# Guide: .NET 10 Minimal API with Camunda BPM  Grist CRUD

Your Name

February 15, 2026

## 1 Introduction

This document describes how to build a **.NET 10 Minimal API** that:

- Supports full CRUD operations (GET, POST, PUT, DELETE) on items
- Uses **Grist** as the data store (via REST API)
- Uses **Camunda BPM** to orchestrate workflows that call these CRUD API endpoints
- Implements Dependency Injection, DTOs, AutoMapper, FluentValidation, and structured API responses

The key idea is that **Camunda orchestrates the workflow** and interacts with your API, which then performs CRUD against Grist.

—

## 2 Prerequisites

- .NET 10 SDK installed
- A running Grist workspace and API key
- Camunda BPM server (docker recommended)
- Camunda Modeler (for BPMN)

—

## 3 Project Creation

Create a new .NET Minimal API:

```
dotnet new web -o WorkflowApi
cd WorkflowApi
```

Add packages:

```
dotnet add package AutoMapper
dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
dotnet add package FluentValidation
dotnet add package FluentValidation.DependencyInjectionExtensions
```

—

## 4 Architecture Overview

### 4.1 Grist as Database

Grist provides a REST API for storing records. Your .NET API will call Grist endpoints like:

- `POST /api/docs/{docId}/tables/Items/records` — create
- `GET /api/docs/{docId}/tables/Items/records` — read
- `POST /.../delete` — delete

Authentication uses an API key in headers.

### 4.2 Camunda BPM Orchestration

You model BPMN workflows where tasks use Camunda's REST connector to call your API's endpoints (GET/POST/PUT/DELETE). For example:

- A process step calls `POST /items` to create a record
- Another step calls `PUT /items/{id}` to update
- A final step calls `DELETE /items/{id}` to delete

Camunda's REST connector lets you set method, URL, headers, and payload for HTTP calls directly in the BPMN task properties. :contentReference[oaicite:0]index=0

—

## 5 Domain Models and DTOs

### 5.1 Entity

```
namespace WorkflowApi.Entities;
public class Item
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
```

### 5.2 DTOs

```
// CreateItemRequest.cs
public class CreateItemRequest
{
    public string Name { get; set; }
    public string Description { get; set; }
}

// UpdateItemRequest.cs
public class UpdateItemRequest
{
    public string Name { get; set; }
    public string Description { get; set; }
}
```

```
// ItemResponse.cs
public class ItemResponse
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
```

—

## 6 AutoMapper Setup

```
// ItemProfile.cs
using AutoMapper;

public class ItemProfile : Profile
{
    public ItemProfile()
    {
        CreateMap<CreateItemRequest, Item>();
        CreateMap<UpdateItemRequest, Item>();
        CreateMap<Item, ItemResponse>();
    }
}
```

—

## 7 FluentValidation

```
// CreateItemValidator.cs
using FluentValidation;

public class CreateItemValidator : AbstractValidator<CreateItemRequest>
{
    public CreateItemValidator()
    {
        RuleFor(x => x.Name).NotEmpty();
    }
}

public class UpdateItemValidator : AbstractValidator<UpdateItemRequest>
{
    public UpdateItemValidator()
    {
        RuleFor(x => x.Name).NotEmpty();
    }
}
```

—

## 8 API Response Wrapper

```
public class ApiResponse<T>
{
    public bool Success { get; set; }
    public T? Data { get; set; }
    public string[] Errors { get; set; } = Array.Empty<string>();
}
```

—

## 9 Service Layer (Grist Integration)

```
// IItemService.cs
public interface IItemService
{
    Task<Item?> GetByIdAsync(Guid id);
    Task<IEnumerable<Item>> GetAllAsync();
    Task<Item> CreateAsync(Item entity);
    Task<Item?> UpdateAsync(Guid id, Item entity);
    Task<bool> DeleteAsync(Guid id);
}

// ItemService.cs
public class ItemService : IItemService
{
    private readonly HttpClient _client;
    private readonly string _docId = "YOUR_GRIST_DOC_ID";

    public ItemService(HttpClient client, IConfiguration config)
    {
        _client = client;
        _client.DefaultRequestHeaders.Authorization =
            new AuthenticationHeaderValue("Bearer", config["GristApiKey"]);
    }

    public async Task<IEnumerable<Item>> GetAllAsync()
    {
        var response = await _client.GetStringAsync(
            $"https://docs.getgrist.com/api/docs/{_docId}/tables/Items/records");
        return JsonSerializer.Deserialize<IEnumerable<Item>>(response)!;
    }

    public async Task<Item?> GetByIdAsync(Guid id) =>
        (await GetAllAsync()).FirstOrDefault(i => i.Id == id);

    public async Task<Item> CreateAsync(Item entity)
    {
        var body = JsonSerializer.Serialize(entity);
        var res = await _client.PostAsync(
            $"https://docs.getgrist.com/api/docs/{_docId}/tables/Items/records",
            new StringContent(body, Encoding.UTF8, "application/json"));
        res.EnsureSuccessStatusCode();
        return JsonSerializer.Deserialize<Item>(await res.Content.
           ReadAsStringAsync())!;
    }

    public async Task<Item?> UpdateAsync(Guid id, Item entity)
    {
        // Grist update patterns may require read-delete-create or PATCH if
           supported
        return entity;
    }
    public async Task<bool> DeleteAsync(Guid id)
    {
        var list = JsonSerializer.Serialize(new[]{ id });
        var res = await _client.PostAsync(
            $"https://docs.getgrist.com/api/docs/{_docId}/tables/Items/records/
                delete",
            new StringContent(list, Encoding.UTF8, "application/json"));
        return res.IsSuccessStatusCode;
    }
}
```

—

## 10 Minimal API Endpoints

```
using AutoMapper;
using FluentValidation;
using WorkflowApi.DTOs;
using WorkflowApi.Entities;

// Configure DI
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddValidatorsFromAssemblyContaining<CreateItemValidator>();
builder.Services.AddHttpClient();
builder.Services.AddScoped<IItemService, ItemService>();

var app = builder.Build();

// GET all
app.MapGet("/items", async (IItemService svc, IMapper mapper) =>
{
    var list = await svc.GetAllAsync();
    return Results.Ok(new ApiResponse<IEnumerable<ItemResponse>>
    {
        Success = true,
        Data = list.Select(i => mapper.Map<ItemResponse>(i))
    });
});

// GET by ID
app.MapGet("/items/{id}", async (Guid id, IItemService svc, IMapper mapper) =>
{
    var item = await svc.GetByIdAsync(id);
    if (item == null)
        return Results.NotFound(new ApiResponse<string> { Success = false, Errors
            = new[]{"Not found"} });
    return Results.Ok(new ApiResponse<ItemResponse>
    {
        Success = true,
        Data = mapper.Map<ItemResponse>(item)
    });
});

// POST
app.MapPost("/items", async (
    CreateItemRequest req,
    IValidator<CreateItemRequest> validator,
    IItemService svc,
    IMapper mapper) =>
{
    var validation = await validator.ValidateAsync(req);
    if (!validation.IsValid)
        return Results.BadRequest(validation.Errors.Select(e=>e.ErrorMessage));
    var entity = mapper.Map<Item>(req);
    var created = await svc.CreateAsync(entity);
    return Results.Ok(new ApiResponse<ItemResponse> { Success=true, Data=mapper.
        Map<ItemResponse>(created) });
});

// PUT
app.MapPut("/items/{id}", async (
    Guid id,
    UpdateItemRequest req,
    IValidator<UpdateItemRequest> validator,
    IItemService svc,
    IMapper mapper) =>
{
    var valid = await validator.ValidateAsync(req);
    if (!valid.IsValid)
        return Results.BadRequest(valid.Errors.Select(e=>e.ErrorMessage));
    var entity = mapper.Map<Item>(req);
    var updated = await svc.UpdateAsync(id, entity);
    if (updated==null) return Results.NotFound();
    return Results.Ok(new ApiResponse<ItemResponse> { Success=true, Data=mapper.
        Map<ItemResponse>(updated) });
});

// DELETE
app.MapDelete("/items/{id}", async (Guid id, IItemService svc) =>
{
    await svc.DeleteAsync(id);
    return Results.Ok(new ApiResponse<string> { Success=true, Data="Deleted" });
});

// Start BPMN
app.MapPost("/workflow/start/{key}", async (string key, IHttpClientFactory http)=>
{
    var client = http.CreateClient();
    var res = await client.PostAsync(
        $"http://localhost:8080/engine-rest/process-definition/key/{key}/start",
        new StringContent("{}", Encoding.UTF8, "application/json"));
    return Results.StatusCode((int)res.StatusCode);
});

app.Run();
```

—

## 11 Modeling the BPMN in Camunda

In Camunda Modeler, create a process:

1. Start Event
2. Service Task using REST Connector
    - Method: POST

- URL: `http://yourapi/items`
- Body: JSON representing CreateItem payload
- Headers: `Content-Type:  application/json`

3. Another Service Task for Update: set method=PUT, URL `/items/{id}`
4. Another Service Task for DELETE if needed
5. End Event

The REST Connector task configuration lets you set Authentication and API key if needed — e.g., to include Grist API key in headers for calls that require it. :contentReference[oaicite:1]index=1

—

## 12 Deploying and Testing

1. Run Camunda server (e.g. via Docker)
2. Deploy your BPMN from Camunda Modeler
3. Start process instance using Camunda UI or API
4. The process calls your API endpoints to create, read, update, or delete records in Grist

—