1. **dotNET là gì?**

dotNET là một nền tảng phát triển phần mềm mạnh mẽ và linh hoạt được Microsoft phát triển. Nó cung cấp một môi trường chạy ứng dụng đa nền tảng và hỗ trợ việc xây dựng các ứng dụng từ máy tính cá nhân đến các hệ thống phân tán.

.NET là viết tắt của "dotNET".

Nền tảng: .NET cung cấp một bộ công cụ, thư viện và các dịch vụ cần thiết để phát triển phần mềm.

Đa ngôn ngữ: có thể sử dụng các ngôn ngữ được phát triển bởi Ms như C#, VB.NET, F# để phát triển ứng dụng trên nền tảng .NET.

Đa nền tảng: Các ứng dụng .NET có thể chạy trên nhiều hệ điều hành khác nhau như Windows, Linux, macOS.

1. Tại sao nên chọn .NET?

Hiệu năng cao: .NET được tối ưu hóa để cung cấp hiệu suất tốt, giúp các ứng dụng chạy nhanh và mượt mà.

Độ tin cậy: .NET có một cộng đồng lớn và được Microsoft hỗ trợ mạnh mẽ, đảm bảo tính ổn định và bảo mật.

Linh hoạt: .NET cho phép xây dựng nhiều loại ứng dụng khác nhau, từ ứng dụng web, ứng dụng desktop đến ứng dụng di động.

1. Có thể làm gì với .NET?

Phát triển ứng dụng web: Xây dựng các trang web động, các ứng dụng web doanh nghiệp, các cửa hàng trực tuyến.

Phát triển ứng dụng desktop: Tạo các ứng dụng chạy trên máy tính cá nhân.

Phát triển ứng dụng di động: Xây dựng các ứng dụng chạy trên các thiết bị di động như điện thoại, máy tính bảng.

Phát triển game: Sử dụng các công cụ và thư viện của .NET để tạo các trò chơi.

Phát triển các dịch vụ đám mây: Xây dựng các ứng dụng chạy trên nền tảng đám mây.

1. Các thành phần chính của .NET

Framework .NET: Cung cấp các lớp và thư viện cơ bản để phát triển ứng dụng.

ASP.NET: Framework để xây dựng các ứng dụng web.

Windows Forms: Framework để xây dựng các ứng dụng desktop.

WPF: Framework để xây dựng các ứng dụng desktop giàu giao diện.

Entity Framework: Framework để tương tác với cơ sở dữ liệu.

LINQ: Ngôn ngữ truy vấn tích hợp để làm việc với dữ liệu.

.NET là một nền tảng phát triển phần mềm mạnh mẽ và linh hoạt, được sử dụng rộng rãi trong việc xây dựng các ứng dụng đa dạng.

1. **ASP.NET là gì?**

ASP.NET là một nền tảng phát triển web mạnh mẽ và linh hoạt, được Microsoft tạo ra để xây dựng các ứng dụng web hiện đại và dịch vụ web. Nó là một phần của .NET Framework, cung cấp cho các nhà phát triển một bộ công cụ và thư viện phong phú để tạo ra các ứng dụng web tương tác, hiệu suất cao.

1. Tại sao nên chọn ASP.NET?

Hiệu năng cao: ASP.NET được tối ưu hóa để cung cấp hiệu suất vượt trội, giúp các ứng dụng web của chạy nhanh và mượt mà hơn.

Linh hoạt: ASP.NET là một phần của hệ thống của dotNET nên những ngôn ngữ như C#, F#, VB.NET đều được hỗ trợ một cách ổn định.

Dễ sử dụng: Với các công cụ và thư viện sẵn có, ASP.NET giúp phát triển ứng dụng web nhanh chóng và dễ dàng hơn.

Cộng đồng lớn: ASP.NET có một cộng đồng người dùng đông đảo, giúp người dùng dễ dàng tìm kiếm tài liệu, giải pháp và hỗ trợ khi cần.

Hỗ trợ đa nền tảng: ASP.NET Core, phiên bản mới nhất của ASP.NET, cho phép triển khai ứng dụng trên nhiều nền tảng khác nhau như Windows, Linux, macOS và Docker.

1. Cách thức hoạt động của ASP.NET

Khi một người dùng truy cập vào một trang web được xây dựng bằng ASP.NET, yêu cầu sẽ được gửi đến máy chủ web. Máy chủ sẽ xử lý yêu cầu này bằng cách thực thi mã ASP.NET và tạo ra HTML để gửi trở lại trình duyệt của người dùng.

1. Các dạng chính của ASP.NET:

ASP.NET Web Forms: Một mô hình lập trình trực quan, giúp dễ dàng tạo các trang web tương tác.

ASP.NET MVC: Một framework dựa trên mô hình MVC (Model-View-Controller), giúp tách biệt các phần của ứng dụng, tạo ra các ứng dụng web dễ bảo trì và mở rộng.

ASP.NET Web API: Một framework để xây dựng các dịch vụ web RESTful, cung cấp dữ liệu cho các ứng dụng di động và ứng dụng web khác.

SignalR: Một thư viện giúp xây dựng các ứng dụng web thời gian thực, cho phép giao tiếp hai chiều giữa máy chủ và trình duyệt.

1. Ứng dụng của ASP.NET:

ASP.NET được sử dụng rộng rãi để xây dựng các loại ứng dụng web khác nhau, bao gồm:

Ứng dụng web doanh nghiệp: Các ứng dụng quản lý nội bộ, CRM, ERP, ...

Trang web thương mại điện tử: Các trang web bán hàng trực tuyến.

Ứng dụng web xã hội: Các mạng xã hội, diễn đàn, ...

Ứng dụng web di động: Các ứng dụng web có thể truy cập từ các thiết bị di động.

1. **PostgreSQL**

**PostgreSQL** (hay còn gọi là Postgres) là một hệ thống quản lý cơ sở dữ liệu quan hệ-đối tượng (object-relational database management system - ORDBMS) mã nguồn mở, được biết đến với độ tin cậy, tính năng phong phú và khả năng mở rộng cao. Postgres được sử dụng rộng rãi trong các ứng dụng web, phân tích dữ liệu, và các hệ thống yêu cầu độ ổn định cao.

**Ưu điểm của PostgreSQL:**

* Mã nguồn mở: Miễn phí sử dụng, tùy chỉnh và phân phối.
* Tính năng phong phú: Hỗ trợ đa dạng các kiểu dữ liệu (bao gồm cả các kiểu dữ liệu không cấu trúc như JSON, XML), các giao dịch phức tạp, các truy vấn nâng cao và nhiều tính năng khác.
* Khả năng mở rộng: Có thể xử lý các cơ sở dữ liệu lớn và các khối lượng dữ liệu khổng lồ.
* Đa nền tảng: Có thể chạy trên nhiều hệ điều hành khác nhau như Windows, Linux, macOS.
* Cộng đồng lớn: Có một cộng đồng người dùng và phát triển sôi động, giúp hỗ trợ và phát triển PostgreSQL.
* An toàn và bảo mật: Cung cấp nhiều tính năng bảo mật như kiểm soát truy cập, mã hóa dữ liệu, sao lưu và phục hồi.
* Tuân thủ tiêu chuẩn SQL: Tuân thủ chặt chẽ tiêu chuẩn SQL, giúp dễ dàng di chuyển dữ liệu giữa các hệ thống.

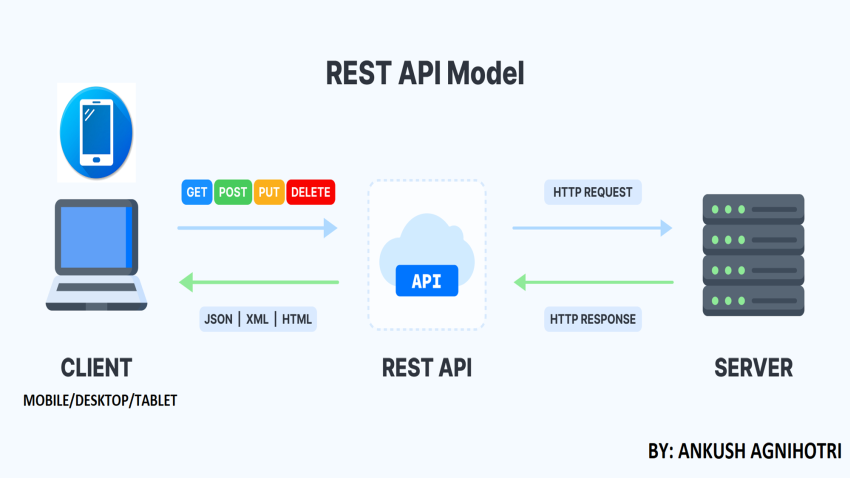
**Nhược điểm của PostgreSQL:**

* Độ phức tạp: Do tính năng phong phú, PostgreSQL có thể phức tạp hơn so với các hệ quản trị cơ sở dữ liệu khác, đòi hỏi người dùng phải có kiến thức sâu hơn.
* Hiệu năng: Mặc dù hiệu năng tốt nhưng có thể không tốt bằng một số hệ quản trị cơ sở dữ liệu chuyên dụng cho một số tác vụ cụ thể.

**Khi nào nên sử dụng PostgreSQL?**

* Các ứng dụng yêu cầu độ tin cậy cao và khả năng xử lý dữ liệu lớn.
* Các ứng dụng cần các tính năng nâng cao như phân tích dữ liệu, các giao dịch phức tạp.
* Các ứng dụng cần một hệ quản trị cơ sở dữ liệu mã nguồn mở và linh hoạt.
* Các ứng dụng cần tuân thủ các tiêu chuẩn SQL.

1. **Liên kết giữa .NET và PostgreSQL qua API**



* 1. **Cách liên kết và mối quan hệ của môi trường**

dotNET và PostgreSQL là hai công nghệ mạnh mẽ, thường được kết hợp để xây dựng các ứng dụng web, dịch vụ và hệ thống back-end. PostgreSQL đóng vai trò là cơ sở dữ liệu để lưu trữ và quản lý dữ liệu, trong khi .NET cung cấp nền tảng và công cụ để phát triển ứng dụng truy xuất và xử lý dữ liệu đó.

* 1. **Cách thức kết nối**

Để kết nối .NET với PostgreSQL, thông thường chúng ta sử dụng **API**. API (Application Programming Interface) là một tập hợp các quy tắc và định nghĩa cho phép các phần mềm khác nhau giao tiếp với nhau.

* 1. **Các công cụ và thư viện phổ biến:**
* **Npgsql:** Đây là ADO.NET data provider phổ biến nhất để kết nối .NET với PostgreSQL. Npgsql cung cấp một cách đơn giản để thực hiện các truy vấn SQL, thực hiện các giao dịch và tương tác với cơ sở dữ liệu.
* **Entity Framework Core:** Một ORM (Object-Relational Mapper) cho phép làm việc với cơ sở dữ liệu bằng các đối tượng C# thay vì viết trực tiếp các câu lệnh SQL. Entity Framework Core có thể kết nối với PostgreSQL thông qua provider Npgsql.
* **Dapper:** Một micro-ORM nhẹ nhàng, nhanh chóng và linh hoạt, cho phép thực hiện các truy vấn SQL một cách trực tiếp và ánh xạ kết quả vào các đối tượng C#.
  1. **Quy trình kết nối cơ bản:**

1. **Cài đặt:** Cài đặt PostgreSQL và các thư viện cần thiết (Npgsql, Entity Framework Core, Dapper) vào dự án .NET thông qua hệ thống NuGet Package Management với hệ thống tìm kiếm trực quan và cũng có thể cài đặt thông qua các lệnh thông qua Terminal
2. **Tạo chuỗi kết nối:** Chuỗi kết nối chứa thông tin để kết nối với cơ sở dữ liệu, bao gồm tên máy chủ, tên cơ sở dữ liệu, tên người dùng và mật khẩu. Nó thường có dạng cơ bản là:

"ConnectionStrings": {

"DefaultConnection": "Host={Địa-chỉ-hoạt-động-của-DB};Database={Tên-csdl};Username={Tên-người-dùng};Password={Mật-khẩu}"

}

Thường câu lệnh nằm trong “*appsetting.json*” để định nghĩa kết nối.

1. **Mở kết nối:** Sử dụng thư viện đã chọn để mở kết nối đến cơ sở dữ liệu.

Các thư viện sẽ được khai báo trong một file “***.csproj***” để hoạt động trong dự án thường dưới dạng câu lệnh:  
<PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL" Version="8.0.10" />

Câu lệnh trên có tác dụng khai báo thư viện Npssql để kết nối cơ sở dữ liệu PostgreSQL với phiên bản sử dụng là 8.0.10

1. **Thực hiện truy vấn:** Thực hiện các truy vấn SQL để truy xuất, chèn, cập nhật hoặc xóa dữ liệu.

Truy vấn có thể sử dụng trực tiếp câu lệnh SQL nhưng thường sử dụng các thư viện truy vấn dữ liệu từ như LINQ để đảm bảo an toàn thông tin dữ liệu và code sẽ dễ đọc và bảo trì hơn.

1. **Đóng kết nối:** Đóng kết nối khi hoàn tất các thao tác với cơ sở dữ liệu.

**Ví dụ đơn giản với Npgsql:**

using Npgsql;

// Chuỗi kết nối

string connectionString = "Server=yourserver;Database=yourdatabase;User Id=youruser;Password=yourpassword;";

using (NpgsqlConnection connection = new NpgsqlConnection(connectionString))

{

connection.Open();

using (NpgsqlCommand command = new NpgsqlCommand("SELECT

\* FROM your\_table", connection))

{

using (NpgsqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

// Xử lý dữ liệu đọc được

Console.WriteLine(reader.GetString(0));

}

}

}

}

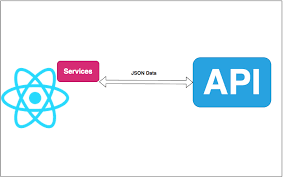
**Lợi ích của việc kết hợp .NET và PostgreSQL:**

* **Hiệu suất cao:** Cả .NET và PostgreSQL đều được tối ưu hóa về hiệu năng, giúp các ứng dụng chạy nhanh và mượt mà.
* **Linh hoạt:** Cung cấp nhiều tùy chọn để tương tác với cơ sở dữ liệu, từ việc viết các câu lệnh SQL trực tiếp đến việc sử dụng các ORM.
* **Cộng đồng lớn:** Cả .NET và PostgreSQL đều có cộng đồng người dùng lớn, giúp dễ dàng tìm kiếm tài liệu, giải pháp và hỗ trợ.
* **Mở rộng:** Có thể dễ dàng mở rộng ứng dụng để xử lý các khối lượng dữ liệu lớn và các yêu cầu phức tạp.

**Ứng dụng thực tế:**

* **Phát triển ứng dụng web:** Xây dựng các trang web thương mại điện tử, các ứng dụng quản lý nội dung, các hệ thống quản lý khách hàng.
* **Phát triển ứng dụng desktop:** Tạo các ứng dụng desktop truy cập dữ liệu từ PostgreSQL.
* **Phân tích dữ liệu:** Xây dựng các ứng dụng phân tích dữ liệu lớn, báo cáo và trích xuất dữ liệu.
* **Dịch vụ web:** Xây dựng các API để cung cấp dữ liệu cho các ứng dụng khác.

1. **Kết nối ReactJS với API**



**ReactJS** là một thư viện JavaScript phổ biến để xây dựng giao diện người dùng (UI) cho các ứng dụng web. **API (Application Programming Interface)** là một tập hợp các quy tắc và định nghĩa cho phép các phần mềm khác nhau giao tiếp với nhau. Việc kết nối ReactJS với API là điều cần thiết để xây dựng các ứng dụng web động, lấy dữ liệu từ phía backend và hiển thị lên giao diện người dùng.

**Tại sao cần kết nối ReactJS với API?**

* **Lấy dữ liệu động:** Thay vì viết cứng dữ liệu vào code, ReactJS có thể lấy dữ liệu từ API để tạo ra các giao diện động và cập nhật.
* **Tách biệt frontend và backend:** Việc tách biệt logic nghiệp vụ và giao diện giúp cho code dễ bảo trì và mở rộng hơn.
* **Tương tác với các hệ thống khác:** API cho phép ReactJS tương tác với các hệ thống khác như cơ sở dữ liệu, dịch vụ thanh toán, v.v.

**Cách thức kết nối**

Có nhiều cách để kết nối ReactJS với API, nhưng phổ biến nhất là sử dụng các thư viện như:

* **Fetch API:** Đây là API gốc của trình duyệt, cung cấp các phương thức để thực hiện các yêu cầu HTTP.
* **Axios:** Một thư viện HTTP client phổ biến, cung cấp các tính năng bổ sung như việc xử lý lỗi, intercept, v.v.
* **React Query:** Một thư viện giúp quản lý các yêu cầu API, caching, và đồng bộ hóa dữ liệu.

**Ví dụ sử dụng Axios**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function App() {

const [posts, setPosts] = useState([]);

useEffect(()

=> {

const fetchPosts = async () => {

const response = await axios.get('https://jsonplaceholder.typicode.com/posts');

setPosts(response.data);

};

fetchPosts();

}, []);

return (

<div>

{posts.map(post => (

<div key={post.id}>

<h2>{post.title}</h2>

<p>{post.body}</p>

</div>

))}

</div>

);

}

export default App;

Trong ví dụ trên:

1. **Import axios:** Thêm thư viện axios vào dự án.
2. **Sử dụng useState:** Tạo một state posts để lưu trữ danh sách các bài viết.
3. **useEffect:** Gọi API để lấy danh sách bài viết khi component được mount.
4. **Hiển thị dữ liệu:** Hiển thị danh sách bài viết trong giao diện.

**Các bước cơ bản khi kết nối ReactJS với API**

1. **Xác định API:** Biết rõ endpoint của API, các phương thức HTTP được hỗ trợ (GET, POST, PUT, DELETE), và định dạng dữ liệu trả về (JSON, XML).
2. **Chọn thư viện:** Chọn một thư viện phù hợp để thực hiện các yêu cầu HTTP.
3. **Gửi yêu cầu:** Sử dụng thư viện đã chọn để gửi yêu cầu đến API.
4. **Xử lý phản hồi:** Xử lý dữ liệu nhận được từ API và cập nhật state của component.
5. **Hiển thị dữ liệu:** Hiển thị dữ liệu trong giao diện người dùng.

**Các vấn đề cần lưu ý**

* **Quản lý trạng thái:** Sử dụng các hook như useState hoặc useReducer để quản lý trạng thái của component.
* **Xử lý lỗi:** Xử lý các lỗi có thể xảy ra trong quá trình gọi API, ví dụ như lỗi mạng, lỗi server.
* **Tối ưu hóa hiệu năng:** Sử dụng các kỹ thuật như caching, lazy loading để cải thiện hiệu năng của ứng dụng.
* **Bảo mật:** Bảo vệ dữ liệu người dùng bằng cách sử dụng HTTPS và các biện pháp bảo mật khác.

**Các thư viện hỗ trợ khác**

* **React Query:** Cung cấp một cách quản lý các yêu cầu API mạnh mẽ hơn, bao gồm caching, refetching, và nhiều tính năng khác.
* **Redux:** Một thư viện quản lý trạng thái toàn cục, có thể được sử dụng để quản lý dữ liệu lấy từ API.

**Vai trò của Models, Controllers, Data và DTOs trong ASP.NET Web API**

**ASP.NET Web API** là một framework mạnh mẽ để xây dựng các dịch vụ web RESTful. Để hiểu rõ cách các thành phần này hoạt động cùng nhau:

**Models**

* **Đại diện cho dữ liệu:** Models đại diện cho các đối tượng trong ứng dụng của, thường tương ứng với các bảng trong cơ sở dữ liệu. Ví dụ: một model Product có thể có các thuộc tính như Id, Name, Price.
* **Xác định cấu trúc dữ liệu:** Các properties của một model xác định cấu trúc của dữ liệu mà nó chứa.
* **Thực thi các quy tắc nghiệp vụ:** Models có thể chứa logic để đảm bảo tính hợp lệ của dữ liệu, ví dụ như kiểm tra ràng buộc, thực hiện tính toán.

**Controllers**

* **Điểm vào của các yêu cầu:** Controllers là điểm đầu tiên tiếp nhận các yêu cầu HTTP từ client.
* **Xử lý logic:** Controllers chứa các phương thức (actions) để xử lý các yêu cầu HTTP khác nhau (GET, POST, PUT, DELETE).
* **Trả về dữ liệu:** Sau khi xử lý yêu cầu, controllers trả về dữ liệu cho client, thường dưới dạng JSON hoặc XML.
* **Gọi đến các service:** Controllers thường không thực hiện trực tiếp các nghiệp vụ logic phức tạp mà sẽ gọi đến các service để thực hiện.

**Data**

* **Truy cập dữ liệu:** Thư mục Data thường chứa các lớp để truy cập vào cơ sở dữ liệu, như các lớp DbContext trong Entity Framework.
* **Thực hiện các truy vấn:** Các lớp này cung cấp các phương thức để thực hiện các truy vấn SQL, truy xuất, cập nhật, thêm mới và xóa dữ liệu.

**DTOs (Data Transfer Objects)**

* **Truyền dữ liệu:** DTOs được sử dụng để truyền dữ liệu giữa các tầng của ứng dụng, thường giữa controller và client.
* **Giấu thông tin:** DTOs có thể chứa một phần hoặc tất cả các thuộc tính của một model, giúp kiểm soát thông tin nào được gửi đến client.
* **Tăng cường bảo mật:** Bằng cách sử dụng DTOs, có thể loại bỏ các thuộc tính nhạy cảm khỏi dữ liệu được trả về cho client.

**Sơ đồ tương tác**

flowchart LR

A(Client) --> B(Controller)

B --> C(Service)

C --> D(Data)

C --> B

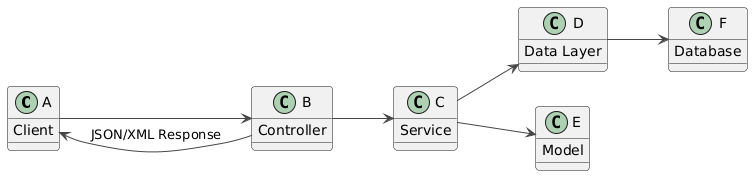
B --> A(JSON/XML)

subgraph Cluster 1

direction TB

C((Service)) --> E((Model))

end



Trong sơ đồ trên:

* **Client:** Gửi yêu cầu đến controller.
* **Controller:** Nhận yêu cầu, gọi đến service để xử lý.
* **Service:** Thực hiện logic nghiệp vụ, truy cập dữ liệu thông qua data layer và trả về kết quả cho controller.
* **Data:** Truy cập cơ sở dữ liệu để lấy hoặc cập nhật dữ liệu.
* **Model:** Đại diện cho dữ liệu trong ứng dụng.
* **DTO:** Được sử dụng để truyền dữ liệu giữa controller và client.

**Ví dụ**

Giả sử có một API để quản lý sản phẩm. Khi một client gửi yêu cầu GET để lấy danh sách sản phẩm, quá trình sẽ diễn ra như sau:

1. Client gửi yêu cầu GET đến controller /api/products.
2. Controller gọi đến service ProductService để lấy danh sách sản phẩm từ cơ sở dữ liệu.
3. Service sử dụng DbContext để truy vấn cơ sở dữ liệu và trả về một danh sách các đối tượng Product.
4. Controller chuyển đổi danh sách Product thành một danh sách các đối tượng ProductDto (chỉ chứa các thuộc tính cần thiết) và trả về cho client dưới dạng JSON.
5. **Migrations trong ASP.NET Web API**

**1. Migrations** trong ASP.NET Core Web API

**Migrations** là một công cụ mạnh mẽ giúp quản lý sự thay đổi cơ sở dữ liệu theo thời gian. Khi người dùng cần thêm một cột mới vào bảng, thay đổi kiểu dữ liệu của một cột, hoặc thậm chí tạo một bảng hoàn toàn mới, migrations sẽ tự động tạo ra các script để thực hiện những thay đổi đó một cách an toàn và có kiểm soát. Có thể tóm gọn các chức năng của Migrations

* **Quản lý phiên bản cơ sở dữ liệu:** Giúp theo dõi lịch sử các thay đổi của cơ sở dữ liệu.
* **Tránh xung đột:** Giảm thiểu rủi ro khi triển khai các thay đổi lên môi trường sản xuất.
* **Dễ dàng khôi phục:** Có thể dễ dàng khôi phục cơ sở dữ liệu về một phiên bản trước đó nếu cần.
* **Tăng tính nhất quán:** Đảm bảo rằng cơ sở dữ liệu trên các môi trường khác nhau luôn đồng bộ.

**2. Cách hoạt động của Migrations**

* **Tạo một migration:** Khi tiến hành thực hiện một thay đổi đối với model (ví dụ: thêm một property), Entity Framework sẽ tạo ra một migration mới. Migration này chứa các lệnh SQL cần thiết để thực hiện thay đổi đó.
* **Áp dụng migration:** có thể áp dụng migration lên cơ sở dữ liệu bằng lệnh Update-Database. Lệnh này sẽ thực thi các script SQL trong migration để cập nhật cơ sở dữ liệu.
* **Quay lại phiên bản trước:** có thể quay lại một phiên bản trước đó của cơ sở dữ liệu bằng lệnh Update-Database -TargetMigration.

**3. Cài đặt và sử dụng (thông qua ví dụ)**

**Cài đặt Entity Framework Core:**

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

**Tạo DbContext:**

public class MyDbContext : DbContext

{

public MyDbContext(DbContextOptions<MyDbContext> options)

: base(options)

{

}

public DbSet<Product> Products

{ get; set; }

}

**Tạo model:**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

**Thêm lệnh vào Startup.cs:**

services.AddDbContext<MyDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

**Tạo initial migration:**

dotnet ef migrations add InitialCreate

**Áp dụng migration:**

dotnet ef database update

**Ví dụ: Thêm một cột mới vào bảng Product**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public bool IsActive { get; set;

} // Thêm cột mới

Sau khi cập nhật model, chạy lại lệnh “*dotnet ef migrations add AddIsActiveColumn*” và “*dotnet ef database update*” để cập nhật cơ sở dữ liệu.

**Lưu ý**

* **Code First:** Migrations thường được sử dụng với phương pháp Code First, tức là định nghĩa model trước, sau đó Entity Framework sẽ tạo ra cơ sở dữ liệu dựa trên model đó.
* **Database First và Model First:** có thể sử dụng Migrations với Database First và Model First, nhưng quá trình sẽ phức tạp hơn.
* **Fluent API:** có thể sử dụng Fluent API để cấu hình thêm các thuộc tính của model và cơ sở dữ liệu.
* **Reverse Engineer:** Nếu đã có một cơ sở dữ liệu hiện tại, có thể sử dụng lệnh Scaffold-DbContext để tạo ra các model và migration tương ứng.