.ToString() + "G";
                        }
                    }
                    else if (itemMetal == 2)
                    {
                        if (hasDiamonds)
                        {
                            tempDescription = "Silver; " +
totalWeight.ToString() + "G with Diamonds";
                        }
                        else
                        {
                            tempDescription = "Silver; " +
totalWeight.ToString() + "G";
                        }
                    }
                    else
                    {
```

```
                        if (hasDiamonds)
                        {
                            tempDescription = "Platinum; " +
totalWeight.ToString() + "G with Diamonds";
                        }
                        else
                        {
                            tempDescription = "Platinum; " +
totalWeight.ToString() + "G";
                        }
                    }

                    cmdAddItem.Parameters.AddWithValue("@description",
tempDescription);
                }

                double tempPrice = calculateMetalPrice(itemMetal,
goldPurity.ToString(), totalWeight);

                cmdAddItem.Parameters.AddWithValue("@currentPrice",
tempPrice);
                cmdAddItem.Parameters.AddWithValue("@date",
DateTime.UtcNow.Date.ToString("yyyy-MM-dd"));
                cmdAddItem.Parameters.AddWithValue("@originalPrice",
tempPrice);

                if (!String.Equals(photoPath, ""))
                {
                    cmdAddItem.Parameters.AddWithValue("@photo", photoPath);
                }
                else
                {
                    cmdAddItem.Parameters.AddWithValue("@photo",
DBNull.Value);
                }


                cmdAddItem.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void createDiamond(string tempDiamondID, string tempItemID,
string cut, char color, string clarity, double carat)
        {
                try
                {
                    SqlCommand cmdAddDiamonds = DBConnection.CreateCommand();
                    cmdAddDiamonds.CommandText = @"INSERT INTO Diamond
(Diamond_ID, ItemID, Diamond_Carat, Diamond_Color, Diamond_Clarity,
Diamond_Cut, Price_of_Diamond)
```

```
                                                    VALUES (@ndid, @niid,
@carat, @color, @clarity, @cut, @dprice)";
                    cmdAddDiamonds.Parameters.AddWithValue("@ndid",
tempDiamondID);
                    cmdAddDiamonds.Parameters.AddWithValue("@niid",
tempItemID);
                    cmdAddDiamonds.Parameters.AddWithValue("@carat", carat);
                    cmdAddDiamonds.Parameters.AddWithValue("@color", color);
                    cmdAddDiamonds.Parameters.AddWithValue("@clarity",
clarity);
                    cmdAddDiamonds.Parameters.AddWithValue("@cut", cut);
                    cmdAddDiamonds.Parameters.AddWithValue("@dprice",
calculateDiamondPrice(carat.ToString(), color.ToString(), clarity, cut));

                    cmdAddDiamonds.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

        }

        public bool getDiamondPrice(string cut, string color, string clarity,
string carat)
        {
            Console.WriteLine();
            Console.WriteLine(calculateDiamondPrice(carat, color, clarity,
cut));

            if (calculateDiamondPrice(carat, color, clarity, cut) != 0.0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //Updating New Item
        public void updateNewItem(string itemID) // Getting and Updating the
total price for item using the updated metal price and diamond price for each
user item
        {
            double tempPrice = 0;

            try
            {
                SqlCommand cmdUpdateDiamondTotal =
                        DBConnection.CreateCommand(); // Updating diamond
price for all diamond in Item database

                cmdUpdateDiamondTotal.CommandText = @"UPDATE Item
```

```
                                                    SET Price_of_Diamonds =
(SELECT SUM(Price_of_Diamond)

FROM Diamond

WHERE Diamond.ItemID = Item.ItemID)
                                                    WHERE ItemID = @ii";
                cmdUpdateDiamondTotal.Parameters.AddWithValue("@ii", itemID);
                cmdUpdateDiamondTotal.ExecuteNonQuery();


                SqlCommand cmdNewItemPrices = DBConnection.CreateCommand(); //
Selecting both prices

                cmdNewItemPrices.CommandText = @"SELECT Price_of_Metal,
Price_of_Diamonds
                                                FROM Item
                                                WHERE ItemID = @ii";
                cmdNewItemPrices.Parameters.AddWithValue("@ii", itemID);

                SqlDataReader rdr = cmdNewItemPrices.ExecuteReader();

                while (rdr.Read())
                {
                    if (rdr[1].ToString() != "")
                    {
                        tempPrice = Double.Parse(rdr[0].ToString()) +
                                    Double.Parse(rdr[1].ToString()); // Adding
both prices if item has diamonds
                    }
                    else
                    {
                        tempPrice = Double.Parse(rdr[0].ToString()); // Adding
only metal price if item has no diamonds
                    }

                }

                rdr.Close();


                SqlCommand cmdUpdateItemPrice = DBConnection.CreateCommand();
// Updating item databases total price

                cmdUpdateItemPrice.CommandText = @"UPDATE Item
                                                  SET Current_Price =
@NewPrice, Original_Price = @NewPrice, Date_Updated = @NewDate
                                                  WHERE ItemID = @ii";
                cmdUpdateItemPrice.Parameters.AddWithValue("@NewPrice",
tempPrice);
                cmdUpdateItemPrice.Parameters.AddWithValue("@NewDate",
DateTime.UtcNow.Date.ToString("yyyy-MM-dd"));
                cmdUpdateItemPrice.Parameters.AddWithValue("@ii", itemID);
                cmdUpdateItemPrice.ExecuteNonQuery();
```

```
                SqlCommand
                    cmdUpdateUserTotal =
                        DBConnection.CreateCommand(); // Updating total item
prices for a user in User database

                cmdUpdateUserTotal.CommandText = @"UPDATE [User]
                                                   SET Total_Price = (SELECT
SUM(Current_Price)

FROM Item

WHERE UserID = @ui)
                                                   WHERE UserID = @ui";
                cmdUpdateUserTotal.Parameters.AddWithValue("@ui",
userIdentification);
                cmdUpdateUserTotal.ExecuteNonQuery();
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

LogInPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Diagnostics.Eventing.Reader;
using HtmlAgilityPack;
using System.Text.RegularExpressions;
using System.Data.Common;

namespace WhatsItWorth
{
    public partial class LogInPage : Form
    {
        private bool idValid, passValid;


        public LogInPage()
        {
            InitializeComponent();
        }
```

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

            try
            {
                sqlConnection.Open();
                MessageBox.Show("DB Connected", "What's It Worth?");

                completeUpdater Updater = new completeUpdater();
                Updater.DBConnection = sqlConnection;
                Updater.checkDateFunction();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }




        private void loginButton_Click(object sender, EventArgs e)
        {
            incorrectLabel.Visible = false;

            if (idValid && passValid)
            {
                try
                {
                    //To Encrypt Password Manually:
Console.WriteLine(Encrypt_Decrypt.EncryptString("pass5678"));
                    // <-- Open
                    SqlCommand cmdCheckLogin = sqlConnection.CreateCommand();

                    cmdCheckLogin.CommandText = @"SELECT User_pass, UserID
                                                  FROM [User]
                                                  WHERE User_email = @email";

                    cmdCheckLogin.Parameters.AddWithValue("@email",
emailTextBox.Text);

                    SqlDataReader rdr = cmdCheckLogin.ExecuteReader();

                    if (rdr.Read())
                    {

                        int userID = (int)rdr[1];

                        if
(String.Equals(Encrypt_Decrypt.DecryptString(rdr[0].ToString().Trim()),
                                passwordTextBox.Text))
                        {
                            rdr.Close();
```

```
                            // //<-- Close
                            ItemPricing itemRefresh = new ItemPricing();
                            itemRefresh.DBConnection = sqlConnection;
                            itemRefresh.userIdentification = userID; //Change
to userID
                            itemRefresh.checkDate();

                            this.Hide();

                            HomePage homeForm = new HomePage(userID); //Change
to userID
                            homeForm.DBConnection = sqlConnection; // Pass the
connection object
                            homeForm.ShowDialog();

                            emailTextBox.Clear();
                            passwordTextBox.Clear();


                            LogInPage loginForm = new LogInPage();
                            loginForm.ShowDialog();
                            this.Close();




                    //this.Show();
                    // <-- Open
                        }
                        else
                        {
                            rdr.Close();
                            incorrectLabel.Text = "Incorrect password. Please
try again!";
                            incorrectLabel.Visible = true;
                        }

                    }
                    else
                    {
                        rdr.Close();
                        incorrectLabel.Text = "Incorrect email. Please try
again!";
                        incorrectLabel.Visible = true;
                    }
                // //<-- Close

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            else
            {
```

```
                incorrectLabel.Text = "Please fix errors before trying
again!";
                incorrectLabel.Visible = true;

                CancelEventArgs ex = new CancelEventArgs();
                emailTextBox_Validating(sender, ex);
                passwordTextBox_Validating(sender, ex);
            }
        }

        private void createAccountButton_Click(object sender, EventArgs e)
        {
            errorProvider1.SetError(emailTextBox, "");
            errorProvider1.SetError(passwordTextBox, "");
            incorrectLabel.Visible = false;
            this.Hide();

            CreateAccountPage createForm = new CreateAccountPage();
            createForm.DBConnection = sqlConnection;
            createForm.ShowDialog();

            emailTextBox.Clear();
            passwordTextBox.Clear();

            this.Show();
        }

        private void visibleButton_Click(object sender, EventArgs e)
        {
            visibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = false;
            unvisibleButton.Visible = true;
        }

        private void unvisibleButton_Click(object sender, EventArgs e)
        {
            unvisibleButton.Visible = false;
            passwordTextBox.UseSystemPasswordChar = true;
            visibleButton.Visible = true;
        }


        private void passwordTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (passwordTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(passwordTextBox, "Please Enter
Password");
                passValid = false;
            }
            else
            {
                errorProvider1.SetError(passwordTextBox, "");
                passValid = true;
```

```
            }
        }

        private void sqlConnection_InfoMessage(object sender,
SqlInfoMessageEventArgs e)
        {

        }

        private void emailTextBox_Validating(object sender, CancelEventArgs e)
        {
            if (emailTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(emailTextBox, "Please Enter User
Email");
                idValid = false;
            }
            else if (InputFormats.EmailValidation(emailTextBox.Text) != true)
            {
                errorProvider1.SetError(emailTextBox, "Email format is not
valid");
                idValid = false;
            }
            else
            {
                errorProvider1.SetError(emailTextBox, "");
                idValid = true;
            }
        }
    }
}
```

packages.config:

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="HtmlAgilityPack" version="1.11.67" targetFramework="net48" />
  <package id="Microsoft.Bcl.AsyncInterfaces" version="8.0.0"
targetFramework="net48" />
  <package id="Microsoft.ReportingServices.ReportViewerControl.Winforms"
version="150.1652.0" targetFramework="net48" />
  <package id="Microsoft.SqlServer.Types" version="160.1000.6"
targetFramework="net48" />
  <package id="Newtonsoft.Json" version="13.0.3" targetFramework="net48" />
  <package id="RestSharp" version="112.1.0" targetFramework="net48" />
  <package id="System.Buffers" version="4.5.1" targetFramework="net48" />
  <package id="System.Memory" version="4.5.5" targetFramework="net48" />
  <package id="System.Numerics.Vectors" version="4.5.0"
targetFramework="net48" />
  <package id="System.Runtime.CompilerServices.Unsafe" version="6.0.0"
targetFramework="net48" />
  <package id="System.Text.Encodings.Web" version="8.0.0"
targetFramework="net48" />
  <package id="System.Text.Json" version="8.0.5" targetFramework="net48" />
```

```
  <package id="System.Threading.Tasks.Extensions" version="4.5.4"
targetFramework="net48" />
  <package id="System.ValueTuple" version="4.5.0" targetFramework="net48" />
</packages>
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
using RestSharp;

namespace WhatsItWorth
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {


            Application.Run(new LogInPage());

        }
    }
}
```

ViewAccount.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WhatsItWorth
{
    public partial class ViewAccount : Form
    {
        public SqlConnection DBConnection;
        public int userIdentification;
        private bool nameValid, emailValid, passValid;
        public ViewAccount(int userID)
        {
            InitializeComponent();
```

```
            userIdentification = userID;
        }

        private void CreateAccountPage_Load(object sender, EventArgs e)
        {
            try
            {
                SqlCommand cmdLoadInfo = DBConnection.CreateCommand();

                cmdLoadInfo.CommandText = @"SELECT User_name, User_email,
User_pass, Date_Created
                                            FROM [User]
                                            WHERE UserID = @ui";
                cmdLoadInfo.Parameters.AddWithValue("@ui",
userIdentification);

                SqlDataReader rdr = cmdLoadInfo.ExecuteReader();

                if (rdr.Read())
                {
                    nameTextBox.Text = rdr[0].ToString().Trim();
                    emailTextBox.Text = rdr[1].ToString().Trim();
                    passwordTextBox.Text =
Encrypt_Decrypt.DecryptString(rdr[2].ToString().Trim());
                    dateLabel.Text = String.Format("{0:D}", rdr[3]);
                }

                rdr.Close();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }


        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }

        private void editAccountButton_Click(object sender, EventArgs e)
        {
            EditAccountConfirmation editAccountForm = new
EditAccountConfirmation(userIdentification);
            editAccountForm.DBConnection = DBConnection;
            editAccountForm.ShowDialog();

            if (editAccountForm.DialogResult == DialogResult.Yes)
            {
                enableControls(true);
                formatLabel.Visible = true;
            }
```

```
        }

        private void enableControls(bool enable)
        {
            nameTextBox.ReadOnly = !enable;
            emailTextBox.ReadOnly = !enable;
            passwordTextBox.ReadOnly = !enable;

            nameTextBox.Enabled = enable;
            emailTextBox.Enabled = enable;
            passwordTextBox.Enabled = enable;

            passwordTextBox.UseSystemPasswordChar = !enable;
            saveButton.Visible = enable;
        }

        private void deleteAccountButton_Click(object sender, EventArgs e)
        {
            DeleteAccountConfirmation deleteForm = new
DeleteAccountConfirmation(userIdentification);
            deleteForm.DBConnection = DBConnection;
            deleteForm.ShowDialog();

            if (deleteForm.DialogResult == DialogResult.Yes)
            {
                try
                {
                    SqlCommand cmdDeleteUser = DBConnection.CreateCommand();
                    cmdDeleteUser.CommandText = @"DELETE FROM [User]
                                                  WHERE UserID = @id";
                    cmdDeleteUser.Parameters.AddWithValue("@id",
userIdentification);
                    cmdDeleteUser.ExecuteNonQuery();

                    Console.WriteLine("User Deleted");

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

                this.DialogResult = DialogResult.Yes;
            }
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if (nameValid && emailValid && passValid)
            {
                try
                {
                    SqlCommand cmdUpdateAccount =
DBConnection.CreateCommand();
                    cmdUpdateAccount.CommandText =
```

```
                        @"UPDATE [User] SET User_name = @name, User_email =
@email, User_pass = @pass
                              WHERE UserID = @ui";
                    cmdUpdateAccount.Parameters.AddWithValue("@name",
nameTextBox.Text);
                    cmdUpdateAccount.Parameters.AddWithValue("@email",
emailTextBox.Text);
                    cmdUpdateAccount.Parameters.AddWithValue("@pass",
Encrypt_Decrypt.EncryptString(passwordTextBox.Text));
                    cmdUpdateAccount.Parameters.AddWithValue("@ui",
userIdentification);

                    cmdUpdateAccount.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

                enableControls(false);
                formatLabel.Visible = false;
            }
            else
            {
                CancelEventArgs ex = new CancelEventArgs();
                nameTextBox_Validating(sender, ex);
                emailTextBox_Validating(sender, ex);
                passwordTextBox_Validating(sender, ex);
            }

        }

        private void nameTextBox_Validating(object sender, CancelEventArgs e)
        {
            if (nameTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(nameTextBox, "Please Enter Name");
                nameValid = false;
            }
            else
            {
                errorProvider1.SetError(nameTextBox, "");
                nameValid = true;
            }
        }

        private void emailTextBox_Validating(object sender, CancelEventArgs e)
        {
            if (emailTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(emailTextBox, "Please Enter User
Email");
                emailValid = false;
            }
            else if (InputFormats.EmailValidation(emailTextBox.Text) != true)
```

```
            {
                errorProvider1.SetError(emailTextBox, "Email format is not
valid");
                emailValid = false;
            }
            else
            {
                errorProvider1.SetError(emailTextBox, "");
                emailValid = true;
            }
        }

        private void passwordTextBox_Validating(object sender, CancelEventArgs
e)
        {
            if (passwordTextBox.Text == string.Empty)
            {
                errorProvider1.SetError(passwordTextBox, "Please Enter
Password");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 1)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 upper case letter");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 2)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 lower case letter");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 3)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 special character");
                passValid = false;
            }
            else if (InputFormats.PassValidation(passwordTextBox.Text) == 4)
            {
                errorProvider1.SetError(passwordTextBox, "Password must
contain 1 number");
                passValid = false;
            }
            else if (passwordTextBox.Text.Length < 8)
            {
                errorProvider1.SetError(passwordTextBox, "Password must have
at least 8 characters");
                passValid = false;
            }
            else
            {
                errorProvider1.SetError(passwordTextBox, "");
                passValid = true;
```

```
            }
        }
    }
}
```

ViewItem.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient; //to use SQLCommand
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Window;

namespace WhatsItWorth
{
    public partial class ViewItem : Form
    {
        public SqlConnection DBConnection;
        public string itemIdentification;
        public ViewItem(string intID)
        {
            InitializeComponent();
            itemIdentification = intID;
        }

        private void ViewItemPage_Load(object sender, EventArgs e)
        {
            try
            {
                SqlCommand cmdLoadInfo = DBConnection.CreateCommand();

                cmdLoadInfo.CommandText = @"SELECT Metal_Type, Metal_Purity,
Total_Weight, Description, Date_Added, Original_Price, Current_Price,
Price_of_Diamonds, Photo_Link
                                            FROM Item
                                            WHERE ItemID = @ii";
                cmdLoadInfo.Parameters.AddWithValue("@ii",
itemIdentification);

                SqlDataReader rdr = cmdLoadInfo.ExecuteReader();

                if (rdr.Read())
                {

                    if (int.Parse(rdr[0].ToString()) == 1)
                    {
                        metalTextBox.Text = "Gold";
```

```
                        purityLabel.Visible = true;
                        purityTextBox.Visible = true;
                        purityTextBox.Text = rdr[1].ToString();
                    }
                    else if (int.Parse(rdr[0].ToString()) == 2)
                    {
                        metalTextBox.Text = "Silver";
                    }
                    else if (int.Parse(rdr[0].ToString()) == 3)
                    {
                        metalTextBox.Text = "Platinum";
                    }

                    totalTextBox.Text = rdr[2].ToString();
                    descriptionTextBox.Text = rdr[3].ToString().Trim();

                    dateTextBox.Text = String.Format("{0:d}", rdr[4]);
                    originalTextBox.Text = String.Format("{0:C}", rdr[5]);
                    currentTextBox.Text = String.Format("{0:C}", rdr[6]);

                    double originalPrice = double.Parse(rdr[5].ToString());
                    double currentPrice = double.Parse(rdr[6].ToString());
                    double change = ((currentPrice - originalPrice) /
originalPrice) * 100;

                    if (currentPrice > originalPrice)
                    {
                        changeTextBox.ForeColor = Color.LawnGreen;
                    }
                    else if (currentPrice < originalPrice)
                    {
                        changeTextBox.ForeColor = Color.Red;
                    }
                    else
                    {
                        changeTextBox.ForeColor = Color.Black;
                    }

                    changeTextBox.Text = (change/100).ToString("P1");

                    if (rdr[7].ToString() != "")
                    {
                        viewList.Visible = true;
                    }

                    if (rdr[8].ToString() != "")
                    {
                        try
                        {
                            itemImageBox.Image = new
Bitmap(rdr[8].ToString().Trim());
                            itemImageBox.Visible = true;
                        }
                        catch (Exception exception)
                        {
```

```
                        Console.WriteLine(exception.Message);
                    }

                }

            }


this.diamondTableAdapter.Fill(this.whatsItWorthDataSet.DiamondTable,
itemIdentification);


                rdr.Close();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void goBackButton_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }

        private void deleteButton_Click(object sender, EventArgs e)
        {
            DeleteItemConfirmation deleteItemForm = new
DeleteItemConfirmation();
            deleteItemForm.ShowDialog();

            if (deleteItemForm.DialogResult == DialogResult.Yes)
            {
                try
                {
                    SqlCommand cmdDeleteItem = DBConnection.CreateCommand();
                    cmdDeleteItem.CommandText = @"DELETE FROM Item
                                                  WHERE ItemID = @ii";
                    cmdDeleteItem.Parameters.AddWithValue("@ii",
itemIdentification);

                    cmdDeleteItem.ExecuteNonQuery();

                    Console.WriteLine("Item Deleted");

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }

                this.DialogResult = DialogResult.Yes;
            }
```

```
        }

        private void photoButton_Click(object sender, EventArgs e)
        {
            Image File;

            OpenFileDialog dialog = new OpenFileDialog();

            dialog.Filter = "Image files (*.jpg, *.png) | *.jpg; *.png";

            if (dialog.ShowDialog() == DialogResult.OK)
            {
                File = Image.FromFile(dialog.FileName);
                itemImageBox.Image = File;
                itemImageBox.Visible = true;

                try
                {
                    SqlCommand cmdUpdateDate = DBConnection.CreateCommand();
                    cmdUpdateDate.CommandText = @"UPDATE Item
                                                  SET Photo_Link = @pl
                                                  WHERE ItemID = @ii";
                    cmdUpdateDate.Parameters.AddWithValue("@pl",
dialog.FileName);
                    cmdUpdateDate.Parameters.AddWithValue("@ii",
itemIdentification);

                    cmdUpdateDate.ExecuteNonQuery();

                    Console.WriteLine("Photo Updated");

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void editButton_Click(object sender, EventArgs e)
        {
            editButton.Visible = false;
            saveButton.Visible = true;

            descriptionTextBox.Enabled = true;
            descriptionTextBox.ReadOnly = false;
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if (!String.Equals(descriptionTextBox.Text.Trim(), ""))
            {
                try
                {
```

```
                    SqlCommand cmdUpdateDescription =
DBConnection.CreateCommand();
                    cmdUpdateDescription.CommandText = @"UPDATE Item
                                                         SET Description = @d
                                                         WHERE ItemID = @ii";
                    cmdUpdateDescription.Parameters.AddWithValue("@d",
descriptionTextBox.Text.Trim());
                    cmdUpdateDescription.Parameters.AddWithValue("@ii",
itemIdentification);

                    cmdUpdateDescription.ExecuteNonQuery();

                    Console.WriteLine("Description Updated");

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            else
            {
                string tempDescription;

                 if (String.Equals(metalTextBox.Text.Trim(), "Gold"))
                 {
                     if (viewList.Visible)
                     {
                         tempDescription = purityTextBox.Text.Trim() + "K
Gold; " + totalTextBox.Text.Trim() + "G with Diamonds";
                     }
                     else
                     {
                         tempDescription = purityTextBox.Text.Trim() + "K
Gold; " + totalTextBox.Text.Trim() + "G";
                     }
                 }
                 else
                 {
                     if (viewList.Visible)
                     {
                         tempDescription = metalTextBox.Text.Trim() + "; " +
totalTextBox.Text.Trim() + "G with Diamonds";
                     }
                     else
                     {
                         tempDescription = metalTextBox.Text.Trim() + "; " +
totalTextBox.Text.Trim() + "G";
                     }
                 }


                try
                {
```

```
                    SqlCommand cmdUpdateDescription =
DBConnection.CreateCommand();
                    cmdUpdateDescription.CommandText = @"UPDATE Item
                                                         SET Description = @td
                                                         WHERE ItemID = @ii";
                    cmdUpdateDescription.Parameters.AddWithValue("@td",
tempDescription.Trim());
                    cmdUpdateDescription.Parameters.AddWithValue("@ii",
itemIdentification);

                    cmdUpdateDescription.ExecuteNonQuery();

                    Console.WriteLine("Description Updated");

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

            editButton.Visible = true;
            saveButton.Visible = false;

            descriptionTextBox.Enabled = false;
            descriptionTextBox.ReadOnly = true;

        }
    }
}
```