MINISTRY OF EDUCATION OF THE REPUBLIC OF BELARUS

EDUCATIONAL INSTITUTION  
 «SUKHOI STATE TECHNICAL UNIVERSITY OF GOMEL»

Automation and Information Systems Faculty

Department of Information Technology

EXPLANATORY NOTE

To term paper

By discipline

"Object-oriented design and programing"

“WPF applications for accounting sales of drinks and snacks in a coffee shop”

Compiled by: student gr. IT-23 Nour Alden Mohammad  
Saad Awawdeh

(signature, date)

Practice leader: Kurochka Konstantin Sergeevich.

(job title)

(signature, date)

Date of protection

Grade

Signatures of the commission members:

Gomel 2024

Contents

[Introduction 4](#_Toc166920811)

[1 Technologies for Solution 5](#_Toc166920812)

[1.1 C# As the Main Programming for my Application 5](#_Toc166920813)

[1.2 Java and C# 5](#_Toc166920814)

[1.3 MSSQL DBMS as a Database 6](#_Toc166920815)

[1.4 ADO.NET Technology 8](#_Toc166920816)

[1.5 Windows Presentation Foundation Farmwork (WPF) 8](#_Toc166920817)

[2 Implementation of the software Product 15](#_Toc166920818)

[2.1 Architecture of the Program 15](#_Toc166920819)

[2.2 Structures for Database 16](#_Toc166920820)

[2.3 Implementation of model classes 17](#_Toc166920821)

[2.4 Implementation of ORM 20](#_Toc166920822)

[2.5 Implementation of ADO 21](#_Toc166920823)

[3. Test and describe user interface (UI/UX) 25](#_Toc166920824)

[3.1 Task has been solved in full. 25](#_Toc166920825)

[3.2 Descript the user interface. 25](#_Toc166920826)

[3.3 Diagram of use case 29](#_Toc166920827)

[3.4 Develop Test 30](#_Toc166920828)

[3.5 Validation Test 32](#_Toc166920829)

[Conclusion 34](#_Toc166920830)

[References 35](#_Toc166920831)

# Introduction

The growing demand for tech-driven solutions, together with changing market needs, is driving companies to invest in initiatives that will make their operations more efficient and profitable.

Today, with the growing number of innovations surrounding business processes and operations, the importance of technology has become inevitable. As a matter of fact, almost all businesses today have technology at their core.

With this reality becoming apparent, businesses are now focusing more on minimizing risks inherent in the deployment of software development enterprises. And it is perfectly normal for them to do so because a lot rests on the tech used.

Enterprise application development is the process of building custom software applications for a business or organization. It differs from other types of application development because it involves longer-term planning, and more complex goals, and requirements.

Today's enterprise software development solutions were created by small business owners who wanted to solve their own problems, lower costs: Enterprise software application development companies can create custom solutions for business at a lower cost, and it’s also a time savings the longer-term planning involved in enterprise application development, with better quality products Enterprise developers have experience in creating software applications, easy to navigate and understand so that people can get the information they need quickly. Also To Increased Productivity Employees can work faster when they have access to all the information they need.

In my case Writing orders and filling data manually by hand is a very hard job, and its cost businesses a lot of money and time, also the possibility of human’s mistakes rise, so here it’s come’s the Software Solutions to help every owner of business to automate a lot of works, and one of this Solutions Is (Coffee Shop Application for Orders).

I will try to do an application as a Coffee shop accounting system and sales of drinks and snacks, I called this coffee shop (Havana). I will try my best to develop this application to solve basic cases that the cashier will meet when he/she communicates with the customer in this Coffee Shope, and also to help the manager to keep control on his Business.

At the next chapters I will explain more information and details about the development details, and technology that I had been used and why I use it, and I will make small comparison between different technology.

# 1 Technologies for Solution

1.1 C# As the Main Programming for my Application

In my application the main language that I chose is C# programming language, which is High Level Programming Language, Type-Safe and its Cross-Platform Language also its one of the oldest, most reliable, and most common computer programming languages, C# is an object-oriented language (OOP), supports many programming paradigms like functional programming (*Procedural Programming*), C# is an open-source language based on .Net, developed by Microsoft.

C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers. (Microsoft n.d.)

C# one of the best recommended languages to develop enterprise applications, support for future updates and future large-scale development at the applications. There are hundreds of programming languages, this application can be developed by, but Microsoft have very clear-full tools, and Architectures, for developing an enterprise application by using many tools called Microsoft Ecosystem, as this application is a windows application that works on a PC.

Being a garbage collected language means there is a whole class of problems I can ignore, I don't have to think about them, and don't have to write code for. This is awesome because it lets me think about what the logic is doing without having to worry so much about these small issues of memory management.

From Generics, to LINQ, to lock-free and lock-poor concurrency structures to the thread pool, the task concept with a sync/await and all the improvements in the C#/.net advanced in leaps and bounds. Sure, if I want to speak about C#, I should speak about (CLR) too, the *Common Language Runtime* (CLR) is just what its name says it is: a runtime that is usable by different and varied programming languages.

The core features of the CLR (such as memory management, assembly loading, security, exception handling, and thread synchronization) are available to any and all programming languages that target it period. For example, the runtime uses exceptions to report errors, so all languages that target the runtime also get errors reported via exceptions. Another example is that the runtime also allows you to create a thread, so any language that targets the runtime can create a thread. (Richter 2012)

1.2 Java and C#

First, I should explain why I didn't choose java as a programming language for my application, java originally created to be a small, reliable, portable, distributed, real-time operating platform, run in any platform, of course, Java is not without its shortcomings, including unfortunately the following: Slow: Java consumes a good deal of memory and is considerably slower than other languages like C/C++.

Also, its Outdated GUI, Java GUI can appear outdated compared to other languages like C#.

There is No backup: Java provides no facility for data backup. And it’s also Not easy to read, Java code can be rather verbose and complicated to read.

But Java has a big problem. It is often perceived as old, but Python is older, and JavaScript is its contemporary. Java has had continuous popularity and success for nearly three decades.

That means that most of developers worked with Java 1.4, which was released two decades ago. Unlike any other platform out there, Java is still compatible with that release. That is fantastic but also creates a sense of disconnect.

Developers compare that highly outdated version of Java to modern incarnation of other languages or platforms. That is an unfair comparison, and Java is a victim of its own success.

There are several sources for information about new Java features but there is a lack in a comprehensive introductory guide to modern Java, that carries us from that old version to the modern world. (Almog 2023)

1.3 MSSQL DBMS as a Database

SQL stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.

SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are; Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as “*Select*”, “*Insert*”, “*Update*”, “*Delete*”, “*Create*”, and “*Drop*” can be used to accomplish almost everything that one needs to do with a database.

MS SQL Server is a *Relational Database Management System* (RDBMS) developed by Microsoft. A Relational database is based on a Relational Model architecture. The data is organized in tables(relations), and the tables are related to each other. Each table has rows and columns(attributes). MS SQL Server is a software product used to administer the database and retrieve information. (DOC n.d.)

SQL Server Management Studio (SSMS) is a management software for SQL Server database engine.

An SSMS is a primary tool to connect and administer the SQL Server database engine and run Transact-SQL on the database.

Management Studio provides an Analysis Services Script project in which you develop and save scripts written in Multidimensional Expressions (MDX), Data Mining Extensions (DMX), and XML for Analysis (XMLA).

You use Analysis Services Scripts projects to perform management tasks or recreate objects, such as database and cubes, on Analysis Services instances. For example, you can develop an XMLA script in an Analysis Services Script project that creates new objects directly on an existing Analysis Services instance. (SSMS n.d.)

**PostgreSQL,** PostgreSQL is a powerful, open-source object relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRESQL project at the University of California at Berkeley and has more than 35 years of active development on the core platform. PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open-source community behind the software to consistently deliver performance and innovative solutions. PostgreSQL runs on all major operating systems, has been ACID-compliant since 2001, and has powerful add-ons such as the popular PostGIS geospatial database extender. It is no surprise that PostgreSQL has become the open-source relational database of choice for many people and organizations.

The main reason why I didn't use PostgreSQL is because it is an object-relational database. The disadvantages/limitations of PostgreSQL, PostgreSQL is not owned by one organization. So, it has had trouble getting its name out there despite being fully featured and comparable to other DBMS systems. Changes made for speed improvement require more work than MySQL as PostgreSQL focuses on compatibility. Many open-source apps support MySQL, but may not support PostgreSQL. On performance metrics, it is slower than MySQL. (postgresql n.d.)

**MySQL** is the world’s most popular open-source database. According to DB Engines, MySQL ranks as the second-most-popular database, behind Oracle Database. MySQL powers many of the most accessed applications, including Facebook, Twitter, Netflix, Uber, Airbnb, Shopify, and Booking.com. Since MySQL is open source, it includes numerous features developed in close cooperation with users over more than 25 years.

Despite MySQL popularity and widespread use, MySQL is not without its drawbacks. These limitations can range from performance issues in large-scale deployments to a lack of advanced features found in other relational database management systems. Understanding these cons is crucial for database administrators and developers in selecting the right tool for their specific needs.

MySQL has performance issues with large-scale databases. and limited support for advanced SQL features. also, inadequate scalability for big data. subpar support for NoSQL complexity in high availability and replication configurations. disadvantage inconsistent performance across different storage engines. weaknesses in handling concurrent transactions. limited security features in community edition resource-intensive for complex queries lack of comprehensive support and documentation.

1.4 ADO.NET Technology

ADO.NET is a family of technologies that allows .NET developers to interact with data in standard, structured, and primarily disconnected ways. Applications written using the .NET Framework depend on .NET class libraries, which exist in special DLL files that encapsulate common programming functionality in an easy-to-access format. Most of the libraries supplied with the .NET Framework appear within the System namespace. System.IO, for instance, includes classes that let you interact with standard disk files and related data streams. The System.Security library provides access to, among other things, data encryption features.

ADO.NET, expressed through the System.Data namespace, implements a small set of libraries that makes consuming and manipulating large amounts of data simple and straightforward.

ADO.NET manages both internal data created in memory and used solely within an application and external data housed in a storage area apart from the application, such as in a relational database or text file. Regardless of the source, ADO.NET generalizes the relevant data and presents it to your code in spreadsheet style rows and columns.

Although ADO.NET manipulates data in tabular form, you can also use ADO.NET to access nontabular data. For instance, an ADO.NET provider could supply access to hierarchical data such as that found in the Windows Registry, as long as that provider expressed the data in a tabular structure for ADO.NET’s use. (Patrick 2010)

1.5 Windows Presentation Foundation Farmwork (WPF)

When .NET first appeared, it introduced a small avalanche of new technologies. There was a whole new way to write web applications (ASP.NET), a whole new way to connect to databases (ADO.NET), new type safe languages (C# and VB.NET), and a managed runtime (the CLR). Not least among these new technologies was Windows Forms, a library of classes for building Windows applications.

Although Windows Forms is a full-featured toolkit, it’s hardwired to old, essential bits of Windows plumbing. Most significantly, Windows Forms relies on the Windows API to create the visual appearance of standard user interface elements such as buttons, text boxes, check boxes, and so on. As a result, these ingredients are essentially customizable.

For example, if you want to create a stylish glow button you need to create a custom control and paint every aspect of the button (in all its different states) using a lower-level drawing model. Even worse, ordinary windows are carved up into distinct regions, with each control getting its own piece of real estate.

As a result, there’s no good way for the painting in one control (for example, the glow effect behind a button) to spread into the area owned by another control. And don’t even think about introducing animated effects such as spinning text, shimmering buttons, shrinking windows, or live previews because you’ll have to paint every detail by hand.

The **Windows Presentation Foundation** (WPF) changed all this by introducing a model with entirely different plumbing. Although WPF includes the standard controls you’re familiar with, it draws every text, border, and background fill itself.

As a result, WPF can provide much more powerful features that let you alter the way any piece of screen content is rendered. Using these features, you can restyle common controls such as buttons, often without writing any code. Similarly, you can use transformation objects to rotate, stretch, scale, and skew anything in your user interface, and you can even use WPF’s baked-in animation system to do it right before the user’s eyes.

And because the WPF engine renders the content for a window as part of a single operation, it can handle unlimited layers of overlapping controls, even if these controls are irregularly shaped and partially transparent.

Underlying WPF is a powerful infrastructure based on DirectX, the hardware accelerated graphics API that’s commonly used in cutting-edge computer games. This means that you can use rich graphical effects without incurring the performance overhead that you’d suffer with Windows Forms.

In fact, you even get advanced features such as support for video files and 3-D content. Using these features (and a good design tool), it’s possible to create eyepopping user interfaces and visual effects that would have been all but impossible with Windows Forms.

It’s also important to note that you can use WPF to build an ordinary Windows application with standard controls and a straightforward visual appearance. In fact, it’s just as easy to use common controls in WPF as it is in the older Windows Forms model. Even better, WPF enhances features that appeal directly to business developers, including a vastly improved data binding model, a set of classes for printing content and managing print queues, and a document feature for displaying large amounts of formatted text.

You’ll even get a model for building page-based applications that run seamlessly in Internet Explorer and can be launched from a website, all without the usual security warnings and irritating installation prompts. Overall, WPF combines the best of the old world of Windows development with new innovations for building modern, graphically rich user interfaces. (MacDonald 2012)

WPF is the best toolkit to use if you want to build a rich desktop application that runs on Windows 7, Windows 10, and Windows 11 in desktop mode (as well as the corresponding versions of Windows Server).

In fact, it’s the *only* general-purpose toolkit that targets these versions of Windows. By comparison, Microsoft’s new Metro toolkit although exciting is limited to Windows 8 systems only. (WPF applications can even be made to run on ancient Windows XP computers, which are still found in many businesses.

The only limitation is that you must configure Visual Studio to target the slightly older .NET 4.0 Framework, rather than .NET 4.5.).

In fact, WPF applications use DirectX no matter what type of user interface you create. That means that whether you’re designing complex three-dimensional graphics (DirectX’s forte) or just drawing buttons and plain text, all the drawing work travels through the DirectX pipeline.

As a result, even the most mundane business applications can use rich effects such as transparency and anti-aliasing. You also benefit from hardware acceleration, which simply means DirectX hands off as much work as possible to the graphics processing unit (GPU), which is the dedicated processor on the video card.

DirectX is more efficient because it understands higher-level ingredients such as textures and gradients that can be rendered directly by the video card. GDI/GDI+ doesn’t, so it needs to convert them to pixel-by-pixel instructions, which are rendered much more slowly by modern video cards. (MacDonald 2012)

**The Architecture of WPF**, uses a multilayered architecture. At the top, your application interacts with a high-level set of services that are completely written in managed C# code. The actual work of translating .NET objects into Direct3D textures and triangles happens behind the scenes, using a lower-level unmanaged component called milcore.dll. milcore.dll is implemented in unmanaged code because it needs tight integration with Direct3D and because it’s extremely performance sensitive. (MacDonald 2012)



* + Figure 1.1. The architecture of WPF.

**XAML**(short for *Extensible Application Markup Language* and pronounced *zammel*) is a markup language used to instantiate .NET objects. Although XAML is a technology that can be applied to many problem domains, its primary role in life is to construct WPF user interfaces. In other words, XAML documents define the arrangement of panels, buttons, and controls that make up the windows in a WPF application.

It’s unlikely that you’ll write XAML by hand. Instead, you’ll use a tool that generates the XAML you need. If you’re a developer, you’ll probably start with Microsoft Visual Studio. Because both tools are equally at home with XAML, you can create a basic user interface with Visual Studio and then hand it off to a crack design team that can polish it up with custom graphics in Expression Blend. In fact, this ability to integrate the workflow between developers and designers is one of the key reasons that Microsoft created XAML.

Once you understand the broad rules of XAML, you’ll know what is and isn’t possible in a WPF user interface and how to make changes by hand when it’s necessary. (MacDonald 2012)

C sharp has a helpful type system at large and small scales, C# has been a statically typed language from the start: your code specifies the types of variables, parameters, values returned from methods, and so on. The more precisely you can specify the shape of the data your code accepts and returns, the more the compiler can help you avoid mistakes.

That’s particularly true as the application you’re building grows. If you can see all the code for your whole program on one screen (or at least hold it all in your head at one time), a statically typed language doesn’t have much benefit. As the scale increases, it becomes increasingly important that your code concisely and effectively communicates what it does. You can do that through documentation, but static typing lets you communicate in a machine-readable way.

As C# has evolved, its type system has allowed more fine-grained descriptions. The most obvious example of this is generics. (Skeet 2019)

Typing Terminology in C#, many terms are used to describe the way programming languages interact with their type system. Some people use the terms weakly typed and strongly typed, but I try to avoid those because they’re not clearly defined and mean different things to different developers. Two other aspects have more consensus: static dynamic typing and explicit implicit typing.

**Static and Dynamic Typing Languages**, Let’s look at each of those in turn. Static and Dynamic Typing Languages that are statically typed are typically compiled languages; the compiler is able to determine the type of each expression and check that it’s used correctly. For example, if you make a method call on an object, the compiler can use the type information to check that there’s a suitable method to call based on the type of the expression the method is called on, the name of the method, and the number and types of the arguments. Determining the meaning of something like a method call or field access is called binding.

Languages that are dynamically typed leave all or most of the binding to execution time. Aside from the dynamic binding introduced in C# 4, C# is a statically typed language.

Even though the choice of which implementation of a virtual method should be executed depends on the execution-time type of the object it’s called on, the binding process of determining the method signature all happens at compile time.

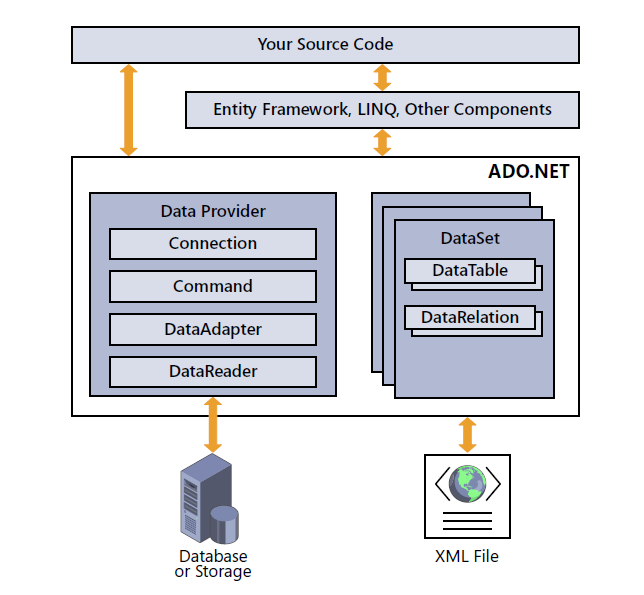
**Explicit and Implicit Typing in C#**, in a language that’s *explicitly typed*, the source code specifies all the types involved. This could be for local variables, fields, method parameters, or method return types.

For example. A language that’s *implicitly typed* allows the developer to omit the types from the source code so some other mechanism (whether it’s a compiler or something at execution time) can infer which type is meant based on other context is mostly explicitly typed. Even before C# 3, there was some implicit typing, such as type inference for generic type arguments.

Arguably, the presence of implicit conversions (such as *int* to *long*) make the language less explicitly typed, too. With those different aspects of typing separated. (Skeet 2019)

**The Major Components of ADO.NET**, The *System.Data* namespace includes many distinct ADO.NET classes that work together to provide access to tabular data. The library includes two major groups of classes: those that manage the actual data within the software and those that communicate with external data systems.

At the data-shaped heart of the library is the *Data Table*. Similar in purpose to tables in a database, the *Data Table* manages all the actual data values that you and your source code ultimately care about. Each *Data Table* contains zero or more rows of data, with the individual data values of each row identified by the table’s column definitions.



* + Figure 1.2 shows the major parts that make up an ADO.NET instance.

At the data-shaped heart of the library is the DataTable. Similar in purpose to tables in a database, the DataTable manages all the actual data values that you and your source code ultimately care about. Each DataTable contains zero or more rows of data, with the individual data values of each row identified by the table’s column definitions.

Each table defines DataColumn items, each representing the individual data values that appear in the table’s records. DataColumn definitions include a data type declaration based on the kind of data destined for each column. For instance, a CustomerLastName column might be defined to use data of type System.String, whereas an OrderSalesTax column could be crafted for use with System.Decimal content.

One DataRow entry exists for each record of data stored within a table, providing access to the distinct columnar data values. ADO.NET includes methods that let you add to, delete from, modify, and query each DataTable object’s rows. For tables connected to an external data storage area, any changes made can be propagated back to the source.

You can optionally establish links between the tables of data using DataRelation entries.

Programmatic limitations can be placed on tables and their data values using Constraint instances.

DataView instances provide a limited or modified view of the rows in a DataTable.

Tables can be grouped together into a DataSet. Some tools that interact with ADO.NET data require that any tables be bound within a DataSet, but if you plan to do some limited work with only a single table, it’s fine to work with just the DataTable instance. DataTable instances and their associated objects are sufficient for working with internal data. To connect with external data from a database, ADO.NET features multiple data providers, including a custom provider for Microsoft SQL Server. Database platforms without a specific provider use the more generic ODBC and OLE DB providers, both included with ADO.NET. Several third-party providers can be purchased or obtained free of charge, which target specific platforms, including Oracle.

All communication with the external data source occurs through a *Connection* object. ADO.NET supports connection pooling for increased efficiency between queries.

SQL queries and data management statements get wrapped in a Command object before being sent to the data source. Commands can include optional Parameter instances that let you call stored procedures or create fill-in-the-blank queries. The DataAdapter object stores standard query definitions for interacting with a database.

Removing the tedium of constantly needing to build SQL statements for each record you want to read or write, and helping to automate some ADO.NET-related tasks. The DataReader object provides fast, read-only access to the results of a query for those times when you just need to get your data quickly.ADO.NET also includes features that let you save an entire DataSet as an XML file and load it back in later. And that’s just the start. You’ll learn how to use all these elements—and more— throughout the upcoming chapters. (Patrick 2010)

# 2 Implementation of the software Product

2.1 Architecture of the Program

This Application should be able to make orders and keep all the data of sales in this Coffee Shop database.

After an order happened the bell should have the date time of the order, name of the buyer, list of drinks and snacks that the customer ordered, and for each product there should've known the cost of it, and the volume or the weight of it, the count also.

The drinks are divided into types like: tea, coffee, juices, and water. Snacks are divided into: hot, cold and sweets. The program has the possibility of searching for drinks and snacks both by name or even part of the name, or even by type or by cost. This program should be able to generate reports on orders, for the specified period, report on sales of each type of product for the specified period of time. All this report should have the possibility to be exported to a text file.

The programming language that I use is C#, with MS SQL DBMS for database and ADO .Net access technology as a data source connection and for design and creating a database of the selected DBMS.

The Tables of this database both schema of C# class architecture and the Database Tables architecture too, attached at the end of this work.

The application contains classes to access data form, done by .Net and display, *add*, *delete* and *edit* from database tables, using ADO .Net Technology and by user-friendly interface.

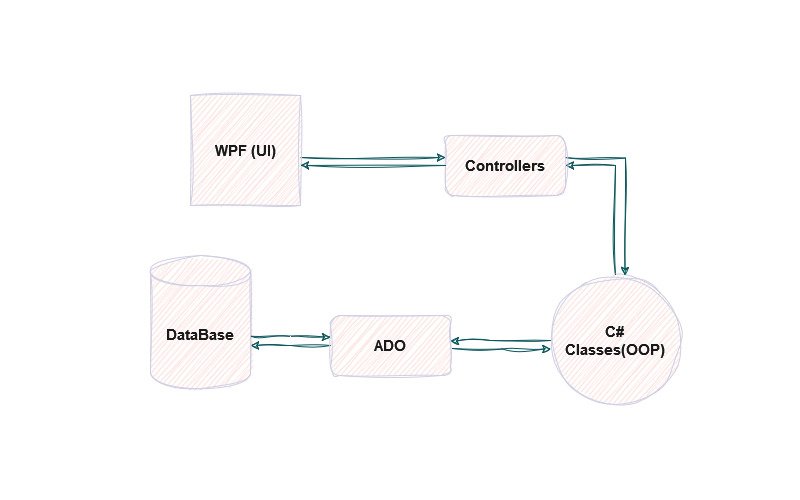
For User Interface (UI), I used WPF (*Windows Presentation Foundation*) technology to develop fast-easily to use UI/UX. WPF, Windows Presentation Foundation, is a UI (user interface) framework that creates desktop client applications.

The WPF development platform supports a broad set of application development features, including an application model, resources, controls, graphics, layout, data binding, documents, and security.

There is five component that I used in my course work, first it is the C# classes and OOP, Then I used ADO.Net technology to Communicate with Database, I Used SQL MS As a Database, Then I used the Controllers to communicate C# classes and the UI, which I used WPF for User Interface.

Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to construct their own ORM tools, Figure 2.1 explain this 5 Components.



* + Figure 2.1 Architecture for the five components.

2.2 Structures for Database

I made Six tables at the database; the main one named **Orders**, and this table should record the consumer order at our Havana Coffee Shop, this tables has five columns; id, name, id buyer, Date Time.

Orders table connect with other table as a foreign key with id\_buyer columns, this allows us to store the buyer’s name, with table name **Buyer**.

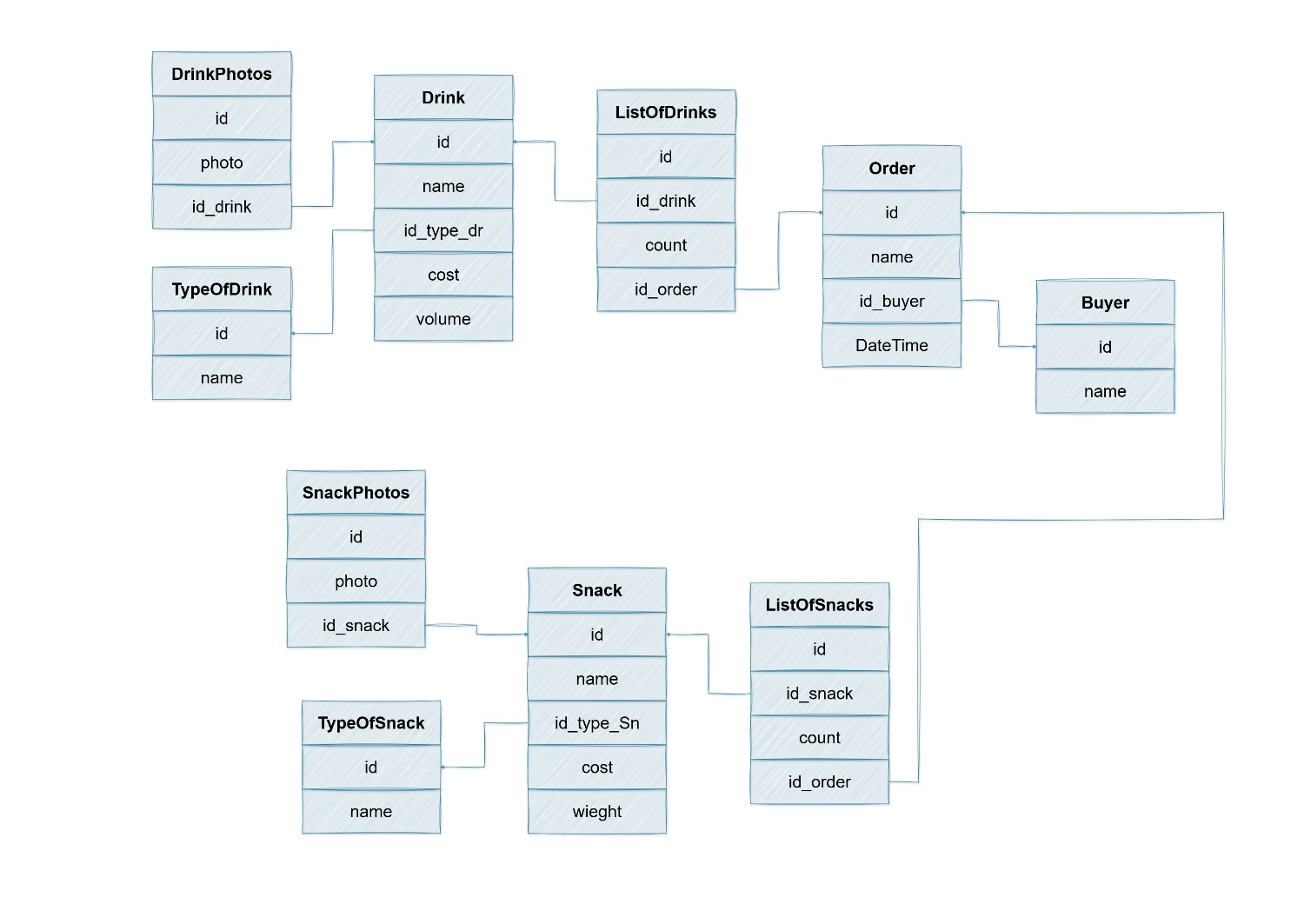
My two main categories that this Coffee shop sell, is tables names, **List of Drink**, **List of Snack**, this tables always me to save the record of the consumers orders later, because sometimes the consumers can order many Drinks or many Snacks in one Order.

List of Drinks has four columns, Id, Id Drink, count, Id Order, **List of Snack** table has four columns also the same, Id, Id Snack, count, Id Order, List of Drink and List of Snack connect with a Foreign at Id Order, column with Order table.

**Drink** Table which presents the Drinks Items that the Coffee Shope sell, Drink table has five columns, Id, name, Id Type drink, Cost, Volume, it’s having a foreign key that connect the Drink table at column Id Type of Drink, to table name **Type of Drink**.

**Snack** Table too have the same columns but the different is the weight, connect with **Type of Snack.**

**Drink Photos**, and **Snack Photo** tables, have columns, Id, photo, Id drink/Id Snack, in photo column storing the photos of, drink and snacks, this tables also connected one with drink, and other with snack table, See Figure 2.2 Show the Architecture of Database.



* + Figure 2.2 The Architecture of Database.

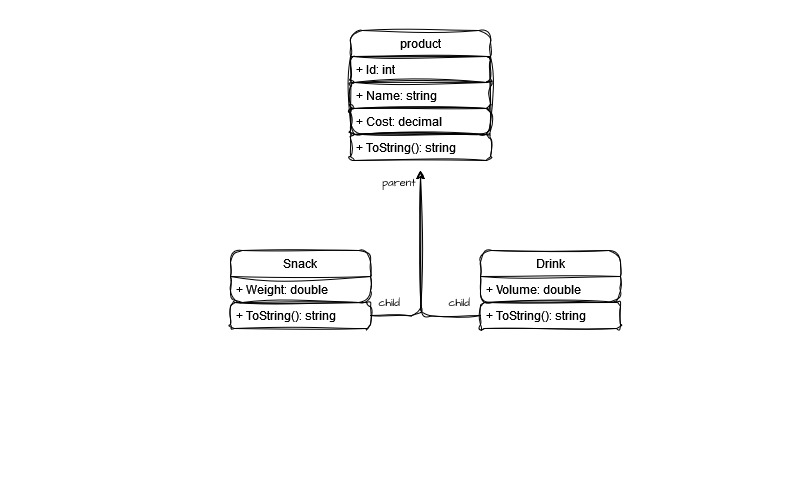
2.3 Implementation of model classes

In a project Library of WPF I created the models of the classes, Figure 2.3 Shows the Classes Models.



* + Figure 2.3 The Classes Models.

Here also I create the Inheritance, By the Parents class called **Product** and the child’s class called Snack and Drink, Figure 2.4 Shows the Classes Inheritance.



* + Figure 2.4 The Classes Inheritance.

The first class is the Parent abstract class named "Product". **Product class** extends the ToString() method to provide a custom string representation of the Product object, returning a formatted string that includes the product's ID, name, and cost. The abstract nature of the class suggests that it should be inherited and extended by other classes to represent specific types of products. The Product class has properties for the product's ID, name, and cost. It also includes a parameterized constructor for initializing the product object with the provided values.

The base class "Product" is the source class from which the **Snack class** is derived. It adds further features, like weight, type of snack ID, and count, that are unique to snacks. Several constructors are provided by the class to initialize the snack object with various combinations of properties. Additionally, it modifies the ToString() function to provide the Snack object a unique string representation that includes the weight of the snack as well as any inherited characteristics from the base class.

**Drink class** is formed like Snack class too. It adds more drink-specific properties like count, type of drink ID, and volume. Several constructors are provided by the class to initialize the drink object with various combinations of properties. Additionally, it modifies the ToString() function to provide the Drink object a unique string representation that includes the volume of the drink as well as any inherited attributes from the base class.

The **Buyer class** contains attributes for the buyer's name and ID, and it represents a buyer object. For initializing the buyer object, it has two constructors: one parameterized and one default. In order to create a unique string representation of the Buyer object and return the buyer's name, the class overrides the ToString() method.

**DrinkPhoto**: This class represents a photo of a drink. It has properties such as an ID, an ImageSource object representing the image, and a Drink object associated with the photo. It includes a constructor to initialize the properties and overrides the ToString() method to provide a string representation of the DrinkPhoto object.

The **SnackPhoto** class following properties: Id (integer): Represents the ID of the snack photo. Image (ImageSource): Represents the image associated with the snack. Snack (Snack): Represents the snack object associated with the photo. The class has a constructor that takes an id, image, and snack parameter to initialize the properties. Additionally, the class overrides the ToString() method to provide a string representation of the Snack object associated with the photo. The string representation includes the ID, name, cost, and weight of the snack.

**ListOfDrinks**: This class represents a list of drinks. It has properties such as an ID, a list of Drink objects, and a count of drinks. It includes constructors to initialize the properties and overrides the ToString() method to provide a string representation of the ListOfDrinks object.

**ListOfSnacks**: This class represents a list of snacks. It has properties such as an ID, a list of Snack objects, and a count of snacks. It includes constructors to initialize the properties and overrides the ToString() method to provide a string representation of the ListOfSnacks object.

**Order**: This class represents an order. It has properties such as an ID, a name, a DateTime object representing the order's date and time, a Buyer object representing the buyer's name, ListOfDrinks and ListOfSnacks objects representing the ordered drinks and snacks, a total cost, and various methods to calculate the count of snacks, count of drinks, and total cost. It includes constructors to initialize the properties and overrides the ToString() method to provide a string representation of the Order object.

The **TypeOfSnack** class represents a type or category of snack. It has properties such as an ID and a name. The class includes two properties: The Id property represents the unique identifier of the type of snack and is of type int.

The Name property represents the name of the type of snack and is of type string. The class has a constructor that allows you to initialize the properties of a TypeOfSnack object. The class overrides the ToString() method to provide a custom string representation of the TypeOfSnack object. The string representation includes the ID and the name of the type of snack.

**TypeOfDrink** class. It has two properties: Id and Name. The class also has a constructor that accepts an id and name parameter to initialize the properties. It overrides the ToString() method to provide a custom string representation of the object.

2.4 Implementation of ORM

Currently our app only saves data while it is running. Once you end the session and close the program, all of the data you provided disappears. We can make that data **persistent** by connecting the application to a database.

This will retain the data even if the application is not running, and allow us to access it when we open a new session.

Let’s connect our WPF application to a relational database and add persistent data storage to our apps. To do so, we need to use **object-relational mapping**.

**Object-Relational Mapping** or **ORM** is a technique for converting data between C# objects and relational databases.

ORM converts data between two incompatible type systems (C# and MSSSQL), such that each model class becomes a table in our database and each instance a row of the table.

To make ORM work in our C# applications, we need an **object-relational mapper** to convert between C# and SQL. When we create and configure a new model class to be stored in a database, a mapper creates a SQL query to make the corresponding table.

Resources of ORM, Database providers, you can work with quite a number of different databases using Entity Framework. The whole idea is to have an agnostic approach so you, in theory, could replace one database for another and your code remains the same.

Using an ORM is about being faster, more productive and about knowing exactly what goes into a database. for most simple applications it's definitely good to use. For applications that need really performant queries you can definitely still use it but you need to be more observant on what SQL your ORM produces.

Sometimes it's good enough and sometimes you need to write those queries by hand using SQL. Typically reporting queries is something I personally don't use ORMs for as they tend to be complex and hard to express in code.

There is more than one ORM choice for .Net. **Entity Framework** is the most known one but there are other ones. You have to decide which one fits your project.

**Linq 2 db** Offers a similar experience to Entity Framework if you look at the syntax alone. Some say the syntax is close to what you get in actual SQL

**Dapper, NHibernate** .Net port of Hibernate. One of the oldest ORMs out there.there are more ORMs out there but the three above are well-known choices. most ORMs lets you define the table structure in code and you can map a class so that it corresponds to a table. The columns are simply properties on the class. Depending on the ORM several approaches are possible

**Schema first**, in this scenario you define a schema of what tables you have, how they relate like 1-1, 1-Many, Man-to-Many and so on. You end up generating code from the schema.

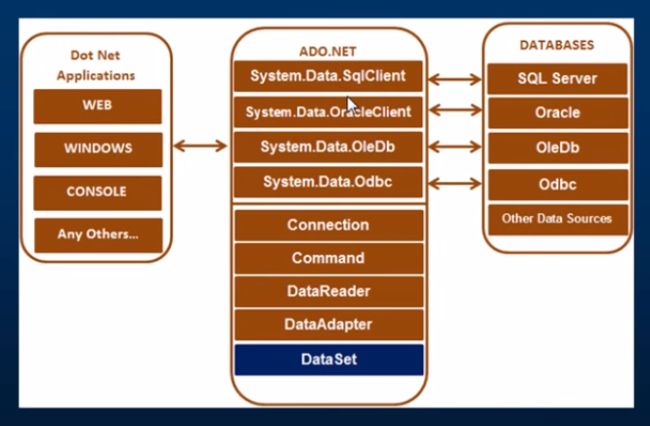
**Code first**, in this approach, you define the code first. Every table corresponds to a class and you can express how everything relates in code. Your ORM will then take a look at your code and generate structural SQL from it.

A lot of ORMs comes with a concept called migrations. A migration is simply a piece of script that either alters the structure of the database or runs a piece of SQL that affects the data like for example seeding the database with some initial data. The idea is that every time you do a change of the database that should be a small transactional change captured in a migration.

That migration can then be applied to the database and thereby the database will be altered in the desired way. For example, adding a customer table to database would be a migration that when applied would create the table in the Database. A Migration can either be expressed as SQL or in Code.

Install and Set up.

2.5 Implementation of ADO



* + Figure 2.5 The ADO .NET Implementation.

In this diagram, various types of applications (Web Application, Console Application, Windows Application, etc.) utilize ADO.NET to connect to databases such as SQL Server, Oracle, OleDb, ODBC, and XML files.

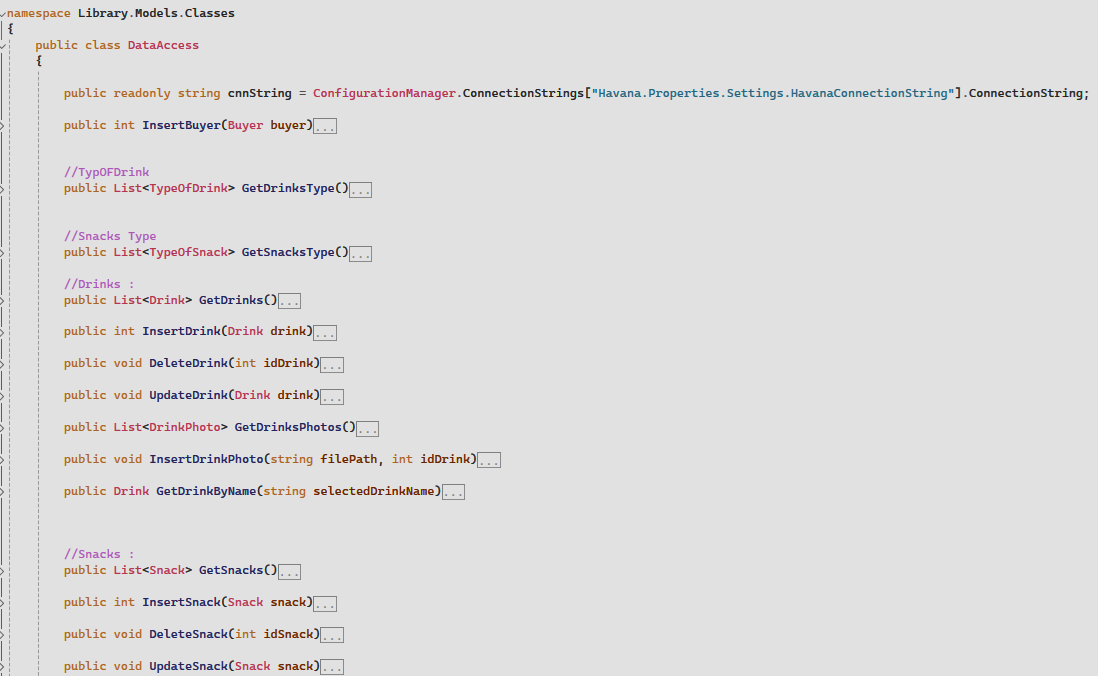
The important classes in ADO.NET are, Connection Class: Used for establishing connections to the database, managing transactions, and connection pooling. Learn more about connection classes in ADO.NET, Command Class: Provides methods for storing and executing SQL statements and Stored Procedures. Commands include ExecuteReader (returns data as rows), ExecuteNonQuery (executes commands that change data), ExecuteScalar (returns a single value), and ExecuteXMLReader (obtains data using an XML stream).

DataReader Class: Used to retrieve data by executing an SQL Select statement and accessing the returned rows. Learn more about Data Readers in C#.

DataAdapter Class: Connects DataSets to databases, facilitating the management of the connection between the application and database tables, views, and Stored Procedures. Learn more about Data Adapters in ADO.NET.

DataSet Class: The core of ADO.NET, consisting of a collection of DataTable objects. Each table contains DataColumn and DataRow objects, and the DataSet also includes a Relations collection for defining relations among Data Table Objects.

To be able to connect to a database using ADO.NET, you need to create a SqlConnection object with a connection string. The connection string includes key-value pairs like server, database, userid, and password.



* + Figure 2.6 The DataAccess Class.

In **DataAccess** class present my ADO implementation at this Application, the full code I provided at the end of this document, in this class I implement the **CRUD** for every table I had, create, Read, Update, and Delete (CRUD) are the four basic functions that models should be able to do, at most.

Because I want our models to provide four basic types of functionalities.the CRUD paradigm is common in constructing WPF applications, because it provides a memorable framework for reminding developers of how to construct full, usable models.

For example, **DeleteDrink()** method, which will delete drink by id , or **UpdateDrink()** method, will update drink information too,



* + Figure 2.7 Delling with photos using ADO.

At figure 2.7 shows how ADO can use ImageSource ad MemoryStream to provide the images from data base, and in this case I store the photo in the database as a byte array

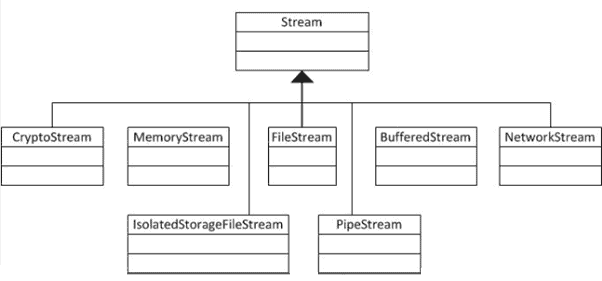
**Stream**, C# includes following standard IO (Input/Output) classes to read/write from different sources like files, memory, network, isolated storage, etc.

**Stream:** System.IO.Stream is an abstract class that provides standard methods to transfer bytes (read, write, etc.) to the source. It is like a wrapper class to transfer bytes. Classes that need to read/write bytes from a particular source must implement the Stream class.

The following classes inherit Stream class to provide the functionality to Read/Write bytes from a particular source:

* **FileStream** reads or writes bytes from/to a physical file, whether it is a .txt, .exe, .jpg, or any other file. FileStream is derived from the Stream class.
* **MemoryStream:** MemoryStream reads or writes bytes that are stored in memory.
* **BufferedStream:** BufferedStream reads or writes bytes from other Streams to improve certain I/O operations' performance.
* **NetworkStream:** NetworkStream reads or writes bytes from a network socket.
* **PipeStream:** PipeStream reads or writes bytes from different processes.
* **CryptoStream:** CryptoStream is for linking data streams to cryptographic transformations.

The following diagram shows the hierarchy of stream classes:

[](https://www.tutorialsteacher.com/Content/images/csharp/stream-heirarchy.png)

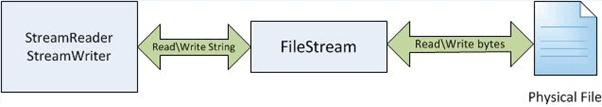
* + Figure 2.8 The hierarchy of stream classes

**StreamReader**: StreamReader is a helper class for reading characters from a Stream by converting bytes into characters using an encoded value. It can be used to read strings (characters) from different Streams like FileStream, MemoryStream, etc.

**StreamWriter**: StreamWriter is a helper class for writing a string to a Stream by converting characters into bytes. It can be used to write strings to different Streams such as FileStream, MemoryStream, etc.

**BinaryReader**: BinaryReader is a helper class for reading primitive datatype from bytes.

**BinaryWriter**: BinaryWriter writes primitive types in binary.

[](https://www.tutorialsteacher.com/Content/images/csharp/stream-relations.png)

* + Figure 2.9 FileStream reads bytes from a physical file

Figure 2.0 shows that FileStream reads bytes from a physical file, and then StreamReader reads strings by converting those bytes to strings. In the same way, the StreamWriter takes a string and converts it into bytes and writes to the FileStream, and then the FileStream writes the bytes to a physical file.

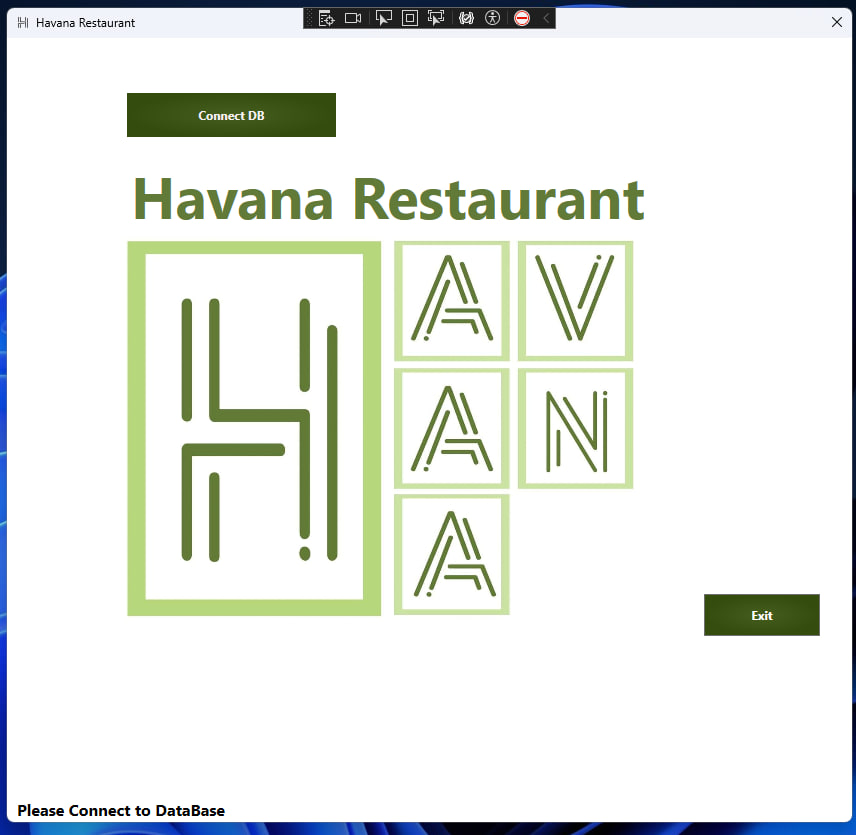
So, the FileStream deals with bytes, whereas StreamReader and StreamWriter deals with strings.

# 3. Test and describe user interface (UI/UX)

3.1 Task has been solved in full.

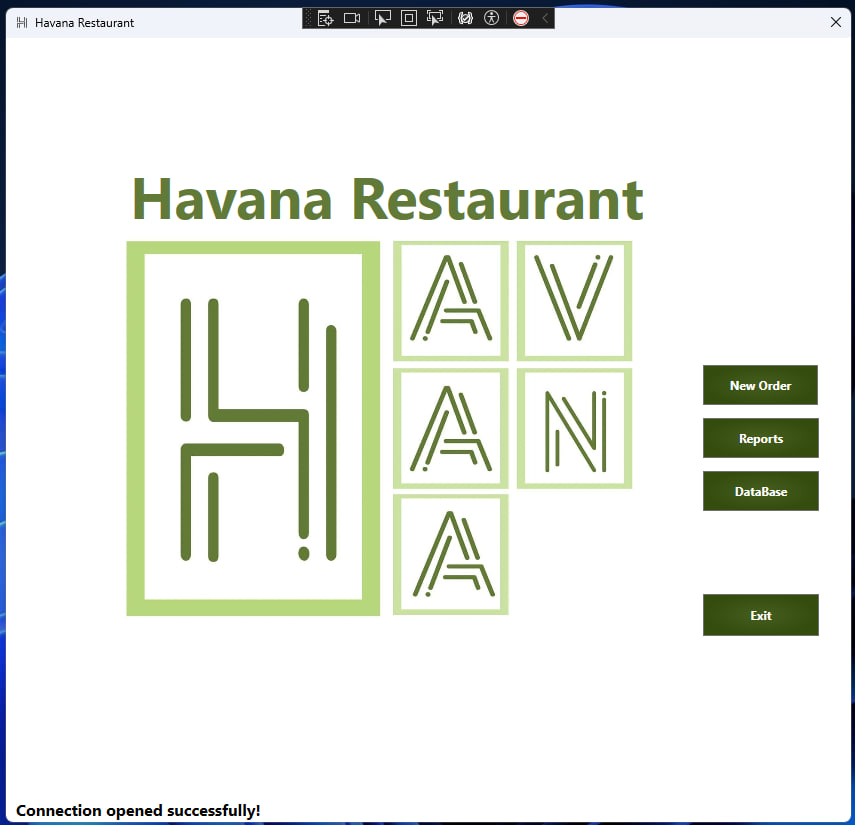
All the requirement that asked for this Application has been done, from the connection to data base, to makes new order to choose the drinks and snakes items, then to calculate the count and the total cost of this order, then to store this all-in database, and to store it in a file print and generate the order

3.2 Descript the user interface.



* + Figure 3.1 Main Page Window.

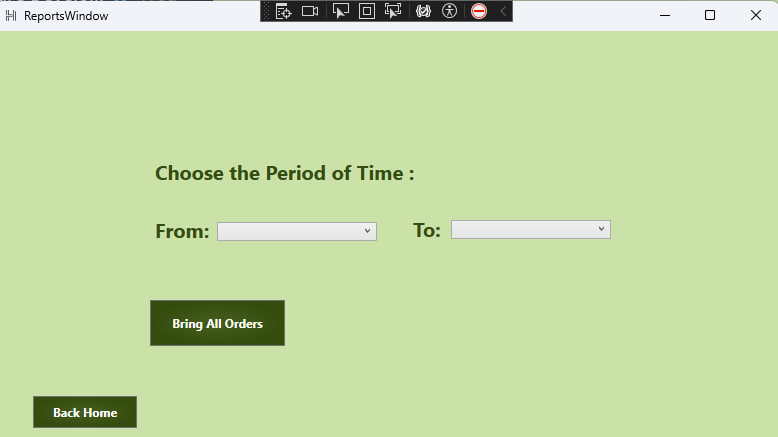
This is the main page of my application this show two buttons, **connect to DB** which is connect to the data base, and **Exit** button to close the program, the status bar on bottom show to the user to Connect to Database.



* + Figure 3.2 Main Page Window shows new Buttons.

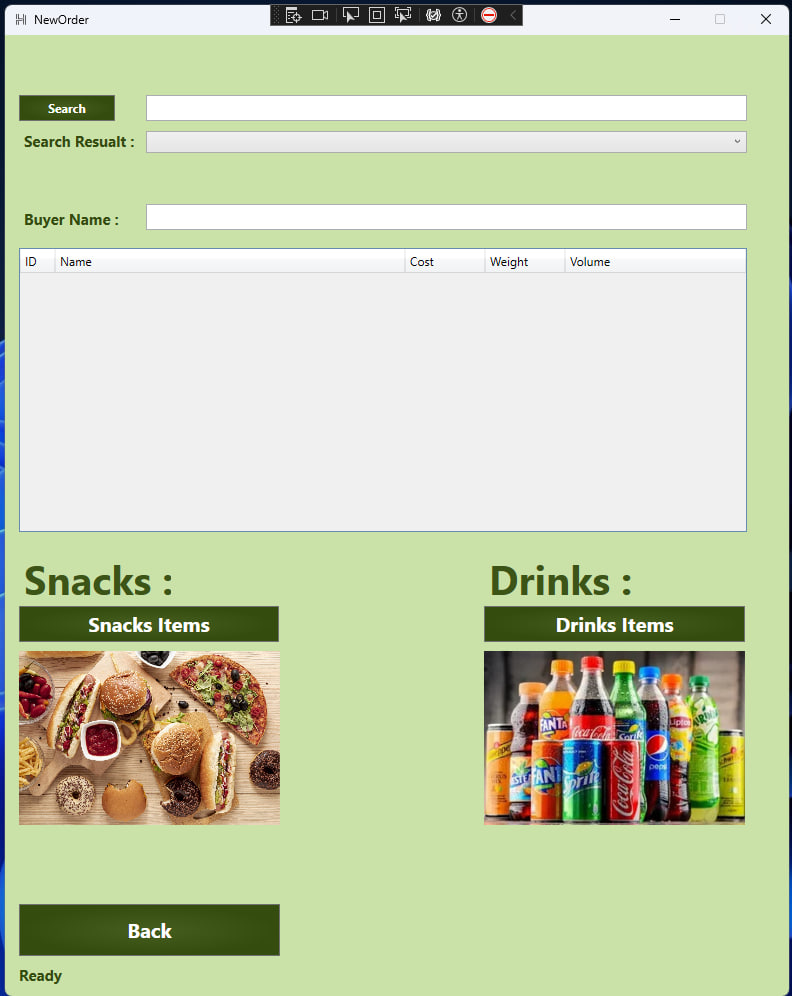
After the user Press to at Connect DB button, this button will disappear and, the status bar show us a Connection opened successfully.

And a new Buttons shows.



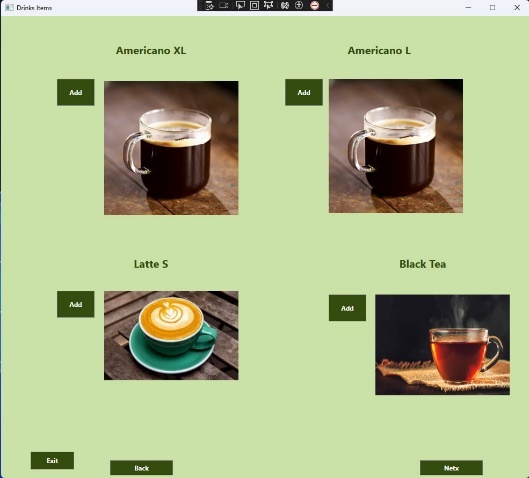
* + Figure 3.3 Report Window.

When the user Press on Reports the Application will open new Window, it has Date time to choose from Time to time , the Bring All Orders Buttons generate this Report and store it in file.txt at the Pc Document.



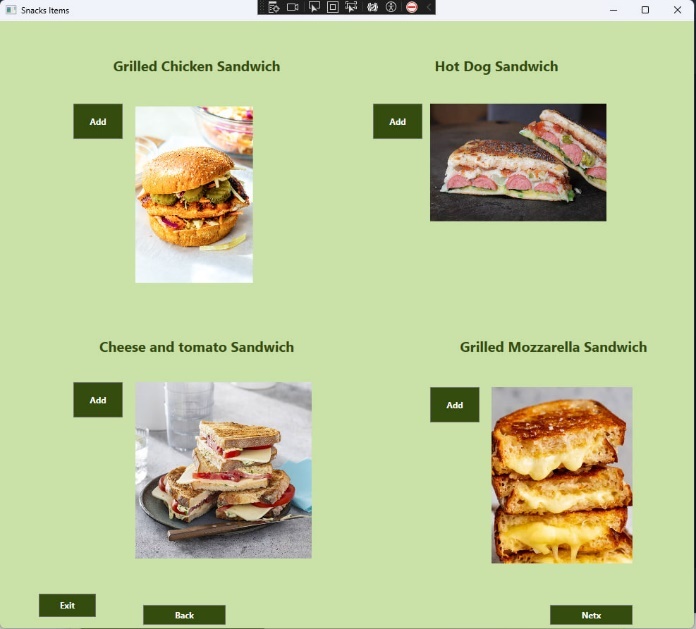
* + Figure 3.4 Order Window.

When the User Press at New Order Button, it chooses this window to him here he can choose between search by name or search by cost.



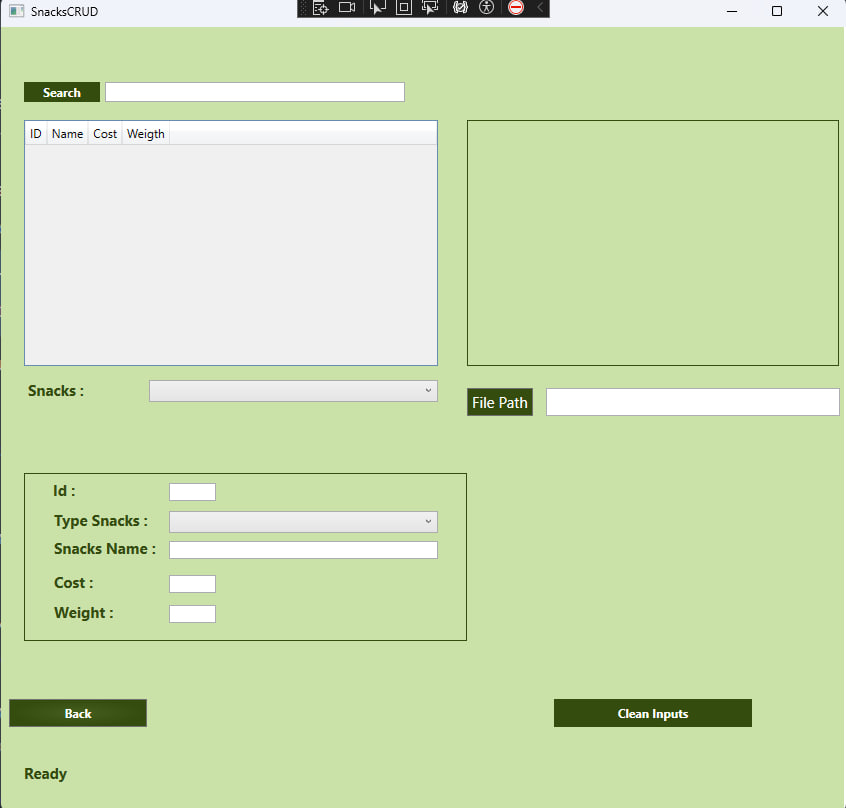
* + Figure 3.5 Shows drinks items

Here the user can add to his order what kind of drink he want.



* + Figure 3.6 Shows the Snacks Items.

Here the user can add to his order what kind of Snakes he want, when the user press to Add the items will be add to his order data grin in the order window.

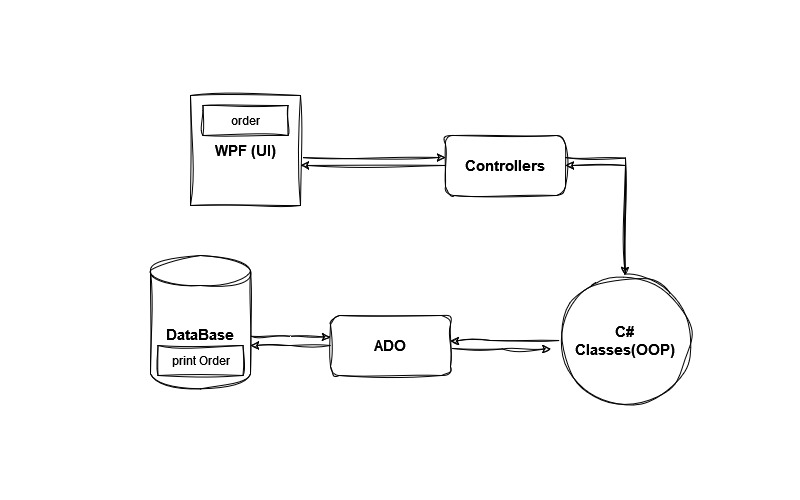


* + Figure 3.7 shows the Snacks Crud.

Here the user can edit or add or delete new Snack items, the buttons will show when the user will fill all the data required to make the prosses, clean Inputs will rest all the input , here we can find the items by search too, when we fined the items am we choose it , the application will fill automatically all the data in this .this textbox’s and combo box’s, sure we can choose also the item by combo box called snacks.

The Drink CRUD window is the same, as Snacks CRUD window.

3.3 Diagram of use case



* + Figure 3.8 Shows the order use case.

Figure 3.8 shows the order use case, when the user creates Order it until it reserve to the data base then from database until we print it back, or when we asked data from data base for different cases.

The application starts with the WPF user interface, where the user can perform various actions, such as adding, editing, or deleting customer records.

The WPF UI communicates with a set of controller classes, which act as the intermediaries between the UI and the business logic. These controllers receive user input from the UI, process the data, and pass it to the appropriate business logic components.

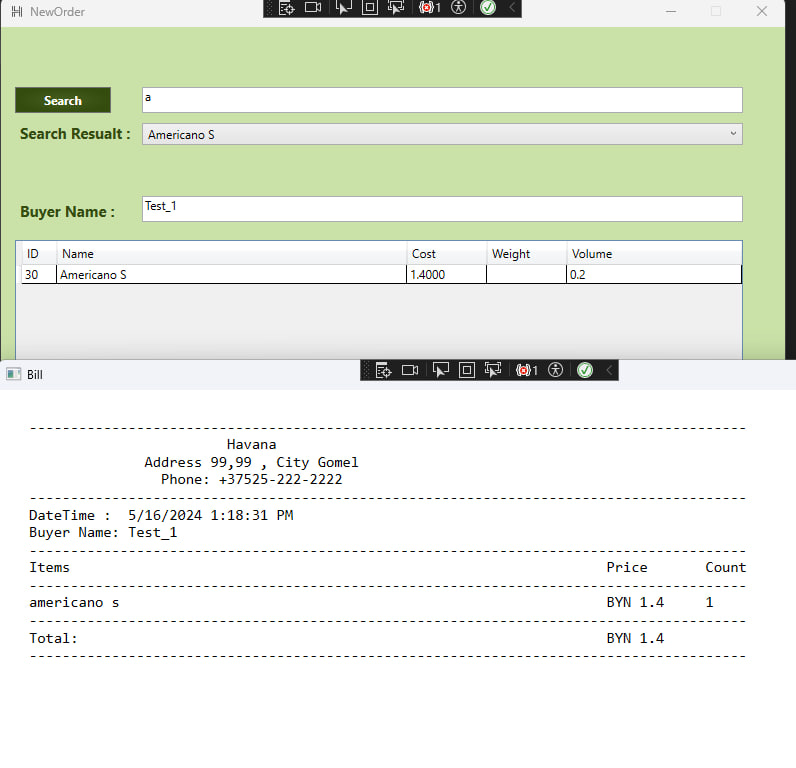
The business logic is implemented using Object-Oriented Programming (OOP) principles in C#. This includes defining classes that represent the various entities in the application, such as an Order class. These classes encapsulate the data and behavior related to customers.

To interact with the database, the application uses ADO.NET, a set of classes and interfaces that provide a data access layer. The controllers use ADO.NET to perform CRUD (Create, Read, Update, Delete) operations on the customer data stored in the database.

The customer data is stored in a relational database, SQL. The ADO.NET classes communicate with the database to execute SQL queries, insert new records, update existing ones, and delete records as needed.

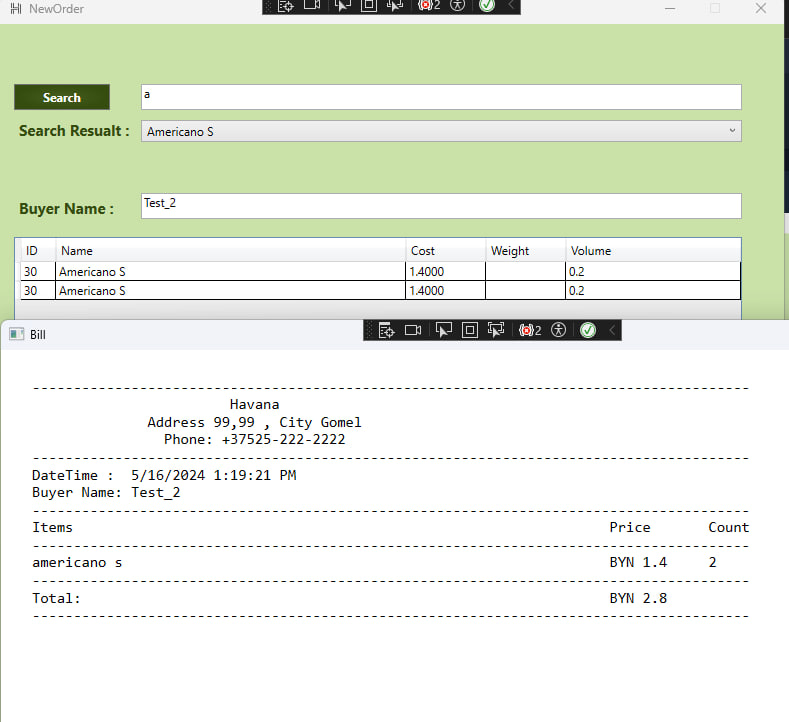
After the database operations are completed, the results are passed back to the WPF UI, where the user interface is updated to reflect the changes. For example, if a new customer was added, the UI would display the new Buyer in the list of buyers.

3.4 Develop Test



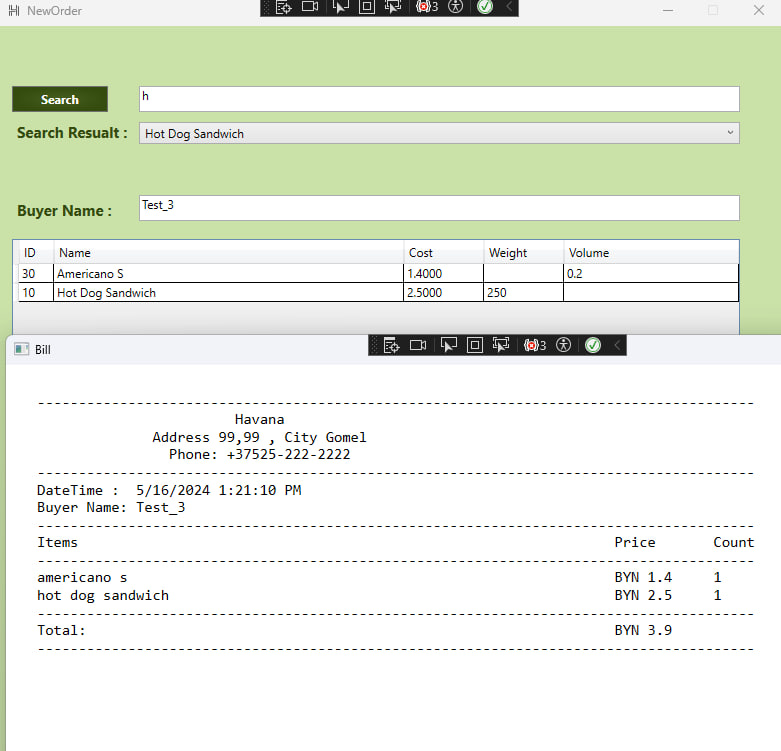
* + Figure 3.9 Test\_1.

In this test I try to order on drink and it give the Right Output, so the test is pass.



* + Figure 3.10 Test\_2

Here in Figure 3.10, I try to make two drinks in the same item to see if the Count will work correctly or not and its work, two small Americano, show us the Right output.



* + Figure 3.11 Test\_3.

Figure 3.11 shows Test of one Snack item and one Drink item and the output it as it should be also.



* + Figure 3.12 Test\_4

Figure 3.12 Test\_4 show order by one Drink and Two snacks and the output is corrected, the count it as should it be and the Toal output is right too .

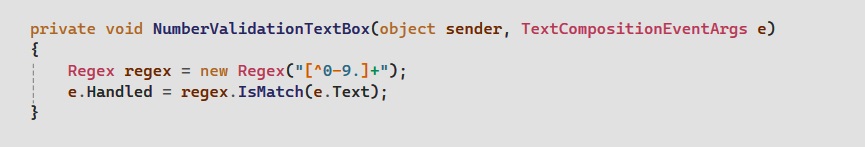
3.5 Validation Test

One of many validations that I did is every method has try and coach, that allow me to create a good way to handle the errors, and there is also the statues bar messages witch is a way that I use to tell the user what he should do in every step at this application, also I try my best to handled many wrong inputs like what I did in numerical numbers.

I let the controller to refuse any input except numbers from 0 to 9 and ‘.’ For float number, or sometimes I try to make the Id is read only to not be Charged somehow.

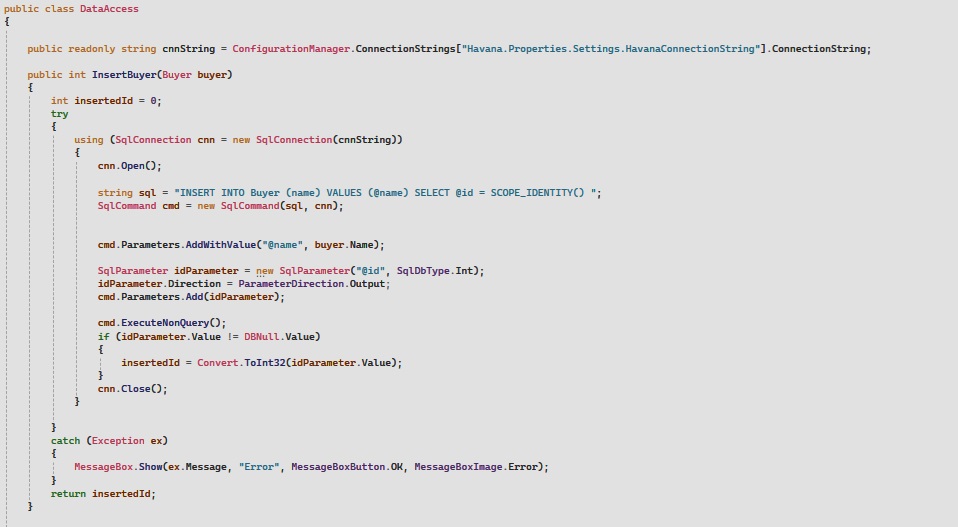


* + Figure 3.13 Show the Read Only for Id Textbox.



* + Figure 3.14 Shows the Number Validation Textbox.

Also, its not allow to change the connection string because that can make a very series security issue for this Application.



* + Figure 3.15 Show the Connection string and the Try catch exception.

In C#, the **try-catch** statement is used to handle exceptions that may occur during the execution of a program. Exceptions are events that disrupt the normal flow of a program and can occur due to various reasons, such as invalid user input, division by zero, file not found, and so on.

The try-catch statement consists of the following parts: try block: This block contains the code that might throw an exception. If an exception is raised within the try block, the program will immediately jump to the catch block.

The catch block: This block contains the code that handles the exception. The catch block specifies the type of exception it can handle, and the code within this block is executed when an exception of that type is thrown.

**Exception Hierarchies**, Exceptions in C# are organized in a hierarchy, with System.Exception being the base class. This means that you can catch a more general exception type and it will also handle more specific exception types that inherit from it.

# Conclusion

There is so many good technologies that I can use, but the best technologies that in my opinion good for my case at this application is: At First the main Programming Language is C#, Second the database is Ms-Sql, with ADO.Net and WPF for Interface.

The best things in all of these technologies that Microsoft build and develop them, with a good communicate between each other’s, And all these technologies are under the Microsoft Ecosystem, for fast and good performance and good support of .NET Community for them all, with future large-scale application, or even future troubles and Exception Handling that I will meet in later on.

All of them can be done by *Microsoft Visual Studio* which is the most famous IDE (Integrated Development Environment), support with Git for Control Versions of My application.

The Application that I create is good for small businesses and still miss a lot of important future like user validation and user authority, also its need more advance logic in the business calculation is need to be faster in ruing time in the future.

# References

Almog, Shai. 2023. *Java 8 to 21.* London: BPB Online.

DOC, MS-SQL. n.d. https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16.

MacDonald, Matthew. 2012. *Pro WPF 4.5 in C#.* Apress.

Microsoft, C# Doc. n.d. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

Patrick, Tim. 2010. *Microsoft ADO.NET 4 Step by Step.* California: O’Reilly Media.

postgresql. n.d. https://www.postgresql.org/docs/.

Richter, Jeffrey. 2012. *CLR via C# .* New York: Microsoft Press.

Skeet, jon. 2019. *C# in Depth .* Shelter Island, NY 11964: Manning.

SSMS, Microsoft. n.d. https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16.

----------------

HavanaRestaurant.xaml.cs

----------------

using Havana.Reports;

using System;

using System.Configuration;

using System.Data.SqlClient;

using System.Windows;

using Havana.DB;

using Havana.Orders;

namespace Havana.Main

{

/// <summary>

/// MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

readonly string connStr;

public MainWindow()

{

InitializeComponent();

connStr = ConfigurationManager.ConnectionStrings["Havana.Properties.Settings.HavanaConnectionString"].ConnectionString;

}

private void CloseButt(object sender, RoutedEventArgs e)

{

Close();

}

public void ShowButton(SqlConnection connection)

{

if (connection != null)

{

ConnectButton.Visibility = Visibility.Hidden;

OrderButton.Visibility = Visibility.Visible;

DataBaseButton.Visibility = Visibility.Visible;

ReportsButton.Visibility = Visibility.Visible;

}

}

public void ConnectButt(object sender, RoutedEventArgs e)

{

using (SqlConnection connection = new SqlConnection(connStr))

{

try

{

connection.Open();

StatusText.Text = "Connection opened successfully!";

// Perform database operations here...

ShowButton(connection);

}

catch (Exception ex)

{

StatusText.Text = "Error: " + ex.Message;

}

finally

{

//connection.Close();

//StatusText.Text = "Connection closed successfully!";

}

}

}

private void OrderButt(object sender, RoutedEventArgs e)

{

NewOrderWindow newOrder = new NewOrderWindow();

this.Visibility = Visibility.Hidden;

newOrder.Show();

}

private void DBButt(object sender, RoutedEventArgs e)

{

DataBaseInfo newOrder = new DataBaseInfo();

this.Visibility = Visibility.Hidden;

newOrder.Show();

}

private void ReportsButt(object sender, RoutedEventArgs e)

{

ReportsWindow reportsWindow = new ReportsWindow();

this.Visibility = Visibility.Hidden;

reportsWindow.Show();

}

}

}

----------------

Bill.xaml.cs

----------------

using Library.Models.Classes;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace Havana.Orders

{

/// <summary>

/// Interaction logic for Bill.xaml

/// </summary>

public partial class Bill : Window

{

public Order Order = null;

public Bill(Order order)

{

InitializeComponent();

ShowTheBill(order);

Order = order;

}

private void ShowTheBill(Order order)

{

string folderPath = System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "Bills");

Directory.CreateDirectory(folderPath);

string fileName = System.IO.Path.Combine(folderPath, $"Bill\_{order.DateTime:yyyyMMdd\_HHmmss}.txt");

try

{

string content = File.ReadAllText(fileName);

BillTextBlock.Text = content;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void PrintBill(Order order)

{

string folderPath = System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "Bills");

Directory.CreateDirectory(folderPath);

string fileName = System.IO.Path.Combine(folderPath, $"Bill\_{order.DateTime:yyyyMMdd\_HHmmss}.txt");

try

{

string content = File.ReadAllText(fileName);

BillTextBlock.Text = content;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void ExitButton\_Click(object sender, RoutedEventArgs e)

{

Close();

}

private void PrintButton\_Click(object sender, RoutedEventArgs e)

{

PrintBill(Order);

}

}

}

----------------------------------

NewOrderWindow.xaml.cs

----------------------------------

using System;

using System.Linq;

using System.Windows;

using Havana.Drinks;

using Havana.Snacks;

using Library.Models.Classes;

using Havana.Main;

using System.Windows.Controls;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

namespace Havana.Orders

{

public partial class NewOrderWindow : Window

{

DataAccess dataAccess = new DataAccess();

ListOfDrinks DrinksList = new ListOfDrinks();

ListOfSnacks SnacksList = new ListOfSnacks();

Buyer buyer = new Buyer();

public bool itemsAdded = false;

public NewOrderWindow()

{

InitializeComponent();

}

public void OpenWindow(Type windowType)

{

Window window = (Window)Activator.CreateInstance(windowType);

this.Visibility = Visibility.Visible;

window.Show();

}

private void DeleteSelectedItemButt\_Click(object sender, RoutedEventArgs e)

{

if (OrderDataGrid.SelectedItem != null)

{

OrderDataGrid.Items.Remove(OrderDataGrid.SelectedItem);

}

}

private void OrderDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

DeleteSelectedItemButt.Visibility = OrderDataGrid.SelectedItem != null ? Visibility.Visible : Visibility.Collapsed;

if (OrderDataGrid.Items.Count == 0)

{

itemsAdded = false;

ShowOrderButtons();

}

}

private void ItemsComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ItemsComboBox.SelectedValue != null)

{

if (ItemsComboBox.SelectedValue is Drink selectedDrink)

{

OrderDataGrid.Items.Add(selectedDrink);

itemsAdded = true;

ShowOrderButtons();

}

else if (ItemsComboBox.SelectedValue is Snack selectedSnack)

{

OrderDataGrid.Items.Add(selectedSnack);

itemsAdded = true;

ShowOrderButtons();

}

}

}

//ItemsButt

private void SnaksItemsButt(object sender, RoutedEventArgs e)

{

OpenWindow(typeof(SnacksItems));

}

private void DrinksItemsButt(object sender, RoutedEventArgs e)

{

OpenWindow(typeof(DrinksItems));

}

private void CheckBuyerName()

{

try

{

string name = BuyerNameTextBox.Text;

if (name != null && !string.IsNullOrWhiteSpace(name))

{

buyer.Id = -1;

buyer.Name = name;

buyer.Id = dataAccess.InsertBuyer(buyer);

OrderTextBlock.Text = name + " Has Been Added!! ";

}

else

{

MessageBox.Show("Please Add The Buyer Name First!", "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public void ShowOrderButtons()

{

if (itemsAdded is true && BuyerNameTextBox.Text != null)

{

Order.Visibility = Visibility.Visible;

}

else

{

Order.Visibility = Visibility.Collapsed;

}

}

//Buttons

private void OrderButt(object sender, RoutedEventArgs e)

{

CheckBuyerName();

Order newOrder = new Order();

DateTime setDateTime = DateTime.Now;

ListOfDrinks listOfDrinks = new ListOfDrinks();

ListOfSnacks listOfSnacks = new ListOfSnacks();

List<Drink> drinks = new List<Drink>();

List<Snack> snacks = new List<Snack>();

newOrder.Id = -1;

foreach (var item in OrderDataGrid.Items)

{

if (item is Drink drink)

{

Drink existingDrink = drinks.FirstOrDefault(d => d.Id == drink.Id);

if (existingDrink != null)

{

existingDrink.Count++;

}

else

{

drink.Count = 1;

drinks.Add(drink);

}

}

else if (item is Snack snack)

{

Snack existingSnack = snacks.FirstOrDefault(s => s.Id == snack.Id);

if (existingSnack != null)

{

existingSnack.Count++;

}

else

{

snack.Count = 1;

snacks.Add(snack);

}

}

}

if (listOfDrinks != null)

{

listOfDrinks.Id = -1;

}

if (listOfSnacks != null)

{

listOfSnacks.Id = -1;

}

//Sign Order data

newOrder.Id = -1;

newOrder.Name = "Table\_1";

newOrder.DateTime = setDateTime;

newOrder.BuyerName = buyer;

listOfDrinks.Drinks = drinks;

listOfSnacks.Snacks = snacks;

listOfDrinks.Count = drinks.Count;

listOfSnacks.Count = snacks.Count;

newOrder.DrinksList = listOfDrinks;

newOrder.SnacksList = listOfSnacks;

newOrder.TotalCost = newOrder.CalculateTotalCost();

ReportGenerator report = new ReportGenerator();

report.CreateBill(newOrder);

try

{

dataAccess.InsertOrder(newOrder);

Bill billWindow = new Bill(newOrder);

billWindow.WindowStartupLocation = WindowStartupLocation.CenterScreen;

billWindow.Show();

newOrder = null;

}

catch (SqlException ex)

{

MessageBox.Show("Failed to insert the order into the database: " + ex.Message);

}

}

private void SearchButt(object sender, RoutedEventArgs e)

{

DrinksList.Drinks = dataAccess.GetDrinks();

ItemsComboBox.Items.Clear();

string searchTerm = SearchTextBox.Text.ToLower();

foreach (Drink drink in DrinksList.Drinks)

{

if (drink.Name.ToLower().StartsWith(searchTerm) || drink.Cost.ToString().ToLower().StartsWith(searchTerm))

{

ItemsComboBox.Items.Add(drink);

}

}

SnacksList.Snacks = dataAccess.GetSnacks();

foreach (Snack snack in SnacksList.Snacks)

{

if (snack.Name.ToLower().StartsWith(searchTerm) || snack.Cost.ToString().ToLower().StartsWith(searchTerm))

{

ItemsComboBox.Items.Add(snack);

}

}

}

private void BackToHavana(object sender, RoutedEventArgs e)

{

MainWindow mainWindow = Application.Current.Windows.OfType<MainWindow>().FirstOrDefault();

if (mainWindow != null)

{

this.Visibility = Visibility.Hidden;

mainWindow.Visibility = Visibility.Visible;

}

}

public void BackToOrderWindow(NewOrderWindow newOrderWindow)

{

if (newOrderWindow != null)

{

this.Visibility = Visibility.Hidden;

newOrderWindow.Visibility = Visibility.Visible;

}

}

public void AddSnackItem(int id)

{

Snack snack = dataAccess.GetSnack(id);

OrderDataGrid.Items.Add(snack);

this.Show();

}

private void OrderDataGrid\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

//Do nothing

}

}

}

------------------------------------

AllOrders.xaml.cs

------------------------------------

using Library.Models.Classes;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace Havana.Reports

{

/// <summary>

/// Interaction logic for Bill.xaml

/// </summary>

public partial class AllOrders : Window

{

public List<Order> Orders = null;

public AllOrders(List<Order> orders)

{

InitializeComponent();

ShowTheBills(orders);

Orders = orders;

}

private void ShowTheBills(List<Order> orders)

{

string folderPath = System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "SummaryReports");

Directory.CreateDirectory(folderPath);

string fileName = System.IO.Path.Combine(folderPath, $"SummaryReport\_{orders[0].DateTime:yyyyMMdd\_HHmmss}.txt");

try

{

string content = File.ReadAllText(fileName);

BillTextBlock.Text = content;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void PrintBill(List<Order> orders)

{

}

private void ExitButton\_Click(object sender, RoutedEventArgs e)

{

Close();

}

private void PrintButton\_Click(object sender, RoutedEventArgs e)

{

PrintBill(Orders);

}

}

}

------------------------------------

ReportsWindow.xaml.cs

------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using Havana.Main;

using Library.Models.Classes;

namespace Havana.Reports

{

public partial class ReportsWindow : Window

{

public ReportsWindow()

{

InitializeComponent();

ShowDatesInComboBox();

}

private void ShowDatesInComboBox()

{

DataAccess dataAccess = new DataAccess();

List<string> dateTimeList = dataAccess.GetOrdersDates();

foreach (string dateTime in dateTimeList)

{

FromDateTime.Items.Add(dateTime);

ToDateTime.Items.Add(dateTime);

}

}

private void BackToHavana(object sender, RoutedEventArgs e)

{

MainWindow mainWindow = Application.Current.Windows.OfType<MainWindow>().FirstOrDefault();

if (mainWindow != null)

{

this.Visibility = Visibility.Hidden;

mainWindow.Visibility = Visibility.Visible;

}

}

private void BringAllOrders\_Click(object sender, RoutedEventArgs e)

{

try

{

string fromDateTime = FromDateTime.SelectedItem.ToString();

string toDateTime = ToDateTime.SelectedItem.ToString();

if(FromDateTime.SelectedItem != null && ToDateTime.SelectedItem != null)

{

DataAccess dataAccess = new DataAccess();

List<Order> orders = dataAccess.GetAllOrders(fromDateTime, toDateTime);

ReportGenerator report = new ReportGenerator();

report.SummeryBills(orders);

AllOrders allOrders = new AllOrders(orders);

allOrders.WindowStartupLocation = WindowStartupLocation.CenterOwner;

allOrders.Show();

}

else

{

MessageBox.Show("Please Add From TO Date Time : ","Error",MessageBoxButton.OK,MessageBoxImage.Error);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

------------------------------------

SnaksItems.xaml.cs

------------------------------------

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using Library.Models.Classes;

using Havana.Orders;

namespace Havana.Snacks

{

/// <summary>

/// Interaction logic for Hot.xaml

/// </summary>

public partial class SnacksItems : Window

{

int currentPage = 0;

int sizePage = 4;

List<SnackPhoto> photos = null;

public SnacksItems()

{

InitializeComponent();

DataAccess dataAccess = new DataAccess();

photos = dataAccess.GetSnacksPhotos();

ShowInfo(currentPage);

}

private void ShowInfo(int page)

{

int startPos = sizePage \* page;

int endPos = startPos + sizePage;

if (endPos > photos.Count)

{

endPos = photos.Count;

}

//Delete Data

for (int i = 0; i < 4; i++)

{

string ImageName = $"Image{i + 1}";

Image image = (Image)FindName(ImageName);

TextBlock textBlock = (TextBlock)FindName($"TextLable{i + 1}");

image.Source = null;

textBlock.Text = "";

string NameButton = $"AddButt{i + 1}";

Button button = (Button)FindName(NameButton);

if (image.Source == null)

{

button.Visibility = Visibility.Hidden;

}

}

//Fill Datat

for (int i = startPos; i < endPos; i++)

{

string ImageName = $"Image{i - startPos + 1}";

Image image = (Image)FindName(ImageName);

if (image != null)

{

image.Source = photos[i].Image;

}

string TexLableName = $"TextLable{i - startPos + 1}";

TextBlock textBlock = (TextBlock)FindName(TexLableName);

if (image != null)

{

textBlock.Text = photos[i].Snack.Name;

}

string buttonName = $"AddButt{i - startPos + 1}";

Button button = (Button)FindName(buttonName);

if (image != null)

{

button.Visibility = Visibility.Visible;

}

}

}

private void Exit(object sender, RoutedEventArgs e)

{

NewOrderWindow newOrderWindow = Application.Current.Windows.OfType<NewOrderWindow>().FirstOrDefault();

newOrderWindow.BackToOrderWindow(newOrderWindow);

this.Close();

}

private void AddButt(object sender, RoutedEventArgs e)

{

int startPos = sizePage \* currentPage;

int endPos = startPos + sizePage;

if (endPos > photos.Count)

{

endPos = photos.Count;

}

List<Button> buttons = new List<Button>();

for (int i = startPos; i < endPos; i++)

{

string buttonName = $"AddButt{i - startPos + 1}";

buttons.Add((Button)FindName(buttonName));

}

Button clickedButton = (Button)sender;

for (int i = startPos; i < endPos; i++)

{

if (buttons[i - startPos] == clickedButton)

{

NewOrderWindow newOrderWindow = Application.Current.Windows.OfType<NewOrderWindow>().FirstOrDefault();

Snack snack = photos[i].Snack;

newOrderWindow.OrderDataGrid.Items.Add(snack);

newOrderWindow.itemsAdded = true;

newOrderWindow.ShowOrderButtons();

newOrderWindow.Show();

break; // Exit the loop once the button is found

}

}

}

private void NextPage(object sender, RoutedEventArgs e)

{

currentPage++;

if (photos.Count < (currentPage \* sizePage))

{

currentPage--;

}

ShowInfo(currentPage);

}

private void Back(object sender, RoutedEventArgs e)

{

currentPage--;

if (currentPage < 0)

{

currentPage = 0;

}

ShowInfo(currentPage);

}

}

}

**Library**

------------------------------

Buyer.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class Buyer

{

public int Id { get; set; }

public string Name { get; set; }

public Buyer() { }

public Buyer(int id , string name)

{

this.Id = id;

this.Name = name;

}

public override string ToString()

{

return $"{Name}" ;

}

}

}

------------------------------

DataAccess.cs

------------------------------

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.IO;

using System.Windows.Media.Imaging;

using System.Windows.Media;

using System;

using System.Windows.Controls.Primitives;

using System.Xml.Linq;

using System.Data;

using System.Windows.Controls;

using System.Windows;

using static Library.Models.Classes.DataAccess;

using System.CodeDom.Compiler;

using System.Collections;

using System.Security.Cryptography;

using System.Data.Common;

using System.Reflection;

using System.Linq;

using System.Globalization;

namespace Library.Models.Classes

{

public class DataAccess

{

public readonly string cnnString = ConfigurationManager.ConnectionStrings["Havana.Properties.Settings.HavanaConnectionString"].ConnectionString;

public int InsertBuyer(Buyer buyer)

{

int insertedId = 0;

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string sql = "INSERT INTO Buyer (name) VALUES (@name) SELECT @id = SCOPE\_IDENTITY() ";

SqlCommand cmd = new SqlCommand(sql, cnn);

cmd.Parameters.AddWithValue("@name", buyer.Name);

SqlParameter idParameter = new SqlParameter("@id", SqlDbType.Int);

idParameter.Direction = ParameterDirection.Output;

cmd.Parameters.Add(idParameter);

cmd.ExecuteNonQuery();

if (idParameter.Value != DBNull.Value)

{

insertedId = Convert.ToInt32(idParameter.Value);

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return insertedId;

}

//TypOFDrink

public List<TypeOfDrink> GetDrinksType()

{

List<TypeOfDrink> typeOfDrinks = new List<TypeOfDrink>();

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string sql = "SELECT id, name FROM TypeOfDrink";

SqlCommand cmd = new SqlCommand(sql, cnn);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

TypeOfDrink typeOfDrink = new TypeOfDrink(reader.GetInt32(0), reader.GetString(1));

typeOfDrinks.Add(typeOfDrink);

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Erorr", MessageBoxButton.OK, MessageBoxImage.Error);

}

return typeOfDrinks;

}

//Snacks Type

public List<TypeOfSnack> GetSnacksType()

{

List<TypeOfSnack> typeOfSnacks = new List<TypeOfSnack>();

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string sql = "SELECT id, name FROM TypeOfSnack";

SqlCommand cmd = new SqlCommand(sql, cnn);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

TypeOfSnack typeOfSnack = new TypeOfSnack(reader.GetInt32(0), reader.GetString(1));

typeOfSnacks.Add(typeOfSnack);

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return typeOfSnacks;

}

//Drinks :

public List<Drink> GetDrinks()

{

List<Drink> drinks = new List<Drink>();

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string Request = "select id, name, cost, volume from dbo.Drink";

SqlCommand cmd = new SqlCommand(Request, cnn);

SqlDataReader Reader = cmd.ExecuteReader();

if (Reader.HasRows)

{

while (Reader.Read())

{

Drink drink = new Drink(Reader.GetInt32(0), Reader.GetString(1), Reader.GetDecimal(2), Reader.GetDouble(3));

drinks.Add(drink);

}

Reader.Close();

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return drinks;

}

public int InsertDrink(Drink drink)

{

int insertedId = 0;

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string insertDrinkSql = "INSERT INTO Drink (name, id\_type\_dr, cost, volume) " +

"VALUES (@name, @id\_type\_dr, @cost, @volume); " +

"SELECT @id = SCOPE\_IDENTITY();";

using (SqlCommand command = new SqlCommand(insertDrinkSql, connection))

{

command.Parameters.AddWithValue("@name", drink.Name);

command.Parameters.AddWithValue("@id\_type\_dr", drink.TypeOfDrinkId);

command.Parameters.AddWithValue("@cost", drink.Cost);

command.Parameters.AddWithValue("@volume", drink.Volume);

SqlParameter idParameter = new SqlParameter("@id", SqlDbType.Int);

idParameter.Direction = ParameterDirection.Output;

command.Parameters.Add(idParameter);

command.ExecuteNonQuery();

if (idParameter.Value != DBNull.Value)

{

insertedId = Convert.ToInt32(idParameter.Value);

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return insertedId;

}

public void DeleteDrink(int idDrink)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "DELETE FROM Drink WHERE Drink.id = @id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@id", idDrink);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

command.ExecuteReader().Close();

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public void UpdateDrink(Drink drink)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "UPDATE Drink SET Name = @Name, Cost = @Cost, Volume = @Volume, id\_type\_dr = @TypeOfDrinkId WHERE Id = @Id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Name", drink.Name);

command.Parameters.AddWithValue("@Cost", drink.Cost);

command.Parameters.AddWithValue("@Volume", drink.Volume);

command.Parameters.AddWithValue("@TypeOfDrinkId", drink.TypeOfDrinkId);

command.Parameters.AddWithValue("@Id", drink.Id);

command.ExecuteNonQuery();

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Erorr", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public List<DrinkPhoto> GetDrinksPhotos()

{

List<DrinkPhoto> drinkPhotos = new List<DrinkPhoto>();

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "Select DP.id, DP.photo ,D.Id, D.name, D.id\_type\_dr ,D.cost, D.volume From DrinkPhotos DP inner join Drink D on D.id = DP.id\_Drink";

SqlCommand command = new SqlCommand(query, connection);

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

int DrinkPhotoId = reader.GetInt32(0);

byte[] photoData = reader.GetFieldValue<byte[]>(reader.GetOrdinal("photo"));

Image ImageSource = new Image();

using (MemoryStream stream = new MemoryStream(photoData))

{

ImageSource.Source = BitmapFrame.Create(stream, BitmapCreateOptions.None, BitmapCacheOption.OnLoad);

int drinkId = reader.GetInt32(2);

string drinkName = reader.GetString(3);

int idTypeOFDrink = reader.GetInt32(4);

decimal drinkCost = reader.GetDecimal(5);

double drinkVolume = reader.GetDouble(6);

Drink drink = new Drink(drinkId, drinkName, drinkCost, drinkVolume, idTypeOFDrink);

DrinkPhoto drinkPhoto = new DrinkPhoto(DrinkPhotoId, ImageSource.Source, drink);

drinkPhotos.Add(drinkPhoto);

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return drinkPhotos;

}

public void InsertDrinkPhoto(string filePath, int idDrink)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

byte[] photoData = File.ReadAllBytes(filePath);

string query = "INSERT INTO DrinkPhotos (photo, id\_Drink) VALUES (@photo, @id\_Drink)";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@photo", photoData);

command.Parameters.AddWithValue("@id\_Drink", idDrink);

command.ExecuteNonQuery();

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public Drink GetDrinkByName(string selectedDrinkName)

{

Drink drink = null;

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "SELECT \* FROM Drink WHERE Name = @selectedDrinkName";

using (SqlCommand command = new SqlCommand(query, connection))

{

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

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

int id = reader.GetInt32(reader.GetOrdinal("id"));

string name = reader.GetString(reader.GetOrdinal("Name"));

decimal cost = reader.GetDecimal(reader.GetOrdinal("Cost"));

double volume = reader.GetDouble(reader.GetOrdinal("Volume"));

drink = new Drink(id, name, cost, volume);

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return drink;

}

//Snacks :

public List<Snack> GetSnacks()

{

List<Snack> snacks = new List<Snack>();

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string request = "SELECT id, name, cost , weigth FROM dbo.Snack";

SqlCommand cmd = new SqlCommand(request, cnn);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

Snack snack = new Snack(reader.GetInt32(0), reader.GetString(1), reader.GetDecimal(2), reader.GetDouble(3));

snacks.Add(snack);

}

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Erorr", MessageBoxButton.OK);

}

return snacks;

}

public int InsertSnack(Snack snack)

{

int insertedId = 0;

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string insertSnackSql = "INSERT INTO Snack (name, id\_type\_sn, cost, weigth) " +

"VALUES (@name, @id\_type\_sn, @cost, @weigth); " +

"SELECT @id = SCOPE\_IDENTITY();";

using (SqlCommand command = new SqlCommand(insertSnackSql, connection))

{

command.Parameters.AddWithValue("@name", snack.Name);

command.Parameters.AddWithValue("@id\_type\_sn", snack.TypeOfSnakId);

command.Parameters.AddWithValue("@cost", snack.Cost);

command.Parameters.AddWithValue("@weigth", snack.Weigth);

SqlParameter idParameter = new SqlParameter("@id", SqlDbType.Int);

idParameter.Direction = ParameterDirection.Output;

command.Parameters.Add(idParameter);

command.ExecuteNonQuery();

if (idParameter.Value != DBNull.Value)

{

insertedId = Convert.ToInt32(idParameter.Value);

}

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return insertedId;

}

public void DeleteSnack(int idSnack)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "DELETE FROM Snack WHERE Snack.id = @id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@id", idSnack);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

command.ExecuteReader().Close();

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public void UpdateSnack(Snack snack)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "UPDATE Snack SET Name = @Name, Cost = @Cost, Weigth = @Weigth, id\_type\_sn = @TypeOfSnackId WHERE Id = @Id";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@Name", snack.Name);

command.Parameters.AddWithValue("@Cost", snack.Cost);

command.Parameters.AddWithValue("@Weigth", snack.Weigth);

command.Parameters.AddWithValue("@TypeOfSnackId", snack.TypeOfSnakId);

command.Parameters.AddWithValue("@Id", snack.Id);

command.ExecuteNonQuery();

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public Snack GetSnack(int id)

{

Snack snack = null;

try

{

using (SqlConnection cnn = new SqlConnection(cnnString))

{

cnn.Open();

string request = "SELECT id, name, cost, weigth FROM dbo.Snack where id =@id";

SqlCommand cmd = new SqlCommand(request, cnn);

SqlParameter idParametr = new SqlParameter("id", id);

cmd.Parameters.Add(idParametr);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

int Id = reader.GetInt32(0);

string name = reader.GetString(1);

decimal cost = reader.GetDecimal(2);

double weight = reader.GetDouble(3);

snack = new Snack(Id, name, cost, weight);

}

cnn.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return snack;

}

public ImageSource GetSnackPhoto(int idSnack)

{

ImageSource imageSource = null;

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "SELECT photo FROM SnackPhotos WHERE id\_Snack = @id\_Snack";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@id\_Snack", idSnack);

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

byte[] photoData = (byte[])reader["photo"];

using (MemoryStream stream = new MemoryStream(photoData))

{

BitmapImage bitmapImage = new BitmapImage();

bitmapImage.BeginInit();

bitmapImage.StreamSource = stream;

bitmapImage.CacheOption = BitmapCacheOption.OnLoad;

bitmapImage.EndInit();

imageSource = bitmapImage;

}

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return imageSource;

}

public List<SnackPhoto> GetSnacksPhotos()

{

List<SnackPhoto> snackPhotos = new List<SnackPhoto>();

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "Select SP.id, SP.photo ,S.id, S.name, S.id\_type\_sn ,S.cost, S.weigth From SnackPhotos SP inner join Snack S on S.id = SP.id\_Snack";

SqlCommand command = new SqlCommand(query, connection);

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

int SnackPhotoId = reader.GetInt32(0);

byte[] photoData = reader.GetFieldValue<byte[]>(reader.GetOrdinal("photo"));

Image ImageSource = new Image();

using (MemoryStream stream = new MemoryStream(photoData))

{

ImageSource.Source = BitmapFrame.Create(stream, BitmapCreateOptions.None, BitmapCacheOption.OnLoad);

int snackId = reader.GetInt32(2);

string snackName = reader.GetString(3);

int idTypeOFSnack = reader.GetInt32(4);

decimal snackCost = reader.GetDecimal(5);

double snackWeigth = reader.GetDouble(6);

Snack snack = new Snack(snackId, snackName, snackCost, snackWeigth, idTypeOFSnack);

SnackPhoto drinkPhoto = new SnackPhoto(SnackPhotoId, ImageSource.Source, snack);

snackPhotos.Add(drinkPhoto);

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return snackPhotos;

}

public void InsertSnackPhoto(string filePath, int snackId)

{

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

byte[] photoData = File.ReadAllBytes(filePath);

string query = "INSERT INTO SnackPhotos (photo, id\_Snack) VALUES (@photo, @id\_Snack)";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@photo", photoData);

command.Parameters.AddWithValue("@id\_Snack", snackId);

command.ExecuteNonQuery();

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public Snack GetSnackByName(string selectedSnackName)

{

Snack snack = null;

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string query = "SELECT \* FROM Snack WHERE Name = @selectedSnackName";

using (SqlCommand command = new SqlCommand(query, connection))

{

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

using (SqlDataReader reader = command.ExecuteReader())

{

if (reader.Read())

{

int id = reader.GetInt32(reader.GetOrdinal("id"));

string name = reader.GetString(reader.GetOrdinal("Name"));

decimal cost = reader.GetDecimal(reader.GetOrdinal("Cost"));

double weigth = reader.GetDouble(reader.GetOrdinal("Weigth"));

snack = new Snack(id, name, cost, weigth);

}

}

}

connection.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

return snack;

}

public void InsertOrder(Order order)

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

// Insert the buyer into the Buyers table

string buyerQuery = "INSERT INTO Buyer (name) VALUES (@BuyerName); SELECT SCOPE\_IDENTITY();";

using (SqlCommand buyerCmd = new SqlCommand(buyerQuery, connection))

{

buyerCmd.Parameters.AddWithValue("@BuyerName", order.BuyerName.Name);

order.BuyerName.Id = Convert.ToInt32(buyerCmd.ExecuteScalar());

}

// Insert the order into the Orders table

string orderQuery = "INSERT INTO Orders (name, DateTime, id\_buyer) VALUES (@Name, @DateTime, @BuyerId); SELECT SCOPE\_IDENTITY();";

using (SqlCommand orderCmd = new SqlCommand(orderQuery, connection))

{

orderCmd.Parameters.AddWithValue("@Name", order.Name);

orderCmd.Parameters.AddWithValue("@DateTime", order.DateTime);

orderCmd.Parameters.AddWithValue("@BuyerId", order.BuyerName.Id);

order.Id = Convert.ToInt32(orderCmd.ExecuteScalar());

}

// Insert the drinks into the OrderItems table

if (order.DrinksList.Drinks != null && order.DrinksList.Drinks.Count > 0)

{

string drinkQuery =

@"

INSERT INTO ListOfDrinks (id\_order, id\_drink, count)

VALUES (@OrderId, @DrinkId, @DrinkCount);

";

foreach (Drink drink in order.DrinksList.Drinks)

{

using (SqlCommand drinkCmd = new SqlCommand(drinkQuery, connection))

{

drinkCmd.Parameters.AddWithValue("@OrderId", order.Id);

drinkCmd.Parameters.AddWithValue("@DrinkId", drink.Id);

drinkCmd.Parameters.AddWithValue("@DrinkCount", order.DrinksList.Count);

drinkCmd.ExecuteNonQuery();

}

}

}

// Insert the snacks into the OrderItems table

if (order.SnacksList.Snacks != null && order.SnacksList.Snacks.Count > 0)

{

string snackQuery =

@"

INSERT INTO ListOfSnacks (id\_order, id\_snack, count)

VALUES (@OrderId, @SnackId, @SnackCount);

";

foreach (Snack snack in order.SnacksList.Snacks)

{

using (SqlCommand snackCmd = new SqlCommand(snackQuery, connection))

{

snackCmd.Parameters.AddWithValue("@OrderId", order.Id);

snackCmd.Parameters.AddWithValue("@SnackId", snack.Id);

snackCmd.Parameters.AddWithValue("@SnackCount", order.SnacksList.Count);

snackCmd.ExecuteNonQuery();

}

}

}

}

}

public List<string> GetOrdersDates()

{

List<string> ordersDates = new List<string>();

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string sqlQuery = @"SELECT DateTime FROM Orders";

using (SqlCommand command = new SqlCommand(sqlQuery, connection))

{

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

DateTime orderDate = reader.GetDateTime(0);

string formattedDate = orderDate.ToString("yyyy-MM-dd HH:mm:ss.fff");

ordersDates.Add(formattedDate);

}

}

}

}

return ordersDates;

}

public List<Order> GetAllOrders(string startDate, string endDate)

{

var orders = new List<Order>();

try

{

using (SqlConnection connection = new SqlConnection(cnnString))

{

connection.Open();

string sqlQuery = @"

SELECT

O.id AS OrderID,

O.Name AS OrderName,

O.DateTime AS OrderDate,

B.id AS BuyerID,

B.name AS BuyerName,

D.id AS DrinkId,

D.name AS DrinkName,

D.cost AS DrinkCost,

D.volume AS DrinkVolume,

LD.id AS ListOfDrinksID,

LD.count AS ListOfDrinkCount,

S.id AS SnackId,

S.name AS SnackName,

S.cost AS SnackCost,

S.weigth AS SnackWeight,

LS.id AS ListOfSnacksID,

LS.count AS ListOfSnacksCount

FROM Orders O

INNER JOIN Buyer B ON O.id\_buyer = B.id

LEFT JOIN ListOfDrinks LD ON O.id = LD.id\_order

LEFT JOIN Drink D ON D.id = LD.id\_drink

LEFT JOIN ListOfSnacks LS ON O.id = LS.id\_order

LEFT JOIN Snack S ON S.id = LS.id\_snack

WHERE (LD.id\_order IS NOT NULL OR LS.id\_order IS NOT NULL)

AND O.DateTime BETWEEN @StartDate AND @EndDate ;";

using (SqlCommand command = new SqlCommand(sqlQuery, connection))

{

command.Parameters.AddWithValue("@StartDate", startDate);

command.Parameters.AddWithValue("@EndDate", endDate);

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

int orderIdIndex = reader.GetOrdinal("OrderID");

if (!reader.IsDBNull(orderIdIndex))

{

int orderId = reader.GetInt32(orderIdIndex);

Order order = orders.FirstOrDefault(o => o.Id == orderId);

if (order == null)

{

order = new Order

{

Id = orderId,

Name = reader.IsDBNull(reader.GetOrdinal("OrderName")) ? null : reader.GetString(reader.GetOrdinal("OrderName")),

DateTime = reader.GetDateTime(reader.GetOrdinal("OrderDate")),

BuyerName = reader.IsDBNull(reader.GetOrdinal("BuyerID")) || reader.IsDBNull(reader.GetOrdinal("BuyerName")) ? null : new Buyer

{

Id = reader.GetInt32(reader.GetOrdinal("BuyerID")),

Name = reader.GetString(reader.GetOrdinal("BuyerName"))

},

DrinksList = !reader.IsDBNull(reader.GetOrdinal("ListOfDrinksID")) ? new ListOfDrinks

{

Id = reader.GetInt32(reader.GetOrdinal("ListOfDrinksID")),

Count = reader.GetInt32(reader.GetOrdinal("ListOfDrinkCount")),

Drinks = new List<Drink>()

} : null,

SnacksList = !reader.IsDBNull(reader.GetOrdinal("ListOfSnacksID")) ? new ListOfSnacks

{

Id = reader.GetInt32(reader.GetOrdinal("ListOfSnacksID")),

Count = reader.GetInt32(reader.GetOrdinal("ListOfSnacksCount")),

Snacks = new List<Snack>()

} : null

};

orders.Add(order);

}

if (!reader.IsDBNull(reader.GetOrdinal("DrinkId")))

{

int drinkId = reader.GetInt32(reader.GetOrdinal("DrinkId"));

string drinkName = reader.GetString(reader.GetOrdinal("DrinkName"));

decimal drinkCost = reader.GetDecimal(reader.GetOrdinal("DrinkCost"));

double drinkVolume = reader.GetDouble(reader.GetOrdinal("DrinkVolume"));

Drink drink = new Drink(drinkId, drinkName, drinkCost, drinkVolume);

order.DrinksList.Drinks.Add(drink);

}

if (!reader.IsDBNull(reader.GetOrdinal("SnackId")))

{

int snackId = reader.GetInt32(reader.GetOrdinal("SnackId"));

string snackName = reader.GetString(reader.GetOrdinal("SnackName"));

decimal snackCost = reader.GetDecimal(reader.GetOrdinal("SnackCost"));

double snackWeight = reader.GetDouble(reader.GetOrdinal("SnackWeight"));

Snack snack = new Snack(snackId, snackName, snackCost, snackWeight);

order.SnacksList.Snacks.Add(snack);

}

order.TotalCost = CalculateTotalCost(order);

}

}

}

}

}

}

catch (Exception ex)

{

// Handle or log the exception

MessageBox.Show(ex.Message, "An error occurred while retrieving orders: ", MessageBoxButton.OK, MessageBoxImage.Error);

}

return orders;

}

private decimal CalculateTotalCost(Order order)

{

decimal totalCost = 0;

if (order.DrinksList != null && order.DrinksList.Drinks != null)

{

foreach (var drink in order.DrinksList.Drinks)

{

totalCost += drink.Cost;

}

}

if (order.SnacksList != null && order.SnacksList.Snacks != null)

{

foreach (var snack in order.SnacksList.Snacks)

{

totalCost += snack.Cost;

}

}

return totalCost;

}

}

}

------------------------------

Drink.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class Drink : Product

{

public double Volume { get; set; }

public int TypeOfDrinkId { get;set;}

public int Count { get; set; }

public Drink(int id, string name, decimal cost, double volume ) : base(id, name, cost)

{

Volume = volume;

}

public Drink(int id, string name, decimal cost, double volume, int typeOfDrinkId) : base(id, name, cost)

{

Volume = volume;

TypeOfDrinkId = typeOfDrinkId;

}

public Drink(int id, string name, int countofDrinks, decimal cost, double volume) : base(id, name, cost)

{

Volume = volume;

this.Count = countofDrinks;

}

public override string ToString()

{

return base.ToString() + $"\t{Volume}";

}

}

}

------------------------------

DrinkPhoto.cs

------------------------------

using Library.Models.Classes;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Media;

namespace Library.Models.Classes

{

public class DrinkPhoto

{

public int Id { get; set; }

public ImageSource Image { get; set; }

public Drink Drink { get; set; }

public DrinkPhoto(int id,ImageSource image ,Drink drink)

{

Id = id;

Image = image;

Drink = drink;

}

public override string ToString()

{

return $"{Drink.Id}\t{Drink.Name}\t{Drink.Cost}\t{Drink.Volume}";

}

}

}

------------------------------

ListOfDrinks.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Data.Common;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class ListOfDrinks

{

public int Id { get; set; }

public List<Drink> Drinks { get; set; }

public int Count { get;set;}

public ListOfDrinks(int id, int Count)

{

this.Id = id;

this.Drinks = new List<Drink>();

this.Count = Count;

}

public ListOfDrinks() { }

public override string ToString()

{

return $"{Id}\t{Drinks}\t{Count}";

}

}

}

------------------------------

ListOfSnacks.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class ListOfSnacks

{

public int Id { get; set; }

public List<Snack> Snacks { get; set; }

public int Count { get; set; }

public ListOfSnacks(int id ,int count)

{

this.Id = id;

this.Snacks = new List<Snack>();

this.Count = count;

}

public ListOfSnacks() { }

public override string ToString()

{

return $"{Id}\t{Snacks}\t{Count}";

}

}

}

------------------------------

Order.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Deployment.Internal;

namespace Library.Models.Classes

{

public class Order

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime DateTime { get; set; }

public Buyer BuyerName { get; set; }

public ListOfDrinks DrinksList { get; set; }

public ListOfSnacks SnacksList { get; set; }

public decimal TotalCost {get;set;}

public Order() { }

public Order(int id,DateTime dateTime, Buyer buyerName, ListOfDrinks drinksList, ListOfSnacks snacksList, decimal totalCost,string name)

{

Id = id;

Name = name;

DateTime = dateTime;

BuyerName = buyerName;

DrinksList = drinksList;

SnacksList = snacksList;

TotalCost = totalCost;

}

public int CalculateCoutOfSnacks()

{

SnacksList.Count = 0;

foreach (Snack snack in SnacksList.Snacks)

{

SnacksList.Count++;

}

return SnacksList.Count;

}

public int CalculateCoutOfDrinks()

{

DrinksList.Count = 0;

foreach (Drink drink in DrinksList.Drinks)

{

DrinksList.Count++;

}

return DrinksList.Count;

}

public decimal CalculateTotalCost()

{

TotalCost = 0;

if (DrinksList != null && DrinksList.Drinks != null)

{

foreach (Drink drink in DrinksList.Drinks)

{

TotalCost += drink.Cost \* drink.Count;

}

}

if (SnacksList != null && SnacksList.Snacks != null)

{

foreach (Snack snack in SnacksList.Snacks)

{

TotalCost += snack.Cost \* snack.Count;

}

}

return TotalCost;

}

public override string ToString()

{

return $"{Id}\t{Name}\t{DateTime}\t{BuyerName}\t{DrinksList}\t{SnacksList}\t{TotalCost}";

}

}

}

------------------------------

Order.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Deployment.Internal;

namespace Library.Models.Classes

{

public class Order

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime DateTime { get; set; }

public Buyer BuyerName { get; set; }

public ListOfDrinks DrinksList { get; set; }

public ListOfSnacks SnacksList { get; set; }

public decimal TotalCost {get;set;}

public Order() { }

public Order(int id,DateTime dateTime, Buyer buyerName, ListOfDrinks drinksList, ListOfSnacks snacksList, decimal totalCost,string name)

{

Id = id;

Name = name;

DateTime = dateTime;

BuyerName = buyerName;

DrinksList = drinksList;

SnacksList = snacksList;

TotalCost = totalCost;

}

public int CalculateCoutOfSnacks()

{

SnacksList.Count = 0;

foreach (Snack snack in SnacksList.Snacks)

{

SnacksList.Count++;

}

return SnacksList.Count;

}

public int CalculateCoutOfDrinks()

{

DrinksList.Count = 0;

foreach (Drink drink in DrinksList.Drinks)

{

DrinksList.Count++;

}

return DrinksList.Count;

}

public decimal CalculateTotalCost()

{

TotalCost = 0;

if (DrinksList != null && DrinksList.Drinks != null)

{

foreach (Drink drink in DrinksList.Drinks)

{

TotalCost += drink.Cost \* drink.Count;

}

}

if (SnacksList != null && SnacksList.Snacks != null)

{

foreach (Snack snack in SnacksList.Snacks)

{

TotalCost += snack.Cost \* snack.Count;

}

}

return TotalCost;

}

public override string ToString()

{

return $"{Id}\t{Name}\t{DateTime}\t{BuyerName}\t{DrinksList}\t{SnacksList}\t{TotalCost}";

}

}

}

------------------------------

Product.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public abstract class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Cost { get; set; }

public Product(int id, string name, decimal cost)

{

this.Id = id;

this.Name = name;

this.Cost = cost;

}

public virtual new string ToString()

{

return $"{Id}\t{Name}\t{Cost}";

}

}

}

------------------------------

ReportGenerator.cs

------------------------------

using Library.Models.Classes;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Linq.Expressions;

using System.Web;

using System.Windows;

using System.Windows.Controls;

using System.Globalization;

namespace Library.Models.Classes

{

public class ReportGenerator

{

public List<Order> OrderList { get; set; }

public ReportGenerator() { }

public ReportGenerator(List<Order> orderList)

{

this.OrderList = orderList;

}

public void CreateBill(Order order)

{

string folderPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "Bills");

Directory.CreateDirectory(folderPath);

string currencySymbol = "BYN ";

int itemNameColumnWidth = 70;

try

{

string fileName = Path.Combine(folderPath, $"Bill\_{order.DateTime:yyyyMMdd\_HHmmss}.txt");

if (File.Exists(fileName))

{

File.Delete(fileName);

}

using (StreamWriter sw = File.CreateText(fileName))

{

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine($"{"".PadRight(24)}Havana".PadRight(45));

sw.WriteLine($"{"".PadRight(14)}Address 99,99 , City Gomel");

sw.WriteLine($"{"".PadRight(16)}Phone: +37525-222-2222");

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine($"DateTime : {order.DateTime.ToString("M/d/yyyy h:mm:ss tt")}");

sw.WriteLine($"Buyer Name: {(order.BuyerName?.Name ?? "")}");

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine("Items".PadRight(itemNameColumnWidth, ' ') + "Price".PadRight(12, ' ') + "Count");

sw.WriteLine("".PadRight(87, '-'));

if (order.DrinksList != null)

{

foreach (Drink drink in order.DrinksList.Drinks)

{

string itemLine = $"{(drink.Name?.ToLower() ?? "")}";

int remainingSpace = itemNameColumnWidth - itemLine.Length;

string priceLine = $"{new string(' ', remainingSpace)}{currencySymbol}{drink.Cost.ToString("0.0#")}";

string countOfItems = $"".PadRight(5, ' ') + $"{order.DrinksList.Count}";

sw.WriteLine($"{itemLine}{priceLine}{countOfItems}");

}

}

if (order.SnacksList != null)

{

foreach (Snack snack in order.SnacksList.Snacks)

{

string itemLine = $"{(snack.Name?.ToLower() ?? "")}";

int remainingSpace = itemNameColumnWidth - itemLine.Length;

string priceLine = $"{new string(' ', remainingSpace)}{currencySymbol}{snack.Cost.ToString("0.0#")}";

string countOfItems = $"".PadRight(5, ' ') + $"{order.SnacksList.Count}";

sw.WriteLine($"{itemLine}{priceLine}{countOfItems}");

}

}

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine($"Total:".PadRight(itemNameColumnWidth) + $"{currencySymbol}{order.TotalCost.ToString("0.##")}");

sw.WriteLine("".PadRight(87, '-'));

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public void SummeryBills(List<Order> orders)

{

string folderPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "SummaryReports");

Directory.CreateDirectory(folderPath);

try

{

string fileName = Path.Combine(folderPath, $"SummaryReport\_{orders[0].DateTime:yyyyMMdd\_HHmmss}.txt");

if (File.Exists(fileName))

{

File.Delete(fileName);

}

using (StreamWriter sw = File.CreateText(fileName))

{

string currencySymbol = "BYN ";

int itemNameColumnWidth = 70;

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine($"{"".PadRight(24)}Havana".PadRight(45));

sw.WriteLine($"{"".PadRight(14)}Address 99,99 , City Gomel");

sw.WriteLine($"{"".PadRight(16)}Phone: +37525-222-2222");

sw.WriteLine("".PadRight(87, '='));

sw.WriteLine("\n\n");

decimal TotalRevenue = 0;

foreach (Order order in orders)

{

sw.WriteLine($"Order ID: {order.Id}");

sw.WriteLine($"DateTime: {order.DateTime.ToString("M/d/yyyy h:mm:ss tt")}");

sw.WriteLine($"Buyer Name: {(order.BuyerName?.Name ?? "")}");

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine("Items".PadRight(itemNameColumnWidth, ' ') + "Price".PadRight(12, ' ') + "Count");

sw.WriteLine("".PadRight(87, '-'));

decimal TotalCost = 0;

if (order.DrinksList.Drinks != null && order.DrinksList.Drinks != null )

{

foreach (Drink drink in order.DrinksList.Drinks)

{

string itemLine = $"{(drink.Name?.ToLower() ?? "")}";

int remainingSpace = itemNameColumnWidth - itemLine.Length;

string priceLine = $"{new string(' ', remainingSpace)}{currencySymbol}{drink.Cost.ToString("0.0#")}";

string countOfItems = $"".PadRight(5, ' ') + $"{order.DrinksList.Count}";

sw.WriteLine($"{itemLine}{priceLine}{countOfItems}");

TotalCost += order.DrinksList.Count \* drink.Cost;

TotalRevenue += TotalCost;

}

}

if (order.SnacksList != null && order.SnacksList.Snacks != null)

{

foreach (Snack snack in order.SnacksList.Snacks)

{

string itemLine = $"{(snack.Name?.ToLower() ?? "")}";

int remainingSpace = itemNameColumnWidth - itemLine.Length;

string priceLine = $"{new string(' ', remainingSpace)}{currencySymbol}{snack.Cost.ToString("0.0#")}";

string countOfItems = $"".PadRight(5, ' ') + $"{order.SnacksList.Count}";

sw.WriteLine($"{itemLine}{priceLine}{countOfItems}");

TotalCost += order.SnacksList.Count \* snack.Cost;

TotalRevenue += TotalCost;

}

}

sw.WriteLine("".PadRight(87, '-'));

sw.WriteLine($"Total:".PadRight(itemNameColumnWidth) + $"{currencySymbol}{TotalCost.ToString("0.##")}");

sw.WriteLine("".PadRight(87, '='));

}

sw.WriteLine();

sw.WriteLine($"Total Revenu = {TotalRevenue}");

sw.WriteLine();

sw.WriteLine("".PadRight(87, '='));

}

MessageBox.Show("Summary report generated successfully.");

}

catch (Exception ex)

{

MessageBox.Show($"Error generating summary report: {ex.Message}");

}

}

}

}

------------------------------

Snack.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class Snack : Product

{

public double Weigth { get; set; }

public int TypeOfSnakId { get; set; }

public int Count { get; set; }

public Snack(int id, string name, decimal cost, double weigth) : base(id, name, cost)

{

this.Weigth = weigth;

}

public Snack(int id, string name, decimal cost, double weigth,int typeOfSnackId ) : base(id, name, cost)

{

this.Weigth = weigth;

this.TypeOfSnakId = typeOfSnackId;

}

public Snack(int id, string name,int count, decimal cost, double weigth) : base(id, name, cost)

{

this.Weigth = weigth;

this.Count = count;

}

public override string ToString()

{

return base.ToString() + $"\t{Weigth}" ;

}

}

}

------------------------------

SnackPhoto.cs

------------------------------

using Library.Models.Classes;

using System;

using System.Windows.Media;

namespace Library.Models.Classes

{

public class SnackPhoto

{

public int Id { get; set; }

public ImageSource Image{ get; set; }

public Snack Snack { get; set; }

public SnackPhoto(int id, ImageSource image, Snack snack)

{

Id = id;

Image = image;

Snack = snack;

}

public override string ToString()

{

return $"{Snack.Id}\t{Snack.Name}\t{Snack.Cost}\t{Snack.Weigth} ";

}

}

}

------------------------------

TypeOfDrink.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class TypeOfDrink

{

public int Id { get; set; }

public string Name { get; set; }

public TypeOfDrink(int id , string name)

{

Id = id;

Name = name;

}

public override string ToString()

{

return $"{Id}\t{Name}";

}

}

}

------------------------------

TypeOfSnack.cs

------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Library.Models.Classes

{

public class TypeOfSnack

{

public int Id { get; set; }

public string Name { get; set; }

public TypeOfSnack(int id, string name)

{

Id = id;

Name = name;

}

public override string ToString()

{

return $"{Id}\t{Name}";

}

}

}