# C# Knowledge

Project 1: Industrial Barcode Verification Software

This project focused on developing a high-performance barcode scanning and verification system tailored for industrial automation. Designed for use in manufacturing environments, the system interfaced with industrial-grade cameras and printers to ensure accurate barcode validation in real-time production lines.

As the .NET Developer, I:

Developed the application using C# with both WinForms and WPF, ensuring a robust and responsive UI.

Integrated with various industrial camera SDKs, including Cognex and Keyence, to capture and process barcode images.

Architected a flexible system supporting two operational modes: ‘printing + verification’ and ‘verification-only’, enabling compatibility with diverse third-party printers.

Optimized for real-time image processing and low-latency decision-making, ensuring high performance in fast-paced industrial environments.

Contributed to ongoing feature enhancements and worked closely with technical support teams to troubleshoot issues and deliver prompt client assistance.

Technologies used:

C#, WinForms, WPF, Cognex SDK, Keyence SDK, Multithreading, Real-time Processing

Project 2: Backend & Web Dashboard – API & Monitoring Interface

This project aimed to provide a centralized backend system and web interface for managing barcode scan data, verification results, and print operations across multiple devices in an industrial setting.

As the Full-Stack Developer, I:

Designed and built RESTful APIs using ASP.NET Core to handle the ingestion and management of barcode scans, print jobs, and verification results.

Used Entity Framework Core for seamless interaction with a SQL Server database, ensuring efficient and reliable data persistence.

Implemented real-time logging and error tracking, allowing the system to record and display print/verification statistics and operational errors.

Developed a ReactJS-based web dashboard that provided live monitoring of barcode validations, device statuses, and detailed logs with filtering capabilities.

Leveraged Axios for API communication and custom CSS for a clean, intuitive UI that offered real-time insights into barcode scanning operations.

Technologies used:

ASP.NET Core, RESTful API, Entity Framework Core, SQL Server, ReactJS, Axios, CSS, SignalR (if used for real-time)

# C# Knowledge

Sự khác biệt giữa System.Boolean và bool ?

Dùng bool ( tên thân mật, đơn giản ngắn gọn) 99 % dùng này

Dùng System.Boolean ( tên đầy đủ, kiểm tra kiểu dữ liệu )

Sự khác biệt giữa dynamic và var ?

Var: trình biên dịch sẽ đoán kiểu dữ liệu khi khai báo, var test = "string" (kiểu string)

Dynamic: trình biên dịch sẽ không đoán, nhưng khi chạy sẽ báo lỗi nếu ko đúng kiểu dynamic test = "string" - chạy test = 10 lỗi. nhưng code ko hiện lỗi còn var thì ngược lại. Dùng khi cần tương tác API dạng data json, khi dữ liệu nhận là động hoặc với các ngôn ngữ dynamic như python hoặc Ruby.

Compile-time error: lỗi biên dịch ( khi code, dịch sang mã máy)

Run-time error: khi chạy ( lỗi ngoại lệ)

Reference Type: Kiểu tham chiếu

(Class, Interface, Array, Delegate) Class, string, object, List, IEnumerable, interface, int[], string[], Action, Func, custom delegate, Queue, Dic.

Value Type: Kiểu dữ liệu

(Numberic, Logic, struct, Enum) int, double, byte, bool, char, Datatime, Guid, custom struct, enum.

Sự khác biệt giữa ToString và $'' ?

Dấu $ sẽ ngắn gọn hơn và nếu kết hợp với {} mình có thể bỏ nhiều giá trị vào 1 chuỗi.

Sự khác biệt giữa string và StringBuilder ?

string: bất biến, không thể thay thế được, khi nối chuỗi, thay thế ký tự, cắt chuỗi con thuộc chất .Net sẽ tạo ra một đối tượng string Mới trong bộ nhớ lưu nội dung đã thay đổi.

StringBuilder: cấp phát một vùng nhớ đệm (buffer) và cho phép bạn thay đổi nội dung của chuỗi ngay tại chỗ( hiệu xuất tốt hơn, tiết kiệm bộ nhớ hơn)

int (kiểu giá trị không thể null - Non-Nullable Value Type) Runtime error khi giá trị null

int? (kiểu giá trị có thể null - Nullable Value Type)

Rỗng(Empty) có nghĩa là có một đối tượng nhưng không có giá trị bên trong (nồi cơm không có cơm)

Null: có nghĩa nó không trỏ vào đâu ko đối tượng ( không có cả nồi cơm, không tồn tại)

C# Knowledge 1

Hàm vô danh là hàm không có tên và được sử dụng tại chỗ

List<int> evenNumbers = numbers.FindAll(delegate(int num) // Ham vo danh

{

return n%==0;

}

Ham lambda sử dụng toán tử suy ra (=>)

+ Phía bên trái của => là các tham số đầu vào (giống như tham số của một phương thức)

+ Phía bên phải của => là thân của biểu thức (khối mã sẽ được thực thi)

Biểu thức lambda là cách hiện đại và được khuyển nghị để viết hàm vô danh.

LINQ (Language Integrated Query) là một tính năng mạnh mẽ trong C# dùng để truy vấn dữ liệu giống cú pháp của SQL. (ascending tang dan) (descending giam dan)

var largeNumbers = from num in numbers

where num > 10

orderby num ascending

select num;

Các cấp độ truy cập trong C# (public, private, protected, internal)

Public: Thành viên hoặc kiểu có thể truy cập từ bất cứ đâu trong cùng một assembly(project) hoặc từ assembly khác.

Private: Thành viên hoặc kiểu chỉ có thể được truy cập bởi các thành viên khác bên trong định nghĩa của chính nó. Không thể truy cập từ bên ngoài lớp đó, ngay cả từ các lớp con.

Protected: Thành viên chỉ có thể được truy cập bởi các thành viên bên trong lớp khai báo nó hoặc bởi các thành viên trong các lớp kế thừa từ lớp đó. Không thể truy cập từ bên ngoài hệ thông phân cấp kế thừa.

Internal: Thành viên hoặc kiểu có thể được truy cập bởi bất kỳ mã nào trong cùng một assembly mà nó được khai báo. Không thể truy cập từ các assembly khác,

Từ hạn chế nhất đế không hạn chế nhất là private -> protected -> internal -> public

Dành cho lớp chỉ có public và internal, còn dành cho thành viên của lớp thì có đầy đủ

4 hướng đối tượng:

Tính đóng gói, tính kế thừa, tính đa hình, tính trừu tượng

Tính đóng gói (Encapsulation): Là việc gói gọn dữ liệu và hành vi(phương thức) liên quan vào một đơn vị duy nhất, và đồng thời ẩn đi các chi tiết triển khai bên trong.

+ Đặc điểm: Bảo vệ dữ liệu, ngăn truy cập, giảm sự phụ thuộc, dùng private cho dữ liệu, và Properties với get/set để kiểm soát truy cập.

Tính kế thừa (Inheritance): Cho phép một lớp mới(lớp con) nhận tất cả các thuộc tính và phương thức từ một lớp hiện có (lớp cha). Lớp con có thể tái sử dụng, mở rộng hoặc đổi hành vi đã kế thừa.

+ Đặc điểm: Tái sử dụng mã, Quan hệ "is-a": Biểu thị mối quan hệ phân cấp (Dog is a Animal), Thêm tính năng mới hoặc ghi đè cái đã có. Dùng dấu : và từ khóa base, virtual, override.

Tính đa hình (Polymorphism): Khả năng các đối tượng khác nhau (thuộc các lớp liên quan) có thể phản ứng khác nhau với cùng một thông điệp/ lời gọi phương thức ( nhiều hình thức )

+ Đặc điểm: Linh hoạt mã, Cho phép viết code tổng quan hơn, xử lý nhiều loại đối tượng khác nhau qua cùng một giao diện, thêm lớp mới không cần sửa code cũ, Dùng ghi đè phương thức( virtual/ override), abstract class, Interface và nạp chồng phương thức.

Tính trừu tượng (Abstraction): Là quá trình đơn giản hóa những gì phức tạp bằng cách hiển thị những thông tin cốt lõi, cần thiết và ẩn đi các chi tiết triển khai không quan trọng.

Tập trung vào "cái gì" một đối tượng làm, thay vì "làm như thế nào"

+ Đặc điểm: Giảm độ phức tạp, chỉ cần biết cách dùng ko cần biết bên trong, tăng tính dễ hiểu, bảo trì, Dùng Abstract Classes và Interfaces là chính, kết hợp với các access modifies ( private, protected)

Virtual: dùng để đánh dấu một thành viên (phương thức, thuộc tính, sự kiện,..) trong lớp cha cho phép các lớp con có thể ghi đè ( override ) lại hành vi của thành viên đó.

Đặc điểm: Nền tảng của tính đa hình, cho phép đối tượng thực tế quyết định hành vi. Nếu lớp con không ghi đè, nó sẽ dùng hành vi mặc định của lớp cha.

Sealed: dùng để ngăn chặn việc kế thừa hoặc ngăn chặn việc ghi đè thêm.

Đặc điểm: Khi áp dụng một phương thức đã được ghi đè (override): các lớp con xa hơn sẽ không thể ghi đè lại phương thức đó nữa, đảm bảo tính toàn vẹn của thiết kế, ngăn chặn sự thay đổi không mong muốn trong hệ thống phân cấp lớp.

Entity Framework (EF):

Object-Relational Mapper (ORM)

Unit test ( black box, white box) (xUnit, NUnit),Assert.Equal, Assert.True, Assert.Throws, Assert.Contains

ASP.NET (MVC)

ASP.NET CORE ( middleware, accent filter, DI )

Đánh Index, tối ưu SQL database, SHA268, MD5 ( mã hóa mà không giải mã lại được )

Kiểu dữ liệu value và Reference

Value lưu ở vùng stack, Reference lưu vùng heap

# C# Knowledge 1

Hàm vô danh là hàm không có tên và được sử dụng tại chỗ

List<int> evenNumbers = numbers.FindAll(delegate(int num) // Ham vo danh

{

return n%==0;

}

Ham lambda sử dụng toán tử suy ra (=>)

+ Phía bên trái của => là các tham số đầu vào (giống như tham số của một phương thức)

+ Phía bên phải của => là thân của biểu thức (khối mã sẽ được thực thi)

Biểu thức lambda là cách hiện đại và được khuyển nghị để viết hàm vô danh.

LINQ (Language Integrated Query) là một tính năng mạnh mẽ trong C# dùng để truy vấn dữ liệu giống cú pháp của SQL. (ascending tang dan) (descending giam dan)

var largeNumbers = from num in numbers

where num > 10

orderby num ascending

select num;

Các cấp độ truy cập trong C# (public, private, protected, internal)

Public: Thành viên hoặc kiểu có thể truy cập từ bất cứ đâu trong cùng một assembly(project) hoặc từ assembly khác.

Private: Thành viên hoặc kiểu chỉ có thể được truy cập bởi các thành viên khác bên trong định nghĩa của chính nó. Không thể truy cập từ bên ngoài lớp đó, ngay cả từ các lớp con.

Protected: Thành viên chỉ có thể được truy cập bởi các thành viên bên trong lớp khai báo nó hoặc bởi các thành viên trong các lớp kế thừa từ lớp đó. Không thể truy cập từ bên ngoài hệ thông phân cấp kế thừa.

Internal: Thành viên hoặc kiểu có thể được truy cập bởi bất kỳ mã nào trong cùng một assembly mà nó được khai báo. Không thể truy cập từ các assembly khác,

Từ hạn chế nhất đế không hạn chế nhất là private -> protected -> internal -> public

Dành cho lớp chỉ có public và internal, còn dành cho thành viên của lớp thì có đầy đủ

4 hướng đối tượng:

Tính đóng gói, tính kế thừa, tính đa hình, tính trừu tượng

Tính đóng gói (Encapsulation): Là việc gói gọn dữ liệu và hành vi(phương thức) liên quan vào một đơn vị duy nhất, và đồng thời ẩn đi các chi tiết triển khai bên trong.

+ Đặc điểm: Bảo vệ dữ liệu, ngăn truy cập, giảm sự phụ thuộc, dùng private cho dữ liệu, và Properties với get/set để kiểm soát truy cập.

Tính kế thừa (Inheritance): Cho phép một lớp mới(lớp con) nhận tất cả các thuộc tính và phương thức từ một lớp hiện có (lớp cha). Lớp con có thể tái sử dụng, mở rộng hoặc đổi hành vi đã kế thừa.

+ Đặc điểm: Tái sử dụng mã, Quan hệ "is-a": Biểu thị mối quan hệ phân cấp (Dog is a Animal), Thêm tính năng mới hoặc ghi đè cái đã có. Dùng dấu : và từ khóa base, virtual, override.

Tính đa hình (Polymorphism): Khả năng các đối tượng khác nhau (thuộc các lớp liên quan) có thể phản ứng khác nhau với cùng một thông điệp/ lời gọi phương thức ( nhiều hình thức )

+ Đặc điểm: Linh hoạt mã, Cho phép viết code tổng quan hơn, xử lý nhiều loại đối tượng khác nhau qua cùng một giao diện, thêm lớp mới không cần sửa code cũ, Dùng ghi đè phương thức( virtual/ override), abstract class, Interface và nạp chồng phương thức.

Tính trừu tượng (Abstraction): Là quá trình đơn giản hóa những gì phức tạp bằng cách hiển thị những thông tin cốt lõi, cần thiết và ẩn đi các chi tiết triển khai không quan trọng.

Tập trung vào "cái gì" một đối tượng làm, thay vì "làm như thế nào"

+ Đặc điểm: Giảm độ phức tạp, chỉ cần biết cách dùng ko cần biết bên trong, tăng tính dễ hiểu, bảo trì, Dùng Abstract Classes và Interfaces là chính, kết hợp với các access modifies ( private, protected)

Virtual: dùng để đánh dấu một thành viên (phương thức, thuộc tính, sự kiện,..) trong lớp cha cho phép các lớp con có thể ghi đè ( override ) lại hành vi của thành viên đó.

Đặc điểm: Nền tảng của tính đa hình, cho phép đối tượng thực tế quyết định hành vi. Nếu lớp con không ghi đè, nó sẽ dùng hành vi mặc định của lớp cha.

Sealed: dùng để ngăn chặn việc kế thừa hoặc ngăn chặn việc ghi đè thêm.

Đặc điểm: Khi áp dụng một phương thức đã được ghi đè (override): các lớp con xa hơn sẽ không thể ghi đè lại phương thức đó nữa, đảm bảo tính toàn vẹn của thiết kế, ngăn chặn sự thay đổi không mong muốn trong hệ thống phân cấp lớp.

# C# .Net Knowledge 2

Entity Framework là một Object-Relational Mapper (ORM) của .NET do Microsoft phát triển. Nó giúp bạn làm việc với cơ sở dữ liệu bằng sử dụng các đối trượng trong code thay vì phải viết trực tiếp câu lệnh SQL.

Tạo một class student (ID, Name)

sau đó dùng LINQ: var students = dbContext.Students.Where(s => s.Name == "John").ToList();

Object-Relation Mapper (ORM): là một công cụ trung gian giữa mô hình hướng đối tượng (OOP) trong lập trình và một hình quan hệ (Relation) của cơ sở dữ liệu=.

Ánh xạ (mapping) giữa class trong code và table trong cơ sở dữ liệu.

Tự động chuyển đổi truy vấn LINQ sang SQL phù hợp để tương tác với database.

Theo dõi thay đổi của object và cập nhật dữ liệu tự động khi gọi SaveChanges()

Data Annotation: là cách gắn attribute (thuộc tính) trực tiếp lên class và property trong C# để mô tả thông tin Entity Framework.

public class Student

{

[Key] // khoa chinh

public int Id {get; set;}

[Required] // khong duoc null

[MaxLength(100)] // Toi da 100 ky tu

public string Name {get; set;}

[Column("BirthDate")] // Dat ten cot la "BirthDate"

public DateTime DateOfBirth {get; set;}

}

Fluent API: là cách cấu hình mô hình bằng code trong OnModelCreating() của DbContext, sử dụng các phương thức gọi nối tiếp (fluent = trôi chảy)

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Student>(entity =>

{

entity.HasKey(s => s.Id); // Khóa chính

entity.Property(s => s.Name)

.IsRequired() // Không null

.HasMaxLength(100); // Giới hạn 100 ký tự

entity.Property(s => s.DateOfBirth)

.HasColumnName("BirthDate"); // Đặt tên cột

});

}

ASP.NET MVC: là một framework của Microsoft dùng để xây dựng ứng dụng web theo mô hình MVC (Model-View-Controller)

Chạy trên nền tảng .Net Framework(Window only), đã bị thay thế bởi ASP.NET Core kể từ 2016

ASP.NET Core: là một framework hiện đại để phát triền ứng dụng web, API, Realtime app. Đa nền tảng, họp chung Web API, MVC, Razor Pages thành một nền tảng duy nhất. Nhanh hơn, nhẹ hơn.

Middleware: là các thành phần trong pipeline xử lý request/reponse của ASP.NET Core.

Đặc điểm: Request đi qua từng middleware theo thứ tự bạn đăng ký.

Mỗi middleware có thể: Xử lý request (xác thực), Gọi middleware tiếp teo (next()) Hoặc chặn requets tại chỗ.

app.Use(async (context, next ) =>

{

Console.WriteLine("Middleware 1: Before");

await next(); // gọi middleware tiếp theo

Console.WriteLine("Middleware 1: After");

}

app.UseMiddleware<YourCustomMiddleware>();

app.UseRouting();

app.UseEndpoints(...);

Action Filter ( hoặc Filter): là cơ chế trong ASP.NET Core MVC để thêm logic trước hoặc sau khi action/controller chạy. Dùng để xác thực, logging, handle lỗi, chuyển hướng.

DI( Dependency Injection ): là cơ chế inject các dependency (phụ thuộc) như service, logger, repository vào controller hoặc class khác thay vì khởi tạo thủ công. (Dễ test, Tách biệt logic, dễ thay đổi implementation.

builder (WebApplicationBuilder) chính là nơi bạn đăng ký và cấu hình các dịch vụ vào Dependency Injection container (DI container)

var builder = WebApplication.CreateBuilder(args);

AddSingleTon<T>: Toàn bộ vòng đời ứng dụng, tạo một instance duy nhất.

AddScoped<T>: Mỗi request mới, Mỗi request tạo một instance.

AddTransient<T>: Mỗi lần gọi, Tạo mới mỗi lần được yêu cầu.

Mô hình Thác nước (Waterfall Model): là một trong những mô hình SDLC(Software Development Life Cycle) ( Làm đúng ngay từ đầu )

1. Thu thập và phân tích yêu cầu (Requirements Gathering & Analysis), Thu thập toàn bộ yêu cầu từ khách hàng và ghi lại cho tiết trong tài liệu SRS( Software Requirement Specification)

2. Thiết kế( Design ): Dựa trên tài liệu y/c, thiết kế hệ thống, thiết kế csdl, thiết kế giao diện.

3. Triển khai/ lập trình ( implementation/Coding): viết mã nguồi dựa trên thiết kế.

4. Kiểm thử (Testing): tiến hành kiểm tra để tìm lỗi, đảm bảo đúng yêu cầu.

5. Triển Khai ( Deployment ): Phần mềm được cài đặt và triển khai trên môi trường thực tế, để người dùng sử dụng.

6. Bảo trì (Maintenace): Sau khi triển khai, phần mềm tiếp tục được hỗ trợ, sửa lỗi pháp sinh và cải tiến nhỏ.

Mô hình Agile (Agile Model): mềm dẻo, linh hoạt, Thích ứng với thay đổi và mang lại giá trị nhanh chóng thông qua sự công tác.

1. Lập kế hoạch Sprint (Sprint Planning): Đội ngũ và khách hàng thống nhát các tính năng sẽ phát triển trong sprint này.

2. Thiết kế & phát triển (Design & Development): Lập trình viên và designer làm việc để hiện thực hóa các tính năng đã chọn.

3. Kiểm thử (Testing): Các tính năng được phát triển trong sprint được kiểm thử ngay lập tức.

4. Xem xét Sprint( Sprint Review ): Đội ngũ trình bày sản phẩm hoạt dộng được cho khách hàng và thu thập phản hồi.

5. Hồi cứu Sprint (Sprint Retrospective): Đội ngũ tự đánh giá quy trình làm việc để cải tiến cho sprint tiếp theo.

Sự khác biệt giữa class và struct là gì?

Class: có thể kế thừa, kiểu dữ liệu là reference type, các thành viên là private mặc định, phù hợp với những object lớn, phức tạp, có thể dùng waste collector cho quản lý

Struct: không kế thừa, Value Type, thành viên là public, phù hợp với những model cô lập, không dùng garbage collector.

Trong C#, base là một từ khóa đặt biệt được dùng trong các lớp dẫn xuất (lớp con) để tương tác với lớp cơ sơ (lớp cha) của nó

1. bắt buộc khi lớp cha có constructor có tham số, để chọn constructor nào của lớp cha sẽ được gọi hoặc truyền dữ liệu lên lớp cha.

2. Dùng base. khi muốn truy cập thuộc tính của lớp cha.

virtual: cho phép ghi đề nếu muốn, lớp cha có cung cấp sẵn logic mặc định, ko bắt buộc override

abstract: là khai báo trừu tượng, không có nội dung, bắt buộc lớp kế thừa phải override phương thức đó, chỉ dùng được trong lớp abstract.

Parallel.For is indeed built on top of the Task Parallel Library (TPL). So you're using Task indirectly - but in a very optimized and high-level way.

Task uses the .Net ThreadPool unless you explicitly configure it not to.

Luồng nền (Background Thread) là gì?

Luồng nền là một luồng (thread) chạy trong ứng dụng của bạn nhưng nó không giữa cho ứng dụng hoạt động. Nếu tất cả các luồng chính ( foreground thread ) của ứng dụng kết thúc, thì ứng dụng sẽ dừng hoàn toàn, dù luồng nền có đang chạy.

Vấn đề của mình là: Hàm "Đa nhiệm" với logic "Đa dạng"

Nếu cứ viết các if/else if hoặc switch dựa vào tên dự án hoặc một flag nào đó, code sẽ rất nhanh chóng trở nên khó quản lý, khó mở rộng và khó kiểm thử. Đây là những dấu hiện rõ ràng cho thấy cần một giải pháp OOP.

Giải pháp: Sử dụng Abstraction( trừu tượng) và Polymorphism (Tính đa hình).

Tạo một Interface hoặc abstract class (lớp trừu tượng) để biểu diễn hành vi của ProcessData()

Interface: phổ biến, chỉ định nghĩa "Hợp đồng" (contract) về phương thức cần có.

IProjectProcessor với phương thức ProcessData()

Abstract Class: đặc biệt nếu có một số logic chung có thể được triển

khai mặc định cho tất cả dự án, hoặc nếu muốn cung cấp các trường fields

AbstractProjectProcessor với phương thức abstract ProcessData()

Tính đa hình: cho phép làm như thế nào

Sau khi định nghĩa abstraction, mỗi dự án sẽ tạo ra một lớp cụ thể ( concrete class ) để triển khai implement hoặc kế thừa (inherit)

ProjectAProcessor implement IProjectProcessor.

# C# Knowledge 3

1NF (First Normal Form): Tất cả các thuộc tính (cột) đều chứa giá trị nguyên tử ( atomic values ), không có nhóm lặp lại (Repeating jobs)

2NF (Second): Đã đạt 1NF, tất các thuộc tính không khóa phải phụ thuộc hàm hoàn toàn bộ khóa chính, loại bỏ các phụ thuộc một phần (partial dependency), giảm sự dư thừa dữ liệu. Tách các bảng ra để bỏ các giá trị phụ thuộc một phần.

3NF (Third): Đạt 2NF và khong phụ thuộc bắc cầu của thuộc tính không khóa vào khóa chính, loại bỏ phụ thuộc bắc cầu, giảm tối đa dư thừa và bất thường dl.

Tại sao ConcurrentQueue lại nhanh hơn queue dùng lock ?

ConcurrentQueue sử dụng thuật toán không khóa (Lock-Free algorithms) hoặc khóa hạt mịn (Fine-Grained Locking):

Lock free: Thay vì chặn luồng khác, chúng sẽ thử lại thao tác nếu có xung đột.

Fine-Grained Locking: Không hoàn toàn không khóa, khóa riêng cho đầu và cuối hàng đợi, cho phép enqueue và dequeue diễn ra song song. lock queue thì khóa nguyên đối tượng queue.

Tham số (parameters): void test(string t, int total) các giá trị khi khởi tạo hàm

Đối số (arguments): test("Thinh", 5) các giá trị bỏ vào hàm khi gọi

out (Output Parameter): Dùng để trả về giá trị của một phương thức. checkedResultDict.TryGetValue(compareString, out var checkedValue)

ref (Reference Parameter): Dùng để sửa đổi giá trị hiện có của một biến. Phương thức có thể đọc và ghi vào biến được truyền vào.

DatabaseCompare(detectModel.Text, ref currentCheckedIndex)

Nếu trong hàm này thay đổi giá trị currentCheckedIndex, giá trị này sẽ thay đổi, thay đổi giá trị gốc.

Đa hình tĩnh (Static Polymorphism): Nhiều phương thức cùng tên trong cùng một lớp, khác như tham số, kiểu tham số, số lượng,.. Add(int a) Add(double a)

Đa hình động (Dynamic Polymorphism): Phương thức có cùng tên nhưng các lớp khác nhau. Lớp cha có phương thức virtual hoặc abstract, lớp con sài override

Use ORM (Object Relational Mapper) and Entity Framewor Core to interact with database:

+ Tạo một mô hình dữ liệu (Model): Student.

+ Thiết lập DbContext để tương tác với cơ sở dữ liệu.

+ Cấu hình kết nối đến SQL Server.

+ Sử dụng Migrations để tạo và cập nhật lược đồ cơ sở dữ liệu.

+ Thực hiện các thao tác CRUD (Create, Read, Update, Delete).

Middleware trong ASP.Net Core: là các thành phần mềm được sắp xếp thành một chuỗi (pipeline) để xử lý các yêu cầu (requests) và phản hồi (Responses) của ứng dụng. Tưởng tượng middleware là đường hầm, mỗi đường hầm sẽ kiểm tra liệu có cho đi tiếp hay không.

1. Routing định tuyến các y/c endpoint

2. Authentication: Xác thực người dùng

3. Authorization: Phần quyền người dùng

4. Quản lý Session, CORS (Cross-Origin Resource Sharing), Chuyển hướng HTTP sang HTTPS.

IDisposable: là một interface trong .NET định nghĩa một phương thức duy nhất: void Dispose(): lớp con sẽ viết hàm dispose các connection, stream trong đây.

Mục đích để giải phóng tài nguyên không được quản lý (unmanaged resources) như: File handles, Network sockets, COM objects, database connections.

static int Reverse(int x)

{

try

{

if (x == 0) return 0;

string sValue = x.ToString();

StringBuilder result = new StringBuilder();

for (int i = sValue.Length - 1; i >= 0; i--)

{

if (!char.IsDigit(sValue[i])) continue;

result.Append(sValue[i]);

}

if (x < 0)

return int.Parse(result.ToString()) \* -1;

return int.Parse(result.ToString());

}

catch (Exception ex)

{

Console.WriteLine("Reverse error:" + ex.Message);

return 0;

}

}

public class Solution

{

public int Reverse(int x)

{

int result = 0;

while (x != 0)

{

int digit = x % 10;

x /= 10;

// Check for overflow/underflow before updating result

if (result > int.MaxValue / 10 || (result == int.MaxValue / 10 && digit > 7))

{

return 0;

}

if (result < int.MinValue / 10 || (result == int.MinValue / 10 && digit < -8))

{

return 0;

}

result = result \* 10 + digit;

}

return result;

}

}

# C# .Net Knowledge 4

Record is a reference type introduced in C# 9.0 that is primarily used to define immutable data models with value-based equality.

Immutable by default, Value-based equality: Two records are equal if all their property values are equal, concise syntax for defining data containers, with expressions.

public record Person(string FirstName, string LastName);

var person1 = new Person("John", "Doe");

var person2 = new Person("John", "Doe");

Console.WriteLine(person1 == person2); // True (compares values)

var person3 = person1 with {FirstName = "Thinh"}

# C# Knowledge 5

Lifecycle hooks in ASP.net: Singleton, Scoped, Trasient

Interceptor: is a design pattern that lets you inject beharvior before or after a method is executed, without modifying the actual method code. ( Loggings, Caching, retry policies, validation, authorization, performance monitoring)

Public api is an API that is intended to be exposed to external users - like customers, partners, mobile, apps or third-party developers.

[ApiController]

[Route("api/users")]

Private api is only used internally within your organization, project or system. It's not intende to be used outside your control.

[ApiController]

[Route("internal/config")]

[Authorize(Roles = "Admin")]

Man in the middle:

Content negotiation

Data connector, lazy load,..

Raw SQL, when use sync, async, logging, caching.

MSSQL (Microsoft SQL Server) là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) do Microsoft phát triển. Nó được thiết kế để lưu trữ, truy xuất và quản lý dữ liệu một cách hiệu quả cho các ứng dụng phần mềm.

1. MSSQL dùng để làm gì?

MSSQL được dùng để:

Quản lý dữ liệu trong các ứng dụng web, desktop, enterprise.

Hỗ trợ truy vấn dữ liệu bằng SQL (Structured Query Language).

Tạo và quản lý bảng, mối quan hệ, chỉ mục, stored procedure, view, trigger,...

Phù hợp cho các hệ thống lớn với yêu cầu bảo mật và hiệu năng cao.

2. Các thành phần chính của MSSQL

SQL Server Database Engine: lõi chính để xử lý lưu trữ và truy vấn dữ liệu.

SQL Server Management Studio (SSMS): công cụ giao diện để quản trị và viết truy vấn.

SQL Agent: lập lịch và tự động hóa các tác vụ.

Integration Services (SSIS): dùng để xử lý và chuyển đổi dữ liệu.

Analysis Services (SSAS): dùng để phân tích dữ liệu (OLAP).

Reporting Services (SSRS): tạo báo cáo.

# C# Knowledge 6

foreach (var entry in \_queue.GetConsumingEnumerable(\_cts.Token))

is used inside the background thread to continously process data from the queue in a thread-safe and blocking way.

\_queue: This is a BlockingCollection<VerificationEntry>, a thread-safe queue.

.GetConsumingEnumerable(...):

- Returns an IEnumerable<T> that removes (consumes) items from the queue one by one.

- Blocks the thread and waits when the queue is empty (instead of busy-waiting)

- Automatically exits when queue is marked complete using \_queue.CompleteAdding()

BlockingCollection<T>: Implement a Producer-Consumer pattern, Consumer thread should wait automatically for new items.

It uses proper synchronization primitives like Monitor.Wait() or semaphores

Perfect for background thread, less CPU usage, slight overhead (blocking/locking)

ConcurrentQueue<T>: It does not block if there's no item to dequeue, just returns false.

Manually checks for data and polling in a while loop.

Faster with lock-free, fine-grained

# WPF Knowledge

WPF (Windows Presentation Foundation) is a UI framework for building desktop applications in .NET. IT uses XAML for UI definition and supports modern features like databinding, templates, styles, and animations. WinForms is older, event-driven, and lacks WPF's styling, binding, and layout capabilities.

Key components: XAML supports, Core rendering APIs, Basic types like DependencyObject. MIL (Media Integration Layer): DirectX redering, CLR: Common Language Runtime.

XAML (Extensible Appication Markup Language) is a declarative language to define UI layout and behavior in WPF. It separates UI from business logic.

StackPanel: Arranges children in a single line (horizontal/vertical)

WrapPanel: Arranges children and wraps to the next line if there's no space.

Responsive layout: Use containers like Grid with \* sizing, Viewbox, or DockPanel. Combine with Min/Max size properties

Grid divides the UI into rows and columns using RowDefinitions and ColumnDefinitions.

Use <Styles> tags in Resources, create ResourceDictionary, or apply themes via DynamicResource and StaticResource

UserControl: Combines existing controls with XAML and code-behind.

CustomControl: Derives from Control and is more reusable, skinnable via ControlTemplate.

Types of data binding in WPF: OneWay, TwoWay, OneTime, OneWaytoSource.

INotifyPropertyChanged: notify the UI when a property value changes, enabling automatic updates in data binding.

Bind a list of objects to a ListBox or ComboBox:

<ListBox ItemsSource={Binding MyItems}" DisplayMemberPath="Name" />

DataTemplate: defines how to display data objects in UI elements like LIst, ItemsControl.

MVVM (Model-View-ViewModel) seperates UI (View), logic (ViewModel), and data (Model), making code more testable and maintainable.

Commands: replaces event handles in ViewModels using ICommand to bind UI actions to logic.

A class implementing ICommand to define reusable commands with logic and CanExecute checks.

Communicating between ViewModels: Use Messenger services (MVVM Light Messenger), EventAggregator (Prism), or shared services.

WPF application using MVVM: Model (data), ViewModels (logic + binding) Views (XAML), Use frameworks like MVVM Light, Prism, or CommunityToolkit.MVVM.

Dispatcher.Invoke or Dispatcher.BeginInvoke to update UI from background threads.

Use RoutedEvent.Register and raise them with RaiseEvent. Routed events can bubble, tunnel, or be direct.

Attached properties are dependency properties defined in one class but used in others, commonly in layout (Grid.Row, Canvas.Left)

Improve performance in large-scale WPF application: VirtualizingStackPanel, avoid unnecessary bindings, Use Freezable objects, Reduce visual tree depth, Defer loading with IsAsync, Data Virtualization.

Nếu chạy R-Link thì không thể chạy 6 line qr + 1 đầu index vì nó sẽ không đồng nhất và R-Link thì không thể quản lý được 2 máy in. Một máy in chỉ có thể in max là 6 đầu (thì không thể 7). Nếu muốn chạy thì phải chạy Dipes-Link như In nông nghiệp quản lý 2 máy in in cùng lúc.

.net Core Knowledge

1. AddSingleton: register a service wit singleton lifetime in the dependency injection (DI) container.

I used AddSingleton to register the BarcodeQueueService because i needed a globally shared instance of the queue throughout the application's lifetime. The queue (ConcurrentQueue<BarcodeData>) acts as a shared buffer betwee the web API controller (producer) and the background worker (consumer). Using Singleton ensures no duplication of state and eliminates the need for synchronization between multiple instances.

2/ AddHostedService is used to run background tasks in ASP.NET Core using a class that implements BackgroundService or IHostedService.

I registerd BarcodeQueueWorker wit AddHostedService to create a background processing service. This allows me to decouple the heavy DB write logic from the controller and let it run independently. The worker runs in the background for the lifetime of the application and periodically dequeues barcode data and writes it to the database in batches.

3. IServiceScopeFactory is used to create a new DI scope manually. It gives access to scoped services (like DbContext) outside of the request pipeline.

In BarcodeQueueWorker, we're outside of the normal web request lifecycle, so we can't inject DbContext directly with a scoped lifetime. Instead, I injected IServiceScopeFactjoory, which lets me manually create a scope inside the background worker. The I resolve the AppDbContext safely within that scope, use it to save batch data, and let it be disposed immediately after.

A subclass of BackgroundService, an abstract class provided by ASP.NET Core for writing long-running background tasks.

BarcodeQueueWorker implements BackgroundService, which lets me run a clean infinite loop with a cancellation toke. Inside the ExecuteAsync method, I implemented logic: Collect into a batch with 100 records and waits 1 millisecond, save it to DB

Scope life cycle: when request hits a controller, ASP.NET Core automatically creates a scope for that request and injects service like DbContext, ILogger, background service like BarcodeQueueWorker are created at application startup, not per request, so they do not have a scope by default.

Using: ensures automatically free memmory, close open DB connections, avoid memory leaks.

Singleton: App lifetime, Controller, Background -> BarcodeQueueService.

Scoped: One Http request/scope , Controllers (auto), Worker (manual), DbContext, HttpContext.

Transient: Every time it's requested, both, lightweight stateless logic.