Coding Challenge 3 – Hospital Management System

Database schema:

Patient Table:

- patientId (Primary Key)
- firstName
- lastName
- dateOfBirth
- gender
- contactNumber
- address

Doctor Table:

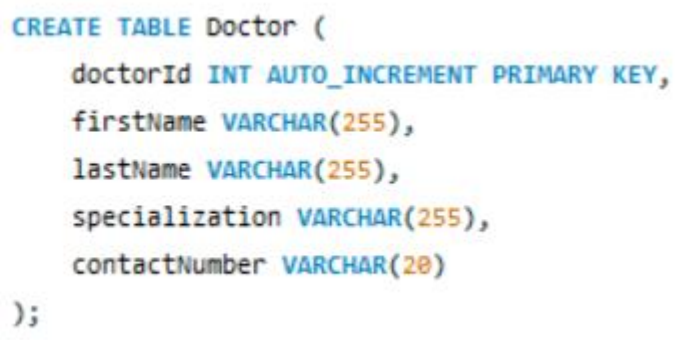
- doctorId (Primary Key)
- firstName
- lastName
- specialization
- contactNumber

Appointment Table:

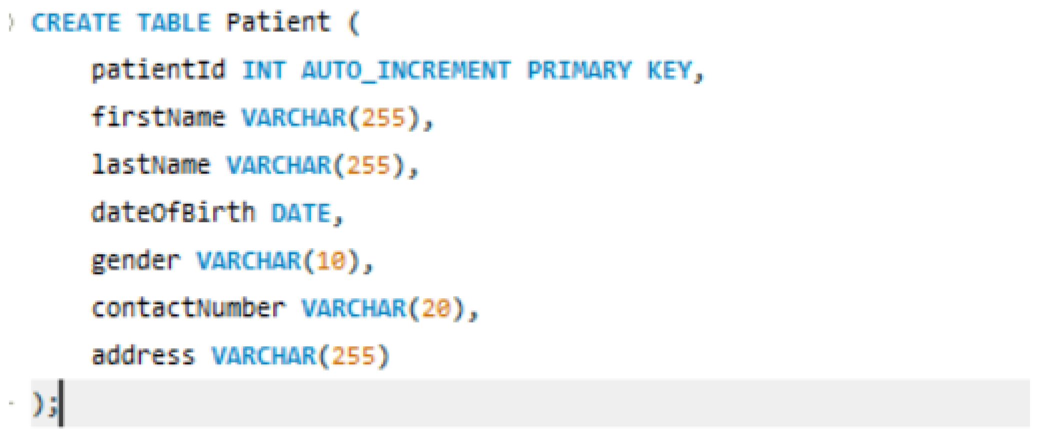
- appointmentId (Primary Key)
- patientId (Foreign Key referencing Patient table)
- doctorId (Foreign Key referencing Doctor table)
- appointmentDate
- description

Database name - hospitalmanagement

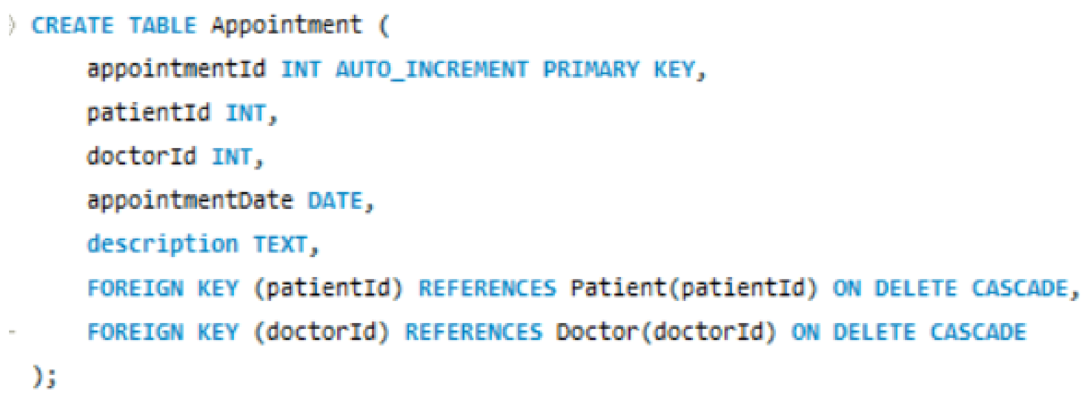
Table creation



```
CREATE TABLE Doctor (
    doctorId INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(255),
    lastName VARCHAR(255),
    specialization VARCHAR(255),
    contactNumber VARCHAR(20)
);
```



```sql
CREATE TABLE Patient (
    patientId INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(255),
    lastName VARCHAR(255),
    dateOfBirth DATE,
    gender VARCHAR(10),
    contactNumber VARCHAR(20),
    address VARCHAR(255)
);
```



```sql
CREATE TABLE Appointment (
    appointmentId INT AUTO_INCREMENT PRIMARY KEY,
    patientId INT,
    doctorId INT,
    appointmentDate DATE,
    description TEXT,
    FOREIGN KEY (patientId) REFERENCES Patient(patientId) ON DELETE CASCADE,
    FOREIGN KEY (doctorId) REFERENCES Doctor(doctorId) ON DELETE CASCADE
);
```

Codes:

Appointment class:

```python
class Appointment:
    def __init__(self, appointmentId=None, patientId=None, doctorId=None,
appointmentDate=None, description=None):
        self.__appointmentId = appointmentId
        self.__patientId = patientId
        self.__doctorId = doctorId
        self.__appointmentDate = appointmentDate
        self.__description = description

    def get_appointmentId(self):
        return self.__appointmentId

    def set_appointmentId(self, appointmentId):
        self.__appointmentId = appointmentId

    def get_patientId(self):
        return self.__patientId

    def set_patientId(self, patientId):
        self.__patientId = patientId

    def get_doctorId(self):
        return self.__doctorId

    def set_doctorId(self, doctorId):
        self.__doctorId = doctorId
    def get_appointmentDate(self):
        return self.__appointmentDate

    def set_appointmentDate(self, appointmentDate):
        self.__appointmentDate = appointmentDate

    def get_description(self):
        return self.__description

    def set_description(self, description):
        self.__description = description

    def __str__(self):
        return f"Appointment ID: {self.__appointmentId}, Patient ID:
{self.__patientId}, Doctor ID: {self.__doctorId}, Appointment Date:
{self.__appointmentDate}, Description: {self.__description}"

    def print_details(self):
        print("Appointment ID:", self.__appointmentId)
        print("Patient ID:", self.__patientId)
        print("Doctor ID:", self.__doctorId)
        print("Appointment Date:", self.__appointmentDate)
        print("Description:", self.__description)
```

Doctor class:

```python
class Doctor:
    def __init__(self, doctorId=None, firstName=None, lastName=None,
specialization=None, contactNumber=None):
        self.__doctorId = doctorId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__specialization = specialization
        self.__contactNumber = contactNumber

    def get_doctorId(self):
        return self.__doctorId

    def set_doctorId(self, doctorId):
        self.__doctorId = doctorId

    def get_firstName(self):
        return self.__firstName

    def set_firstName(self, firstName):
        self.__firstName = firstName

    def get_lastName(self):
        return self.__lastName

    def set_lastName(self, lastName):
        self.__lastName = lastName

    def get_specialization(self):
        return self.__specialization

    def set_specialization(self, specialization):
        self.__specialization = specialization
    def get_contactNumber(self):
        return self.__contactNumber

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def __str__(self):
        return f"Doctor ID: {self.__doctorId}, Name: {self.__firstName}
{self.__lastName}, Specialization: {self.__specialization}, Contact Number:
{self.__contactNumber}"

    def print_details(self):
        print("Doctor ID:", self.__doctorId)
        print("First Name:", self.__firstName)
        print("Last Name:", self.__lastName)
        print("Specialization:", self.__specialization)
        print("Contact Number:", self.__contactNumber)
```

Patients class:

```
class Patient:
    def __init__(self, patientId=None, firstName=None, lastName=None,
dateOfBirth=None, gender=None, contactNumber=None, address=None):
        self.__patientId = patientId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__dateOfBirth = dateOfBirth
        self.__gender = gender
        self.__contactNumber = contactNumber
        self.__address = address

    def get_patientId(self):
        return self.__patientId

    def set_patientId(self, patientId):
        self.__patientId = patientId

    def get_firstName(self):
        return self.__firstName

    def set_firstName(self, firstName):
        self.__firstName = firstName

    def get_lastName(self):
        return self.__lastName

    def set_lastName(self, lastName):
        self.__lastName = lastName

    def get_dateOfBirth(self):
        return self.__dateOfBirth

    def set_dateOfBirth(self, dateOfBirth):
        self.__dateOfBirth = dateOfBirth

    def get_gender(self):
        return self.__gender

    def set_gender(self, gender):
        self.__gender = gender
    def get_contactNumber(self):
        return self.__contactNumber

    def set_contactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def get_address(self):
        return self.__address

    def set_address(self, address):
        self.__address = address

    def __str__(self):
        return f"Patient ID: {self.__patientId}, Name: {self.__firstName}
{self.__lastName}, Date of Birth: {self.__dateOfBirth}, Gender:
{self.__gender}, Contact Number: {self.__contactNumber}, Address:
{self.__address}"

    def print_details(self):
        print("Patient ID:", self.__patientId)
        print("First Name:", self.__firstName)
        print("Last Name:", self.__lastName)
        print("Date of Birth:", self.__dateOfBirth)
        print("Gender:", self.__gender)
        print("Contact Number:", self.__contactNumber)
        print("Address:", self.__address)
```

Exception class:

```
class PatientNumberNotFoundException(Exception):
    def __init__(self):
        self.message = "Patient number not found in the database"
        super().__init__(self.message)
```

Util Classes:

DBConnection class:

```
from util.PropertyUtil import PropertyUtil
import mysql.connector


class DBConnection:
    connection = None

    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            connection_parameters = PropertyUtil.getPropertyString()
            DBConnection.connection =
mysql.connector.connect(**connection_parameters)
            if DBConnection.connection.is_connected():
                print("Connected to MySQL database")
        return DBConnection.connection


# con = DBConnection.getConnection()
# cur = con.cursor()
# cur.execute("show tables")
# res = cur.fetchall()
# for i in res:
#     print(i)
# con.close()
```

PropertyUtil class:

```
class PropertyUtil:
    @staticmethod
    def getPropertyString():
        with open("../database_properties.txt", "r") as file:
            properties = {}
            for line in file:
                key, value = line.strip().split("=")
                properties[key.strip()] = value.strip()
        return properties
```

Interface for services:

IHospitalService class:

```
from abc import ABC, abstractmethod


class IHospitalService(ABC):
    @abstractmethod
    def getAppointmentById(self, appointmentId):
        pass

    @abstractmethod
    def getAppointmentsForPatient(self, patientId):
        pass

    @abstractmethod
    def getAppointmentsForDoctor(self, doctorId):
        pass

    @abstractmethod
    def scheduleAppointment(self, appointment):
        pass

    @abstractmethod
    def updateAppointment(self, appointment):
        pass

    @abstractmethod
    def cancelAppointment(self, appointmentId):
        pass
```

Implementation of interface:

HospitaServiseImpl class:

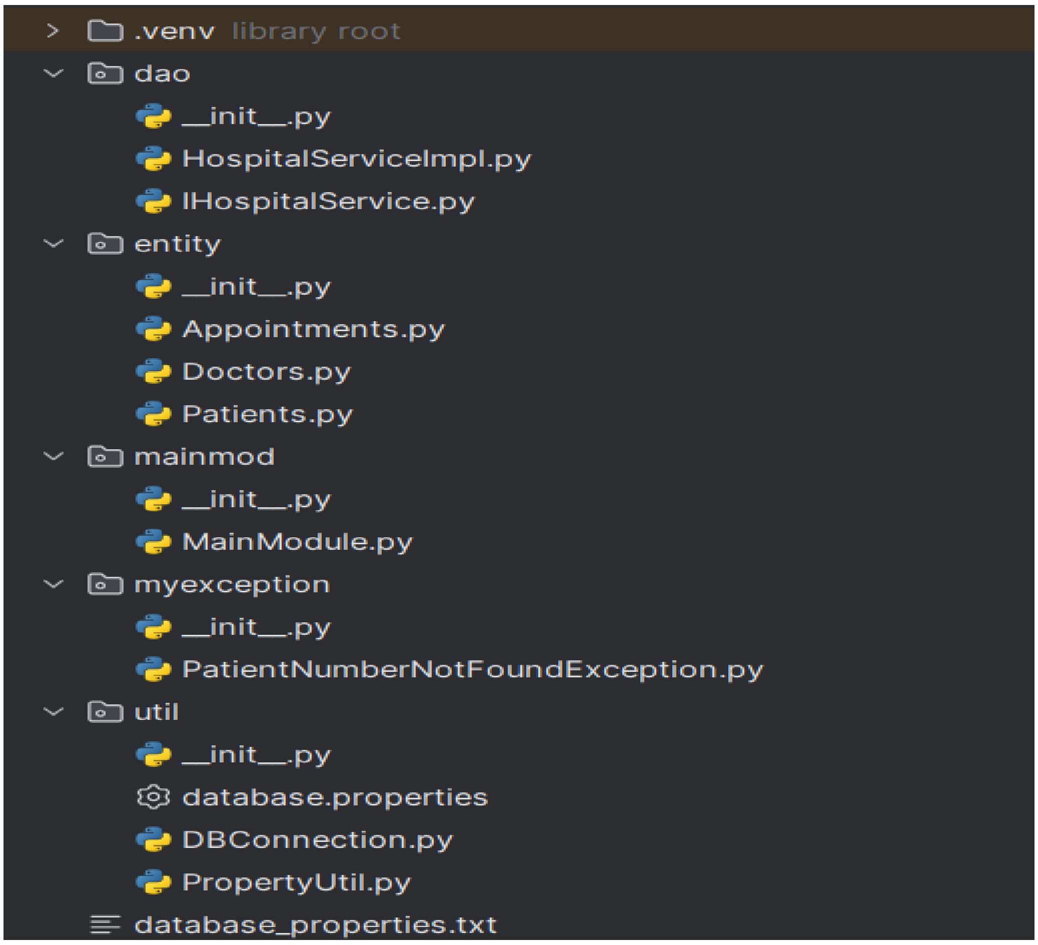
```
from mysql.connector import Error
from dao.IHospitalService import IHospitalService
from entity.Appointments import Appointment
from myexception.PatientNumberNotFoundException import
PatientNumberNotFoundException


class HospitalServiceImpl(IHospitalService):
    def __init__(self, db_connection):
        self.db_connection = db_connection

    def getAppointmentById(self, appointmentId):
        try:
            cursor = self.db_connection.cursor(dictionary=True)
            cursor.execute("SELECT * FROM appointment WHERE appointmentId =
%s", (appointmentId,))
            appointment_record = cursor.fetchone()
            cursor.close()

            if appointment_record:
                appointment = Appointment(
                    appointmentId=appointment_record['appointmentId'],
                    patientId=appointment_record['patientId'],
                    doctorId=appointment_record['doctorId'],
                    appointmentDate=appointment_record['appointmentDate'],
                    description=appointment_record['description']
                )
                return appointment
            else:
                raise PatientNumberNotFoundException()

        except Error as e:
            print("Error getting appointment by ID:", e)
            return None

    def getAppointmentsForPatient(self, patientId):
        try:
            cursor = self.db_connection.cursor(dictionary=True)
            cursor.execute("SELECT * FROM appointment WHERE patientId =
%s", (patientId,))
            appointment_records = cursor.fetchall()
            cursor.close()

            appointments = []
            for appointment_record in appointment_records:
                appointment = Appointment(
                    appointmentId=appointment_record['appointmentId'],
                    patientId=appointment_record['patientId'],
                    doctorId=appointment_record['doctorId'],
                    appointmentDate=appointment_record['appointmentDate'],
                    description=appointment_record['description']
                )
                appointments.append(appointment)

            return appointments
        except Error as e:
            print("Error getting appointments for patient:", e)
            return []

    def getAppointmentsForDoctor(self, doctorId):
        try:
            cursor = self.db_connection.cursor(dictionary=True)
            cursor.execute("SELECT * FROM appointment WHERE doctorId = %s",
(doctorId,))
            appointment_records = cursor.fetchall()
            cursor.close()

            appointments = []
            for appointment_record in appointment_records:
                appointment = Appointment(
                    appointmentId=appointment_record['appointmentId'],
                    patientId=appointment_record['patientId'],
                    doctorId=appointment_record['doctorId'],
                    appointmentDate=appointment_record['appointmentDate'],
                    description=appointment_record['description']
                )
                appointments.append(appointment)

            return appointments

        except Error as e:
            print("Error getting appointments for doctor:", e)
            return []

    def scheduleAppointment(self, appointment):
        try:
            if not self.isValidAppointment(appointment):
                print("Invalid appointment data.")
                return False
            if self.hasAppointmentConflict(appointment):
                print("Appointment conflicts with existing appointments.")
                return False

            cursor = self.db_connection.cursor()
            query = ("INSERT INTO appointment (patientId, doctorId,
appointmentDate, description) VALUES (%s, %s, %s, "
                     "%s)")
            data = (appointment.get_patientId(),
appointment.get_doctorId(), appointment.get_appointmentDate(),
                    appointment.get_description())
            cursor.execute(query, data)
            self.db_connection.commit()
            new_appointment_id = cursor.lastrowid
            print("Your Appointment ID:", new_appointment_id)
            cursor.close()
            # print("Appointment Scheduled...")
            return True

        except Error as e:
            print("Error scheduling appointment:", e)
            return False

    def updateAppointment(self, appointment):
        try:
            # if not self.isValidAppointment(appointment):
            #     print("Invalid appointment data.")
            #     return False

            if self.hasAppointmentConflict(appointment):
                print("Appointment conflicts with existing appointments.")
                return False

            cursor = self.db_connection.cursor()
            query = ("UPDATE appointment SET patientId = %s, doctorId = %s, 
appointmentDate = %s, description = %s "
                     "WHERE appointmentId = %s")
            data = (appointment.get_patientId(), 
appointment.get_doctorId(), appointment.get_appointmentDate(),
                    appointment.get_description(),
                    appointment.get_appointmentId())
            cursor.execute(query, data)
            self.db_connection.commit()
            cursor.close()
            # print("Updated Successfully....")
            return True

        except Error as e:
            print("Error updating appointment:", e)
            return False

    def cancelAppointment(self, appointmentId):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM appointment WHERE appointmentId = 
%s", (appointmentId,))
            appointment_exists = cursor.fetchone()
            if not appointment_exists:
                print("Appointment with ID", appointmentId, "does not 
exist.")
                cursor.close()
                return False
            query = "DELETE FROM appointment WHERE appointmentId = %s"
            cursor.execute(query, (appointmentId,))
            self.db_connection.commit()
            cursor.close()
            # print("Appointment Cancelled...")
            return True

        except Error as e:
            print("Error canceling appointment:", e)
            return False

    def isValidAppointment(self, appointment):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM patient WHERE patientId = %s", 
(appointment.get_patientId(),))
            patient_exists = cursor.fetchone()
            cursor.execute("SELECT * FROM doctor WHERE doctorId = %s", 
(appointment.get_doctorId(),))
            doctor_exists = cursor.fetchone()
            cursor.close()
            return patient_exists is not None and doctor_exists is not None
        except Error as e:
            print("Error validating appointment:", e)
            return False
    def hasAppointmentConflict(self, appointment):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM appointment WHERE doctorId = %s
AND appointmentDate = %s",
                           (appointment.get_doctorId(),
appointment.get_appointmentDate()))
            conflicting_appointments = cursor.fetchall()
            cursor.close()
        except Error as e:
            print("Error checking appointment conflict:", e)
            return False

    def showAllDoctors(self):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM doctor")
            doctors = cursor.fetchall()
            cursor.close()
            print("All Doctor Details:")
            for doctor in doctors:
                print(doctor)

        except Error as e:
            print("Error fetching doctor details:", e)

    def showAllPatients(self):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM patient")
            patients = cursor.fetchall()
            cursor.close()
            print("All Patient Details:")
            for patient in patients:
                print(patient)

        except Error as e:
            print("Error fetching patient details:", e)

    def showAllAppointments(self):
        try:
            cursor = self.db_connection.cursor()
            cursor.execute("SELECT * FROM appointment")
            appointments = cursor.fetchall()
            cursor.close()

            print("All Appointment Details:")
            for appointment in appointments:
                print(appointment)

        except Error as e:
            print("Error fetching appointment details:", e)
```

Main method to trigger all the method in services

MainModule class:

```
from dao.HospitalServiceImpl import HospitalServiceImpl
from entity.Appointments import Appointment
from util.DBConnection import DBConnection


class MainModule:
    def __init__(self):
        db_connection = DBConnection.getConnection()
        self.service = HospitalServiceImpl(db_connection)

    def main(self):
        while True:
            print("\nMenu:")
            print("1. Get Appointment by ID")
            print("2. Get Appointments for Patient")
            print("3. Get Appointments for Doctor")
            print("4. Schedule Appointment")
            print("5. Update Appointment")
            print("6. Cancel Appointment")
            print("7. Show all doctors")
            print("8. Show all patients")
            print("9. Show all appointments")
            print("10. Exit")

            choice = input("Enter your choice: ")

            if choice == '1':
                appointment_id = int(input("Enter appointment ID: "))
                appointment =
self.service.getAppointmentById(appointment_id)
                print("Appointment Details:", appointment)

            elif choice == '2':
                patient_id = int(input("Enter patient ID: "))
                appointments =
self.service.getAppointmentsForPatient(patient_id)
                print("Appointments for Patient:")
                for appointment in appointments:
                    print(appointment)

            elif choice == '3':
                doctor_id = int(input("Enter doctor ID: "))
                appointments =
self.service.getAppointmentsForDoctor(doctor_id)
                print("Appointments for Doctor:")
                for appointment in appointments:
                    print(appointment)

            elif choice == '4':
                patient_id = int(input("Enter patient ID: "))
                doctor_id = int(input("Enter doctor ID: "))
                appointment_date = input("Enter appointment date (YYYY-MM-
DD): ")
                description = input("Enter appointment description: ")
                appointment = Appointment(None,patientId=patient_id,
doctorId=doctor_id, appointmentDate=appointment_date,
                                          description=description)
                success = self.service.scheduleAppointment(appointment)
                if success:
                    print("Appointment scheduled successfully.")
                else:
                    print("Failed to schedule appointment.")

            elif choice == '5':
                appointment_id = int(input("Enter appointment ID to update: 
"))
                apt = self.service.getAppointmentById(appointment_id)
                if apt is None:
                    print("Appointment with ID", appointment_id, "not 
found.")
                    continue
                patient_id = input("Enter patient ID(if dont want to change 
doctor give the old one): ")
                doctor_id = input("Enter doctor ID (if dont want to change 
doctor give the old one): ")
                appointment_date = input("Enter appointment date (YYYY-MM-
DD) (press enter if you don't want to "
                                         "update): ")
                description = input("Enter appointment description (press 
enter if you don't want to update): ")
                appointment = Appointment(appointmentId=appointment_id, 
patientId=patient_id, doctorId=doctor_id,
                                          appointmentDate=appointment_date, 
description=description)
                success = self.service.updateAppointment(appointment)
                if success:
                    print("Appointment updated successfully.")
                else:
                    print("Failed to update appointment.")

            elif choice == '6':
                appointment_id = int(input("Enter appointment ID: "))
                success = self.service.cancelAppointment(appointment_id)
                if success:
                    print("Appointment canceled successfully.")
                else:
                    print("Failed to cancel appointment.")

            elif choice == '7':
                self.service.showAllDoctors()

            elif choice == '8':
                self.service.showAllPatients()
            elif choice == '9':
                self.service.showAllAppointments()
            elif choice == '10':
                print("Exiting...")
                break

            else:
                print("Invalid choice. Please enter a valid option.")


if __name__ == "__main__":
    main_module = MainModule()
    main_module.main()
```

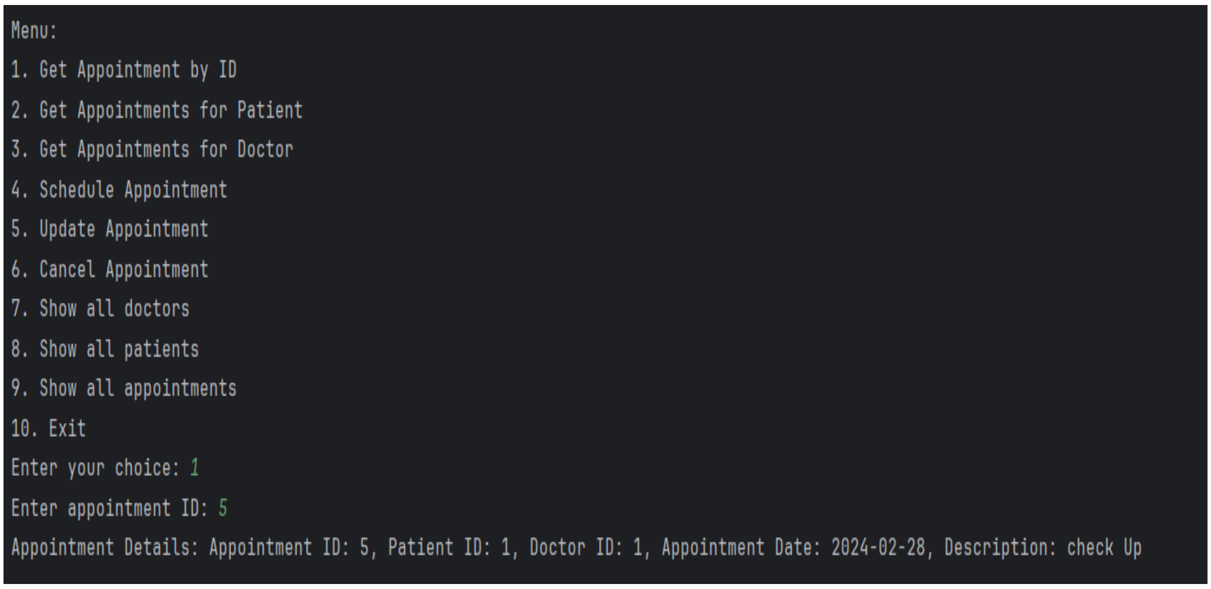
File Structure:



```
> .venv library root
v dao
    __init__.py
    HospitalServiceImpl.py
    IHospitalService.py
v entity
    __init__.py
    Appointments.py
    Doctors.py
    Patients.py
v mainmod
    __init__.py
    MainModule.py
v myexception
    __init__.py
    PatientNumberNotFoundException.py
v util
    __init__.py
    database.properties
    DBConnection.py
    PropertyUtil.py
  database_properties.txt
```

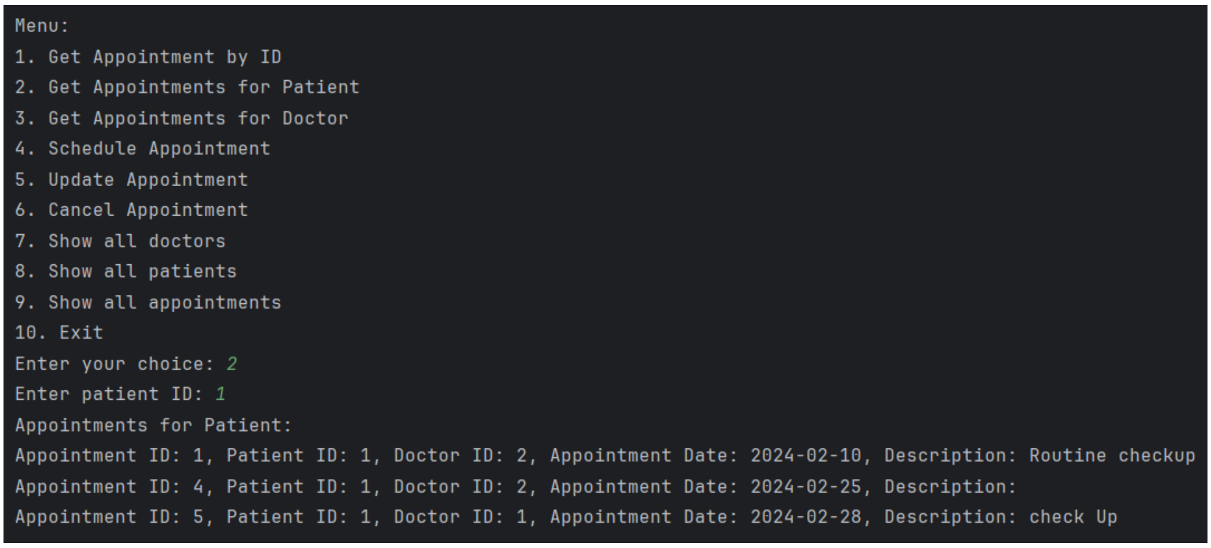
Working of the system:

Get appointment by id:



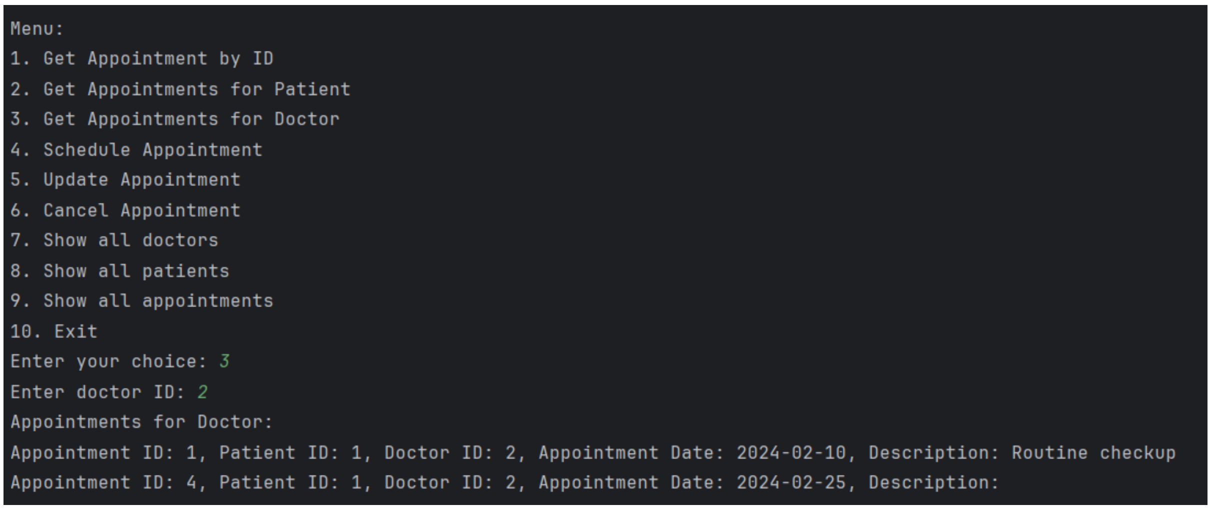
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 1
Enter appointment ID: 5
Appointment Details: Appointment ID: 5, Patient ID: 1, Doctor ID: 1, Appointment Date: 2024-02-28, Description: check Up
```

Get appointments for a patient:



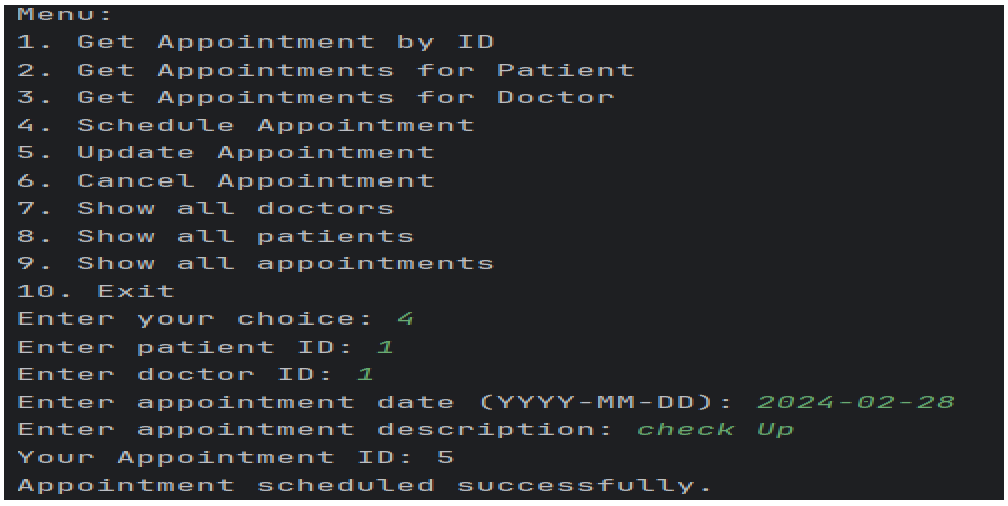
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 2
Enter patient ID: 1
Appointments for Patient:
Appointment ID: 1, Patient ID: 1, Doctor ID: 2, Appointment Date: 2024-02-10, Description: Routine checkup
Appointment ID: 4, Patient ID: 1, Doctor ID: 2, Appointment Date: 2024-02-25, Description:
Appointment ID: 5, Patient ID: 1, Doctor ID: 1, Appointment Date: 2024-02-28, Description: check Up
```

Get appointments for doctor:



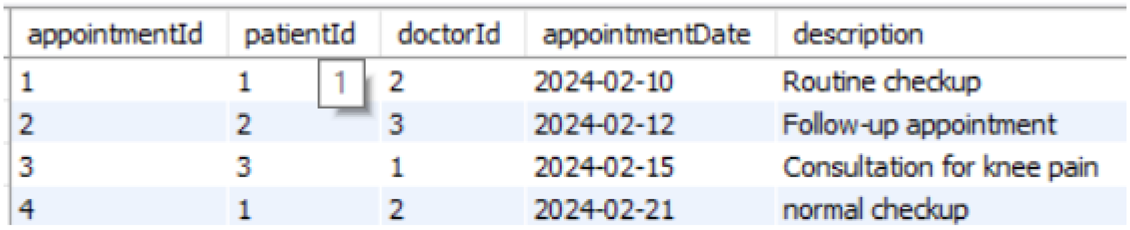
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 3
Enter doctor ID: 2
Appointments for Doctor:
Appointment ID: 1, Patient ID: 1, Doctor ID: 2, Appointment Date: 2024-02-10, Description: Routine checkup
Appointment ID: 4, Patient ID: 1, Doctor ID: 2, Appointment Date: 2024-02-25, Description:
```

Schedule Appointment:



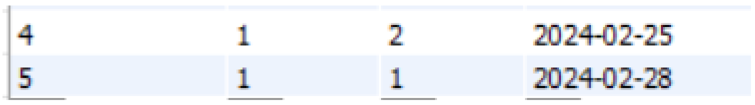
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 4
Enter patient ID: 1
Enter doctor ID: 1
Enter appointment date (YYYY-MM-DD): 2024-02-28
Enter appointment description: check Up
Your Appointment ID: 5
Appointment scheduled successfully.
```

Table before scheduling an appointment



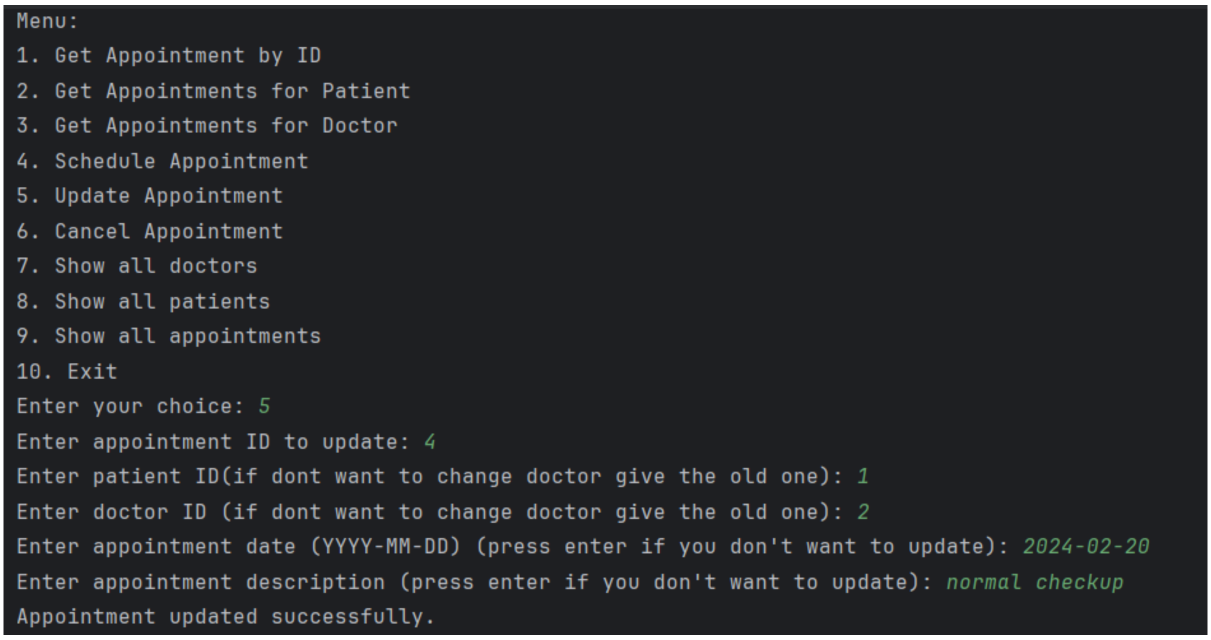
appointmentId	patientId	doctorId	appointmentDate	description
1	1	2	2024-02-10	Routine checkup
2	2	3	2024-02-12	Follow-up appointment
3	3	1	2024-02-15	Consultation for knee pain
4	1	2	2024-02-21	normal checkup

Updated table after scheduling an appointment



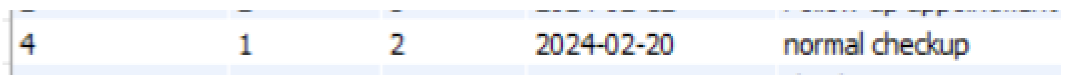
4	1	2	2024-02-25
5	1	1	2024-02-28

Update appointment:



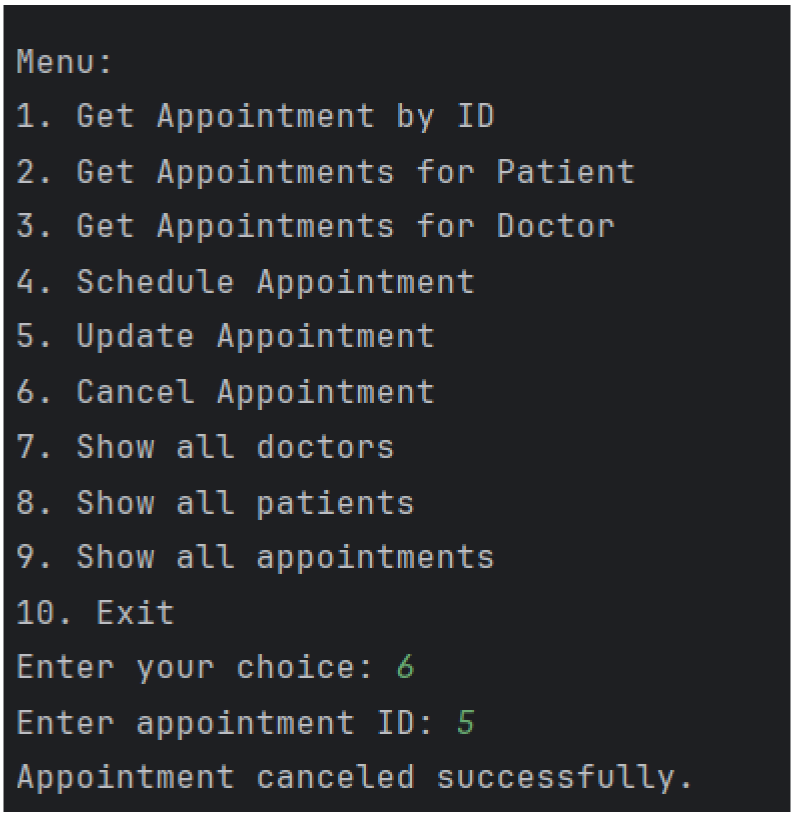
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 5
Enter appointment ID to update: 4
Enter patient ID(if dont want to change doctor give the old one): 1
Enter doctor ID (if dont want to change doctor give the old one): 2
Enter appointment date (YYYY-MM-DD) (press enter if you don't want to update): 2024-02-20
Enter appointment description (press enter if you don't want to update): normal checkup
Appointment updated successfully.
```

Date of appointment changed in database



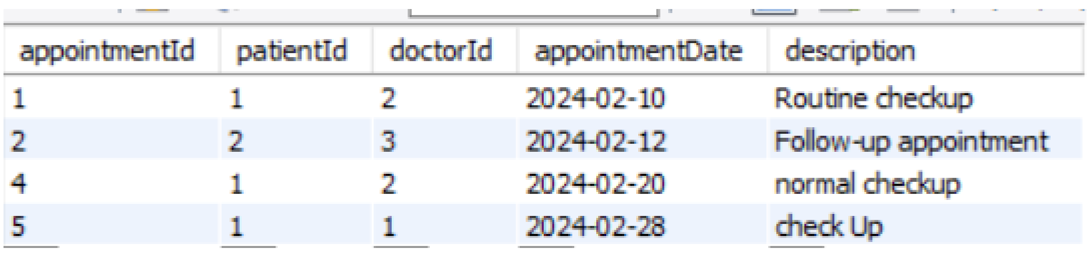
4	1	2	2024-02-20	normal checkup

Cancel Appointment:



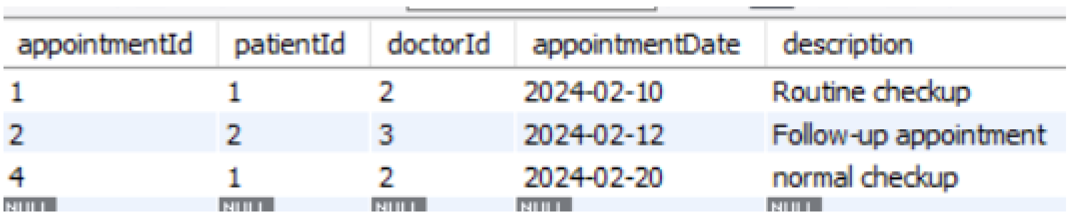
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 6
Enter appointment ID: 5
Appointment canceled successfully.
```

Database Before cancelation of appointment:



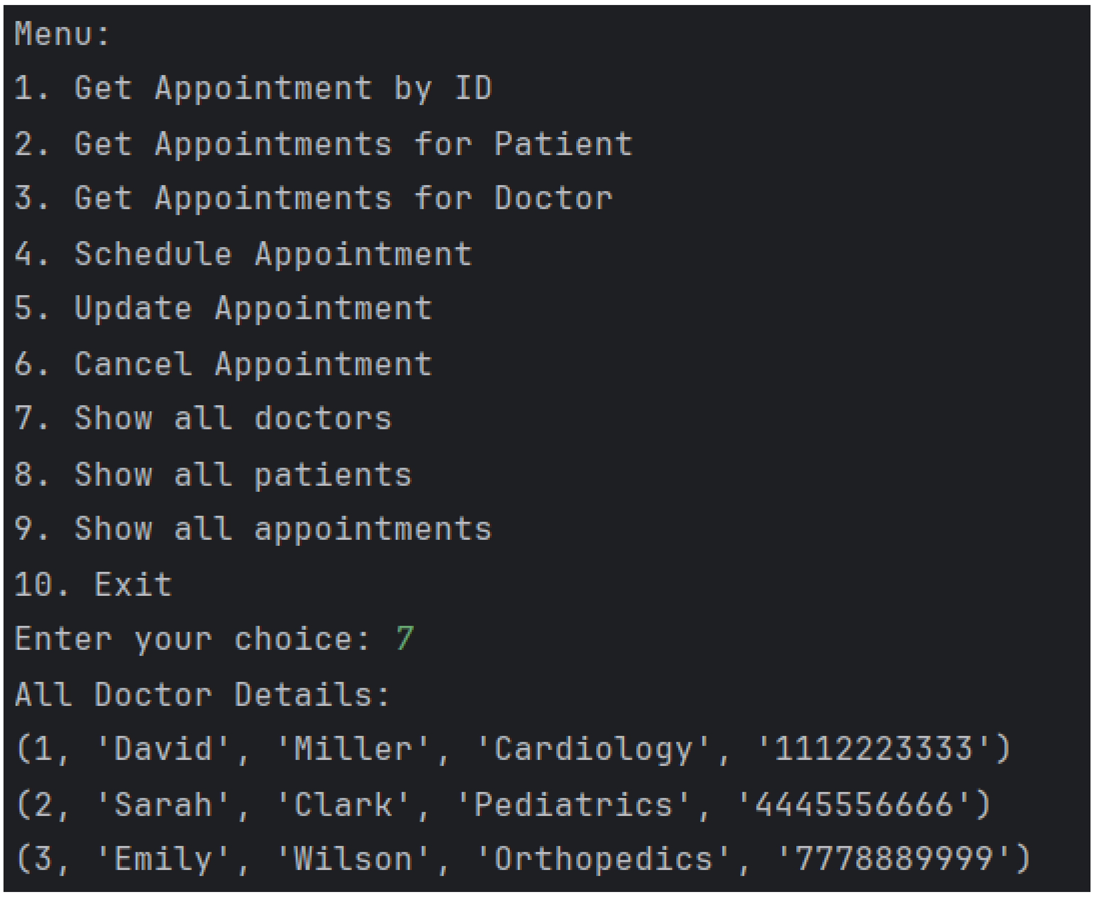
appointmentId	patientId	doctorId	appointmentDate	description
1	1	2	2024-02-10	Routine checkup
2	2	3	2024-02-12	Follow-up appointment
4	1	2	2024-02-20	normal checkup
5	1	1	2024-02-28	check Up

After cancelation:



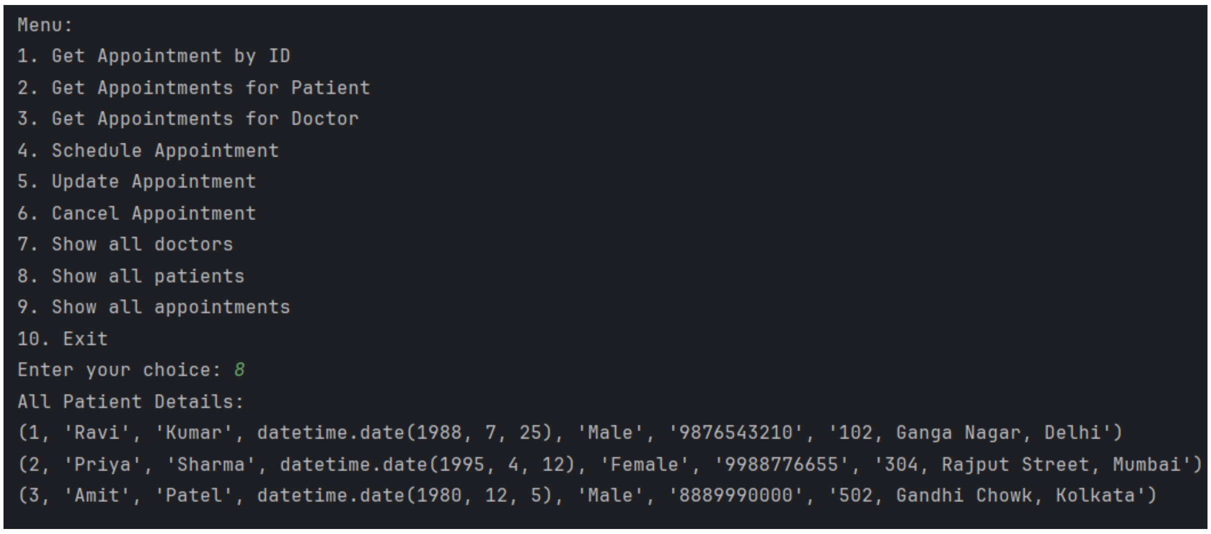
appointmentId	patientId	doctorId	appointmentDate	description
1	1	2	2024-02-10	Routine checkup
2	2	3	2024-02-12	Follow-up appointment
4	1	2	2024-02-20	normal checkup
NULL	NULL	NULL	NULL	NULL

Show all doctors:



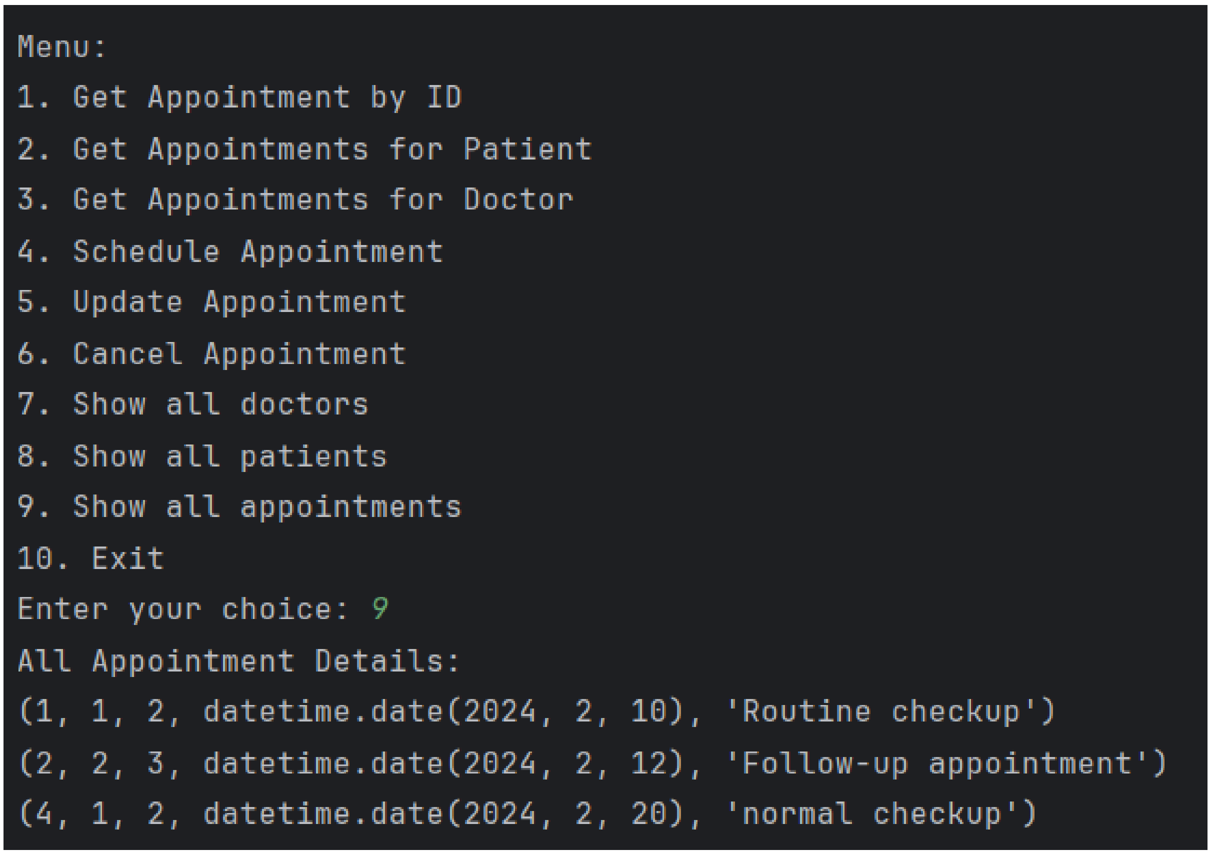
```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 7
All Doctor Details:
(1, 'David', 'Miller', 'Cardiology', '1112223333')
(2, 'Sarah', 'Clark', 'Pediatrics', '4445556666')
(3, 'Emily', 'Wilson', 'Orthopedics', '7778889999')
```

Show all patients:



```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 8
All Patient Details:
(1, 'Ravi', 'Kumar', datetime.date(1988, 7, 25), 'Male', '9876543210', '102, Ganga Nagar, Delhi')
(2, 'Priya', 'Sharma', datetime.date(1995, 4, 12), 'Female', '9988776655', '304, Rajput Street, Mumbai')
(3, 'Amit', 'Patel', datetime.date(1980, 12, 5), 'Male', '8889990000', '502, Gandhi Chowk, Kolkata')
```

Show all appointments:



```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
7. Show all doctors
8. Show all patients
9. Show all appointments
10. Exit
Enter your choice: 9
All Appointment Details:
(1, 1, 2, datetime.date(2024, 2, 10), 'Routine checkup')
(2, 2, 3, datetime.date(2024, 2, 12), 'Follow-up appointment')
(4, 1, 2, datetime.date(2024, 2, 20), 'normal checkup')
```

Submitted By – Asutosh Mishra