


VERSION	1.0
FECHA	26/04/2022
CODIGO	

## VERSIONES DEL DOCUMENTO

Versión	Nombre	Actividad	Cargo
1.0	DIEGO ARIZALA	Creación del documento	Desarrollador Senior

	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		

## Contenido

VERSIONES DEL DOCUMENTO ..... 1

INFRAESTRUCTURA ..... 3

COMPONENTES TECNOLOGICOS DE LA SOLUCION ..... 4

PAQUETES EXTERNOS LIBRERÍAS Y COMPONENTES..... 6

    Para la capa de servicios: ..... 6


    Para la capa de visualización: ..... 6

CAPA DE DATOS DE LA SOLUCION ..... 6

    MODELO DE BASE DE DATOS: ..... 7

CONSUMO DE SERVICIOS ..... 8

NATURALEZA DEL DOCUMENTO	CONFIDENCIAL	PAGINA	2
--------------------------	--------------	--------	---

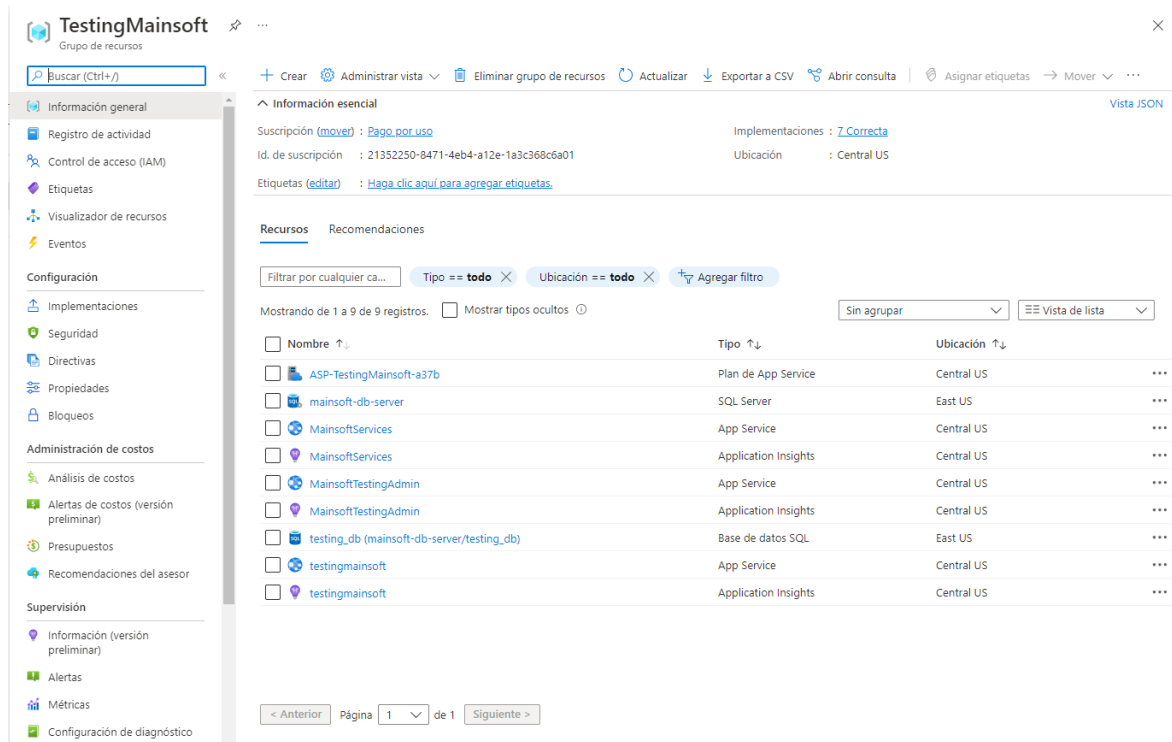
	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		

## INFRAESTRUCTURA

La aplicación testing cuenta con una infraestructura basada en la nube la cual comprende los siguientes componentes

- Base de datos SQL Server para la capa de datos
- AppService de Azure para la capa de visualización
- AppService de Azure para la capa de servicios
- AppService de Azure para la capa de administración (BackOffice)


Así las cosas la infraestructura se encuentra alojada en un grupo de recursos de Azure, visualizada de la siguiente forma:

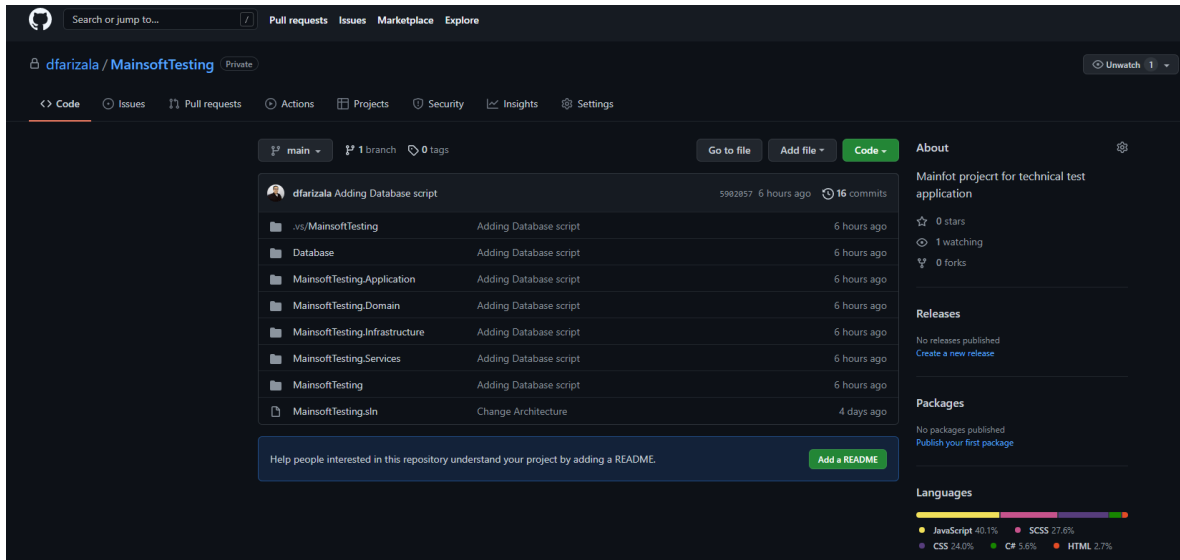


Nombre	Tipo	Ubicación
ASP-TestingMainsoft-a37b	Plan de App Service	Central US
mainsoft-db-server	SQL Server	East US
MainsoftServices	App Service	Central US
MainsoftServices	Application Insights	Central US
MainsoftTestingAdmin	App Service	Central US
MainsoftTestingAdmin	Application Insights	Central US
testing_db (mainsoft-db-server/testing_db)	Base de datos SQL	East US
testingmainsoft	App Service	Central US
testingmainsoft	Application Insights	Central US

Como se puede apreciar, cada uno de los componentes de la infraestructura tiene configurada una instancia de Application Insights, con lo cual se busca monitorear y mejorar el rendimiento de cada uno de los componentes

El código de cada uno de los componentes de la aplicación está alojado en un repositorio GitHub, el cual solo puede accederse de manera privada por los integrantes del equipo:

	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		

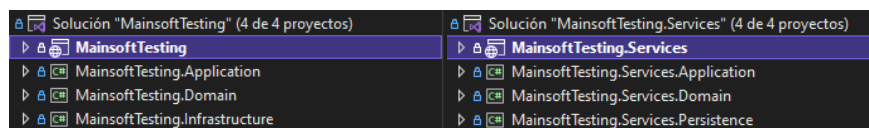


## COMPONENTES TECNOLOGICOS DE LA SOLUCION

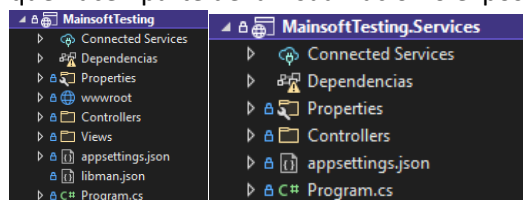
La solución está desarrollada en Visual Studio 2022, edición Community, usando como framework de desarrollo ASP.NET Core 6.0.


El patrón de diseño usado para la aplicación es hibrido, ya que es una combinación entre Modelo-Vista-Controlador, DDD y Clean Architecture, separando cada una de las capas de la siguiente forma:

- Se han implementado dos soluciones, una para los servicios y una para el front, cumpliendo así con el modelo de arquitectura orientada a servicios (SOA)

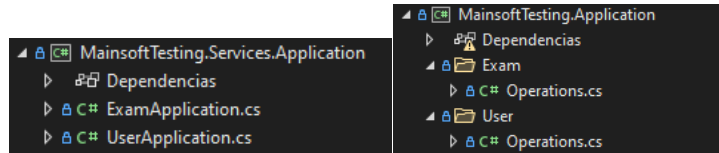


- Cada una de las soluciones posee las siguientes capas, separadas por proyecto
  - Capa de visualización:** En esta capa se almacenan todas la vistas y controladores que hacen parte de la visualización o exposición

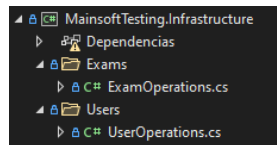


	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		

- **Capa de aplicación:** En esta capa se incluye toda la lógica de negocio y validaciones que debe hacer la solución

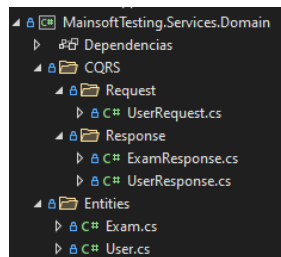


- **Capa de Infraestructura:** En esta capa se incluyen todas las funciones de conectividad de la aplicación con elementos o componentes externos, ya sean servicios REST u otro tipo de componentes

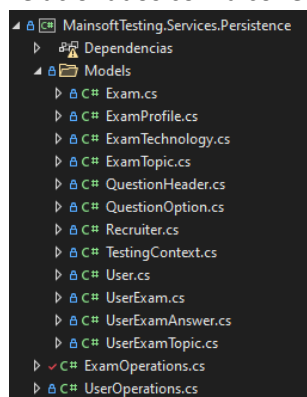



- **Capa de dominio:** En esta capa se alojan todas las definiciones de entidades y solicitudes de cada uno de los componentes de la solución

Esta capa viene dividida en secciones como las entidades y una sección de CQRS donde se alojan todas las consultas y comandos que se usan dentro de la solución



- **Capa de persistencia:** En esta capa se encuentran todas las operaciones que derivan en la base de datos, con el fin de que aquí solo se hagan procesos relacionados con la conexión de la solución con la base de datos








	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		







## PAQUETES EXTERNOS LIBRERÍAS Y COMPONENTES

Los paquetes usados en la solución se encuentran a continuación:

Para la capa de servicios:


	<b>AutoMapper</b> por Jimmy Bogard A convention-based object-object mapper	4.2.1 11.0.1
	<b>Microsoft.EntityFrameworkCore</b> por Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	6.0.4
	<b>Microsoft.EntityFrameworkCore.SqlServer</b> por Microsoft Microsoft SQL Server database provider for Entity Framework Core.	6.0.4
	<b>Microsoft.EntityFrameworkCore.Tools</b> por Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	6.0.4
	<b>Swashbuckle.AspNetCore</b> por Swashbuckle.AspNetCore Swagger tools for documenting APIs built on ASP.NET Core	6.2.3 6.3.1

Para la capa de visualización:

	<b>AutoMapper</b> por Jimmy Bogard A convention-based object-object mapper	4.2.1 11.0.1
	<b>Microsoft.AspNet.WebApi.Client</b> por Microsoft This package adds support for formatting and content negotiation to System.Net.Http.	5.2.8
	<b>Microsoft.VisualStudio.Web.CodeGeneration.Design</b> por Microsoft Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	6.0.3
	<b>Newtonsoft.Json</b> por James Newton-King Json.NET is a popular high-performance JSON framework for .NET	13.0.1
	<b>Swashbuckle.AspNetCore</b> por Swashbuckle.AspNetCore Swagger tools for documenting APIs built on ASP.NET Core	6.3.0 6.3.1
	<b>System.Net.Http</b> por Microsoft Provides a programming interface for modern HTTP applications, including HTTP client components that allow applications to consume web services over HTTP and HTTP components that can be used by both clients and servers for parsing HTTP headers.	4.3.4

## CAPA DE DATOS DE LA SOLUCION

NATURALEZA DEL DOCUMENTO	CONFIDENCIAL	PAGINA	6
--------------------------	--------------	--------	---

	DOCUMENTO DE ARIQUITECTURA		CONSECUTIVO
	VERSION	1.0	
	FECHA	26/04/2022	
	CODIGO		

Para la capa de datos de la solución, se ha designado como motor de conexión entre la base de datos y la capa de servicios Microsoft Entity Framework Core versión 6.0, con el fin de poder establecer una conexión por objetos usando la premisa “Code First” de Entity Framework:

```

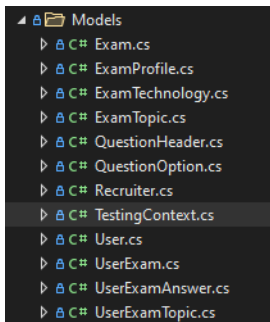
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace MainsoftTesting.Services.Persistence.Models
{
    13 referencias
    public partial class TestingContext : DbContext
    {
        5 referencias
        public TestingContext()
        {
        }

        0 referencias
        public TestingContext(DbContextOptions<TestingContext> options)
            : base(options)
        {
        }
    }
}

```

En el archivo TextingContext.cs se puede visualizar todo el esquema de Entity Framework Core para cada uno de los componentes, así como en la carpeta Models se pueden evidenciar las entidades de cada uno de los objetos de base de datos:



## MODELO DE BASE DE DATOS:

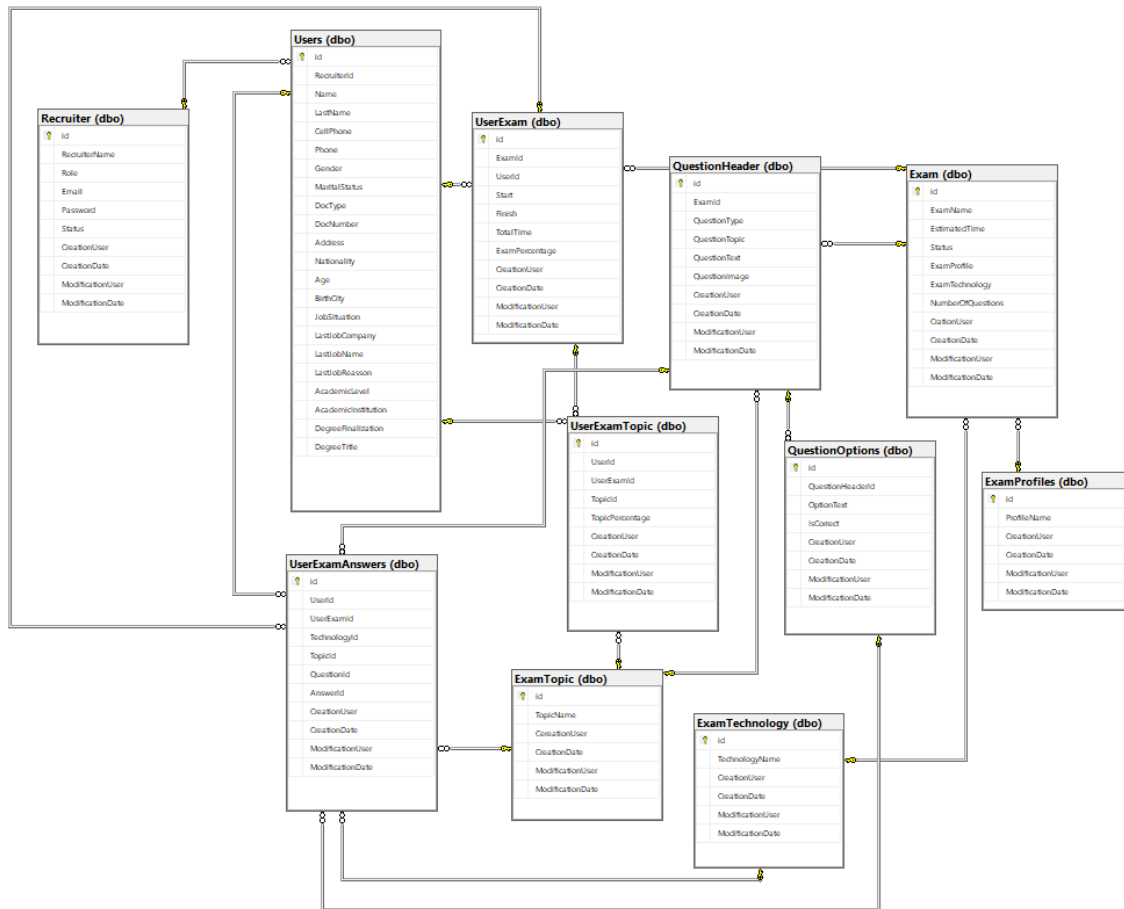
Las tablas creadas para esta solución son las siguientes:

```

dbo.Exam
dbo.ExamProfiles
dbo.ExamTechnology
dbo.ExamTopic
dbo.QuestionHeader
dbo.QuestionOptions
dbo.Recruiter
dbo.UserExam
dbo.UserExamAnswers
dbo.UserExamTopic
dbo.Users

```

El modelo entidad relación de la base de datos se puede observar a continuación



## CONSUMO DE SERVICIOS

La capa de visualización consume servicios por medio del protocolo HTTP, hacia la capa de servicios los cuales son de tipo REST API:

```

    async static public Task<GetExamsResponse> GetExams()
    {
        GetExamsResponse _Result = new GetExamsResponse();

        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri(_baseUrl);
            client.DefaultRequestHeaders.Clear();
            client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
            HttpResponseMessage Res = await client.GetAsync("Exam");
            if (Res.IsSuccessStatusCode)
            {
                var UserList = Res.Content.ReadAsStringAsync().Result;
                _Result = JsonConvert.DeserializeObject<GetExamsResponse>(UserList);
            }

            return _Result;
        }
    }

```