

[콘텐츠로 건너뛰기](#)

나의 말:

ASP.NET CORE 8.0 기반의 REST API 를 이용한 서비스 프로그래밍을 만들거야.
MVC 패턴을 사용할거야

ChatGPT 의 말:

좋습니다! ASP.NET Core 8.0 기반의 REST API 서비스를 MVC 패턴으로 구성하는 것은 좋은 선택입니다. MVC(Model-View-Controller) 패턴은 구조화된 코드 작성을 도와주며 유지보수성과 확장성을 높여줍니다.

REST API 개발에서는 주로 "View"를 사용하지 않지만, ASP.NET Core MVC 구조를 기반으로 Controller 와 Model 을 이용하여 API 를 구성합니다. 뷰 대신 JSON 을 반환하게 되는 것이죠.

🔧 기본 구성

✓ 프로젝트 생성 (CLI 기준)

```
dotnet new webapi -n MyApiProject cd MyApiProject
```

📁 디렉터리 구조 (예시)

```
MyApiProject/  
├── Controllers/           ← Controller (API 엔드포인트)  
│   └── ProductsController.cs  
├── Models/               ← 데이터 모델  
│   └── Product.cs  
├── Services/             ← 비즈니스 로직  
│   └── ProductService.cs  
└── Repositories/         ← DB 접근 계층 (선택적)
```

```

|   └── ProductRepository.cs
|
|── Program.cs           ← 앱 시작점
|── appsettings.json     ← 설정 파일

```

📁 예제 코드

1. Model (Models/Product.cs)

```

namespace MyApiProject.Models { public class Product { public int Id { get;
set; } public string Name { get; set; } = string.Empty; public decimal
Price { get; set; } } }

```

2. Service (Services/ProductService.cs)

```

using MyApiProject.Models; namespace MyApiProject.Services { public class
ProductService { private static readonly List<Product> _products = new();
public List<Product> GetAll() => _products; public Product? GetById(int id)
=> _products.FirstOrDefault(p => p.Id == id); public void Add(Product
product) { product.Id = _products.Count + 1; _products.Add(product); }
public bool Delete(int id) { var product = GetById(id); if (product ==
null) return false; _products.Remove(product); return true; } } }

```

3. Controller (Controllers/ProductsController.cs)

```

using Microsoft.AspNetCore.Mvc; using MyApiProject.Models; using
MyApiProject.Services; namespace MyApiProject.Controllers { [ApiController]
[Route("api/[controller]")] public class ProductsController :
ControllerBase { private readonly ProductService _productService; public
ProductsController(ProductService productService) { _productService =
productService; } [HttpGet] public ActionResult<List<Product>> Get() =>
_productService.GetAll(); [HttpGet("{id}")] public ActionResult<Product>
Get(int id) { var product = _productService.GetById(id); return product ==
null ? NotFound() : Ok(product); } [HttpPost] public IActionResult
Post(Product product) { _productService.Add(product); return
CreatedAtAction(nameof(Get), new { id = product.Id }, product); }
[HttpDelete("{id}")] public IActionResult Delete(int id) { var result =
_productService.Delete(id); return result ? NoContent() : NotFound(); } } }

```

4. Program.cs 설정

```

using MyApiProject.Services; var builder =
WebApplication.CreateBuilder(args); builder.Services.AddControllers();
builder.Services.AddSingleton<ProductService>(); // 의존성 주입
builder.Services.AddEndpointsApiExplorer();

```

```
builder.Services.AddSwaggerGen(); var app = builder.Build();  
app.UseSwagger(); app.UseSwaggerUI(); app.UseHttpsRedirection();  
app.UseAuthorization(); app.MapControllers(); app.Run();
```

□ 테스트 방법

- Swagger UI: `https://localhost:port/swagger`
 - 또는 Postman 으로 GET, POST, DELETE 요청 테스트 가능
-

+ 확장 아이디어

- Repository 패턴 도입
 - Entity Framework Core 로 DB 연동
 - DTO, AutoMapper 사용
 - 인증/인가 (JWT)
 - 유닛 테스트, 통합 테스트
-

다음으로 도와줄 수 있는 것

- Entity Framework Core 연동 (SQL Server 등)
- JWT 인증 구현
- Docker 컨테이너화
- 유닛 테스트 작성
- CI/CD 파이프라인 구성

필요한 부분 있으면 알려줘. 각 단계를 하나씩 구현해도 좋아요.

나의 말:

내가 코드 달라고 할 때까지는 무엇을 해야되는지에 대해서 대화를 우선적으로 하자. 데이터베이스는 sQLite 를 사용할것이고 EF CORE 이용할 거야

ChatGPT 의 말:

좋아요! 코드 요청 전까지는 설계, 아키텍처, 그리고 구현 계획 위주로 이야기해보죠.

1. 프로젝트 설계 및 계획 단계

1) 요구사항 정리

- 어떤 기능들이 필요한지? (예: CRUD, 페이징, 검색 등)
- 데이터 모델의 주요 속성은 무엇인지
- 인증/인가가 필요한지 여부

2) 아키텍처 설계

- MVC 패턴을 어떻게 적용할지
- 서비스 레이어와 리포지토리 패턴 도입 여부
- Entity Framework Core 를 활용한 DB 연동 전략

3) 데이터베이스 설계

- SQLite 의 장단점 고려 (가볍고 설치 간편, 단 동시성 이슈 존재)
 - EF Core 모델링 (엔티티, 관계, 제약조건 등)
 - 마이그레이션 전략 (자동, 수동)
-

2. SQLite + EF Core 기반 API 개발 프로세스

1) 데이터 모델 정의

- EF Core 엔티티 클래스 설계 (POCO)
- DB 관계, 데이터 타입, 인덱스, 제약 조건 정의

2) DbContext 설정

- SQLite 연결 문자열 구성
- 마이그레이션 활성화 및 초기 DB 생성

3) 서비스 레이어 설계

- 비즈니스 로직과 데이터 접근 분리

- 트랜잭션 처리 방안 고려

4) 컨트롤러 설계

- RESTful API 경로 및 HTTP 메서드 정의
- DTO 를 통한 데이터 입출력 명세 (자동 매핑 가능)

5) 테스트 계획

- 단위 테스트, 통합 테스트 전략 수립
 - Postman, Swagger 활용 방안
-

3. 구체적으로 준비할 사항

- SQLite DB 파일 위치와 초기 스키마
 - EF Core 마이그레이션 커맨드 흐름 및 관리법
 - 앱 설정 파일(appsettings.json)에 DB 연결 문자열 설정
 - EF Core 의 비동기 메서드 활용 방침 (async/await)
 - 개발 환경 (IDE, CLI, 도구)
-

질문

- 데이터 모델은 어느 정도 구체화 되었나요?
 - 기본 CRUD 이외에 추가로 필요한 기능이나 규칙 있나요?
 - 인증이나 권한 관련 계획은 있나요? (JWT 등)
 - 프로젝트 규모(몇 개 API, 동시 사용자 수 예상 등)는 어떻게 되나요?
-

원하는 부분부터 차근차근 같이 짚어가면서 준비해요!

나의 말:

만들 내용은 Todo 리스트를 만들려고 해. 속성은 id, 작성자, 내용, 작성일자로 구성되어 있어. 환경설정과 ef CORE 사용은 일반적인 사용법을 이용할 거야

ChatGPT의 말:

아, Todo 리스트라면 굉장히 깔끔하고 기본적인 REST API 예제로 딱 좋죠!

Todo 리스트 서비스 설계 요약

1. 데이터 모델

- Id (int, PK)
 - Author (string) - 작성자
 - Content (string) - 할 일 내용
 - CreatedAt (DateTime) - 작성일자
-

2. 주요 기능 (API 엔드포인트)

- 전체 **Todo** 조회 (GET /api/todos)
 - 특정 **Todo** 조회 (GET /api/todos/{id})
 - 새 **Todo** 추가 (POST /api/todos)
 - **Todo** 수정 (PUT /api/todos/{id})
 - **Todo** 삭제 (DELETE /api/todos/{id})
-

3. EF Core + SQLite 사용 방향

- DbContext 클래스 생성 (예: TodoContext)
- DbSet<Todo> 속성 정의
- SQLite 연결 문자열 appsettings.json 에 설정
- 마이그레이션으로 DB 스키마 생성 및 관리
- 기본 CRUD 작업은 EF Core 비동기 메서드 활용 (ex. FindAsync, AddAsync, SaveChangesAsync)

4. 구조

- **Models/ToDo.cs** : 데이터 모델 정의
 - **Data/ToDoContext.cs** : EF Core DbContext
 - **Services/ToDoService.cs** (선택적) : 비즈니스 로직 분리
 - **Controllers/ToDoController.cs** : API 엔드포인트 구현
-

5. 고려 사항

- 날짜 자동 생성 처리 (CreatedAt 초기값 설정)
 - 입력 데이터 검증 (Model Validation)
 - 예외 처리 (NotFound, BadRequest 등 HTTP 상태 코드)
 - 비동기 처리 원칙 준수
-

다음 단계

1. 프로젝트 생성과 EF Core, SQLite 패키지 설치
 2. Todo 모델 클래스 설계
 3. **ToDoContext** DbContext 설정 및 연결 문자열 구성
 4. 마이그레이션 및 DB 생성 준비
-

필요하면 각 단계별 세부사항이나 코드 예제를 요청해 주세요!
먼저 어떤 부분을 진행해볼까요?

나의 말:

패키지 설치 명령어를 통해 터미널에서 진행하고 싶어

ChatGPT의 말:

좋아요! 터미널에서 명령어로 패키지를 설치하는 게 가장 빠르고 깔끔하죠.

1. ASP.NET Core 8.0 Web API 프로젝트 생성

```
dotnet new webapi -n TodoApi cd TodoApi
```

2. EF Core 및 SQLite 패키지 설치

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite dotnet add package Microsoft.EntityFrameworkCore.Design
```

- Microsoft.EntityFrameworkCore.Sqlite: SQLite DB 사용을 위한 EF Core 프로바이더
 - Microsoft.EntityFrameworkCore.Design: 마이그레이션 등 디자인 타임 도구 지원
-

3. (선택) EF Core Tools 설치 — 마이그레이션 명령어를 사용하려면

```
dotnet tool install --global dotnet-ef
```

설치되어 있다면 생략 가능하고, 설치 확인은:

```
dotnet ef --version
```

4. 마이그레이션 및 DB 업데이트 명령어 예시

- 마이그레이션 생성:

```
dotnet ef migrations add InitialCreate
```


- 데이터베이스 업데이트 (마이그레이션 반영):

```
dotnet ef database update
```

필요하면 각 명령어 사용법과 순서도 자세히 알려드릴게요!

다음 단계로 진행할 준비되면 말씀해 주세요!

나의 말:

```
var builder = WebApplication.CreateBuilder(args); // Add services to the container.
builder.Services.AddControllersWithViews(); var app = builder.Build(); // Configure
the HTTP request pipeline. if (!app.Environment.IsDevelopment())
{ app.UseExceptionHandler("/Home/Error"); // The default HSTS value is 30 days. You
may want to change this for production scenarios, see https://aka.ms/aspnetcore-
hsts. app.UseHsts(); } app.UseHttpsRedirection(); app.UseStaticFiles();
app.UseRouting(); app.UseAuthorization(); app.MapControllerRoute( name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}"); app.Run(); EF Core 를 이용해
app.db 라는 sqlite 데이터베이스를 사용하는데 필요한 정보를 알려줘
```

ChatGPT의 말:

지금 제공해주신 코드는 ASP.NET Core MVC 템플릿 기본 구조입니다.

여기에 **EF Core**를 이용해서 `app.db`라는 **SQLite** 데이터베이스를 연결하려면 다음과 같은 설정과 구성이 필요합니다.

1. appsettings.json 에 **SQLite** 연결 문자열 추가

```
{ "ConnectionStrings": { "DefaultConnection": "Data Source=app.db" },
// ... 다른 설정들 }
```

2. EF Core 관련 패키지 설치 (터미널에서)

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite dotnet add package
Microsoft.EntityFrameworkCore.Design
```

3. DbContext 클래스 작성 (예: Data/DbContext.cs)

```
using Microsoft.EntityFrameworkCore; using YourNamespace.Models; // Todo
모델 네임스페이스 namespace YourNamespace.Data { public class TodoContext :
DbContext { public TodoContext(DbContextOptions<TodoContext> options) :
base(options) { } public DbSet<Todo> Todos { get; set; } } }
```

4. Program.cs 에 DbContext 등록 및 연결 문자열 사용

```
var builder = WebApplication.CreateBuilder(args); // SQLite 연결 문자열
읽기 var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection"); //
DbContext 등록 builder.Services.AddDbContext<TodoContext>(options =>
options.UseSqlite(connectionString));
builder.Services.AddControllersWithViews(); var app = builder.Build(); //
나머지 미들웨어 및 라우팅 구성 (생략) app.Run();
```

5. Todo 모델 예시 (Models/Todo.cs)

```
using System; namespace YourNamespace.Models { public class Todo { public
int Id { get; set; } public string Author { get; set; } = string.Empty;
public string Content { get; set; } = string.Empty; public DateTime
CreatedAt { get; set; } } }
```

6. 마이그레이션 및 DB 생성

터미널에서 프로젝트 폴더 내에서 아래 명령어 실행:

dotnet ef migrations add InitialCreate dotnet ef database update

요약

| 단계 | 작업 내용 |
|-----------------|--|
| 1. 연결 문자열 설정 | appsettings.json 에 Data Source=app.db 추가 |
| 2. 패키지 설치 | EF Core SQLite, Design 패키지 추가 |
| 3. DbContext 작성 | SQLite DB 와 매핑하는 컨텍스트 생성 |
| 4. DI 설정 | Program.cs 에서 DbContext 등록 및 연결 |
| 5. 모델 정의 | Todo 모델 클래스 작성 |
| 6. 마이그레이션 | DB 스키마 생성 및 적용 |

필요하면 각 단계별 자세한 예제도 준비해드릴게요!
궁금한 점 있나요?

나의 말:

REst api 컨트롤러도 등록해야 하지 않아

ChatGPT 의 말:

네, 맞아요! REST API 를 만들려면 **컨트롤러도 등록**해야 합니다. 다만, 기본적으로 ASP.NET Core 에서는 `AddControllers()`나 `AddControllersWithViews()`를 호출하면 컨트롤러들이 자동으로 스캔되고 등록됩니다.

1. Program.cs 서비스 등록 시

```
builder.Services.AddControllers(); // REST API 용 (View 없음)
```

또는

```
builder.Services.AddControllersWithViews(); // MVC + View 포함
```

보통 REST API 만 할 경우 `AddControllers()`를 씁니다.

2. 컨트롤러 클래스 작성 예

```
using Microsoft.AspNetCore.Mvc; using YourNamespace.Data; using
YourNamespace.Models; using Microsoft.EntityFrameworkCore; namespace
YourNamespace.Controllers { [ApiController] [Route("api/[controller]")]
public class TodosController : ControllerBase { private readonly
TodoContext _context; public TodosController(TodoContext context)
{ _context = context; } [HttpGet] public async
Task<ActionResult<IEnumerable<Todo>>> GetTodos() { return await
_context.Todos.ToListAsync(); } [HttpGet("{id}")] public async
Task<ActionResult<Todo>> GetTodo(int id) { var todo = await
_context.Todos.FindAsync(id); if (todo == null) return NotFound(); return
todo; } [HttpPost] public async Task<ActionResult<Todo>> PostTodo(Todo
todo) { todo.CreatedAt = DateTime.UtcNow; _context.Todos.Add(todo); await
_context.SaveChangesAsync(); return CreatedAtAction(nameof(GetTodo), new
{ id = todo.Id }, todo); } [HttpPut("{id}")] public async
Task<ActionResult> PutTodo(int id, Todo todo) { if (id != todo.Id) return
BadRequest(); _context.Entry(todo).State = EntityState.Modified; try
{ await _context.SaveChangesAsync(); } catch (DbUpdateConcurrencyException)
{ if (!_context.Todos.Any(e => e.Id == id)) return NotFound(); else
throw; } return NoContent(); } [HttpDelete("{id}")] public async
Task<ActionResult> DeleteTodo(int id) { var todo = await
_context.Todos.FindAsync(id); if (todo == null) return NotFound();
_context.Todos.Remove(todo); await _context.SaveChangesAsync(); return
NoContent(); } } }
```

3. 라우팅 설정

- 기본적으로 `MapControllers()` 호출만 하면 `[ApiController]`와 `[Route]` 어노테이션을 읽어 라우팅됩니다.
`app.MapControllers();`
-

정리

- `builder.Services.AddControllers();`로 API 컨트롤러 등록
 - `app.MapControllers();`로 라우팅 활성화
 - `[ApiController]` + `[Route("api/[controller]")]` 어노테이션으로 REST API 엔드포인트 구현
-

필요하면 컨트롤러 전체 코드를 더 자세히 설명해 드릴게요!

첨부

검색

학습하기

음성

ChatGPT 는 실수를 할 수 있습니다. 중요한 정보는 재차 확인하세요.