# Activity 1

## Definition of an Algorithm

An algorithm is a procedure or a finite list of well-defined instructions or rules which establish a way to accomplish a particular task or solve a particular problem. Algorithms are the basis for computer science and programming-they constitute a detailed, constructive procedure for obtaining a desired output from any given input. (Garg, 2023)

## Process of Building an Application

When we build an application it involves several stages, each stage has specific tasks and objectives. Let's consider these stages,

### 1. Planning and analyzing

* **Gathering requirements**
  + **Define Objectives -** Understand and document what the user needs from the application and what kind of output they expect.
  + **Identify Constraints -** Consider technical, operational, and business constraints that might impact the project. (Garg, 2023)
* **Feasibility Study**

Usually, there are three feasibilities that we need to consider when we develop an application.

* + **Technical Feasibility –** technologies that are available to use for the application.
  + **Economic Feasibility –** the cost that is needed to develop the application and that cost should be affordable and it should worth it.
  + **Operational Feasibility –** infrastructure that the user or the organization for which we developing the application should have ability to support the application we developing. (Garg, 2023)

### 2. Design

* **System Design**
  + **Architectural Design –** drawing flow chart, DFD (Data Flow Diagrams), Database Structure (ERD, Database Schema). (Garg, 2023)
* **Interface Design**
  + **User Interface (UI) Design –** creating wireframes, prototype UI design etc.
  + **User Experience (UX) Design –** designing a working prototype of the design to ensure the user interface designed is satisfying user requirements. (Garg, 2023)
* **Algorithm Design**
  + **Choose Algorithms –** determining the functions, classes data structures and need to define algorithms to get the desired output that we need.
  + **Design and Optimize -** detailed designs for the algorithms designed and optimize them for get maximum performance. (Garg, 2023)

### 3. Development

* **Coding**
  + **Implement Features –** by choosing a proper language and an IDE to develop the application we have to develop the designed application and get the exact output that the user needs.
  + **Follow Coding Standards –** using coding standards we can make our code well organized and easy to understand. (Garg, 2023)

### 4. Testing

* **Unit Testing**

Testing each component of the application isolated and determine that each component gives the output that we need. (Garg, 2023) (GeeksForGeeks, 2024)

* **Integration Testing**

In this test we ensure that the components of the application are integrated properly. That means the data flowing one component to other is working correctly and they are connected as we need (Garg, 2023) (GeeksForGeeks, 2024)

* **System Testing**

Test the complete application to ensure that full application is satisfying the user requirement. (Garg, 2023) (GeeksForGeeks, 2024)

* **User Acceptance Testing (UAT)**

By giving the final application to users to check whether they can use it well and give feedback to what to improve. (Garg, 2023) (GeeksForGeeks, 2024)

### 5. Deployment

* **Prepare for Deployment**
  + **Configure Environments –** set up the environment as the application need such as servers, databases, other devices that communicate with the application, etc.
  + **Deploy Application –** take the application to the configured environment and set up the application. (Garg, 2023)
* **Monitor and Support**
  + **Monitor Performance –** monitor the application often and make sure there no errors and bugs and security of the application.
  + **Provide Support –** support for user about any error or bug that will occurs when using the application. (Garg, 2023)

### 6. Maintenance

* **Bug Fixes and Updates**
  + **Address Issues –** fix any bug or error that occurs and push security updates and other updates when the applications are needed.
  + **Implement Improvements –** add new features for user need and give them as they need. (Garg, 2023)
* **Documentation**
  + **Update Documentation -** Maintain and update technical and user documentation to reflect changes and improvements. (Garg, 2023)

### 7. Evaluation and Review

* **Performance Review**
  + **Assess Success -** Evaluate the application’s performance against the original goals and requirements.
  + **Collect Feedback -** Gather feedback from users and stakeholders to identify areas for improvement.
* **Plan Future Enhancements**
  + **Continuous Improvement -** Based on evaluation and feedback, plan for future updates and enhancements to keep the application relevant and effective. (Garg, 2023)

## Algorithms for given Scenarios

### A. Fibonacci Sequence Algorithm

#### Algorithm for Fibonacci Sequence

To compute the “nth” Fibonacci number, start by initializing the first two Fibonacci numbers- “F0​=0” and “F1​=1”. If the input “n” is 0, return “F0​”. If “n” is 1, return “F1​”. For “n” greater than 1, use a loop to compute the Fibonacci numbers from the 2nd up to the “nth” position. In each iteration, calculate the current Fibonacci number “Fi​” as the sum of the two preceding Fibonacci numbers “Fi−1​” and “Fi−2​”. After the loop completes, return the “nth” Fibonacci number “Fn”​.

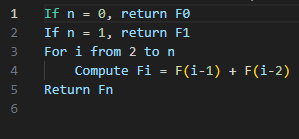


Figure 1 Algorithm for Fibonacci Sequence

#### C# Implementation

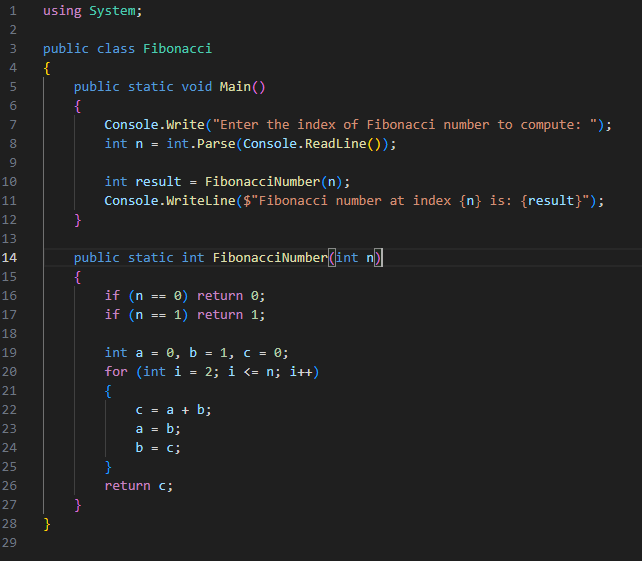


Figure 2 Fibonacci Sequence C# Implementation

### B. Factorial Algorithm

#### Algorithm for Factorial Calculation

To compute the factorial of a non-negative integer “n”, follow these steps - Start by checking if “n” is 0 or 1; if so, return 1 since the factorial of both 0 and 1 is 1. If “n” is greater than 1, initialize a variable called result to 1. Then, iterate from 2 up to “n”, multiplying result by each integer in this range. After completing the iteration, result will hold the factorial of “n”. Return this value as the final result.

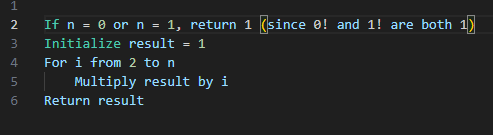


Figure 3 Algorithm for Factorial Calculation

#### C# Implementation

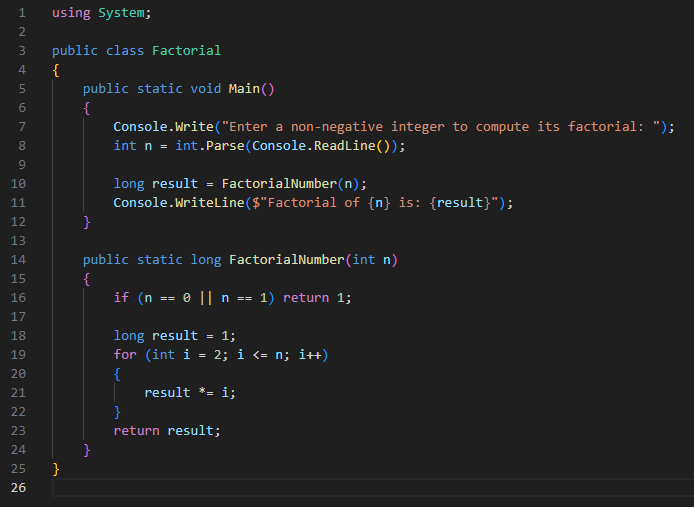


Figure 4 Factorial Calculation C# Implementation

## Analyzing the Process of Writing Code and Addressing Potential Challenges

Writing effective code involves several best practices and principles that ensure codes are clean, maintainable, and functional. The following is an analysis of each step in the process, including challenges a developer might face with some suggestions on overcoming them. (Falconer, 2022)

### 1. Start with a Plan

Writing code should always begin with a plan in which a developer can manipulate and modify within an efficient development process. Start by defining the inputs and outputs your program or function will have. Inputs refer to the data that will be fed to your program, while outputs refer to the results it will give out or data it will produce. Such aspects help in building the purpose and functionality of your code. Next, define what the desired output or results are to be, making sure that you understand what the outcome should be, and any limitations or special requirements that may affect how it is implemented.

Divide the problem into smaller sub-activities to simplify writing code for the problem. In particular, it describes a step-by-step process through which inputs are transformed into outputs. This helps in developing a way through which one may solve the problem systematically. Choose appropriate data structures depending on the type of data and the operations to be performed on the data - if the data represents lists of items in order, then arrays or lists can be used; for key-value pairs of data, dictionaries would be ideal.

Handling edge cases is critical because edge cases are abnormal or extreme values that may affect how your program behaves. Consider what edge cases you would want to handle, such as an empty input or a very large data set. Most coding challenges are with incomplete requirements or lack of good edge case handling. Poorly defined or incomplete requirements often lead to incorrect implementation. Make sure to iron out as much as possible with stakeholders. Not considering edge conditions will also fire up the bugs; therefore, comprehensive testing should be vital in testing your code against a number of conditions and ensuring that it acts as expected. (Falconer, 2022)

### 2. Write Meaningful Variable and Function Names

First and foremost, readability and clarity depend on meaningful variable and function naming when one writes the code. The use of different naming conventions structures code and/or standardizes it in one way or another for better understanding.

#### Camel Case

Camel Case is a popular naming convention in which the first letter of the variable or function name is in lowercase, and each subsequent word in that name starts off with an uppercase. Examples include firstName, lastName, and printFullName(). This convention is often used in languages like JavaScript and Java. (Falconer, 2022)

#### Pascal Case

Just like Camel Case, except that it starts with an uppercase letter. Examples include FirstName, LastName, and PrintFullName(). This is a common naming convention for naming classes in languages like C# and .NET. (Falconer, 2022)

#### Snake Case

This uses underscores to separate the words and all lowercase letters. Examples include first\_name, last\_name, and print\_full\_name(). This would be typical in Python and Ruby. (Falconer, 2022)

#### Kebab Case

This uses hyphens to separate words, all in lowercase, such as first-name, last-name, and print-full-name(). While this is not that common within most programming languages, it's used extensively in URL slugs and certain configuration files. (Falconer, 2022)

In regard to the naming convention, common challenges involve inconsistency and names that are not descriptive. Inconsistency in naming conventions makes code less readable, hence difficult to understand and follow by other developers. Following consistent naming conventions either from project rules or team rules will reduce this effect. Moreover, using non-descriptive names like temp or data might camouflage the purpose of variables or functions. Instead, use names which describe well their role or purpose; this makes your code more readable and easier to maintain. (Falconer, 2022)

### 3. Write Small, Modular Functions

Amongst the key practices that are usually suggested for writing clean, maintainable code is to write small, modular functions. A small function generally does no more than one task or responsibility. That makes it less complex because it is easy to understand, test, or reuse. You will reduce complex operations into smaller pieces. It enhances readability and reduces the chances of introducing bugs. (Falconer, 2022)

Other issues with this methodology are over-fragmentation and duplication. Over fragmentation is a condition where the code is fragmented down into a large number of small functions. Because it is always difficult to follow the overall structure, the trick is to balance this out by gathering together related functionality into coherent units without fragmenting the code excessively. (Falconer, 2022)

Another challenge is duplication, similar or identical code repeated in different functions. Minimize this by designing functions to be as reusable as possible and consolidating duplicated logic into shared functions. This will keep your code tidy, efficient, thereby facilitating ease of maintenance and extensibility over time. (Falconer, 2022)

### 4. Use Data Structures Appropriately

The ability to utilize data structures appropriately is integral to writing efficient and effective code. Choice of data structure directly impinges on the performance and, to a degree, on the complexity of your program.

Selection Challenges in Data Structures - Using Improper Data Structure - Includes data structure overhead, using the wrong data structure for a particular job results in inefficient code. For instance, a list is not the best data structure when an application frequently inserts or removes elements since operations like those are less efficient compared to using a linked list or a hash table. To avoid this, the most important thing is to give lots of thought to the ways the application will need to access data and its performance needs. So, in the case that quick lookups are needed, then a hash table could be used, otherwise ordered data would be best represented by a balanced tree. (Falconer, 2022)

The challenges include overhead, whereby some data structures introduce unnecessary complexity or consumption of resources. For instance, while performing simple tasks, one must avoid the use of complex data structures; this is so because they introduce these overheads, which in turn may make the code difficult to maintain and understand. To mitigate this, weigh the trade-offs between different data structures against ease of implementation, memory usage, and execution time. This will, in turn, allow for a fine-tuning of performance and maintainability of code, depending on the chosen data structure that best suits your needs. (Falconer, 2022)

### 5. Comment on the Code Necessarily

Commenting on code is one of the most important practices in making code readable and maintainable, but it has to be thoughtful for it to work.

Other challenges with comments involve out-of-date comments and over-commenting. The former relates to comments being left behind when code changes are implemented without updating the comments reflecting these changes. This can be confusing in that it will mislead other developers subsequently. Therefore, revise comments when code is changed to keep them in sync. (Falconer, 2022)

Over-commenting is the challenge wherein comments result in the code becoming noisier and less readable. Too many comments will make the code harder to read, and often obscure the logic rather than illuminate it. Instead, comment on why you're doing things, especially complex or nonobvious sections of your code. Comments should add value - explain why a block of code exists, detail logic or decisions that are not immediately obvious from the code itself. Strike a balance whereby comments clarify but do not overwhelm or divert attention from the main purpose of the code. (Falconer, 2022)

### 6. Indent Your Code for Readability

It is important to indent code because readability improves, and that is how maintenance should be done to a codebase. Proper indentation has the effect of separating blocks visually, hence highlighting how a program is structured. This will make it easier for other developers to follow through with your logic.

Some challenges that arise with indentation are inconsistent indentation and nested code. Inconsistent indentation causes problems in reading and, therefore, understanding your code, which finally creates errors and messes. To avoid this, one should be consistent with the indentation style throughout the codebase. The use of features in Integrated Development Environments-IDE-for code formatting keeps off many of the manual errors and, therefore, maintains consistency. (Falconer, 2022)

#### Nested code

When the code has blocks deeply inside of each other. It will be quite hard to read and manage. Too much nesting usually indicates an opportunity for refactoring. Refactor to reduce excessive indentation. Break up larger, complex functions into smaller, digestible ones; early returns and helper functions can be used to help with readability. This will make your code more understandable, can debug and maintainable. (Falconer, 2022)

### 7. Use Whitespace to Improve Readability

Of all things, in the process of writing clean and readable code, whitespace is the most important. Efficient whitespace will greatly help the person who actually reads your code to follow it much more comfortably.

Some of the challenges associated with the use of whitespace are cluttered code and misalignment. If too little whitespace is used, it is hard to read because logical blocks and different sections may run together. Therefore, whitespace must be used judiciously to separate distinct blocks of code, such as separation of functions, logical sections within a function, and statement types. This separation allows a reader to visually distinguish parts of your code and hence increases readability. (Falconer, 2022)

Other challenges include misalignment - inconsistent whitespace might cause code to be misaligned. It's a pain to read and makes tracking the structure of the code more difficult. Apply consistent indentation and spacing throughout the codebase. Align code elements such as variable declarations, parameters, and blocks for an orderly appearance. By addressing these challenges, you will manage to write code that is as functional as it is readable and maintainable.. (Falconer, 2022)

### 8. Use Arrays and Loops for Efficiency

Efficient use of arrays and loops definitely optimizes the performance of working with data within your program. Good design and good management are going to affect how efficient and maintainable your code is.

Other challenges of arrays and loops include inefficient loops and array management. Inefficient loops can take many forms, including a lot of iterations and redundant computation. While writing loops, one needs to put optimization in thought in order to reduce unnecessary operations and computation. This can be through the minimization of nested loop complexity, deployment of better algorithms, and deployment of early exits where possible to enhance loop performance. (Falconer, 2022)

Managing arrays is also problematic, especially when big data is an issue. The management of big arrays leads rather quickly to memory consumption and performance constraints in an application. Such large arrays can be managed by lists or hash tables that are optimized data structures in regard to the specific task at hand. Secondly, data segmentation, pagination, or streaming of large arrays are recommended techniques for handling those without overloading memory. (Falconer, 2022)

Solutions to these challenges will introduce ways that efficiency can be introduced into your code to handle large data sets or complex operations.

(Falconer, 2022)

### 9. Write Self-Documenting Code Whenever Possible

One of the important practices in writing readable, maintainable software is self-documenting code. Self-documenting code refers to code that is structured and named in such a way as to be readily understood by itself with little or no need for additional comments.

Writing self-documenting code tends mostly to come with challenges when associated with complexity and readability.

Complexity in code is when the logic is intrinsically labyrinthine and can't be captured with naming alone. Even with properly chosen variable and function names, complex logic may require additional comments or documentation explaining a particular purpose of why certain decisions are taken. In this case, careful commenting has to be done to explain the logic lying behind the code without over-commenting the same. (Falconer, 2022)

Next, comes the readability challenge. The fact that something may be self-documenting doesn't make that readable to people who are less comfortable with the code or at all. Solution - This shall include naming convention consistency, clarity in structure and proper formatting, which will further enhance the readability along with maintainability of smaller pieces of code after refactoring from a block of intricate code. Pay attention to these aspects so that your code may be self-documenting, readable, and the need for heavy external documentation is diminished, but others can understand and work with your code. (Falconer, 2022)

### 10. Don’t Repeat Yourself (DRY)

The DRY principle is one of the most important rules of software development where you try to limit duplication whenever possible by reusing code (you do not repeat yourself). Here are ways you can apply the DRY principle to various aspects of programming -

#### Functions and Modules

Encapsulation is a big element of the DRY principle. As such, you can use functions and modules as a way to encapsulate any reuseable code to cut down on duplication in your codebase. For example, Functions let you define a given piece of logic and call it according to your needs. Modules allow you to define a unit organization of related functions and code that are used together in a cohesive unit, so you write 'modular' code and don't duplicate similar pieces of logically grouped code where possible. (Falconer, 2022)

#### Data Structures

Saving storage is another aspect of the DRY principle. If you choose the correct data structure, you can save redundant data. For instance, if I were using a set, that will save me from saving duplicate elements. And with a hash table, I could quickly look up records and processes, and as a result, I wouldn't have to reprocess or resize my data. (Falconer, 2022)

#### Inheritance

Reuseable code can come from inheritance, as you can share code between classes. A base class can define functionality shared by derived 'child' classes. Instead of duplicating code for similar classes, derived classes can derive from a base class and pull in the base class's common functionality. (Falconer, 2022)

#### Libraries

Building off common libraries is another way to help maintain DRY. Libraries have functions and functionality built into them for common tasks, that you spend a bit of time integrating into your code; as such, you don't need to build a common function over and over again in each project you work on. As with any integration with third-party applications, take care to test and ensure you don't run into integration issues for any library - compatibility can be common with 3rd party libraries. (Falconer, 2022)

More challenges you might run into applying the DRY principle could be dealing with over-engineering or integration issues. Over-engineering could lead to problem over-complexity. Again, being practical about your data storage solution, while writing code as reusable is core to the DRY principle is key to limit unnecessary complexity. One area of that could include working with third-party libraries that one may introduce complex compatibility issues and the idea is the same - test to validate you are using acceptable solution that will integrate well to work well and be driven the way the libraries intend to be used in your application. (Falconer, 2022)

### 11. Write SOLID Code

The SOLID principles are essential principles for writing flexible and maintainable code. Below is a brief description of each of the principles and their troubles -

#### SOLID Principles

##### Single Responsibility Principle (SRP)

This principle states that a class or module should only have one reason to change, or in other words, one job or responsibility. This principle ensures that all components of your code are focused and easier to work with. (Falconer, 2022)

##### Open/Closed Principle (OCP)

Design your modules so that they are open for extension but closed for modification. In other words, greater module behavior is possible without changing the module implementation. The Open/Closed Principle allows for flexible systems that permit additional extension of functionality through code with greater ease and efficiency than the original implementation. (Falconer, 2022)

##### Liskov Substitution Principle (LSP)

Subclasses should replace base classes without altering the desirable properties of the program. In other words, subclasses are substitutable for instances of the base class without altering the contents of their abstractions, (i.e., for example, the correctness and performance of the original Base class). (Falconer, 2022)

##### Interface Segregation Principle (ISP)

Interfaces must be based on the interests of the client. Instead of a single, large interface, separate interfaces should be implemented for the clients based on their needs. This principle ensures that clients do not have to rely on interfaces that invoke client performance with methods that the client will never use. (Falconer, 2022)

##### Dependency Inversion Principle (DIP)

Depend on abstractions and not concrete implementations. High-level modules should not depend on low-level modules; both should depend on abstractions. The principle of abstraction lessens the coupling between isolated code features in your system so that changes to functions can be implemented with limited awareness of coupling conflicts. (Falconer, 2022)

Obstacles associated with following SOLID principles include ease of adherence and the refactoring overhead. Following SOLID principles very strictly can, on occasion, increase complexity resulting in the design becoming too abstracted or muddled into different pieces. Finding and following an appropriate amount of adherence is important to balance what is useful from SOLID and not be overly complex in a code base. Also, they might find themselves in a position where the existing code has to incur a refactoring overhead to apply SOLID principles, requiring adversarial changes to existing code. Planning and executing to cost and onboard any changes resulting in codified variety and apply newfound understanding of the principles can significantly improve existing code and finds efficiencies without introducing new problems to existing code. (Falconer, 2022)

# Activity 2

## Programming Paradigms Overview

### 1. Procedural Programming

Procedural programming denotes a programming kind that is based on a step-by-step sequence of procedures or instructions to solve a problem. Procedural programming organizes programs as sequences of functions (or procedures (Moo ICT, n.d.)).

#### Characteristics

##### Procedures/Functions

In with procedural programming, code is organized into reusable blocks of code called procedures or functions. Each of these blocks is written to perform a specific task, and can be called using standard syntax from different parts of the program - functions enable modularity and the reuse of similar code, and they isolate a single piece of functionality. (Moo ICT, n.d.)

##### Sequential Execution

A program written in procedural programming is typically executed in a sequential manner - meaning that executed code will follow a known linear structure. Any lines of code will execute from top to bottom, until the end of the block is reached, unless instructed otherwise using control statements. (Moo ICT, n.d.)

##### Local and Global Variables

Variables in procedural programming can be local variables or global variables. Local variables are declared and used within a function or procedure, and are accessible only via that function. Conversely, global variables are declared outside of functions or procedures, and will be accessible from anywhere within the program. Combining local and global variables can enable both encapsulation, and broader accessibility (Moo ICT, n.d.)

##### Control Structures

Procedural programming utilizes control states, such as loops (for, while) or conditionals (if, switch) used for a sequential flow to expedite the flow logic. Loops enable program flow through the repetition of an action, while conditionals enable program flow based on yes/no decisions controlling the flow of a program. (Moo ICT, n.d.)

##### Modularity

Programs are divided into smaller functions or modules that are manageable. Modular programming improves code readability, allows simple testing of isolated components, and allows for simple debugging for future work. Each function/module is used to address some aspect of the program which helps break the program into logical and manageable aspects. (Moo ICT, n.d.)

#### Example

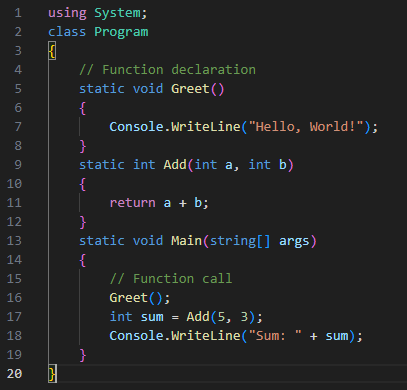


Figure 5 Procedural Paradigm

#### Applications

Essentially, procedural programming is most appropriate for simple programming tasks in which you can delineate the problem into succinct procedures or functions. This can complement circumstances where the problem can be solved with linear, definable, manageable steps. The ability to place several tasks in neatly defined functions keeps tasks relatively approachable, organized, and practical, particularly in straightforward sections of the program. (Moo ICT, n.d.)

### 2. Event-Driven Programming

Event-driven programming is a programming paradigm where the program's flow is directed by events, such as the user or signals generated by the computer or the system. This idiom is commonly used for graphical user interface (GUI) applications, web applications, and real-time systems with dynamic event-driven programs or functionalities triggered by various events. (Moo ICT, n.d.)

#### Characteristics

##### Event Source

An event source is an event-generating object or component. For example, a button on a user interface can be an event source that means an event will be generated when the user clicks on this button. Event sources can also include all user interface objects, such as text fields, menus, and server or application-based components, such as a sensor or a network connection. (Moo ICT, n.d.)

##### Event Listener

An event listener is a code or method that is waiting for the event to occur. It is basically code that is always running and looking for the trigger that an event has occurred, whether user-related, or changes in the computer's state. Event listeners generate responses to events generated from a particular place. (Moo ICT, n.d.)

##### Event Handler

An event handler responds to the event with the execution of some function or method. When an event occurs the appropriate event listener will call the appropriate event handler, at which point the events the programmer will include some action, such as updating the display, or doing a calculation. (Moo ICT, n.d.)

#### Example

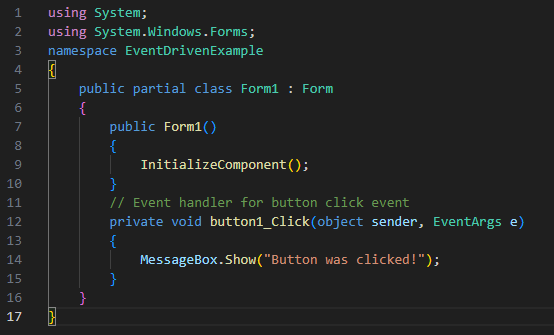


Figure 6 Event Driven Paradigm

#### Applications

##### Graphical User Interfaces

These are used to handle several simultaneous events that a user initiates with the help of a GUI. In desktop applications, event-driven code runs user interaction actions for button clicks, menu selection, window resizing, and many other such events. In web applications, event-driven programming handles user input events, form submissions, and generates dynamic content updates. (Moo ICT, n.d.)

##### Responsive User Interfaces

Event-driven programming provides a technique in numerous applications where responsive and interactive interfaces are required, be it desktop or web apps. Applications requiring providing immediate feedback for user experiences and real-time updates include examples like - a web chat application makes use of event-driven programming so as to ensure that the chat window updates upon the arrival of new messages without refreshing any web page. (Moo ICT, n.d.)

### 3. Object-Oriented Programming (OOP)

Object-oriented programming, abbreviated as OOP, is known as a coding paradigm that encompasses the use of "objects" comprising the associated data and the protocols on processing data. OOP was developed under the premise of simulating the intersections of entities/reminder, therefore resulting in a suitable design for compartmentalizing and managing code in a modular manner for a better design. (Moo ICT, n.d.)

#### Characteristics

##### Encapsulation

Encapsulation is the process of packing data (attributes) and code as a single unit or class. This means that parts of the code are organized in the program as discrete objects with their own copy of various attributes and code (methods).

##### Inheritance

Inheritance enables a new class (subclass) to derive attributes and methods from a given class (superclass). This provides benefits for code reuse and develops a relationship of hierarchy amongst classes. For instance a Dog class that inheriting from a Animal class, inherits those attributes and methods that are common to all animals while being able to add it's own unique attributes and methods. Inheritance also better organizes and extends the functionalities of the code.

##### Polymorphism

Polymorphism allows objects of different classes to be treated as an object of a common superclass. In other words, we can define a method in the superclass and override it in any subclass, using the same method call on different objects that respond appropriately to its class. For instance, a Shape superclass might define a draw method that is overridden by subclasses, Circle and Rectangle, that define how to draw their shapes or objects.

##### Abstraction

Abstraction is the idea of concealing complex implementation details and presenting the necessary features of an object when defining it. Abstraction when implemented provides a clear and concise interface for interacting with an object, while that interface hides the complexities of implementation. For example, a Car class might define methods like startEngine and drive, but would not present the complexities of implementation details of its internal engine mechanics. (Moo ICT, n.d.)

#### Example

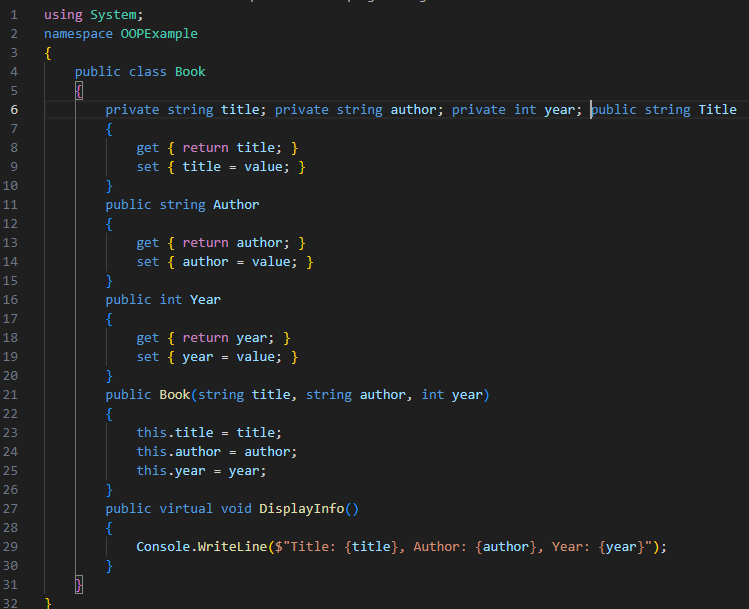
****

Figure 7 Onject Oriented paradigm Defineing Base Class

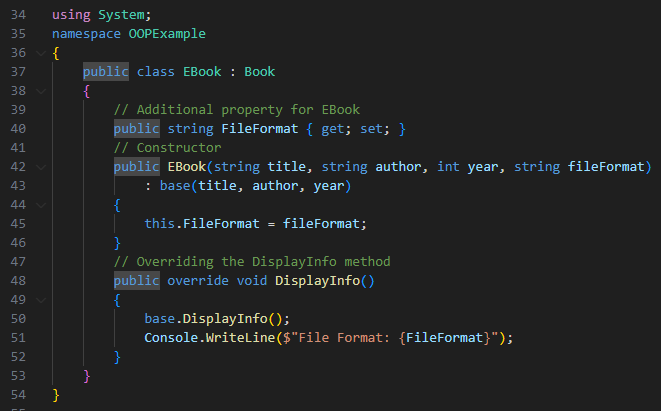


Figure 8 Object Oriented Paradigm Defining Derived Class

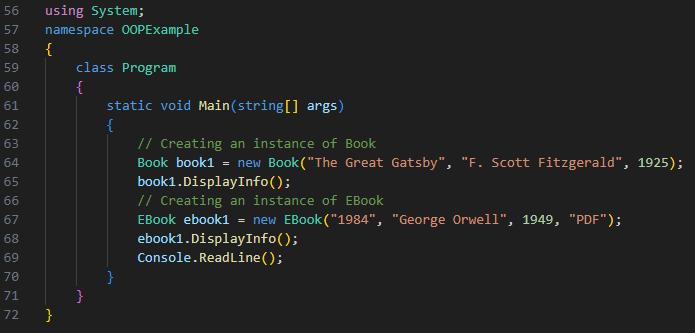


Figure 9 Object Oriented paradigm Defining Main Program

#### Applications

##### Complex Systems

One of the advantages of OOP is that it finds its best application in complex systems, where the modeling of real entities in the world is vitally important. This results in a modular, structured system that models relationships in the real world and interactions between these entities, where encapsulation of data and behavior is performed. Thus, it brings up clearer designs, simpler, better scalability when dealing with large and complex software systems. (Moo ICT, n.d.)

##### Large-Scale Software Development

The OOP approach can be found in many large-scale software systems, including frameworks, enterprise applications, etc., where modularity, code reuse, and maintainability matter much. That is so because OOP enables one to break the code into manageable chunks, which makes it easier to build up and maintain during the lifetime. For instance, if some enterprise applications are built using OOP, then modeling of business processes, user interactions, and data management will get encapsulated, leading to better design/ manipulation than if OOP had not been used. (Moo ICT, n.d.)

##### Game Development

In game development, OOP is applied everywhere, where there can be a number of entities in existence, for instance, but each has different attributes and/or behaviors while all having some level of shared functionality. This allows for inheritance and polymorphism whereby game developers can easily create a base class with the shared behaviours-that is, whether movement or interaction-yet still allow that entity to have unique classes to function as needed. (Moo ICT, n.d.)

##### Frameworks and Libraries

Most of the well-known frameworks/libraries for programming are based on OOP. For example, web development frameworks like Django (Python) and Ruby on Rails (Ruby) apply OOP principles for maintaining the components in models, views, and controllers, giving a clear separation of functions.. (Moo ICT, n.d.)

# Activity 3

## Pseudo Code

### Employee Application

#### Employee Login

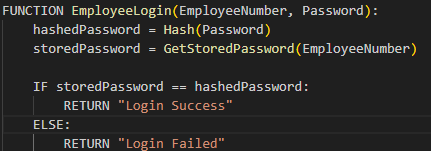


Figure 10 Pseudo Code Employee Login

#### Apply for Leaves

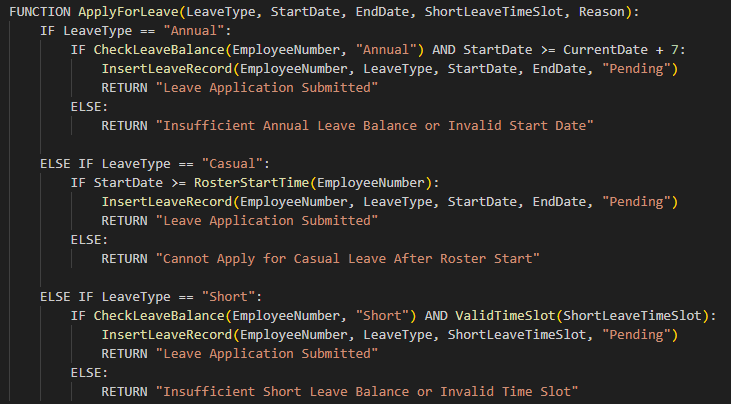


Figure 11 Pseudo Code Apply for Leaves

#### Leave Status

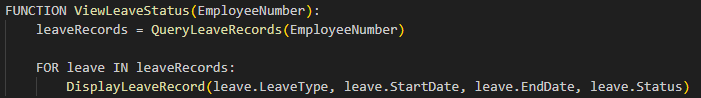


Figure 12 Pseudo Code view Leave Status

#### Delete Applied Leaves

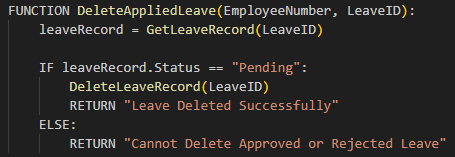


Figure 13 Pseudo Code Delete Applied Leaves

#### View Leave Balance and History

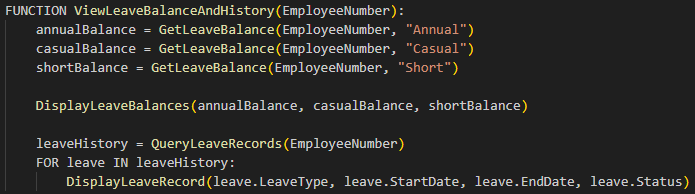


Figure 14 Pseudo Code View Leave Balance and History

### Admin Application

#### Admin Login

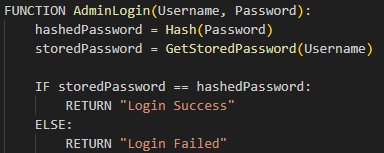


Figure 15 Pseudo Code Admin Login

#### Register New Employee

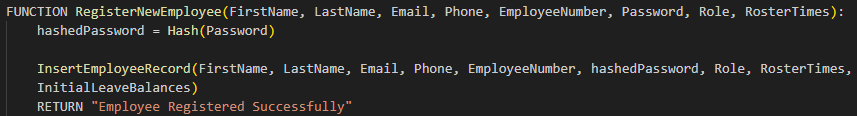


Figure 16 Pseudo Code Register New Employee

#### Define Leave and Roaster Settings

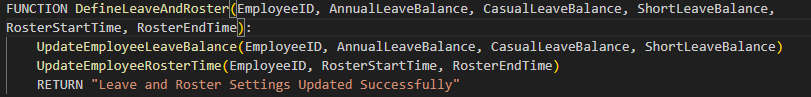


Figure 17 Pseudo Code Leaves and Roaster Settings

#### Approve or Reject Leave Requests

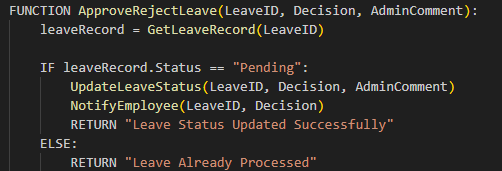


Figure 18 Pseudo Code Approve or Reject leave Requests

#### View Leave Reports

#### 

Figure 19 Pseudo Code View Leave Reports

#### 

## Database Structure

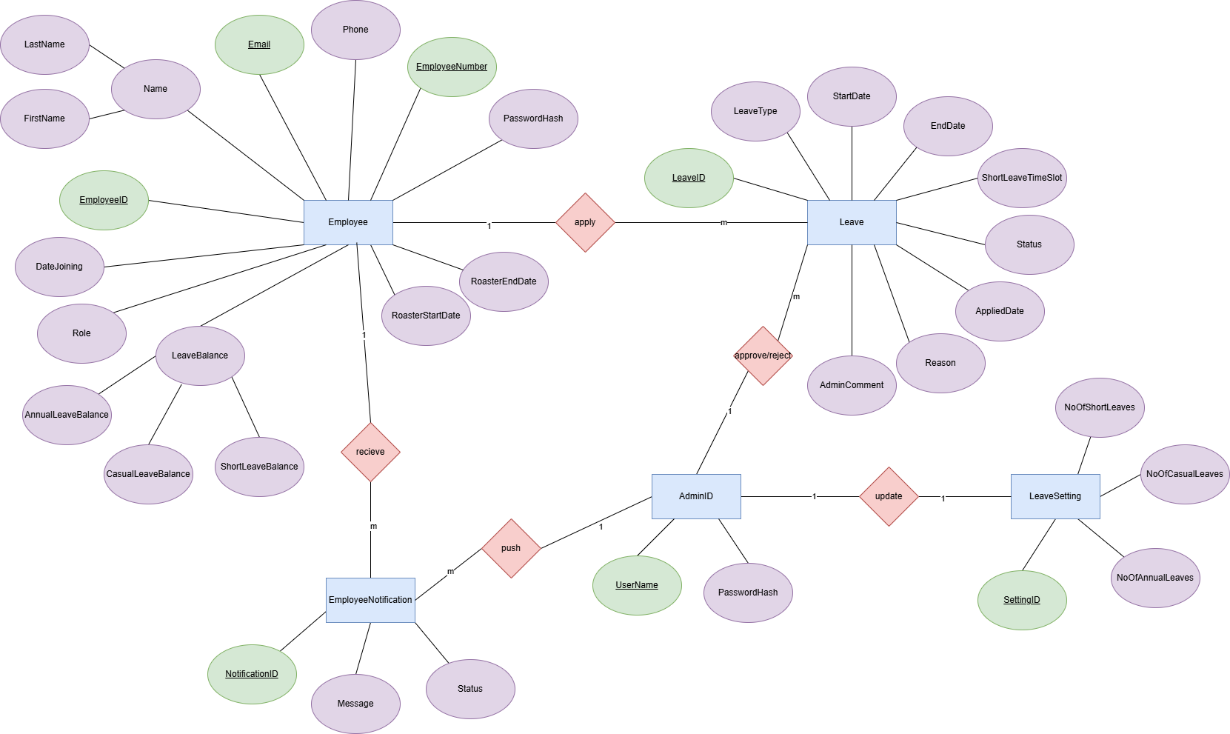


Figure 20 Database Structure ERD

## Enhanced Algorithm with Features of Visual Studio

### 1. Employee Application

#### 1.1 Employee Login

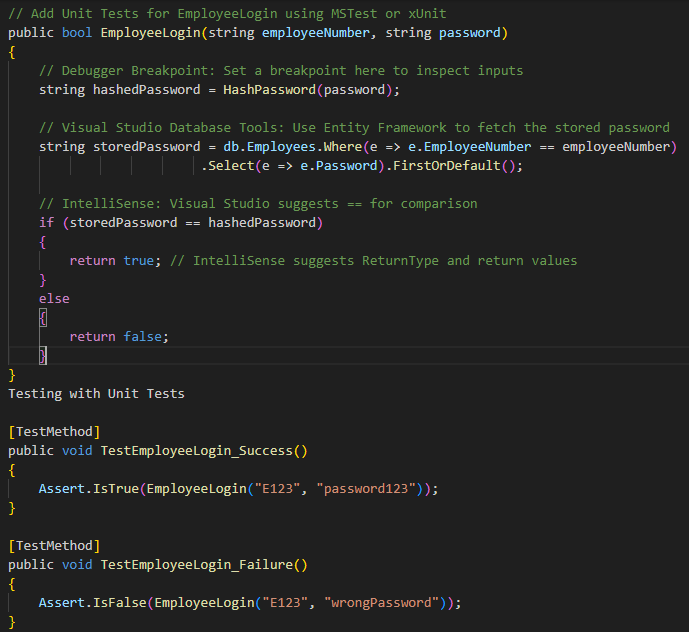


Figure 21 Enhanced Employee Login Algorithm

#### 1.2 Apply for Leave

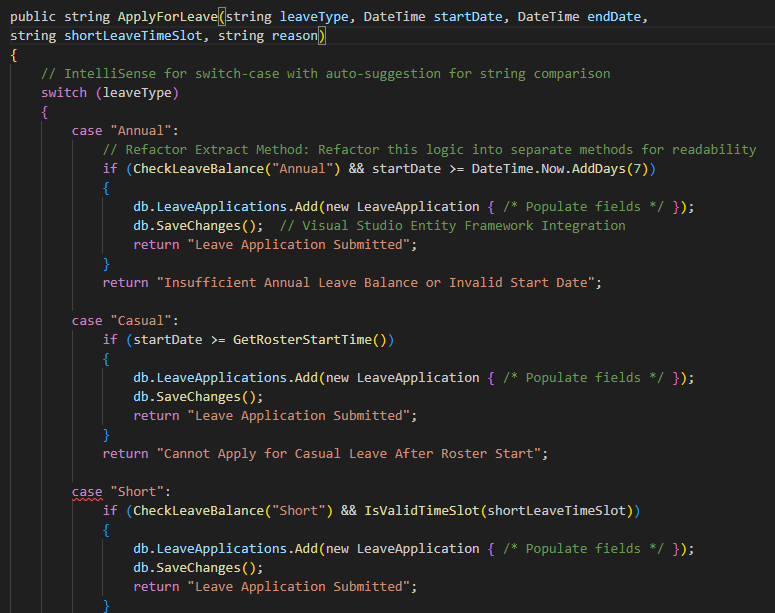


Figure 22 Enhanced Apply Leave Algorithm

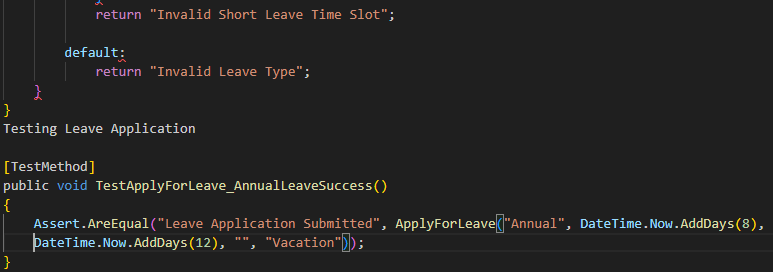


Figure 23Enhanced Apply Leave Algorithm

#### 1.3 View Leave Status

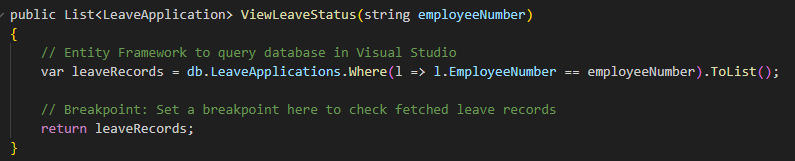


Figure 24 Enhanced Leave Status Algorithm

#### 1.4 Delete Applied Leave

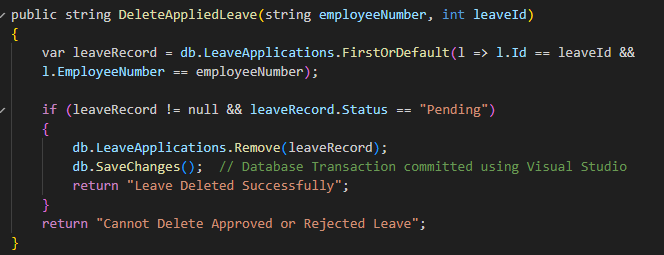


Figure 25 Enhanced Delete Applied Leave Algorithm

#### 1.5 View Leave Balance and History

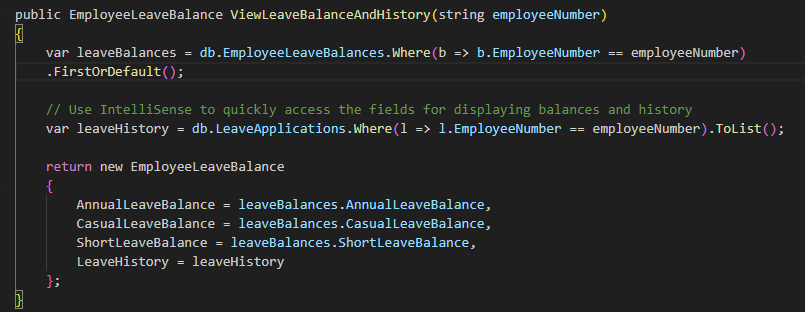


Figure 26 Enhanced View History Algorithm

### 2. Admin Application

#### 2.1 Admin Login

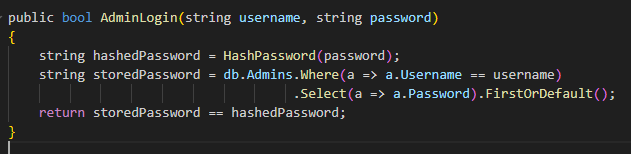


Figure 27 Enhanced Admin Login Algorithm

#### 2.2 Register New Employee

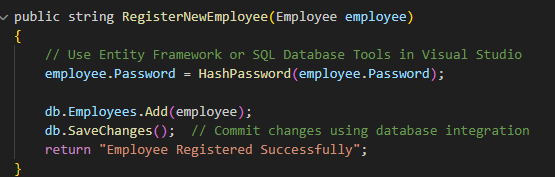


Figure 28 Enhanced Register Employee Algorithm

#### 2.3 Define Leave and Roster Settings

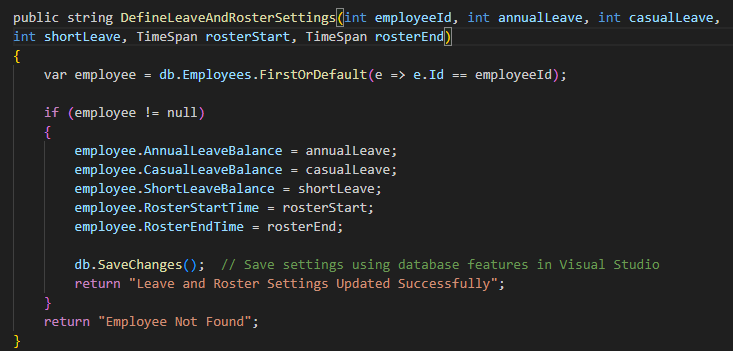


Figure 29 Enhances Setting Algorithm

#### 2.4 Approve/Reject Leave

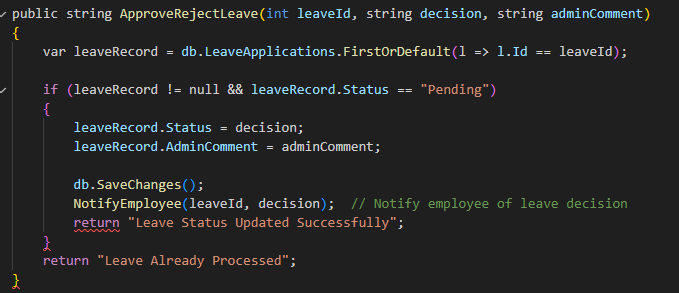


Figure 30 Enhanced Approve/Reject Leave Algorithm

#### 2.5 View Leave Reports

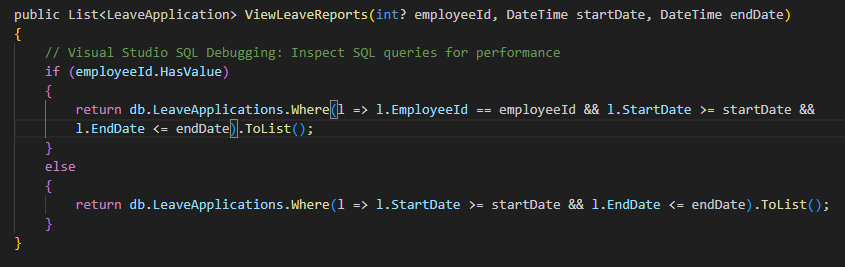


Figure 31 Enhanced Reporting Algorithm

## IDE

### What is an IDE?

An Integrated Development Environment (IDE) refers to a software program providing comprehensive tools for programmers and software developers. An IDE will typically integrate all or most of the tools a programmer needs to develop software into one integrated environment. (sourcecodester, 2023)

#### Key components of an IDE

* Source Code Editor A text editor that may offer features including syntax highlighting, code formatting, and autocompletion.
* Compiler set to compile the developer's source code or interpret it as the code is being developed.
* Debugger, which is part of the code development process since the IDE provides a tool that enables the program to be run in a step-by-step line execution with errors being identified by the debugger.
* Build Automation Tools designed to manage the compile, link, package, and test phases of the developer's code.
* Version Control Integration component (i.e. has Git integration capabilities). (sourcecodester, 2023)

Examples of popular IDEs include Microsoft Visual Studio, Eclipse IDE, IntelliJ IDEA IDE, PyCharm IDE, and Xcode IDE.

### IDE Development vs Manual Development

#### 1. IDE Development

When developers develop code by using an IDE, they have access to a variety of features that provide advantages some of them are,

* Code Assistance

IDEs typically provide features such as IntelliSense, autocompletion and real-time syntax checking to help the developers write code faster and in less time. (sourcecodester, 2023)

* Debugger Tools

The majority of IDE tools will provide an integrated debugging tool for the developer to use to set breakpoints, identify and inspect various variables, and step through the execution of code. (sourcecodester, 2023)

* Error Detection

An IDE will highlight the developer's errors immediately such as syntax errors, typos, or inadvertent inconsistent code segments to the developer in the code. (sourcecodester, 2023)

* Integrated Build & Compile

An IDE simplifies the process of compiling and running the developer's code to a click or a hot-key. (sourcecodester, 2023)

* Direct Version Control

Direct communication with a version control system (i.e. Git) for the repository of code being developed, using a direct integration right from the IDE is standard practice. (sourcecodester, 2023)

* Refactoring Tools

Automated functionality is provided by some IDEs to refactor code, highlighting where changes are in code and making it easy for the developer to change the structure of the existing software code without having to change every occurrence of the same code. (sourcecodester, 2023)

* Real-Time Collaboration

Some IDEs provide integration with plugins to provide real-time collaboration for multiple developers to work on the same code. (sourcecodester, 2023)

#### 2. Manual Development

In the process of manual development, developers intentionally refrain from working in a fully integrated environment, i.e., an IDE, in favor of each component of the development process completed individually and with their respective tools. Procedurally, developers may use a plain text editor. Such editors could include Notepad, Vim, or Sublime Text and should not constitute part of IDE with advanced IDE features such as code completion. (sourcecodester, 2023)

Developers will then have to compile the code through the terminal or command line interface by typing the compiler commands, such as javac or gcc. Developers can use external debugging tools, for example, GDB (for C/C++), or debugging through print statements, which involve the manual insertion of print statements along with tracking the values of variables. (sourcecodester, 2023)

Developers should then have to manually create and manage build scripts, which also includes making a directory structure in which to save the source files and file to store dependencies and libraries, without any automated project configuration. Developers will use Git or other version control through the command line without a graphical user interface. There is no extra syntax checking or warning until compiling rather than during your coding practice. (sourcecodester, 2023)

### IDE vs Manual

Table 1 IDE vs Manual Coding

|  |  |  |
| --- | --- | --- |
| Feature | IDE Development | Manual Development |
| Code Assistance | Autocompletion, IntelliSense, code snippets | None or minimal |
| Error Detection | Real-time error highlighting | Detected only during compilation |
| Debugging | Built-in debuggers with breakpoints, watches | External debugging tools or print logs |
| Compiling & Running | One-click build and run | Manual invocation of compiler in terminal |
| Version Control | Integrated tools for Git, SVN, etc. | Managed via command line or separate tool |
| Refactoring | Automated refactoring tools | Manual refactoring |
| Real-Time Collaboration | Integrated collaboration tools (e.g., Live Share) | Usually done via file sharing |

(sourcecodester, 2023)

# Activity 4

## Developed GUI Application

### Admin Application

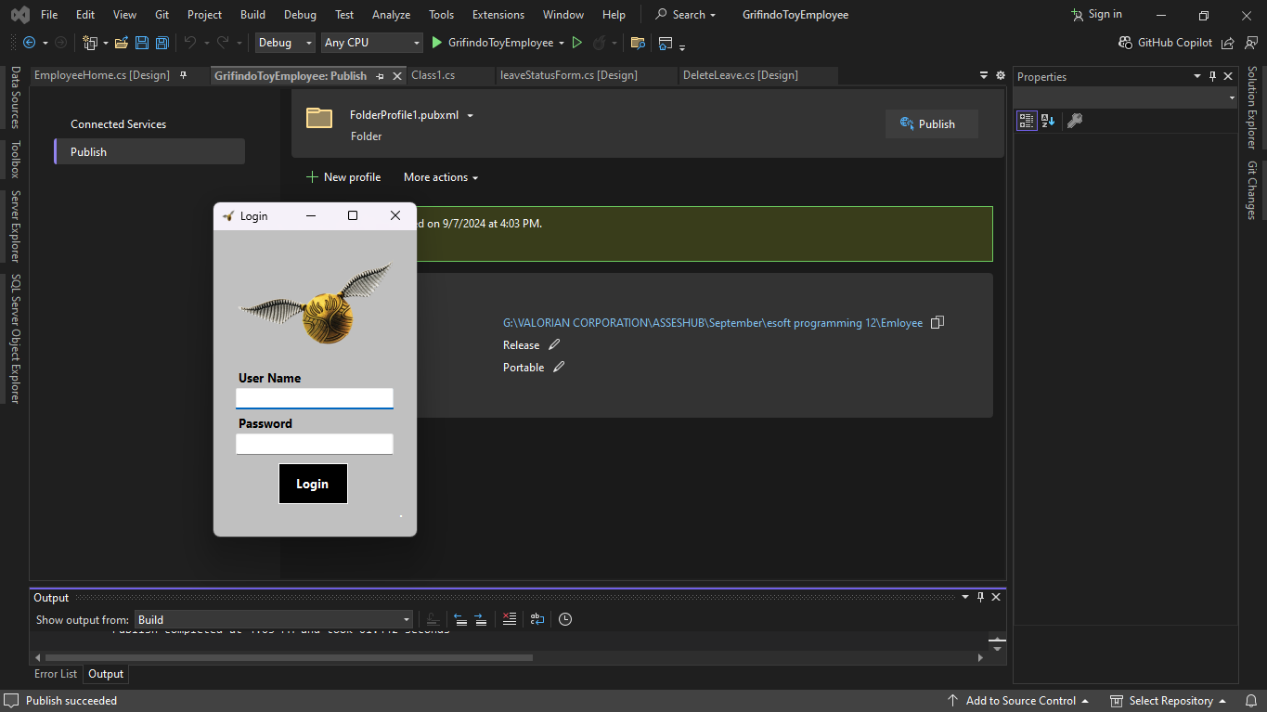


Figure 32 Admin Login

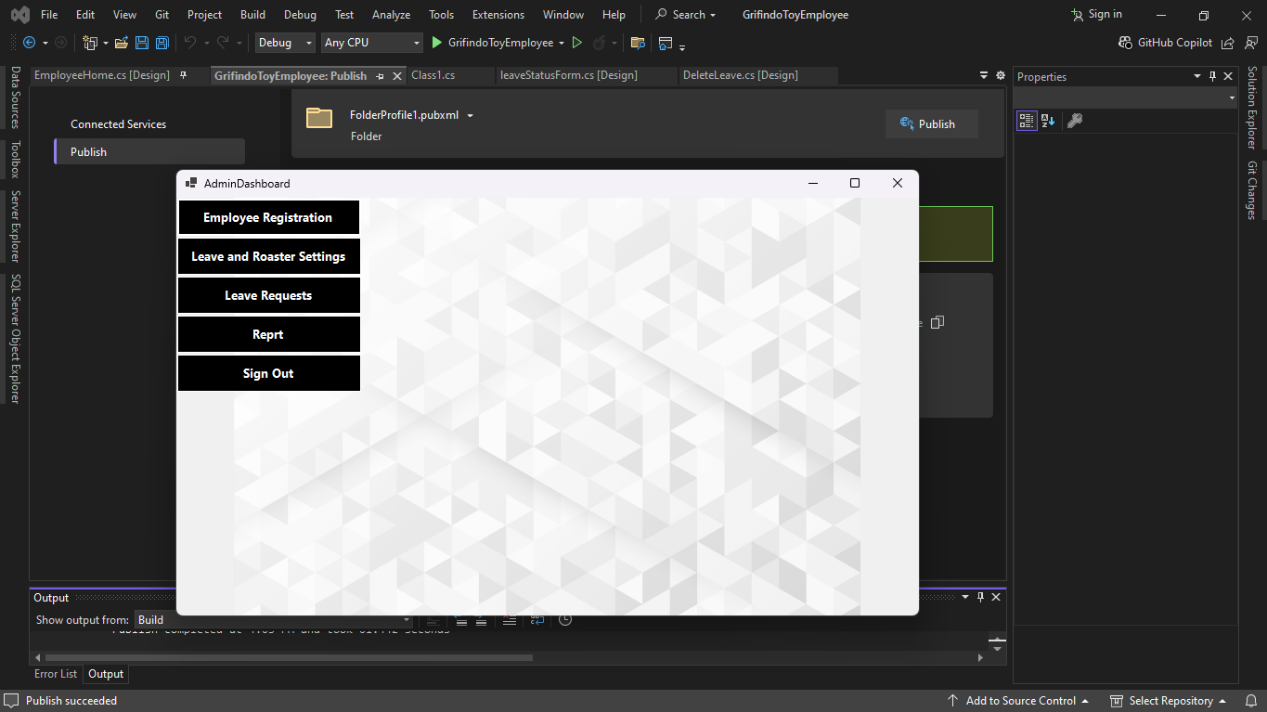


Figure 33 Admin Dashboard

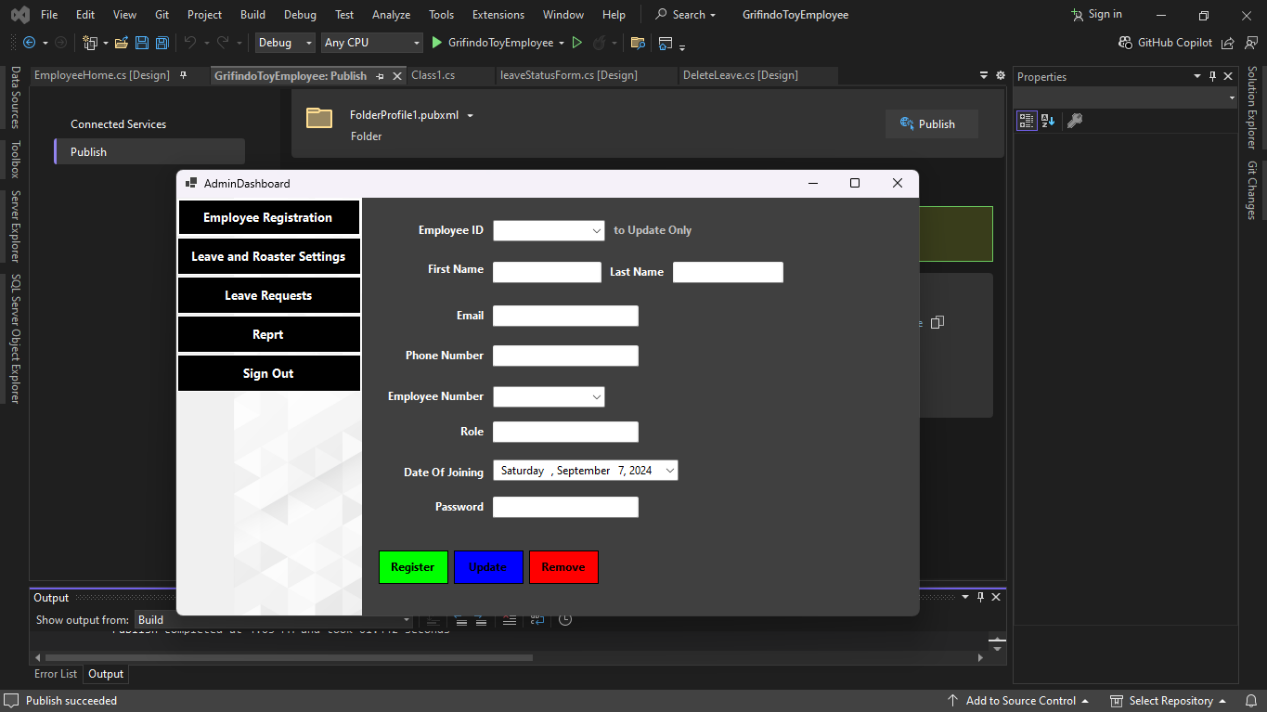


Figure 34 Employee Registration

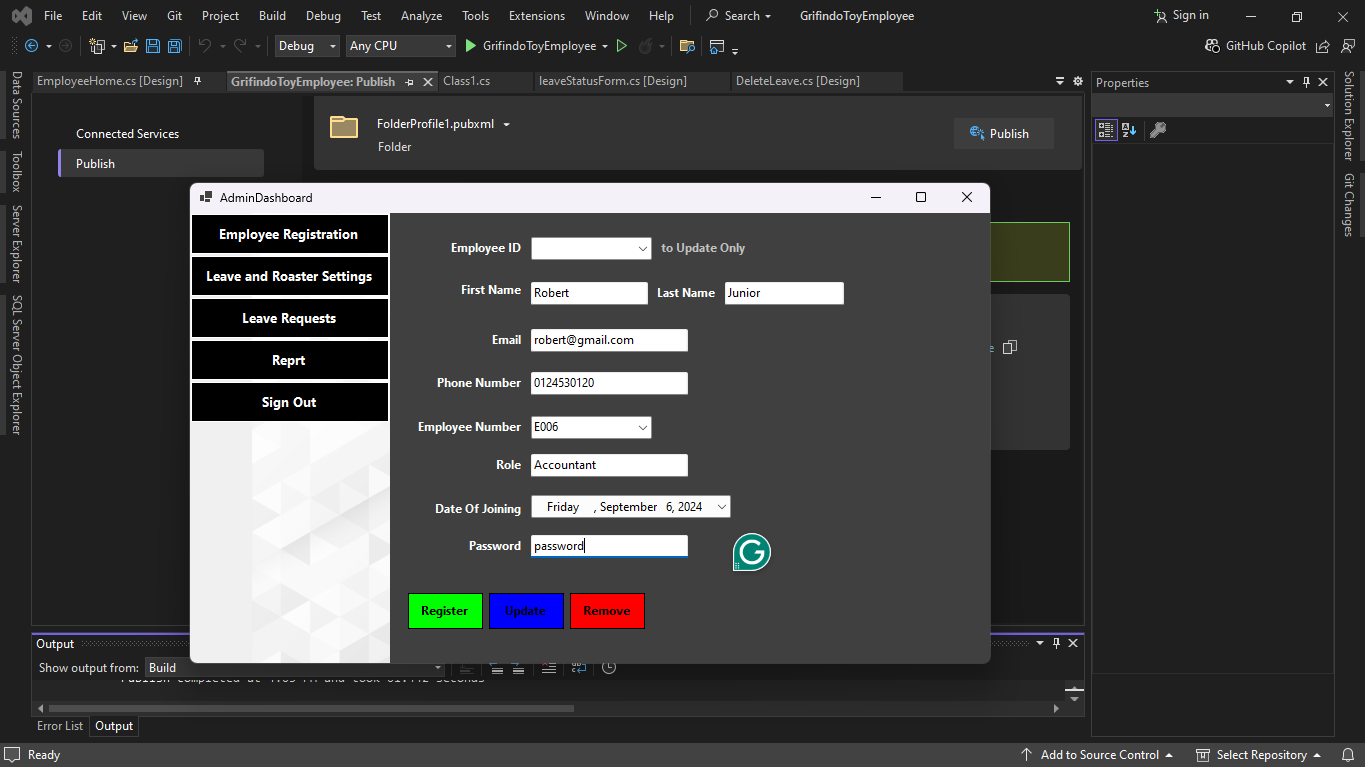


Figure 35 Registering an Employee

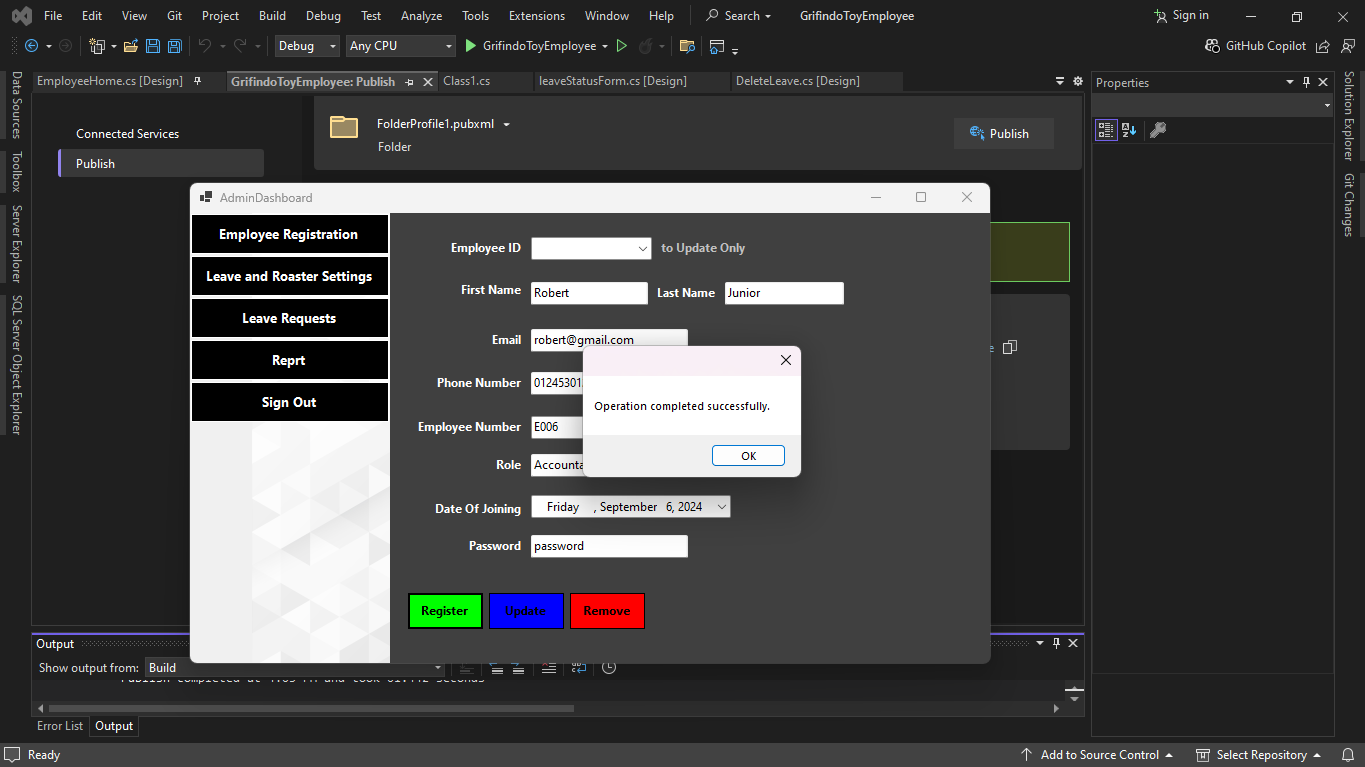


Figure 36 Registering an Employee

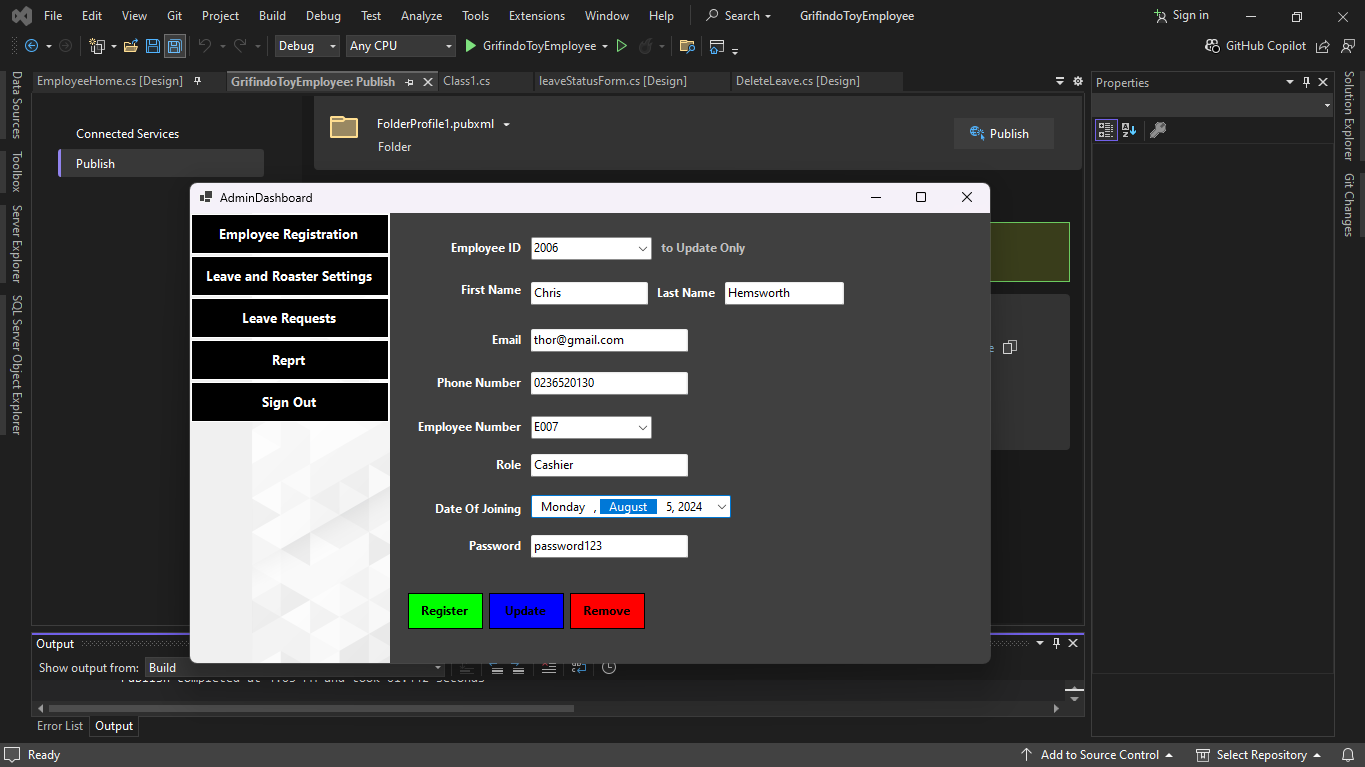


Figure 37 updating existing Employee

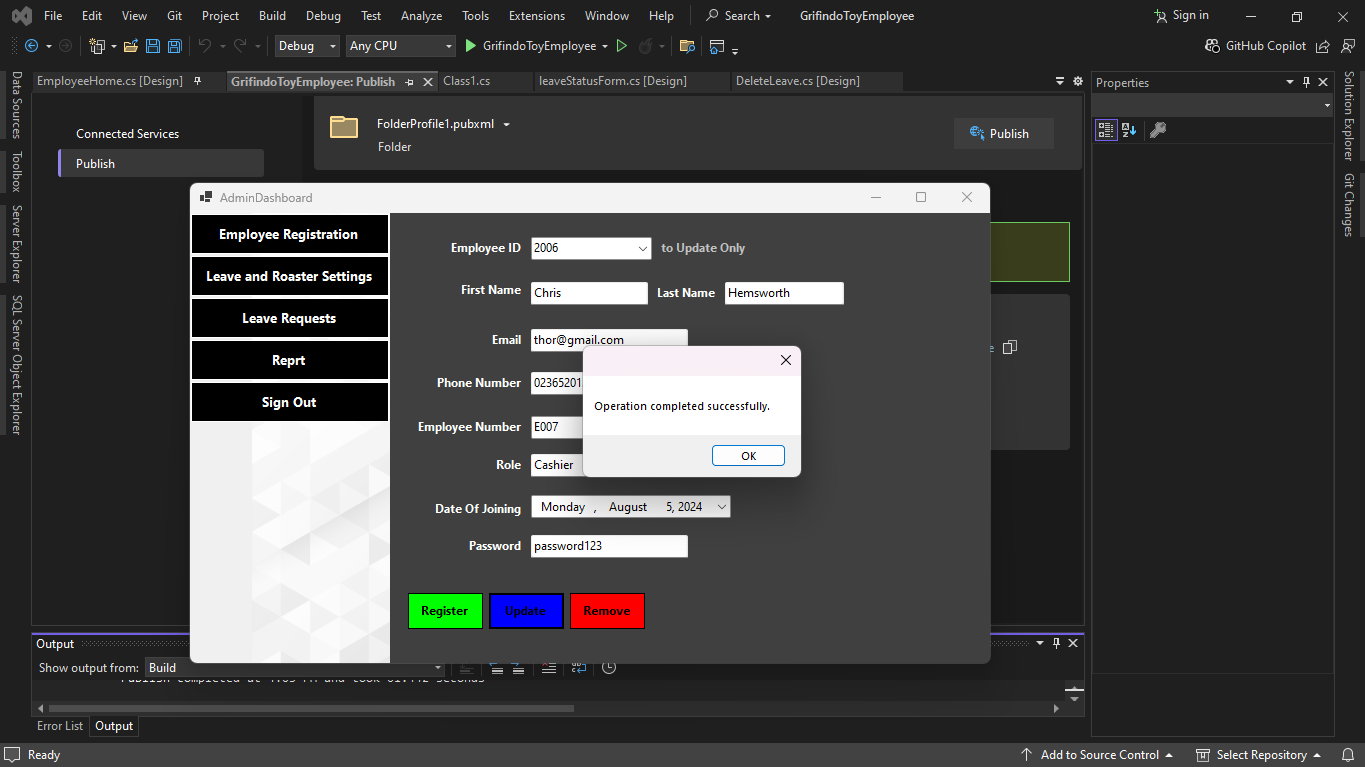


Figure 38 updating existing Employee



Figure 39 Removing Employee

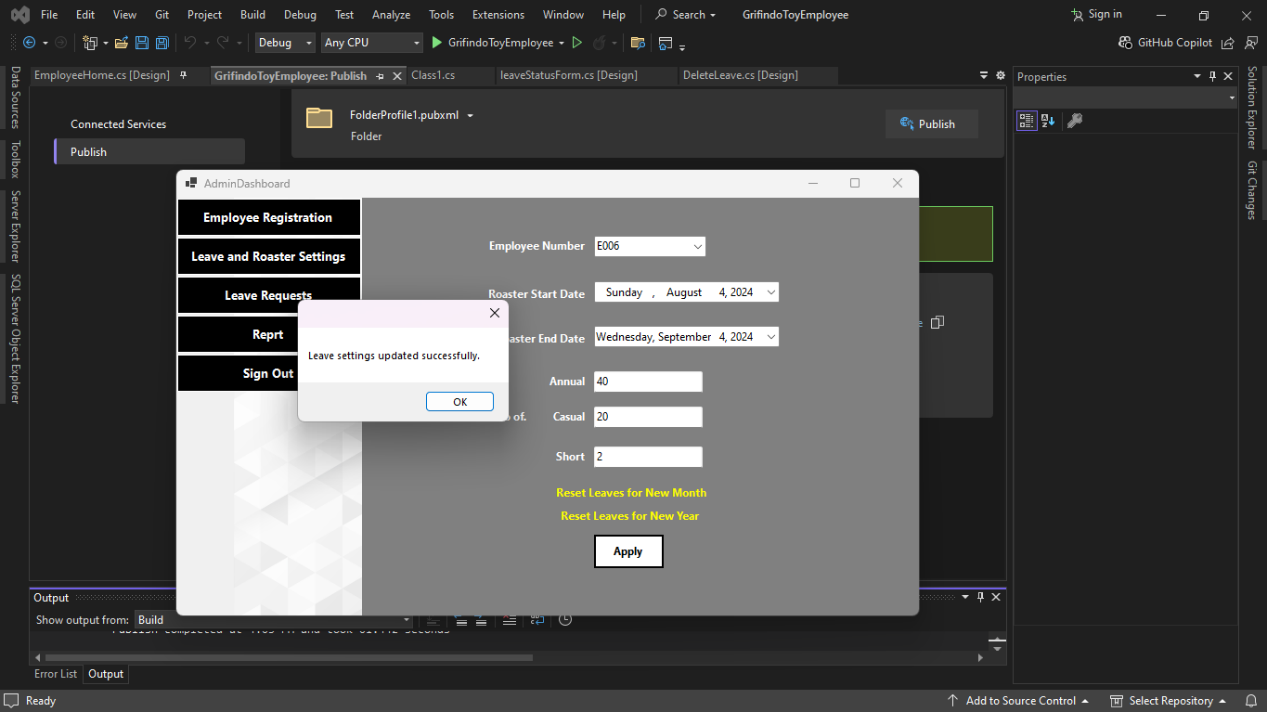


Figure 40 Leave and Roaster Settings

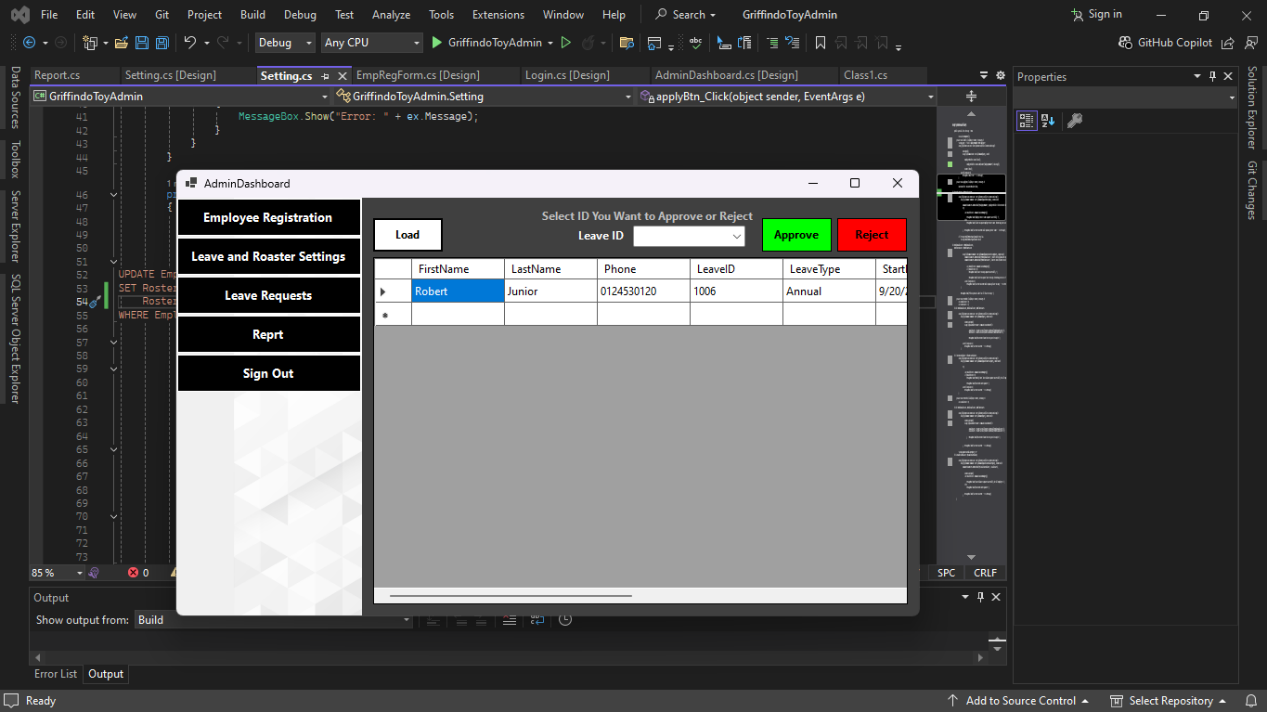


Figure 41 Leave Request

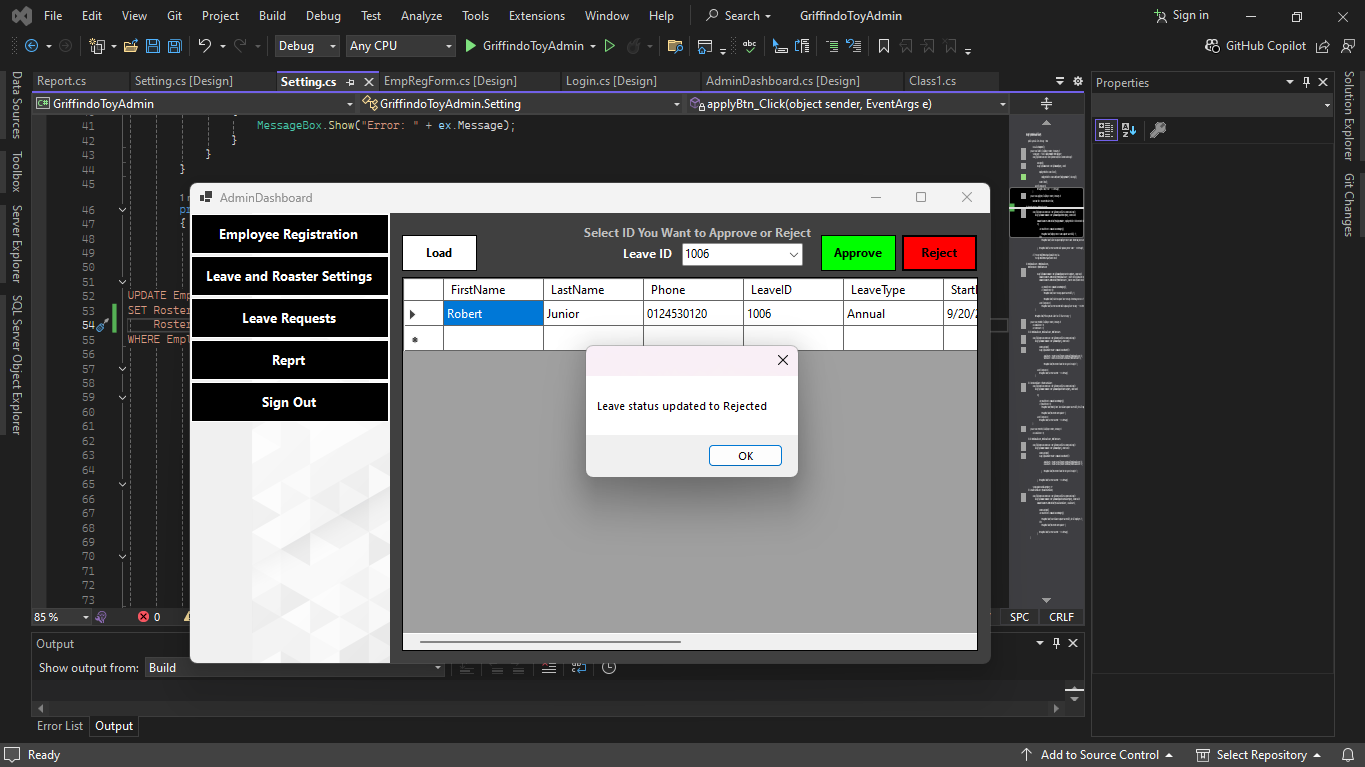


Figure 42 Approve or Reject Leave Request

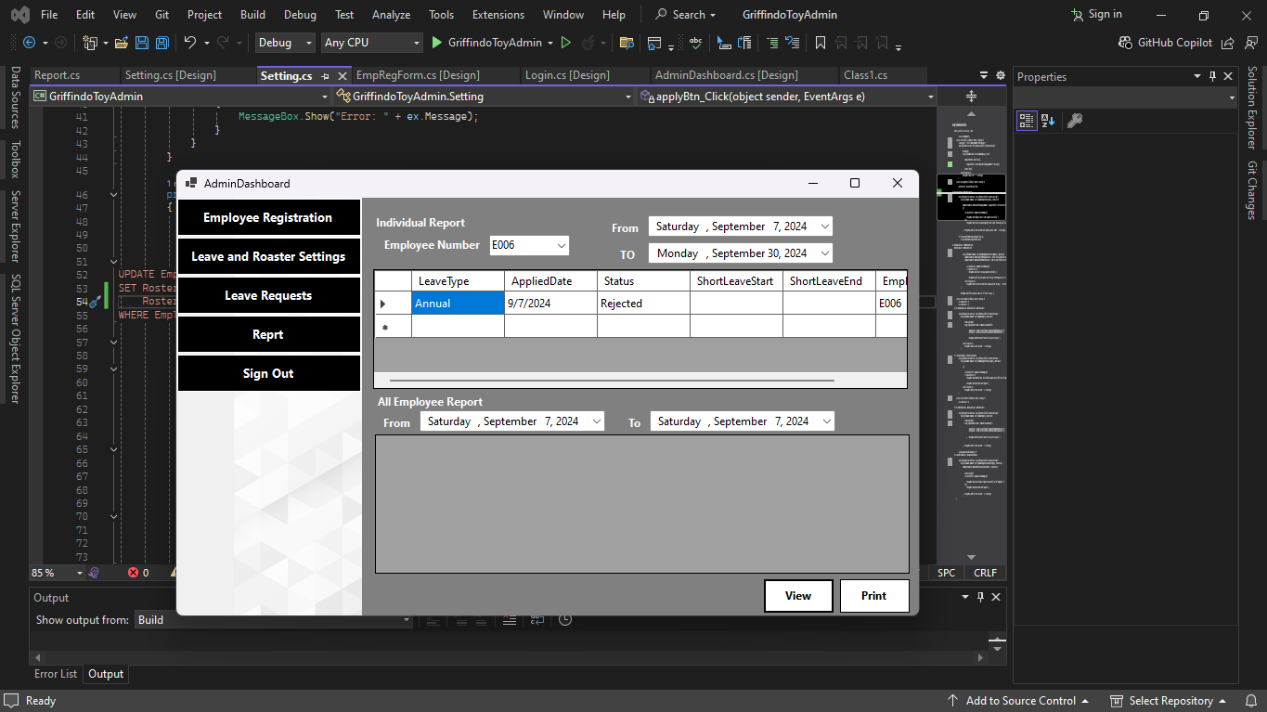


Figure 43 Individual Report

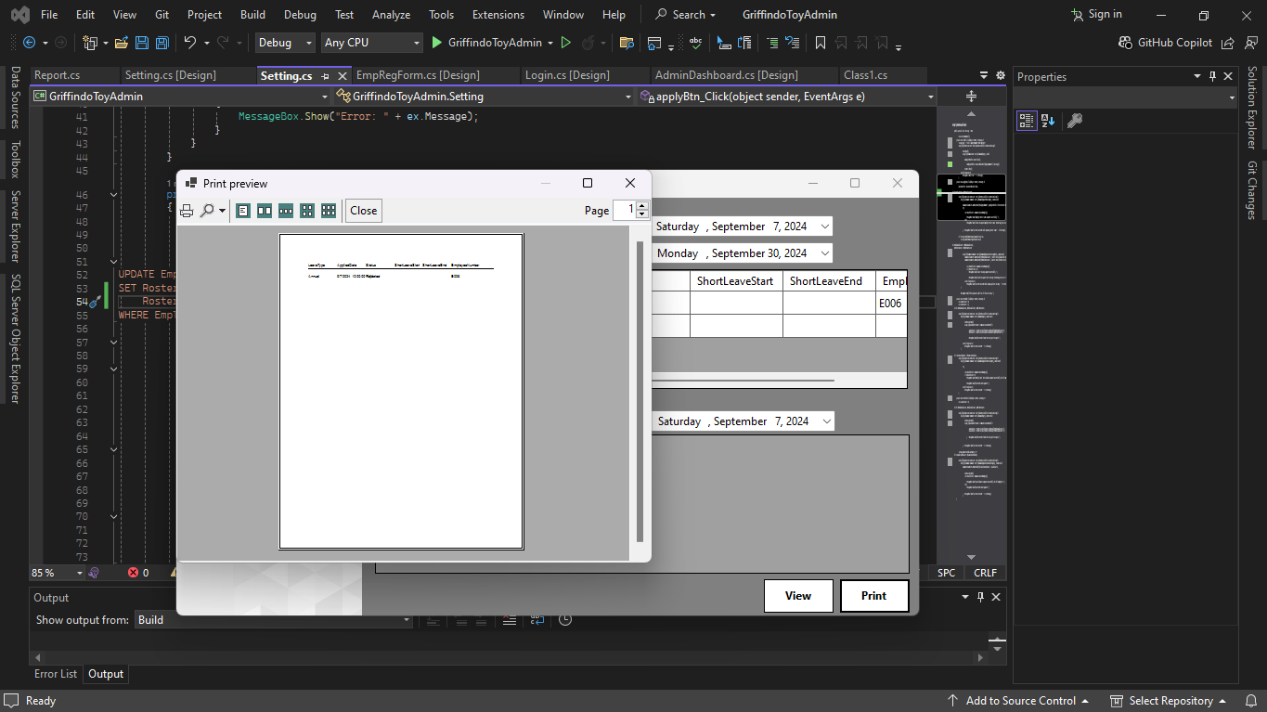


Figure 44 Printing report

### Employee Application

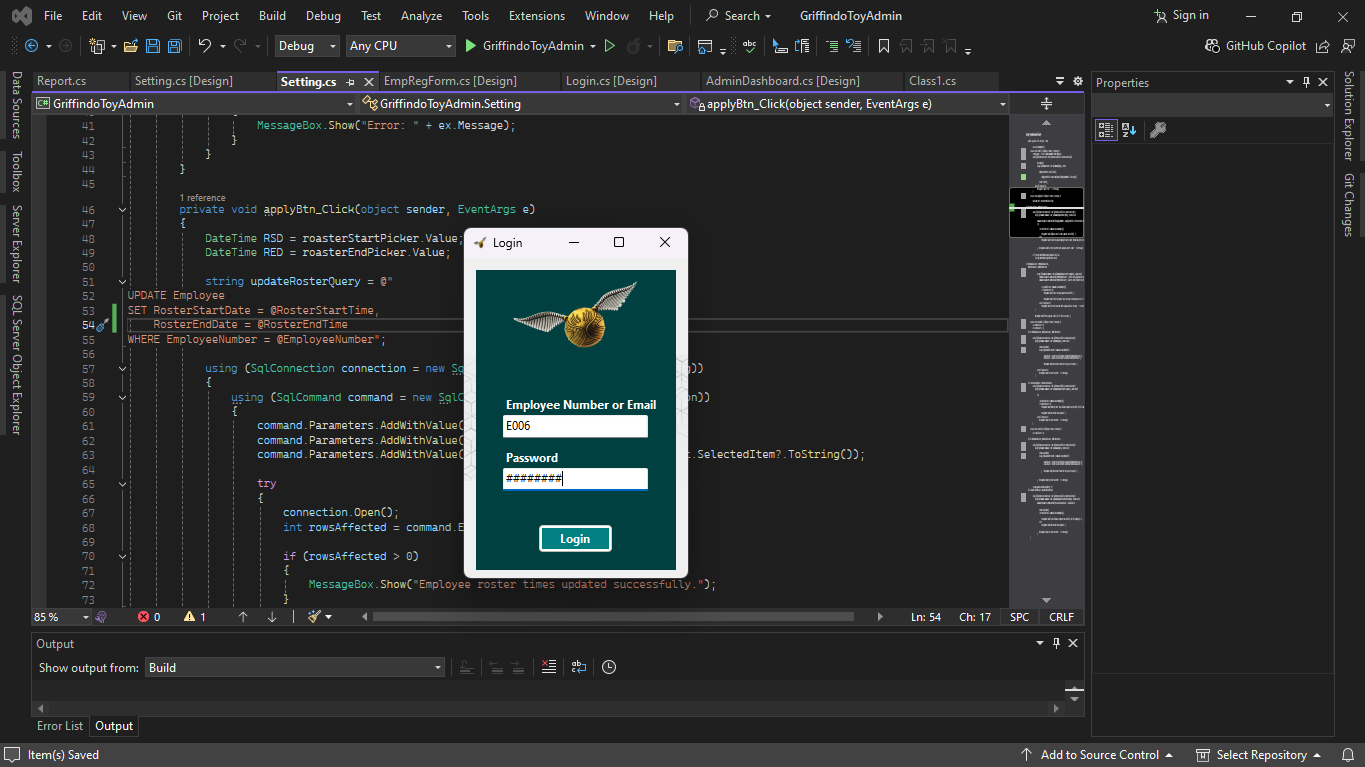


Figure 45 Employee Login

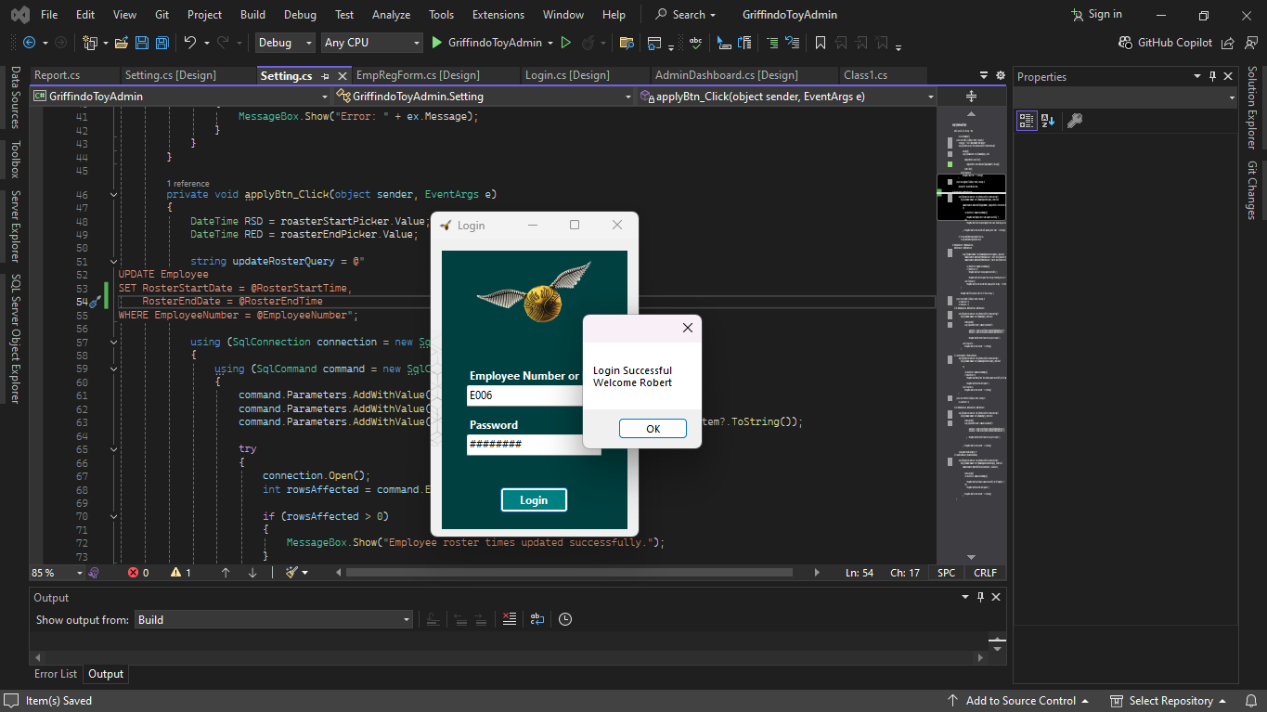


Figure 46 Employee Login

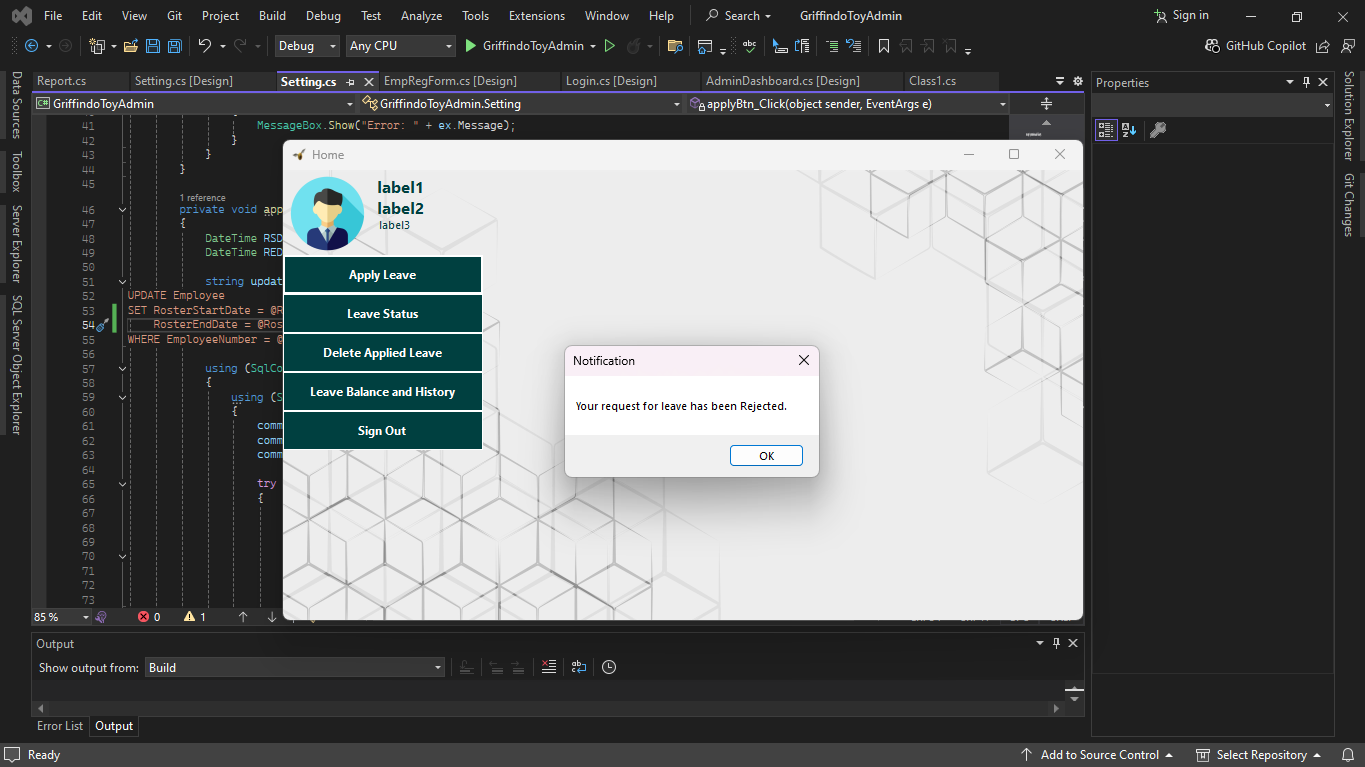


Figure 47 Receiving notification for applied Leaves

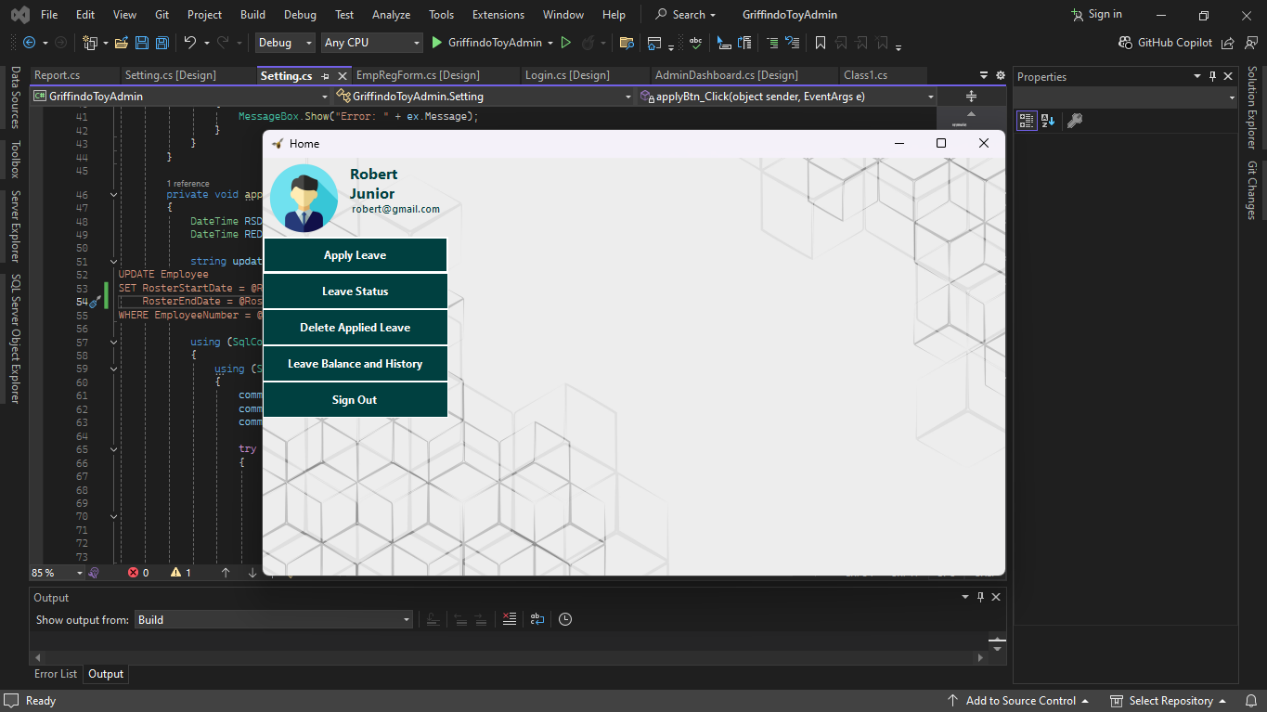


Figure 48 Insert Dashboard

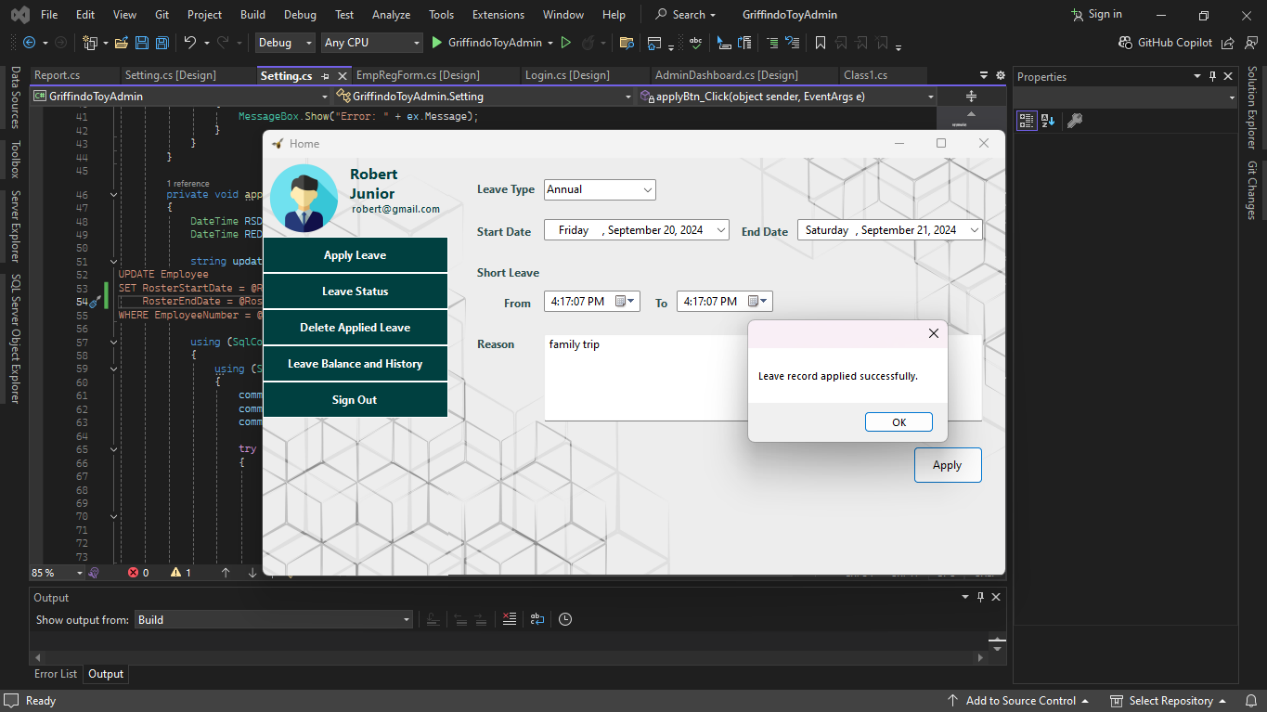


Figure 49 Apply for Leave



Figure 50 Leave Status

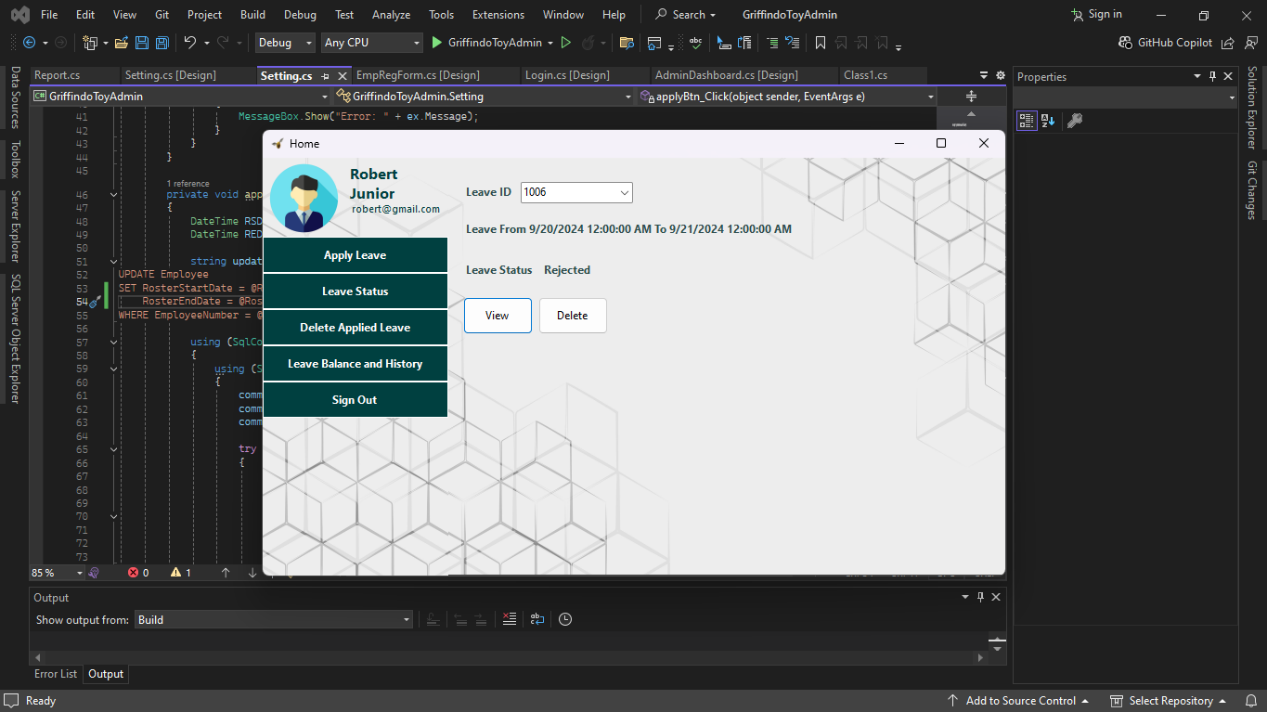


Figure 51 View before Delete

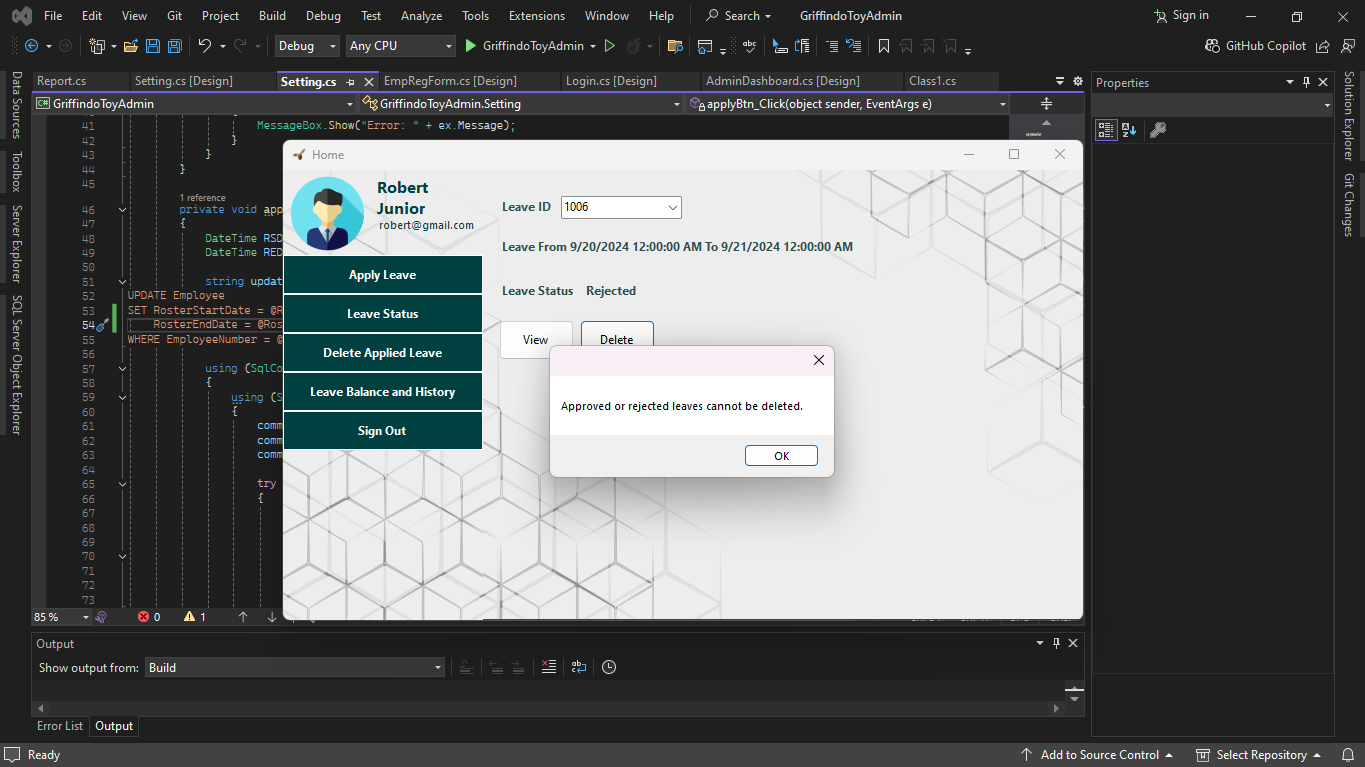


Figure 52 Deleting Applied Leaves

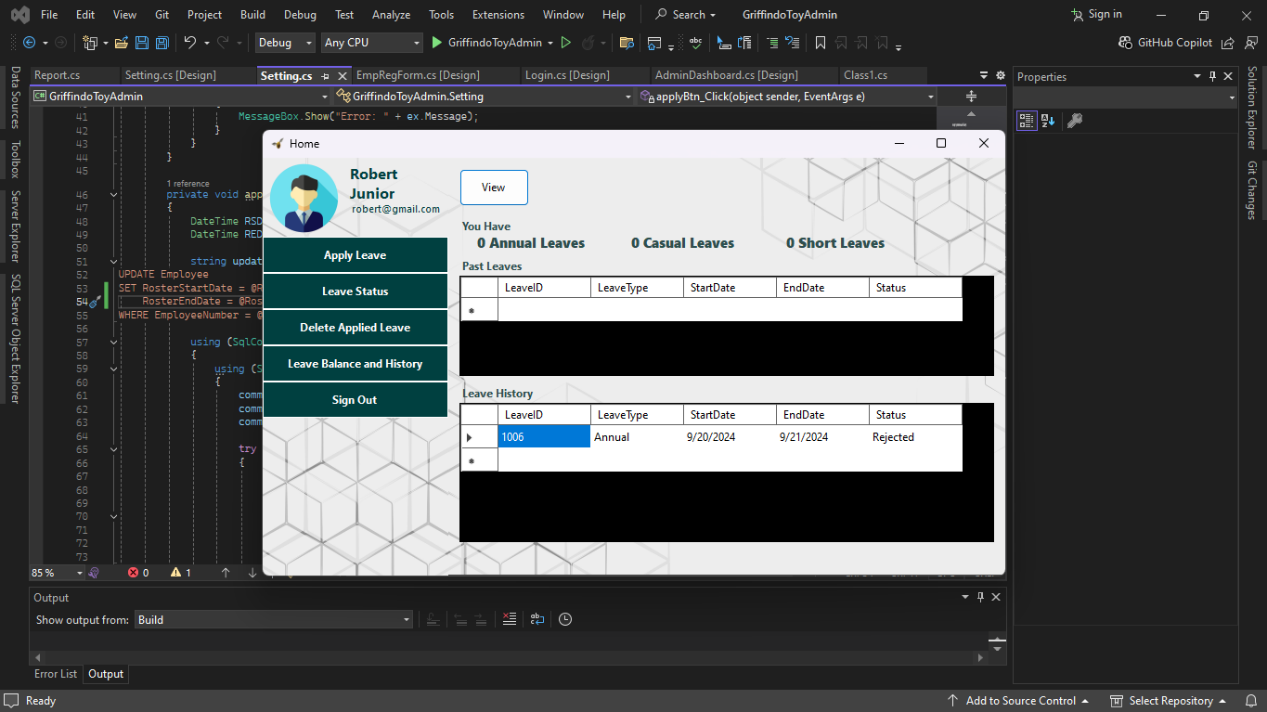


Figure 53 Leave Balance and History

## Debugging Process and Facilities in an IDE

Debugging is one of the most critical areas in software development, intended for finding bugs or errors in your code, isolating, and removing them. Major modern IDEs, like Visual Studio, have a set of tools and methods to make this process easier by boosting productivity and quality of code. Let's take a closer look at the debugging process and debugging facilities in an IDE. (Microsoft, 2023)

### 1. Preparing Code for Debugging

#### Code Analyzer

* **Red Squiggles (Errors) –** when coding in Visual Studio if there is an error it will show a red squiggle line under the point that has the error. This will help to identify an error before running or building the application. It should be a syntax error or something. (Microsoft, 2023)
* **Green Squiggles (Warnings) –** from a green squiggle Visual Studio shows a warning that is risky to the program. It's not a bug or error while running the code it will cause some errors. As an example, this is showing to null variables. (Microsoft, 2023)

#### Quick Actions (Light Bulb Icon)

* **Error Fixes –** this shown up near red squiggles to show how to fix error. It shows how can that error be fixed. (Microsoft, 2023)

### 2. Fixing Exceptions (Run-Time Errors)

#### Exception Handling

* **Exception Helper –** if a runtime error occurs exception helper shows the error. (Microsoft, 2023)

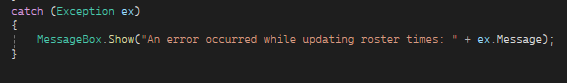


Figure 54 exception showing error

* **Try/Catch Blocks –** this is used to catch unexpected errors. (Microsoft, 2023)
* **Specific Exception Handling –** using try/catch and exception we can handle unexpected errors that can occur while running. As an example, we can handle database connection errors accurately using this technique. (Microsoft, 2023)

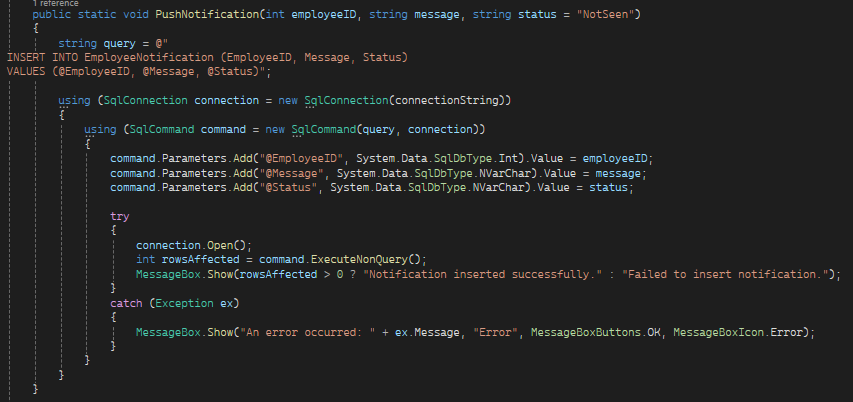


Figure 55 Exception Handling

## How Debugging Process Used to Develop Secure Robust Applications

### 1. Locating and Fixing Security Vulnerabilities

#### a. Input Validation Issues

Testing Edge Cases - The process of debugging allows for testing with unexpected or malicious inputs with the intent of finding problems in input validation, such as SQL injection or cross-site scripting (XSS).

Example - One could utilize a debugger to see how the inputs of SQL commands or scripts are processed. Variables should be sanitized so that any potential attacks are blocked.

#### b. Authentication and Authorization Analysis

The following can be debugged - authentication flows. It ensures debugging handles credentials securely-for example, passwords hashing and storage of tokens.

Authorization checks - It ascertains that users cannot access resources they are not supposed to have. Debugging handles the authorization logic.

#### c. Exception Handling

Handling sensitive information - This ensures that errors don't reveal sensitive information; debugging ensures leakages in error messages.

Minimizing security risks - Ensure exceptions are well caught and handled in order to avoid revealing vulnerabilities.

### 2. Increasing Robustness with Thorough Testing

#### a. Debugging during Stress and Load

Simulate High Load - Employ debugging to understand how the application will behave under stress, hence finding performance bottlenecks.

Allow for Concurrency - Make sure race conditions and deadlocks are avoided.

#### b. Ensuring Data Integrity

Inspection of Data Manipulation - With debugging, trace manipulations with the data to ensure consistency of data, hence preventing corruption or loss.

Validating State Transitions - Ensuring that state transitions always produce valid application states, either during normal execution or during debugging. c. Resource Management Memory Leak - Debugging is meant to locate and destroy memory leaks whenever possible by freeing them to be frugal with resources. File and Network Resources - Debugging should ensure that file handles and network connections among other resources are appropriately managed and closed where necessary. 3. Ensuring improvement in code maintainability and quality a. Ensuring Correctness of Logic and Flow

Logical Flows - understanding how a code behaves - debugging has a role in validating logical flows so that the code behaves the way it is supposed to. Review Changes - The work of debugging ensures changes are reviewed for not creating new bugs. b. Employing Assertions for Intent Verification Documenting Assumptions - Assertions document assumptions in the code. It is important that debugging ensure these assumptions hold. Fail Fast - Assertions stop the code from continuing with an error early in development; thus, fixing becomes easier. c. Reducing Error Handling and Logging

Effective Logging - Debugging ensures that the logs capture useful diagnostic information.

Error Recovery - The tests for error recovery mechanisms ensure the application handles failures gracefully.

### 4. Use of Advanced Debugging Tools

#### a. Static and Dynamic Analysis Tools

Static Code Analysis - Early detection of vulnerabilities or code issues is achieved by tools before the application executes.

Dynamic Analysis - This is about using tools to observe an application at runtime and highlight issues, such as memory leaks or security risks.

#### b. Profiling and Performance Monitoring

Performance Profiling - Use tools to identify performance bottlenecks, including inefficient algorithms. Security Profiling - Examine how an application interacts with its environment to identify security risks.

## Coding Standards Used for Griffindo Toy Employee Leave Management System

Griffindo Toy Employee Leave Management System - I followed a number of coding standards that guarantee security, performance, readability, and maintainability to be reflected throughout the application.

### 1. Naming Schemes

Method Names - Methods had to start in PascalCase. For example, methods like GetEmployeeNumber follow this convention, usually used in C# because it improves readability and makes names look consistent.

Variable Names - I have used camelCase for variables like employeeID, employeeNumber, and result. That way, the variable names are descriptive and clear about the purpose they serve in an application. (Pal, 2024)

### 2. Parameterized Queries

Preventing SQL Injection - To make the code secure against SQL injection attacks, I have used parameterized queries in interactions with databases. For instance, the query string uses parameters like @eid to insert data safely into SQL statements.

Using AddWithValue - I have used the cmd.Parameters.AddWithValue method for adding parameters to the SQL commands. This securely passes the values onto the database query, ensuring that data is handled in a secure manner. (Pal, 2024)

### 3. Exception Handling

Try-Catch Block - Try-catch blocks have been used to handle exceptions that may pop up during the database operations. This ensures the application acts appropriately when unthought-of errors pop up without hanging.

Detailed Error Messages - In the catch block, I captured exceptions, leveraging MessageBox.Show for error messages in order to diagnose the problems-take meaningful feedback from users or developers in case something goes wrong. (Pal, 2024)

### 4. Resource Management

Using Statement - In my code, I used the using statement for proper database connection and command disposal. Examples are shown below using SqlConnection and SqlCommand; the using statement automatically disposes resources after use to avoid memory leaks and to provide assurance that database connections are properly closed.

Automatic Disposal - The using statement will automatically take care of the Dispose for these objects, like SqlConnection and SqlCommand. Therefore, I am not explicitly calling Dispose. It enhances code cleanliness and reduces the likelihood of memory leak-related issues. (Pal, 2024)

### 5. Code Readability

Comments - Although I may not have provided inline comments everywhere, I have commented on each block of code which was complicated or involved a critical operation.

Self-Descriptive Variable Names - I have used clear, descriptive variable names throughout the app. For example, with variables like leaveType, leaveBalance, and leaveStatus, one can tell in which way they are going to be put to use by just looking at their names and hence improve overall code readability. (Pal, 2024)

### 6. SQL Execution

ExecuteScalar Method - I had used this when, for instance, one is expecting a single value returned from the database such as querying EmployeeNumber on login operations. This is efficient and appropriate for operations that deal with a single result query. (Pal, 2024)

### 7. Handling Null and DBNull

Checked for DBNull.Value or null - Before performing any string or other type of conversion on database results, I check first for DBNull.Value and null. This ensures that the application handles situations where no records found or returns 'no matching records' with grace and does not result in runtime exceptions/errors. (Pal, 2024)

### 8. Connection String Management

Class-Level Connection String - I'm setting the connection string via a class-level property, namely Class1.connectionString. This enables the SoC, since database connection details are in one place and easier to change if connection settings update throughout the application. (Pal, 2024)

## Role and Purpose of Coding Standards

### Purpose of Coding Standards

Table 2 Purpose of using coding standards

|  |  |  |
| --- | --- | --- |
| Category | Aspect | Details |
| Consistency and Readability | Uniformity | Coding standards ensure consistent style across the codebase (naming conventions, indentation, comments), improving readability and maintainability. |
|  | Readability | Well-defined standards make code more readable, helping developers understand code more easily for debugging, review, and maintenance. |
| Maintainability | Ease of Updates | Code adhering to standards is easier to update and refactor, reducing the chances of introducing errors when making changes. |
|  | Reduced Technical Debt | Standards encourage clean, well-structured code, minimizing technical debt and future issues. |
| Collaboration and Team Efficiency | Smooth Integration | Coding standards allow seamless integration of code from multiple developers, reducing errors and making merges easier. |
|  | Code Reviews | Standards streamline code reviews, allowing reviewers to focus on logic and functionality instead of style or formatting issues. |
| Quality Assurance | Error Prevention | Standards include guidelines for error handling and input validation, helping write more robust code and reducing bugs and vulnerabilities. |
|  | Best Practices | Standards promote the use of best practices and design patterns, leading to higher-quality, maintainable code. |
| Onboarding and Knowledge Transfer | Faster Onboarding | New team members can quickly understand and contribute to the codebase by following established standards, reducing training time. |
|  | Knowledge Transfer | Standards facilitate better understanding and collaboration between team members, ensuring code can be maintained by different people. |
| Necessity for Teams | Team Cohesion | Shared coding standards foster team alignment, reducing misunderstandings and ensuring consistent efforts. |
|  | Reduced Conflict | Clear guidelines minimize disagreements over style and structure, fostering a more harmonious team environment. |
|  | Improved Code Quality | Enforcing standards helps teams produce higher-quality, more maintainable code. |
|  | Increased Productivity | Consistent coding practices reduce issues during development and integration, improving overall team productivity. |
| Necessity for Individuals | Personal Efficiency | Following standards helps individuals develop a structured approach to coding, improving their efficiency and reducing time spent on common issues. |
|  | Fewer Distractions | Adhering to standards allows individuals to avoid common coding errors, enabling them to focus on problem-solving and feature implementation. |
|  | Skill Development | Learning and adhering to coding standards helps individuals develop good habits, improve their coding skills, and follow best practices and design patterns. |
|  | Career Advancement | Proficiency in coding standards is valuable for career growth, demonstrating professionalism and attention to detail. |

(Codacy, 2023)

# References

Codacy, 2023. *Coding Standards: What Are They and Why Are They Important?* [Online] Available at: <https://blog.codacy.com/coding-standards> [Accessed 7 September 2024].

Falconer, J., 2022. *How to Write Good Code: 10 Beginner-friendly Techniques for Instant Results*. [Online] Available at: <https://www.sitepoint.com/how-to-write-good-code/> [Accessed 6 September 2024].

Garg, N., 2023. *How to develop an Algorithm from Scratch | Develop Algorithmic Thinking*. [Online] Available at: <https://www.geeksforgeeks.org/how-to-develop-an-algorithm-from-scratch-develop-algorithmic-thinking/> [Accessed 6 September 2024].

GeeksForGeeks, 2024. *Types of Software Testing*. [Online] Available at: <https://www.geeksforgeeks.org/types-software-testing/> [Accessed 6 September 2024].

Microsoft, 2023. *Debugging techniques and tools to help you write better code*. [Online] Available at: <https://learn.microsoft.com/en-us/visualstudio/debugger/write-better-code-with-visual-studio?view=vs-2022> [Accessed 6 September 2024].

Microsoft, n.d. *What can you do with Visual Studio?* [Online] Available at: <https://visualstudio.microsoft.com/vs/features/>.

Moo ICT, n.d. *What is Procedural, Event Driven and Object Orientated Programming*. [Online] Available at: <https://www.mooict.com/what-are-procedural-event-driven-and-object-orientated-programming/> [Accessed 6 September 2024].

Pal, S.K., 2024. *Coding Standards and Guidelines*. [Online] Available at: <https://www.geeksforgeeks.org/coding-standards-and-guidelines/> [Accessed 7 September 2024].

Pankaj, p., 2022. *Differences between Procedural and Object Oriented Programming*. [Online] Available at: <https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/> [Accessed 6 September 2024].

sourcecodester, 2023. *IDE vs Code Editor: A Comprehensive Guide*. [Online] Available at: <https://www.sourcecodester.com/blog/16921/ide-vs-code-editor-comprehensive-guide.html> [Accessed 7 September 2024].