# CORS

* For the resources to access the API we’ll need to setup the CORS policy
* Note: CORS is redone below with the policy with extension.

## Program.cs

### Add cors as a service

A screen shot of a computer program

Description automatically generated

### Add Cores to middleware

Assuming your font end url is <http://localhost:4200>.

A black screen with green and yellow text

Description automatically generated

* Cors will be redone as a policy below with extensions [here](#_RegisterCors_Method)
* And the urls will be moved to app settings [here](#_CORS)

# AppSettings

## CORS

Add URLs for the CORS to app settings

### appsettings.json



### appsettings.Development.json



### Core/Constants/ConfigKeyConstants.cs

Add the key to ConfigKeyConstants

A screen shot of a computer program

Description automatically generated

# Extensions

* Stuff that we have placed in programs.cs to for add DbContext, DI and above CORS will be moved to extensions to clear up programs.cs.
* Create a new folder Core/Extensions

## ConfigExtensions.cs

* Create a new file as ConfigExtensions.cs
* Make sure to rename the namespace as OAuth2.WebApi.Core.Extensions

We will put different methods here to retrieve info from the appsettings

using OAuth2.WebApi.Core.Constants;

namespace OAuth2.WebApi.Core.Extensions;

public static class ConfigExtensions

{

    /\* Generic GetSectionValue methods start \*/

    public static T GetSectionValue<T>(this IConfiguration config, string sectionName, T defaultValue)

    {

        if (!config.GetSection(sectionName).Exists())

        {

            return defaultValue;

        }

        var sValue = config.GetSection(sectionName).Get<T>();

        return sValue;

    }

    /\* Generic GetSectionValue methods end \*/

    /\* Helper methods to get the common items using the above GetSectionValue \*/

    public static string GetDefaultConnectionString(this IConfiguration config)

    {

        var connectionString = config.GetConnectionString(ConfigKeyConstants.DefaultConnection);

        return connectionString;

    }

    public static List<string> GetAllowSpecificOrigins(this IConfiguration config)

    {

        var allowSpecificOrigins = config.GetSectionValue<List<string>>(ConfigKeyConstants.AllowSpecificOrigins, null);

        return allowSpecificOrigins;

    }

    public static string GetLoggingLevelDefault(this IConfiguration config)

    {

        var loggingLevelDefault = config.GetSectionValue<string>(ConfigKeyConstants.LoggingLevelDefault, string.Empty);

        return loggingLevelDefault;

    }

    public static string GetLoggingLevelMsApnetCore(this IConfiguration config)

    {

        var loggingLevelDefault = config.GetSectionValue<string>(ConfigKeyConstants.LoggingLevelMsAspNetCore, string.Empty);

        return loggingLevelDefault;

    }

}

## ServiceExtensions.cs

* Create a new file as ServiceExtensions.cs
* Make sure to rename the namespace as OAuth2.WebApi.Core.Extensions

namespace OAuth2.WebApi.Core.Extensions;

public static class ServiceExtensions

{

}

### RegisterDbContext Method

Move the DbContext registration from programs.cs to here

Add using

using Microsoft.EntityFrameworkCore;

and the method will use the ConfigExtensions to pick the connection string

    public static void RegisterDbContext(this IServiceCollection services, IConfiguration configuration)

    {

        services.AddDbContext<DataContext>(opt =>

        {

            opt.UseSqlite(configuration.GetDefaultConnectionString());

        });

    }

### RegisterServices Method

Move the Repository and BusinessLogic registration here

Add using

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Data.Repositories;

And add the following method

    public static void RegisterServices(this IServiceCollection services, IConfiguration configuration)

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

    }

### RegisterCors Method

Register cors as a policy and return the policy to be used by the middleware

    public static string RegisterCors(this IServiceCollection services, IConfiguration configuration)

    {

        var myAllowSpecificOrigins = "\_myAllowSpecificOrigins";

        //https://stackoverflow.com/questions/42858335/how-to-hardcode-and-read-a-string-array-in-appsettings-json

        var allowedSpecificOrigins = configuration.GetAllowSpecificOrigins();

        if (allowedSpecificOrigins != null && allowedSpecificOrigins.Any())

        {

            services.AddCors(options =>

            {

                options.AddPolicy(name: myAllowSpecificOrigins,

                                policy =>

                                {

                                    policy.WithOrigins(allowedSpecificOrigins.ToArray())

                                    .AllowAnyHeader()

                                    .AllowAnyMethod();

                                });

            });

        }

        return myAllowSpecificOrigins;

    }

# Programs.cs use ServiceExtensions

## Using Statements Not Needed Due To Extension Use

using Microsoft.EntityFrameworkCore;

using OAuth2.WebApi.Core.Constants;

using OAuth2.WebApi.Core.Data.BusinessLogic;

using OAuth2.WebApi.Core.Data.Repositories;

using OAuth2.WebApi.Core.DB;

## Using Statement Needed

using OAuth2.WebApi.Core.Extensions;

## Here is the updated Add Services section

/\*Custom Section Start\*/

IConfiguration configuration = builder.Configuration;

builder.Services.RegisterDbContext(configuration);

builder.Services.RegisterServices(configuration);

var myAllowSpecificOrigins = builder.Services.RegisterCors(configuration);

/\*Custom Section End\*/

## Here is the updated Use Cors section

//CUSTOM: Start

//ordering is important here. UseCors before UseAuthentication and MapControllers

//

app.UseCors(myAllowSpecificOrigins);

//CUSTOM: END