DR.TechnicalInterview

## Exercise

### Introduction

This exercise is designed around completing a series of tickets which centre on the PDR.TechnicalInterview solution. As you complete these tickets work under the assumption that the system is live and actively used by clients - as a result making breaking changes is something to avoid.

We’d like you to submit your work as a github repository (or other public source control provider repository) where possible. As you complete the exercise we’d like you to commit regularly, with appropriate commit messages so that we can gain an understanding of the process you used to reach your eventual solution.

Prior to modifying the code, please pull the solution and try running it locally. The system is far from perfect but it should run and you should find you’re able to hit endpoints through the swagger UI without problem. If this is not the case please get in touch as soon as possible.

The tickets you need to complete are below, there is a solution overview as well to help you get started - good luck!

#### Ticket 1

**What:** Fix any failing tests in the solution

**Why:** The CI/CD pipeline is currently blocked from release as the master branch contains failing tests.

#### Ticket 2

**What:** Currently an endpoint exists for adding a new booking to the system. This endpoint is hit by patients when they come to book an appointment.

Each booking links a doctor to a patient and sets a time for their meeting together. At present there’s no time based validation on this endpoint and so multiple patients are able to book an appointment with the same doctor at the same time. In addition patients can book appointments in the past. Add validation checks on the endpoint to ensure patients can’t book appointments in the past, and to ensure that a patient can’t book an appointment with a doctor who is already busy.

**Why:** Improve patient experience by ensuring that a patient can’t book an appointment that they’ve already missed. Improve doctor utilisation by ensuring a doctor can’t be scheduled to see two patients at the same time.

#### Ticket 3

**What:** Patients want to be able to cancel appointments. Add a new endpoint which allows a patient to cancel appointments. Update the GetPatientNextAppointment endpoint to show the next uncancelled appointment. Update other parts of the system as required to handle cancelled appointments in a sensible way.

## **Why:** Patients want to be able to cancel appointments

## 

## System Overview

The PDR.Technical.Interview solution contains a simple API used for booking online patient appointments. This document is designed to help familiarise you with the system and introduce some terminology, packages and quirks you may be unfamiliar with.

### Terminology

#### **Patient**

Patients are the primary end users of the system. In a fully developed system, the patient would be the individual who enters into a video consultation with a doctorin order to receive treatment. Each patient in the system must have a unique email address.

#### **Doctor**

Doctors use the system in order to consult with patients. A patient will book an appointment to see a specific doctor using the API. When it’s time for their appointment, they will consult with the doctor online. Each doctor in the system must have a unique email address.

#### **Order**

An order is synonymous with a booking. When a patient books an appointment, an order is created. This order contains details of the appointment start and end time and also links a patient to a doctor.

#### **Clinic**

A clinic represents the real-world GP that a given patient is registered with.

#### **SurgeryType**

Clinics will be using one of two systems to manage their patients and prescriptions. For a doctor to consult with a patient, they need to be using the same system as the patient’s registered clinic. The two SurgeryTypes are names SystemOne and SystemTwo within the PDR.TechnicalInterview solution.

## 

### Packages

#### **NUnit**

Used as the unit testing framework for this solution.

#### **Moq**

Popular mocking framework

#### **FluentAssertions**

Provides a set of extension methods to let you more naturally specify the expected outcome of a unit test:



#### **AutoFixture**

Tool to automate non-relevant Test Fixture Setup. Models can be automatically populated with random values through easy syntax. Creating a value using a fixture strongly indicates that the value in question is irrelevant to the outcome of a test. This can aid readability. The following code creates an AddPatientRequest with randomised values  
  


#### **Entity Framework Core**

EF core is used for data access throughout the solution

### Other quirks

* An in memory database is used for this solution. Updates to the data will not persist between runs.
* For your convenience, data is seeded into the database at the start of each application run (See PDR.PatientBooking.Data.DataSeed)
* If you run the application a Swagger doc will load. You can use this to easily try the various endpoints in-browser