https://github.com/ramazanorhanor/KatmanliMimariMVC5

# Katmanlı bir mimaride dikkat edilecek noktalar.

**RamazanOrhan Proje Şablonuna uygun Product CRUD mini proje**yi çıkaracağım:

* **Katmanlı mimari**: UI / Service / Data / Domain / Infrastructure / CrossCutting
* **Dependency Injection (Autofac)**
* **Repository + UnitOfWork pattern**
* **SaveChanges() sadece UnitOfWork’de**
* **AJAX/Fetch + async/await**
* **CSRF/XSS/SQL Injection korumaları**
* **ViewModel/DTO kullanımı**

| **Katman** | **Hangi katmana referans verir?** | **Açıklama** |
| --- | --- | --- |
| **UI** | **Service**, **CrossCutting** | Controller’lar service katmanını kullanır. CrossCutting içinde DI, logging, helper’lar olabilir. **Data veya Domain’e direkt referans vermez.** |
| **Service** | **Data**, **Domain**, **Infrastructure** | Business logic çalışır. Repository ve UnitOfWork burada çağrılır. DTO ↔ Entity mapping burada yapılır. |
| **Data** | **Domain** | Repository’ler ve DbContext entity’leri kullanır. |
| **Domain** | – | Sadece entity tanımları, bağımlılık yok. |
| **Infrastructure** | – | Logger, caching vb. helper’lar sağlar. |
| **CrossCutting** | – | DI config ve yardımcı policy’ler içerir. UI veya Service’e bağımlı olmamalıdır. |

Katmanlar ve Amaçları

| **Katman** | **Proje Türü** | **Framework** | **Amaç** |
| --- | --- | --- | --- |
| **UI** | ASP.NET MVC 5 | 4.7.2 | View, Controller, AJAX endpoint |
| **Service** | Class Library | 4.7.2 | İş kuralları, DTO ↔ Entity mapping |
| **Data** | Class Library | 4.7.2 | EF 6.2, Repository + UnitOfWork |
| **Domain** | Class Library | 4.7.2 | POCO Entity tanımları |
| **Infrastructure** | Class Library | 4.7.2 | Logger, Exception Handling, Security Helpers |
| **CrossCutting** | Class Library | 4.7.2 | Autofac DI, Caching, Security Policies |

* Katmanlar: **Domain → Data → Service → UI**
* DI: **Autofac**, UI içinde setup
* CRUD: **Product**, DTO ile
* UI: **AJAX + Fetch + async/await**, JSON Result
* Security: **CSRF**, DTO mapping, SaveChanges **sadece Service’de**

2 NuGet Paketleri

| **Paket** | **Versiyon** | **Katman** |
| --- | --- | --- |
| EntityFramework | 6.2.0 | Data,Service,UI |
| Autofac | 4.9.4 | CrossCutting (UI)\* |
| Autofac.Mvc5 | 4.0.2 | CrossCutting |
| Newtonsoft.Json | 13.0.1 | UI |
| Microsoft.jQuery.Unobtrusive.Ajax | 3.2.6 | UI |

jQuery 3.2.6

## 1️⃣ Domain Katmanı – Entities

|  |
| --- |
| namespace Domain\_Katmani.Entities  {  /// <summary>  /// Nedir: Ürün entity’si, DB’de Products tablosuna karşılık gelir.  /// Amaç: Domain modelini temsil eder.  /// Ne işe yarar: Service ve Data katmanında Product verilerini taşımak için kullanılır.  /// Enterprise Best Practice: POCO olarak tasarlanır, bağımlılık içermez.  /// </summary>  public class Product  {  public int Id { get; set; } // Primary Key  public string Name { get; set; } // Ürün adı  public decimal Price { get; set; } // Ürün fiyatı  }  } |

## 2️⃣ Data Katmanı – Repository

|  |
| --- |
| using Domain\_Katmani.Entities;  using System.Data.Entity;  namespace Data\_Katmani  {  public class AppDbContext : DbContext  {  public AppDbContext() : base("DefaultConnection") { }  public DbSet<Product> Products { get; set; }  }  }  Burası UI\_Web Web.config kısmında yazılır  <connectionStrings>  <add name="DefaultConnection" connectionString="Data Source=RORHAN\MSSQLSERVERDEV;Initial Catalog=YapayZekaDB;Integrated Security=True" providerName="System.Data.SqlClient" />  </connectionStrings> |
| using System.Linq;  namespace Data\_Katmani  {  /// <summary>  /// Nedir: Generic repository, tüm entityler için CRUD metotlarını sağlar  /// Amaç: Veri erişim kodunu tekrar kullanabilir hale getirmek  /// Ne işe yarar: Service katmanı repository üzerinden veri çekip/ekleyebilir  /// Enterprise Best Practice: Repository sadece veri erişimi yapar, SaveChanges içermez  /// </summary>  public interface IRepository<T> where T : class  {  IQueryable<T> GetAll();  T GetById(int id);  void Add(T entity);  void Update(T entity);  void Delete(T entity);  }  } |
| using System.Data.Entity;  using System.Linq;  namespace Data\_Katmani  {  public class Repository<T> : IRepository<T> where T : class  {  protected readonly AppDbContext \_context;  protected readonly DbSet<T> \_dbSet;  public Repository(AppDbContext context)  {  \_context = context;  \_dbSet = context.Set<T>();  }  public IQueryable<T> GetAll() => \_dbSet;  public T GetById(int id) => \_dbSet.Find(id);  public void Add(T entity) => \_dbSet.Add(entity);  public void Update(T entity) => \_context.Entry(entity).State = EntityState.Modified;  public void Delete(T entity) => \_dbSet.Remove(entity);  }  } |

## 3️⃣ Service Katmanı – Business Logic + SaveChanges

|  |
| --- |
| namespace Service\_Business.DTOs  {  public class ProductDto  {  public int Id { get; set; }  public string Name { get; set; }  public decimal Price { get; set; }  }  } |
| using Data\_Katmani;  using Domain\_Katmani.Entities;  using System;  namespace Service\_Business.UnitofWorkPattern  {  public interface IUnitOfWork : IDisposable  {  IRepository<Product> Products { get; }  int SaveChanges();  }  } |
| using Data\_Katmani;  using Domain\_Katmani.Entities;  namespace Service\_Business.UnitofWorkPattern  {  public class UnitOfWork : IUnitOfWork  {  private readonly AppDbContext \_context;  public IRepository<Product> Products { get; private set; }  public UnitOfWork(AppDbContext context)  {  \_context = context;  Products = new Repository<Product>(\_context);  }  public int SaveChanges() => \_context.SaveChanges();  public void Dispose() => \_context.Dispose();  }  } |
| using Service\_Business.DTOs;  using System.Linq;  namespace Service\_Business.Abstract  {  public interface IProductService  {  void AddProduct(ProductDto dto);  void UpdateProduct(ProductDto dto);  void DeleteProduct(int id);  IQueryable<ProductDto> GetAllProducts();  }  } |
| using Data\_Katmani;  using Domain\_Katmani.Entities;  using Service\_Business.Abstract;  using Service\_Business.DTOs;  using Service\_Business.UnitofWorkPattern;  using System.Linq;  namespace Service\_Business.Concrete  {  /// <summary>  /// Nedir: Product business logic  /// Amaç: Repository üzerinden CRUD işlemleri yapar, transaction yönetir  /// Ne işe yarar: SaveChanges Service katmanında yapılır, UI sadece DTO gönderir  /// Enterprise Best Practice: UnitOfWork Service’de, Repository sadece veri erişim  /// </summary>  public class ProductService : IProductService  {  private readonly IUnitOfWork \_uow;  private readonly IRepository<Product> \_productRepository;  public ProductService(IRepository<Product> productRepository,IUnitOfWork uow)  {  \_uow = uow;  \_productRepository = productRepository;  }  public void AddProduct(ProductDto dto)  {  var product = new Product  {  Name = dto.Name,  Price = dto.Price  };  \_uow.Products.Add(product);  \_uow.SaveChanges();  }  // Benzer şekilde Update, Delete, GetAll metotları  public void UpdateProduct(ProductDto dto)  {  var product = \_productRepository.GetById(dto.Id);  if (product == null) return;  product.Name = dto.Name;  product.Price = dto.Price;  \_productRepository.Update(product);  \_uow.SaveChanges();  }  public void DeleteProduct(int id)  {  var product = \_productRepository.GetById(id);  if (product == null) return;  \_productRepository.Delete(product);  \_uow.SaveChanges();  }  public IQueryable<ProductDto> GetAllProducts()  {  return \_productRepository.GetAll()  .Select(p => new ProductDto  {  Id = p.Id,  Name = p.Name,  Price = p.Price  });  }  // public IQueryable<ProductDto> GetAllProducts() => \_productRepository.GetAll();  }  } |
|  |
|  |

## 4️⃣ UI Katmanı – Controller

|  |
| --- |
| using Service\_Business.Abstract;  using Service\_Business.DTOs;  using System.Linq;  using System.Web;  using System.Web.Mvc;  namespace UI\_Web.Controllers  {  /// <summary>  /// Nedir: Product için MVC Controller  /// Amaç: UI’den gelen CRUD taleplerini Service katmanına yönlendirmek  /// Ne işe yarar: JSON ve View ile kullanıcıya veri sunar  /// Enterprise Best Practice: CSRF koruması, async ve AJAX uyumlu, DTO kullanımı  /// </summary>  public class ProductController : Controller  {  private readonly IProductService \_productService;  // Constructor Injection (Autofac ile register edilmiş)  public ProductController(IProductService productService)  {  \_productService = productService;  }  [HttpGet]  public ActionResult Index()  {  return View();  }  [HttpGet]  public JsonResult GetAllProducts()  {  var products = \_productService.GetAllProducts()  .Select(p => new ProductDto  {  Id = p.Id,  Name = p.Name,  Price = p.Price  }).ToList();  return Json(products, JsonRequestBehavior.AllowGet);  }  [HttpGet]  public ActionResult Create()  {  return View();  }  [HttpPost]  [ValidateAntiForgeryToken]  public JsonResult Create(ProductDto dto)  {  if (!ModelState.IsValid)  return Json(new { success = false, message = "Validation failed" });  \_productService.AddProduct(dto);  return Json(new { success = true, message = "Product added successfully" });  }  [HttpPost]  [ValidateAntiForgeryToken]  public JsonResult Update(ProductDto dto)  {  if (!ModelState.IsValid)  return Json(new { success = false, message = "Geçersiz veri" });  \_productService.UpdateProduct(dto);  return Json(new { success = true });  }  // [HttpPost]  //[ValidateAntiForgeryToken]  public JsonResult Delete(int id)  {  //https://localhost:44397/Product/Delete/3  \_productService.DeleteProduct(id);  return Json(new { success = true });  }  }  } |
|  |

## 5️⃣ UI Katmanı – View

### View (Index.cshtml)

|  |
| --- |
| @{  ViewBag.Title = "Product CRUD";  }  <h2>Product CRUD</h2>  <form id="productForm">  <input type="hidden" id="Id" />  <input type="text" id="Name" placeholder="Product Name" />  <input type="number" id="Price" placeholder="Price" />  <button type="submit">Save</button>  </form>  <ul id="productList"></ul>  @section scripts{  <script>  async function loadProducts() {  const response = await fetch('@Url.Action("GetAllProducts")');  const products = await response.json();  const list = document.getElementById('productList');  list.innerHTML = '';  products.forEach(p => {  const li = document.createElement('li');  li.textContent = `${p.Name} - ${p.Price}`;  list.appendChild(li);  });  }  document.getElementById('productForm').addEventListener('submit', async (e) => {  e.preventDefault();  const token = document.querySelector('input[name="\_\_RequestVerificationToken"]').value;  const dto = {  Id: document.getElementById('Id').value,  Name: document.getElementById('Name').value,  Price: parseFloat(document.getElementById('Price').value)  };  await fetch('@Url.Action("Create")', {  method: 'POST',  headers: {  'Content-Type': 'application/json',  'RequestVerificationToken': token  },  body: JSON.stringify(dto)  });  await loadProducts();  });  // Page load  loadProducts();  </script>  } |

### View (Create.cshtml)

|  |
| --- |
| @\*@using (Html.BeginForm(null, null, FormMethod.Post, new { id = "productForm" }))  {  @Html.AntiForgeryToken()  <input type="text" id="Name" name="Name" placeholder="Product Name" />  <input type="number" id="Price" name="Price" placeholder="Price" />  <button type="submit">Save</button>  }  <script src="~/Scripts/jquery-3.6.0.min.js"></script>  <script>  $("#productForm").submit(function (e) {  e.preventDefault();  $.ajax({  url: '@Url.Action("Create", "Product")',  type: 'POST',  data: $(this).serialize(),  success: function (res) {  if (res.success) {  alert(res.message);  } else {  alert("Error: " + res.message);  }  }  });  });  </script>\*@  @model Service\_Business.DTOs.ProductDto  @using (Html.BeginForm("Create", "Product", FormMethod.Post, new { id = "productForm" }))  {  @Html.AntiForgeryToken()  <input type="text" id="Name" name="Name" placeholder="Product Name" />  <input type="number" id="Price" name="Price" placeholder="Price" />  <button type="submit">Save</button>  }  <script src="~/Scripts/jquery-3.6.0.min.js"></script>  <script>  $("#productForm").submit(function (e) {  e.preventDefault();  $.ajax({  url: '@Url.Action("Create", "Product")',  type: 'POST',  data: $(this).serialize(),  success: function (res) {  if (res.success) {  alert(res.message);  window.location.href = '@Url.Action("Index","Product")';  } else {  alert("Error: " + res.message);  }  }  });  });  </script> |

Projeyi ayağa kaldırmak istediğimizde => https://localhost:44314/Product/Index

No parameterless constructor defined for this object. Hatası ile karşılaşırız. Bunun nedeni

|  |
| --- |
| // Constructor Injection (Autofac ile register edilmiş)  public ProductController(IProductService productService)  {  \_productService = productService;  }  Bu kod dur Autofac implementasyonunu yap hata düzelir. |

## Config Ayarları

|  |
| --- |
| UI\_Web\Web.config  <connectionStrings>  <add name="DefaultConnection" connectionString="Data Source=RORHAN\MSSQLSERVERDEV;Initial Catalog=YapayZekaDB;Integrated Security=True" providerName="System.Data.SqlClient" />  </connectionStrings>  <entityFramework>  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />  <providers>  <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />  </providers>  </entityFramework> |
| UI\_Web\Global.asax.cs  using System.Web.Mvc;  using System.Web.Optimization;  using System.Web.Routing;  using UI\_Web.Autofac\_DI;  namespace UI\_Web  {  public class MvcApplication : System.Web.HttpApplication  {  protected void Application\_Start()  {  AreaRegistration.RegisterAllAreas();  FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);  RouteConfig.RegisterRoutes(RouteTable.Routes);  BundleConfig.RegisterBundles(BundleTable.Bundles);  AutofacConfig.RegisterDependencies();  }  }  } |
| using Autofac;  using Autofac.Integration.Mvc;  using System.Web.Mvc;  using Service\_Business.Concrete;  using Service\_Business.Abstract;  using Service\_Business.UnitofWorkPattern;  using Data\_Katmani;  namespace UI\_Web.Autofac\_DI  {  public class AutofacConfig  {  public static void RegisterDependencies()  {  var builder = new ContainerBuilder();  // Controller register  // builder.RegisterControllers(Assembly.Load("UI\_Web"));  // Controller’ları register et  builder.RegisterControllers(typeof(MvcApplication).Assembly);  // DbContext register  builder.RegisterType<AppDbContext>()  .AsSelf()  .InstancePerRequest();  // Generic repository register  builder.RegisterGeneric(typeof(Repository<>))  .As(typeof(IRepository<>))  .InstancePerRequest();  // UnitOfWork ve Repository  builder.RegisterType<UnitOfWork>().As<IUnitOfWork>().InstancePerRequest();  // Service layer  builder.RegisterType<ProductService>().As<IProductService>().InstancePerRequest();  // Build container and set MVC resolver  var container = builder.Build();  DependencyResolver.SetResolver(new AutofacDependencyResolver(container));  }  }  } |

# Proje Notları

✅ Product CRUD – RamazanOrhan Proje Şablonu ile

1. Katmanlar ve Amaçları

| Katman | Proje Türü | Framework Versiyon | Amaç |

| ------------------ | ------------- | -------------------- | ------------------------------------------------------ |

| \*\*UI (Web)\*\* | ASP.NET MVC 5 | .NET Framework 4.7.2 | Kullanıcı arayüzü, Controller, View, AJAX endpointleri |

| \*\*Service\*\* | Class Library | .NET Framework 4.7.2 | İş kuralları, Validasyon, DTO/ViewModel mapping |

| \*\*Data\*\* | Class Library | .NET Framework 4.7.2 | Entity Framework (Repository + UoW) |

| \*\*Domain\*\* | Class Library | .NET Framework 4.7.2 | POCO Entity tanımları |

| \*\*Infrastructure\*\* | Class Library | .NET Framework 4.7.2 | Logger, Exception handling, Security helpers |

| \*\*CrossCutting\*\* | Class Library | .NET Framework 4.7.2 | IoC Container (Autofac), Caching, Security policies |

2. Framework ve Paketler

| Paket | Versiyon | Katman |

| --------------------------------- | -------- | ------------ |

| EntityFramework | 6.2.0 | Data,Service,UI |

| Autofac | 4.9.4 | CrossCutting |

| Autofac.Mvc5 | 4.0.2 | CrossCutting |

| Newtonsoft.Json | 13.0.1 | UI |

| Microsoft.jQuery.Unobtrusive.Ajax | 3.2.6 | UI |

3. Referans ve Bağımlılıklar

| Katman | Referanslar |

| ------------------ | ------------------------------------- |

| \*\*UI\*\* | Service, CrossCutting |

| \*\*Service\*\* | Data, Domain, Infrastructure |

| \*\*Data\*\* | Domain |

| \*\*Domain\*\* | (Bağımsız) |

| \*\*Infrastructure\*\* | (Bağımsız) |

| \*\*CrossCutting\*\* | Service, Infrastructure, Data, Domain |

----------------------------------------------------------------------------------

4. Kodlama Standartları

Repository Pattern → SaveChanges sadece UnitOfWork içinde.

Dependency Injection → Autofac ile katman bağımlılıkları çözülür.

ViewModel / DTO → UI ile Service arasında veri transferi.

Security Practices:

SQL Injection → EF parameterized query.

XSS → Razor HTML encode + AntiXssEncoder.

CSRF → [ValidateAntiForgeryToken].

----------------------------------------------------------------------------------

Doğru Akış

Repository Katmanı

Sadece Entity operasyonlarını içerir (Add, Update, Delete, GetAll, GetById).

SaveChanges() burada asla olmamalı.

Amaç: DB işlemlerini tek tek yapar, transaction yönetimi yapmaz.

UnitOfWork (UoW) Katmanı

Tüm repository’leri tek bir noktada toplar.

Transaction yönetimi ve SaveChanges() burada yapılır.

Böylece birden fazla repository üzerinde tek commit uygulanabilir.

Service Katmanı

İş kurallarını ve business logic’i uygular.

Repository ve UnitOfWork üzerinden veri okuma/yazma işlemlerini yönetir.

Örnek: ProductService içinde \_uow.Products.Add(product) ve \_uow.SaveChanges() çağrılır.

Böylece UI katmanı veya Controller, direct SaveChanges() çağırmak zorunda kalmaz.

SRP (Single Responsibility Principle) korunur: Repository sadece veri erişimi, Service sadece iş kuralları.

Transaction yönetimi merkezi: birden fazla entity aynı anda commit edilebilir.

Test edilebilirlik artar: Repository mock’lanabilir.

UI katmanı sadece DTO gönderir, SaveChanges() ile uğraşmaz.

----------------------------------------------------------------------------------

MvcApplication nedir?

MvcApplication UI projesindeki Global.asax.cs’de tanımlı sınıftır.

Yani builder.RegisterControllers(typeof(MvcApplication).Assembly); diyor ki:

"UI projesindeki tüm Controller’ları bul ve Autofac’a register et."

Eğer Visual Studio sana kırmızı çizgi gösteriyorsa:

MvcApplication sınıfını bulamıyor demektir.

Genellikle namespace yanlış veya dosya eksik.

2️⃣ Hangi referanslar gerekli?

Autofac register için UI projesi kendi içinde olmalı.

builder.RegisterControllers sadece UI assembly’sini kullanır.

Bu sebeple UI projesi, Data/Service/Domain katmanlarını reference olarak eklemelidir:

----------------------------------------------------------------------------------

🔹 Doğru Katman Akışı

UI (ASP.NET MVC)

└─ Controllers → ProductController(IProductService)

(DTO gönderir, async AJAX/Fetch çağrısı yapar)

Service (Class Library)

└─ ProductService(IProductService)

- Repository<Product> kullanır

- İş kuralları uygular

- SaveChanges() ve UnitOfWork burada yapılır

Data (Class Library)

└─ Repository<Product>

- Add, Update, Delete, GetAll/GetById

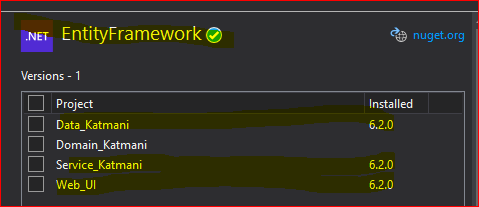
- SaveChanges içermez

Domain (Class Library)

└─ Entities/Product

- POCO entity

--------------------------------------------------------------------------------



# SON