**ASP.NET Core 3.0 기반의**

**Web API 개발 샘플 가이드**

|  |  |
| --- | --- |
| **작성자** | **이우형** |
| **법인** | **아이라이브** |
| **부서** |  |
| **작성일** | **2020.05.07** |
| **버전** | **1.0** |

**개정이력**

| 버전 | 작성일 | 변경내용 | 작성자 | 승인자 |
| --- | --- | --- | --- | --- |
| 1.0 | 2020.05.07 | 초안 작성 | 이우형 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

내용

[**1.** **개요** 4](#_Toc40370388)

[1.1 웹 프로젝트 만들기 4](#_Toc40370389)

[1.2 소스설명 7](#_Toc40370390)

[1.2.1 appsettings.json 7](#_Toc40370391)

[1.2.2 TbTestApi.cs 7](#_Toc40370392)

[1.2.3 TbTestApiContext.cs 8](#_Toc40370393)

[1.2.4 Startup.cs 9](#_Toc40370394)

[1.2.5 TbTestApisController.cs 10](#_Toc40370395)

[1.2.6 ITestApiService.cs 12](#_Toc40370396)

[1.2.7 TestApiService.cs 12](#_Toc40370397)

[1.2.8 index.html 13](#_Toc40370398)

[1.2.9 functions.js 13](#_Toc40370399)

[2. NuGet 설치 정보 14](#_Toc40370400)

# **개요**

ASP.NET Core는 C#을 사용하여 Web API라고도 하는 RESTful 서비스 만들기를 지원합니다.

요청을 처리하기 위해 Web API는 컨트롤러를 사용합니다. (ControllBase에서 파생되는 클래스)

이 문서는 ASP.NET Core를 사용하여 요청을 처리하는 서비스와 컨트롤러의 사용방법 및 Web API 빌드 작업의 기본사항, Entity Framework를 활용한 CRUD 샘플 가이드를 목적으로 작성하였습니다.

|  |  |  |  |
| --- | --- | --- | --- |
| APIAPI | 설명Description | 요청 본문Request body | 응답 본문Response body |
| GET /api/TbTestApisGET /api/TodoItems | 리스트 모두 가져오기Get all to-do items | 없음None | 항목의 배열Array of to-do items |
| GET /api/TbTestApis/{id}GET /api/TodoItems/{id} | ID로 항목 가져오기Get an item by ID | 없음None | 항목To-do item |
| POST /api/TbTestApisPOST /api/TodoItems | 새 항목 추가Add a new item | 항목To-do item | 항목To-do item |
| PUT /api/TbTestApis/{id}PUT /api/TodoItems/{id} | 기존 항목 업데이트  Update an existing item | 항목To-do item | 없음None |
| DELETE /api/TbTestApis/{id}    DELETE /api/TodoItems/{id} | 항목 삭제    Delete an item | 없음None | 없음 |

1. ****웹 프로젝트 만들기****

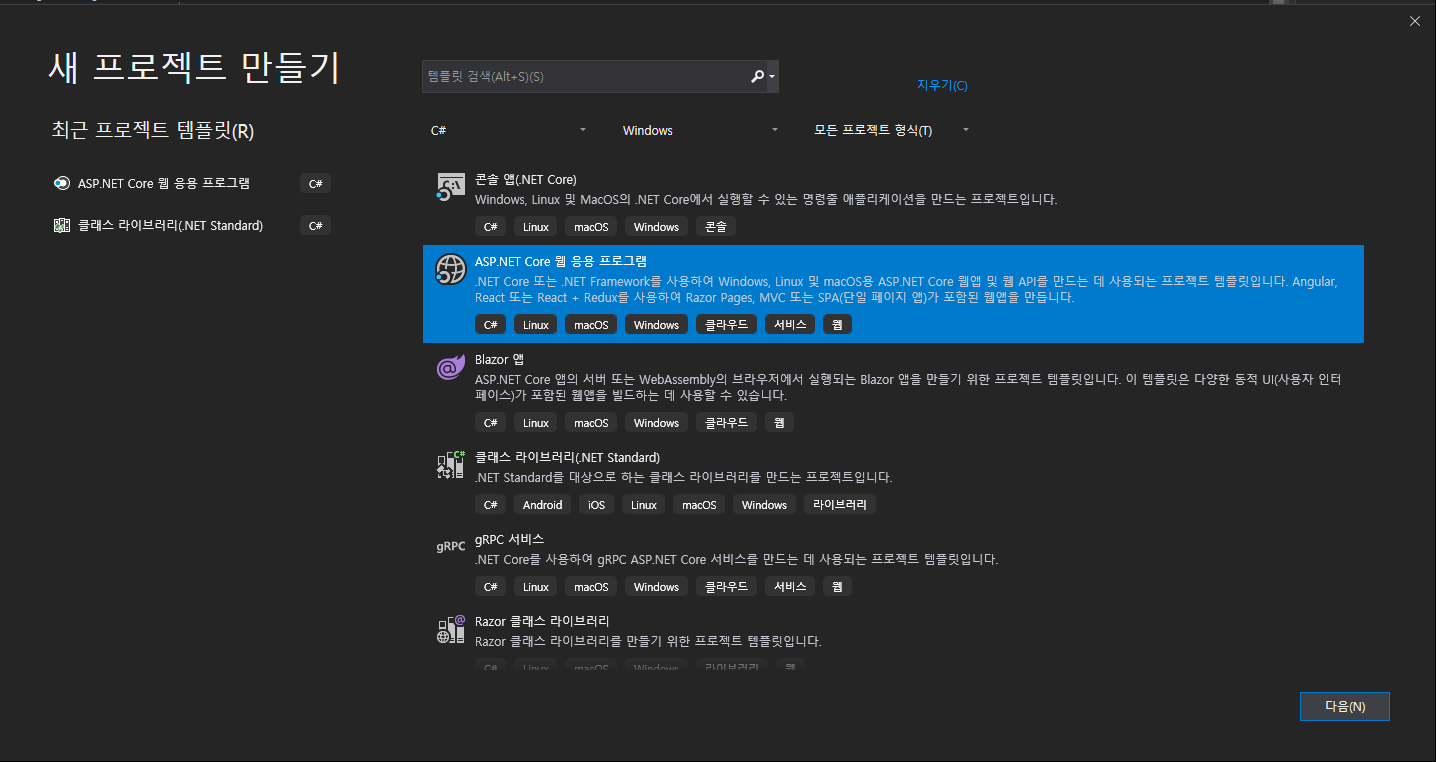
A. Visual Studio의 파일 메뉴에서 새로만들기 > 프로젝트를 선택 합니다.

B. "ASP.NET Core 웹 응용프로그램" 템플릿을 선택 합니다.

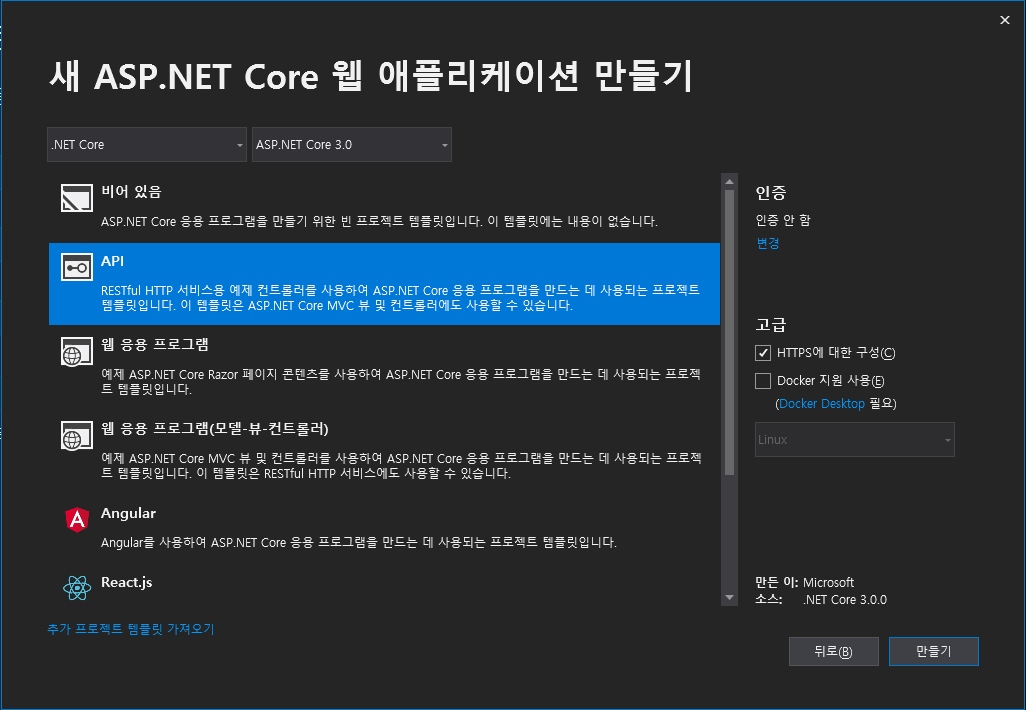
C. 프로젝트 명을 지정하고 만들기를 클릭 합니다.

D. "새 ASP.NET Core 웹 애플리케이션 만들기" 팝업에서

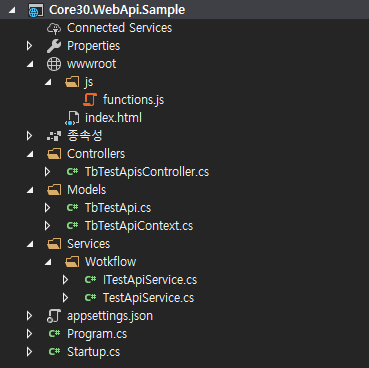
".NET.Core" 및 "ASP.NET Core 3.0"이 선택되었는지 확인 후 만들기를 클릭 합니다.



"ASP.NET Core 웹 응용프로그램" 템플릿을 선택



".NET.Core" 및 "ASP.NET Core 3.0"이 선택되었는지 확인.



해당 프로젝트 관련 생성 된 파일 설명

|  |  |
| --- | --- |
| 프로젝트 구성 | 설명 |
| Core30.WebApi.Sample | 프로젝트 명 |
| Function.js | API 호출 스크립트 |
| index.html | 기본 CRUD가 적용된 웹 페이지 |
| TbTestApisController.cs | REST API로 전달받은 요청을 라우팅을 통해 매핑하고 처리하는 클래스 |
| TbTestApi.cs | 데이터베이스에 저장된 데이터를 관리하기 위한 클래스 |
| TbTestApiContext.cs | Dbcontext에서 파생되어 모델에 포함할 Entity를 지정하는 클래스 |
| ITestApiService.cs | (DI)종속성 주입 인스턴스를 생성하는 클래스 |
| TestApiService.cs | 인스턴스에 종속되고 서비스를 구현하는 클래스 |
| appsettings.json | 환경 구성 파일 |
| Program.cs | 호스팅 환경 구성 및 실행 |
| Startup.cs | 호스팅 환경 구성 및 파이프라인을 정의하는 클래스 |

1. ****소스설명****
   * 1. appsettings.json

{

"ConnectionStrings": {

"EKP\_WORKFLOW\_Connection": "Server=10.2.211.70,3920;Database=EKP\_WORKFLOW;

User Id=app\_svc;Password=svc3920!;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

프로젝트를 구동하기 위한 환경 설정 값으로 DB 연결 및 기타 서비스에 대한 환경 설정을 할 수 있습니다. 해당 파일은 IConfiguration을 통하여 활용이 가능 합니다.

* + 1. TbTestApi.cs

public class TbTestApi

{

public int Seq { get; set; }

public string Subject { get; set; }

public string Content { get; set; }

public DateTime? GreateDt { get; set; }

}

모델 클래스는 EF Core에 대한 어떠한 종속성도 없는 POCO(Plain Old CLR Object)클래스 입니다.

이 클래스는 데이터베이스에 저장 될 데이터의 속성만을 정의 합니다.

* + 1. TbTestApiContext.cs

DbContext는 데이터 모델에 맞게 Entitiy Framework 기능을 (읽기, 생성, 수정, 삭제) 조정하는 주 클래스 입니다.

public class TbTestApiContext : DbContext

{

//DbContext를 상속받고, base를 통해서 생성자의 부모 클래스 호출을 위해 base를 사용한다.

public TbTestApiContext(DbContextOptions<TbTestApiContext> options)

: base(options)

{

}

//DB의 테이블에 해당하는 것으로 필요한 만큼 DbSet을 지정할 수 있다.

//(테이블의 컬럼 정의는 DbSet<T>의 "T" 에 정의 됨)

public DbSet<TbTestApi> TbTestApis { get; set; }

//Context에서 OnModelCreating 메서드를 재정의하고 ModelBuilder API를 사용하여 모델을 구성할 수 있습니다.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Fluent API(흐름 API)에 의한 Entity 구성 (Fluent API란 함수들을 작성하고 나면, 마치 그 문장이 영어 문장처럼 읽히는 API)

modelBuilder.Entity<TbTestApi>(entity =>

{

// Primary key 적용

entity.HasKey(e => e.Seq).HasName("PK\_\_TB\_TEST\_API");

// entity가 mapping 해야 할 테이블명 지정

entity.ToTable("TB\_TEST\_API");

// Column 및 속성 구성

entity.Property<int>(e => e.Seq)

.HasColumnName("SEQ");

entity.Property(e => e.Subject)

.HasColumnName("SUBJECT")

.HasMaxLength(100);

entity.Property(e => e.Content)

.HasColumnName("CONTENT")

.HasMaxLength(1000);

entity.Property(e => e.GreateDt)

.HasColumnName("CREATE\_DT")

.HasColumnType("datetime");

});

}

}

Entity의 집합인 TbTestApi 모델에 대한 Dbset< TbTestApi > 속성을 만들어야 하며

이 가이드의 샘플 소스에서는 Seq를 기본키로 구성합니다.

* + 1. Startup.cs

Startup는 반드시 Configure()가 있어야 하며 ConfigureServices()메서드를 가질 수 있습니다. Configure() 메서드는 Pipeline을 설정하는 역할을 하며 ConfigureServices()메서드는 웹 프로그램에서 어떤 Framework Service를 사용할지를 지정하거나 개발자가 지정한 형식에 대한 Dependancy Injection를 정의하는 역활을 합니다.

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<TbTestApiContext>(options => options.UseSqlServer(Configuration.GetConnectionString("EKP\_WORKFLOW\_Connection")));

services.AddScoped<ITestApiService, TestApiService>();

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage(); // 개발환경에서만 예외처리 미들웨어 사용.

}

app.UseDefaultFiles();//default.htm, default.html, index.htm, index.html등의 파일 검색

app.UseStaticFiles();//wwwroot를 가리키는 파일을 기본값으로 설정하고 정적 파일을 제공

app.UseHttpsRedirection();// https Redirect 미들웨어는 HTTP 요청을 HTTPS로 Redirect함

app.UseRouting();// 들어오는 HTTP 요청을 일치시켜 앱의 실행 가능 EndPoint로 보낸다.

app.UseAuthorization();// 사용자에게 보안 리소스에 액세스할 수 있는 권한을 부여

app.UseEndpoints(endpoints => // 요청 파이프라인에 MapControllers 엔드포인트 추가.

{

endpoints.MapControllers(); // 경로를 지정하지 않고 작업의 끝점을 추가합니다.

});

}

※Dependancy Injection : 각 클래스간의 의존관계 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것을 말함.

샘플에서는 AddDbContext<Tcontext>를 사용하여 데이터베이스와 연결 가능한 DbContext 서비스를 컨테이너에 추가하고, 개발자가 만든 서비스를 Dependancy Injection로 정의하기 위해 AddScoped를 사용하여 컨테이너 내부에서의 수명을 결정한 후 추가하고 있습니다.

그리고 services.AddControllers()는 RestAPI로 제공 될 Controllers를 구성 합니다.

* + 1. TbTestApisController.cs

[ApiController]

[Route("api/[controller]")]

public class TbTestApisController : ControllerBase

{

private readonly ITestApiService \_testApiService;

public TbTestApisController(ITestApiService testApiService)

{

\_testApiService = testApiService;

}

// GET: api/TbTestApis

[HttpGet]

public async Task<ActionResult<IEnumerable<TbTestApi>>> GetTbTestApis()

{

var result = await \_testApiService.GetTestApiList();

return result;

}

// GET: api/TbTestApis/5

[HttpGet("{id}")]

public async Task<ActionResult<TbTestApi>> GetTbTestApi(int id)

{

var result = await \_testApiService.GetTestApiItem(id);

if (result == null) {

return NotFound();

}

return result;

}

// PUT: api/TbTestApis/5

[HttpPut("{id}")]

public async Task<IActionResult> PutTbTestApi(int id, TbTestApi tbTestApi)

{

if (id != tbTestApi.Seq) {

return BadRequest();

}

try {

await \_testApiService.PutTestApiItem(id, tbTestApi);

}

catch (DbUpdateConcurrencyException) {

if (!\_testApiService.GetTestApiExists(id)) {

return NotFound();

} else{

throw;

}

}

return NoContent();

}

// POST: api/TbTestApis

// To protect from overposting attacks, please enable the specific properties you w

// more details see https://aka.ms/RazorPagesCRUD.

[HttpPost]

public async Task<ActionResult<TbTestApi>> PostTbTestApi(TbTestApi tbTestApi)

{

await \_testApiService.PostTestApi(tbTestApi);

// 201(요청받은 작업을 성공함)

return CreatedAtAction("GetTbTestApi", new { id = tbTestApi.Seq }, tbTestApi);

}

// DELETE: api/TbTestApis/5

[HttpDelete("{id}")]

public async Task<ActionResult<TbTestApi>> DeleteTbTestApi(int id)

{

var tbTestApi = await \_testApiService.DeleteTestApiItem(id);

if (tbTestApi == null)

{

return NotFound();

}

return tbTestApi;

}

}

// POST: api/TbTestApis

[HttpPost]

public async Task<ActionResult<TbTestApi>> PostTbTestApi(TbTestApi tbTestApi)

{

await \_testApiService.PostTestApi(tbTestApi);

// 201(요청받은 작업을 성공함)

return CreatedAtAction("GetTbTestApi", new { id = tbTestApi.Seq }, tbTestApi);

}

// DELETE: api/TbTestApis/5

[HttpDelete("{id}")]

public async Task<ActionResult<TbTestApi>> DeleteTbTestApi(int id)

{

var tbTestApi = await \_testApiService.DeleteTestApiItem(id);

if (tbTestApi == null)

{

return NotFound();

}

return tbTestApi;

}

}

Asp.net Core는 Web API라고하는 RESTful 서비스 만들기(스캐폴드)를 지원합니다.

요청을 처리하기 위해 Web API는 Controller를 사용하고 ApiController의 특성을 컨트롤러

클래스에 적용하여 API 동작을 사용할 수 있습니다.

Route는 컨트롤러를 API의 접속 URL로 지정하여 매핑 할 수 있습니다.

샘플에서는Startup.cs에UseEndpoints.MapController() 를 구성하여 라우트 된 컨트롤러를

매핑하고 “api/[controller]” 방식으로 Service를 호출 할 수 있습니다.

* + 1. ITestApiService.cs

public interface ITestApiService

{

Task<List<TbTestApi>> GetTestApiList();

Task<TbTestApi> GetTestApiItem(int id);

Task<int> PutTestApiItem(int id, TbTestApi tbTestApi);

bool GetTestApiExists(int id);

Task<int> PostTestApi(TbTestApi tbTestApi);

Task<TbTestApi> DeleteTestApiItem(int id);

}

Service는 인터페이스를 사용하여 직접 구현하지 않고 단지 정의만을 가지며, Controller에서 생성자를 통해 명시적으로 종속성을 주입(DI)하고 사용할 수 있습니다.

* + 1. TestApiService.cs

public class TestApiService : ITestApiService

{

private readonly TbTestApiContext \_context;

public TestApiService(TbTestApiContext context)

{

\_context = context;

}

public async Task<List<TbTestApi>> GetTestApiList()

{

// ToListAsync는 Linq와 비슷한 비동기 메서드이며 DB에서 쿼리를 실행하지 않기 때문에

// Where나 OrderBy 등 연산자를 사용할 수 없음 (ToList 의 비동기 버전)

// : \_context(TbTestApiContext)의 TbTestApis 속성을 읽어 TbTestApi 엔터티 집합에서

// 목록을 가져옴

return await \_context.TbTestApis.ToListAsync();

}

public async Task<TbTestApi> GetTestApiItem(int id)

{

// HttpGet Edit 매서드는 ID 매개변수를 받아 EF의 FindAsync를 사용하여 TbTestApi를

// 검색하고 선택된 레코드를 Edit 보기에 반환한다

return await \_context.TbTestApis.FindAsync(id);

}

public async Task<int> PutTestApiItem(int id, TbTestApi tbTestApi)

{

// DB에 존재하지만 변경이 이루어진 Entity(tbTestApi)가 있는 경우 \_context에

// Entity를 첨부하고 수정 할 수 있음

\_context.Entry(tbTestApi).State = EntityState.Modified;

// 비동기 저장

return await \_context.SaveChangesAsync();

}

public bool GetTestApiExists(int id)

{

// Any는 TbTestApis의 시퀀스(Seq)에 요소(id)가 존재하는지 확인

return \_context.TbTestApis.Any(e => e.Seq == id);

}

public async Task<int> PostTestApi(TbTestApi tbTestApi)

{

// Entity(tbTestApi) 새 인스턴스를 추가.

\_context.TbTestApis.Add(tbTestApi);

return await \_context.SaveChangesAsync();

}

public async Task<TbTestApi> DeleteTestApiItem(int id)

{

var tbTestApi = await \_context.TbTestApis.FindAsync(id);

if (tbTestApi != null)

{

// 지정된 Entity 삭제

\_context.TbTestApis.Remove(tbTestApi);

await \_context.SaveChangesAsync();

}

return tbTestApi;

}

}

Service는 파생 클래스를 통하여 인터페이스를 구현 합니다.

* + 1. index.html

- CRUD와 관련된 HTML 레이아웃 입니다.

* + 1. functions.js

- Controller의 API를 호출하기 위한 스크립트 입니다.

1. ****NuGet 설치 정보****

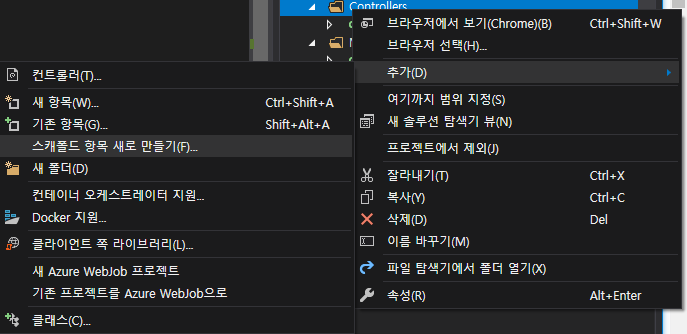
.NET Core 3.0 개발 플랫폼 기반의 Visual Studio Professional 2019를 기본 IDE로 합니다.

아래의 표는 Web API샘플을 개발하기 위한 NuGet 솔루션 패키지 목록입니다.

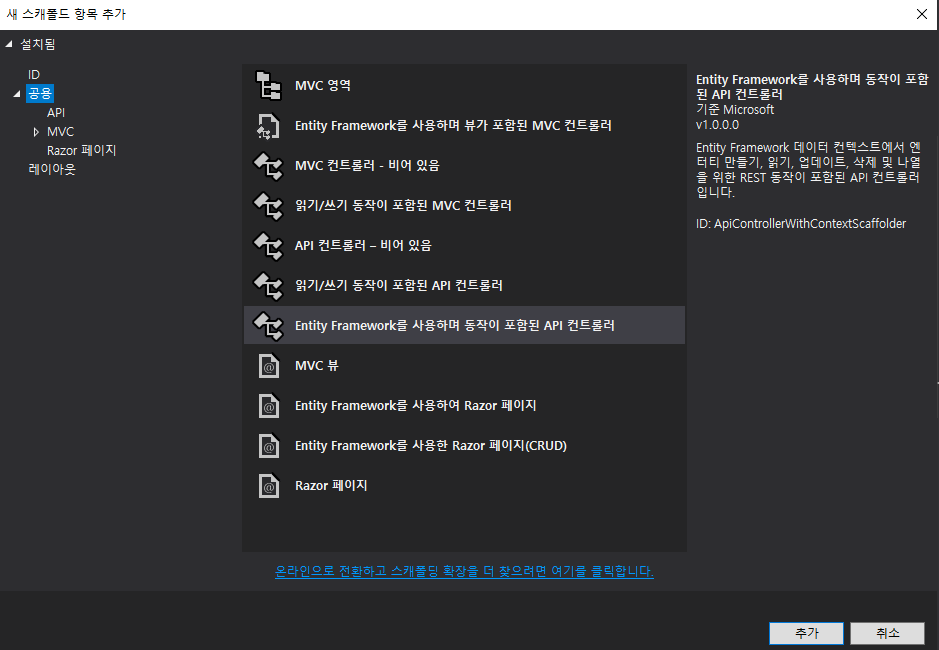
|  |  |  |
| --- | --- | --- |
| 설치 솔루션 패키지 | 권장 버전 | 비고 |
| Microsoft.EntityFrameworkCore | 3.0 |  |
| Microsoft.EntityFrameworkCore.SqlServer | 3.0 |  |

1. ****Controller 스캐폴드****

**A. Controllers폴더를 마우스 우클릭 하여 “추가>스캐폴드 항목 새로 만들기(F)”를 선택 합니다.**

****

**B.** “**Entity Framework를 사용하며 동작이 포함된 API 컨트롤러”**를 선택하고 **추가**를 선택합니다.

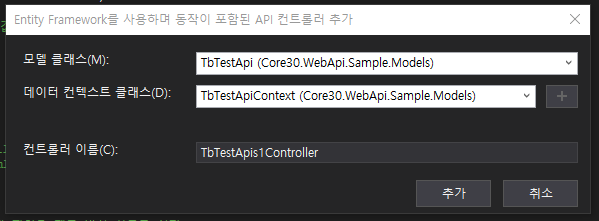
****

**C. Entity Framework를 사용하며 동작이 포함된 API 컨트롤러 추가** 대화 상자에서

- **모델 클래스**에서 **TbTestApi (Core30.WebApi.Sample.Models)**을 선택.

**- 데이터 컨텍스트 클래스**에서 **“TbTestApiContext (Core30.WebApi.Sample.Models)”** 선택.

**- 추가**를 선택.

****