System Implementation | Hospital

A brief outline of the application

Hospital is a simple application, executed in a console, which allows users to login as a doctor, a receptionist or a patient. Doctor's view allows users to review prescription requests and issues prescriptions to their patients. Receptionist's view allows users to view and cancel appointments, as well as appointment requests made by patients. Patient's view offers two possibilities: to request a prescription from a doctor, or to request an appointment with a doctor.

Application execution

In order to start using the Hospital app, one needs to open a console window and navigate to the app's folder. From within the folder, execute the following command in the console:

```
python main.py
```

This command will start the application and present the initial menu where users can choose which interface they would like to see (doctor, receptionist or nurse).

In case you are starting the app in Codio, please open Main.py file and press the Run button, or execute the following command in Terminal.

```
python3 Main.py
```

I have prepared some sample data for the hospital application, e.g. login data, doctors, patients, etc.

Login data

Please use the following login data for the **doctor's** view:

Employee Number	Name
10	Christopher

Please use the following login data for the **receptionist 's** view:

Employee Number	Name
1	Gina

Please use the following login data for the **patient's** view:

ID	Name
1	Djordje

Design of data structure and algorithms

Assuming that some requirements were omitted from the provided class diagram, I decided to implement a very primitive version of an authentication system, which compares two values from the database (typically employee number and name) and based on the outcome either logs in the user, or informs them of the mismatch. For the purpose of achieving the desired functionality, certain data structures and algorithms had to be defined. Starting with the database, I decided to keep everything simple and use the most basic form of the SQLite database. When the app starts for the first time, it creates a database called `hospital.db` in the `db` folder. The `create_tables()` function creates the following tables: patient, receptionist, healthcare_professional, prescription, appointment, and appointment_schedule. I decided to go with this approach as it does not require any additional installations, i.e. all the used modules come with Python. All classes have been implemented in their respective files and imported based on the need to keep the project well structured and code easy to read and navigate. Some classes, like Nurse or Prescription have been implemented based on the class diagram, but I haven't used them, as I relied more on the database. Concerning the "find next available" functionality, I achieved that by displaying available doctors to patients when they want to request and appointment. I didn't know how else to implement it, as we are not using dates anywhere in the app.

Class, file and function rationale

`main.py`

Starting point for the app. Responsible for calling the initial functions like the creation of the tables and presenting the main menu.

`database.py`

Responsible for database and table creation. Central point for all SQL related function, as it exports the cursor used in most classes.

`Doctor()`

Inherits the HealthcareProfessional class. Responsible for all doctor related functions, like doctor's login, menu, prescription listing and issuing.

`Patient()`

I omitted the class parameters for this class, as I rely on the database to support that part of the functionality. Responsible for all patient related functions, like patient's login, menu, requesting prescriptions and appointments. When a patient requests an appointment, the app creates an entry in the appointment table.

`Receptionist()`

I omitted the class parameters for this class, as I rely on the database to support that part of the functionality. Responsible for all receptionist related functions, like receptionist's login, menu, confirming and canceling appointments. When a receptionist confirms an appointment request, that request is deleted from the appointment table and inserted in the appointment_schedule table. Apart from that the app also sets the requested doctor as occupied.

Testing

In the GIFs below, you can see the evidence of app testing. In case the gifs don't play automatically on your local machine, you can get open this file in your browser by clicking this link: https://github.com/savanovic-essex/system-implementation/blob/master/README.md

Doctor's view

In the GIF below you can see the functionalities that the Doctor class can do. When a user logs in as a doctor, they will be able to see prescription requests and issue them to their patients.

Patient's view

In the GIF below you can see the functionalities that the Patient class can do. When a user logs in as a patient, they will be able to request an appointment or a prescription from a doctor.

Receptionist's view

In the GIF below you can see the functionalities that the Receptionist class can do. When a user logs in as a receptionist, they will be able to see confirmed appointments in the appointment schedule, as well as cancel them. Apart from the first functionality, they will be able to see appointment requests and either confirm or cancel them.