Appointments Manager

#dotnet #cli #container #webapi #CQRS

Features

- Feature based traceability. We will organize all code by feature at Domain, Application and Storage layers. In this example, which manages Appointments, we will have all related code in an ./Appointments folder in Domain, Application and Storage

Create new solution along with four projects to hold Domain, Storage, Application and WebAPI

```
git init
curl -OL https://raw.githubusercontent.com/cmeegamarachchi/reffiles/master/Dotnet.gitignore
mv Dotnet.gitignore .gitignore

REM optional update web template pack
dotnet new -i Microsoft.DotNet.Web.ProjectTemplates
REM optional update nunit pack
dotnet new -i NUnit3.DotNetNew.Template

REM create solution
dotnet new sln --name AppointmentsManager

REM create projects
dotnet new classlib --name AppointmentsManager.Domain -f netcoreapp3.0
dotnet new classlib --name AppointmentsManager.Application -f netcoreapp3.0
dotnet new nunit --name AppointmentsManager.Application.Tests -f netcoreapp3.0
dotnet new classlib --name AppointmentsManager.Storage -f netcoreapp3.0
dotnet new nunit --name AppointmentsManager.Storage.Tests -f netcoreapp3.0
dotnet new webapi --name AppointmentsManager.WebAPI -f netcoreapp3.0

REM add projects to solutions
dotnet sln AppointmentsManager.sln add AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
dotnet sln AppointmentsManager.sln add AppointmentsManager.Application/AppointmentsManager.Application.csproj
dotnet sln AppointmentsManager.sln add
AppointmentsManager.Application.Tests/AppointmentsManager.Application.Tests.csproj
dotnet sln AppointmentsManager.sln add AppointmentsManager.Storage/AppointmentsManager.Storage.csproj
dotnet sln AppointmentsManager.sln add AppointmentsManager.Storage.Tests/AppointmentsManager.Storage.Tests.csproj
dotnet sln AppointmentsManager.sln add AppointmentsManager.WebAPI/AppointmentsManager.WebAPI.csproj

REM fix dependencies
cd ./AppointmentsManager.Application
dotnet add reference ../AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
dotnet add reference ../AppointmentsManager.Storage/AppointmentsManager.Storage.csproj
cd ..

cd ./AppointmentsManager.Application.Tests
dotnet add reference ../AppointmentsManager.Application/AppointmentsManager.Application.csproj
dotnet add reference ../AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
dotnet add reference ../AppointmentsManager.Storage/AppointmentsManager.Storage.csproj
cd ..

cd ./AppointmentsManager.Storage
dotnet add reference ../AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
cd ..

cd ./AppointmentsManager.Storage.Tests
dotnet add reference ../AppointmentsManager.Storage/AppointmentsManager.Storage.csproj
dotnet add reference ../AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
cd ..

cd ./AppointmentsManager.WebAPI
dotnet add reference ../AppointmentsManager.Domain/AppointmentsManager.Domain.csproj
dotnet add reference ../AppointmentsManager.Storage/AppointmentsManager.Storage.csproj
dotnet add reference ../AppointmentsManager.Application/AppointmentsManager.Application.csproj
cd ..

REM Add MediatR to WebAPI and Application projects
cd ./AppointmentsManager.Application
dotnet add package MediatR
cd ..

cd ./AppointmentsManager.WebAPI
dotnet add package MediatR.Extensions.Microsoft.DependencyInjection
cd ..

REM Add Newtonsoft.Json to WebAPI and Storage projects
cd ./AppointmentsManager.Storage
dotnet add package Newtonsoft.Json
cd ..

REM run build and test to verify
dotnet build
```

```
dotnet test
```

Domain

Our system stores appointments. We will have entities `Appointment`, `AppointmentSummary` modeling our domain

Lets add `Appointment` and `AppointmentSummary` to our domain

```
cd ./AppointmentsManager.Domain/
rm Class1.cs
mkdir Appointments
cd Appointments
touch Appointment.cs
touch AppointmentSummary.cs
cd ../..
```

AppointmentSummary.cs

```
using System;

namespace AppointmentsManager.Domain.Appointments
{
    public class AppointmentSummary
    {
        public string Id { get; set; }
        public string Place { get; set; }
        public DateTime DateTime { get; set; }
        public bool Deleted { get; set; }
    }
}
```

Appointment.cs

```
using System;

namespace AppointmentsManager.Domain.Appointments
{
    public class Appointment
    {
        public string Id { get; set; }
        public string Place { get; set; }
        public string Description { get; set; }
        public string Comments { get; set; }
        public DateTime DateTime { get; set; }
    }
}
```

Storage

Next lets implement storage. Lets start by defining an interface which would give us the ability to have different flavours such as file-system based, blob based, sql based or document-db based or mixed implementations

```
cd ./AppointmentsManager.Storage/
rm Class1.cs
mkdir Appointments
cd Appointments
touch IAppointmentStorage.cs
cd ../..
```

```
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;

namespace AppointmentsManager.Storage.Appointments
{
    public interface IAppointmentStorage
    {
        Task<AppointmentSummary[]> GetAppointmentSummaries();
        Task<Appointment> GetAppointment(string id);
        Task Save(Appointment appointment);
    }
}
```

Lets implement file based storage and lets start by writing a test. What we write would be an integration test which would save an appointment and attempt to load it

Code for our test

```
cd ./AppointmentsManager.Storage.Tests/
rm UnitTest1.cs
mkdir Appointments
cd Appointments
touch FileSystemAppointmentStorageTests.cs
cd ../..
```

```
using System;
using System.IO;
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;
using AppointmentsManager.Storage.Appointments;
using NUnit.Framework;

namespace AppointmentsManager.Storage.Tests.Appointments
{
    [TestFixture]
    public class FileSystemAppointmentStorageTests
    {
        [Test]
        public async Task Can_save_and_read_appointment()
        {
            var appointment = new Appointment();
            appointment.Id = Guid.NewGuid().ToString();
            appointment.Place = "Alaska";
            appointment.Comments = "this is a comment";
            appointment.Description = "this is a description";
            appointment.DateTime = new DateTime(2009, 11, 23);

            var tempPath = Path.GetTempPath();

            await new FileSystemAppointmentStorage(tempPath).Save(appointment);

            var newAppointment = await new FileSystemAppointmentStorage(tempPath).GetAppointment(appointment.Id);

            Assert.IsTrue(appointment.Id == newAppointment.Id);
            Assert.IsTrue(appointment.Place == newAppointment.Place);
            Assert.IsTrue(appointment.Comments == newAppointment.Comments);
            Assert.IsTrue(appointment.Description == newAppointment.Description);
            Assert.IsTrue(appointment.DateTime == newAppointment.DateTime);
        }
    }
}
```

And initial version for `./AppointmentsManager.Storage/Appointments/FileSystemAppointmentStorage` code

```
cd ./AppointmentsManager.Storage/Appointments
touch FileSystemAppointmentStorage.cs
cd ../..
```

```
using System.Diagnostics.CodeAnalysis;
using System.IO;
using System.Threading.Tasks;
using Newtonsoft.Json;
using AppointmentsManager.Domain.Appointments;

namespace AppointmentsManager.Storage.Appointments
{
    [SuppressMessage("ReSharper", "ConvertToUsingDeclaration")]
    public class FileSystemAppointmentStorage: IAppointmentStorage
    {
        private readonly string _storagePath;

        public FileSystemAppointmentStorage(string storagePath)
        {
            _storagePath = storagePath;
        }

        public Task<AppointmentSummary[]> GetAppointmentSummaries()
        {
            throw new System.NotImplementedException();
        }
```

```
        public async Task<Appointment> GetAppointment(string id)
        {
            var fileName = $"{id}.json";

            Appointment appointment = null;

            using (var file = File.OpenText($"{_storagePath}\\{fileName}"))
            {
                JsonSerializer serializer = new JsonSerializer();
                await new TaskFactory().StartNew(() =>
                {
                    appointment = (Appointment) serializer.Deserialize(file, typeof(Appointment));
                });
            }

            return appointment;
        }

        public async Task Save(Appointment appointment)
        {
            var fileName = $"{appointment.Id}.json";

            await using (var file = File.CreateText($"{_storagePath}\\{fileName}"))
            {
                JsonSerializer serializer = new JsonSerializer();
                await new TaskFactory().StartNew(() =>
                {
                    serializer.Serialize(file, appointment);
                });
            }
        }

        public Task Delete(string id)
        {
            throw new System.NotImplementedException();
        }

    }
}
```

Appointment metadata

We use a separate file to keep track of appointment metatdata. There will be many requests for high-level appointment details such as list of appointments, appointment date, place. We use AppointmentSummary class to track these and appointment-catalogue.json to store them. In a mixed or relational database world, a table would be an ideal storage medium for the catalogue

Lets design a integration test to model following use case:
Adding an appointment should create appointment summary

```
//FileSystemAppointmentStorageTests.cs

[Test]
public async Task Adding_appointment_creates_appointment_summary()
{
    var appointment = new Appointment();
    appointment.Id = Guid.NewGuid().ToString();
    appointment.Place = "Alaska";
    appointment.Comments = "this is a comment";
    appointment.Description = "this is a description";
    appointment.DateTime = new DateTime(2009, 11, 23);

    var tempPath = Path.GetTempPath();

    await new FileSystemAppointmentStorage(tempPath).Save(appointment);

    var appointmentSummary = await new FileSystemAppointmentStorage(tempPath).GetAppointmentSummaries();

    Assert.IsTrue(appointmentSummary.Any(i => i.Id == appointment.Id));
}
```

And updated appointment storage code

```
using System.Diagnostics.CodeAnalysis;
```

```
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Newtonsoft.Json;
using AppointmentsManager.Domain.Appointments;

namespace AppointmentsManager.Storage.Appointments
{
    [SuppressMessage("ReSharper", "ConvertToUsingDeclaration")]
    public class FileSystemAppointmentStorage: IAppointmentStorage
    {
        private readonly string _storagePath;

        private const string AppointmentCatalogueFileName = "appointment_catalogue.json";

        public FileSystemAppointmentStorage(string storagePath)
        {
            _storagePath = storagePath;
        }

        public async Task<AppointmentSummary[]> GetAppointmentSummaries()
        {
            var appointmentSummaryList = new AppointmentSummary[0];

            var fileName = $"{_storagePath}/{AppointmentCatalogueFileName}";

            if (File.Exists(fileName))
            {
                using (var catalogueFile = File.OpenText(fileName))
                {
                    JsonSerializer serializer = new JsonSerializer();
                    await new TaskFactory().StartNew(() =>
                    {
                        appointmentSummaryList =
                            (AppointmentSummary[]) serializer.Deserialize(catalogueFile,
typeof(AppointmentSummary[]));
                    });
                }
            }

            return appointmentSummaryList;
        }

        public async Task<Appointment> GetAppointment(string id)
        {
            var fileName = $"{id}.json";

            Appointment appointment = null;

            using (var file = File.OpenText($"{_storagePath}\\{fileName}"))
            {
                JsonSerializer serializer = new JsonSerializer();
                await new TaskFactory().StartNew(() =>
                {
                    appointment = (Appointment) serializer.Deserialize(file, typeof(Appointment));
                });
            }

            return appointment;
        }

        [SuppressMessage("ReSharper", "VariableHidesOuterVariable")]
        public async Task Save(Appointment appointment)
        {
            AppointmentSummary[] UpdateAppointmentCatalogue(AppointmentSummary[] list, Appointment source)
            {
                var appointmentSummary = new AppointmentSummary();

                appointmentSummary.Id = source.Id;
                appointmentSummary.Place = source.Place;
                appointmentSummary.DateTime = source.DateTime;
                appointmentSummary.Deleted = false;

                list = list.Append(appointmentSummary).ToArray();

                return list;
            }

            var fileName = $"{appointment.Id}.json";

            await using (var file = File.CreateText($"{_storagePath}\\{fileName}"))
            {
                JsonSerializer serializer = new JsonSerializer();
                await new TaskFactory().StartNew(() =>
                {
                    serializer.Serialize(file, appointment);
```

```
                });
            }

            var appointmentCatalogue = await GetAppointmentSummaries();
            appointmentCatalogue = UpdateAppointmentCatalogue(appointmentCatalogue, appointment);

            await Save(appointmentCatalogue);
        }

        public Task Delete(string id)
        {
            throw new System.NotImplementedException();
        }

        private async Task Save(AppointmentSummary[] appointmentSummaryList)
        {
            await using (var file = File.CreateText($"{_storagePath}\\{AppointmentCatalogueFileName}"))
            {
                JsonSerializer serializer = new JsonSerializer();
                await new TaskFactory().StartNew(() =>
                {
                    serializer.Serialize(file, appointmentSummaryList);
                });
            }
        }
    }
}
```

Delete an appointment

In `FileSystemAppointmentStorage`, when a file is deleted, its corresponding catalogue entry is marked as deleted and its data file is renamed to <id>.deleted.json. Once an appointment is deleted `GetAppointment` for the appointment will return null

As usual, we start by writing a test and then updating code to make the test pass

```
[Test]
public async Task Deleting_appointment_deletes_appointment()
{
    var tempPath = Path.GetTempPath();

    // given an appointment
    var appointment = new Appointment();
    appointment.Id = Guid.NewGuid().ToString();
    appointment.Place = "Alaska";
    appointment.Comments = "this is a comment";
    appointment.Description = "this is a description";
    appointment.DateTime = new DateTime(2009, 11, 23);

    await new FileSystemAppointmentStorage(tempPath).Save(appointment);

    //when appointment is deleted
    await new FileSystemAppointmentStorage(tempPath).Delete(appointment.Id);

    //then appointment can no longer be loaded
    var deletedAppointment = await new FileSystemAppointmentStorage(tempPath).GetAppointment(appointment.Id);
    Assert.IsNull(deletedAppointment);

    //and then appointment summary will have be marked as deleted
    var appointmentSummary = await new FileSystemAppointmentStorage(tempPath).GetAppointmentSummaries();

    Assert.IsTrue(appointmentSummary.First(i => i.Id == appointment.Id).Deleted);
}
```

And the code

```
using System.Diagnostics.CodeAnalysis;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Newtonsoft.Json;
using AppointmentsManager.Domain.Appointments;

namespace AppointmentsManager.Storage.Appointments
{
    [SuppressMessage("ReSharper", "ConvertToUsingDeclaration")]
    public class FileSystemAppointmentStorage: IAppointmentStorage
```

```
   {
...

        public async Task<Appointment> GetAppointment(string id)
        {
            var fileName = $"{_storagePath}/{id}.json";

            Appointment appointment = null;

            if (File.Exists(fileName))
            {
                using (var file = File.OpenText(fileName))
                {
                    JsonSerializer serializer = new JsonSerializer();
                    await new TaskFactory().StartNew(() =>
                    {
                        appointment = (Appointment) serializer.Deserialize(file, typeof(Appointment));
                    });
                }
            }

            return appointment;
        }


        [SuppressMessage("ReSharper", "VariableHidesOuterVariable")]
        public async Task Delete(string id)
        {
            AppointmentSummary[] DisableAppointment(AppointmentSummary[] list, string id)
            {
                var appointmentSummary = list.FirstOrDefault(i => i.Id == id);
                if (appointmentSummary != null) appointmentSummary.Deleted = true;
                return list;
            }

            var appointmentSummaries = await GetAppointmentSummaries();
            appointmentSummaries = DisableAppointment(appointmentSummaries, id);

            await Save(appointmentSummaries);

            var oldPath = $"{_storagePath}/{id}.json";

            if (File.Exists(oldPath))
            {
                var newPath = $"{_storagePath}/{id}.deleted.json";

                File.Move(oldPath, newPath);
            }
        }

    }
}
```

Update an appointment

Updates to an appointment will be handled in Save method. Save method will first save the entity using the same code, which would override any entity with the same id, and it would update catalogue entry instead of creating a new one

Out test in this case should modify description and place of the appointment and check for the modifications

```
[Test]
public async Task Updating_appointment_updates_appointment()
{
    var tempPath = Path.GetTempPath();

    // given an appointment
    var appointment = new Appointment();
    appointment.Id = Guid.NewGuid().ToString();
    appointment.Place = "Alaska";
    appointment.Comments = "this is a comment";
    appointment.Description = "this is a description";
    appointment.DateTime = new DateTime(2009, 11, 23);

    await new FileSystemAppointmentStorage(tempPath).Save(appointment);

    //when the appointment is updated
    appointment.Place = "Colombo";
    appointment.Description = "description is updated";
```

```
    await new FileSystemAppointmentStorage(tempPath).Save(appointment);

    //then updated appointment can be loaded
    var updatedAppointment = await new FileSystemAppointmentStorage(tempPath).GetAppointment(appointment.Id);
    Assert.IsTrue(updatedAppointment.Place == "Colombo");
    Assert.IsTrue(updatedAppointment.Description == "description is updated");

    //and then updated appointment details are correctly propagated to catalogue
    var catalogue = await new FileSystemAppointmentStorage(tempPath).GetAppointmentSummaries();
    var appointmentSummary = catalogue.First(i => i.Id == appointment.Id);
    Assert.IsTrue(appointmentSummary.Place == "Colombo");
}
```

And the code update

```
[SuppressMessage("ReSharper", "VariableHidesOuterVariable")]
public async Task Save(Appointment appointment)
{
    AppointmentSummary[] UpdateAppointmentCatalogue(AppointmentSummary[] list, Appointment source)
    {
        AppointmentSummary appointmentSummary;

        // if found, update otherwise add-new
        if (list.Any(i => i.Id == source.Id))
        {
            appointmentSummary = list.First(i => i.Id == source.Id);
        }
        else
        {
            appointmentSummary = new AppointmentSummary();
            list = list.Append(appointmentSummary).ToArray();
        }

        appointmentSummary.Id = source.Id;
        appointmentSummary.Place = source.Place;
        appointmentSummary.DateTime = source.DateTime;
        appointmentSummary.Deleted = false;

        return list;
    }

    var fileName = $"{appointment.Id}.json";

    await using (var file = File.CreateText($"{_storagePath}\\{fileName}"))
    {
        JsonSerializer serializer = new JsonSerializer();
        await new TaskFactory().StartNew(() =>
        {
            serializer.Serialize(file, appointment);
        });
    }

    var appointmentCatalogue = await GetAppointmentSummaries();
    appointmentCatalogue = UpdateAppointmentCatalogue(appointmentCatalogue, appointment);

    await Save(appointmentCatalogue);
}
```