Guide till Ditt Första Fullstack-projekt

Innehåll

Arkitektur

Din applikation är uppbyggd med följande komponenter:

- **Frontend**: Blazor WebAssembly (klienten)
- **Backend**: ASP.NET Core Web API (servern)
- **Databas**: MongoDB
- **Shared Library**: Delad kod mellan frontend och backend

Viktiga Komponenter

1. Autentisering och Auktorisering

```
public class TokenDto
{
    public string Token { get; set; }
    public DateTime Expiration { get; set; }
}
```

JWT (JSON Web Token) används för att hantera användarautentisering.

2. Repository Pattern

```
public interface IProductRepository
{
    Task<IEnumerable<Product>> GetAllAsync();
    Task<Product> GetByIdAsync(string id);
    Task<Product> CreateAsync(Product product);
    // ...
}
```

Repository hanterar alla databasoperationer.

3. Unit of Work Pattern

```
public interface IUnitOfWork
{
    IProductRepository Products { get; }
    ICustomerRepository Customers { get; }
    IOrderRepository Orders { get; }
    Task<bool> SaveChangesAsync();
}
```

Hanterar transaktioner och repositories.

Viktiga Koncept

1. Dependency Injection

```
// I Program.cs
builder.Services.AddScoped<IProductService, ProductService>();
builder.Services.AddScoped<IProductRepository, ProductRepository>();
```

2. Razor Components

```
@page "/admin/products"
@using Microsoft.AspNetCore.Components.Authorization
@using TheLoadingBean.Client.Services
@using TheLoadingBean.Shared.DTOs
```

3. API Controllers

```
[ApiController]
[Route("api/[controller]")]
public class ProductController : ControllerBase
{
    private readonly IProductService _productService;
    // ...
}
```

Bästa Praktik

1. Separation of Concerns

- Controllers hanterar HTTP-requests
- Services innehåller affärslogik
- Repositories hanterar databasoperationer
- DTOs hanterar dataöverföring

2. Validering

```
[Required]
[Range(0, double.MaxValue)]
public decimal Price { get; set; }
```

3. Felhantering

```
try
{
    await ProductService.CreateProductAsync(createProductModel);
    ToastService.ShowSuccess("Product created successfully!");
}
catch (Exception ex)
{
    ToastService.ShowError(ex.Message);
}
```

Utvecklingsflöde

1. Definiera DTOs för dataöverföring
2. Skapa repositories för databasoperationer
3. Implementera services för affärslogik
4. Skapa API controllers
5. Bygga Blazor-komponenter för användargränssnittet
6. Implementera autentisering och auktorisering
7. Lägg till validering och felhantering
8. Testa med unit tests

Tips för Utveckling

- Använd versionering (Git) för att spåra ändringar
- Skriv unit tests för viktig funktionalitet
- Följ SOLID-principerna
- Använd dependency injection för lös koppling
- Implementera proper error handling
- Använd logging för felsökning

DTOs (Data Transfer Objects)

DTOs används för att överföra data mellan olika lager i applikationen:

Auth-relaterade DTOs

- `LoginDto`: Inloggningsdata
- `RegisterDto`: Registreringsdata

- `TokenDto`: JWT-token och utgångsdatum

Produkt-relaterade DTOs

- `ProductResponseDto`: Produktdata från server till klient
- `CreateProductDto`: Skapa nya produkter
- `UpdateProductDto`: Uppdatera befintliga produkter

Kund-relaterade DTOs

- `CustomerResponseDto`: Kunddata från server till klient
- `CreateCustomerDto`: Skapa nya kunder
- `UpdateCustomerDto`: Uppdatera befintliga kunder

Order-relaterade DTOs

- `OrderResponseDto`: Orderdata från server till klient
- `CreateOrderDto`: Skapa nya ordrar
- `UpdateOrderDto`: Uppdatera befintliga ordrar
- `OrderItemDto`: Hantera enskilda orderrader

Fördelar med DTOs

1. **Säkerhet**: Kontroll över exakt vilken data som skickas
2. **Validering**: Inbyggda valideringsregler
3. **Separation av ansvar**: Separerar presentationslagret från affärslogiken
4. **Prestanda**: Optimerad dataöverföring

Detta är ett utmärkt första fullstack-projekt eftersom det täcker många viktiga koncept inom modern webbutveckling. Projektet ger dig praktisk erfarenhet av både frontend- och backend-utveckling, samtidigt som det introducerar viktiga arkitekturmönster och bästa praxis inom utveckling.