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

Specialty 1-40 05 01 «Information Systems and Technologies (by Industries) »

Course Work

Compiled by: student gr. IT-23 Нур-Эль-Дин Ававда

(signature, date)

Practice leader: Курочка Константин Сергеевич

(job title)

(signature, date)

Date of protection

Grade

Signatures of the commission members:

Gomel 2024

Contents

[Introduction 3](#_Toc162214394)

[Chapter.01 Technologies for Solutions: 4](#_Toc162214395)

[1.1 Why I Used C# as the main Programming for my Application? 4](#_Toc162214396)

[1.2 The difference between java and C#? 4](#_Toc162214397)

[1.3 Why did I choose MSSQL DBMS as a Database? 5](#_Toc162214398)

[1.4 What is MS SQL Server? 5](#_Toc162214399)

[1.5 What is SQL Server Management Studio SSMS? 5](#_Toc162214400)

[1.6 Why didn't I use PostgreSQL and What is PostgreSQL? 6](#_Toc162214401)

[1.7 Why didn't I use MySQL? 6](#_Toc162214402)

[1.8 What is ADO.NET Technology and why we use it? 7](#_Toc162214403)

[1.9 What is WPF? 7](#_Toc162214404)

[1.10 Why I use WPF not another GUI Technology for Example Windows Application? 8](#_Toc162214405)

[1.11 The Architecture of WPF? 9](#_Toc162214406)

[1.12 What is XAML? 10](#_Toc162214407)

[1.13 Dose C# Have A helpful type system at large and small scales? 10](#_Toc162214408)

[1.14 Typing terminology in C#? 10](#_Toc162214409)

[1.15 What is Static and Dynamic Typing Languages? 11](#_Toc162214410)

[1.16 What is Explicit and Implicit Typing in C#? 11](#_Toc162214411)

[1.17 What is the Major Components of ADO.NET? 11](#_Toc162214412)

[Conclusion 13](#_Toc162214413)

[References 14](#_Toc162214414)

# Introduction

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).* In this course work 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*).

So, this program should be able to make orders and keep all the data of sales in this Coffee Shop database. After an order 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 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 provided as A1 file of 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.

I 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, 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.

# Chapter.01 Technologies for Solutions:

1.1 Why I Used 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 Core. 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 The difference between 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 Why did I choose 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.

1.4 What is MS SQL Server?

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.)

1.5 What is SQL Server Management Studio SSMS?

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 re-create 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.)

1.6 Why didn't I use PostgreSQL and What is 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.)

1.7 Why didn't I use MySQL?

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 its 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.

1.8 What is ADO.NET Technology and Why We Use it?

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—data created in memory and used solely within an application—and external data—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)

1.9 What is 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)

1.10 Why I Use WPF Not Another GUI Technology For Example Windows Application?

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)

1.11 The Architecture of WPF?

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



1.12 What is XAML?

*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. More important, by exploring the tags in a WPF XAML document, you can learn a bit about the object model that underpins WPF user interfaces and get ready for the deeper exploration to come. (MacDonald, 2012)

1.13 Dose **C# Have 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)

1.14 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.*

1.15 What is 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 (and described in chapter 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.

1.16 What is Explicit and Implicit Typing in C#?

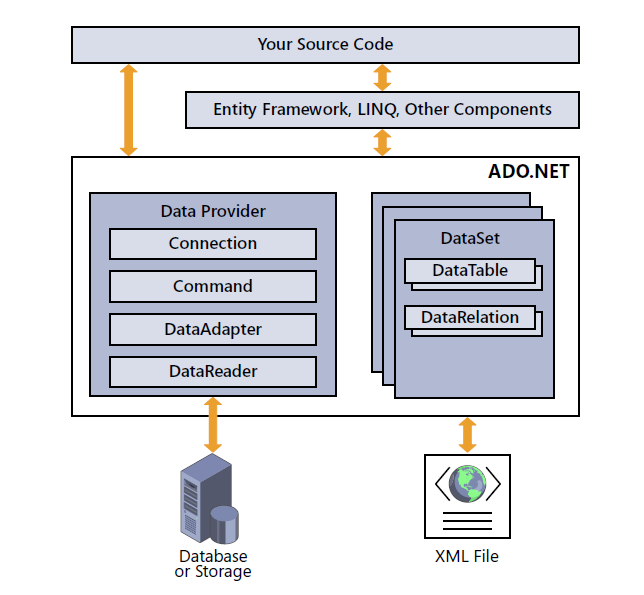
In a language that’s *explicitly typed*, the source code specifies all the types involved. This could be for local variables, fields, method parameters, or method return types, for example. A language that’s *implicitly typed* allows the developer to omit the types from the source code so some other 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)

1.17 What is 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)

# 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.

References

1. Almog, S. (2023). *Java 8 to 21.* London: BPB Online.
2. DOC, M.-SQL. (n.d.). Retrieved from https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16
3. MacDonald, M. (2012). *Pro WPF 4.5 in C#.* Apress.
4. Microsoft, C.# Doc (n.d.). Retrieved from https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
5. Patrick, T. (2010). *Microsoft ADO.NET 4 Step by Step.* California: O’Reilly Media.
6. postgresql. Doc (n.d.). Retrieved from https://www.postgresql.org/docs/
7. Richter, J. (2012). *CLR via C# .* New York: Microsoft Press.
8. Skeet, j. (2019). *C# in Depth .* Shelter Island, NY 11964: Manning.
9. SSMS, Doc (n.d.). Retrieved from https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16