Software Design Document (SDD)

Doctor Appointment Booking

Revision History

Version	Date	Changes
1.0	25/04/2025	Initial draft (Modular Monolith)

1. Introduction 3

  1.1 Purpose 3

  1.2 Scope 3

2. System Overview 3

  2.1 System Architecture 3

  2.2 High-Level Diagram 7

3. Module Design 9

  3.1 Doctor Availability Module 9

  3.2 Appointment Booking Module 11

  3.3 Doctor Appointment Management Module 13

  3.4 Notification Module 15

  3.5 Shared Module 16

  3.5 Host Module 16

4. Testing Approach 17

  7.1 Doctor Availability Module 17

  7.2 Appointment Booking Module 18

5. Deployment 19

1. Introduction

1.1 Purpose

This document outlines the architectural design of the Doctor Appointment Booking System, detailing system structure, components, and key functionalities. The solution establishes a robust backend system to facilitate medical appointment scheduling, with emphasis on API development and business requirement fulfillment.

1.2 Scope

This System Design Document provides:

- Architectural overview and component diagrams
- Module descriptions
- Testing Approach
- Deployment

2. System Overview

2.1 System Architecture

In this section, we describe the system's high-level architecture, covering the chosen architectural style, the development phases, and the main components of the system.

Architectural Patterns Development phases

This system serves as a practical assessment for implementing various architectural styles. The development will progress through four distinct phases:

➢ Phase 1: Modular Monolith Implementation

This phase focus on implementing modular monolith with different architecture patterns like Layered Architecture,Hexagonal Architecture and Clean Architecture.

➢ Phase 2: Domain-Driven Design (DDD) Integration

This phase to refactor the monolith using DDD principles.

➢ Phase 3: Microservices Implementation

This phase to decompose the monolith into microservices

➢ Phase 4: Containerized Deployment

This phase to package services using Docker containers and Orchestrate deployment with Kubernetes (K8s).

Key Components

In this section, we outline the system's core components. Since this is a backend system, we focus on the backend technologies and data persistence methods.

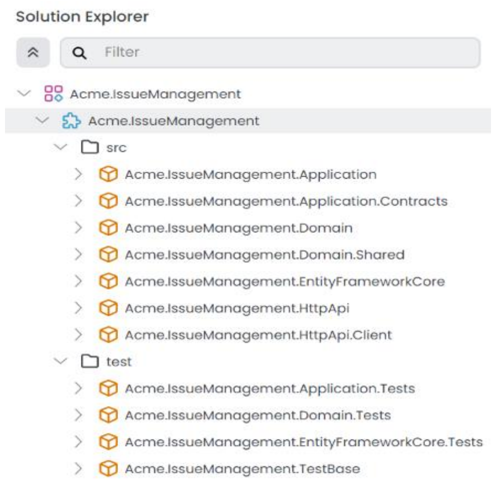
➢ Backend (API/Server):

This system was built using ASP.NET Boilerplate (ABP). Having worked with ABP on previous projects, I wanted to explore different implementation patterns using this framework. Below is a brief overview of ABP and the project structure:

ASP.NET Boilerplate (ABP) offers various startup templates designed for different application requirements. Each template provides a production-ready .NET solution foundation, enabling developers to quickly begin application development. For this project, we utilized the following ABP templates:

- **Single-Layer Solution:** Creates a single-project solution. Recommended for building an application with a simpler and easy to understand architecture. *Implemented in the host module for its clean, uncomplicated structure*
- **Application (Layered)**: A fully layered (multiple projects) solution based on Domain Driven Design practices. Used as the base template for all modules with Customization to accommodate each module's specific architectural needs. We will provide a brief description of its layered structure below.

When you create a new layered solution, you will see a tree structure similar to the one below in the Solution Explorer panel:

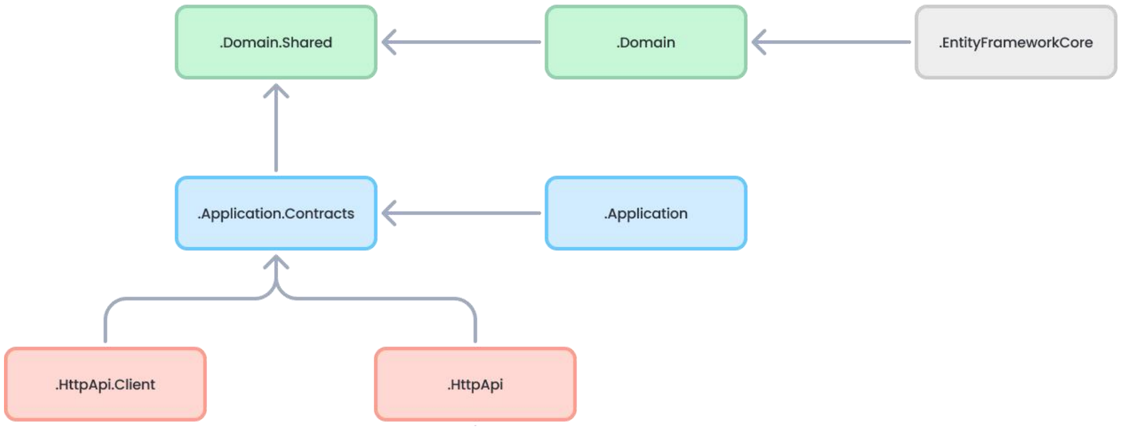


```
Solution Explorer
Filter
Acme.IssueManagement
  Acme.IssueManagement
    src
      Acme.IssueManagement.Application
      Acme.IssueManagement.Application.Contracts
      Acme.IssueManagement.Domain
      Acme.IssueManagement.Domain.Shared
      Acme.IssueManagement.EntityFrameworkCore
      Acme.IssueManagement.HttpApi
      Acme.IssueManagement.HttpApi.Client
    test
      Acme.IssueManagement.Application.Tests
      Acme.IssueManagement.Domain.Tests
      Acme.IssueManagement.EntityFrameworkCore.Tests
      Acme.IssueManagement.TestBase
```

Projects are organized as ***src*** and ***test*** folders:

- ***src*** folder contains the actual module which is layered.
- ***test*** folder contains unit & integration test

The diagram below illustrates the application's layers and project dependencies:



.Domain.Shared Project

- ✓ This project contains constants, enums and other objects these are actually a part of the domain layer, but needed to be used by all layers/projects in the solution.
- ✓ This project has no dependency on other projects in the solution. All other projects depend on this one directly or indirectly

.Domain Project

- ✓ This is the domain layer of the solution. It mainly contains entities, aggregate roots, domain services, value objects, repository interfaces and other domain objects.
- ✓ Depends on the .Domain.Shared because it uses constants, enums and other objects defined in that project

.Application.Contracts Project

- ✓ This project mainly contains application service interfaces and Data Transfer Objects (DTO) of the application layer. It exists to separate the interface & implementation of the application layer. In this way, the interface project can be shared to the clients as a contract package.
- ✓ Depends on the .Domain.Shared because it may use constants, enums and other shared objects of this project in the application service interfaces and DTOs.

.Application Project

- ✓ This project contains the application service implementations of the interfaces defined in the .Application.Contracts project.
- ✓ Depends on the .Application.Contracts project to be able to implement the interfaces and use the DTOs. Depends on the .Domain project to be able to use domain objects (entities, repository interfaces... etc.) to perform the application logic.

.EntityFrameworkCore Project

- ✓ This is the integration project for the EF Core. It defines the DbContext and implements repository interfaces defined in the .Domain project.
- ✓ Depends on the .Domain project to be able to reference to entities and repository interfaces.

.DbMigrator Project

- ✓ This is a console application that simplifies the execution of database migrations on development and production environments. When you run this application, it:

  i. Creates the database if necessary.
  ii. Applies the pending database migrations.
  iii. Seeds initial data if needed

.HttpApi Project

- ✓ This project is used to define your API Controllers.Most of the time you don't need to manually define API Controllers since ABP's Auto API Controllers feature creates them automagically based on your application layer. However, in case of you need to write API controllers, this is the best place to do it.
- ✓ Depends on the .Application.Contracts project to be able to inject the application service interfaces.

.HttpApi.Client Project

- ✓ This is a project that defines C# client proxies to use the HTTP APIs of the solution. You can share this library to 3rd-party clients, so they can easily consume your HTTP APIs in their Dotnet applications (For other types of applications, they can still use your APIs, either manually or using a tool in their own platform)
- ✓ Most of the time you don't need to manually create C# client proxies, .HttpApi.Client.ConsoleTestApp project is a console application created to demonstrate the usage of the client proxies.

✓ Depends on the .Application.Contracts project to be able to share the same application service interfaces and DTOs with the remote service.

The above explanation briefly described the layered architecture used in most modules and for more details you can check ABP documentation. The Module Design sections will demonstrate how we customized this architecture for different patterns

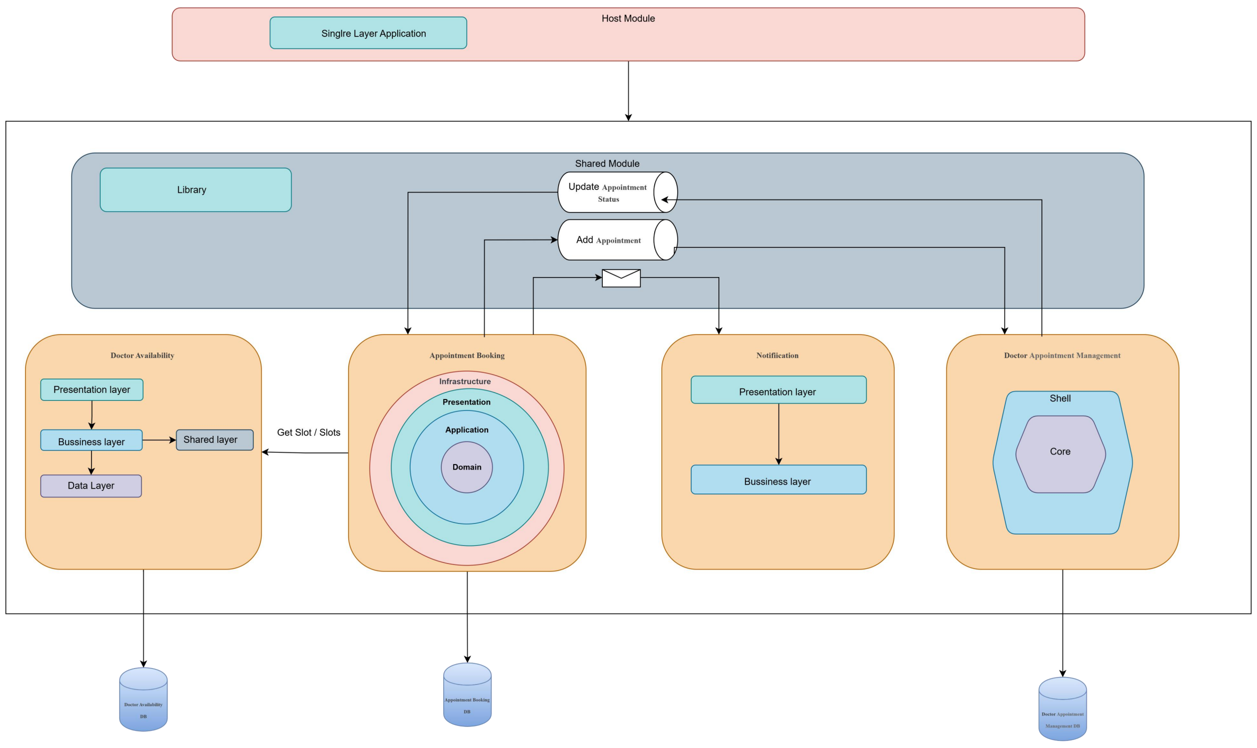
➢ Data Persistence:

Since the data persistence approach was flexible in this system, a relational database was selected to evaluate its compatibility and challenges within a microservices architecture. Given that the system was built using the .NET stack, SQL Server was chosen as the database for seamless integration.

2.2 High-Level Diagram

Modular Monolith Implementation

The below diagram show the high level implementation of Modular Monolith:



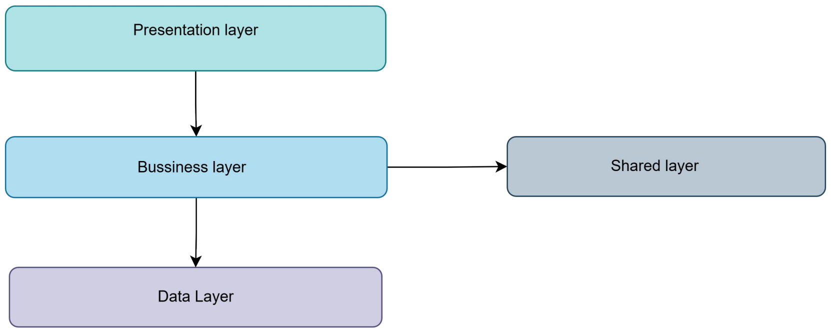
Microservices Implementation

To be implemented later

3. Module Design

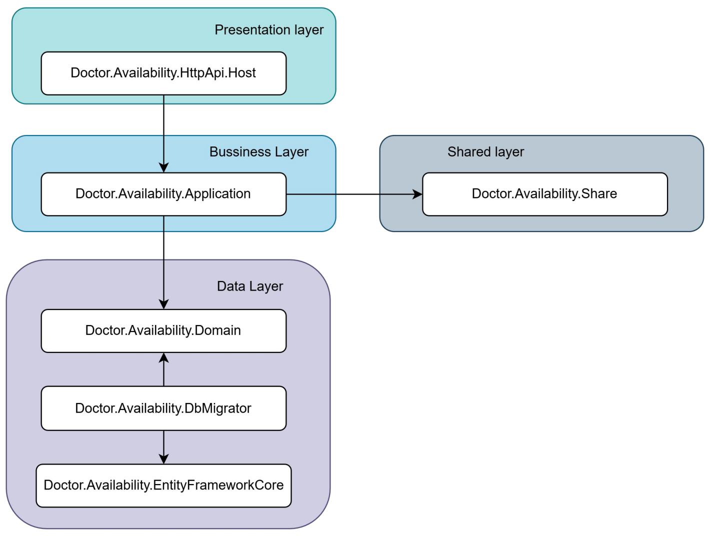
3.1 Doctor Availability Module

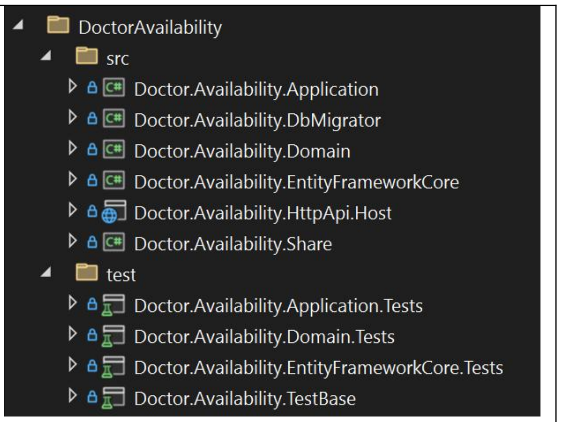
- **Purpose**: this module to help doctor to manage slots.
- **Functions**:
  - **Function A**: As a doctor, I want to be able to list my slots
  - **Function B**: As a doctor, I want to be able to add new slots where a single time slot
- **Architecture Pattern :** Layered architecture



- **Project structure:**

The following diagram illustrates how we adapted the layered architecture template's projects to implement the layered architecture .


Presentation layer
Doctor.Availability.HttpApi.Host
Bussiness Layer
Doctor.Availability.Application
Shared layer
Doctor.Availability.Share
Data Layer
Doctor.Availability.Domain
Doctor.Availability.DbMigrator
Doctor.Availability.EntityFrameworkCore

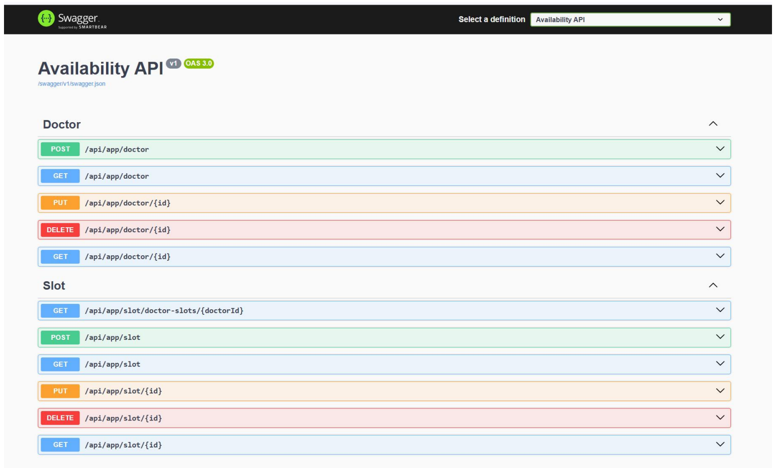


```
DoctorAvailability
  src
    Doctor.Availability.Application
    Doctor.Availability.DbMigrator
    Doctor.Availability.Domain
    Doctor.Availability.EntityFrameworkCore
    Doctor.Availability.HttpApi.Host
    Doctor.Availability.Share
  test
    Doctor.Availability.Application.Tests
    Doctor.Availability.Domain.Tests
    Doctor.Availability.EntityFrameworkCore.Tests
    Doctor.Availability.TestBase
```

- **Dependencies**: This module doesn't depend on other module

- **API Endpoints:**

Method	Endpoint	Description
GET	/api/app/slot/doctor-slots/{doctorId}	Get Doctor available slots
POST	/api/app/slot	Add new Slot



Availability API v1 OAS 3.0
/swagger/v1/swagger.json

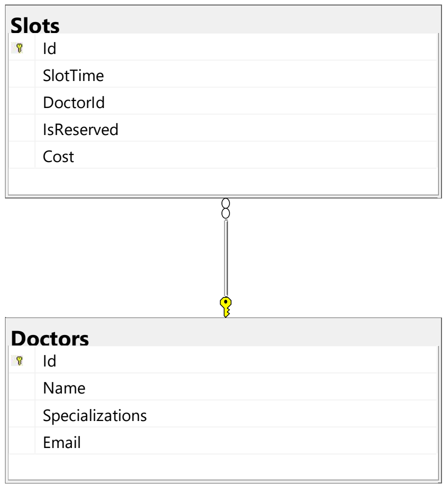
Doctor

POST	/api/app/doctor
GET	/api/app/doctor
PUT	/api/app/doctor/{id}
DELETE	/api/app/doctor/{id}
GET	/api/app/doctor/{id}

Slot

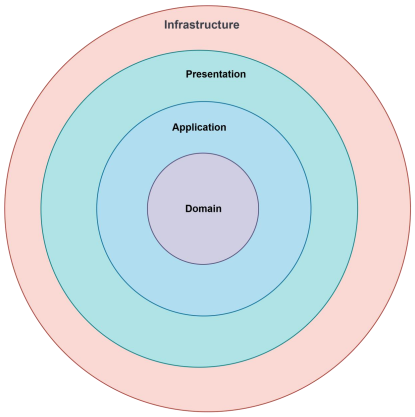
GET	/api/app/slot/doctor-slots/{doctorId}
POST	/api/app/slot
GET	/api/app/slot
PUT	/api/app/slot/{id}
DELETE	/api/app/slot/{id}
GET	/api/app/slot/{id}

- **Database Schema**
  - ✓ **Tables**:
    - ■ Slots(id, SlotTime, DoctorId, IsReserved, Cost)
    - ■ Doctors(id, Name, Specialization, Email)
  - ✓ **ERD**



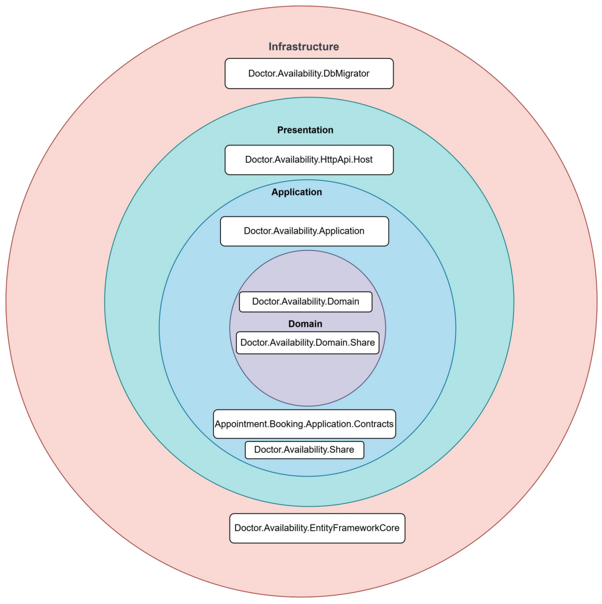
3.2 Appointment Booking Module

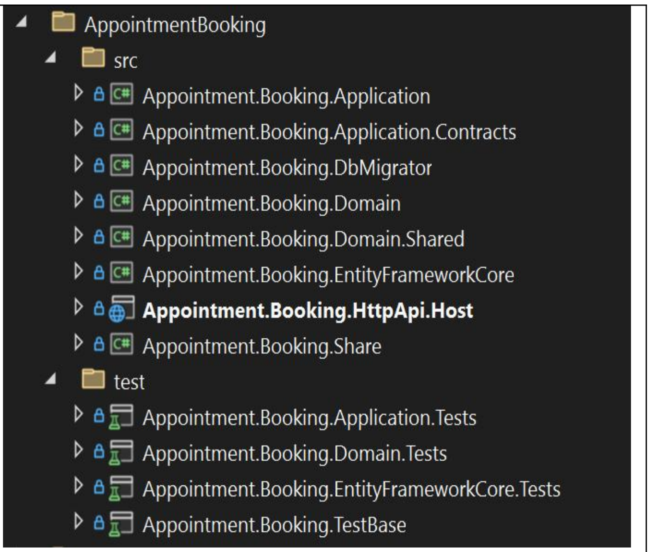
- **Purpose**: This module to help patient to book appointment.
- **Functions**:
    a. **Function A**: As a Patient, I want to be able to view all doctors' available (only) slots
    ○ **Function B**: As a patient, I want to be able to book an appointment in a free slot.
- **Architecture Pattern :** Clean architecture



- **Project structure:**

The following diagram illustrates how we adapted the layered architecture template's projects to implement the Clean architecture .

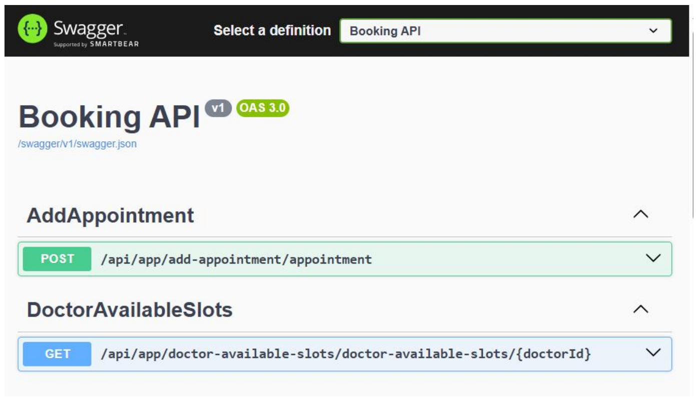

Infrastructure
Doctor.Availability.DbMigrator
Presentation
Doctor.Availability.HttpApi.Host
Application
Doctor.Availability.Application
Doctor.Availability.Domain
Domain
Doctor.Availability.Domain.Share
Appointment.Booking.Application.Contracts
Doctor.Availability.Share
Doctor.Availability.EntityFrameworkCore



```
AppointmentBooking
  src
    Appointment.Booking.Application
    Appointment.Booking.Application.Contracts
    Appointment.Booking.DbMigrator
    Appointment.Booking.Domain
    Appointment.Booking.Domain.Shared
    Appointment.Booking.EntityFrameworkCore
    Appointment.Booking.HttpApi.Host
    Appointment.Booking.Share
  test
    Appointment.Booking.Application.Tests
    Appointment.Booking.Domain.Tests
    Appointment.Booking.EntityFrameworkCore.Tests
    Appointment.Booking.TestBase
```

- **Dependencies**: This module depends on the below modules
    - Doctor Availability Module: To retrieve a doctor's available time slots and ensure a slot is open before booking.
    - Shared module: To raise an event for the notification module to send book appointment confirmation email and raise created appointment event.
- **API Endpoints:**

Method	Endpoint	Description
POST	/api/app/add-appointment/appointment	Book appointment
GET	/api/app/doctor-available-slots/doctor-available-slots/{doctorId}	Get Doctor available appoinment

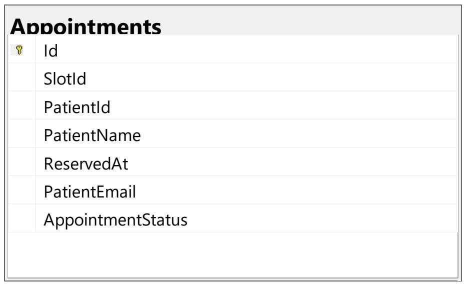


Booking API v1 OAS 3.0
/swagger/v1/swagger.json

AddAppointment
POST /api/app/add-appointment/appointment

DoctorAvailableSlots
GET /api/app/doctor-available-slots/doctor-available-slots/{doctorId}

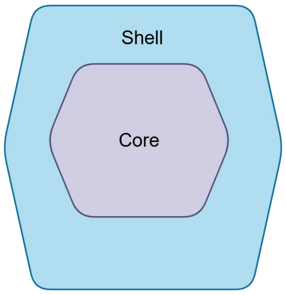
- **Database Schema**
    - ✓ **Tables**:
        - ■ Appoinments(id, SlotId, PatientId, PatientName,ReservedAt, PatientEmail, AppoinmentStatus)
    - ✓ **ERD**



Appointments
Id
SlotId
PatientId
PatientName
ReservedAt
PatientEmail
AppointmentStatus

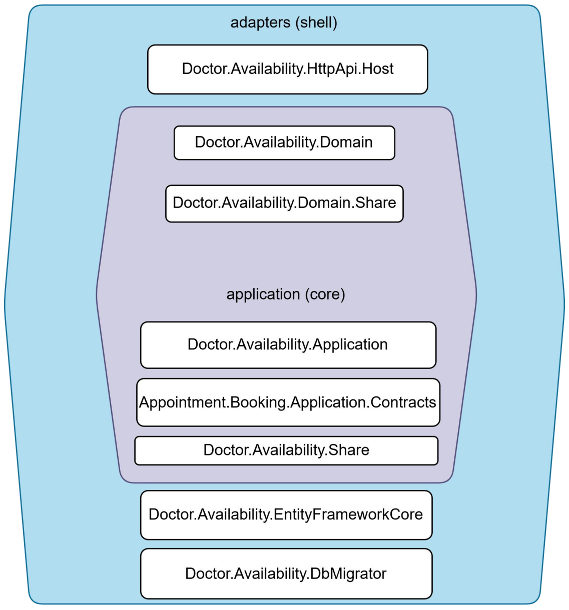
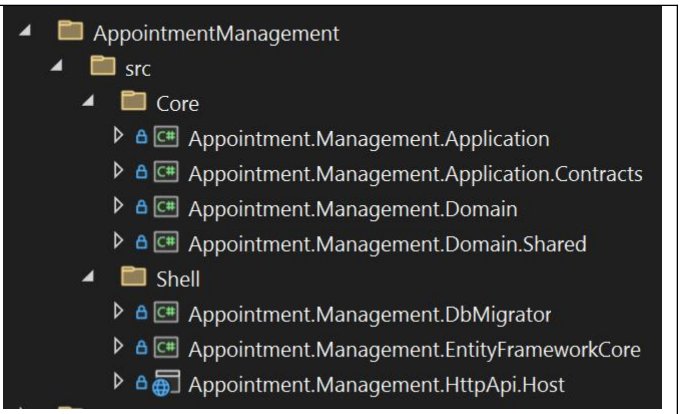
3.3 Doctor Appointment Management Module

- **Purpose**: This module to help doctor to manage appointment
- **Functions**:
  - **Function A**: As a Doctor, I want to be able to view my upcoming appointments
  - **Function B**: As a Doctor, I want to be able to mark appointments as completed or cancel them if necessary.
- **Architecture Pattern :** Hexagonal architecture



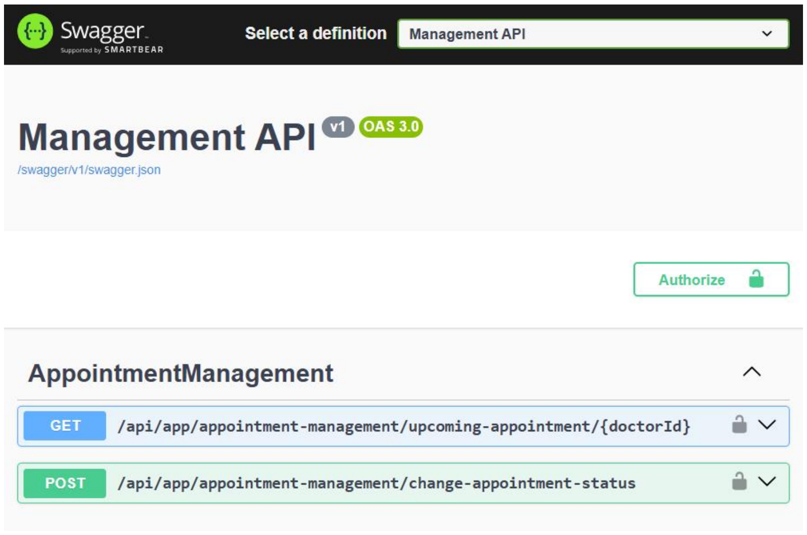
- **Project structure:**

The following diagram illustrates how we adapted the layered architecture template's projects to implement the Hexagonal architecture .

 adapters (shell) Doctor.Availability.HttpApi.Host Doctor.Availability.Domain Doctor.Availability.Domain.Share application (core) Doctor.Availability.Application Appointment.Booking.Application.Contracts Doctor.Availability.Share Doctor.Availability.EntityFrameworkCore Doctor.Availability.DbMigrator	 AppointmentManagement src Core Appointment.Management.Application Appointment.Management.Application.Contracts Appointment.Management.Domain Appointment.Management.Domain.Shared Shell Appointment.Management.DbMigrator Appointment.Management.EntityFrameworkCore Appointment.Management.HttpApi.Host

- **Dependencies**: This module depends on
  - Doctor Availability Module: To retrieve a doctor's upcoming slots.
  - Shared module: To raise an event to update appointment status and to listen to create appointment event.
- **API Endpoints:**

Method	Endpoint	Description
Get	/api/app/appointment-management/upcoming-appointment/{doctorId}	Get doctor upcoming appointment
Post	/api/app/appointment-management/change-appointment-status	Cancel or complete appointment



Management API

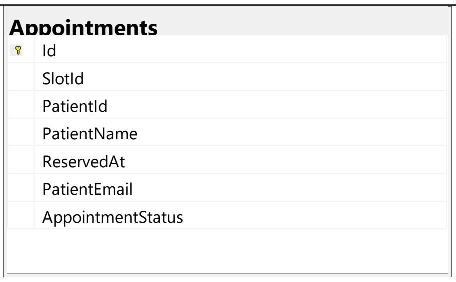
/swagger/v1/swagger.json

AppointmentManagement

GET /api/app/appointment-management/upcoming-appointment/{doctorId}

POST /api/app/appointment-management/change-appointment-status

- **Database Schema**
  - ✓ **Tables**:
    - ■ Appoinments(id, SlotId, PatientId, PatientName,ReservedAt, PatientEmail, AppoinmentStatus)
  - ✓ **ERD**

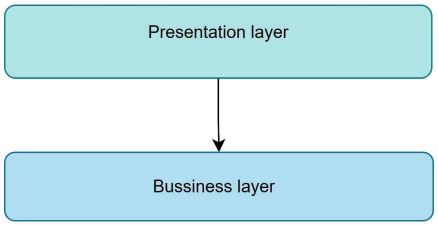


Appointments

Id
SlotId
PatientId
PatientName
ReservedAt
PatientEmail
AppointmentStatus

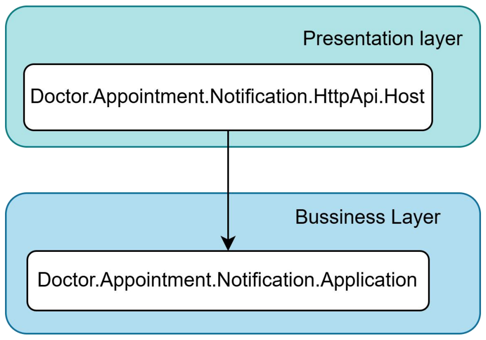
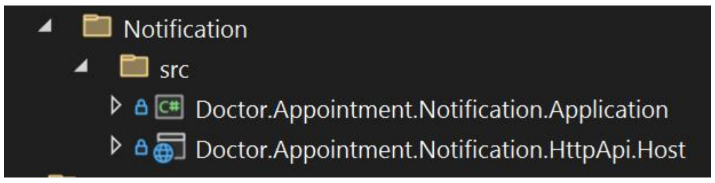
3.4 Notification Module

- **Purpose**: This module Send notifications.
- **Functions**:
    - **Function A**: Send Email to user/s
- **Architecture Pattern :** simple Layered architecture



- **Project structure:**

The following diagram illustrates how we adapted the simple layered architecture template's projects to implement the layered architecture .

 Presentation layer Doctor.Appointment.Notification.HttpApi.Host Bussiness Layer Doctor.Appointment.Notification.Application	

```
Notification
  src
    Doctor.Appointment.Notification.Application
    Doctor.Appointment.Notification.HttpApi.Host
```

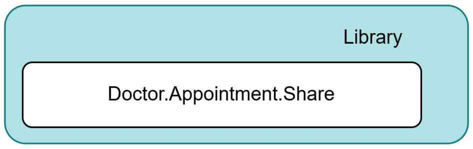
- **Dependencies**: This module depends on shared module to listen to notification event.

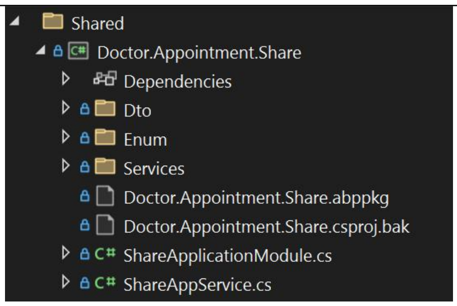
3.5 Shared Module

- **Purpose**: This module handle shared functionality between modules
- **Functions**:
  - **Function A**: Raise events
- **Architecture Pattern :** Simple Library

- **Project structure:**

The following diagram show simple shared library





```
Shared
  Doctor.Appointment.Share
    Dependencies
    Dto
    Enum
    Services
    Doctor.Appointment.Share.abppkg
    Doctor.Appointment.Share.csproj.bak
    ShareApplicationModule.cs
    ShareAppService.cs
```

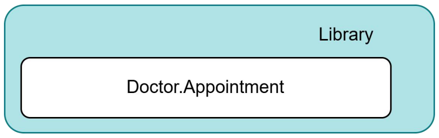
- **Dependencies**: This module doesn't depend on any module.

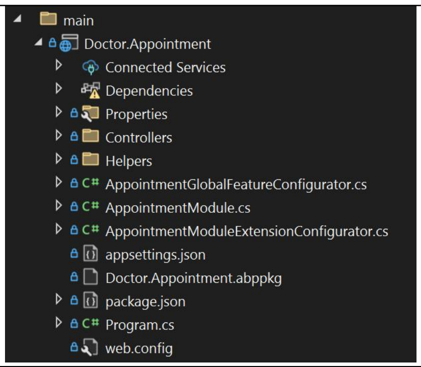
3.5 Host Module

- **Purpose**: This module use to be able to lunch all modules together
- **Architecture Pattern :** Single layer

- **Project structure:**

The following diagram show simple shared library





```
main
  Doctor.Appointment
    Connected Services
    Dependencies
    Properties
    Controllers
    Helpers
    AppointmentGlobalFeatureConfigurator.cs
    AppointmentModule.cs
    AppointmentModuleExtensionConfigurator.cs
    appsettings.json
    Doctor.Appointment.abppkg
    package.json
    Program.cs
    web.config
```

- **Dependencies**: This module depend on all modules.

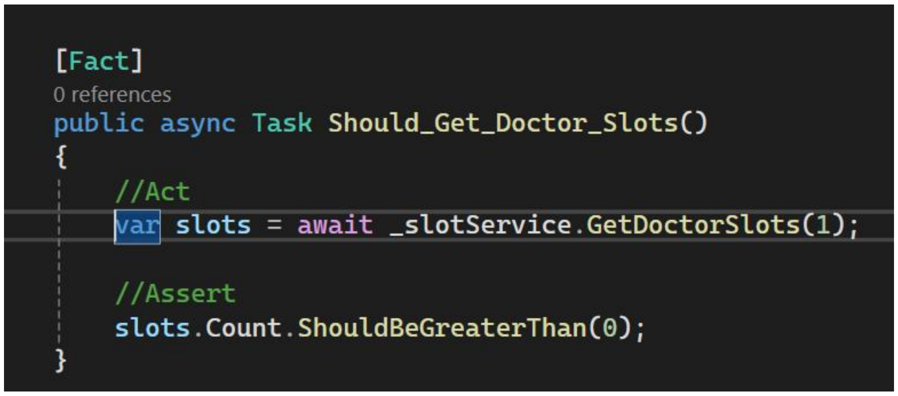
4. Testing Approach

The testing strategy combines unit and integration tests, which will be progressively implemented during each development phase

7.1 Doctor Availability Module

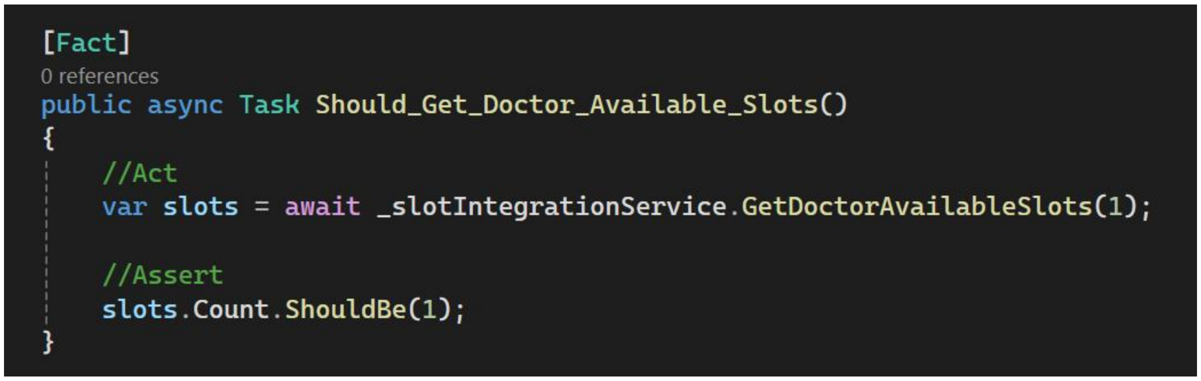
The following unit tests have currently been implemented in the application layer:

- Get Doctor all slots



```
[Fact]
0 references
public async Task Should_Get_Doctor_Slots()
{
    //Act
    var slots = await _slotService.GetDoctorSlots(1);

    //Assert
    slots.Count.ShouldBeGreaterThan(0);
}
```

- Get Doctor available slots



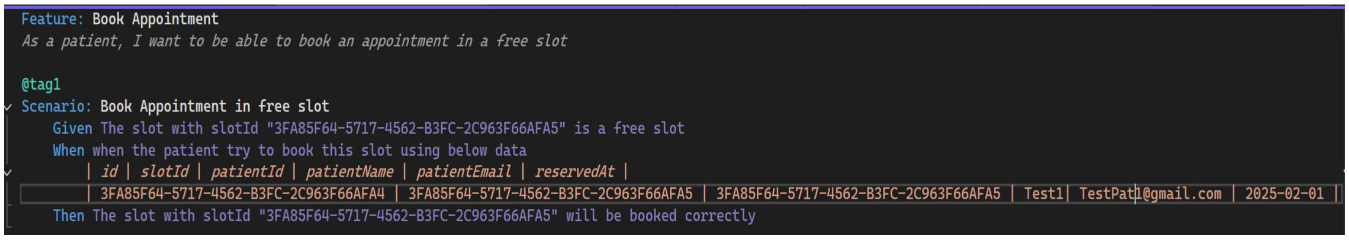
```
[Fact]
0 references
public async Task Should_Get_Doctor_Available_Slots()
{
    //Act
    var slots = await _slotIntegrationService.GetDoctorAvailableSlots(1);

    //Assert
    slots.Count.ShouldBe(1);
}
```

7.2 Appointment Booking Module

he following unit tests have currently been implemented using BDD in the domain service module:

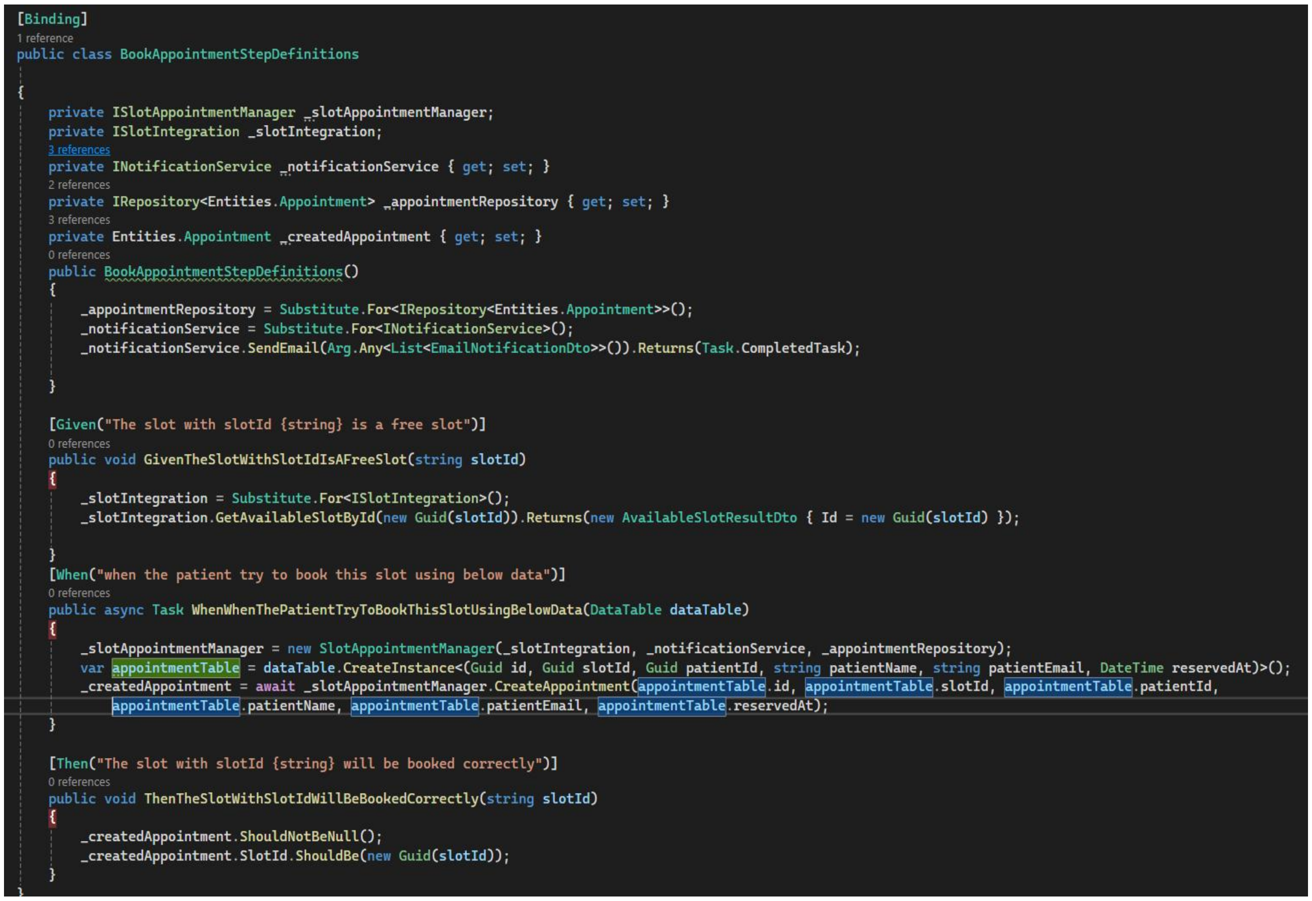
➢ Book new apponment

Feature class



```
Feature: Book Appointment
As a patient, I want to be able to book an appointment in a free slot

@tag1
Scenario: Book Appointment in free slot
    Given The slot with slotId "3FA85F64-5717-4562-B3FC-2C963F66AFA5" is a free slot
    When when the patient try to book this slot using below data
        | id | slotId | patientId | patientName | patientEmail | reservedAt |
        | 3FA85F64-5717-4562-B3FC-2C963F66AFA4 | 3FA85F64-5717-4562-B3FC-2C963F66AFA5 | 3FA85F64-5717-4562-B3FC-2C963F66AFA5 | Test1| TestPat1@gmail.com | 2025-02-01 |
    Then The slot with slotId "3FA85F64-5717-4562-B3FC-2C963F66AFA5" will be booked correctly
```

Step Definition



```
[Binding]
1 reference
public class BookAppointmentStepDefinitions

{
    private ISlotAppointmentManager _slotAppointmentManager;
    private ISlotIntegration _slotIntegration;
    3 references
    private INotificationService _notificationService { get; set; }
    2 references
    private IRepository<Entities.Appointment> _appointmentRepository { get; set; }
    3 references
    private Entities.Appointment _createdAppointment { get; set; }
    0 references
    public BookAppointmentStepDefinitions()
    {
        _appointmentRepository = Substitute.For<IRepository<Entities.Appointment>>();
        _notificationService = Substitute.For<INotificationService>();
        _notificationService.SendEmail(Arg.Any<List<EmailNotificationDto>>()).Returns(Task.CompletedTask);

    }

    [Given("The slot with slotId {string} is a free slot")]
    0 references
    public void GivenTheSlotWithSlotIdIsAFreeSlot(string slotId)
    {
        _slotIntegration = Substitute.For<ISlotIntegration>();
        _slotIntegration.GetAvailableSlotById(new Guid(slotId)).Returns(new AvailableSlotResultDto { Id = new Guid(slotId) });

    }
    [When("when the patient try to book this slot using below data")]
    0 references
    public async Task WhenWhenThePatientTryToBookThisSlotUsingBelowData(DataTable dataTable)
    {
        _slotAppointmentManager = new SlotAppointmentManager(_slotIntegration, _notificationService, _appointmentRepository);
        var appointmentTable = dataTable.CreateInstance<(Guid id, Guid slotId, Guid patientId, string patientName, string patientEmail, DateTime reservedAt)>();
        _createdAppointment = await _slotAppointmentManager.CreateAppointment(appointmentTable.id, appointmentTable.slotId, appointmentTable.patientId,
            appointmentTable.patientName, appointmentTable.patientEmail, appointmentTable.reservedAt);
    }

    [Then("The slot with slotId {string} will be booked correctly")]
    0 references
    public void ThenTheSlotWithSlotIdWillBeBookedCorrectly(string slotId)
    {
        _createdAppointment.ShouldNotBeNull();
        _createdAppointment.SlotId.ShouldBe(new Guid(slotId));
    }
}
```

7.3 Doctor Appointment Management Module

To be implemented later

7.4 Notification Module

To be implemented later

7.5 Shared Module

To be implemented later

5. Deployment

Will be handled in the last phase