Add EF Core NPGSQL (Hangisini sorarlarsa o eklenecek)

.**NET CLI (View > Terminal > cd ./{ProjectName} yapılacak önce)** (Burada hangi projeye eklenecekse önce o seçilmeli)

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL **Program.cs DataLayer proj**

**PackageManager Console**

NuGet\Install-Package Npgsql.EntityFrameworkCore.PostgreSQL **Program.cs DataLayer proj**

Add EF Core SQLServer (Hangisini sorarlarsa o eklenecek)

dotnet add package Microsoft.EntityFrameworkCore.SQLServer **Program.cs projda**

NuGet\Install-Package Microsoft.EntityFrameworkCore.SQLServer **Program.cs projda**

Add EF Core

dotnet add package Microsoft.EntityFrameworkCore **Program.cs ve DataLayer projda**

dotnet add package Microsoft.EntityFrameworkCore.Design(Migration **DataLayer proj ve program.cs**)

dotnet add package Microsoft.EntityFrameworkCore.Relational

dotnet add package Microsoft.EntityFrameworkCore.Tools (migration update-database **DataLayer projda**)

dotnet add package Microsoft.EntityFrameworkCore.InMemory

dotnet add package Microsoft.EntityFrameworkCore.Proxies (LazyLoading için)

dotnet tool install --global dotnet-ef

NuGet\Install-Package Microsoft.EntityFrameworkCore

NuGet\Install-Package Microsoft.EntityFrameworkCore.Design (Migration ve program.cs in oraya)

NuGet\Install-Package Microsoft.EntityFrameworkCore.Relational

NuGet\Install-Package Microsoft.EntityFrameworkCore.Tools

NuGet\Install-Package Microsoft.EntityFrameworkCore.Proxies

NuGet\Install-Package Microsoft.EntityFrameworkCore.InMemory

AutoMapper

dotnet add package AutoMapper

NuGet\Install-Package AutoMapper -Version 14.0.0

API Swagger

dotnet add package Swashbuckle.AspNetCore **Program.cs in olduğu yere**

NuGet\Install-Package Swashbuckle.AspNetCore **Program.cs in olduğu yere**

appSettings.Develeopment.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},"Jwt": {

"SecretKey": "v@Cc1nE!Pr0jeKt2025\_T0ken\_S3cretKey\_X",

"Issuer": "BurcuSozayApp",

"Audience": "BurcuSozay",

"AccessTokenExpirationMinutes": 15,

"RefreshTokenExpirationDays": 7

},

"ConnectionStrings": {

"PostgresConnection": "Host=localhost;Database=vaccinedb;Username=postgres;Password=1234qqq"

},"Redis": {

"ConnectionString": "localhost:6379", // Redis sunucunuza göre ayarlayın

"InstanceName": "VaccineCache:"

},"AllowedOrigins": "http://localhost:3000,https://localhost:44395"

}

Hata: Eğer şu hata alınıyorsa SecretKey kısa gelmiştir. Daha uzun bir string kullanılmalı.

IDX10720: Unable to create KeyedHashAlgorithm for algorithm 'HS256', the key size must be greater than: '256' bits, key has '232' bits.

lunchSettings

Eğer projede otomatik olarak WebAPI’nin swagger’ı açmasını istiyorsan lunchSettings’de aşağıdaki ayarları eklemek gerekir.

Eğer IISExpress’de proje başlatılıyorsa,

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

},

"launchUrl": "swagger"

}

"launchBrowser": true, ve "launchUrl": "swagger" değerlerinin eklendiği kontrol edilmelidir.

Bearer Token JWT

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer **Program.cs in olduğu yere**

NuGet\Install-Package Microsoft.AspNetCore.Authentication.JwtBearer

DependencyInjection

dotnet add package Microsoft.Extensions.DependencyInjection.Abstractions **Program.cs**

builder.Services.AddScoped(typeof(IRepository<>), typeof(Repository<>));

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>(); **Program.cs**

ElasticSearch

docker pull docker.elastic.co/elasticsearch/elasticsearch-wolfi:9.0.0

docker run -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:7.5.2

dotnet add package NEST

dotnet add package Elasticsearch.Net

Microsoft.Identity

dotnet add package Microsoft.AspNetCore.Identity **PassswordHash için**

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore **(IdentityDbContext için)**

**1234qqqQ. İçin Password Hash: AQAAAAIAAYagAAAAEPrgn1neXYwO3p+/k2iOnMe4EF3w7CNZG8YLLts7WwYIV6vnUrGKaEzQq2tUKO2XSA==**

Migration NpgSQL Code First

**DBContext ve Entity’ler eklenecek. Daha sonra efcore için yukarıdaki seed işleminin çalışması için db migration eklenmesi gerek. PackageManager’da aşağıdaki komutlar sırasıyla yazılır.**

*Eğer CodeFirst* ***appsettings.Development.json*** *üzerinden yapılacaksa ve* ***IDesignTimeDbContextFactory<AppDbContext>*** *ile code first yapılacaksa önce ortam development ortamı olduğu bildirilmeli ve aşağıdaki kod kullanılmalı . Burada migration yapmak için şu paketlerin Data katmanında olması gerekir. Daha sonra migration ekleme komutaları çalışacak. Bu işlemler .sln root dizinden yapılmalı. Bu iş yapılırken* ***Microsoft.EntityFrameworkCore.Design*** *nuget paket de webapi.csproj da olmalı*

***dotnet add package Microsoft.Extensions.Configuration***

***dotnet add package Microsoft.Extensions.Configuration.Json***

***dotnet add package Microsoft.EntityFrameworkCore.Design***

$env:ASPNETCORE\_ENVIRONMENT = "Development" **=> Bu işlem VaccineApp.Data Katmanında yapılıyor**

**Eğer migrationdan önce bir hata alınırsa mutlaka bunu çalıştır.**

* dotnet tool install --global dotnet-ef
* **Aşağıdaki 2 işlem .snl root dizininde yapılacak**

dotnet ef migrations add InitialCodeFirstMigration --context AppDbContext --project VaccineApp.Data/VaccineApp.Data.csproj --startup-project VaccineApp.WebAPI/VaccineApp.WebAPI.csproj

dotnet ef database update --project VaccineApp.Data/VaccineApp.Data.csproj --startup-project VaccineApp.WebAPI/VaccineApp.WebAPI.csproj

* **Eğer Migration silinmesi gerekirse bu işlem yine .snl root dizininde yapılacak. Sonra yukarıdaki işlemleri tekrar yap.**

dotnet ef migrations remove --project VaccineApp.Data/VaccineApp.Data.csproj --startup-project VaccineApp.WebAPI/VaccineApp.WebAPI.csproj

* **Eğer aynı csproj içinde migration yapılacaksa bunları yap**

dotnet ef migrations add InitialCodeFirstMigration --context AppDbContext

dotnet ef database update --context AppDbContext

Enable-Migrations

add-migration EmptyMigration

update-database

**Daha sonra seed işlemi için uygulama ayağa kaldırılır.**

Migration NpgSQL DB First

Eğer hazırda bir db varsa ve buradan entity modeller yaratılacaksa bu komut kullanılır**. Eğer public adında bir şema yok derse db export import yapmak lazım.**

**dotnet tool install --global dotnet-ef**

dotnet ef dbcontext scaffold "Host=localhost;Database=vaccinedb;Username=postgres;Password=1234qqq" Npgsql.EntityFrameworkCore.PostgreSQL --output-dir Context --context AppDbContext --no-onconfiguring --force –verbose --use-database-names

RabbitMq Kurulum

dotnet add package RabbitMQ.Client **(Kuyruk mekanizması nerede olacaksa. WorkerServis)**

docker run -d --hostname rabbitmq --name rabbitmq -e RABBITMQ\_DEFAULT\_USER=admin -e RABBITMQ\_DEFAULT\_PASS=123456 -p 5672:5672 -p 15672:15672 rabbitmq:3-management

Redis Kurulumu

~~dotnet add package StackExchange.Redis~~  **Program.cs in olduğu yere**

dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis

docker run -d --name redis-stack -p 6379:6379 -p 8001:8001 redis/redis-stack:latest

UnitOfWorkTransactionFilter

UnioOfWork transaction yönetimini sağlamak için servislerin başına bir filtre koyulur

dotnet add package Microsoft.AspNetCore.Mvc.Abstractions **Business katmanı**

builder.Services.AddScoped<UnitOfWorkTransactionFilter>(); **Program.cs**

builder.Services.AddControllers(options =>{

options.Filters.Add<UnitOfWorkTransactionFilter>();

});

Nuget

Nuget paketlerde herhangi bir hata alınması durumunda şu kodu kullan

**dotnet nuget locals all --clear**

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

using VaccineApp.Business.AutoMapper;

using VaccineApp.Business.Base;

using VaccineApp.Business.Register;

using VaccineApp.Business.Repository;

using VaccineApp.Business.UnitOfWork;

using VaccineApp.Data.Context;

using VaccineApp.DataSeed;

using VaccineApp.ViewModel.Options;

public class Program{

private static void Main(string[] args){

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>(options => options

.UseLazyLoadingProxies() // tüm Entity DB Relationlar virtual yapılmalı

.UseNpgsql(builder.Configuration.GetConnectionString("PostgresConnection")));

// Redis (IDistributedCache icin StackExchange Redis kullanimi)

builder.Services.AddStackExchangeRedisCache(options =>{

options.Configuration = builder.Configuration["Redis:ConnectionString"];

options.InstanceName = "VaccineCache:";

});

builder.Services.AddScoped(typeof(IRepository<,>), typeof(Repository<,>));

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

builder.Services.AddScoped<AuditActionFilter>();

builder.Services.AddHttpContextAccessor(); // Username alabilmek icin

builder.Services.AddAutoMapper(typeof(AutoMappingProfile));

builder.Services.AddComponents();

var allowedOrigins = builder.Configuration["AllowedOrigins"]?.Split(",") ?? new[] { "\*" };

builder.Services.AddCors(options =>{

options.AddDefaultPolicy(policy =>{

policy.WithOrigins(allowedOrigins)

.AllowAnyHeader()

.AllowAnyMethod();

});

});

// Add services to the container.

builder.Services.AddControllers(options => {

//options.Filters.Add<UnitOfWorkTransactionFilter>();

options.Filters.AddService<AuditActionFilter>();

});

// JWT konfigürasyonu (appsettings.json -> "Jwt" section kullanılmalı)

var jwtConfig = builder.Configuration.GetSection("Jwt");

var key = Encoding.ASCII.GetBytes(jwtConfig["SecretKey"]);

builder.Services.Configure<JWTSettingOptions>(jwtConfig);

builder.Services.AddAuthentication(options =>{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>{

options.TokenValidationParameters = new TokenValidationParameters{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtConfig["Issuer"],

ValidAudience = jwtConfig["Audience"],

IssuerSigningKey = new SymmetricSecurityKey(key)

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

// Swagger JWT Auth desteklemesi

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme{

Description = "JWT Authorization header using the Bearer scheme. Example: 'Bearer {token}'",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.Http,

Scheme = "bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement{{

new OpenApiSecurityScheme{ Reference = new OpenApiReference{ Type = ReferenceType.SecurityScheme, Id = "Bearer" },

Scheme = "oauth2",

Name = "Bearer",

In = ParameterLocation.Header},Array.Empty<string>() }});

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment()){

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(options => { options.SwaggerEndpoint("/swagger/v1/swagger.json", "Vaccine API"); });

}

app.UseHttpsRedirection();

app.UseMiddleware<TransactionMiddleware>();

app.UseCors();

app.UseAuthentication(); // Auth middleware aktif edilmeli

app.UseAuthorization();

app.MapControllers();

VaccineDbSeed.SeedDatabase(app, builder.Environment.IsDevelopment());

app.Run();

}

}

RabbitMQ Kuyruk Yapısı Oluşturma Publlisher Worker Servis

dotnet add package RabbitMQ.Client **(Paketi eklenir)**

**Hata**: [error] ACCESS\_REFUSED - Login was refused using authentication mechanism PLAIN. For details see the broker logfile. **guest/guest ile sadece localhost'tan bağlanabilirsin!**

**Çözüm**: var factory = new ConnectionFactory { HostName = "localhost" }; **yerine** **şununla değiştir**

var factory = new RabbitMQ.Client.ConnectionFactory() { HostName = "localhost", UserName = "admin" ,Password = "123456" };

Eğer elinde username şifre yoksa kullanıcı oluşturma adımları; **guest/guest ile sadece localhost'tan bağlanabilirsin!**

rabbitmqctl add\_user buro 1234qqqQ.

rabbitmqctl set\_user\_tags buro administrator

rabbitmqctl set\_permissions -p / buro ".\*" ".\*" ".\*"

sonra <http://localhost:15672> üzerinden erişim sağlayabiliyor musun kontrol et. Kodda şu değişikliği yap

var factory=new ConnectionFactory(){HostName="localhost",UserName="buro",Password="1234qqqQ.", VirtualHost = "/" };

**Hata**: [error] closing AMQP connection (10.0.0.2:5672 -> 10.0.0.1:5672): user 'admin' in vhost '/' unauthorized to access queue 'soket\_data\_queue'

**Çözüm:** rabbitmqctl add\_user buro 1234qqqQ.

rabbitmqctl set\_permissions -p / buro ".\*" ".\*" ".\*"

await using var rabbitConn = await factory.CreateConnectionAsync();

await using var channel = await rabbitConn.CreateChannelAsync();

await channel.QueueDeclareAsync(queue: "soket\_data\_queue", durable: false, exclusive: false, autoDelete: false);

**Hata:** [info] user 'admin' authenticated and granted access to vhost '/'

[error] closing AMQP connection ... missed heartbeats from client, timeout: 60s

**Çözüm:** RabbitMQ sunucusu, **60 saniye boyunca istemciden “heartbeat” (canlılık mesajı)** alamıyor ve bağlantıyı kapatıyor. Kodunda bağlantı kurduktan sonra uzun süre işlem yapmıyorsan veya uygulaman donduysa, RabbitMQ "bu istemci cevap vermiyor" diyerek bağlantıyı sonlandırıyor. Kodun QueueDeclareAsync satırında ya da başka bir yerde *takılı* kalıyor olabilir. Özellikle **debug modda breakpoint ile uzun süre beklediğinde** heartbeat mesajı gönderilemez ve bağlantı kapanır.

RabbitMQ Consumer Worker Servis

Eğer burada PostgreSQL DB’ye kayıt atılacaksa eklenecek paketler;

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

Bu projeden veya herhangi bir projeden WebAPI’ye istek atılacağı zaman JWT Bearer Token alınıp o şekilde http isteği atılması gerekir. Bu nedenle WebAPI’deki AccountControllerdan Token almak için Login metodunu çağırmak gerekir. Bunun için izlenmesi gereken adımlar;

dotnet add package Microsoft.Extensions.Http **(isteğin atılacağı mikroservise eklenecek)**

services.AddHttpClient(); **(İsteğin atılacağı mikroservis Program.cs’e ekle)**

using Microsoft.Extensions.DependencyInjection; using Microsoft.Extensions.Http; **(using ekle)**

services.Configure<ServiceAccountOptions>(configuration.GetSection("ServiceAccount"));

public RabbitMqConsumerWorker(

ILogger<RabbitMqConsumerWorker> logger,

IOptions<ServiceAccountOptions> serviceAccountOptions,

IHttpClientFactory httpClientFactory) **(İsteğin atılacağı .cs’in constractor’a inject et)**

Redis Distributed Lock

Çoklu worker ortamında (Kubernetes veya başka bir cluster) **aynı outbox mesajı iki kez işlenmesin** istenirse bunun için **dağıtık kilit (distributed lock)** gerekir. Tek node Redis olduğundan RedLock yerine DistributedLock nuget paketi kullanıldı. Gerekli paketler;

dotnet add package DistributedLock

dotnet add package StackExchange.Redis

dotnet add package Microsoft.Extensions.Http

protected override async Task ExecuteAsync(CancellationToken stoppingToken){

var locker = new RedisDistributedSynchronizationProvider(\_redis.GetDatabase());

while (!stoppingToken.IsCancellationRequested) {

try{

var client = \_httpClientFactory.CreateClient();

// WebAPI'den işlenmemiş mesajları çek

var messages = await client.GetFromJsonAsync<List<OutboxMessageDto>>(

"https://localhost:44395/api/OutboxMessage/UnprocessedList",

cancellationToken: stoppingToken);

if (messages == null) { await Task.Delay(2000, stoppingToken); continue; }

foreach (var msg in messages) {

var lockKey = $"outbox-msg-lock:{msg.Id}";

await using (var handle = await locker.TryAcquireLockAsync(lockKey, TimeSpan.FromSeconds(5))){

if (handle == null)

continue; // Başka worker işliyor

try

{

// Burada işleme kodu (örn. event publish, başka servise POST)

// --- Örneğin alarmı RabbitMQ'ya publish et, vs. ---

// Mesajı ilgili API'ye gönder (ör: WebAPI endpointine)

// React Client Notification Gönder

// İşlenince WebAPI'ye bildir

var markResponse = await client.PostAsync($"https://localhost:44395/api/OutboxMessage/MarkProcessed/{msg.Id}", null, stoppingToken);

markResponse.EnsureSuccessStatusCode();

}

catch (Exception ex) {

// Hatalı durumda logla, DB’ye işlenmemiş olarak bırak

}

}

}

await Task.Delay(2000, stoppingToken);

}

catch (Exception ex) {

// Network veya global hata varsa bir süre bekle

await Task.Delay(5000, stoppingToken);

}

}

}

React

**Cmd console’dan**

npx create-react-app <my-app-name> (Eğer çalışmazsa aşağıdakiler yapılacak)

npm cache clean –force

npm install -g npm@latest

npm install -g create-react-app

npm install react-router-dom

npm install react-bootstrap

npm i @material-ui/core --legacy-peer-deps

npm config set legacy-peer-deps true (eğer dependency hatası verirse)

npm install react-router-dom

npm install axios

npm install ajv@^6 ajv-keywords@^3 (burada hangi paketse)

npm install @mui/material @emotion/react @emotion/styled

npm install

npm start

Sonra client klasörüne gidip sağ tuş VSCode ile aç diyoruz. Chrome’da kodun görünmesi için package-json’a eklenir.

"build": "GENERATE\_SOURCEMAP=false react-scripts build",

"winBuild": "set \"GENERATE\_SOURCEMAP=false\" && react-scripts build",

Eğer uygulama oluşturulduktan sonra dependency hatası alınırsa şu komutlar yapılacak

rm node\_modules -r -fo

rm package-lock.json -fo

npm install react-scripts@latest

React projesini VS2022’ye eklemek için **Solutions > Add > Existing Web Site** diyerek klasörün olduğu dizin seçilerek VS2022’ye eklenir. **VS2022’de Web Proje** eklenerek yapılırsa **npm start** komutu yerine **npm dev** kullanılıyor.

Örnek package.json aşağıda burada proje ismi açılan **{react-app-name}** ile değiştirilmelidir.

{ "name": "hastane.client",

  "version": "0.1.0",

  "private": true,

  "build": "GENERATE\_SOURCEMAP=false react-scripts build",

  "winBuild": "set \"GENERATE\_SOURCEMAP=false\" && react-scripts build",

  "dependencies": {

    "@testing-library/dom": "^10.4.0",

    "@testing-library/jest-dom": "^6.6.3",

    "@testing-library/react": "^16.3.0",

    "@testing-library/user-event": "^13.5.0",

    "ajv": "^6.12.6",

    "ajv-keywords": "^3.5.2",

    "axios": "^1.10.0",

    "react": "^19.1.0",

    "react-dom": "^19.1.0",

    "react-router-dom": "^7.6.2",

    "react-scripts": "^5.0.1",

    "web-vitals": "^2.1.4"

  },

  "scripts": {

    "start": "react-scripts start",

    "build": "react-scripts build",

    "test": "react-scripts test",

    "eject": "react-scripts eject"

  },

  "eslintConfig": {

    "extends": [

      "react-app",

      "react-app/jest"

    ]

  },

  "browserslist": {

    "production": [

      ">0.2%",

      "not dead",

      "not op\_mini all"

    ],

    "development": [

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ]

  }

}

SignalR

**dotnet add package RabbitMQ.Client**

**dotnet add package MassTransit.RabbitMQ**

**dotnet add package Microsoft.AspNetCore.SignalR.Client**

* Bildirimi Rabbit’e atan OutboxPublisher’da **MassTransit.RabbitMQ** ve **RabbitMQ.Client** paketi olmalı
* Rabbitten bunları alcak ayrıca bir NotificationWorkerServis ve **MassTransit.RabbitMQ ile Microsoft.AspNetCore.SignalR.Client** paketi olmalı
* Consume edildiği yerden SignalR Hub’a gönderimi ve bunu alacak bir Hub oluşturulması gerekiyor.

Hub içinb WebAPI tarafına Hub tipinde bir class eklenmesi gerekiyor.

namespace VaccineApp.WebAPI.Hubs {

public class NotificationHub : Hub {

public async Task SendErrorMessage(string message) {

await Clients.All.SendAsync("ReceiveError", message);

}

}}

react tarafında ise Terminalde aşağıdaki paketin yüklenmesi gerekiyor

**npm install @microsoft/signalr**

**npm install** react-toastify

import React, { useState } from "react";

import api from "../js/api-call";

import { useAuth } from "../js/AuthContext";

export default function Login() {

    const { login } = useAuth();

    const [username, setUserName] = useState("");

    const [password, setPassword] = useState("");

    const [error, setError] = useState(null);

    const handleSubmit = async (e) => {

        e.preventDefault();

        try {

            const response = await api.post("/Account/Login", { username, password });

            if (response.data && response.data.accessToken) {

                login(response.data);

                setError(null);

            } else {

                setError("Login failed.");

            }

        } catch (err) {

            setError("Kullanıcı adı veya şifre hatalı.");

        }

    };

    return (

        <div className="login-page">

            <h1>Login</h1>

            <form onSubmit={handleSubmit}>

                <div className="form-group">

                    <input type="text" placeholder="Usename" value={username} onChange={e => setUserName(e.target.value)} />

                </div>

                <div className="form-group">

                    <input type="password" placeholder="Password" value={password} onChange={e => setPassword(e.target.value)} />

                </div>

                {error && <div style={{ color: "red" }}>{error}</div>}

                <div>

                    <button type="submit">Login</button>

                </div>

            </form>

        </div>

    );

}

Soket Programlama