REPORT ON FINAL PROJECT

1. INTRODUCTION

This hospital database management system (DBMS) supports three user roles: patient, doctor, and pharmacist. In this setup, for simplicity, we assume every user has a unique name. During testing, also for simplicity, each user will use their phone number as their password; however, this is optional and new users may select a password of their choice.

2. DATABASE SCHEMA

The database includes 6 tables, which are listed below. The creation of the database is done by file "*db_creation_1.py*".

1. Patients

Patient_id	Name	Date_of_birth	Gender	Phone

   - Patient_id is the **primary key** and "INTEGER". It is set as "autoincrement".
   - Name is "TEXT", unique and cannot be null (login would be impossible otherwise).
   - Date_of_birth is "TEXT" and checks are in place within the "*checks.py*" file to make sure it is in YYYY-MM-DD format.
   - Gender is "TEXT" and can be male, female or other
   - Phone is "INTEGER"
2. Doctors

Doctor_id	Name	Specialization	Phone

   - Doctor_id is the **primary key** and "INTEGER". It is set as "autoincrement". This will not create any issue with "patient_id" because, even if the values coincide, they are always clearly labeled.
   - Name is "TEXT", unique and cannot be null
   - Specialization is "TEXT" and indicates what are of medicine they operate in
   - Phone is "INTEGER"
3. Appointments

Appointment_id	Patient_id	Doctor_id	Date	Time	Status

   - Appointment_id is the **primary key** and "INTEGER". It is set as "autoincrement".
   - Patient_id and Doctor_id are foreign keys referencing respectively the Patients and Doctors table.

- Date and Time are “TEXT” and there are checks within the “*checks.py*” file to ensure the format is correct (YYYY-MM-DD and HH:MM)
- Status is “TEXT” and indicates whether the appointment happened (“DONE”), was cancelled (“CANCELLED”) or will happen in the future (“SCHEDULED”).

4. Medication

Patient_id	Doctor_id	Prescription_date	Medicine_name	Dispensed	High_risk

- Patient_id and Doctor_id are foreign keys referencing respectively the Patients and Doctors table.
- Prescription_date indicates the date the medicine was prescribed. It is type “TEXT”.
- Dispensed indicates whether the medicine was dispensed or not. It is “NO” as default but can be changed by the pharmacists.
- High_risk indicates whether the medication is at a high risk of abuse. In that case, the pharmacists will need to contact the doctor before dispensing it. It is set to “NO” as default but can be changed by doctors when prescribing.

5. Pharmacist

Pharmacist_id	Name	Phone

- Pharmacist_id is the **primary key** and is an “INTEGER” set to autoincrement
- Name is “TEXT”
- Phone is “INTEGER”

6. Users (used for the login)

Name	Role	Hashed_password

- Name is the primary key, which is the reason we have the assumption of names being unique in our world
- Role indicates whether the person is a pharmacist, doctor or patient and is type “TEXT”
- Hashed_passoword is the hashed version of the password chosen by the user. The password is not stored in plain text for safety reasons.

3. ACTIONS

Each type of user can perform different actions. After logging in the user will be prompted to write which action they want to perform (a list of the options is provided). Some actions might require additional input, but those will be explicitly mentioned in this section of the report. After an action is completed, the user will be prompted to say whether he wishes to complete more actions or not. If yes, he is taken back at to

actions choices, otherwise the program restarts from the login. The checks are contained in the file "*check.py*".

- Login (file: "*passwords_p.py*"): users are prompted to insert their names, roles (doctor, pharmacist or patients) and password. If the user is a patient and the name is not in the database, he will be redirected to a registration page to input the rest of the data needed (date of birth, gender etc.). Doctors and pharmacists cannot do this, as obviously their data is inputted by an (in our case imaginary) HR department. The login will return the name and role which are necessary to run the following functions.
- Patients (file: "*patient_actions_p.py*"):
  1. See appointments (**get_patient_appointments(name)**): patients can see what appointments the have booked, including the date, time, name of the doctor, the status of the appointment and the appointment id. In case they have no appointments booked, a "No appointment booked" message will appear.
  2. Update appointments (**update_appointment_date_patient(name)**): patients can change the date and time of their appointments. For this, they need to input the appointment id. As they might forget it, if they input an id which is wrong or does not belong to them, they will be shown all their appointments with their ids and given another chance)
  3. Cancel appointments (**cancel_appoinment_patient(name)**): as for the appointments updating, the user needs to input the appointment id. They will then be prompted to confirm the cancellation. If they input "y" then the status of the appointment is moved from "scheduled" to "cancelled".
  4. Book appointments(**book_appointment_patient(name)**): patients can book new appointments with a doctor of their choice. Checks are in place to ensure they input the name of a doctor within the database and correct date/time.
  5. Check medications they have been prescribed (**check_medication_patient(name)**): they can see all the medication prescribed to them and the status of the medication (dispensed or not)
  6. Contact a pharmacist (**check_medication_patient(name)**): if the medicine was not dispensed, patients are able to contact a pharmacist (the phone numbers will be displayed).
  7. See a list of all doctors working at the hospital (**see_doctors()**): the query will show all the names and corresponding specialization.
- Doctors (file: "*doctor_actions_p.py*"):

1. See a list of their patients (**patient_list(name)**): a list of all the names of patient the doctor has ever had an appointment with will be displayed
2. Check the medication they have prescribed (**check_medications_doctor(name)**): a list of all the medication prescribed by this doctor, together with the status will be displayed.
3. Prescribe medication (**check_medications_doctor(name)**): a new medication will be added in the medication table. The doctor will have to indicate the patient id, date, name of the medicine and whether it's high risk or not.
4. Update appointments (**update_appointment_date_doctor(name)**): like the patients, doctors can also change appointment dates and time. The same constraint and mechanics apply, the query is slightly different since we start from the doctor_id instead of the patient_id.
5. See the appointments (**get_doctor_appointments(name)**): like the patients, doctors can also see the full list of appointments they have booked. Clearly, instead of the name of the doctor and the specialty, it will show the name and id of the patient.
6. Book appointments (**book_appointment_doctor(name)**): works the same way as the patient booking appointment. The doctor does not need to input the id of the patient but the name. Checks are in place to ensure the date, time and name are valid.
7. Cancel appointments (**cancel_appointment_doctor(name)**): appointment cancellation works the same way for doctor and for patients. The difference in the query is solely due to the use of doctor_id instead of patient_id.

- Pharmacist (file: "*pharmacist_actions_p.py*"):
  1. Dispense medication(**dispense_medication()**): the pharmacist needs to input the patient and medication name. If the medicine is not high risk, a query will automatically update the entry to say that the medicine was dispensed.
  2. Contact a doctor(**dispense_medication()**): if the medicine is high risk, a notice will appear telling the pharmacist to contact the doctor that prescribed the medicine and displaying the appropriate phone number. After, the pharmacist will have to declare whether they gained approval from the doctor, if not a message will appear stating that the medication cannot be dispensed. Otherwise, the same query as above will change the status of the medication to dispensed.

4. POPULATION, TESTING AND USE

The functions to populate the database are stored in the file "*db_population_functions.py*". These are used very rarely in the actual program (for example when adding an appointment or medication) but are used to add the population data necessary to demonstrate the functioning of the database. This data is available at the file "*db_population.py*". Running this file twice will cause the program to break since there is a uniqueness constraint on the names.

If you wanted to test it yourself, a "*delete.py*" file is available. Running it will delete all the contents of the database. The tables will have to be created by running the "*db_creation_1.py*" file, and only after that the "*db_population.py*" file will work. These functions are not meant to be accessed and used directly by the users, so checks are not implemented within them. However, in the cases where they are called by the program during user interaction, checks for datatype and validity are in place. To see the full content of the database it's enough to run the "*view_tables.py*" file.

To use the system as intended you should run the "*main_p.py*" file. Now you will be able to perform all the actions mentioned above. First, you will need to log in with your credentials. The login details below are the ones used in the demo video, but as stated above all the users have their phone number as their password so they can all be tested and you are able to add whatever new doctor, patient or pharmacist you would like using "*db_population_functions.py*" (in the case of pharmacists, their identity does not matter at all outside of the login, but I did create two users to make it more realistic).

Role	Name	Password
Patient	Elsa	111222344
Doctor	Troye	211222334
Pharmacist	Nike	31222339