Appointments Manager

#dotnet #cli #container #webapi #CQRS #PATCH #REST #JsonPatch

Home: https://github.com/cmeegamarachchi/architecture_appointment_manager.git

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 ..
```

```
cd ./AppointmentsManager.WebAPI
dotnet add package Microsoft.AspNetCore.Mvc.NewtonsoftJson -version 3.0.3
cd ..

REM Add Fluent-Validation to WebAPI and Application projects
cd ./AppointmentsManager.Application
dotnet add package FluentValidation.AspNetCore
cd ..

cd ./AppointmentsManager.WebAPI
dotnet add package FluentValidation.AspNetCore
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);
       Task Delete(string id);
   }
}
```

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]
       [Category("Integration")]
       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]
[Category("Integration")]
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]
[Category("Integration")]
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]
[Category("Integration")]
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);
}
```

Application

Application layer implements use cases. We use CQRS in the application layer as our main pattern. Same as Domain layer, we use folder per feature to organize our code

Following use cases are implemented

- Should be able to get a list of Appointment summaries
- Return empty list if no Appointment summaries are found
- Given appointment id, return appointment
- Given appointment id, return null if no matching appointment found
- Add new appointment and return appointment id of newly created appointment
- Update appointment
- Delete appointment

Test first development

Development of each use-case will start with by writing a unit/integration test

Class in Class

Each user case is represented as a command or a query and will be named to reflect the main use-case addressed by it. Within each command/query, we will have genetically named classes such as Command, Query, Handeller .. While this looks like an anti pattern, in this case its not as once name spaced, they makes the intent very clear

Use case : Return list of summaries

The test

```
cd ./AppointmentsManager.Application.Tests/
rm UnitTest1.cs
mkdir Appointments
cd Appointments
touch GetAppointmentSummaryListQueryTest.cs
cd ../..


cd ./AppointmentsManager.Application/
rm Class1.cs
mkdir Appointments
cd Appointments
touch GetAppointmentSummaryListQuery.cs
cd ../..
```

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class GetAppointmentSummaryListQueryTest
    {
        [Test]
        [Category("Integration")]
        public async Task Returns_list_of_appointment_summaries()
        {
            // given there appointments
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

            // 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){ AppointmentCatalogueFileName = catalogueFileName
}.Save(appointment);
```

```
            // when GetAppointmentSummaryListQuery is called
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };
            var query = new AppointmentsManager.Application.Appointments.GetAppointmentSummaryListQuery.Query();
            var queryHandler = new
AppointmentsManager.Application.Appointments.GetAppointmentSummaryListQuery.Handler(appointmentStorage);
            var appointmentSummaryList = await queryHandler.Handle(query, CancellationToken.None);

            // then list of AppointmentSummary entries are returned
            Assert.IsTrue(appointmentSummaryList.Length == 1);
            Assert.IsTrue(appointmentSummaryList[0].Place == "Alaska");
        }

    }
}
```

The code

```
using System.Diagnostics.CodeAnalysis;
using System.Threading;
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;
using AppointmentsManager.Storage.Appointments;
using MediatR;

namespace AppointmentsManager.Application.Appointments
{
    public class GetAppointmentSummaryListQuery
    {
        [SuppressMessage("ReSharper", "ClassNeverInstantiated.Global")]
        public class Query: IRequest<AppointmentSummary[]> { }

        public class Handler : IRequestHandler<Query, AppointmentSummary[]>
        {
            private readonly IAppointmentStorage _appointmentStorage;

            public Handler(IAppointmentStorage appointmentStorage)
            {
                _appointmentStorage = appointmentStorage;
            }

            public async Task<AppointmentSummary[]> Handle(Query request, CancellationToken cancellationToken)
            {
                return await _appointmentStorage.GetAppointmentSummaries();
            }
        }
    }
}
```

Use case: Return empty list if no Appointment summaries are found

The test

```
[Test]
[Category("Integration")]
public async Task Returns_empty_list_when_no_appointments_are_available()
{
    // given there are no appointments
    var tempPath = Path.GetTempPath();
    var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

    var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };

    // when GetAppointmentSummaryListQuery is called
    var query = new AppointmentsManager.Application.Appointments.GetAppointmentSummaryListQuery.Query();
    var queryHandler = new
AppointmentsManager.Application.Appointments.GetAppointmentSummaryListQuery.Handler(appointmentStorage);
    var appointmentSummaryList = await queryHandler.Handle(query, CancellationToken.None);

    // then an empty list is returned
    Assert.IsTrue(appointmentSummaryList.Length == 0);

}
```

The code

Same code for previous use case, covers this one as well

Use case: Given appointment id, return appointment

The test

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class GetAppointmentQueryTest
    {
        [Test]
        [Category("Integration")]
        public async Task Returns_an_appointment()
        {
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

            // 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){ AppointmentCatalogueFileName = catalogueFileName
}.Save(appointment);

            // when GetAppointmentQuery is executed
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };
            var query = new AppointmentsManager.Application.Appointments.GetAppointmentQuery.Query { Id =
appointment.Id};
            var queryHandler = new
AppointmentsManager.Application.Appointments.GetAppointmentQuery.Handler(appointmentStorage);
            var result = await queryHandler.Handle(query, CancellationToken.None);

            // then appointment is returned
            Assert.IsNotNull(result);
            Assert.IsTrue(result.Id == appointment.Id);
            Assert.IsTrue(result.Place == appointment.Place);
            Assert.IsTrue(result.Comments == appointment.Comments);
            Assert.IsTrue(result.Description == appointment.Description);
        }
    }
}
```

The code

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;
using AppointmentsManager.Storage.Appointments;
using MediatR;

namespace AppointmentsManager.Application.Appointments
{
    public class GetAppointmentQuery
    {
        public class Query : IRequest<Appointment>
        {
            public string Id { get; set;}
        }

        public class Handler : IRequestHandler<Query, Appointment>
        {
```

```
            private readonly IAppointmentStorage _appointmentStorage;

            public Handler(IAppointmentStorage appointmentStorage)
            {
                _appointmentStorage = appointmentStorage;
            }

            public async Task<Appointment> Handle(Query request, CancellationToken cancellationToken)
            {
                Appointment result = null;

                var appointmentSummary = (await _appointmentStorage.GetAppointmentSummaries())
                    .FirstOrDefault(i => i.Id == request.Id);

                if (appointmentSummary != null)
                {
                    result = await _appointmentStorage.GetAppointment(request.Id);
                }

                return result;
            }
        }
    }
}
```

Use case: Given appointment id, return null if no matching appointment found

The test

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class GetAppointmentQueryTest
    {
        [Test]
        [Category("Integration")]
        public async Task Returns_an_appointment()
        {
          ...
        }

        [Test]
        [Category("Integration")]
        public async Task Returns_null_for_no_matching_appointment_id()
        {
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

            // given no appointment

            // when GetAppointmentQuery is executed with an arbitrary id
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };
            var query = new AppointmentsManager.Application.Appointments.GetAppointmentQuery.Query { Id = "123" };

            var queryHandler = new
AppointmentsManager.Application.Appointments.GetAppointmentQuery.Handler(appointmentStorage);
            var result = await queryHandler.Handle(query, CancellationToken.None);

            // then null is returned
            Assert.IsNull(result);
        }
    }
}
```

The code

Same code for previous use case, covers this one as well

User case: Add new appointment and return appointment id of newly created appointment

The test

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class CreateAppointmentCommandTest
    {
        [Test]
        [Category("Integration")]
        [SuppressMessage("ReSharper", "UseObjectOrCollectionInitializer")]
        public async Task Creates_appointment()
        {
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

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

            // when CreateAppointmentCommand is executed
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath)
                {AppointmentCatalogueFileName = catalogueFileName};
            var commandHandler =
                new AppointmentsManager.Application.Appointments.CreateAppointmentCommand.Handler(appointmentStorage);
            var appointmentId = await commandHandler.Handle(command, CancellationToken.None);

            // then appointment is created
            var createdAppointment = await new FileSystemAppointmentStorage(tempPath)
                {AppointmentCatalogueFileName = catalogueFileName}.GetAppointment(command.Id);
            Assert.IsNotNull(createdAppointment);

            // and then id of the appointment is returned
            Assert.IsTrue(appointmentId == createdAppointment.Id);
            Assert.IsTrue(command.Id == createdAppointment.Id);
        }
    }
}
```

The code

```
using System;
using System.Diagnostics.CodeAnalysis;
using System.Threading;
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;
using AppointmentsManager.Storage.Appointments;
using FluentValidation;
using MediatR;

namespace AppointmentsManager.Application.Appointments
{
    public class CreateAppointmentCommand
    {
        public class Command : IRequest<string>
        {
            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; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
```

```
            RuleFor(x => x.Place).NotEmpty();
        }
    }

    public class Handler : IRequestHandler<Command, string>
    {
        private readonly IAppointmentStorage _appointmentStorage;

        public Handler(IAppointmentStorage appointmentStorage)
        {
            _appointmentStorage = appointmentStorage;
        }

        [SuppressMessage("ReSharper", "UseObjectOrCollectionInitializer")]
        public async Task<string> Handle(Command command, CancellationToken cancellationToken)
        {
            var appointment = new Appointment();
            appointment.Id = command.Id ?? new Guid().ToString();
            appointment.Place = command.Place;
            appointment.Comments = command.Comments;
            appointment.Description = command.Description;
            appointment.DateTime = command.DateTime ?? DateTime.Now;

            await _appointmentStorage.Save(appointment);

            return appointment.Id;
        }
    }
}
```

Use case: Update an appointment

The test

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class UpdateAppointmentCommandTest
    {
        [Test]
        [Category("Integration")]
        [SuppressMessage("ReSharper", "UseObjectOrCollectionInitializer")]
        public async Task Creates_appointment()
        {
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

            // 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){ AppointmentCatalogueFileName = catalogueFileName
}.Save(appointment);

            // and given an updated appointment
            appointment.Place = "Colombo";

            // when UpdateAppointmentCommand is executed
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };
            var command = new AppointmentsManager.Application.Appointments.UpdateAppointmentCommand.Command();
            command.Id = appointment.Id;
            command.Place = "Colombo";
            command.Comments = appointment.Comments;
```

```
            command.Description = appointment.Description;
            command.DateTime = appointment.DateTime;
            var commandHandler = new
AppointmentsManager.Application.Appointments.UpdateAppointmentCommand.Handler(appointmentStorage);
            var result = await commandHandler.Handle(command, CancellationToken.None);

            // then appointment is updated
             var updatedAppointment = await appointmentStorage.GetAppointment(appointment.Id);
            Assert.IsTrue(updatedAppointment.Place == "Colombo");

            // then appointment-id is returned
             Assert.IsNotNull(result == appointment.Id);
        }
    }
}
```

The code

```
using System;
using System.Diagnostics.CodeAnalysis;
using System.Threading;
using System.Threading.Tasks;
using AppointmentsManager.Domain.Appointments;
using AppointmentsManager.Storage.Appointments;
using FluentValidation;
using MediatR;

namespace AppointmentsManager.Application.Appointments
{
    [SuppressMessage("ReSharper", "ClassNeverInstantiated.Global")]
    public class UpdateAppointmentCommand
    {
        public class Command: IRequest<string>
        {
            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; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.Id).NotEmpty();
                RuleFor(x => x.Place).NotEmpty();
            }
        }

        public class Handler : IRequestHandler<Command, string>
        {
            private readonly IAppointmentStorage _appointmentStorage;

            public Handler(IAppointmentStorage appointmentStorage)
            {
                _appointmentStorage = appointmentStorage;
            }

            [SuppressMessage("ReSharper", "UseObjectOrCollectionInitializer")]
            public async Task<string> Handle(Command command, CancellationToken cancellationToken)
            {
                var appointment = new Appointment();
                appointment.Id = command.Id;
                appointment.Place = command.Place;
                appointment.Comments = command.Comments;
                appointment.Description = command.Description;
                appointment.DateTime = command.DateTime ?? DateTime.Now;

                await _appointmentStorage.Save(appointment);

                return appointment.Id;
            }
        }
    }
}
```

Use case: Deletes an appointment

The test

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

namespace AppointmentsManager.Application.Tests.Appointments
{
    [TestFixture]
    public class DeleteAppointmentCommandTest
    {
        [Test]
        public async Task Deletes_appointment()
        {
            var tempPath = Path.GetTempPath();
            var catalogueFileName = $"{Guid.NewGuid()}_catalogue.json";

            // 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){ AppointmentCatalogueFileName = catalogueFileName
}.Save(appointment);

            // when DeleteAppointmentCommand is executed
            var appointmentStorage = new FileSystemAppointmentStorage(tempPath){ AppointmentCatalogueFileName =
catalogueFileName };
            var command = new AppointmentsManager.Application.Appointments.DeleteAppointmentCommand.Command();
            command.Id = appointment.Id;
            var commandHandler = new
AppointmentsManager.Application.Appointments.DeleteAppointmentCommand.Handler(appointmentStorage);
            await commandHandler.Handle(command, CancellationToken.None);

            // then appointment is deleted
            var deletedAppointment = await appointmentStorage.GetAppointment(appointment.Id);
            Assert.IsNull(deletedAppointment);
        }
    }
}
```

The code

```
using System.Threading;
using System.Threading.Tasks;
using AppointmentsManager.Storage.Appointments;
using FluentValidation;
using MediatR;

namespace AppointmentsManager.Application.Appointments
{
    public class DeleteAppointmentCommand
    {
        public class Command : IRequest
        {
            public string Id { get; set; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.Id).NotEmpty();
            }
        }

        public class Handler : IRequestHandler<Command>
        {
            private readonly IAppointmentStorage _appointmentStorage1;
```

```
            public Handler(IAppointmentStorage _appointmentStorage)
            {
                _appointmentStorage1 = _appointmentStorage;
            }

            public async Task<Unit> Handle(Command command, CancellationToken cancellationToken)
            {
                await _appointmentStorage1.Delete(command.Id);

                return Unit.Value;
            }
        }
    }
}
```

Web API

Configuration from environment

We will load configuration from environment variables

Initial setup

Updates launchSettings.json

launchSettings.json provides a way for us to tweek launch parameters as well as configure environment variables. Note I have swapped order of profiles to make `AppointmentsManager.WebAPI` the default profile

```
{
 "$schema": "http://json.schemastore.org/launchsettings.json",
 "iisSettings": {
   "windowsAuthentication": false,
   "anonymousAuthentication": true,
   "iisExpress": {
     "applicationUrl": "http://localhost:4897",
     "sslPort": 44398
   }
 },
 "profiles": {
   "AppointmentsManager.WebAPI": {
     "commandName": "Project",
     "launchBrowser": true,
     "launchUrl": "appointment",
     "applicationUrl": "http://localhost:5000",
     "environmentVariables": {
       "ASPNETCORE_ENVIRONMENT": "Development",
       "FILESYSTEM_APPOINTMENT_STORAGE_BASE_FOLDER": "C:/Users/DEVADMIN/AppData/Local/Temp/FileSystemApplicationStorage/"
     }
   },
   "IIS Express": {
     "commandName": "IISExpress",
     "launchBrowser": true,
     "launchUrl": "appointment",
     "environmentVariables": {
       "ASPNETCORE_ENVIRONMENT": "Development",
       "FILESYSTEM_APPOINTMENT_STORAGE_BASE_FOLDER": "C:/Users/DEVADMIN/AppData/Local/Temp/FileSystemApplicationStorage/"
     }
   }
 }
}
```

Activate environment variable loader

```
namespace AppointmentsManager.WebAPI
{
   public class Program
   {
       public static void Main(string[] args)
       {
           CreateHostBuilder(args).Build().Run();
       }

       public static IHostBuilder CreateHostBuilder(string[] args) =>
           Host.CreateDefaultBuilder(args)
               .ConfigureAppConfiguration((hostingContext, config) =>
               {
                   config.AddEnvironmentVariables();
               })
```

```
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Disable HTTP redirection while in development mode. In Startup.cs
Add CORS support
Add mediator support
Add JsonSoft support
Register services with DI

```
using AppointmentsManager.Storage.Appointments;
using MediatR;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AppointmentsManager.WebAPI
{
   public class Startup
   {
       public Startup(IConfiguration configuration)
       {
           Configuration = configuration;
       }

       public IConfiguration Configuration { get; }

       // This method gets called by the runtime. Use this method to add services to the container.
       public void ConfigureServices(IServiceCollection services)
       {
           {
               var basePath = Configuration["FILESYSTEM_APPOINTMENT_STORAGE_BASE_FOLDER"];
               services.AddScoped<IAppointmentStorage>((options) => new FileSystemAppointmentStorage(basePath));
           }

           services.AddMediatR(typeof(AppointmentsManager.Application.Appointments.CreateAppointmentCommand).Assembly);

           services.AddControllers().AddNewtonsoftJson();

           services.AddCors(options =>
           {
               options.AddPolicy("CorsPolicy",
                   policy => { policy.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin(); });
           });
       }

       // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
       public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
       {
           if (env.IsDevelopment())
           {
               app.UseDeveloperExceptionPage();
           }

           if (env.IsProduction())
           {
               app.UseHttpsRedirection();
           }

           app.UseRouting();

           app.UseAuthorization();

           app.UseCors("CorsPolicy");

           app.UseEndpoints(endpoints =>
           {
               endpoints.MapControllers();
           });
       }
   }
}
```

Lets implement our controller now

The interesting part is PATH request where user sends a JsonPatch request in, which we use to update model (https://docs.microsoft.com/en-us/aspnet/core/web-api/jsonpatch?view=aspnetcore-3.1)

```
cd ./AppointmentsManager.WebAPI/
rm ./Controllers/WeatherController.cs
```

```
touch ./Controllers/AppointmentController.cs
cd ..

using System.Diagnostics.CodeAnalysis;
using System.Threading.Tasks;
using AppointmentsManager.Application.Appointments;
using AppointmentsManager.Domain.Appointments;
using MediatR;
using Microsoft.AspNetCore.JsonPatch;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;

namespace AppointmentsManager.WebAPI.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class AppointmentController: ControllerBase
    {
        private readonly IMediator _mediator;

        public AppointmentController(IMediator mediator)
        {
            _mediator = mediator;
        }

        [HttpGet]
        public async Task<ActionResult<AppointmentSummary[]>> Get()
        {
            var result =
                await _mediator.Send(
                    new GetAppointmentSummaryListQuery.Query());

            return Ok(result);
        }

        [HttpPost]
        public async Task<ActionResult<string>> Post(CreateAppointmentCommand.Command command)
        {
            var result =
                await _mediator.Send(command);

            return Ok(result);
        }

        [HttpPut]
        public async Task<ActionResult<string>> Put(UpdateAppointmentCommand.Command command)
        {
            var result =
                await _mediator.Send(command);

            return Ok(result);
        }

        [HttpPatch]
        [Route("{id}")]
        [SuppressMessage("ReSharper", "InvertIf")]
        [SuppressMessage("ReSharper", "UseObjectOrCollectionInitializer")]
        public async Task<ActionResult<string>> Patch(
            [FromBody] JsonPatchDocument<UpdateAppointmentCommand.Command> commandPatch, string id)
        {
            var appointment = await  mediator.Send(new GetAppointmentQuery.Query {Id = id});
            var appointment = await _mediator.Send(new GetAppointmentQuery.Query {Id = id});

            if (appointment != null)
            {
                var command = new UpdateAppointmentCommand.Command();
                command.Id = id;
                command.Comments = appointment.Comments;
                command.Place = appointment.Place;
                command.Description = appointment.Description;
                command.DateTime = appointment.DateTime;

                commandPatch.ApplyTo(command,ModelState);

                if (ModelState.IsValid)
                {
                    await _mediator.Send(command);
                }
                else
                {
                    var serializableModelState = new SerializableError(ModelState);
                    var modelStateJson = JsonConvert.SerializeObject(serializableModelState);

                    return BadRequest(modelStateJson);
                }

                return Ok(id);
```

```
            }

            return NotFound();
        }

        [HttpDelete]
        [Route("{id}")]
        public async Task<ActionResult> Delete(string id)
        {
            var command = new DeleteAppointmentCommand.Command { Id = id } ;
            await _mediator.Send(command);

            return NoContent();
        }
    }
}
```