DocApp

-Shreyas B, Nithin B

Detailed Description of the Project (Documentation)

Table of content

1. Introduction:

1.1 Background

DocApp provides an Online Doctor Appointment system for the users to book appointments.

1.2 Purpose

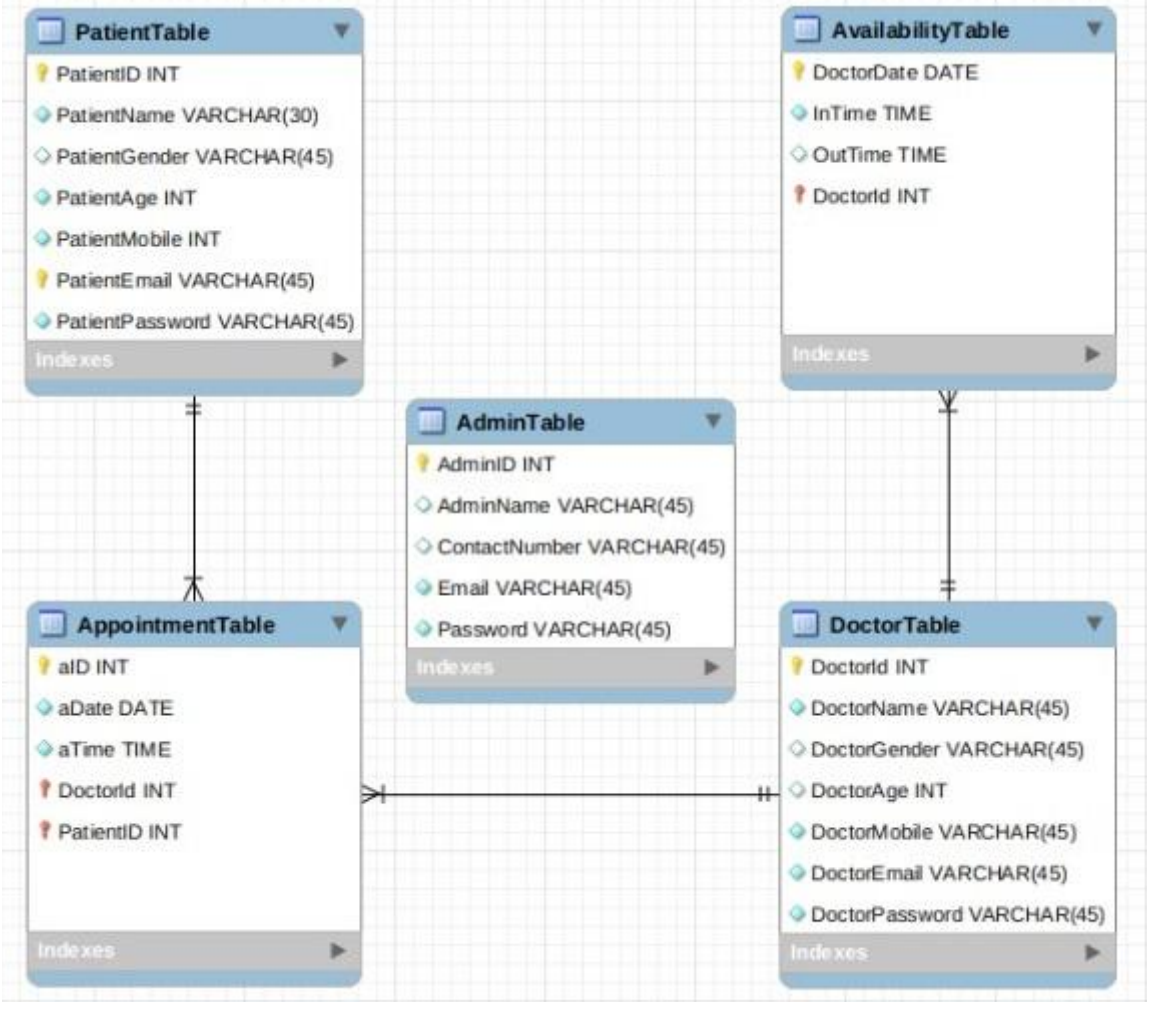
DocApp plans to develop “Online Appointment System” - a CLI application where patients will be able to fill in their details just and easily set up an appointment with the doctors based on a time-slot.

1.3 Scope

The scope of the DocApp will be to provide the functionality as described in the **Functional Requirements document**. The system will be developed on a Windows 10/Linux machine using Java, JDBC and MySQL.

2. Global Data Structures and Shared Data Functions

This section describes the structure of n tables to be used for the implementation of requirements as stated in the specification. Refer APPENDIX for detailed table structure.



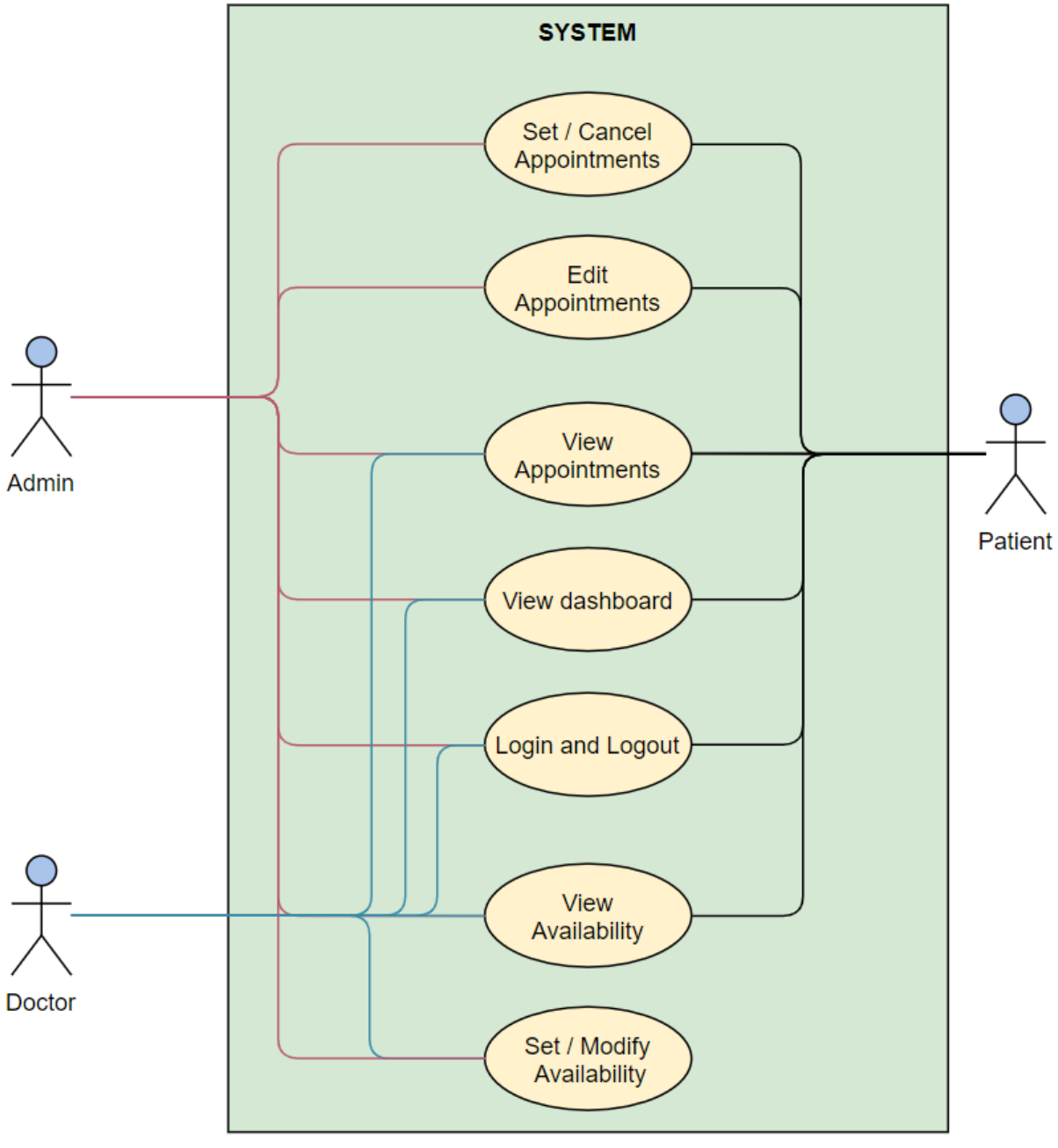
3. High Level Design

This section describes the high level design diagrams Use case diagram with Use Case definition, Sequence Diagram and Class Diagram which provides a visual representation of the requirements , logical flow and their class representations.

3.1. *Use Case Diagrams*

The requirements of a system can be represented using a use case model in the Use Case Diagram. The use case diagrams for the actors of this case study are given as below.

3.1.1 Use Case Diagram for Admin, Doctor and Patient



3.2. *Use Case Definition*

Generally, in a design document, Use case definitions should be written for all the *Requirements* of the system. Below shown Use Case Definition “Admin cancels Appointment” is the use-case definition.

3.2.1 Cancel Appointment Details

Admin performing delete operation on Appointment Details. **USE CASE #**		Doc-001
Goal		The admin should be able to delete/cancel the Appointment.
Preconditions		Appointment should be present in the database and admin should be logged in.
Success End Condition		Appointment should be deleted from the database. Proper message is shown.
Failed End Condition		No change in Appointment details in the database. Proper message is shown.
Primary, Secondary Actors		Primary: Administrator
Trigger		Admin Initiative
DESCRIPTION	**Step**	**Action**
1		Click on the 'View Appointments' option.
2		Enter Appointment ID to view a specific detail or View all Appointments. The details with delete/modify option are shown.
3		Click on the delete option. Confirmation message is shown.
4		Select the ID to be deleted.
Step		**Branching Action**
1		Enter invalid FlightID.
2		Select Cancel on confirmation. Appointment details are not deleted.
Related Information/Use cases		*(Doc-002)* Add Appointment Details.
Priority		*P1*
Performance		2sec (excluding user input)
Frequency		Once in a week
Assumptions		*Doc-001* is implemented and Appointments exist in DB and Admin could login successfully.

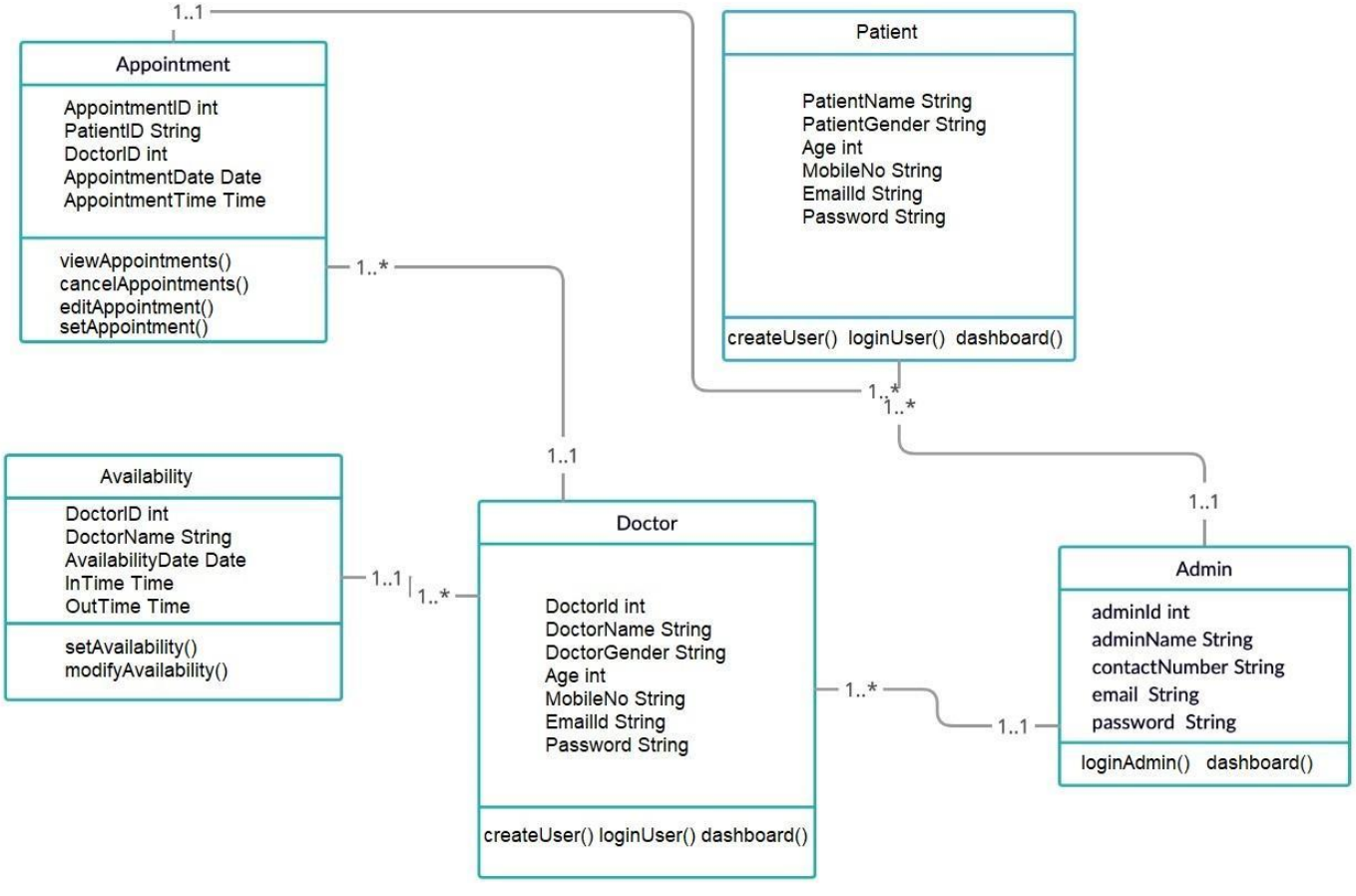
3.2.2 Add Appointment Details

Patient performing Add Flight Schedule operation **USE CASE #**	*Doc-002*
Goal	The patient should be able to add schedule details.
Preconditions	Patient should be successfully logged in.
Success End Condition	Appointment Schedule is added in the database and ScheduleID should be auto-generated.
Failed End Condition	Failure in adding Appointment Schedule information.

Primary, Secondary Actors		Primary-Administrator
Trigger		Patient Initiative
DESCRIPTION	**Step**	**Action**
1		Click on the 'Add Appointment' option.
2		Enter valid Doctor ID and Schedule information.
3		Click on Add Appointment option.
Step		**Branching Action**
1		Enter invalid/Incomplete Schedule information.
2		Click on Add Appointment option.
Related Information/Use cases		
Priority		*P1*
Performance		*2 secs (excluding user interaction)*
Frequency		Once every 2 days.
Assumptions		Patient and Doctor exists in the DB.

3.3. *Class Diagram*

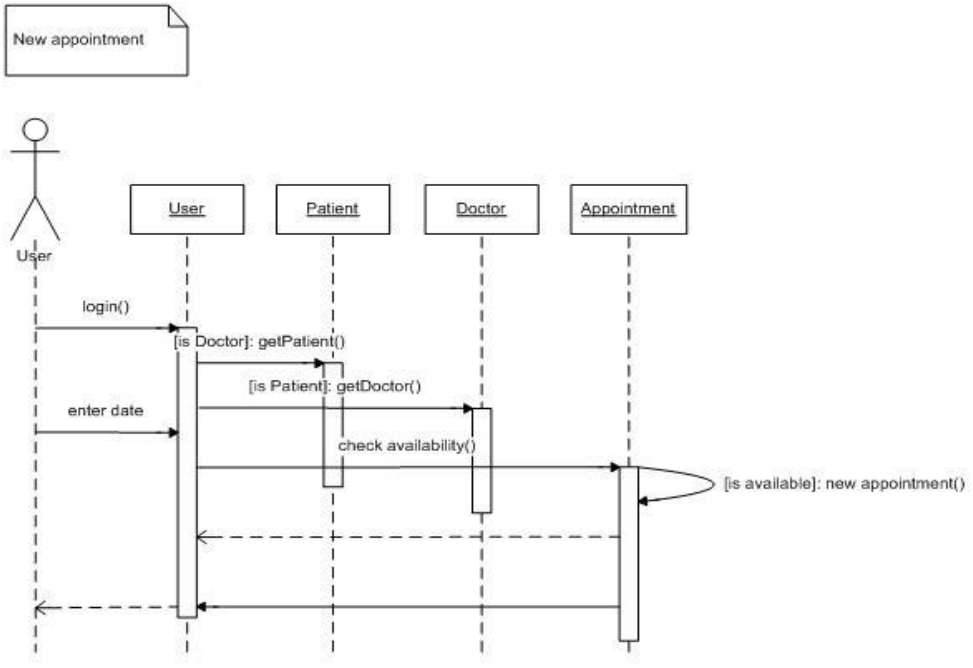
The class diagram is a very basic concept in an object-oriented world. Class diagrams demonstrate a model, describing what attributes and behavior it has rather than describing the methods for accomplishing operations. Class diagrams are very useful in representing relationships between classes and interfaces.



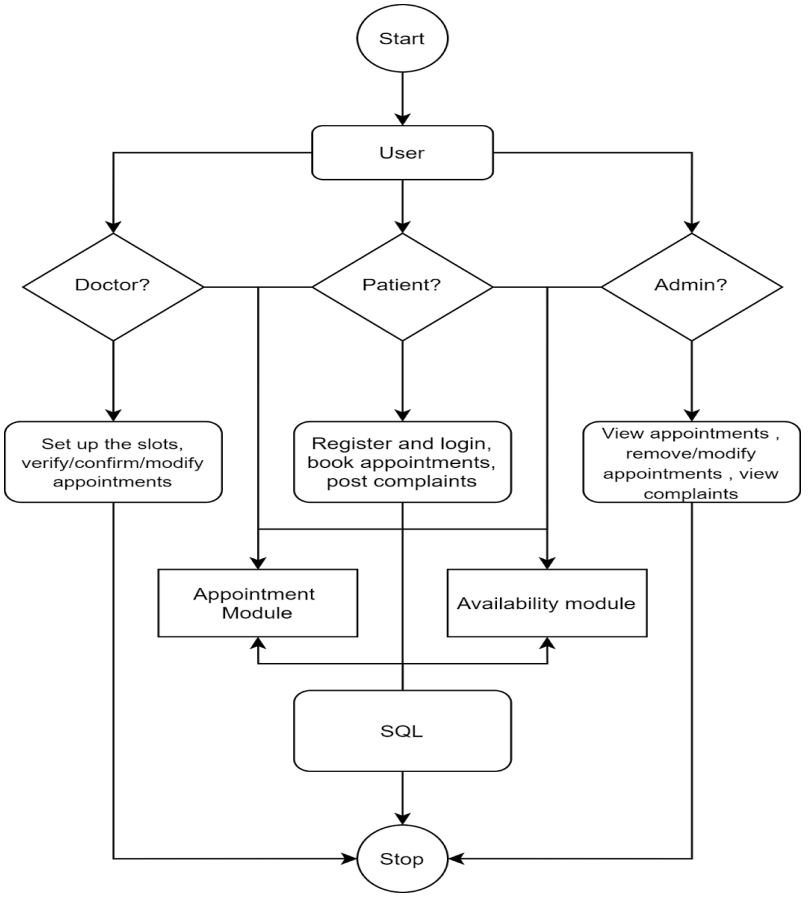
3.4. *Sequence Diagram*

A graphical representation of a module's function invoking functions of other modules in order to achieve a task (specific user requirement) is called a sequence diagram. A sequence diagram for the add process is given below for reference.

3.4.1 Patient adds new Appointment



3.5. Overall working *Flowchart*



3.6. *Classes*

This section provides a brief outlook on the respective classes to be used for the implementation.

Class Name	Attributes	Data Type
Patient	PatientID	Int
	PatientName	String
	PatientGender	String
	PatientAge	Int
	PatientMobile	Int
	PatientEmail	String
	PatientPassword	String
Doctor	DoctorId	Int
	DoctorName	String
	DoctorGender	String
	DoctorAge	Int
	DoctorMobile	String
	DoctorEmail	String
	DoctorPassword	String
Admin	AdminID	Int
	AdminName	String
	ContactNumber	String
	Email	String
	Password	String
Availability	DoctorDate	Date
	IntTime	Time
	OutTime	Time
	DoctorId	Int
Appointment	aID	Int
	aDate	Date
	aTime	Time
	DoctorId	Int
	PatientID	Int

3.7. *UI Templates*

3.7.1 UI Principle

The UI [Presentation Layer] should be designed with the below mentioned principles which helps easy interaction by the user to the application.

3.7.2 UI controls and Usage Principle

The below table provides information on the UI Controls. Based on the User Interface type, the corresponding controls can be used to receive the user input.

UI Type	Controls	Description
Entry	CLI inputs	Operations like inserting details to the databases
Decision Control	Case Options	Operations like selecting the functionalities or users
Table View	CLI inputs	Operations like viewing appointment and availability tables.

3.7.3 UI Template

This section contains the design template for the dashboard [Fig. 1] that will be displayed at the time of start of this application and actor specific area[Fig. 2].

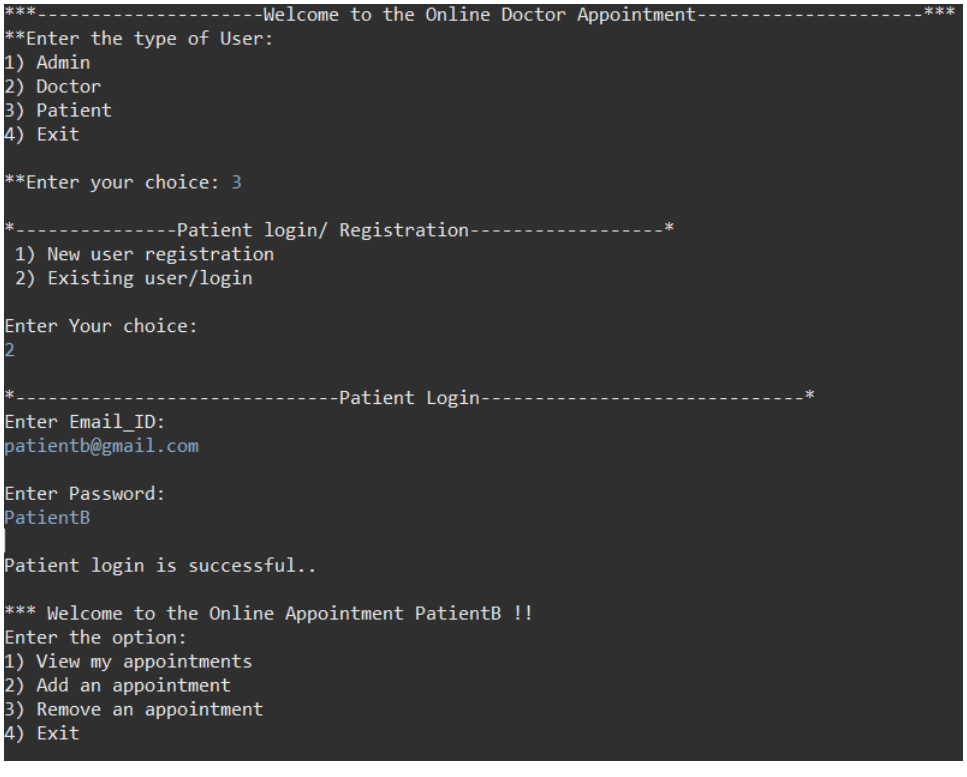


Fig. 1

```
Enter the option:
1) View my appointments
2) Add an appointment
3) Remove an appointment
4) Exit
2
 ***The list of all the available doctor dates (order by Date):
 +-----------------+-----------------+-----------------+-----------------+
 |    Doctor_ID    |   Doctor_Date   |     In_Time     |     Out_Time    |
 +-----------------+-----------------+-----------------+-----------------+
 |       1         |    2021-06-21   |     08:00:00    |     18:00:00    |
 |       2         |    2021-06-21   |     08:00:00    |     18:00:00    |
 |       3         |    2021-06-21   |     08:00:00    |     18:00:00    |
 |       4         |    2021-06-21   |     08:00:00    |     18:00:00    |
 |       5         |    2021-06-21   |     08:00:00    |     18:00:00    |
 |       1         |    2021-06-22   |     08:00:00    |     18:00:00    |
 |       2         |    2021-06-22   |     08:00:00    |     18:00:00    |
 |       3         |    2021-06-22   |     08:00:00    |     18:00:00    |
 |       4         |    2021-06-22   |     08:00:00    |     18:00:00    |
 |       5         |    2021-06-22   |     08:00:00    |     18:00:00    |
 |       2         |    2021-06-23   |     11:00:00    |     18:00:00    |
 |       2         |    2021-06-24   |     10:00:00    |     20:00:00    |
 +-----------------+-----------------+-----------------+-----------------+

Enter the Doctor Id:
2
Enter your Appointment date(yyyy-mm-dd):
2021-06-24
Enter your Appointment time(hh:mm:ss):
14:00:00
--> Your Appointment slot has been Successfully created. <--
```

Fig. 2

4. Critical Functions and Focus for Testing

viewAppointments(), editAppointments(), setAppointments(), cancelAppointments(), viewAvailability(), setAvailability(), modifyAvailability()

5. Limitations

- Administrator cannot register. Hence, the admin's profile has to be maintained at the backend.
- CLI restricts the user friendliness.
- Dynamic notification system is not implemented.
- Waiting list for patients are not implemented.
- Customer’s credit card details have to be already entered in the database.

6. APPENDIX

<< EMPTY >>