**Integration**

Frontend-Backend Connection  
The integration process involves:  
CORS Configuration: Enable cross-origin requests from Angular (localhost:4200) to API (localhost:7000)  
HttpClient Setup: Configure Angular services to communicate with [ASP.NET](http://ASP.NET) Core API endpoints  
Data Binding: Connect Angular components to backend data sources  
Error Handling: Implement proper exception management across the full stack. note: i have created angular web app app for inventory management but not included -> HttpClient Setup: Configure Angular services to communicate with [ASP.NET](http://ASP.NET) Core API endpoints and Data Binding: Connect Angular components to backend data sources., so give me step by step procedure to do this backend in visual studio community. i want it like when i update the inventary dashboard by signning in as user, supplier to add new project, remove or no of quantities sold, no of quantities available in inventory, user name and id mentioned in user, seller it has to be updated in backed and store in sql. i hope this is the right. later i have to integrate it sre agent to monitor the code review done by github copilot for coding standards, checking, instructions and specifications by giving prompts to github copilot. basically sre agent will monitor the code review done by github copilot. this is the current scenario of the project. now tell me if the front end and backend pf this project must be done in visual studio community to integrate both or just backend in VS community. and tell me step by step how to do this in VS community 2022. i need the steps from the screen shot attached.

Based on your question and the Visual Studio 2022 start screen, here's the clarification and step-by-step procedure:

**Architecture Clarification**

**Recommended Setup:**

* **Backend**: Visual Studio Community 2022 ([ASP.NET](http://ASP.NET) Core Web API + SQL Server)
* **Frontend**: VS Code (Angular 20 application - keep your existing Angular app)
* **Integration**: HTTP communication between separate applications

You **do not need** to move your Angular frontend to Visual Studio Community. Keep it in VS Code and create only the backend API in Visual Studio Community.

**Step-by-Step Backend Creation in Visual Studio Community 2022**

**Step 1: Create New** [**ASP.NET**](http://ASP.NET) **Core Project**

From the Visual Studio 2022 start screen (your screenshot):

1. Click **"Create a new project"** (the bottom option in the "Get started" section)
2. In the project template search:
   * Type "[ASP.NET](http://ASP.NET) Core Web API" in the search box
   * Select "[ASP.NET](http://ASP.NET) Core Web API" template
   * Click **Next**
3. **Configure your project:**
   * Project name: InventoryManagement.API
   * Location: Choose your preferred folder
   * Solution name: InventoryManagement
   * Click **Next**
4. **Additional information:**
   * Framework: **.NET 8.0** (or latest)
   * Authentication type: **None** (for now)
   * Check ✅ **Configure for HTTPS**
   * Check ✅ **Enable OpenAPI support** (Swagger)
   * Uncheck ☐ **Use controllers** (we'll add them manually)
   * Click **Create**

**Step 2: Install Required NuGet Packages**

1. Right-click on your project → **Manage NuGet Packages**
2. Go to **Browse** tab and install:

Microsoft.EntityFrameworkCore.SqlServer  
Microsoft.EntityFrameworkCore.Tools  
Microsoft.EntityFrameworkCore.Design  
Microsoft.AspNetCore.Cors

**Step 3: Create Data Models**

1. Right-click project → **Add** → **New Folder** → Name it Models
2. Right-click Models folder → **Add** → **Class** → Name it Inventory.cs

using System.ComponentModel.DataAnnotations;  
  
namespace InventoryManagement.API.Models  
{  
 public class Inventory  
 {  
 [Key]  
 public int InventoryID { get; set; }  
   
 [Required]  
 [StringLength(100)]  
 public string ItemName { get; set; }  
   
 [Required]  
 public int StockQty { get; set; }  
   
 [Required]   
 public int ReorderQty { get; set; }  
   
 [Required]  
 public int PriorityStatus { get; set; } // 0 = Low, 1 = High  
   
 public DateTime CreatedDate { get; set; } = DateTime.Now;  
 public DateTime? LastUpdated { get; set; }  
   
 // Navigation property  
 public virtual ICollection<Stock> Stocks { get; set; } = new List<Stock>();  
 }  
   
 public class User  
 {  
 [Key]  
 public int UserID { get; set; }  
   
 [Required]  
 [StringLength(100)]  
 public string UserName { get; set; }  
   
 [Required]  
 [EmailAddress]  
 [StringLength(150)]  
 public string Email { get; set; }  
   
 [Required]  
 public string UserType { get; set; } // Admin, User, Supplier  
   
 public DateTime CreatedDate { get; set; } = DateTime.Now;  
 public bool IsActive { get; set; } = true;  
 }  
   
 public class Stock  
 {  
 [Key]  
 public int StockID { get; set; }  
   
 [Required]  
 public int InventoryID { get; set; }  
   
 [ForeignKey("InventoryID")]  
 public virtual Inventory Inventory { get; set; }  
   
 [Required]  
 public int Quantity { get; set; }  
   
 [Required]  
 public string TransactionType { get; set; } // IN, OUT, ADJUSTMENT  
   
 public string Remarks { get; set; }  
   
 public int UserID { get; set; } // Who performed the transaction  
   
 public DateTime TransactionDate { get; set; } = DateTime.Now;  
 }  
}

**Step 4: Create Database Context**

1. Create folder Data
2. Add class InventoryDbContext.cs:

using Microsoft.EntityFrameworkCore;  
using InventoryManagement.API.Models;  
  
namespace InventoryManagement.API.Data  
{  
 public class InventoryDbContext : DbContext  
 {  
 public InventoryDbContext(DbContextOptions<InventoryDbContext> options) : base(options)  
 {  
 }  
   
 public DbSet<Inventory> Inventories { get; set; }  
 public DbSet<User> Users { get; set; }  
 public DbSet<Stock> Stocks { get; set; }  
   
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 // Inventory configuration  
 modelBuilder.Entity<Inventory>(entity =>  
 {  
 entity.HasKey(e => e.InventoryID);  
 entity.Property(e => e.ItemName).IsRequired().HasMaxLength(100);  
 });  
   
 // User configuration  
 modelBuilder.Entity<User>(entity =>  
 {  
 entity.HasKey(e => e.UserID);  
 entity.HasIndex(e => e.Email).IsUnique();  
 });  
   
 // Stock configuration with relationships  
 modelBuilder.Entity<Stock>(entity =>  
 {  
 entity.HasKey(e => e.StockID);  
 entity.HasOne(s => s.Inventory)  
 .WithMany(i => i.Stocks)  
 .HasForeignKey(s => s.InventoryID)  
 .OnDelete(DeleteBehavior.Cascade);  
 });  
 }  
 }  
}

**Step 5: Configure Connection String**

1. Open appsettings.json
2. Add connection string:

{  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 },  
 "AllowedHosts": "\*",  
 "ConnectionStrings": {  
 "DefaultConnection": "Server=(localdb)\\MSSqlLocalDB;Database=InventoryManagementDB;Trusted\_Connection=true;MultipleActiveResultSets=true;Encrypt=false"  
 }  
}

**Step 6: Configure Services in Program.cs**

Replace Program.cs content:

using Microsoft.EntityFrameworkCore;  
using InventoryManagement.API.Data;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// Add services  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
  
// Add Entity Framework  
builder.Services.AddDbContext<InventoryDbContext>(options =>  
 options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));  
  
// Add CORS for Angular app  
builder.Services.AddCors(options =>  
{  
 options.AddPolicy("AllowAngularApp", policy =>  
 {  
 policy.WithOrigins("http://localhost:4200") // Angular dev server  
 .AllowAnyHeader()  
 .AllowAnyMethod()  
 .AllowCredentials();  
 });  
});  
  
var app = builder.Build();  
  
// Configure pipeline  
if (app.Environment.IsDevelopment())  
{  
 app.UseSwagger();  
 app.UseSwaggerUI();  
}  
  
app.UseHttpsRedirection();  
app.UseCors("AllowAngularApp"); // Enable CORS  
app.UseAuthorization();  
app.MapControllers();  
  
app.Run();

**Step 7: Create Controllers**

1. Create folder Controllers
2. Add InventoryController.cs:

using Microsoft.AspNetCore.Mvc;  
using Microsoft.EntityFrameworkCore;  
using InventoryManagement.API.Data;  
using InventoryManagement.API.Models;  
  
namespace InventoryManagement.API.Controllers  
{  
 [Route("api/[controller]")]  
 [ApiController]  
 public class InventoryController : ControllerBase  
 {  
 private readonly InventoryDbContext \_context;  
   
 public InventoryController(InventoryDbContext context)  
 {  
 \_context = context;  
 }  
   
 // GET: api/Inventory  
 [HttpGet]  
 public async Task<ActionResult<IEnumerable<Inventory>>> GetInventories()  
 {  
 return await \_context.Inventories.ToListAsync();  
 }  
   
 // GET: api/Inventory/5  
 [HttpGet("{id}")]  
 public async Task<ActionResult<Inventory>> GetInventory(int id)  
 {  
 var inventory = await \_context.Inventories.FindAsync(id);  
 if (inventory == null) return NotFound();  
 return inventory;  
 }  
   
 // POST: api/Inventory  
 [HttpPost]  
 public async Task<ActionResult<Inventory>> PostInventory(Inventory inventory)  
 {  
 \_context.Inventories.Add(inventory);  
 await \_context.SaveChangesAsync();  
 return CreatedAtAction("GetInventory", new { id = inventory.InventoryID }, inventory);  
 }  
   
 // PUT: api/Inventory/5  
 [HttpPut("{id}")]  
 public async Task<IActionResult> PutInventory(int id, Inventory inventory)  
 {  
 if (id != inventory.InventoryID) return BadRequest();  
   
 inventory.LastUpdated = DateTime.Now;  
 \_context.Entry(inventory).State = EntityState.Modified;  
   
 try  
 {  
 await \_context.SaveChangesAsync();  
 }  
 catch (DbUpdateConcurrencyException)  
 {  
 if (!InventoryExists(id)) return NotFound();  
 throw;  
 }  
   
 return NoContent();  
 }  
   
 // DELETE: api/Inventory/5  
 [HttpDelete("{id}")]  
 public async Task<IActionResult> DeleteInventory(int id)  
 {  
 var inventory = await \_context.Inventories.FindAsync(id);  
 if (inventory == null) return NotFound();  
   
 \_context.Inventories.Remove(inventory);  
 await \_context.SaveChangesAsync();  
   
 return NoContent();  
 }  
   
 // POST: api/Inventory/UpdateStock  
 [HttpPost("UpdateStock")]  
 public async Task<ActionResult> UpdateStock([FromBody] StockUpdateRequest request)  
 {  
 var inventory = await \_context.Inventories.FindAsync(request.InventoryID);  
 if (inventory == null) return NotFound();  
   
 // Update inventory quantity  
 if (request.TransactionType == "IN")  
 inventory.StockQty += request.Quantity;  
 else if (request.TransactionType == "OUT")  
 inventory.StockQty -= request.Quantity;  
 else  
 inventory.StockQty = request.Quantity; // ADJUSTMENT  
   
 inventory.LastUpdated = DateTime.Now;  
   
 // Create stock transaction record  
 var stockTransaction = new Stock  
 {  
 InventoryID = request.InventoryID,  
 Quantity = request.Quantity,  
 TransactionType = request.TransactionType,  
 Remarks = request.Remarks,  
 UserID = request.UserID,  
 TransactionDate = DateTime.Now  
 };  
   
 \_context.Stocks.Add(stockTransaction);  
 await \_context.SaveChangesAsync();  
   
 return Ok(new { message = "Stock updated successfully", newQuantity = inventory.StockQty });  
 }  
   
 private bool InventoryExists(int id)  
 {  
 return \_context.Inventories.Any(e => e.InventoryID == id);  
 }  
 }  
   
 public class StockUpdateRequest  
 {  
 public int InventoryID { get; set; }  
 public int Quantity { get; set; }  
 public string TransactionType { get; set; } // IN, OUT, ADJUSTMENT  
 public string Remarks { get; set; }  
 public int UserID { get; set; }  
 }  
}

**Step 8: Create Database Migration**

1. Open **Package Manager Console** (Tools → NuGet Package Manager → Package Manager Console)
2. Run these commands:

Add-Migration InitialCreate  
Update-Database

**Step 9: Test the API**

1. Press **F5** or **Ctrl+F5** to run the project
2. Swagger UI should open automatically
3. Test the endpoints to ensure they work

**Step 10: Next - Connect Your Angular App**

In your existing Angular app (VS Code), update your service to connect to this API:

// In your Angular service  
private baseUrl = 'https://localhost:7000/api'; // Adjust port number

This backend will now:

* ✅ Handle inventory CRUD operations
* ✅ Track stock movements with user information
* ✅ Store all data in SQL Server
* ✅ Provide CORS-enabled API for your Angular frontend
* ✅ Support transactions (add/remove quantities)
* ✅ Maintain user tracking for all operations

Your Angular app stays in VS Code, and this .NET API runs in Visual Studio Community 2022. They communicate via HTTP APIs.

⁂

1. image.jpg