GROUP ASSIGNMENT

CT108-3-1-PYP

Python Programming (102024-ABA)

HAND OUT DATE: October 2024

HAND IN DATE: 19th January 2025

LECTURER: AZIAH BINTI ABDOLLAH

NO	STUDENT NAME	TP NUMBER
1.	Syed Kazim (Group Leader)	TP084275
2.	Pawan Dinushan	TP084968
3.	Mukarramin Bin Rahman	TP084693
4.	Mohammad Yahya	TP082928
5.	Nour Mohamed	TP081664

Table of Contents

Case Study B: Hospital Management System (HMS)

"Introduction and Assumptions"

INTRODUCTION

The Hospital Management System is a Python-based system devised to help the smooth running of a hospital. Special features are intelligently set up for various roles within the system. This system helps to manage the hospital resources, records of patients, and staff activities efficiently. This reduces the workload of administrators so that doctors and nurses can concentrate on good care.

This system has different functions for different roles. These include Administrators, Doctors, Nurses, Receptionists, and Patients. This ensures that every user can only open those tools and information important for their work.

For example, Administrators can perform user account management, generate hospital reports, and monitor hospital resources. They may also create, update, or delete users; view hospital statistics, and amend operational policies and rules. The doctors can update the treatment plan of the patients. Receptionists do the registration and appointment of the patients. The nurses play a vital role in monitoring the health of the patients, keeping a record of vital signs, and ensuring that the medications are administered timely, whereas the Doctors diagnose and plan for treatment.

Data is safely stored in files (.txt) to keep a record of patients, staff, appointments, and billing. This way of using files ensures that data is saved and can be easily accessed.

In a nutshell, the HMS provides a straightforward and efficient system for the management of hospitals so that staff can spend more time caring for patients and less on administrative duties. The HMS works well for hospitals of different sizes.

ASSUMPTIONS

1. **User Authentication and Role-Based Access Control**
    - Each user is assigned a unique role (Administrator, Doctor, Nurse, Receptionist, or Patient).
    - A secure login system ensures that users can only access functionalities permitted by their role.

2. **Data Storage**
    - Patient, staff, and hospital-related data are stored in structured files (.txt) to ensure persistence across sessions.
    - Separate files are used for different types of data:
        - Appointment.txt - Stores appointment details, including patient names, doctor assignments, appointment dates, and times.
        - Billing.txt - Maintains billing records for patients,
        - Check_in.txt - Tracks patient check-ins, including admission dates and assigned rooms.
        - Check_out.txt - Tracks patient check-outs, including discharge dates
        - Medical_records.txt - Stores detailed medical records for each patient, including diagnosis, treatments, and prescriptions.

- Medication_logs.txt - Tracks the administration of medications to patients, ensuring timely delivery and dosage accuracy.
- Observation.txt - Records observations made by Nurses, including patient vitals and any notable changes in condition.
- Patients.txt - Stores personal details of all registered patients, such as contact information, insurance details, and medical history summary.
- Policies.txt - Contains hospital policies and operational rules set by Administrators.
- Report.txt - Stores generated reports, including statistics on hospital usage, occupancy rates, and resource utilization.
- Resourse.txt - Manages hospital resources, such as available beds, medical equipment, and room statuses.
- Servive_used.txt - Logs services utilized by patients during their stay, including treatments, diagnostics, and other facilities.
- Users.txt - Maintains user account details, including roles and login credentials for all system users.

3. **Role-Specific Functionalities**
   - Each role has access to specific features:
     - **Administrators** manage user accounts, hospital statistics, and resources.
     - **Doctors** update patient records and manage follow-ups.
     - **Nurses** handle daily patient care and medication logs.
     - **Receptionists** manage patient registrations, appointments, and billing.
     - **Patients** can view personal records, book appointments, and access billing details.

4. **Error Handling and Validation**
   - The system includes input validation to prevent invalid entries (e.g., incorrect data formats or missing fields).
   - File access errors are handled wil error handling, with appropriate messages displayed to the user if there is an error.

5. **Scalability**

   - The system design allows future enhancements, such as incorporating additional roles (e.g., pharmacists) or connecting to an external database for larger-scale operations.

"Design (Pseudocode)"

Administrator's Role:

Menu Function:

- **Administrator_Menu()**

6 main functions:

- **Manage_User_Accounts( )**
- **View_Statistics( )**
- **Generate_Report( )**
- **Manage_Hospital_Resources( )**
- **Set_And_View_Operational_Policies( )**
- **Search_Fucntion()**

15 Helper Functions:

- **Create_User( )**
- **Update_User( )**
- **Delete_User( )**
- **Add_Bed( )**
- **Remove_Bed( )**
- **Update_Bed_Status( )**
- **Set_Policies( )**
- **View_Policies( )**
- **Validate_User_ID( )**
- **Validate_User_Contact( )**
- **Validate_User_Role( )**
- **Read_File( )**
- **Write_File( )**
- **Append_File( )**
- **get_user_counts( )**

Pseudo Code:

Global Variables:

SET valid_roles = ["Admin", "Doctor", "Nurse", "Patient"]

SET contact_length = 10

FUNCTION Validate_User_ID(user_id)

""" This function validates if user ID is numeric"""

START

DISPLAY "Validating if user ID is numeric"

IF NOT user_id IS NUMERIC THEN:

    DISPLAY "Error: User ID must be numeric"

    RETURN FALSE

ELSE:

    RETURN TRUE

END FUNCTION

FUNCTION Validate_User_Contact(contact)

""" This function validates if contact number has 10 digits"""

START

DISPLAY "Validating if Contact Number has 10 digits"

IF LENGTH(contact) != 10 OR NOT contact IS NUMERIC THEN:

    DISPLAY "Error: Contact number must be 10 digits"

    RETURN FALSE

ELSE:

    RETURN TRUE

END FUNCTION

FUNCTION Validate_User_Role(role, valid_roles)

""" This function validates if role exists in valid roles"""

START

DISPLAY "Validating User Role"

IF role NOT IN valid_roles THEN

    DISPLAY "Error: Invalid role! Valid roles are: valid_roles"

    RETURN FALSE

ELSE:

    RETURN TRUE

END FUNCTION

FUNCTION Read_File(file_name, heading = TRUE)

""" This function reads data from file and returns heading and data"""

START

TRY :

    OPEN FILE (file_name) IN READ MODE

    READ ALL LINES FROM FILE AS LIST

    STRIP WHITESPACE FROM LINES

IF heading IS TRUE THEN:

    SET heading = FIRST LINE

    SET data = REMAINING LINES SPLIT BY COMMAS **(,)**

    RETURN heading, data

ELSE :

    SET data = ALL LINES SPLIT BY COMMAS

    RETURN data

Except:

    DISPLAY "Error: File file_name not found"

    RETURN NONE, NONE

    CATCH OTHER EXCEPTION AS e

    DISPLAY "Unexpected error: e"

    RETURN NONE, NONE

END FUNCTION

FUNCTION Write_File(file_name, data, heading = TRUE)

""" This function writes data in file by joining through (,)"""

START

OPEN FILE (file_name) IN WRITE MODE

IF heading IS TRUE THEN:

    #WRITE heading TO FILE

    FOR EACH line IN (data) DO:

        START of FOR LOOP

        WRITE line JOINED BY COMMAS TO FILE

        CLOSE FILE

        DISPLAY "File written successfully"

    END OF FOR LOOP

Except:

    DISPLAY "Error: File file_name not found"

    CATCH OTHER EXCEPTION AS e

    DISPLAY "Unexpected error: e"

END FUNCTION

FUNCTION Append_File(file_name, data)

""" This function appends data in File from end by joining through (,)"""

START

OPEN FILE (file_name) IN APPEND MODE

FOR EACH line IN (data) DO :

START OF FOR LOOP

WRITE line JOINED BY COMMAS(,) TO FILE

END OF FOR LOOP

CLOSE FILE

DISPLAY "Data appended successfully"

END FUNCTION

FUNCTION get_user_counts():

START

"""This function counts no. of doctors and nurses (staff) in hospital"""

SET total_doctors = 0

SET total_nurses = 0

heading, data = CALL read_file("Users.txt")

for (i) in range(len(data)):

START OF FOR LOOP

role = data[i][2].strip()

if role == 'Doctor':

#assuming role [2] in file where heading = UserID,Name,Role,Contact

SET total_doctors += 1

elif role == 'Nurse':

SET total_nurses += 1

END OF FOR LOOP

return total_doctors, total_nurses

END OF FUNCTION

FUNCTION View_Statistics()

"""This function views Overall Hospital Statistics from Resources File"""

START

#READ TOTAL PATIENTS FROM FILE "patients.txt"

heading1, patients = CALL Read_File(“Patients.txt”)

#STORE data in VARIABLE (total_patients)

SET total_patients = len(patients)

#READ TOTAL DOCTORS & NURSES FROM FILE "Users.txt"

heading2, users = CALL Read_File(“Users.txt”)

#STORE DATA in VARIABLES (total_doctors),(total_nurses)

SET total_doctors = COUNT DOCTORS IN users

SET total_nurses = COUNT NURSES IN users

#READ AND CALCULATE AVAILABLE BEDS FROM FILE "resources.txt"

heading, resources = CALL Read_File("Resources.txt") SET available_beds = CONVERT_TO_INT(resources[0][2])

DISPLAY ("Total Patients: total_patients" DISPLAY "Total Doctors: total_doctors" DISPLAY "Total Nurses: total_nurses" DISPLAY "Available Beds: available_beds")

END OF FUNCTION

FUNCTION Generate_Report()

"""This function computes and generates a report of hospital resources into Report File"""

START

#READ TOTAL PATIENTS FROM FILE "patients.txt"

heading1, patients = CALL Read_File(“Patients.txt”)

#READ TOTAL DOCTORS & NURSES FROM FILE "Users.txt"

heading2, users = CALL Read_File(“Users.txt”)

SET total_doctors = COUNT DOCTORS IN users

SET total_nurses = COUNT NURSES IN users

#READ HOSPITAL RESOURCES FROM RESOURCES FILE

heading3, resources = CALL Read_File(“Resources.txt”)

SET total_beds = CONVERT_TO_INT(resources[0][0])

SET occupied_beds = CONVERT_TO_INT(resources[0][1])

SET available_beds = CONVERT_TO_INT(resources[0][2])

SET Xray Machines = CONVERT_TO_INT(resources[0][3])

SET Ultra Sound Machines = CONVERT_TO_INT(resources[0][4])

SET Report = ["Total Patients: total_patients", "Total Doctors: total_doctors", "Total Nurses: total_nurses", "Total Beds: total_beds", "Occupied Beds: occupied_beds", "Available Beds: available_beds"]

CALL Write_File("Report.txt",Report)

DISPLAY ("Report generated successfully!")

END OF FUNCTION

FUNCTION Create_User()

""" This function creates a new user after validating the given information"""

START

DISPLAY "Please Input User ID, User Name, User Role, User Contact"

INPUT user_id, user_name, user_role, User_contact

Save data= [user_id, user_name, user_role, User_contact]

CALL Validate_User_ID(user_id)

CALL Validate_User_Role(user_role)

CALL Validate_User_Contact(user_contact)

CALL Append_File("Users.txt",data)

DISPLAY "New User has been created Successfully!"

END OF FUNCTION

FUNCTION Update_User()

""" This function updates the user after validating the new information about existing user"""

START

DISPLAY “Please enter a user ID to update”

INPUT user_id

CALL Validate_User_ID(user_id)

DISPLAY ”Enter new userID, UserRole, UserContact to update”

INPUT new_user_id, new_user_role, new_user_contact

CALL Validate_User_ID(user_id)

CALL Validate_User_Role(user_role)

CALL Validate_User_Contact(user_contact)

#READ AND SAVE ALL DATA INTO A LIST

heading, data = CALL Read_File(“Users.txt”)

SET found to “False”

For EACH USER [1] IN list:

START OF FOR LOOP

IF user_id == list[O] MATCHES IN FILE

SET found = “True”

UPDATE IN List [0] = new_user_id, new_user_role, new_user_contact

BREAK FOR LOOP

ELSE:

DISPLAY “User not Found!”

END OF FOR LOOP

IF found == “True”:

#WRITE DATA BACK INTO FILE

CALL Write_File(“Users.txt”, data, heading)

DISPLAY “A New User has been updated in File Successfully!”

ELSE:

DISPLAY “User Not Found!”

END OF FUNCTION

FUNCTION Delete_User()

""" This function deletes the user after validating the existance of given user"""

START

DISPLAY (“Enter a user to delete”)

INPUT user_id

CALL Validate_User_ID(user_id)

#READ AND SAVE ALL DATA INTO A List

heading, data = CALL Read_File(“Users.txt”)

SET Found = “False”

FOR EACH USER IN list:

    START OF FOR LOOP

    IF user_id == userID MATCHES IN FILE:

        SET found = “True”

        DELETE FROM List [current location]

    ELSE:

        DISPLAY “User Not Found!”

    END OF FOR LOOP

IF found == “True”

    #WRITE UPDATED List BACK INTO FILE

    CALL Write_File(“Users.txt”, data, heading)

    DISPLAY “User has been Deleted from File Successfully!”

ELSE:

    DISPLAY “User Not Found!”

END OF FUNCTION

FUNCTION View_Policies()

"""This function views Policies from Policies File"""

START

READ POLICIES from policies.txt

Heading, data = CALL Read_File(“Policies.txt”)

Display "Following are the current POlicies"

DISPLAY (heading, data)

END OF FUNCTION

FUNCTION Set_Policies()

"""This function Sets Policies in Policies File"""

START

READ POLICIES from policies.txt

Heading, data = CALL Read_File(“Policies.txt”)

Display "Following are the current POlicies"

DISPLAY (heading, data)

DISPLAY "Enter New Policy"

INPUT Policy

#OPEN FILE **"Policies.txt"** in WRITE MODE

CALL Write_File(“Policies.txt”, Policy)

DISPLAY "Policies updated successfully)

END OF FUNCTION

FUNCTION Add_Bed()

"""This function adds a bed to Hospital's available resources"""

START

#OPEN FILE **“Resources.txt”** IN READ MODE

heading, resources = CALL Read_File(“Resources.txt”)

SET total_beds = CONVERT_TO_INT(resources[0][0])

SET occupied_beds = CONVERT_TO_INT(resources[0][1])

SET available_beds = CONVERT_TO_INT(resources[0][2])

SET Xray Machines = CONVERT_TO_INT(resources[0][3])

SET Ultra Sound Machines = CONVERT_TO_INT(resources[0][4])

#Add 1 more bed in Total_Beds

total_beds = total_beds + 1

#WRITE Total_beds, Occupied_Beds IN FILE

CALL Write_File("Resources.txt", heading, resources)

DISPLAY "A new Bed has been added Successfully!"

END OF FUNCTION

FUNCTION Remove_Bed()

"""This function removes a bed from Hospital's available resources"""

START

#OPEN FILE **"Resources.txt"** IN READ MODE

heading, resources = CALL Read_File("Resources.txt")

SET total_beds = CONVERT_TO_INT(resources[0][0])

SET occupied_beds = CONVERT_TO_INT(resources[0][1])

SET available_beds = CONVERT_TO_INT(resources[0][2])

SET Xray Machines = CONVERT_TO_INT(resources[0][3])

SET Ultra Sound Machines = CONVERT_TO_INT(resources[0][4])

IF total_beds > occupied_beds THEN:

    #Add 1 more bed in Total_Beds

    total_beds = total_beds - 1

    #WRITE Total_beds, Occupied_Beds IN FILE

    CALL Write_File("Resources.txt", heading, resources)

    DISPLAY ("A new Bed has been removedSuccessfully!")

ELSE:

    DISPLAY ("Cannot remove a bed because all beds are Occupied currently!")

END OF FUNCTION

FUNCTION Update_Bed()

"""This function updates a bed to Hospital's available resources"""

START

#READ AND SAVE total_beds, occupied_beds FROM FILE

heading, resources = CALL Read_File("Resources.txt")

DISPLAY ("Please choose an option")

DISPLAY:(

"1. Mark Bed as Occupied"

"2. Mark Bed as Available")

SET total_beds = CONVERT_TO_INT(resources[0][0])

SET occupied_beds = CONVERT_TO_INT(resources[0][1])

SET available_beds = CONVERT_TO_INT(resources[0][2])

SET Xray Machines = CONVERT_TO_INT(resources[0][3])

SET Ultra Sound Machines = CONVERT_TO_INT(resources[0][4])

INPUT choice

IF choice == 1 THEN:

occupied_beds = occupied_beds + 1

ELSE IF choice == 2 THEN:

Occupied_beds = occupied_beds – 1

#WRITE total_beds, occupied_beds BACK IN FILE

CALL Write_File("Resources.txt", heading,resources)

DISPLAY ("Bed status updated Successfully!")

END OF FUNCTION

FUNCTION Display_Administrator_Menu()

"""This function provides menu for handling tasks of Administrator's Role"""

START

DISPLAY "Administrator Menu"

START WHILE LOOP

DISPLAY "Enter your Choice to proceed"

DISPLAY OPTIONS(

"1. Manage User Accounts"

"2. View Hospital Statistics"

"3. Generate Report"

"4. Manage Hospital Resources"

"5. Set Operational Policies"

“6. Search Function”

"7. Exit")

INPUT CHOICE

IF choice == 1 THEN:

CALL **Manage_User_Accounts()**

ELSE IF choice == 2 THEN :

CALL **View_Statistics()**

ELSE IF choice == 3 THEN:

CALL **Generate_Report()**

ELSE IF choice ==4 THEN:

CALL **Manage_Hospital_Resources()**

ELSE IF choice == 5 THEN:

CALL **Set_And_View_Operational_policies()**

ELSE IF choice == 6 THEN:

Call **Search_Function()**

ELSE IF choice == 7 THEN:

Exit WHILE LOOP

ELSE:

Display "Invalid choice"

END OF IF

END WHILE LOOP

END OF FUNCTION

FUNCTION Manage_User_Accounts()

"""This function provides menu for handling creating, updating or deleting users in Users File"""

START

DISPLAY "User Account Options \n Enter your Choice to proceed"

DISPLAY (

"1. Create User"

"2. Update User"

"3. Delete User")

INPUT CHOICE

IF choice == 1 THEN:

    CALL create_user()

ELSE IF choice == 2 THEN:

    CALL update_user()

ELSE IF choice == 3 THEN:

    CALL delete_user()

ELSE:

    DISPLAY "Invalid Choice"

END OF IF

END OF FUNCTION

FUNCTION Set_Operational_Policies()

"""This function provides menu for handling Setting and viewing current Hospital Policies from Policies File"""

START

DISPLAY "Choose from the options"

DISPLAY(

"1. View Policies"

"2. Set Policies")

INPUT choice

IF choice == 1 THEN:

CALL **View_Policies()**

ELSE IF choice == 2 THEN:

CALL **Set_Policies()**

ELSE:

DISPLAY "Invalid Choice”

END OF IF

END OF FUNCTION

FUNCTION Manage_Hospital_Resources()

"""This function provides menu for handling adding, removing and updating bed in Resources File"""

START

DISPLAY ("Choose your option")

DISPLAY(

"1. Add Beds"

"2. Remove Beds"

"3. Update Bed Status")

INPUT choice

IF choice == 1 THEN:

CALL **Add_Bed()**

ELSE IF choice == 2 THEN:

CALL **Remove_Bed()**

ELSE IF choice == 3 THEN:

CALL **Update_Bed()**

END OF IF

END OF FUNCTION

FUNCTION Search_User()

"""Special Feature: I have added a Search function to find and display user by Name or ID"""

START

DISPLAY "Choose an option:"

DISPLAY "1. Search in Users (Hospital Staff)"

DISPLAY "2. Search in Patients"

INPUT choice

IF choice == 1 THEN:

DISPLAY "1. Search by UserID"

DISPLAY "2. Search by UserName"

INPUT search_type

IF search_type == 1 THEN:

DISPLAY "Enter UserID to search:"

INPUT user_id

heading, data = CALL Read_File("Users.txt")

SET found = FALSE

FOR EACH line IN data DO:

START OF FOR LOOP

IF line[0] == user_id THEN:

DISPLAY "User found!"

DISPLAY "UserID: line[0], UserName: line[1], UserRole: line[2], UserContact: line[3]"

SET found = TRUE

BREAK

END FOR

IF NOT found THEN:

DISPLAY "No user found with UserID: user_id"

```
ELSE IF search_type == 2 THEN:
        DISPLAY "Enter UserName to search:"
        INPUT user_name
        heading, data = CALL Read_File("Users.txt")
        SET found = FALSE

        FOR EACH line IN data DO
            START OF FOR LOOP:
                IF line[1].LOWER() == user_name.LOWER() THEN
                DISPLAY "User found!"
                DISPLAY "UserID: line[0], UserName: line[1], UserRole: line[2], UserContact:
                line[3]"
                SET found = TRUE
                BREAK
            END FOR

IF NOT found THEN:
            DISPLAY "No user found with UserName: user_name"
ELSE:
        DISPLAY "Invalid search type"

ELSE IF choice == 2 THEN:
        DISPLAY "1. Search by PatientID"
        DISPLAY "2. Search by Patient Name"
        INPUT search_type

        IF search_type == 1 THEN:
                DISPLAY "Enter PatientID to search:"
                INPUT patient_id
                heading, data = CALL Read_File("Patients.txt")
                SET found = FALSE
        FOR EACH line IN data DO:
                START OF FOR LOOP
```

```
            IF line[0] == patient_id THEN:
                DISPLAY "Patient found!"
            DISPLAY "Patient Details: line[ALL_FIELDS]"
            SET found = TRUE
            BREAK
        END FOR

        IF NOT found THEN:
            DISPLAY "No patient found with PatientID: patient_id"

        ELSE IF search_type == 2 THEN:
            DISPLAY "Enter Patient Name to search:"
            INPUT patient_name
            heading, data = CALL Read_File("Patients.txt")
            SET found = FALSE
        FOR EACH line IN data DO:
            START OF FOR LOOP
            IF line[1].LOWER() == patient_name.LOWER() THEN:
            DISPLAY "Patient found!"
            DISPLAY "Patient Details: line[ALL_FIELDS]"
            SET found = TRUE
            BREAK
            END FOR

    IF NOT found THEN:
        DISPLAY "No patient found with Patient Name: patient_name"
    ELSE:
        DISPLAY "Invalid search type"
ELSE:
    DISPLAY "Invalid choice"
END FUNCTION
```

.....END OF PSEUDO CODE.....

Doctor's Role:

Pseudo Code:

DEFINE CONSTANTS:

```
user_records_file = "Users.txt"
patient_records_file = "Patients.txt"
schedule_appointment_file = "Appointments.txt"
medical_history_file = "Medical_Records.txt"
```

DEFINE FUNCTION doctor_login():

```
PRINT login header
LOOP:
    PROMPT doctor_id
    IF validate_doctor_id(doctor_id):
        PRINT welcome message
        CALL doctor_menu(doctor_id)
        BREAK
    ELSE:
        PRINT invalid ID message
        PROMPT retry option
        IF retry == 'N':
            PRINT goodbye message
            EXIT
        ELSE:
            RECURSE doctor_login()
```

DEFINE FUNCTION doctor_menu(doctor_id):

```
LOOP:
    PRINT menu options
    PROMPT choice
    IF choice == "1":
        CALL view_patient_list(doctor_id)
```

```
    ELIF choice == "2":
        CALL update_patient_record()
    ELIF choice == "3":
        CALL schedule_follow_up(doctor_id)
    ELIF choice == "4":
        CALL view_medical_history()
    ELIF choice == "5":
        CALL issue_discharge_approval()
    ELIF choice == "6":
        PRINT goodbye message
        EXIT
    ELSE:
        PRINT invalid choice message
```

DEFINE FUNCTION validate_doctor_id(doctor_id):

```
  TRY:
    OPEN user_records_file
    FOR each line in file:
      IF doctor_id is in line:
        RETURN True
  CATCH FileNotFoundError:
    PRINT file not found message
  RETURN False
```

DEFINE FUNCTION view_patient_list(doctor_id):

```
  TRY:
    OPEN patient_records_file
    PRINT patients header
    SET found = False
    FOR each line in file:
      IF doctor_id is in line:
        PARSE patient details
```

```
            PRINT patient details
            SET found = True
        IF not found:
            PRINT no patients assigned message
    CATCH FileNotFoundError:
        PRINT file not found message
    CALL go_back_or_exit(doctor_id)
```

DEFINE FUNCTION update_patient_record():

```
    TRY:
        PROMPT patient_id, diagnosis, prescription, treatment_plan
        VALIDATE all fields are filled
        OPEN medical_history_file
        READ lines
        OPEN medical_history_file in write mode
        FOR each line:
            IF line contains patient_id:
                UPDATE diagnosis, prescription, treatment_plan
                WRITE updated line
        PRINT success or failure message
    CATCH FileNotFoundError:
        PRINT file not found message
```

DEFINE FUNCTION schedule_follow_up(doctor_id):

```
    TRY:
        PROMPT patient_id, date, time
        APPEND appointment details to schedule_appointment_file
        PRINT success message
    CATCH FileNotFoundError:
        PRINT file not found message
    CALL go_back_or_exit(doctor_id)
```

DEFINE FUNCTION view_medical_history():

```
TRY:
    PROMPT patient_id_input
    OPEN medical_history_file
    FOR each line in file:
        IF line contains patient_id_input:
            PRINT medical history
            RETURN
    PRINT no history found message
CATCH FileNotFoundError:
    PRINT file not found message
```

DEFINE FUNCTION issue_discharge_approval():

```
TRY:
    PROMPT patient_id, discharge_status
    IF discharge_status == "YES":
        OPEN patient_records_file
        READ and MODIFY lines for discharge approval
        WRITE updated lines
        PRINT success or failure message
    ELSE:
        PRINT discharge canceled message
CATCH FileNotFoundError:
    PRINT file not found message
```

DEFINE FUNCTION go_back_or_exit(doctor_id):

```
    PROMPT choice
    IF choice == 'M':
        CALL doctor_menu(doctor_id)
    ELIF choice == 'E':
        PRINT goodbye message
        EXIT
    ELSE:
        PRINT invalid input message
        RECURSE go_back_or_exit(doctor_id)

CALL doctor_login()
```

…….END OF PSEUDOCODE……

Nurse's Role:

Pseudo Code:

FUNCTION get Daily Patient List()

```
DECLARE patient List AS LIST

TRY
    OPEN 'daily_patients.txt' IN READ MODE AS file

        WHILE NOT END OF file
            DECLARE line AS STRING
        READ line FROM file

                DECLARE patient Data AS LIST OF STRING
        Patient Data = SPLIT line BY ','

            IF LENGTH(patient Data) >= 3 THEN
                DECLARE patient Info AS DICTIONARY
          Patient Info['name'] = patient Data[0]       // Patient Name
          Patient Info['room_number'] = patient Data[[1]](#__1) // Room Number
          Patient Info['care_plan'] = patient Data[[2]](#__2)   // Care Plan

                APPEND patient Info TO patient List
        END IF
    END WHILE

        CLOSE file

       RETURN patient List

  EXCEPT File Not Found Error
    PRINT "Error: Daily patient list file not found."
```

```
    RETURN EMPTY LIST

  EXCEPT Exception AS e
    PRINT "An error occurred: " + e message
    RETURN EMPTY LIST
END FUNCTION
```

FUNCTION getDailyPatientList()

```
  DECLARE patientList AS LIST

    OPEN FILE "PatientList.txt" IN READ MODE AS file

   WHILE NOT END OF FILE
    READ line FROM file

        SPLIT line INTO patientID, name, roomNumber, carePlan

         DECLARE patientRecord AS DICTIONARY
    patientRecord["PatientID"] = patientID
    patientRecord["Name"] = name
    patientRecord["RoomNumber"] = roomNumber
    patientRecord["CarePlan"] = carePlan

        ADD patientRecord TO patientList
  END WHILE

   CLOSE FILE file

   OPEN FILE "Vitals.txt" IN WRITE MODE AS vitalsFile

    FOR EACH patient IN patientList
     WRITE patient["PatientID"], patient["Vitals"], patient["Observations"] TO vitalsFile
```

```
    END FOR

      CLOSE FILE vitalsFile

    RETURN patientList
END FUNCTION
```

FUNCTION Prepare Room()

```
      IF Room is occupied THEN
        PRINT "Room is occupied. Please wait."
        RETURN

       CALL Clean Room()

       CALL Set Up Equipment()

      CALL Check Supplies()

       CALL Arrange Furniture()

      PRINT "Room is ready for the procedure."
    END FUNCTION
```

FUNCTION Clean Room()

```
    PRINT "Cleaning the room..."
  END FUNCTION
```

FUNCTION Set Up Equipment()

```
    PRINT "Setting up necessary equipment...
  END FUNCTION

  FUNCTION Check Supplies()
```

```
        PRINT "Checking supplies..."
        END FUNCTION

    FUNCTION Arrange Furniture()
        PRINT "Arranging furniture..."
    END FUNCTION

END Prepare Room For Procedure



END OF FUNCTION
```

FUNCTION log medication administration()

```
Begin
FUNCTION log Medication Administration (patient Id, medication, dosage, time Administered)
        Log Entry = CREATE new Log Entry
      Log Entry patient Id = patient Id
     Log Entry medication = medication
    Log Entry dosage = dosage
     Log Entry time Administered = time Administered
    ADD log Entry TO Medication Log



    RETURN "Medication logged successfully for patient: " + patient Id
END FUNCTION
```

FUNCTION report Emergency(patient Id, emergency Details)

```
        Assigned Doctor = GET Assigned Doctor FOR patient Id
      IF assigned Doctor IS NOT NULL THEN
        notification = CREATE Notification;
        Notification patient Id = patient Id;
        Notification emergency Details = emergency Details;
```

```
    SEND notification TO assigned Doctor;
   RETURN "Emergency reported successfully to doctor: " + assignedDoctor.name;
    ELSE
       RETURN "Error: No assigned doctor found for patient: " + patient Id;
    END IF
END FUNCTION
```

.....END OF PSEUDOCODE.....

Receptionist's Role:

PSEUDOCODE

START

FUNCTION display_menu()

PRINT "Welcome to Receptionist Management System"

PRINT "1. Register a new patient"

PRINT "2. Update Patient Information"

PRINT "3. Schedule an appointment"

PRINT "4. View hospital services, doctors, and departments"

PRINT "5. Check-in a patient"

PRINT "6. Check-out a patient"

PRINT "7. Record services used"

PRINT "8. Generate and view billing"

PRINT "9. Send Appointment Reminders"

PRINT "10. Exit"

FUNCTION register_patient()

INPUT patient_id

INPUT name

REPEAT UNTIL valid numeric input:

INPUT age

REPEAT UNTIL valid gender input ('M' or 'F'):

INPUT gender

REPEAT UNTIL valid numeric input:

INPUT contact

INPUT address

INPUT condition

INPUT doctor_id

APPEND patient details (patient_id, name, age, gender, contact, address, condition, doctor_id) TO "Patients.txt"

PRINT "Patient registered successfully"

FUNCTION update_patient_info()

INPUT patient_id

TRY TO READ "Patients.txt" INTO records

IF file not found:

PRINT "No patients found"

RETURN

FOR each record in records:

IF record matches patient_id:

PROMPT user to update or keep existing name

REPEAT UNTIL valid numeric input OR keep current contact:

INPUT new contact

PROMPT user to update or keep current address

UPDATE record in "Patients.txt"

PRINT "Patient information updated successfully"

RETURN

PRINT "Patient ID not found"

FUNCTION schedule_appointment()

INPUT patient_id

INPUT doctor_id

REPEAT UNTIL valid date format (YYYY-MM-DD):

INPUT appointment_date

REPEAT UNTIL valid time format (HH:MM):

INPUT appointment_time

APPEND appointment details (patient_id, doctor_id, appointment_date, appointment_time) TO "Appointments.txt"

PRINT "Appointment scheduled successfully"

FUNCTION view_hospital_info()

PRINT "Hospital Information"

PRINT "Available Services:"

PRINT "- General Consultation"

PRINT "- Pediatrics"

PRINT "- Cardiology"

PRINT "- Orthopedics"

PRINT "- Dermatology"

PRINT "Doctors:"

PRINT "- ID: D001, Name: Dr. Susan Clark, Specialty: Cardiology, Contact: 2345678901"

PRINT "- ID: D002, Name: Dr. John Smith, Specialty: Pediatrics, Contact: 1122334455"

PRINT "- ID: D003, Name: Dr. Robert Darris, Specialty: Orthopedics, Contact: 1222334555"

PRINT "- ID: D004, Name: Dr. Linda Johnson, Specialty: Dermatology, Contact: 1123334445"

PRINT "Departments:"

PRINT "- Emergency"

PRINT "- Outpatient"

PRINT "- Inpatient"

PRINT "- Radiology"

PRINT "- Pharmacy"

FUNCTION check_in_patient()

INPUT patient_id

INPUT check_in_time (YYYY-MM-DD HH:MM)

APPEND check-in details (patient_id, check_in_time) TO "check_in.txt"

PRINT "Patient checked in successfully"

FUNCTION check_out_patient()

INPUT patient_id

INPUT check_out_time (YYYY-MM-DD HH:MM)

APPEND check-out details (patient_id, check_out_time) TO "check_out.txt"

PRINT "Patient checked out successfully"

FUNCTION record_services()

INPUT patient_id

INPUT service_name

REPEAT UNTIL valid numeric input:

INPUT service_cost

APPEND service details (patient_id, service_name, service_cost) TO "services_used.txt"

PRINT "Service recorded successfully"

FUNCTION generate_billing()

INITIALIZE empty dictionary billing

TRY TO READ "services_used.txt" INTO records

IF file not found:

PRINT "No services recorded"

RETURN

FOR each record in records:

EXTRACT patient_id, service_name, service_cost

ADD service_cost to billing[patient_id]

INPUT patient_id

IF patient_id exists in billing:

DISPLAY total amount

APPEND billing details (patient_id, total_amount) TO "Billing.txt"

ELSE:

PRINT "No billing information found"

FUNCTION send_appointment_reminders():

GET tomorrow's date (tomorrow_date)

TRY to open Appointments.txt

READ all lines into the list of appointments

IF Appointments.txt is not found:

PRINT "No appointments found."

RETURN

SET reminder_found = False

FOR each appointment in the appointments list:

SPLIT appointment data into fields

GET appointment_date from the appointment data

```
    IF appointment_date matches tomorrow_date:
      GET patient_id and doctor_id from the appointment data
      TRY to open Patients.txt
        READ all lines into the list of patients
      IF Patients.txt is not found:
        PRINT "No patient data found."
        RETURN
      SET patient_contact = None
      FOR each patient in the patients list:
        SPLIT patient data into fields
        IF patient_id matches the patient's ID:
          SET patient_contact to the contact number of the patient
          BREAK out of the loop
      IF patient_contact is not None:
        PRINT "Reminder: Patient ID {patient_id} has an appointment tomorrow on
{appointment_date} with Doctor ID {doctor_id}."
        PRINT "Reminder sent to Patient ID {patient_id} at Contact: {patient_contact}"
        SET reminder_found = True
      ELSE:
        PRINT "Could not find contact information for Patient ID {patient_id}."
  IF reminder_found is False:
    PRINT "No appointments found for tomorrow."
  ELSE:
    PRINT "Reminder process complete."
```

FUNCTION main()

```
  WHILE TRUE:
    CALL display_menu()
    INPUT user choice
    IF choice == '1':
      CALL register_patient()
    ELSE IF choice == '2':
```

```
        CALL update_patient_info()
    ELSE IF choice == '3':
        CALL schedule_appointment()
    ELSE IF choice == '4':
        CALL view_hospital_info()
    ELSE IF choice == '5':
        CALL check_in_patient()
    ELSE IF choice == '6':
        CALL check_out_patient()
    ELSE IF choice == '7':
        CALL record_services()
    ELSE IF choice == '8':
        CALL generate_billing()
    ELSE IF choice == '9':
        CALL send_appointment_reminders()
    ELSE IF choice == '10':
        PRINT "Exiting system"
        BREAK
    ELSE:
        PRINT "Invalid choice. Try again"
CALL main()
END
```

.....END OF PSEUDOCODE.....

Patient's Role:

Pseudo Code

DEFINE CONSTANTS:
patients_file = "patients.txt"
appointments_file = "appointments.txt"
feedback_file = "feedback.txt"
users_file = "Users.txt"

DEFINE FUNCTION patient_menu():
LOOP:
PRINT patient menu options
PROMPT choice

IF choice == 1:
CALL view_personal_info()

ELIF choice == 2:
CALL book_appointment()

ELIF choice == 3:
CALL view_appointments()

ELIF choice == 4:
CALL update_personal_information()

ELIF choice == 5:
CALL give_feedback()

ELIF choice == 6:
PRINT goodbye message

EXIT

ELSE:
PRINT invalid choice message

DEFINE FUNCTION validate_date(date_text):
TRY:
CONVERT date_text to datetime using format "%Y-%m-%d"
RETURN True
EXCEPT ValueError:
RETURN False
DEFINE FUNCTION validate_time(time_text):

TRY:
CONVERT time_text to datetime using format "%H:%M"
RETURN True
EXCEPT ValueError:
RETURN False
DEFINE FUNCTION is_future_appointment(date, time):

TRY:
COMBINE date and time into appointment_datetime
IF
appointment_datetime is greater than current datetime:
RETURN True
ELSE:
RETURN False
EXCEPT ValueError:
RETURN False

DEFINE FUNCTION get_doctors_list(file_path):
TRY:
OPEN file at file_path
FOR each line in file:
IF line contains doctor info:
STORE doctor name and ID in doctors dictionary
RETURN doctors dictionary

EXCEPT FileNotFoundError:
PRINT file not found message
EXCEPT Exception as e:
PRINT error message
RETURN empty doctors dictionary

DEFINE FUNCTION book_appointment():
CALL get_doctors_list("Users.txt")
IF no doctors available:
PRINT error message
EXIT

PROMPT patient_id, doctor, date, time
VALIDATE inputs (doctor exists, date format, time format, appointment in the future)
APPEND appointment to "appointments.txt"
PRINT success message

DEFINE FUNCTION view_personal_info():
TRY:
OPEN "patients.txt"
PROMPT patient_id
FOR each line in file:
IF line starts with patient_id:
PRINT personal information
BREAK

IF patient ID not found:
PRINT not found message
EXCEPT FileNotFoundError:
PRINT file not found message

DEFINE FUNCTION view_appointments():
TRY:
OPEN "appointments.txt"
PROMPT patient_id
PRINT patient's appointments
IF no appointments found:
PRINT no appointments message
EXCEPT FileNotFoundError:
PRINT file not found message

DEFINE FUNCTION update_personal_information():

TRY:
OPEN "patients.txt"
PROMPT patient_id
FOR each line in file:
IF line starts with patient_id:
PRINT current information
PROMPT new information (name, contact, address)
REPLACE old information with new details
WRITE updated lines to file
PRINT success message
IF patient ID not found:
PRINT not found message
EXCEPT FileNotFoundError:
PRINT file not found message

DEFINE FUNCTION give_feedback():
TRY:
OPEN "feedback.txt" in append mode
PROMPT patient_id and feedback
APPEND feedback to file
PRINT success message
EXCEPT Exception as e:
PRINT error message

DEFINE FUNCTION initialize_files():
CHECK if files "patients.txt", "appointments.txt", "feedback.txt" exist
IF any file is missing, CREATE an empty file

CALL initialize_files()
CALL patient_menu()

.....END OF PSEUDOCODE.....

“Explanation of Programming Concepts”

Application of Storage Types:

List:

Use Case: Storing a list of patient records or daily tasks.

Example:

```
patients = ["P001", "P002", "P003"]  # List of patient ID
daily_tasks = ["check vitals", "medication", "observations"]
```

Tuple: Use Case: A tuple is used to store fixed data which should not change like, a patient's information which is unalterable.

For example : Copy patient_info = ("John Doe", "Room 101", "Heart Condition") # Tuple of patient details

Dictionary:

Use Case: Map the Patient ID to their patient data through storing the patient information in a dictionary key-value pairs format.

Example:

```
patient_record = {
   "PatientID": "P001",
   "Name": "John Doe",
   "RoomNumber": "101",
   "CarePlan": "Regular check-ups"
}
```

Text Files:

Use Case: Data like patient records, observation records, and logs of medication can be stored for future retrieval. For example: Writing to a file:

Example:

```
with open("patients.txt", "w") as file:
   file.write("P001,John Doe,Room 101,Heart Condition\n")
```

And, reading from a file:

```
with open("patients.txt", "r") as file:
   for line in file:
      print(line.strip())
```

Application of Control Structure:

Conditional Statement:

Use case: Decisions based solely upon user input-or patient data.

Example:

```
if patient_condition == "critical":
   print("Immediate attention required.")
elif patient_condition == "stable":
   print("Monitor regularly.")
else:
   print("Routine check.")
```

Loops

Usage: A list of patients or jobs.

Example:

Print(f"Checking vitals for {patient}.").

While:

A case for use: By repeatedly prompting the nurse until he provides a valid answer or exits.

Example: Copy While True:

```
   choice = input("select an option (1-5):").
   If the choice is in ['1', '2', '3', '4', '5']:
      break  # If a valid option is made, exit the loop; otherwise:
      print( "Invalid option. Please try again." ) Functions:
```

Use Case: Examples include updating patient records and gathering information. Example: Copy def update_patient_vitals(patient_id, new_vitals)

```
   # Update patient vitals (pass)
```

Application of TRY and EXCEPT:

Administrator

```
252  def Update_Bed_Status(file_name, action):
253      """This Function handles wll operations related to bed status: add, remove or update"""
254      try:
255          heading, data = read_file(file_name)
256
257          if heading is None:
258              return #Exit function if file has no heading
259          total_beds = int(data[0][0])
260          occupied_beds = int(data[0][1])
261          available_beds = int(data[0][2])
262
263          if action == "add":
264              total_beds += 1
265              available_beds += 1
266
267          elif action == "remove":
268              if total_beds > occupied_beds:
269                  total_beds -= 1
270                  available_beds -= 1
271              else:
272                  print("Cannot remove a bed as all beds are occupied currently")
273
274          elif action == "update":
275              print("Please choose an option")
276              Choice = int(input(f"""1. Mark Bed as Occupied\n2. Mark Bed as Available"""))
277              if Choice == 1:
278                  occupied_beds += 1
279                  available_beds -= 1
280              elif Choice == 2:
281                  occupied_beds -= 1
282                  available_beds += 1
283              else:
284                  print("Invalid choice")
285                  return
s > Administrator.py
```

Explanation:

The function Update_Bed_Status has the role of adding new bed, deleting the existing one or updating the bed status. The try block encompass all the procedures for updating the bed status including, creation of new bed and setting one to occupied status. The except FileNotFoundError block address the situation where the file is not existing. The except Exception block captures any other error that might occur in the process and then it prints a message without causing the program to end.

Doctor

```
70    # Validate Doctor ID
71    def validate_doctor_id(doctor_id):
72        try:
73            with open(user_records_file, "r") as file:
74                for line in file:
75                    if doctor_id in line:
76                        return True
77        except FileNotFoundError:
78            print("*" * 40)
79            print(f"{'-----⛔User records file not found.⛔-----':^40}")
80            print("*" * 40)
81        return False
```

Explanation:

The program tries to read the user_records_file using file reading permission.Because the above file never existed in my computer, Python will alert FileNotFoundError, and the follow code will be run. Another type of error message gives the user a message stating that the file is missing and then the program will proceed without stopping.

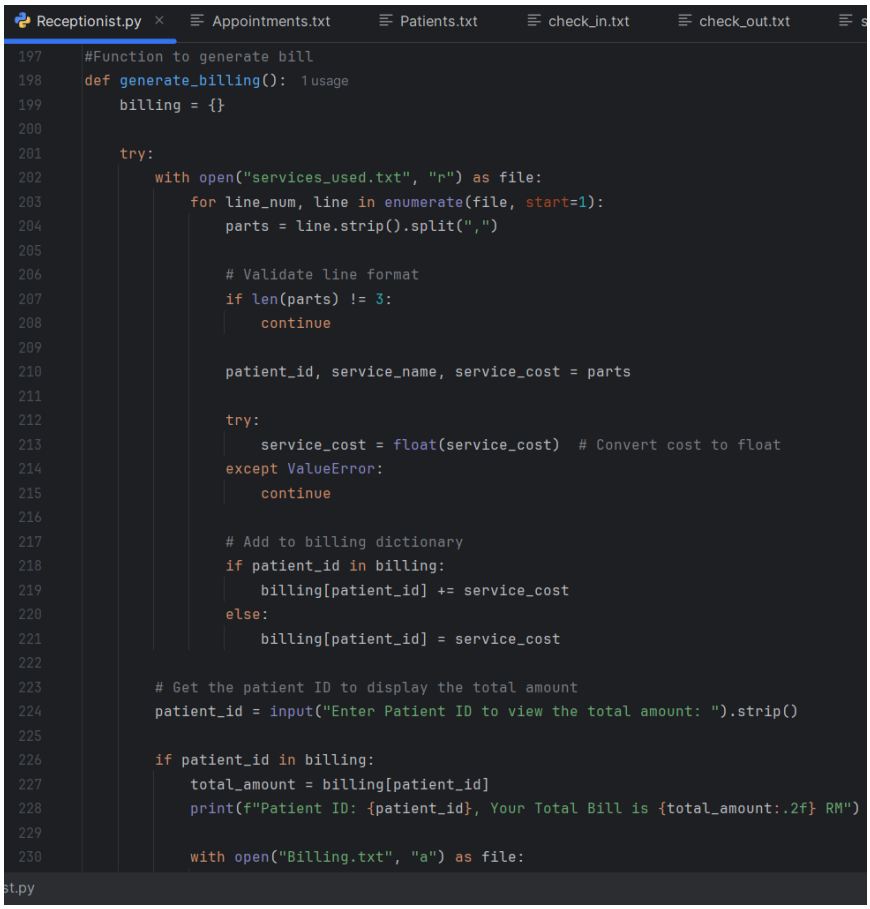
Patient

```
def validate_time(time_text):
    try:
        hour, minute = map(int, time_text.split(":"))
        assert 0 <= hour < 24
        assert 0 <= minute < 60
        return True
    except (ValueError, AssertionError):
        return False
```

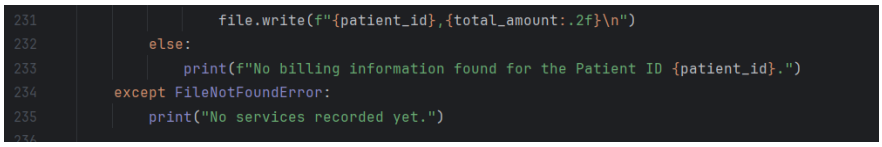
Explantation:

The program tries to split the time_text (e.g., "14:”30” by the colon (:), after that the hour and the minute will be converted into the integer values. Using assert it also verifies if the hour lies between 0 and 23 and the minute between 0 and 59. If the time input is invalid (e.g., "25:While using zero as an hour or 70 as a minute, it returns either ValueError or AssertionError, which are caught in this except block. The function returns False, this represents a situation to mean that the time is invalid.This validation saves the user from entering invalid times, which would certainly crash the program.

Receptionist



```python
#Function to generate bill
def generate_billing():  1 usage
    billing = {}

    try:
        with open("services_used.txt", "r") as file:
            for line_num, line in enumerate(file, start=1):
                parts = line.strip().split(",")

                # Validate line format
                if len(parts) != 3:
                    continue

                patient_id, service_name, service_cost = parts

                try:
                    service_cost = float(service_cost)  # Convert cost to float
                except ValueError:
                    continue

                # Add to billing dictionary
                if patient_id in billing:
                    billing[patient_id] += service_cost
                else:
                    billing[patient_id] = service_cost

        # Get the patient ID to display the total amount
        patient_id = input("Enter Patient ID to view the total amount: ").strip()

        if patient_id in billing:
            total_amount = billing[patient_id]
            print(f"Patient ID: {patient_id}, Your Total Bill is {total_amount:.2f} RM")

            with open("Billing.txt", "a") as file:
```



```python
                file.write(f"{patient_id},{total_amount:.2f}\n")
        else:
            print(f"No billing information found for the Patient ID {patient_id}.")
    except FileNotFoundError:
        print("No services recorded yet.")
```

Explanation:

The try block tries to open the services_used.txt file and it processes each record in order to compute for the billing of each patient. Within this block, another try is employed for trials to convert the value in the service_cost to a float. If it fails, it just goes on to the next line of data.If the file is not found, the except block catches the FileNotFoundError and prints a message: "No services recorded yet."

Nurse

```python
def record_observation(patient_id, observation):
    """Record observations for a specific patient."""
    try:
        with open('observations.txt', 'a') as file:
            file.write(f"{patient_id},{observation}\n")  # Append new observation
        print("Observation recorded successfully.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Explanation:

It has to open the file observations.txt in the append mode to write a new observation associated with a patient_id, so the new observation will then be added to the end of the file, and does not affect or overwrite existing observations.If, for example, the file is locked it will cause an error, so it is the generic Exception block will catch such an error and print its error message.

Application of Validation:

Validation is the process of checking whether the data entered by a user or retrieved from an external source meets specific criteria before it is processed or stored in a programming context. Validation is crucial in avoiding mistakes and ensuring that the data is consistent and that users have a better experience because they will be confident that the information they have entered is correct, complete, or formatted appropriately. Validation is used in the system created by ensuring that data entered in the patient age in numeric form, patient contact number, appointment date and time only, gender input. Encryption, checksums, data sanitization, and other similar validations verify that the data being collected is correct, consistent, and prevents us from processing wrong information and breaking the system. It will ensure that the system can process user inputs correctly and operate more robustly.

This is the explanation fixing all the validations we used, with references specifically to the functions where validations are applied. That could help you to easily follow the code from your screenshots in your assignment.

1. Validate User Role:

Function: def validate_user_role():

Example:

```
"""......START.....................Administrator's Code.....................................""
# Constants
valid_roles = ["Admin", "Doctor", "Nurse"]
```

```
def validate_user_role(role):
    """Validating if role exists in valid roles"""
    if role not in valid_roles:
        print(f"Error: Invalid role!. Valid roles ar {valid_roles}\n")
        return False
    else:
        return True
```

Explaination:

This validation ensures that the User's role input consists of either (Administrator, Doctor, Nurse or Receptionist). If the user types in anything which is not in User role it will display error and ask to input again.

2. Validate User Contact:

```
def validate_user_contact(contact):
    """Validating if contact number has 10 digits"""
    if len(contact) != contact_length or not contact.isdigit():
        print(f"Error: Contact number must be of 10 digits")
        return False
    else:
        return True
```

```
User_Contact = input("Enter User Contact: ").strip()
if not validate_user_contact(User_Contact):
    return  # if contact length is not 10 digits then stop the function
```

Explaination:

This validation ensures that the User's contact number input consists solely of numerical digits with a limit of **contact_length = 10**. If the user types in anything which is not numeric it will display error and ask to input again.

3. Validate Patient Contact Number

Function: register_patient()

Validation: To enter only numeric values in the contact number.

EXAMPLE:

```
while True:
    contact = input("Enter patient's contact number: ").strip()
    if contact.isdigit():
        break
    else:
        print("Invalid input. Contact number must be numeric.")
```

Explanation: This validation ensures that the patient's contact number input consists solely of numerical digits. If the user types in anything which is not numeric, they are repeatedly asked to enter a valid contact number.

4. Validate Appointment Date Format

Function: schedule_appointment()

Validation: Make sure that the appointment date must be in the YYYY-MM-DD format.

EXAMPLE:

```
def schedule_appointment():
    patient_id = input("Enter patient ID: ").strip()
    doctor_id = input("Enter doctor ID: ").strip()

    while True:
        appointment_date = input("Enter appointment date (YYYY-MM-DD): ").strip()
        if len(appointment_date) == 10 and appointment_date.count("-") == 2:
            break
        else:
            print("Invalid date format. Please use YYYY-MM-DD.")

    while True:
        appointment_time = input("Enter appointment time (HH:MM): ").strip()
        if len(appointment_time) == 5 and appointment_time.count(":") == 1:
            break
        else:
            print("Invalid time format. Please use HH:MM.")

    with open("Appointments.txt", "a") as file:
        file.write(f"{patient_id},{doctor_id},{appointment_date},{appointment_time}\n")

    print(f"Appointment scheduled for patient ID {patient_id} with doctor ID {doctor_id}.")
```

Explanation: This validates that the user is entering one of the valid formats, where the total date length is 10 characters and has exactly two hyphens.

5. Validate Appointment Time Format

Function: schedule_appointment()

Validation: Checks if the provided time is in HH:MM format.

EXAMPLE

```
def schedule_appointment():
    patient_id = input("Enter patient ID: ").strip()
    doctor_id = input("Enter doctor ID: ").strip()

    while True:
        appointment_date = input("Enter appointment date (YYYY-MM-DD): ").strip()
        if len(appointment_date) == 10 and appointment_date.count("-") == 2:
            break
        else:
            print("Invalid date format. Please use YYYY-MM-DD.")

    while True:
        appointment_time = input("Enter appointment time (HH:MM): ").strip()
        if len(appointment_time) == 5 and appointment_time.count(":") == 1:
            break
        else:
            print("Invalid time format. Please use HH:MM.")

    with open("Appointments.txt", "a") as file:
        file.write(f"{patient_id},{doctor_id},{appointment_date},{appointment_time}\n")

    print(f"Appointment scheduled for patient ID {patient_id} with doctor ID {doctor_id}.")
```

Explanation: When an appointment time is entered, this validation will check its length to ensure it is the right number (5 characters), then check that it contains a colon and nothing else to make sure that the appointment time is in HH:MM format.

6. Numerical Check (Age Validation)

Function: register_patient()

Validation: Check if the age input is numeric.

EXAMPLE:

```
# Function to register a new patient
def register_patient():
    patient_id = input("Enter patient ID (unique): ").strip()
    name = input("Enter patient's name: ").strip()

    while True:
        age = input("Enter patient's age: ").strip()
        if age.isdigit():
            break
        else:
            print("Invalid input. Please enter a valid age (numeric).")
```

Explanation: When this data is provided there is validation that ensures the patient's age inputted uses a number. The code repeatedly prompts for a valid input when the user enters a non-numeric input.

5. Content of the Gender Validation (Choice Check)

Function: register_patient()

Validation: Makes sure gender entered is either M or F no matter the case.

Example:

```
while True:
    gender = input("Enter patient's gender (M/F): ").strip().upper()
    if gender in ["M", "F"]:
        break
    else:
        print("Invalid input. Please enter 'M' for Male or 'F' for Female.")
```

Explanation: This validates the user gender if user gender is the string M or string F The upper() method makes this validation case-insensitive, which means that both "m" and "M" are acceptable.

6. File Existence Validation

Function: Read_File() (or any function where data is read from a file)

Validation: Verifies that the file exists prior to reading from it

EXAMPLE

```python
def read_file(file_name, heading=True):
    """Reads data from file and returns heading and data"""
    try:
        with open(file_name, 'r') as file:
            lines = file.readlines()

        filtered_lines = [line.strip() for line in lines]

        if heading:  # if file has heading
            heading = filtered_lines[0]
            data = [line.split(",") for line in
                    filtered_lines[1:]]  # spliting data by commas after heading from 2nd line
            return heading, data

        else:  # if file has no heading
            data = [line.split(",") for line in filtered_lines]  # spliting data by commas when there is no heading
            return data

    except FileNotFoundError:
        print(f"Error: The file {file_name} is not available")
        return None, None
    except Exception as e:
        print(f"An unexpected error has been occurred: {e}")
        return None, None
```

Explanation: In the same way, if the file that stores data does not exist, it catches a FileNotFoundError and gives a friendly error message to the user that he should register patients first.

7. Validate User & Patient Login

Function: validate_login(), validate_patient_login

```python
def validate_login(user_id):
    try:
        with open("Users.txt", "r") as file:
            for line in file:
                # Skip the header line
                if line.startswith("UserID"):
                    continue
                # Split each line into fields (UserID, Name, Role, Contact)
                fields = line.strip().split(",")
                if user_id.strip() == fields[0].strip():  # Check exact match with UserID
                    return True
        print("User ID not found.")  # If no match is found
        return False
    except FileNotFoundError:
        print("Error: The file 'Users.txt' does not exist.")
        return False


def validate_patient_login(patient_id):
    try:
        with open("Patients.txt", "r") as file:
            for line in file:
                # Skip the header line
                if line.startswith("UserID"):
                    continue
                # Split each line into fields (UserID, Name, Role, Contact)
                fields = line.strip().split(",")
                if user_id.strip() == fields[0].strip():  # Check exact match with UserID
                    return True
        print("User ID not found.")  # If no match is found
        return False
    except FileNotFoundError:
        print("Error: The file 'Users.txt' does not exist.")
        return False
```

Explaination:

These two login validations ensures that the patient's ID and User's ID input exists in Users.txt and Patients.txt file. If the user types in ID which is not in the files, it will respond with error.

Flow of Add Function:

Administrator Role: Manage User Accounts (Add User)

Files Used: Users.txt
Purpose: Adds a new user to the Users.txt file.

Validating User ID to Ensure It Is Numeric

The following function, validate_user_id, checks if a given user ID consists only of numeric characters. If the user ID is not numeric, it displays an error message and returns False. Otherwise, it returns True, confirming the ID is valid.

```
Use validate_user_id to ensure the ID is numeric. (with code snippet)
def validate_user_id(user_id):
Validating if user ID is numeric
    if not user_id.isdigit():
        print("Error: User ID must be numeric")
        return False
    else:
        return True
```

Validating User Role to Ensure It Is Valid

The validate_user_role function checks if the given user role exists in a predefined list of valid roles. If the role is not found, it displays an error message listing the valid roles and returns False. If the role is valid, it returns True.

```
def validate_user_role(role):
    """Validating if role exists in valid roles"""
    if role not in valid_roles:
        print(f"Error: Invalid role!. Valid roles ar {valid_roles}\n")
        return False
    else:
        return True
```

Appending User Details to Users.txt

The append_file function appends user data to a file (Users.txt), ensuring the data is added at the end, with each field separated by commas. If all user validations pass, the user data is appended, and a success message is displayed.

```
def append_file(file_name, data):
    """Appends data in File from end by joining through (,)"""
    with open(file_name, 'a') as file:
        for line in data:
            file.write(",".join(line) + "\n")

Display a success message.
 #If all validations pass then create the user.
    new_user_data = [[User_ID, User_Name, User_Role, User_Contact]]
    append_file("Users.txt", new_user_data)
    print(f"""User Created Successfully""")
```

Doctor Role: Add Medical Records

Files Used: MedicalRecords.txt, Users.txt, Patients.txt
Purpose: Adds medical records for a patient to MedicalRecords.txt.

Prompting the Doctor to Input Patient Details and Validating Doctor's ID

The doctor is prompted to input patient details such as diagnosis, prescription, and treatment plan. The doctor's ID is validated using the validate_doctor_id function to ensure the ID exists in the user records file.

```
Use validate_doctor_id to verify the doctor's ID.
# Validate Doctor ID
def validate_doctor_id(doctor_id):
    try:
        with open(user_records_file, "r") as file:
            for line in file:
                if doctor_id in line:
                    return True
    except FileNotFoundError:
        print("*" * 40)
        print(f"{'-----⛔User records file not found.⛔-----':^40}")
        print("*" * 40)
    return False
```

Patient Role: Book Appointment

Files Used: Appointments.txt, Users.txt, Patients.txt
Purpose: Adds a new appointment to Appointments.txt.

Booking an Appointment and Validating Appointment Details

This allows a patient to book an appointment by inputting their Patient ID, Doctor's Name, Date, and Time. It validates the date and time formats, checks if the doctor is available, and appends the appointment details to Appointments.txt in the format [Patient ID, Doctor Name, Date, Time].

```
Validate the date format using validate_date.
def validate_time(time_text):
    try:
        hour, minute = map(int, time_text.split(":"))
        assert 0 <= hour < 24
        assert 0 <= minute < 60
        return True
    except (ValueError, AssertionError):
        return False
def book_appointment():
    users_file = "Users.txt"
    appointments_file = "appointments.txt"

    doctors = get_doctors_list(users_file)
    if not doctors:
        print("No doctors available to book an appointment.")
        return

    try:
        with open(appointments_file, "a") as file:
            patient_id = input("Enter your Patient ID: ")

            doctor = input("Enter the doctor's name: ")
            while doctor not in doctors:
                print("Invalid doctor name. Available doctors:")
                for name in doctors.keys():
                    print(f"- {name}")
                doctor = input("Enter the doctor's name: ")

            date = input("Enter the appointment date (YYYY-MM-DD): ")
            while not validate_date(date):
                print("Invalid date format. Please use YYYY-MM-DD.")
                date = input("Enter the appointment date (YYYY-MM-DD): ")

            time = input("Enter the appointment time (HH:MM): ")
            while not validate_time(time):
                print("Invalid time format. Please use HH:MM.")
                time = input("Enter the appointment time (HH:MM): ")
```

```
        while not is_future_appointment(date, time):
            print("The appointment date and time must be in the future.")
            date = input("Enter the appointment date (YYYY-MM-DD): ")
            while not validate_date(date):
                print("Invalid date format. Please use YYYY-MM-DD.")
                date = input("Enter the appointment date (YYYY-MM-DD): ")

            time = input("Enter the appointment time (HH:MM): ")
            while not validate_time(time):
                print("Invalid time format. Please use HH:MM.")
                time = input("Enter the appointment time (HH:MM): ")

        file.write(f"{patient_id},{doctor},{date},{time}\n")
        print("Appointment booked successfully.")

    except Exception as e:
        print(f"Error booking appointment: {e}")
```

Receptionist Role: Register Patient

Files Used: Patients.txt, Appointments.txt, Services_used.txt, Billing.txt
Purpose: Adds a new patient record to Patients.txt.

Registering a New Patient

This allows the receptionist to input patient details such as ID, Name, Age, Gender, Contact, Address, Condition, and Assigned Doctor ID. It validates certain fields like age, gender, and contact number to ensure that they are in the correct format. The patient details are then appended to the Patients.txt file.

```
Prompt the receptionist to input patient details (ID, Name, Age, Gender, Contact, Address, Condition, Assigned Doctor ID).
def register_patient():
    patient_id = input("Enter patient ID (unique): ").strip()
    name = input("Enter patient's name: ").strip()

    while True:
        age = input("Enter patient's age: ").strip()
        if age.isdigit():
            break
        else:
            print("Invalid input. Please enter a valid age (numeric).")

    while True:
        gender = input("Enter patient's gender (M/F): ").strip().upper()
        if gender in ["M", "F"]:
            break
        else:
            print("Invalid input. Please enter 'M' for Male or 'F' for Female.")

    while True:
        contact = input("Enter patient's contact number: ").strip()
        if contact.isdigit():
            break
        else:
            print("Invalid input. Contact number must be numeric.")

    address = input("Enter patient address: ").strip()

    condition = input("Enter patient condition: ").strip()

    doctor_id = input("Enter patient Assigned Doctor ID: ").strip()
with open("Patients.txt", "a") as file:
    file.write(f"{patient_id},{name},{age},{gender},{contact},{address},{condition},{doctor_id}\n")

print(f"Patient {name} registered successfully.")
```

Flow of Update/Modify/Edit Function:

Administrator Role: Update User Details

Files Used: Users.txt
Purpose: Updates the details of an existing user in Users.txt.

Updating and Deleting User Details

This allows for updating and deleting user details after performing necessary validations. The update function checks whether the new user details are valid, and if they are, it updates the user's information in the Users.txt file. The delete function checks if a user exists and then removes the corresponding user from the file.

```
def Update_User():
    """Updating the user after validating the new information about existing user"""
    current_User_id = input("Please enter a userID to update")
    if not validate_user_id(current_User_id):
        return #if userID is not numeric then stop the function

    new_User_ID = input("Enter a new userID")
    if not validate_user_id(new_User_ID):
        return #if userID is not numeric then stop the function

    new_User_Name = input("Enter a new User Name")
    if not new_User_Name:
        return #if username is empty then stop the function

    new_User_Role = input("Enter a new User Role")
    if not validate_user_role(new_User_Role):
        return #if user role is not valid then stop the function

    new_User_Contact = input("Enter a new User Contact")
    if not validate_user_contact(new_User_Contact):
        return #if contact length is not 10 digits then stop the function

    #if all validations pass then update the user
    heading, data = read_file("Users.txt")
    found = False

    for i in range(len(data)):
        user = data[i]
        if user[0] == current_User_id:
            found = True
            data[i] = [new_User_ID,new_User_Name,new_User_Role,new_User_Contact]
            break
```

Doctor Role: Update Medical Records

Files Used: MedicalRecords.txt
Purpose: Updates a patient's medical records in MedicalRecords.txt.

Updating a Patient's Medical Record

This allows you to update the medical record of a patient, including details such as Diagnosis, Prescription, and Treatment Plan. It ensures that all fields are filled out, searches for the patient by their ID, and updates the file with the new information if the ID is found.

```
def update_patient_record():
    try:
        patient_id = input("Enter Patient ID: ").strip().upper()
        print("Enter new details for the patient:")
        diagnosis = input("Diagnosis: ").strip()
        prescription = input("Prescription: ").strip()
        treatment_plan = input("Treatment Plan: ").strip()
        if not diagnosis or not prescription or not treatment_plan:
            print("All fields must be filled.")
            return False
        updated = False
        # Reading the file
        with open(medical_history_file, "r") as file:
            lines = file.readlines()
        # Writing updated information back to the file
        with open(medical_history_file, "w") as file:
            for line in lines:
                fields = line.strip().split(",")
                if fields[0] == patient_id:
                    fields[2] = diagnosis
                    fields[3] = prescription
                    fields[4] = treatment_plan
                    updated = True
                file.write(",".join(fields) + "\n")
        if updated:
            print("-" * 40)
            print(f"{'-----✅ Patient record updated successfully. ✅-----':^40}")
            print("-" * 40)
        else:
            print("-" * 40)
            print(f"{'-----⛔ Patient ID not found. ⛔-----':^40}")
            print("-" * 40)
    except FileNotFoundError:
        print("-" * 40)
        print(f"{'-----⛔ Medical history file not found. ⛔-----':^40}")
        print("-" * 40)
        return False
    return updated
```

Patient Role: Update Personal Information

Files Used: Patients.txt
Purpose: Updates the patient's information in Patients.txt.

Updating Personal Information of a Patient

This allows a patient to update their personal information (Name, Contact, and Address) in the patient records file. It ensures that the patient's current information is displayed, and prompts the patient to either keep the existing data or provide new details. If the patient ID is found, the data is updated and saved.

```
def update_personal_information():
    try:
        patient_id = input("Enter your Patient ID: ")
        found = False
        lines = []
        with open("patients.txt", "r") as file:
            for line in file:
                if line.startswith(patient_id):
                    found = True
                    current_info = line.strip().split(",")
                    print("Current Information:")
                    print(f"Name: {current_info[1]}\nContact: {current_info[4]}\nAddress: {current_info[5]}")

                    # Prompt for each field to update or keep the same
                    name = input(f"Name (current: {current_info[1]}): Keep same? (y/n): ")
                    if name.lower() == "n":
                        current_info[1] = input("Enter new Name: ")

                    contact = input(f"Contact (current: {current_info[4]}): Keep same? (y/n): ")
                    if contact.lower() == "n":
                        current_info[4] = input("Enter new Contact: ")

                    address = input(f"Address (current: {current_info[5]}): Keep same? (y/n): ")
                    if address.lower() == "n":
                        current_info[5] = input("Enter new Address: ")

                    # Rebuild the updated line
                    lines.append(",".join(current_info) + "\n")
                else:
                    lines.append(line)

        if not found:
            print("Patient ID not found.")
        else:
            with open("patients.txt", "w") as file:
                file.writelines(lines)
                print("Information updated successfully.")
    except FileNotFoundError:
        print("Patient records file not found.")
```

Receptionist Role: Update Appointment Details

Files Used: Appointments.txt
Purpose: Updates the details of an appointment in Appointments.txt.

Updating Patient Information

This allows updating a patient's information (name, contact number, home address, medical condition, and assigned doctor ID) in the Patients.txt file. If the patient ID is found, the user can update any field or leave it blank to keep the current value. The changes are then saved back to the file.

```
Validate inputs and overwrite updated details.
 # Function to update patient information
def update_patient_info():
    print("\nUpdate Patient Information")
    patient_id = input("Enter Patient ID to update: ").strip()

    try:
        with open("Patients.txt", "r") as file:
            records = file.readlines()
    except FileNotFoundError:
        print("No patients found. Please register a patient first.")
        return

    updated = False
    for i, record in enumerate(records):
        data = record.strip().split(",")
        if data[0] == patient_id:
            print("Leave the field blank to keep the current value.")
            new_name = input("Enter new name: ").strip() or data[1]

            while True:
                new_contact = input("Enter new contact number: ").strip() or data[4]
                if new_contact.isdigit():
                    break
                else:
                    print("Invalid input. Contact number must be numeric.")

            new_address = input("Enter new home address: ").strip() or data[5]
            new_condition = input("Enter new medical condition: ").strip() or data[6]
            new_doctor_id = input("Enter new doctor ID: ").strip() or data[7]
        new_address = input("Enter new home address: ").strip() or data[5]
        new_condition = input("Enter new medical condition: ").strip() or data[6]
        new_doctor_id = input("Enter new doctor ID: ").strip() or data[7]

        records[i] = f"{data[0]},{new_name},{data[2]},{data[3]},{new_contact},{new_address},{new_condition},{new_doctor_id}\n"
        updated = True
        break

    if updated:
        with open("Patients.txt", "w") as file:
            file.writelines(records)
        print("Patient information updated successfully.")
    else:
        print("Patient ID not found.")
```

Nurse Role: Update Patient Vitals

Files Used: Patients.txt
Purpose: Updates a patient's vitals in Patients.txt.

Updating Patient Vitals

This allows a nurse to update patients vitals in the Patients.txt file. It searches for the patient by ID and, if found, updates the patient's vitals with the new values provided. An appropriate error message is displayed if the file does not exist or an error occurs.

```
def update_patient_vitals(patient_id, new_vitals):
    """Update the patient's vitals in the patient records."""
    try:
        with open('patients.txt', 'r') as file:
            lines = file.readlines()

        with open('patients.txt', 'w') as file:
            for line in lines:
                if line.startswith(patient_id):
                    line = f"{patient_id},{new_vitals}\n"  # Update line with new vitals
                file.write(line)
        print("Patient vitals updated successfully.")
    except FileNotFoundError:
        print("Error: Patient records file not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Flow of Delete Function:

The delete function is used to remove a user data from a file after verifying the existence of the data in the file.Below are the step-by-step explanation of its flow:

1. **Input Prompt:**
    - The function starts by asking the user to input the userID of the user that needs to be deleted.
2. **Validation:**
    - It calls the validate_user_id function to ensure that the entered user_id is numeric and exists in the .txt file. If the validation fails, the function stops and prints user id not found.
3. **File Reading:**
    - The function is used to extract the data from the Users.txt file.
4. **Search Operation:**
    - The function goes through the file and locates the data provided by the user.
5. **Conditional Check:**
    - If a matching user is found The user record is deleted from that list by means of the del statement.The revised list is written back to the file with the write_file function.
    - A success message is displayed, confirming the deletion.
6. **File Update:**
    - The Users.txt file is refreshed with the updated user list, maintaining the header, and data consistency.
7. **Feedback to the User:**
    - The function prints a feedback if the deletion was successful or unsuccessful.

Example:

```
def Delete_User():  1 usage
    """Deleting the user after validating the existance of given user"""

    user_id = input(f"Enter a user to delete")
    if not validate_user_id(user_id):
        return #if user id is not numeric then stop the function

    heading, users_list = read_file("Users.txt")

    found = False
    for i in range(len(users_list)):
        if user_id == users_list[i][0]:  # Comparing user_id to find location of user data
            found = True
            del users_list[i]  # Removing user data from list
            break

    if found:
        write_file( file_name: "Users.txt",users_list, heading)
        print(f"User {user_id} has been deleted Successfully")

    else:
        print(f"User not found!")
```

Flow of Generating Reports:

Generate_Report()

The Generate_Report() function computes and generates the hospital report by collecting vital statistics, and then writing it into a report file (Report. txt). Let's understand the flow of this function, step by step:

Counting Total Patients:

o Then the function initializes a counter (total_patients = 0), after that it reads the patient data from the Patients. Learning about each entry in the.txt file using read_file() function It pulls the data (not counting the heading), counts how many patients are in the file. The count is stored in total_patients.

o EXAMPLE:

```
def Generate_Report():
    """This function computes and generates a r
    total_patients = 0
    heading, data = read_file("Patients.txt")
    for line in data:
        total_patients += 1
```

Total Doctors and Nurses:

o The get_user_counts() is assumed to get total no of doctors and nurses in the hospital system. These sums will be stored in total_doctors and total_nurses respectively.

o Example

```
def get_user_counts():
    """This function counts no. of doctors and nurses (staff) in hospital"""
    total_doctors = 0
    total_nurses = 0
    heading, data = read_file("Users.txt")

    for i in range(len(data)):
        role = data[i][2].strip()
        if role == 'Doctor':  # assuming role [2] in file where heading = UserID,Name,Role,Contact
            total_doctors += 1
        elif role == 'Nurse':
            total_nurses += 1
    return total_doctors, total_nurses
```

```
total_doctors, total_nurses = get_user_counts()  # getting total doctors and nurses
```

Data Resources — Reading Hospital Reading Hospital Resource Data:

o The next step the function reads hospital resource data from the Resources. Using the read_file() function, you read the.txt file It includes the number of total beds, occupied beds, and available beds. Processing the first line, the relevant details would be stored in total_beds, occupied_beds and available_beds.

Example

```
heading, data2 = read_file("Resources.txt")
total_beds = int(data2[0][0])
occupied_beds = int(data2[0][1])
available_beds = int(data2[0][2])
xay_Machines = int(data[0][3])
Ultrasound_Machines = int(data[0][4])
```

Writing Data to Report File:

o Then the write_file() function is called two times to write the generated report. As a first step it writes the data out of the Patients. xlsx or.txt file (as well as the patient information)Is in the Report. txt file. It then writes a custom summary of the hospital report, containing the total number of patients, doctors, nurses, and state of bed occupancy.

Example:

```
write_file("Report.txt", data)
report_data = [(f"""Hospital Report\nTotal patients: {total_patients}\nTotal Doctors: {total_doctors}\nTo
write_file("Report.txt",[report_data])
print("Report has been generated Successfully!")
```

Summary of the Flow

The function gathers data from various sources (patient count from Patients. txt, doctor/nurse count from get_user_counts(), hospital resource info from Resources. txt).

Data Processing: Used for count of patients, fetching doctors and nurses, extraction of beds and their status.

Writing the Report: The system writes the processed data to a report file (Report. txt) patient details and a hospital summary.

In this example, we run the program with a command such as, "python report_generator.py", which generates a report and writes it to a file.

Example Output in Report File

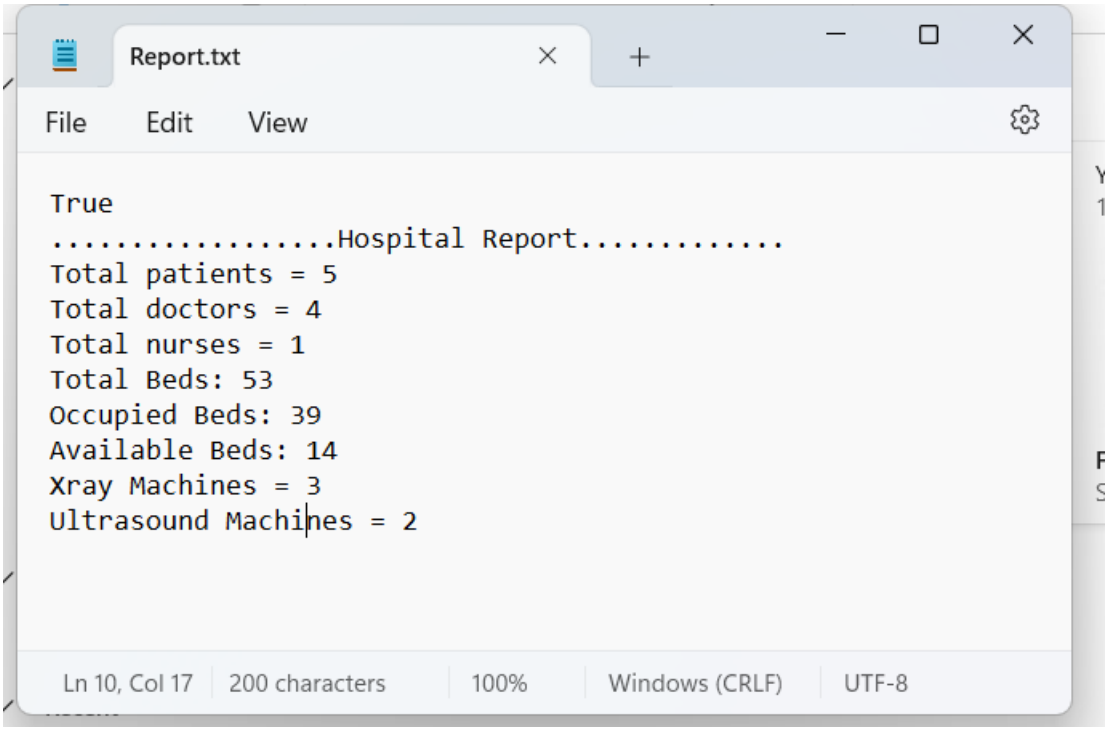
The report it generates will look something like:

Example of output file

It then saves the report to Report. txt which are available to view and print.

Conclusion

The data flow for the Generate_Report() function is intuitive: it takes out requirement data from a few sources, processes it, calculates key hospital metrics, and dumps the result to a report file. Fhe function formats and displays all necessary hospital statistics, such as patient numbers and bed availability, in an easily readable manner accessible to end-users or administrators.



“Additional Features”

Additional Feature by Administrator (Kazim - TP084275)

Code (first half):

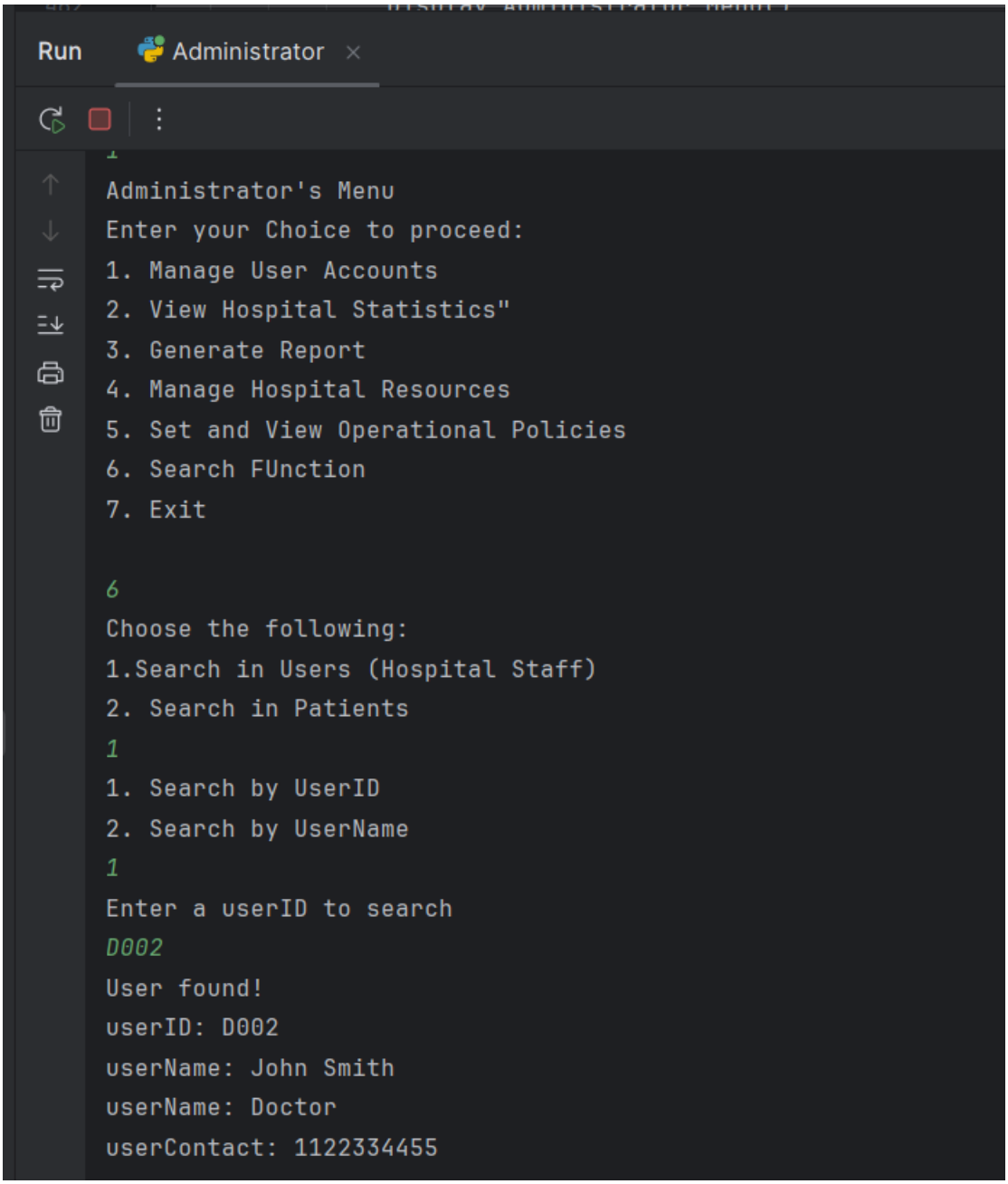
```
def Search_User():  1 usage
    """Special Feature: I have added a Search function to find and display user by Name or ID"""

    CHOICE = int(input("Choose the following: \n1.Search in Users (Hospital Staff) \n2. Search in Patients\n"))

    if CHOICE == 1:             #Performs search in Users.txt file (Hospital Staff)
        search_type = int(input("1. Search by UserID\n2. Search by UserName\n"))

        if search_type == 1:
            user_id = input("Enter a userID to search\n")
            heading, data = read_file("Users.txt")
            found = False
            for line in data:
                if line[0] == user_id:
                    found = True
                    print(f"User found! \nuserID: {line[0]} \nuserName: {line[1]} \nuserName: {line[2]} \nuserContact: {line[3]}\n")
                    break
            if not found:
                print(f"No user found with userID: {user_id}")

        elif search_type == 2:
            user_name = input("Enter a userName to search\n").lower()
            heading, data = read_file("Users.txt")
            found = False
            for line in data:
                if line[1].lower() == user_name:
                    found = True
                    print(f"User found! \nuserID: {line[0]} \nuserName: {line[1]} \nUserRole: {line[2]} \nUserContact: {line[3]}\n")
                break
            if not found:
                print(f"No user found with userName: {user_name}")

        else:
            print("Invalid search type. Please enter 1 or 2.")

    elif CHOICE == 2:               #Performs search in Patients.txt file
```

Code (Second half)

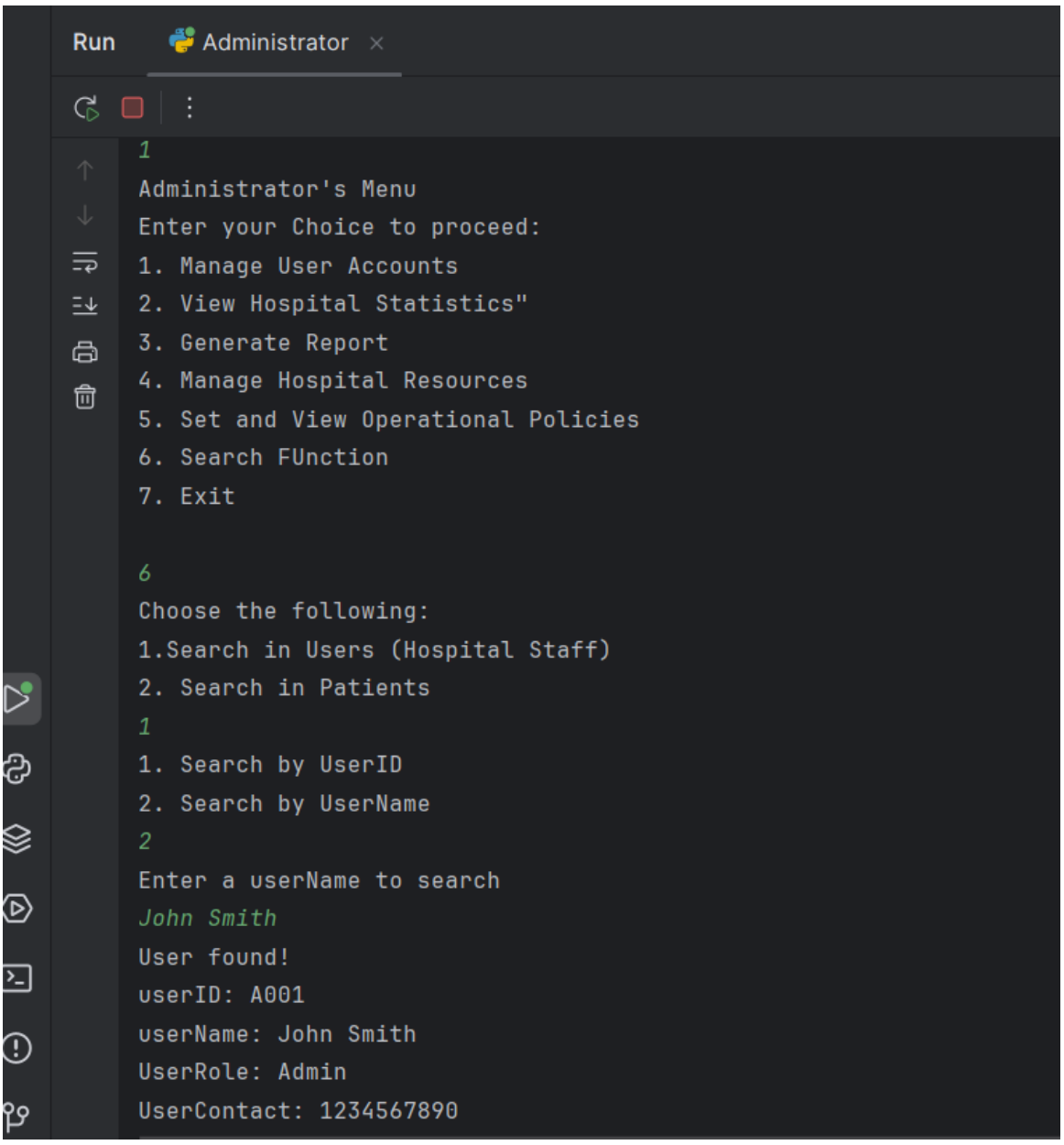
```
68  def Search_User():  1 usage
103     elif CHOICE == 2:               #Performs search in Patients.txt file
104         search_type = int(input("1. Search by PatientID\n2. Search by Patient Name\n"))
105         if search_type == 1:            #Search by ID
106             patient_id = input("Enter a PatientID to search\n")
107             heading, data = read_file("Patients.txt")
108             found = False
109             for line in data:
110                 if line[0] == patient_id:
111                     found = True
112                     print(f"""Patient found! \nPatientID: {line[0]} \nName: {line[1]} \nAge: {line[2]}
113                     \nGender: {line[3]}\nContact: {line[4]}\nAddress: {line[5]} \nCondition: {line[6]}
114                     \nDoctor Assigned: {line[7]}\n Insurance Status: {line[8]}\n""")
115                     break
116             if not found:
117                 print(f"No patient found with patientID: {patient_id}")
118
119         elif search_type == 2:          #Search by Name
120             patient_name = input("Enter a Patient Name to search\n").lower()
121             heading, data = read_file("Patients.txt")
122             found = False
123             for line in data:
124                 if line[1].lower() == patient_name:
125                     found = True
126                     print(f"""Patient found! \nPatientID: {line[0]} \nName: {line[1]} \nAge: {line[2]}
127                     \nGender: {line[3]}\nContact: {line[4]}\nAddress: {line[5]} \nCondition: {line[6]}
128                     \nDoctor Assigned: {line[7]}\n Insurance Status: {line[8]}""")
129                     break
130             if not found:
131                 print(f"No patient found with Name: {patient_name}")
132
133         else:
134             print("Invalid search type. Please enter 1 or 2.")
135     else:
136             print("Invalid search type. Please enter 1 or 2.")
```

Output of Search Function:

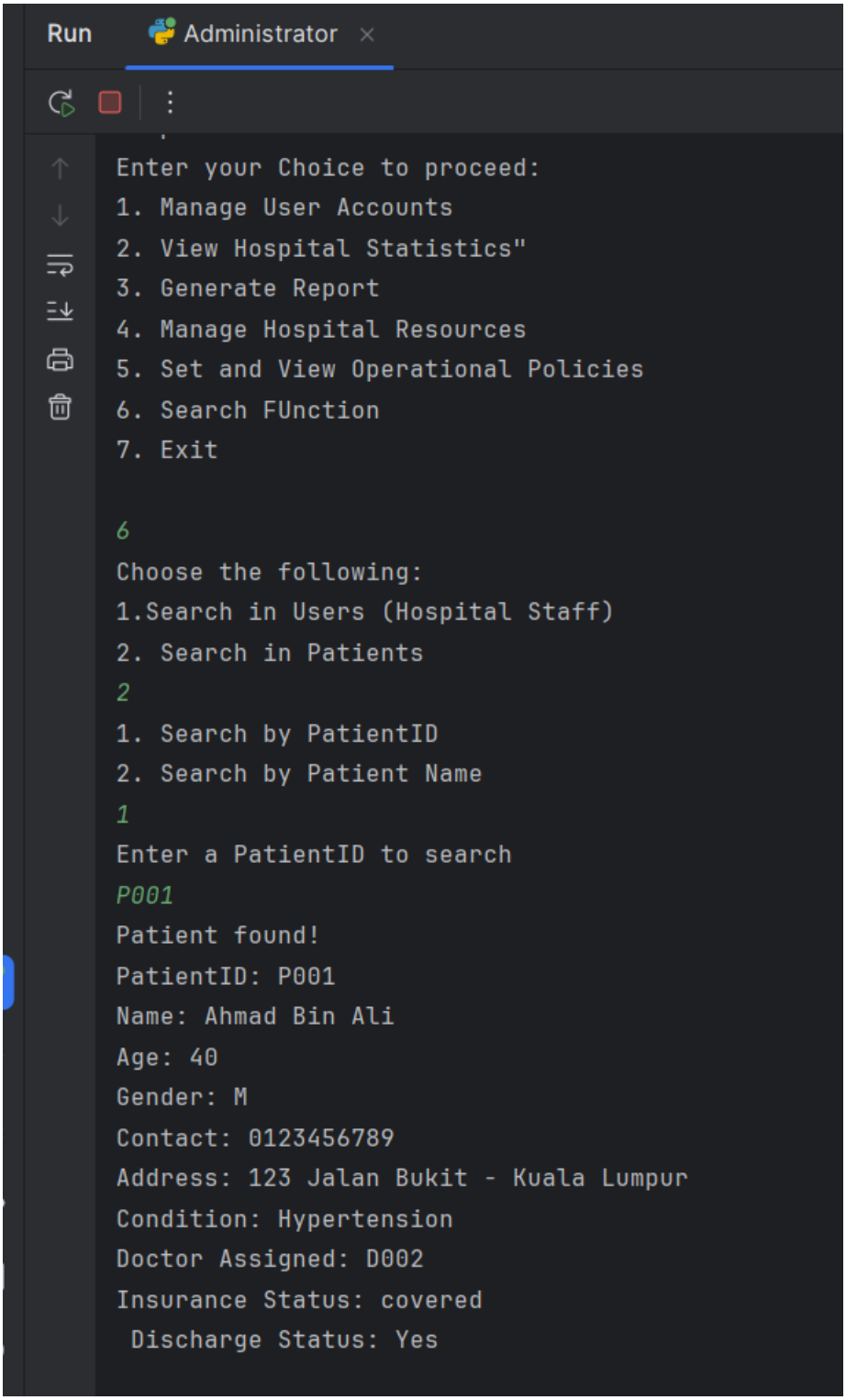
- **Search by UserID:**



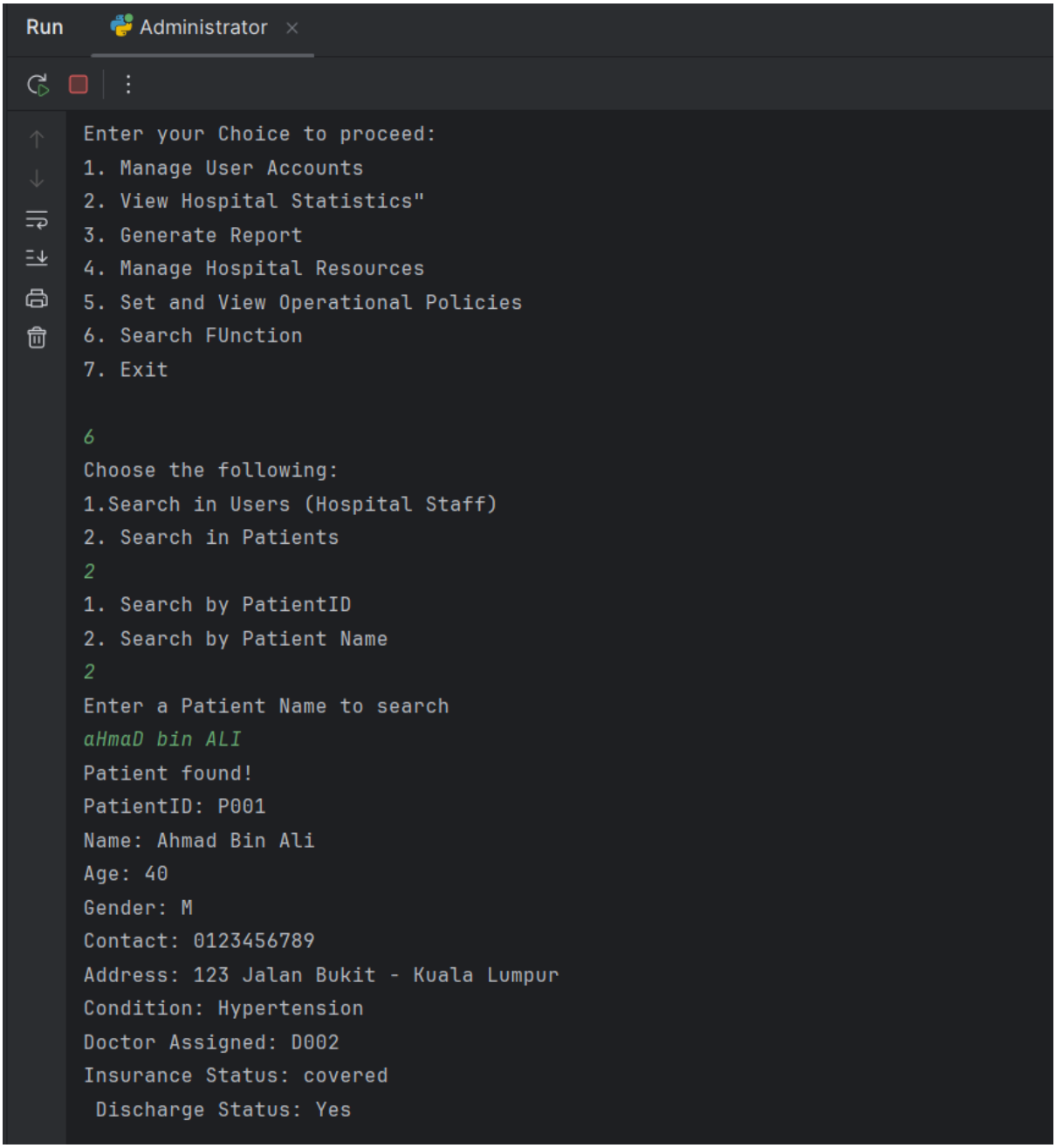
- **Search by UserName:**



- **Search by PatientID:**



- **Search by Patient Name:**



Explaination:

Enhancements to Administrator Role: Search FunctionAs a key enhancement to the Administrator role, the Search Function now allows hospital staff (Users) and patients to quickly search for information related to data. Two handy search methods are offered, one by ID with boys and girls of the same name, and by Name used as case-insensitive. The function reads dynamically the data from the files (Users. txt or Patients. txt) and it gives you detailed information on staff as User ID, Name, Role and Contact or complete patient information from medical to discharge status. Input validation secure the searches and clear error messages gracefully handles inappropriate inputs and not found records. The ODA functionality has the capability of storing two separate datasets on a same machine, and this feature is built with ease of use and scale-ability in mind, making it efficient, reliable, and successful for daily administrative processes.

Feature Overview

Using the Search Function, the Administrator can:

Hospital Staff (Users) search:

o User ID: Find personnel by their ID.

o By User Name: Name search (case insensitive) for staff.

Search in Patients:

o By Patient ID: Get patient information by unique patient ID.

o By Patient Name: Search for patients by name (not case sensitive).

Special Characteristics

Dual Dataset Scope:

o The function is also flexible with the ability to search in both hospital staff (Users) and patients.

Multiple Search Types:

o Provides deux search methodologies for every dataset:

By ID: Ensures you will get the correct data.

 By Name: Search not case sensitive, hence add convenience to users.

Comprehensive Details:

o For patients, it shows specific details, such as:

actually its all the pieces of information that are required for patient.: Patient ID, Name, Age, Gender, Contact No, Address, Condition and doctor assigned and Insurance Status and Discharged Status.

o User ID, Name, Role and Contact for staff

Error Handling:

o Validates inputs like User ID and Patient ID which help to handle invalid inputs gracefully.

o Shows clear messages when no suitable records found

Reusability and Scalability:

o The function is modular as it used different components here (Read_File, validation functions), so it is reusable and is easy to extend.

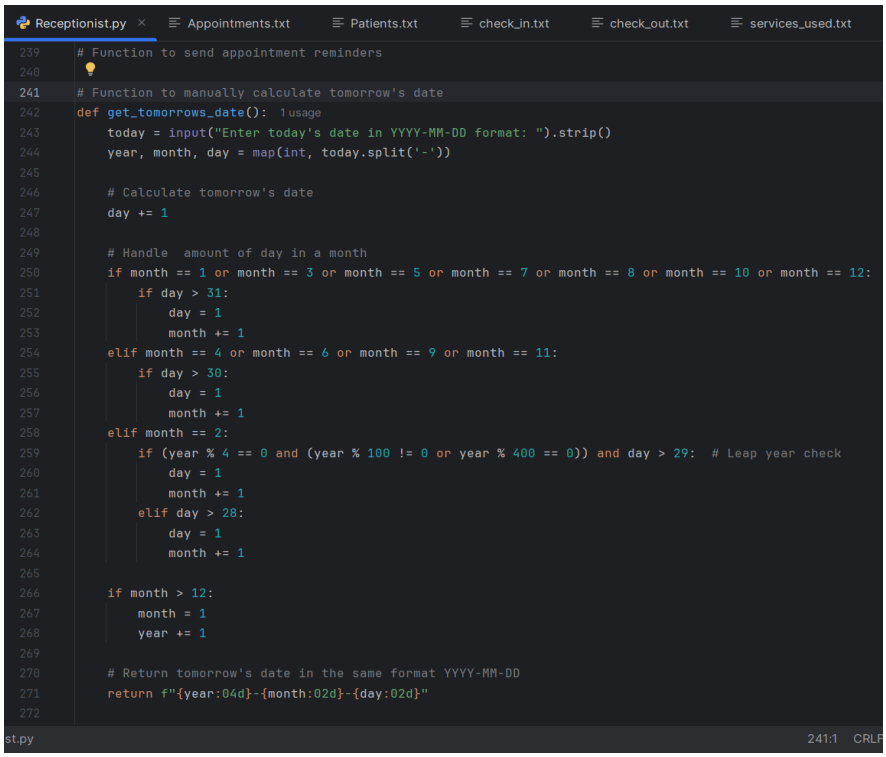
o Tickets for other search criteria (e.g., search by role, contact) can be created with little to no changes to the existing structure.

Additional Feature by Receptionist

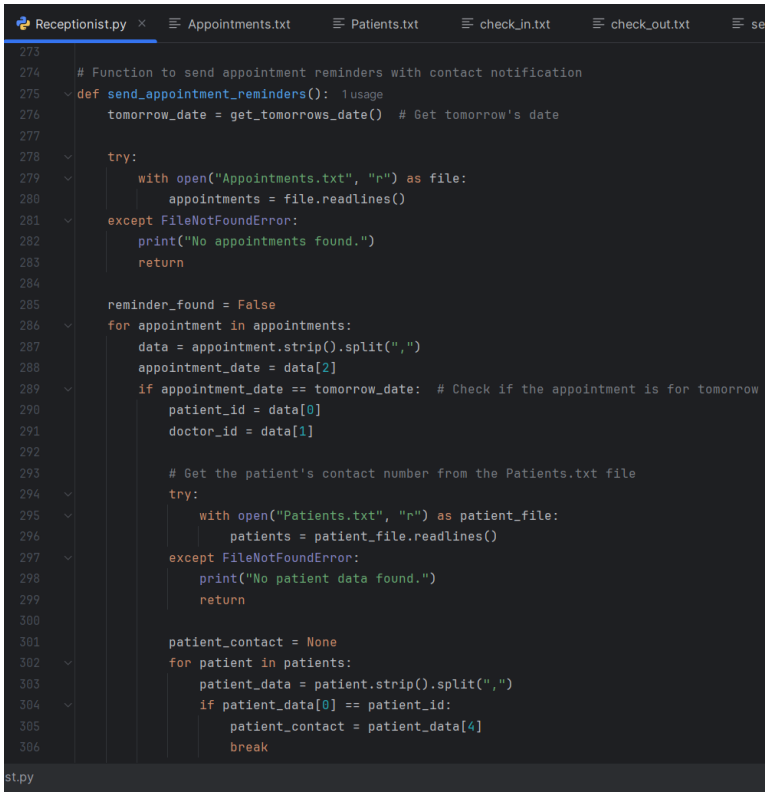
Send Reminder special feature Explanation:

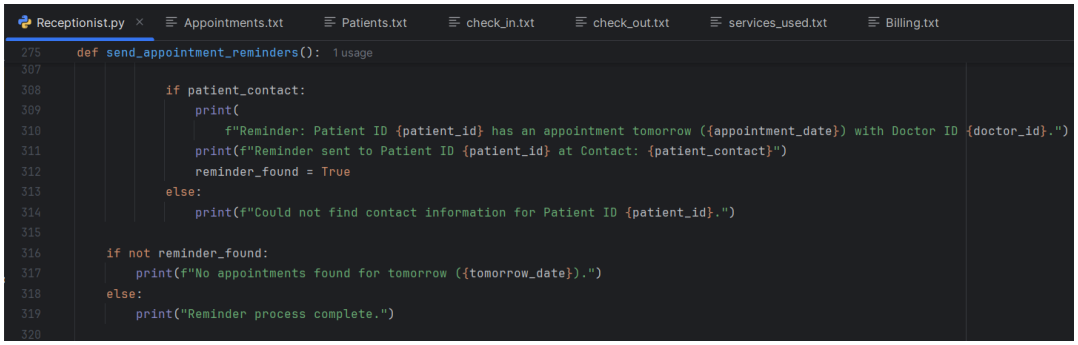
The send_appointment_reminders() function calls alerts patients of next day appointments in a bid to minimize no-shows that compromise the working of a hospital. It begins with finding out tomorrow's date using a function called get_tomorrows_date() which takes care of such special conditions as month end and leap year. Once the date is set, it calls the `Appointments.txt` file, each line encodes information about scheduled appointments there in with details of the patient ID, the doctor ID, date and time. The function loops through these records as it searches for the appointments that are matching tomorrow's date. For each match it also brings the patient id together with the doctor id. To send a reminder, it matches the patient ID with the records in the Patients.txt file to get the patients phone number. If the contact is found, a reminder message is created and sent, it appears like a notification. The message contains the patient ID, appointment date, doctor ID and a contact number or an email address. If there are no appointments for tomorrow, the function tells the user. Likewise, if the patient's contact information is not available, it records a log message to show the problem. These notifications are useful to keep patient relationships active, to better manage the schedule of the doctor, and for other hospital organization reasons. At the present time, this implementation just outputs reminders to the console, which is less useful for time-sensitive tasks, however, it can be modified to make actual notifications via, for example, SMS or email, which would make the tool much more practical and effective in real-life environments.

Code for function get_tomorrows_date()

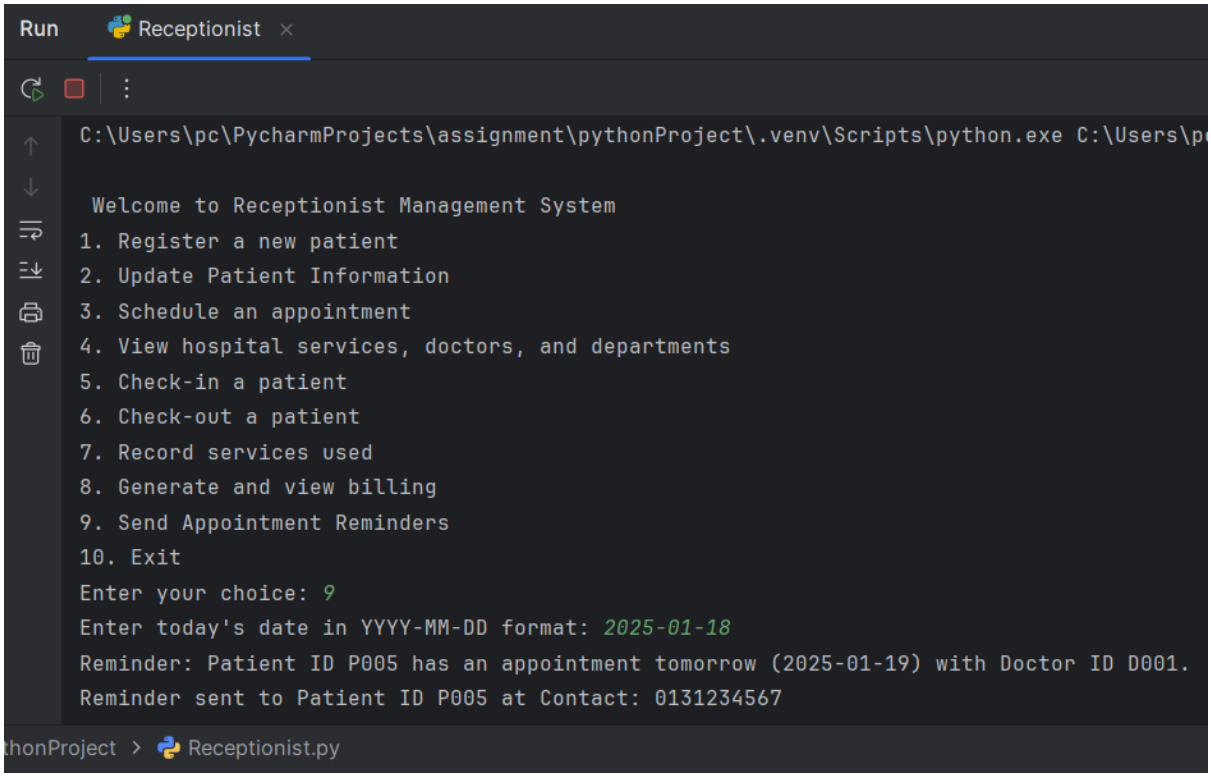


Code for the function send_appointment_reminders()


Receptionist.py
Appointments.txt
Patients.txt
check_in.txt
check_out.txt
Function to send appointment reminders with contact notification
def send_appointment_reminders(): 1 usage
tomorrow_date = get_tomorrows_date() # Get tomorrow's date
try:
with open("Appointments.txt", "r") as file:
appointments = file.readlines()
except FileNotFoundError:
print("No appointments found.")
return
reminder_found = False
for appointment in appointments:
data = appointment.strip().split(",")
appointment_date = data[2]
if appointment_date == tomorrow_date: # Check if the appointment is for tomorrow
patient_id = data[0]
doctor_id = data[1]
# Get the patient's contact number from the Patients.txt file
try:
with open("Patients.txt", "r") as patient_file:
patients = patient_file.readlines()
except FileNotFoundError:
print("No patient data found.")
return
patient_contact = None
for patient in patients:
patient_data = patient.strip().split(",")
if patient_data[0] == patient_id:
patient_contact = patient_data[4]
break


Receptionist.py
Appointments.txt
Patients.txt
check_in.txt
check_out.txt
services_used.txt
Billing.txt
def send_appointment_reminders(): 1 usage
if patient_contact:
print(
f"Reminder: Patient ID {patient_id} has an appointment tomorrow ({appointment_date}) with Doctor ID {doctor_id}.")
print(f"Reminder sent to Patient ID {patient_id} at Contact: {patient_contact}")
reminder_found = True
else:
print(f"Could not find contact information for Patient ID {patient_id}.")
if not reminder_found:
print(f"No appointments found for tomorrow ({tomorrow_date}).")
else:
print("Reminder process complete.")

Output of send appointment reminder function



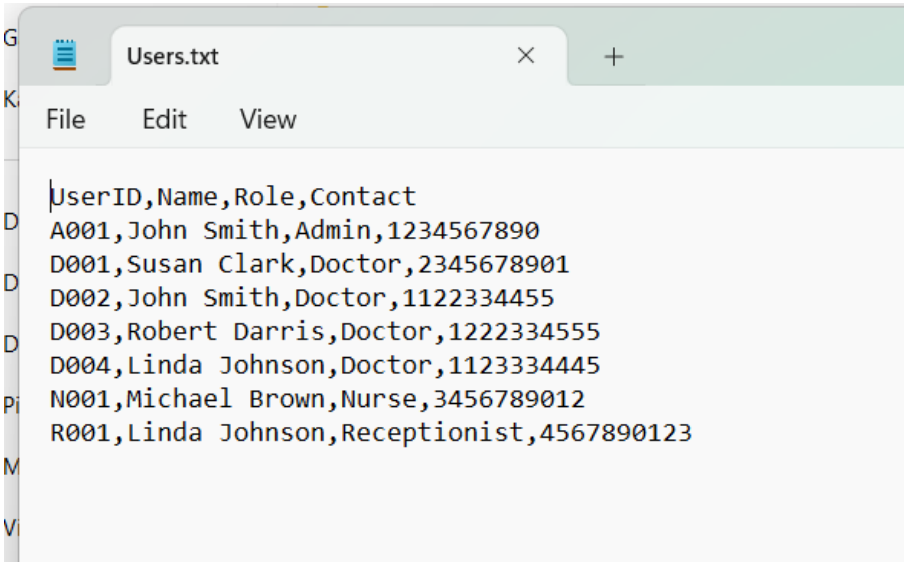
"Screenshots of Input/Output with Explanation"

Input/Output of Administrator Role:

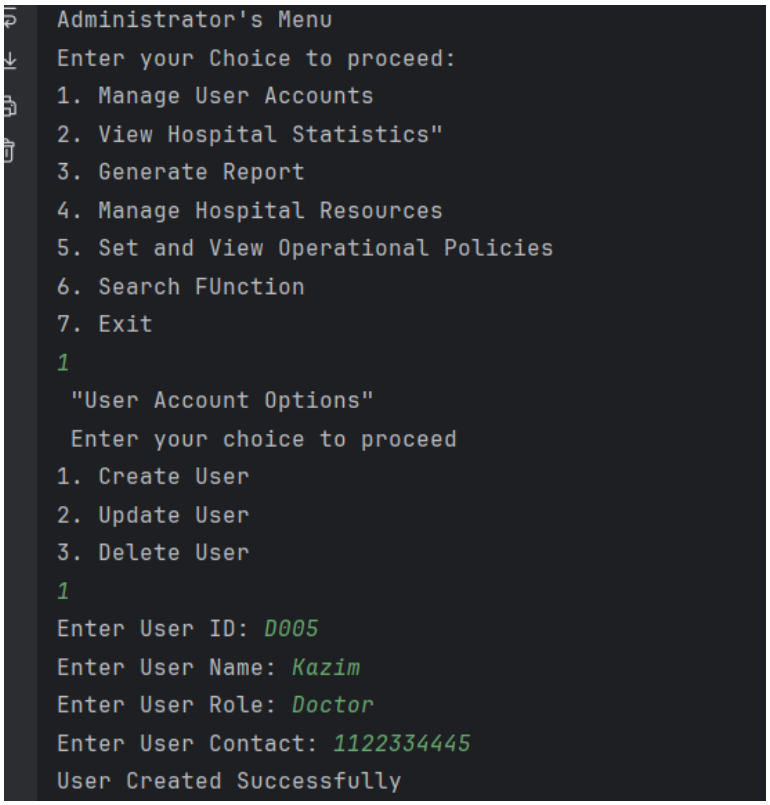
1. Manage user accounts for all roles (create, update, delete users)

Creating a new user:

- File before creating a new user.

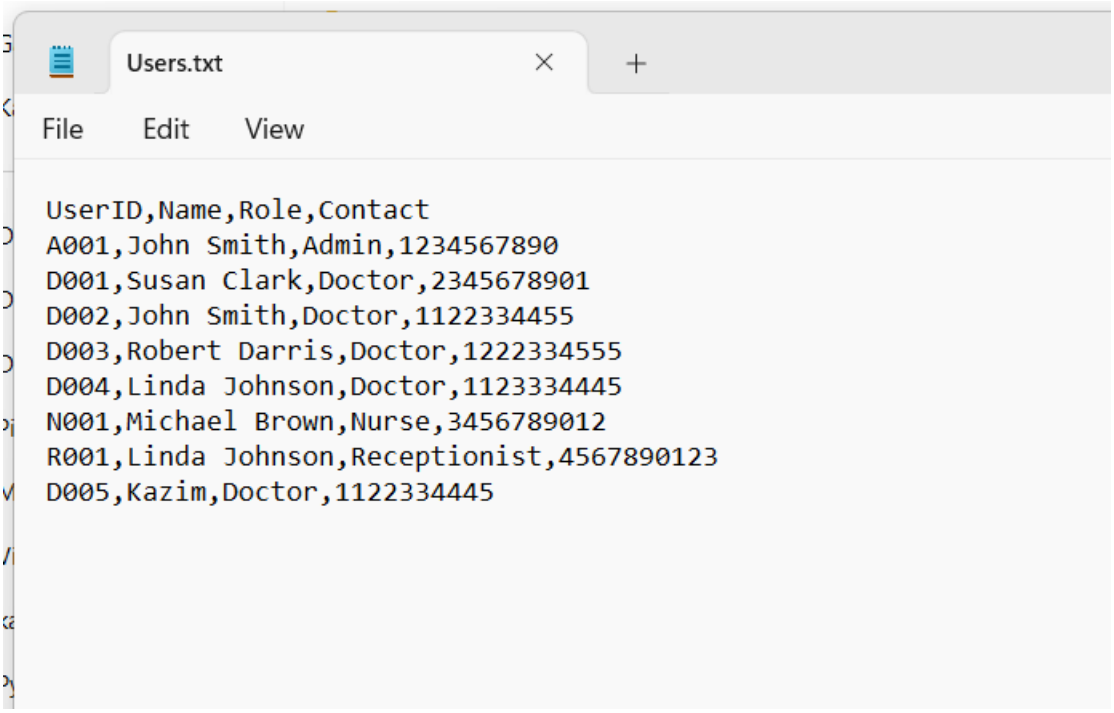


Input:



Output:

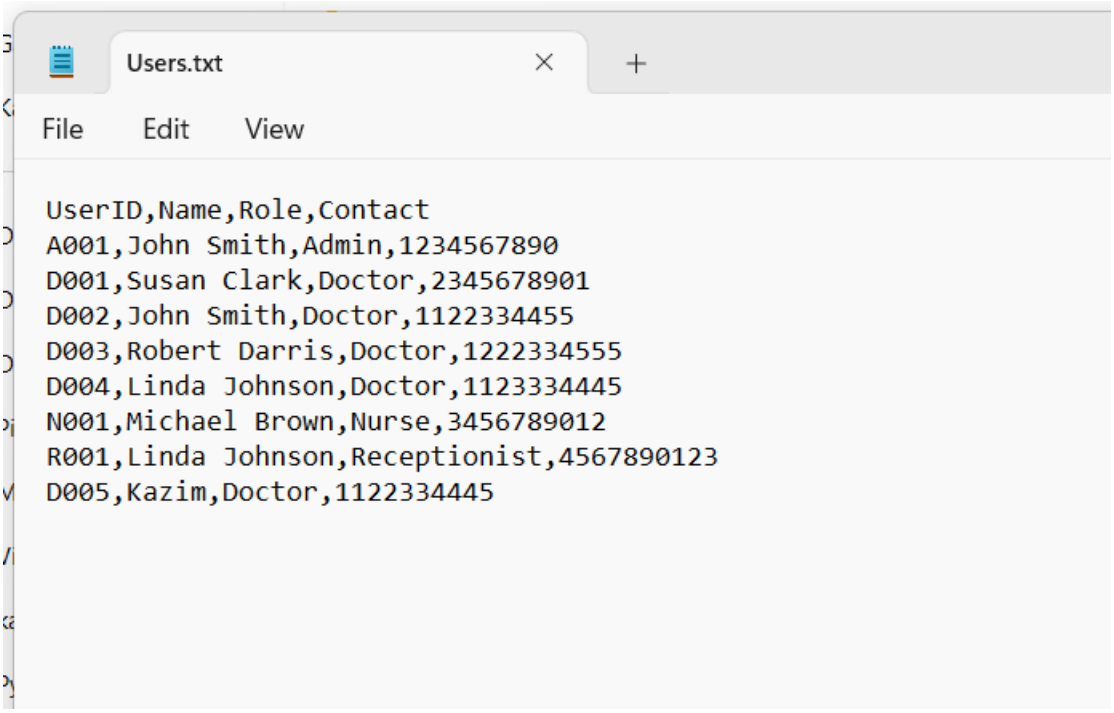
- File after creating a new user



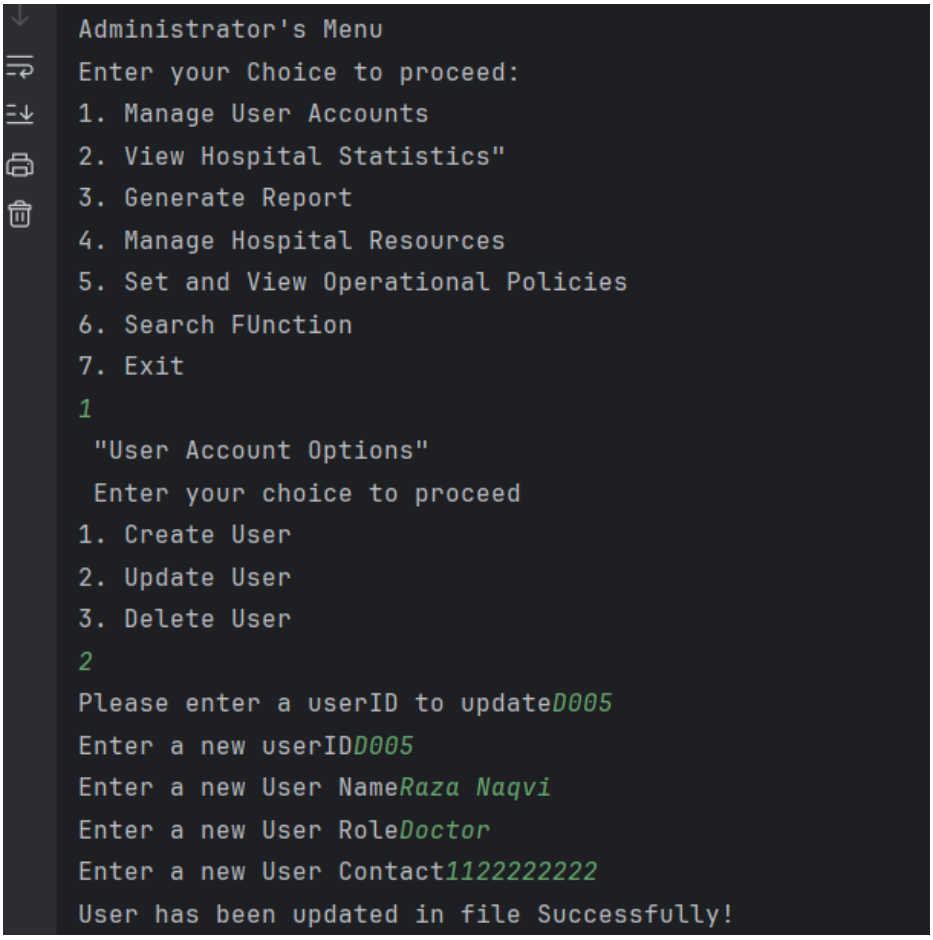
Explaination:
The **Create_User()** function validates the inputs of user and when all validations pass, the new user details are appended to the **Users.txt** file. In the example above UserID (**D005**), Name(**Kazim**), Role(**Doctor**), Contact(**1122334445**) were provided by the user in input and after successful validation it was appended to Users.txt file.

Updating user:

- File before updating user

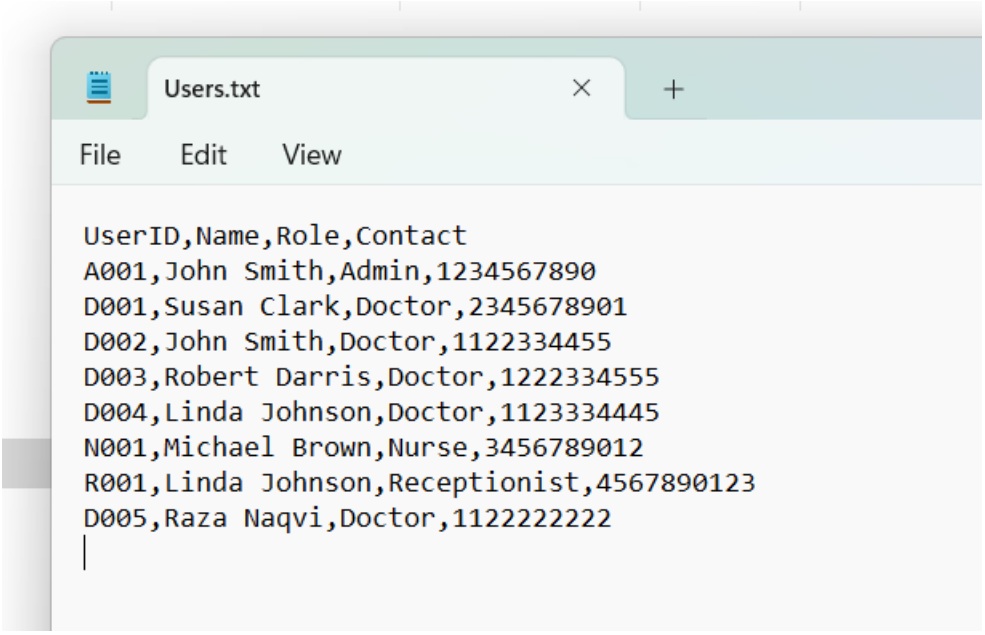


Input:



Output:

- File after updaing User

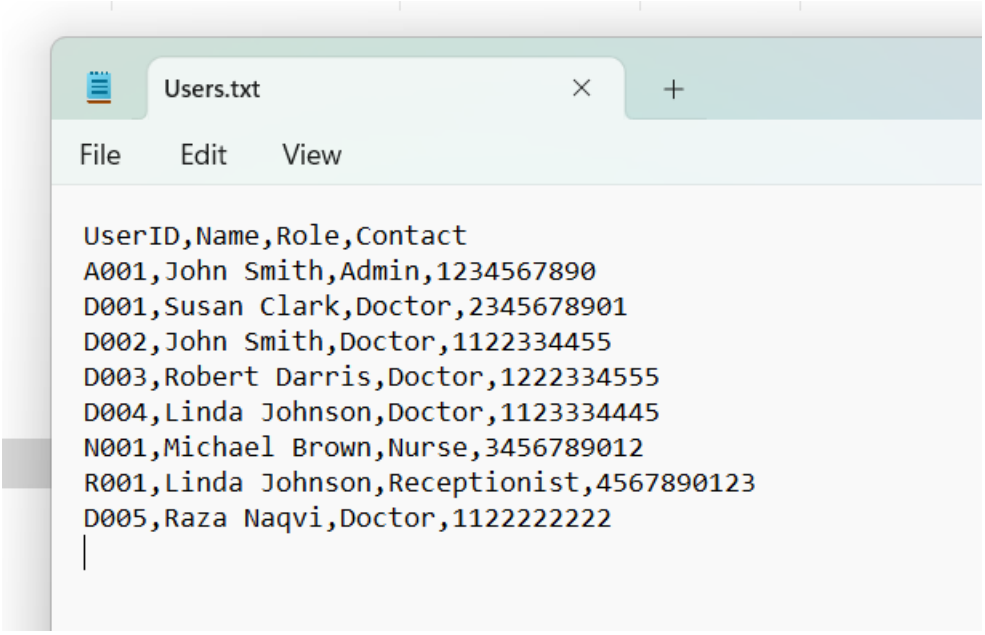


Explaination:

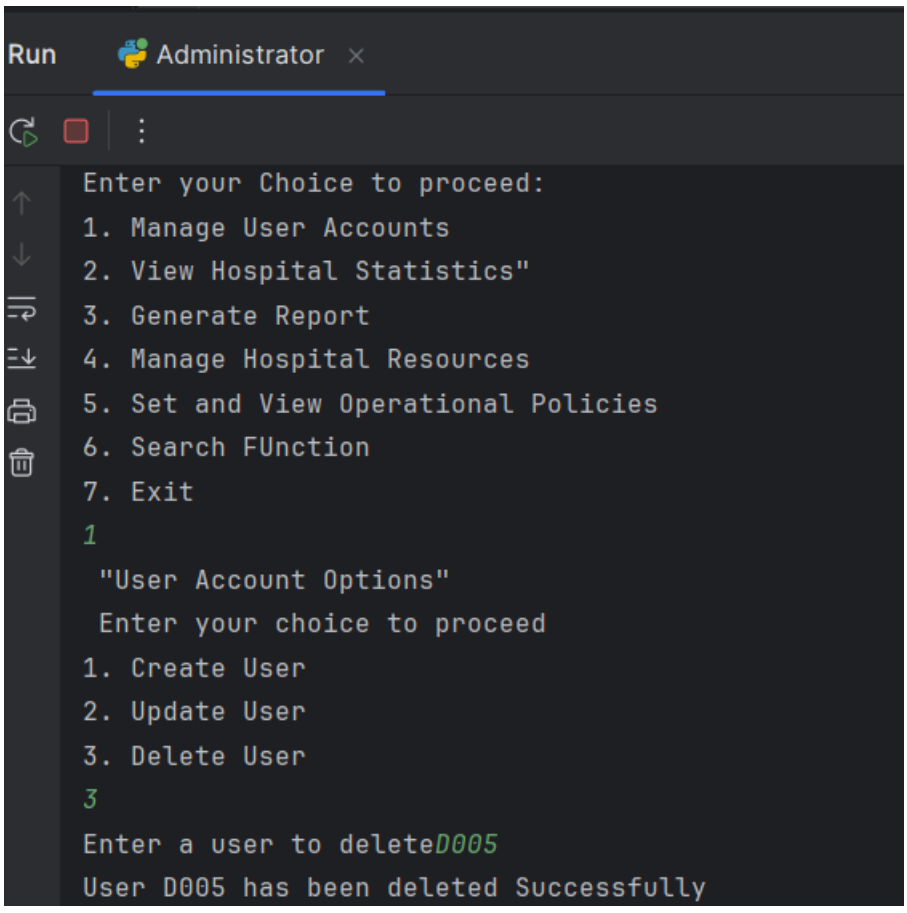
The **Update_User()** function checks if the entered User_id which user wants to update is existing in the Users.txt file or not. Then the function validates the inputs of user and when all validations pass, the new details are updated and data is reqritten to the **Users.txt** file. In the example above UserID (**D005**) was provided by the user first and then after validation new details were asked by the user upon which this was entered UserID (**D005**) Name(**Raza Naqvi**), Role(**Doctor**), Contact(**1122222222**) were provided by the user in input and after successful validation it was overwrite at the location of userid (**D005**) where previously UserID (**D005**), Name(**Kazim**), Role(**Doctor**), Contact(**1122334445**) existed.

Deleting a user:

- File before deleting a user.

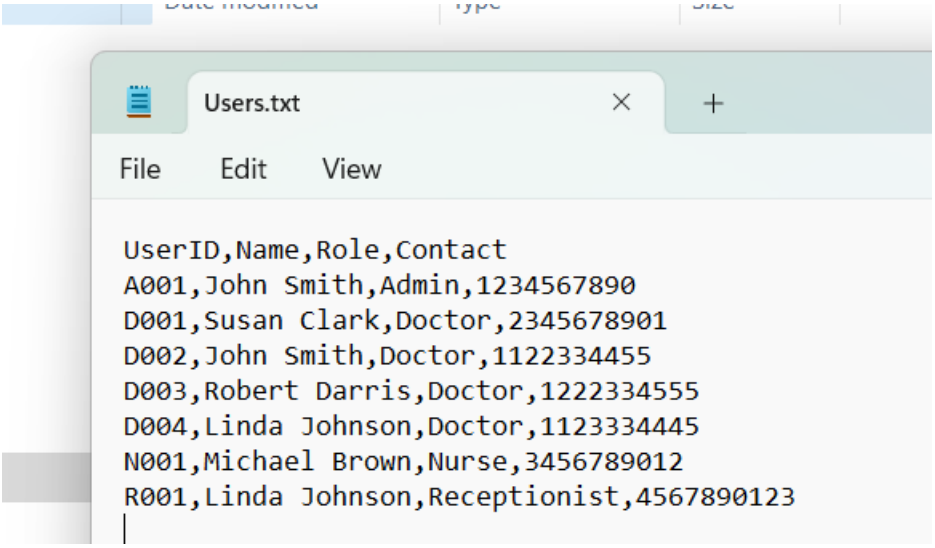


Input:



Output:

- File after deleting a user.

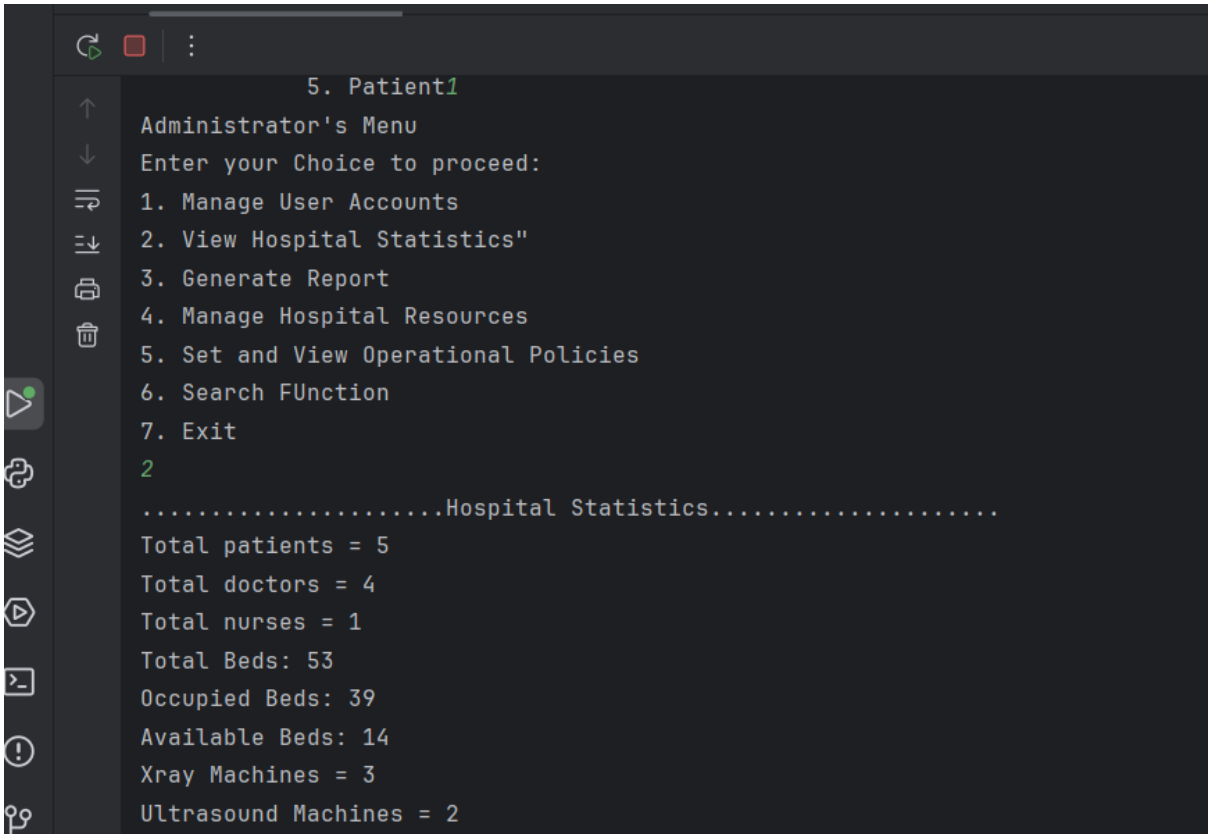


Explaination:

The **Delete_User()** function checks if the entered **User_id** which user wants to delete is existing in the Users.txt file or not. Then after the successful validation function removes the user data from **Users.txt** file. In example above UserID (**D005)** was entered by the user and after checking if this user id exists the user details of (**D005**) were removed from **Users.txt** file.

2. View overall hospital statistics (e.g., total number of patients, doctors, available beds)

Input:

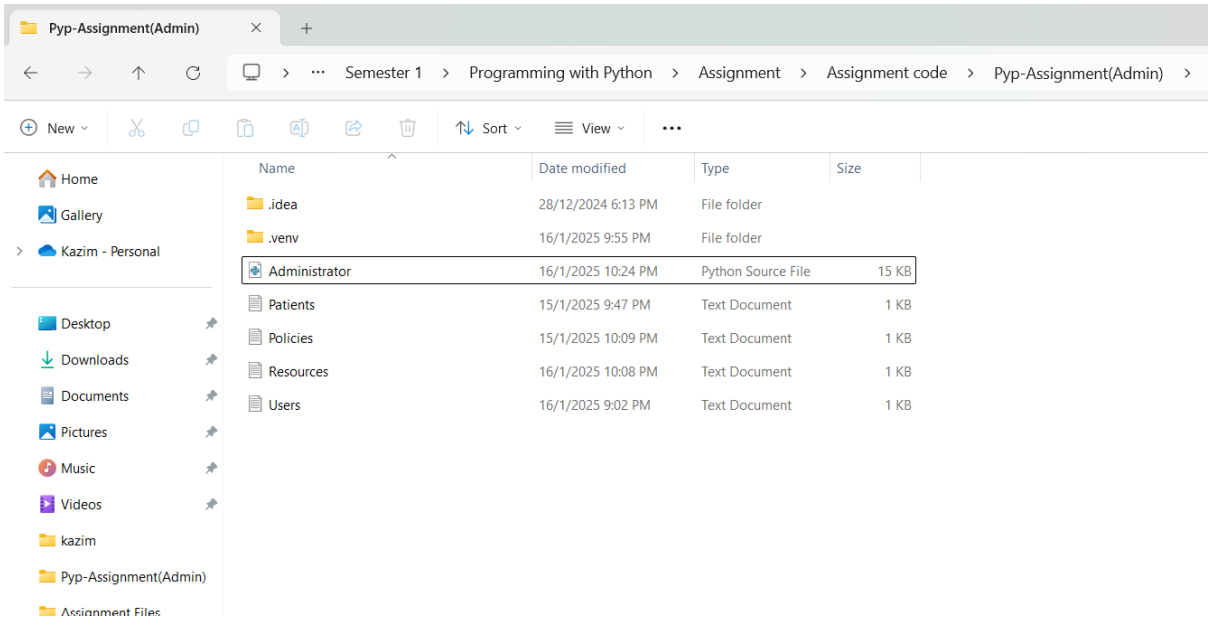


Explaination:

The Function **View_Statistics()** reads data from **Users.txt, Patientis.txt** and **Resources.txt** to compute and display as Hospital Statistics.

3. Generate reports on hospital usage and occupancy rates.

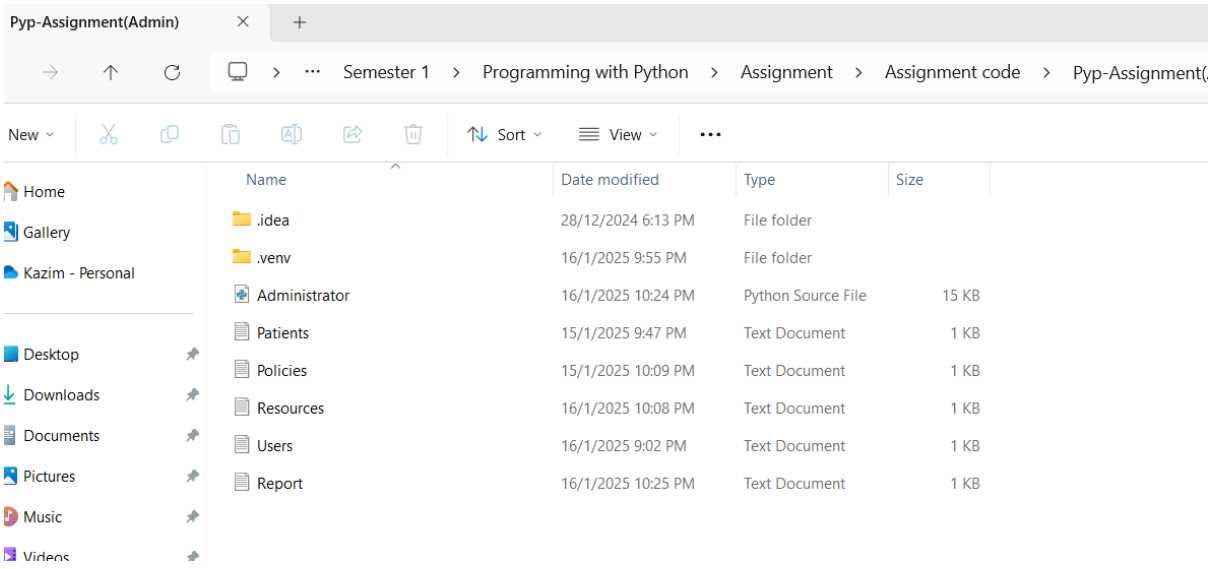
Before generating report



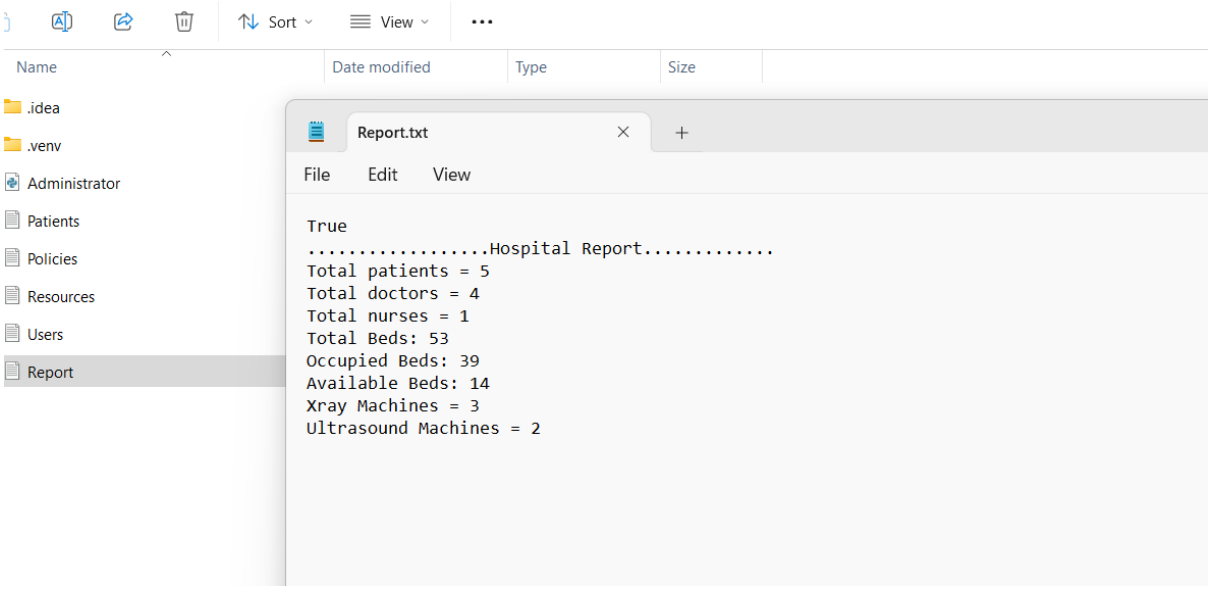
Input:

```
Administrator's Menu
Enter your Choice to proceed:
1. Manage User Accounts
2. View Hospital Statistics"
3. Generate Report
4. Manage Hospital Resources
5. Set and View Operational Policies
6. Search FUnction
7. Exit
3
Report has been generated Successfully!
```

Directory after report generated:



Output:



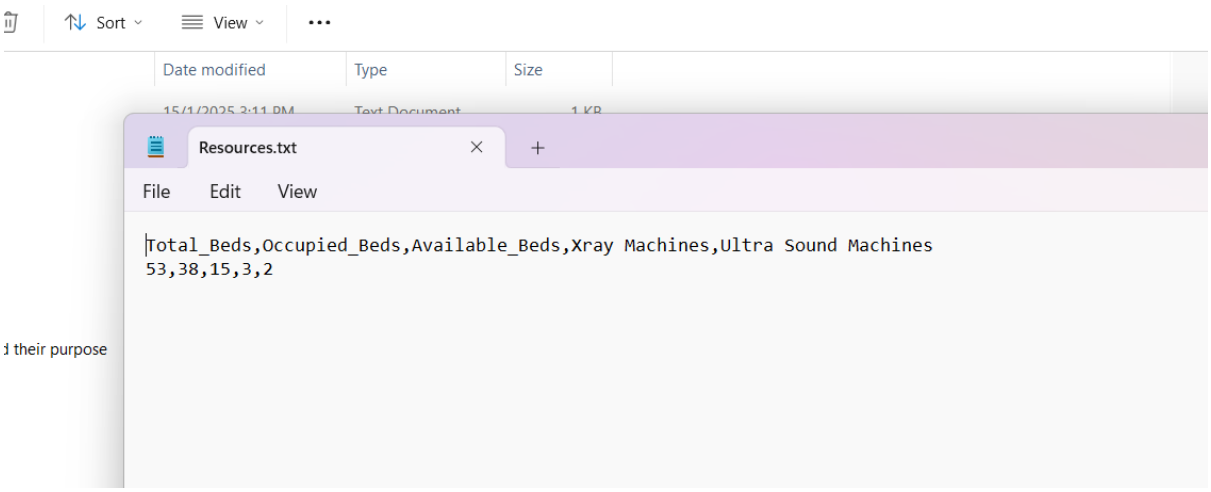
Explaination:

The function **Generate_Report()** combines data from **Users.txt, Patients.txt** and **Resources.txt** into a structured report and creates a new file **Report.txt** and save the report in it.

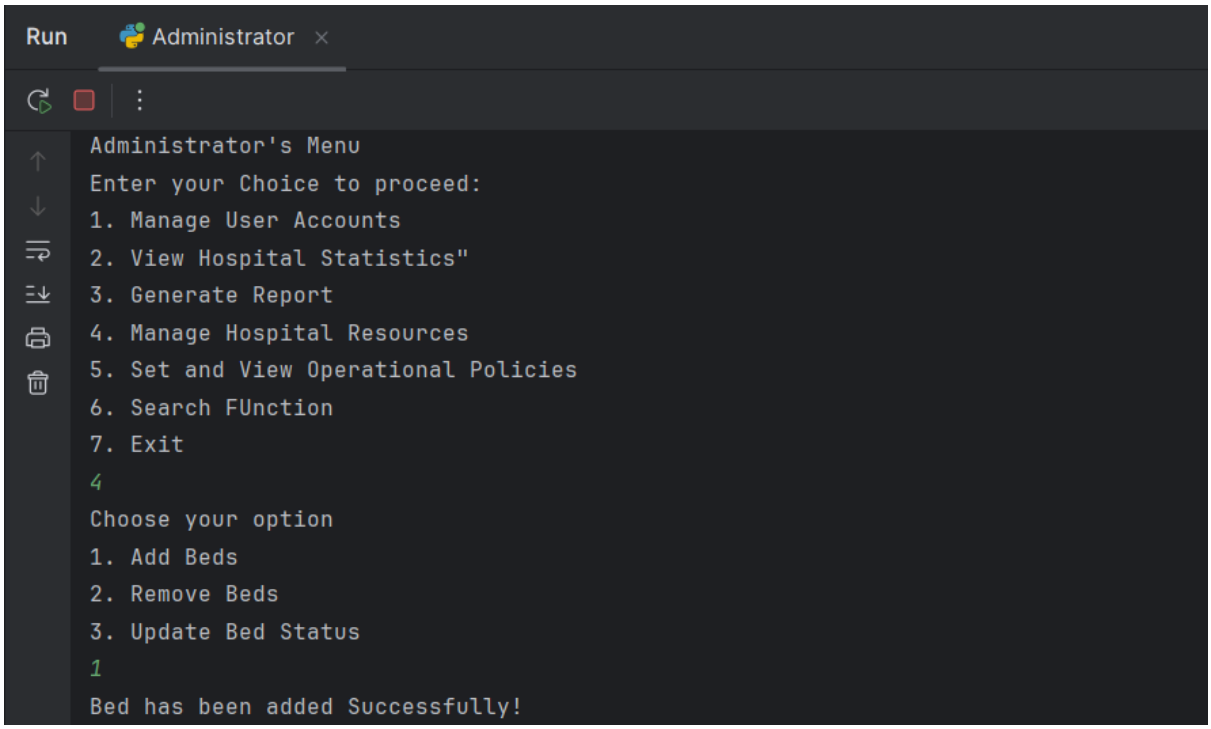
4. Manage hospital resources (e.g., add or remove beds, update facility status).

Adding a Bed:

- File before adding a bed

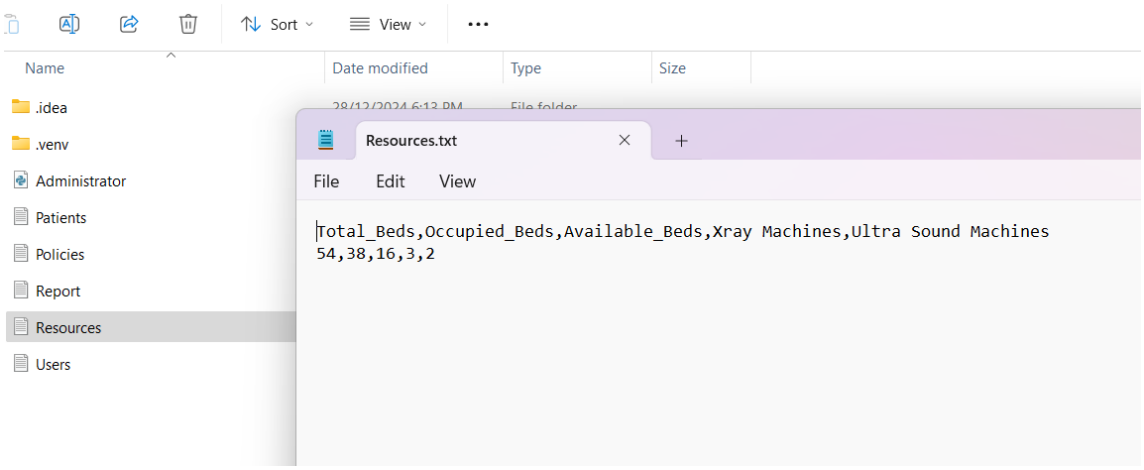


Input:



Output:

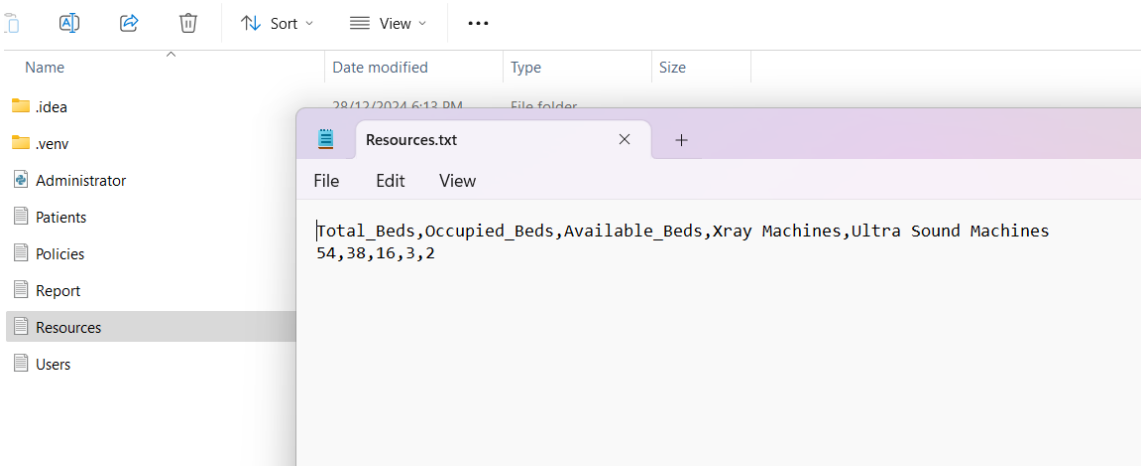
- File after adding a bed



Explaination:
The **Add_Bed()** function increments the **Total_Beds** and **Available_Beds** count in **Resources.txt** file by 1. In the example above in Resources file, the **Total_Beds** = **53**, **Available_Beds** = **15** initially, which changed when a bed was added and new data in Resources became Total_Beds = **54** and Available_Beds = **16.**

Removing a Bed:

- File before removing a bed

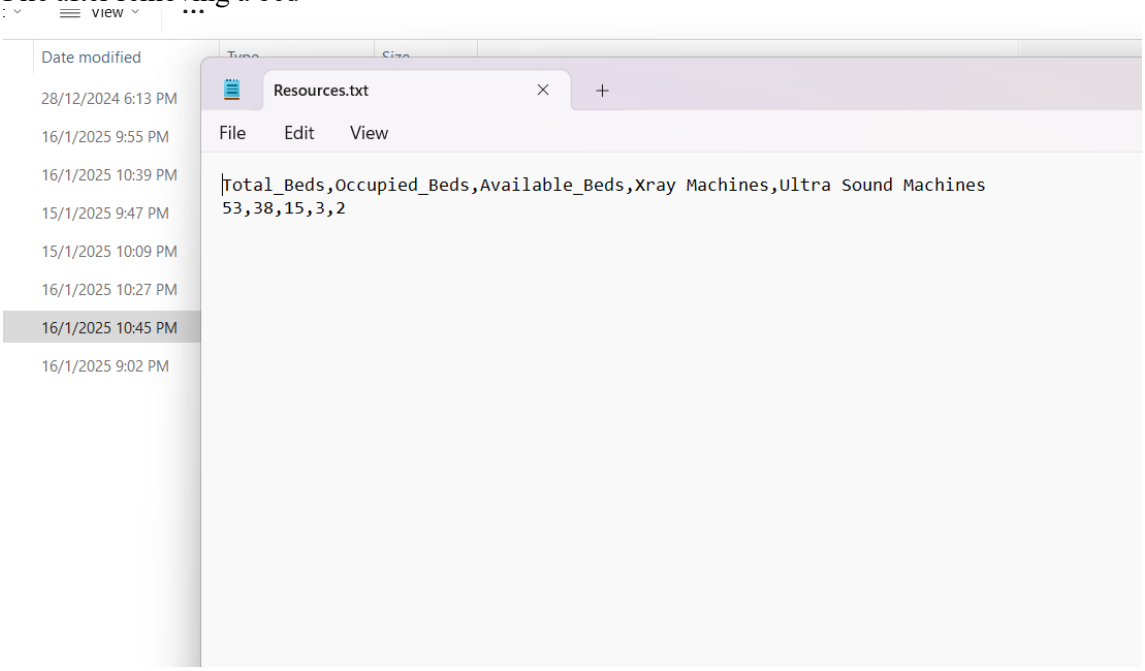


Input:

```
Enter your Choice to proceed:
1. Manage User Accounts
2. View Hospital Statistics"
3. Generate Report
4. Manage Hospital Resources
5. Set and View Operational Policies
6. Search FUnction
7. Exit
4
Choose your option
1. Add Beds
2. Remove Beds
3. Update Bed Status
2
Bed has been removeed Successfully!
```

Output:

- File after removing a bed

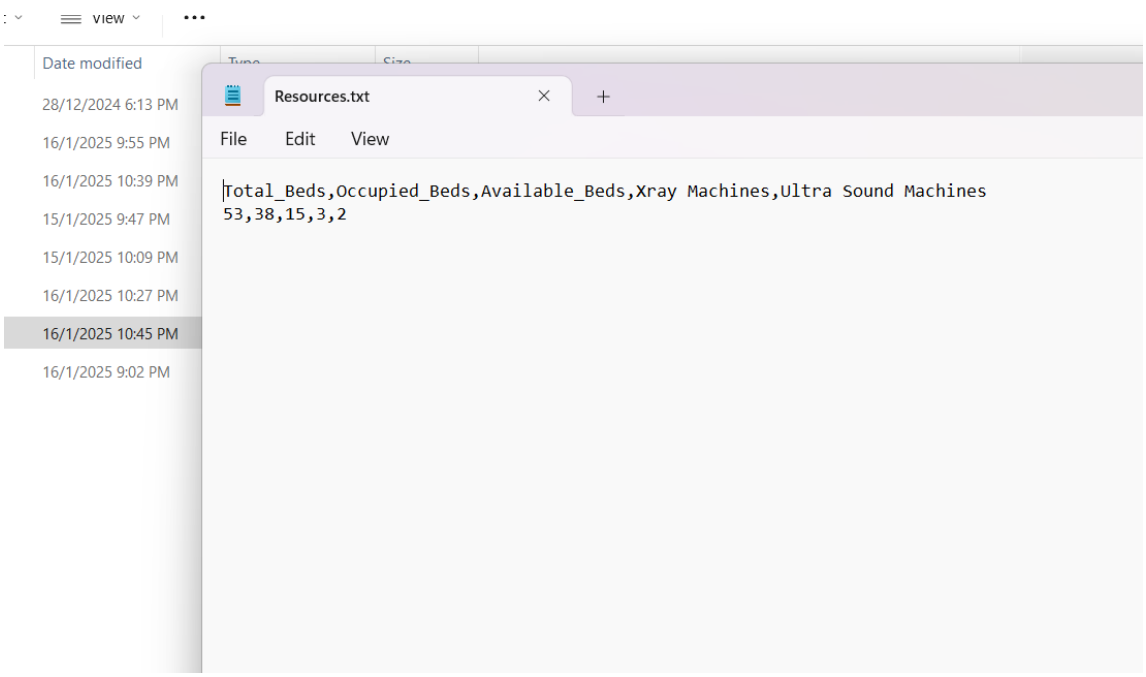


Explanation:

The **Remove_Bed()** function checks first if occupied beds are less than total beds and then only decrements the **Total_Beds** and **Available_Beds** count in **Resources.txt** file by 1. In the example above in Resources file, the **Total_Beds** = **54**, **Available_Beds = 16** initially, which changed when a bed was removed and new data in Resources became Total_Beds = **53** and Available_Beds = **15.**

Updating bed Status(Occupied):

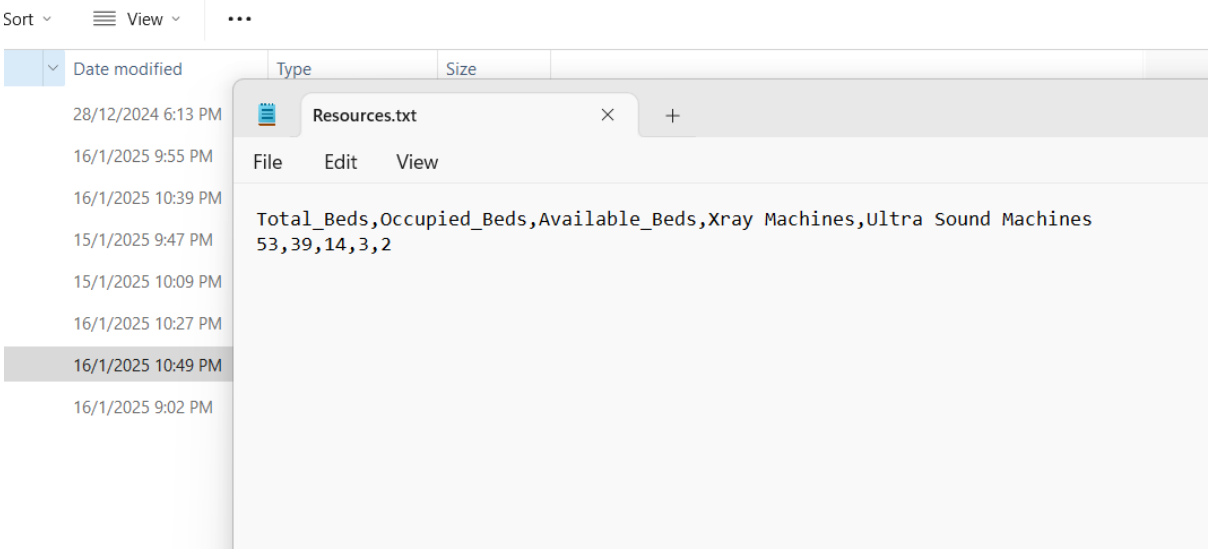
- File before updating bed status



Input:

```
Enter your Choice to proceed:
1. Manage User Accounts
2. View Hospital Statistics"
3. Generate Report
4. Manage Hospital Resources
5. Set and View Operational Policies
6. Search FUnction
7. Exit
Choose your option
1. Add Beds
2. Remove Beds
3. Update Bed Status
3
Please choose an option
1. Mark Bed as Occupied
2. Mark Bed as Available1
Bed has been updateed Successfully!
```

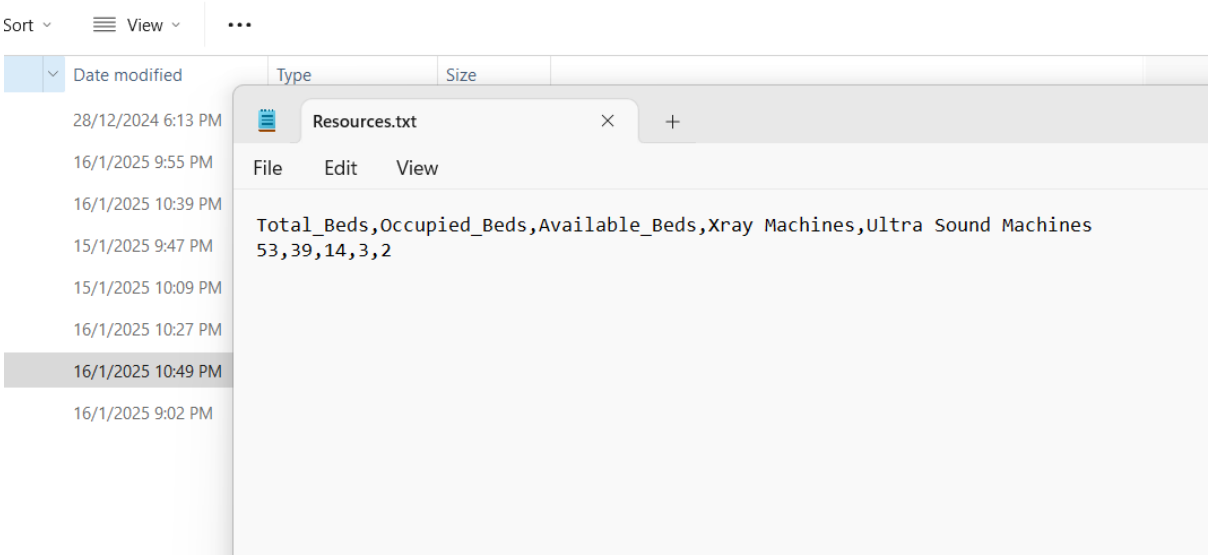
- **Output:**
  File after updating bed status to Occupied



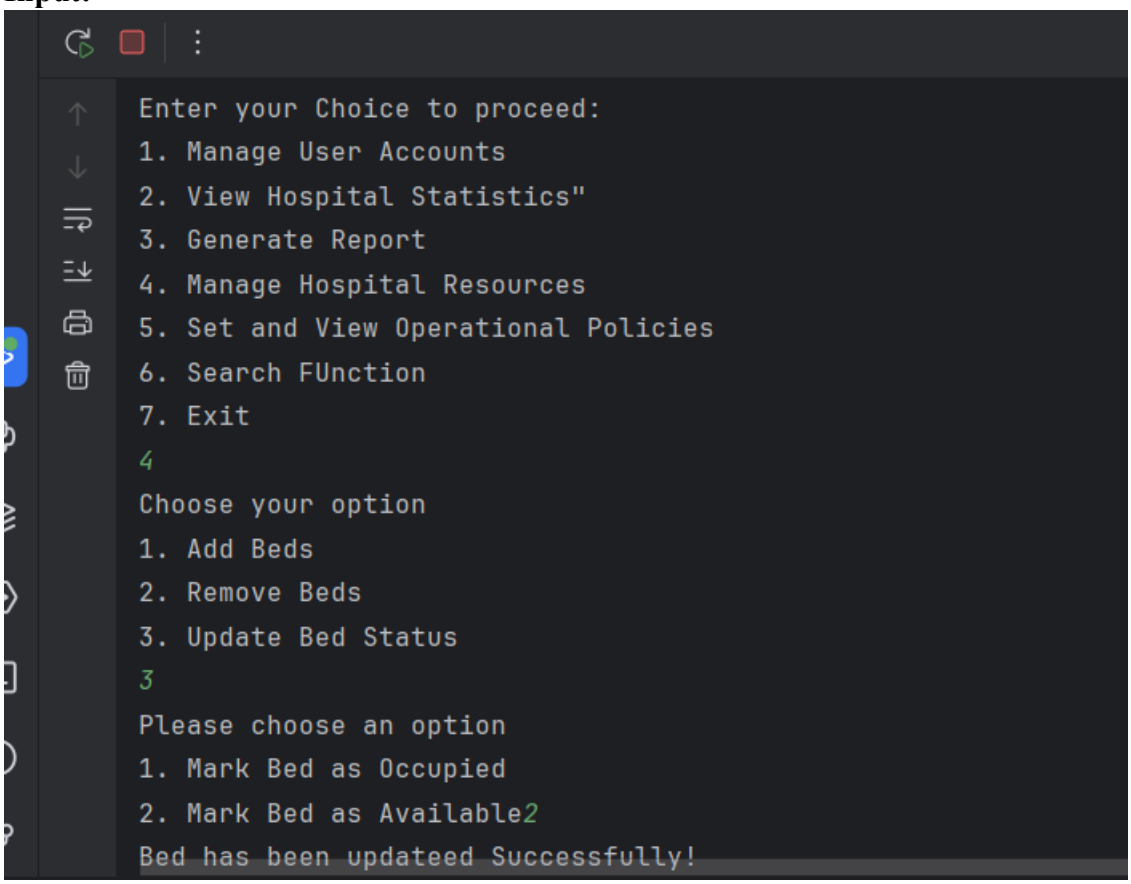
Explaination:
The **Update_Bed()** function asks first if user wants to update the bed status to occupied or availabel. If option choosed is **"Mark Bed as Occupied"**, then Occupied_Beds is incremented in **Resources.txt** file by 1. In the example above in **Resources.txt** file, the **Occupied_Beds = 38** initially, which changed when a bed was occupied the new data in **Resources.txt** became **Occupied_Beds = 39.**

Updating bed Status(Available):

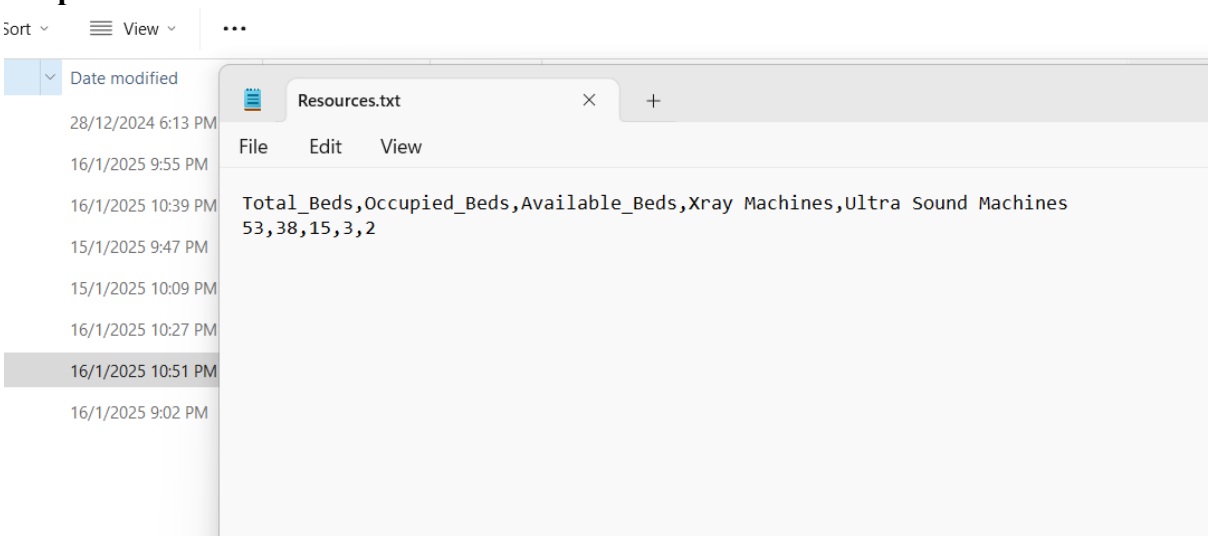
- File before updating bed status



Input:



Output:



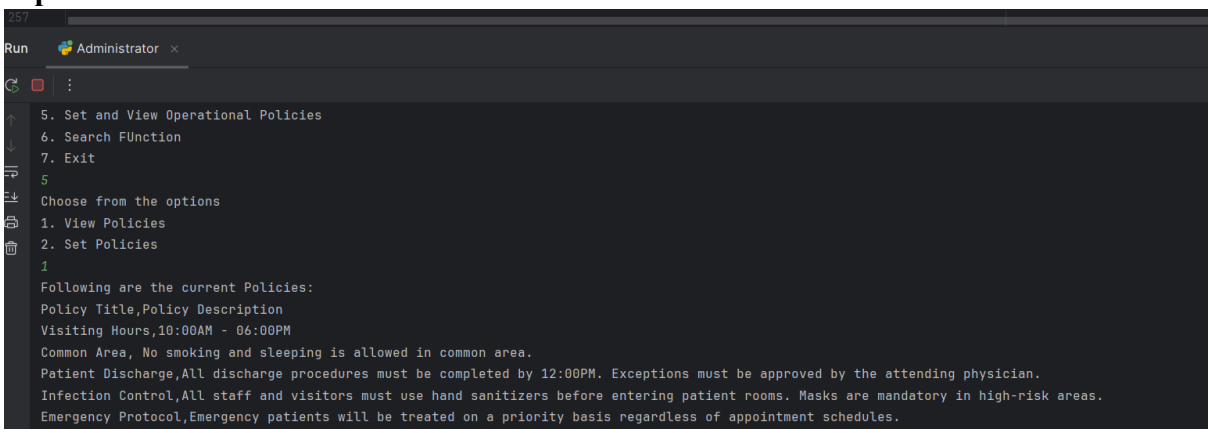
Explanation:

The **Update_Bed()** function asks first if user wants to update the bed status to occupied or available. If option choosed is **"Mark Bed as Available"**, then Occupied_Beds is decremented in **Resources.txt** file by 1. In the example above in **Resources.txt** file, the **Occupied_Beds = 39** initially, which changed when a bed was **(marked as available)** then new data in **Resources.txt** became **Occupied_Beds = 38.**

5. Set operational rules or policies for the hospital system.

- **View Policies:**

Input:

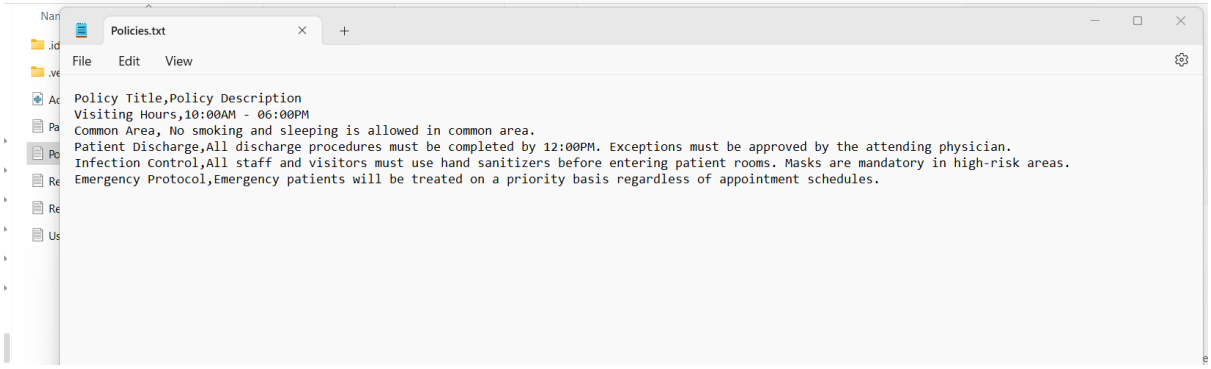


Explaination:

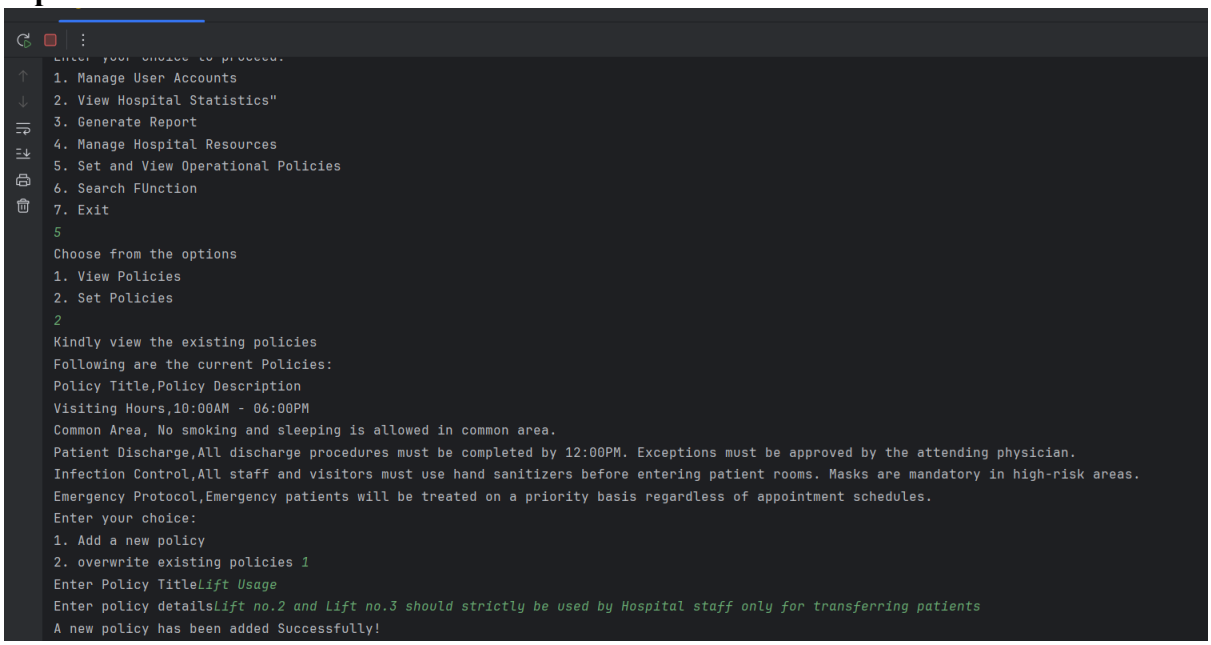
The **View_Policies()** function reads all the policies listed in **Policies.txt** file and displays it to user.

- **Set Policies:**

File before setting policies.

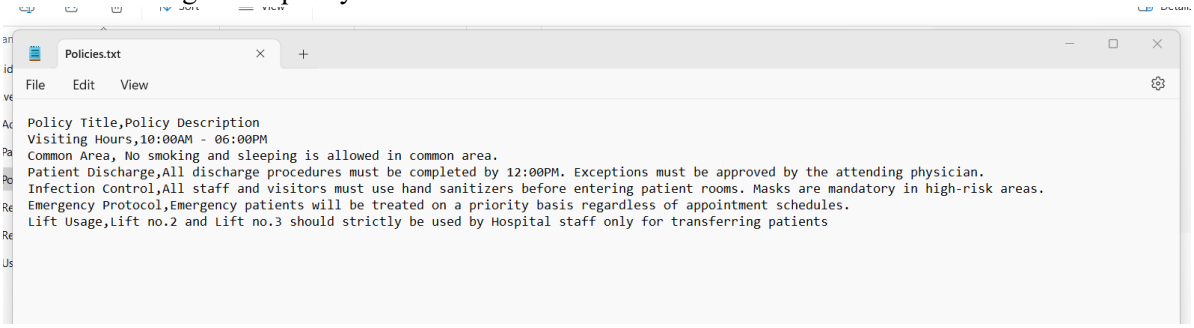


Input:



Output:

- File after adding a new policy

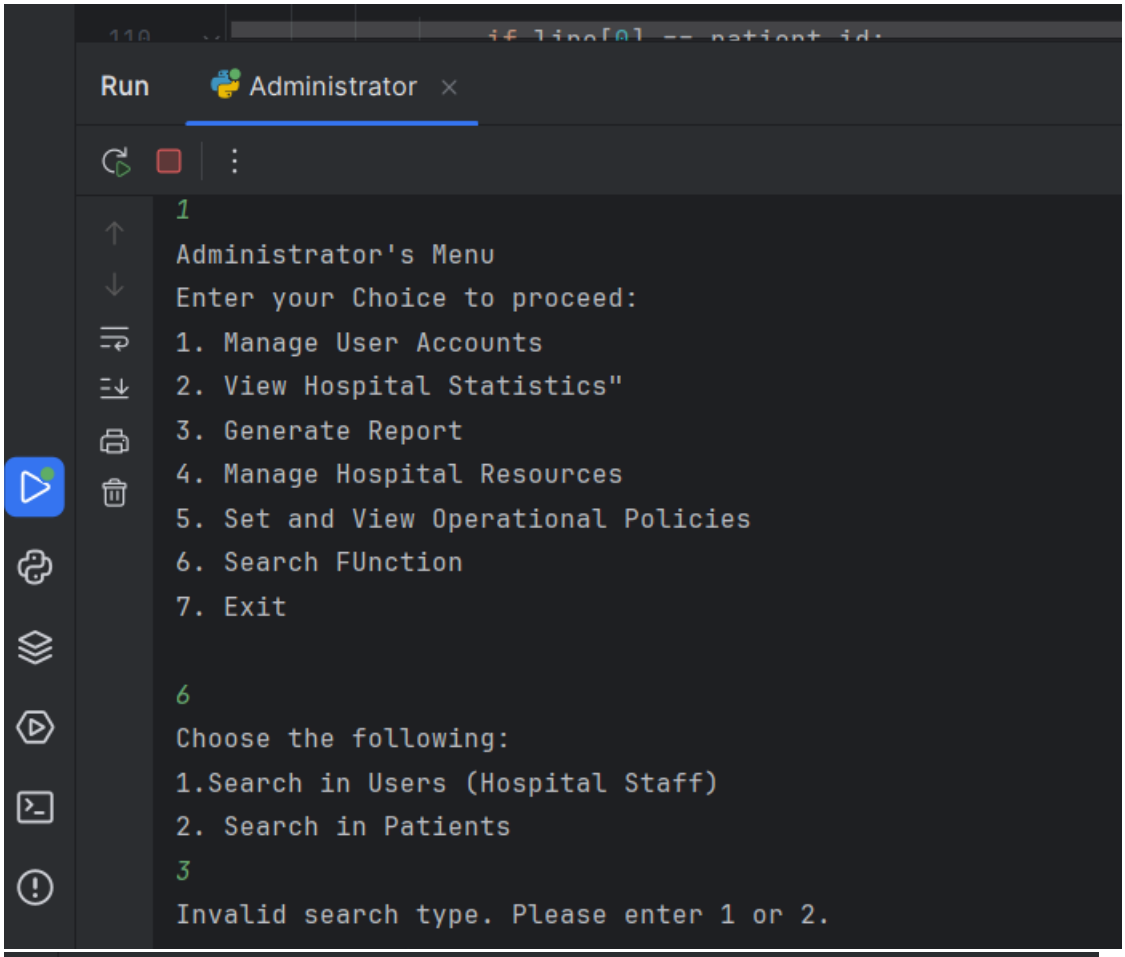


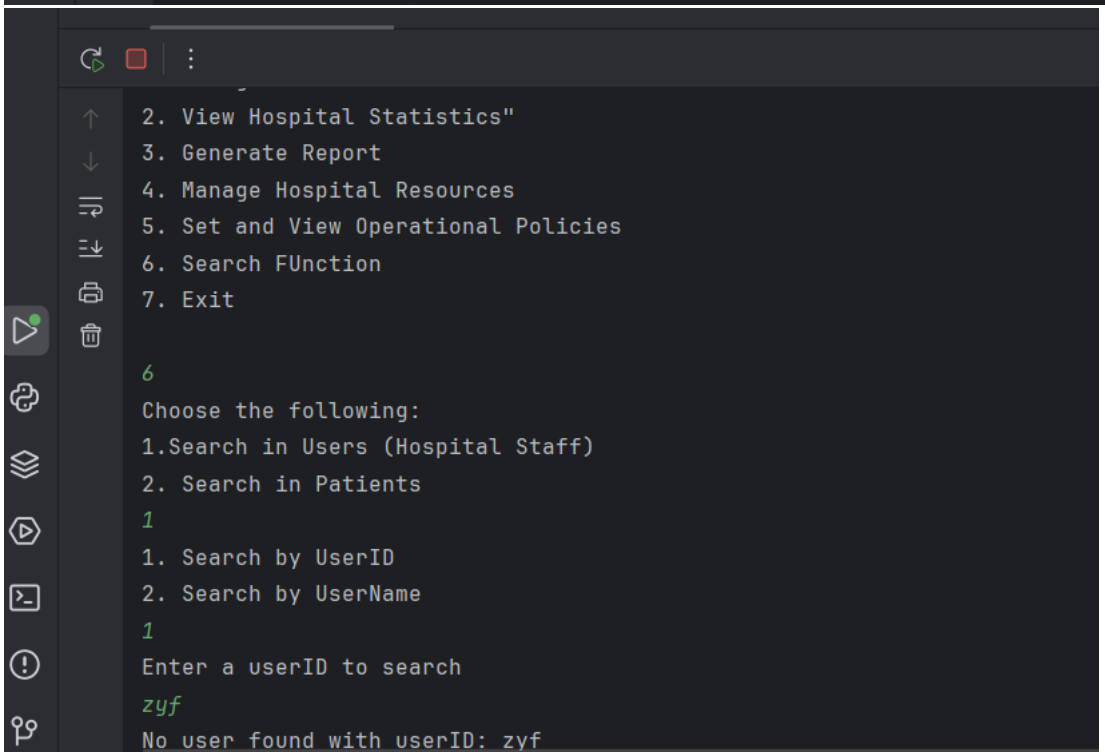
Explaination:

The View_Policies() function reads all the policies listed in **Policies.txt** file and displays it to user. The **Set_Policies()** function appends a new policy in **Policies.txt** file or overwrites it as required by the user. In the example above a new policy of **("Lift Usage, Lift no.2 and Lift no.3 should strictly be used by Hospital staff only for transferring patients.")** was stored in **Policies.txt** as input by user.

.....

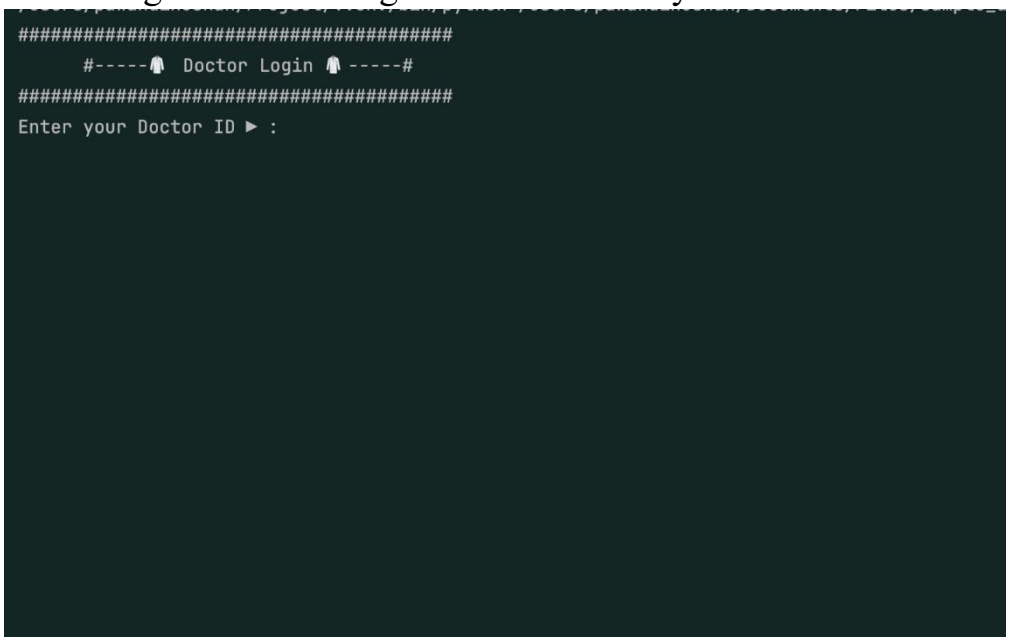
- **Errors and validations:**



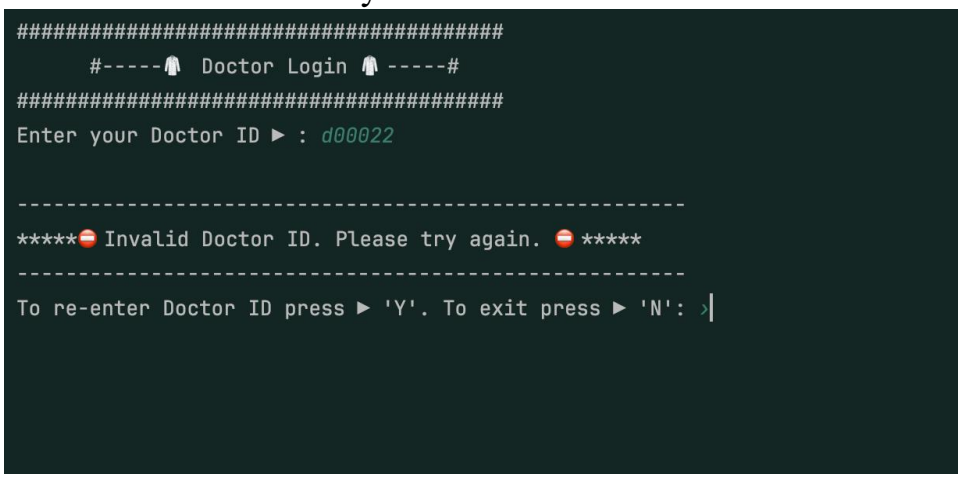


Input/Output of Doctor's Role:

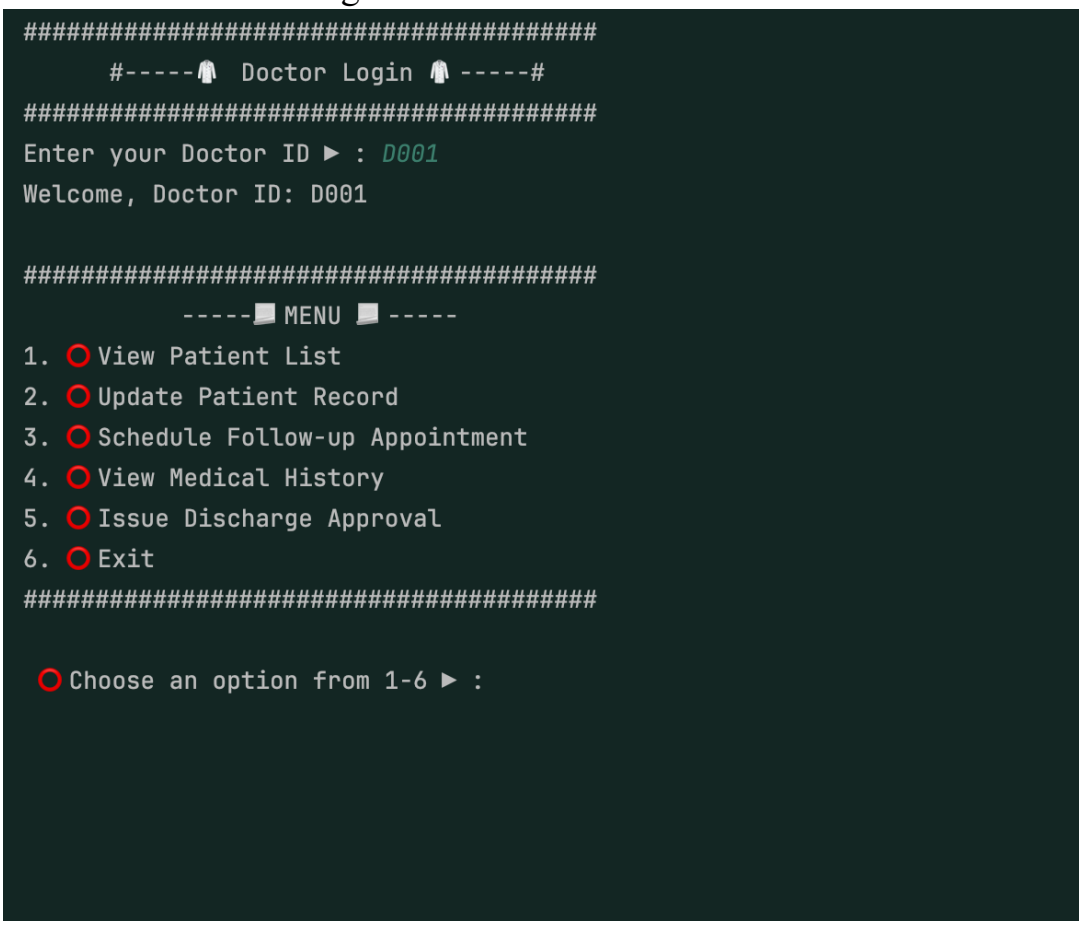
main mage after selecting Doctor from the system menu

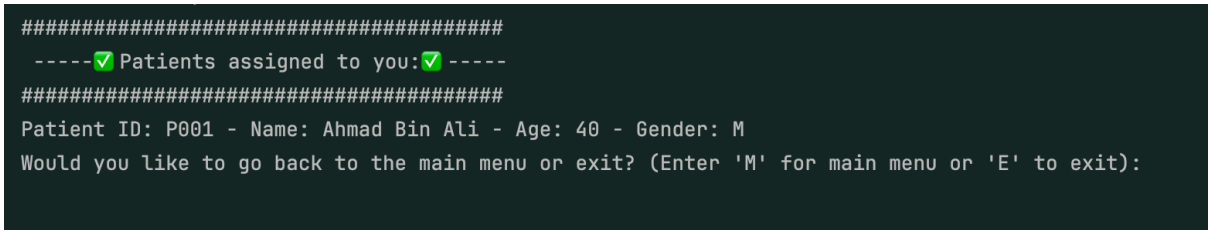


IF the input ID is invalid the above message will be printed and if input 'Y' the user will be taken to the login menu or if the user press 'N' the user will be exited from the doctor system



Doctor main menu after login in with the valid ID.





If the user input 1 the following data will be printed and if the user enters “M” user will be redirected to the main menu or if the user press “E” the user will be exited form the Doctor system

```
########################################
      #-----🥼 Doctor Login 🥼-----#
########################################
Enter your Doctor ID ▶ : D001
Welcome, Doctor ID: D001

########################################
           -----📋 MENU 📋-----
1. ⭕ View Patient List
2. ⭕ Update Patient Record
3. ⭕ Schedule Follow-up Appointment
4. ⭕ View Medical History
5. ⭕ Issue Discharge Approval
6. ⭕ Exit
########################################

 ⭕ Choose an option from 1-6 ▶ : 2
Enter Patient ID: P001
Enter new diagnosis, prescription, and treatment plan:
Diagnosis: Diabetes
Prescription: Maintain suger controll and take pills
Treatment Plan: Moniter suger level
```

If the user enters 2 option. The user will be asked to input the necessary details and it will be updated to the .txt files

```
PatientID, Date,Diagnosis,Prescription,TreatmentPlan,Followups
P001, 2-2-2025,High Blood Pressure Medication,Lifestyle changes including low-sodium diet and exercise,Monitor blood pressure regularly,After 1 month
```

The patient data before inputting the new data

```
PatientID, Date,Diagnosis,Prescription,TreatmentPlan,Followups
P001, 2-2-2025,Diabeties,suger controll and suger pils,moniter suger level,After 1 month
```

The patient data after inputting the new data

```
Enter Patient ID: p00e
Enter new details for the patient:
Diagnosis: test
Prescription: test
Treatment Plan: test
----------------------------------------
 -----⛔ Patient ID not found. ⛔-----
----------------------------------------
```

The following message will be printed If wrong patient id is entered

```
#######################################
      #-----👨‍⚕️ Doctor Login 👨‍⚕️-----#
#######################################
Enter your Doctor ID ▶ : D001
Welcome, Doctor ID: D001

#######################################
          -----📋 MENU 📋-----
1. ⭕ View Patient List
2. ⭕ Update Patient Record
3. ⭕ Schedule Follow-up Appointment
4. ⭕ View Medical History
5. ⭕ Issue Discharge Approval
6. ⭕ Exit
#######################################

 ⭕ Choose an option from 1-6 ▶ : 3
Enter Patient ID: P002
Appointment Date (YYYY-MM-DD): 2030-12-25
Appointment Time (HH:MM): 12:45
---------------------------------------
-----✅ Appointment scheduled successfully. ✅-----
---------------------------------------
Would you like to go back to the main menu or exit? (Enter 'M' for main menu or 'E' to exit):
```

If the user enters 3 option the user will be asked the following data to input

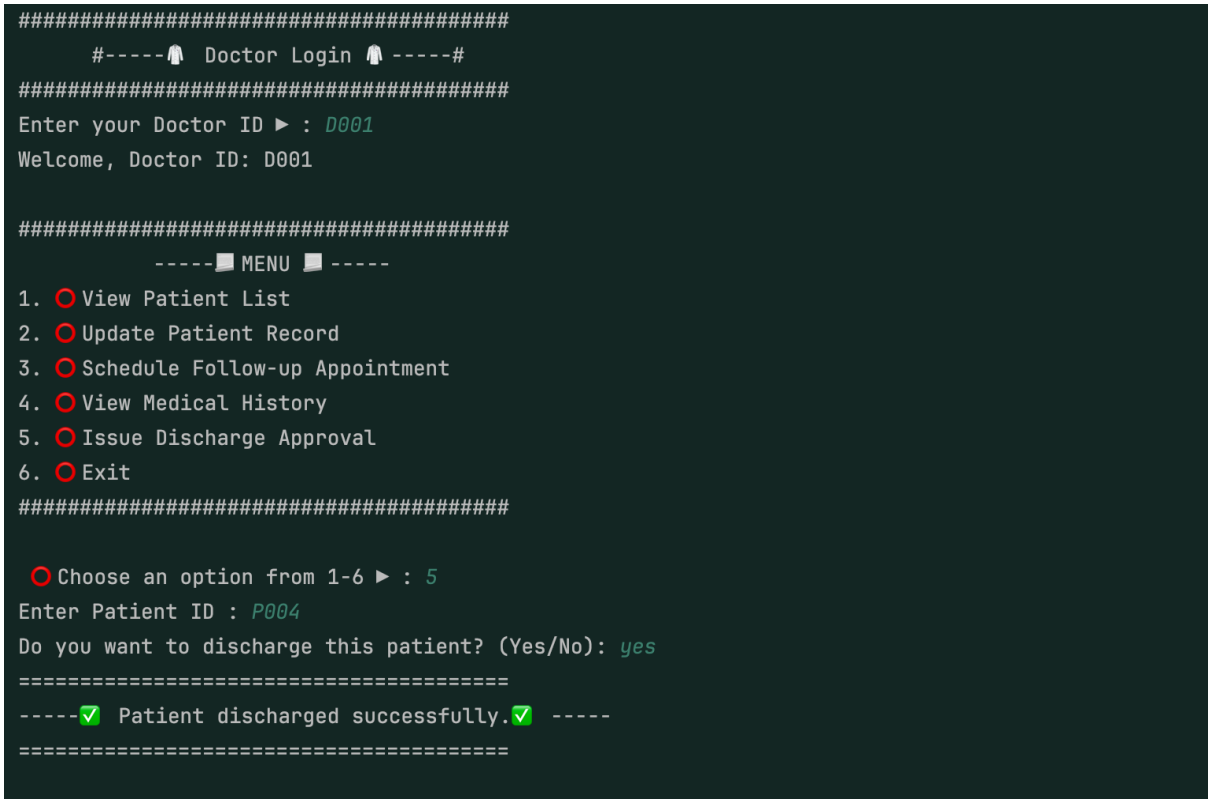
```
1  Patient ID,Doctor ID,Date,Time
2  P001,D002,2025-01-15,10:00
3  P002,D004,2025-01-16,14:30
4  P003,D005,2025-01-17,09:00
5  P004,D003,2025-01-18,11:00
6  P005,D001,2025-01-19,15:00
7
```

.txt file before scheduling Follow-up Appointment

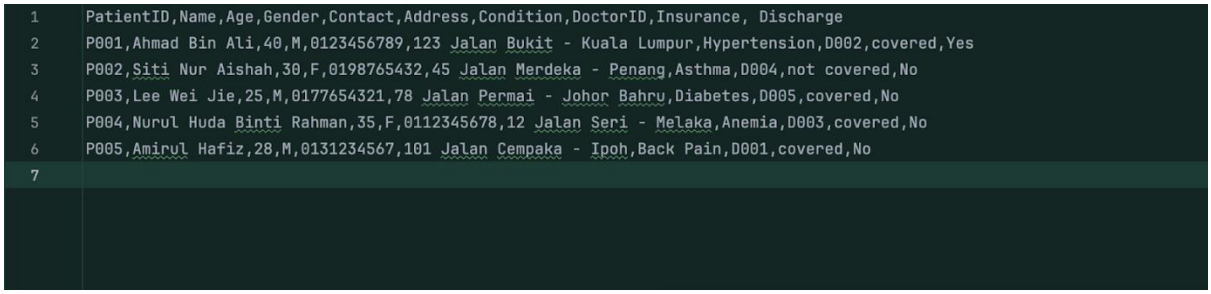
```
1  Patient ID,Doctor ID,Date,Time
2  P001,D002,2025-01-15,10:00
3  P002,D004,2025-01-16,14:30
4  P003,D005,2025-01-17,09:00
5  P004,D003,2025-01-18,11:00
6  P005,D001,2025-01-19,15:00
7  P002,D001,2035-12-25,12:50
8
```

.txt file after scheduling Follow-up Appointment

If the user enters 4 option the user will be asked the following data to input



If the user enters 5 option the user will be asked the following data to input



.txt file before approving discharge

```
PatientID,Name,Age,Gender,Contact,Address,Condition,DoctorID,Insurance, Discharge
P001,Ahmad Bin Ali,40,M,0123456789,123 Jalan Bukit - Kuala Lumpur,Hypertension,D002,covered,Yes
P002,Siti Nur Aishah,30,F,0198765432,45 Jalan Merdeka - Penang,Asthma,D004,not covered,No
P003,Lee Wei Jie,25,M,0177654321,78 Jalan Permai - Johor Bahru,Diabetes,D005,covered,No
P004,Nurul Huda Binti Rahman,35,F,0112345678,12 Jalan Seri - Melaka,Anemia,D003,covered,Yes
P005,Amirul Hafiz,28,M,0131234567,101 Jalan Cempaka - Ipoh,Back Pain,D001,covered,No
```

.txt file before approving discharge

```
########################################
      #-----👔 Doctor Login 👔 -----#
########################################
Enter your Doctor ID ▶ : D001
Welcome, Doctor ID: D001

########################################
          -----📋 MENU 📋 -----
1. ⭕ View Patient List
2. ⭕ Update Patient Record
3. ⭕ Schedule Follow-up Appointment
4. ⭕ View Medical History
5. ⭕ Issue Discharge Approval
6. ⭕ Exit
########################################

 ⭕ Choose an option from 1-6 ▶ : 6

----------------------------------------
         -----👋 Goodbye! 👋 -----
----------------------------------------


Process finished with exit code 0
```

IF the option is 6 user will be excited from the program

.....

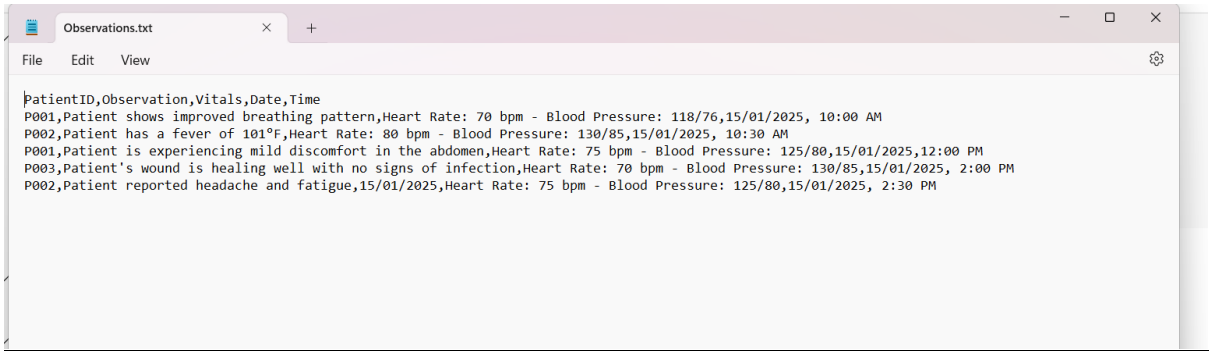
Input/Output & Explaination of Nurse's Role:

update_patient_vitals

Input

```python
def update_patient_observation(patient_id, new_observation, heartrate, blood_pressure):  1 usage
    """Update the patient's observation and vitals in the observation records."""
    patient_id = prompt_for_valid_patient_id()  # Ensure valid patient ID is entered
    try:
        with open('Observations.txt', 'r') as file:
            lines = file.readlines()

        with open('Observations.txt', 'w') as file:
            updated = False
            current_time = get_current_time()

            for line in lines:
                if line.startswith(patient_id):
                    line = f"{patient_id},{new_observation},Heartrate:{heartrate},BP:{blood_pressure},{current_time}\n"  # Update line with new data
                    updated = True
                file.write(line)

            if not updated:
                file.write(
                    f"{patient_id},{new_observation},Heartrate:{heartrate},BP:{blood_pressure},{current_time}\n")  # Add new record if patient not found
        print("Patient observation and vitals updated successfully.")
    except FileNotFoundError:
        print("Error: Observation records file not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Output



Explanation:

The update_patient_vitals function is responsible for updating a patient's vital signs and observations in the observations.txt file. It reads the existing patient records, modifies the relevant entry, and writes the updated information back to the file.

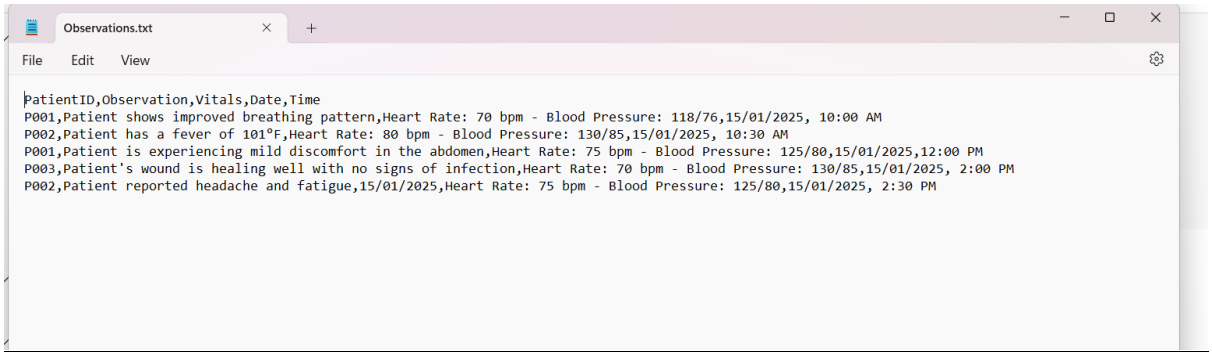
- **Function Purpose:** To update a patient's vital signs and observations in the observations.txt file.

Recording an Observation

Input:

```
def update_patient_observation(patient_id, new_observation, heartrate, blood_pressure):  1 usage
    """Update the patient's observation and vitals in the observation records."""
    patient_id = prompt_for_valid_patient_id()  # Ensure valid patient ID is entered
    try:
        with open('Observations.txt', 'r') as file:
            lines = file.readlines()

        with open('Observations.txt', 'w') as file:
            updated = False
            current_time = get_current_time()

            for line in lines:
                if line.startswith(patient_id):
                    line = f"{patient_id},{new_observation},Heartrate:{heartrate},BP:{blood_pressure},{current_time}\n"  # Update line with new data
                    updated = True
                file.write(line)

            if not updated:
                file.write(
                    f"{patient_id},{new_observation},Heartrate:{heartrate},BP:{blood_pressure},{current_time}\n")  # Add new record if patient not found
        print("Patient observation and vitals updated successfully.")
    except FileNotFoundError:
        print("Error: Observation records file not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Output:



Explanation:

The **Recording an Observation** functionality allows nurses to document specific observations about a patient's condition. This is crucial for maintaining accurate medical records and ensuring continuity of care.

Function Overview

The function responsible for recording observations typically appends a new entry to a file (e.g., observations.txt) that logs patient observations and patient vitals made by the nurse.

- **"Heart Rate: 72 bpm, Blood Pressure: 120/80"**

Logging Medication

Input:

```python
def log_medication(patient_id, medication_details):  1 usage
    """Log medication administered to a patient."""
    try:
        with open('medications.txt', 'a') as file:
            file.write(f"{patient_id},{medication_details}\n")  # Append medication log
        print("Medication logged successfully.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

Output:

PatientID, MedicationDetails, Date, Time
P001, Aspirin 100mg, 2025-01-15, 10:00 AM
P002, Amoxicillin 500mg, 2025-01-15, 11:30 AM

Explanation:

The logging medication functionality allows a nurse to record the details of medications administered to patients. This is crucial for maintaining accurate medical records and ensuring proper patient care.

Function: log_medication

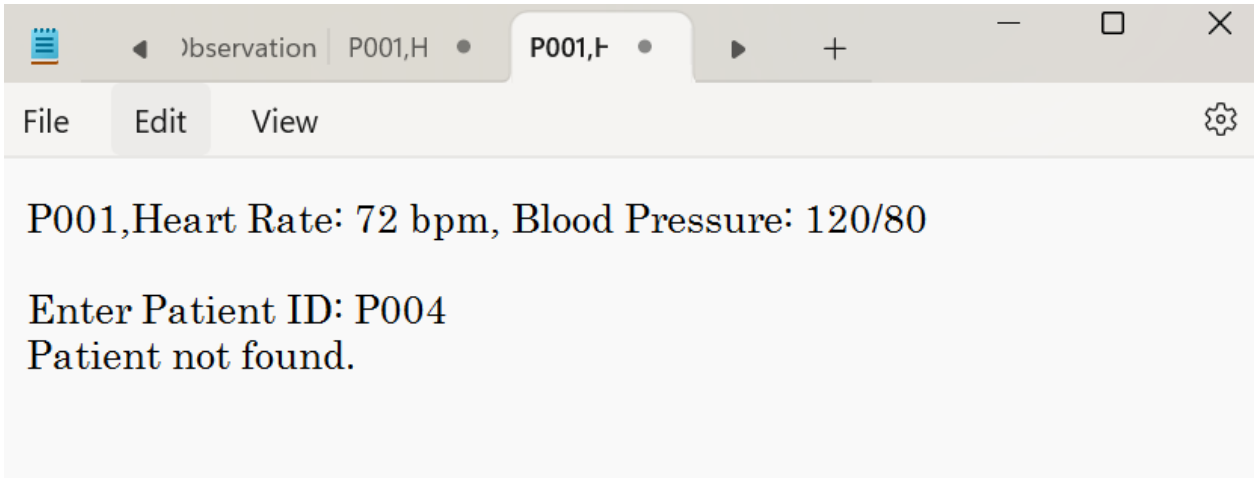
Here's a breakdown of how the logging medication process works, including inputs, outputs, and expected behavior.

Retrieving Patient Information

Input:

```python
def retrieve_patient_info(patient_id):  1 usage
    """Retrieve and display a patient's information and vitals."""
    try:
        with open('patients.txt', 'r') as file:
            for line in file:
                if line.startswith(patient_id):
                    return line.strip()  # Return patient info
        return "Patient not found."
    except FileNotFoundError:
        return "Error: Patient records file not found."
    except Exception as e:
        return f"An error occurred: {e}"
```

Output:



Explanation:

The **Retrieve Patient Information** functionality allows a nurse to access and display specific details about a patient based on their unique patient ID. This is essential for monitoring patient health and ensuring that the nurse has up-to-date information.

Function Overview

The relevant function in the code is retrieve_patient_info(patient_id)

- **Input**:
    - **Option Selected**: 4 (Retrieve Patient Info)
    - **Patient ID**: Entered by the nurse (e.g., P001 or P004).
- **Output**:
    - If the patient ID exists: Displays the patient's information in the format P001,Heart Rate: 72 bpm, Blood Pressure: 120/80.
    - If the patient ID does not exist: Displays the message Patient not found..

This functionality is crucial for nurses to quickly access patient data, ensuring they can provide timely and informed care.

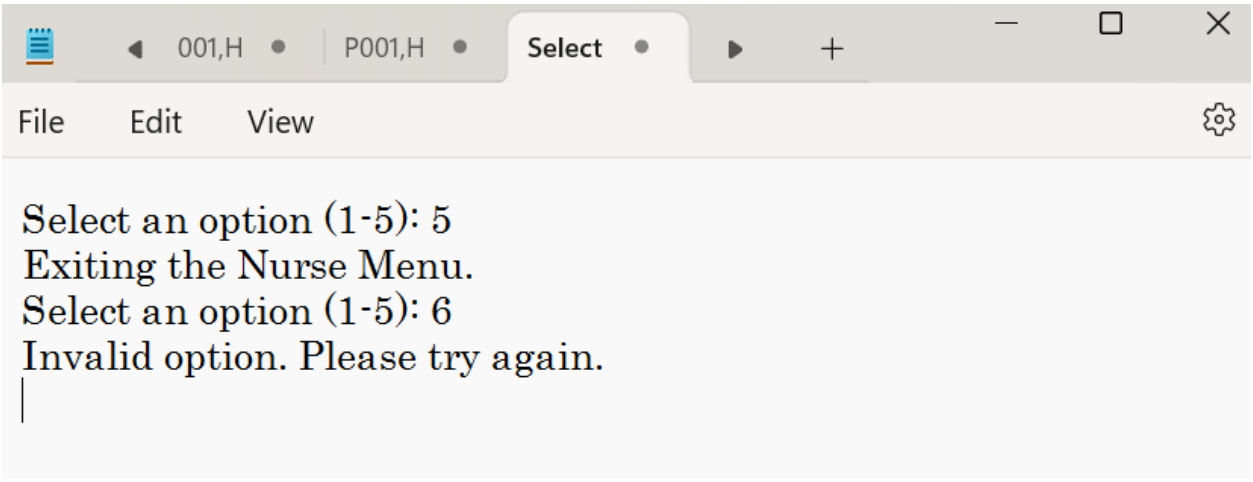
The main() Function

Input:

```
def main():  1 usage
    while True:
        print("\nNurse Menu:")
        print("1. Update Patient Vitals")
        print("2. Record Observation")
        print("3. Log Medication")
        print("4. Retrieve Patient Info")
        print("5. Exit")

        choice = input("Select an option (1-5): ")

        if choice == '1':
            patient_id = input("Enter Patient ID: ")
            new_vitals = input("Enter new vitals: ")
            update_patient_vitals(patient_id, new_vitals)
        elif choice == '2':
            patient_id = input("Enter Patient ID: ")
            observation = input("Enter observation: ")
            record_observation(patient_id, observation)
        elif choice == '3':
            patient_id = input("Enter Patient ID: ")
            medication_details = input("Enter medication details: ")
            log_medication(patient_id, medication_details)
        elif choice == '4':
            patient_id = input("Enter Patient ID: ")
            info = retrieve_patient_info(patient_id)
            print(info)
```

```
        choice = input("Select an option (1-5): ")

        if choice == '1':
            patient_id = input("Enter Patient ID: ")
            new_vitals = input("Enter new vitals: ")
            update_patient_vitals(patient_id, new_vitals)
        elif choice == '2':
            patient_id = input("Enter Patient ID: ")
            observation = input("Enter observation: ")
            record_observation(patient_id, observation)
        elif choice == '3':
            patient_id = input("Enter Patient ID: ")
            medication_details = input("Enter medication details: ")
            log_medication(patient_id, medication_details)
        elif choice == '4':
            patient_id = input("Enter Patient ID: ")
            info = retrieve_patient_info(patient_id)
            print(info)
        elif choice == '5':
            print("Exiting the Nurse Menu.")
            break
        else:
            print("Invalid option. Please try again.")

if __name__ == "__main__":
    main()
```

Output:



Explanation:

- **Function Purpose**: The main() function provides a user-friendly interface for nurses to manage patient information effectively.

- **Inputs**: User selections for various operations (1-5) and specific patient data when needed.
- **Outputs**: Confirmation messages for successful operations, patient information retrieval, or error messages for invalid inputs.
- **Flow**: The function loops until the nurse decides to exit, ensuring that multiple operations can be performed in one session.

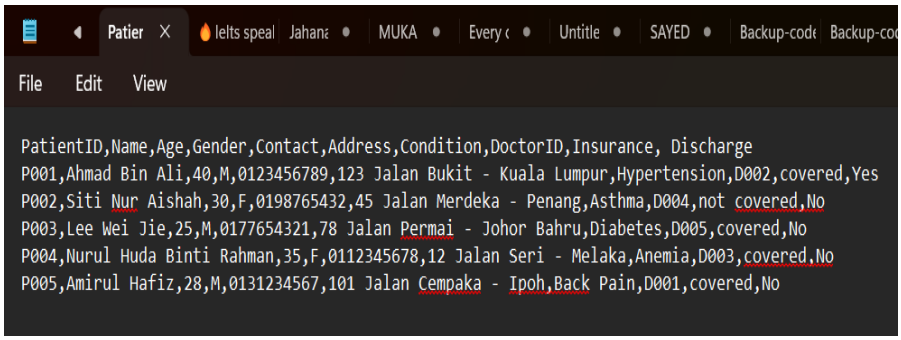
.....

Input/Output & Explaination of Receptionist's Role:

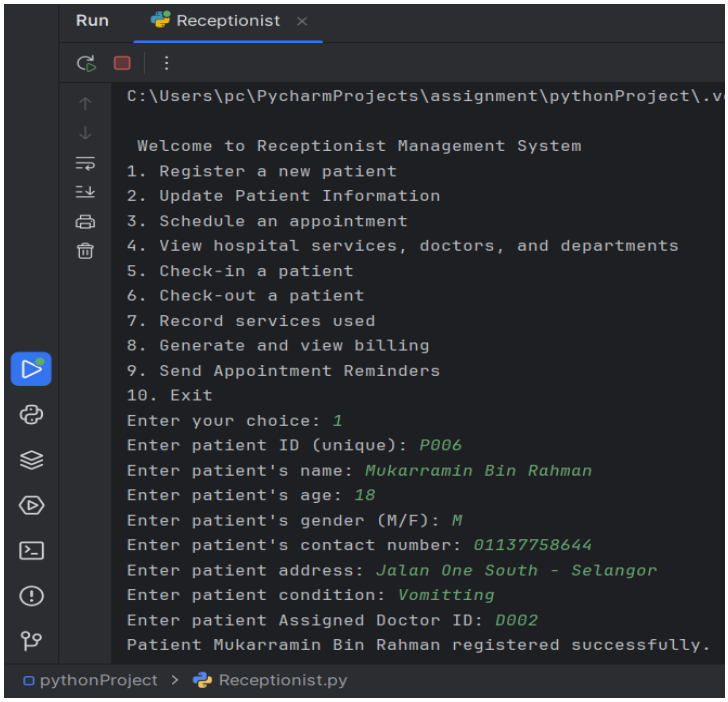
1. **Register new patients and update existing patient information.**

Registering a new patient:

- File Before Registering New Patient

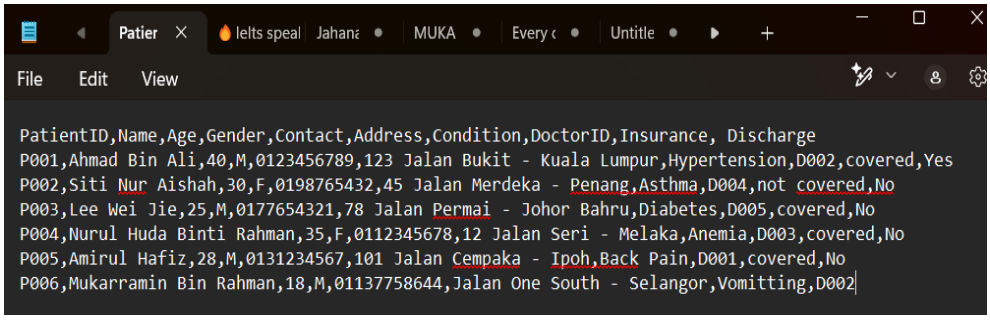


INPUT:



OUTPUT:

- File after Registering a new patient

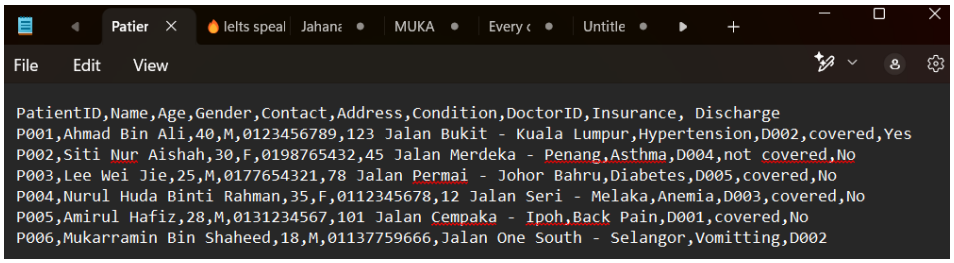


EXPLANATION:

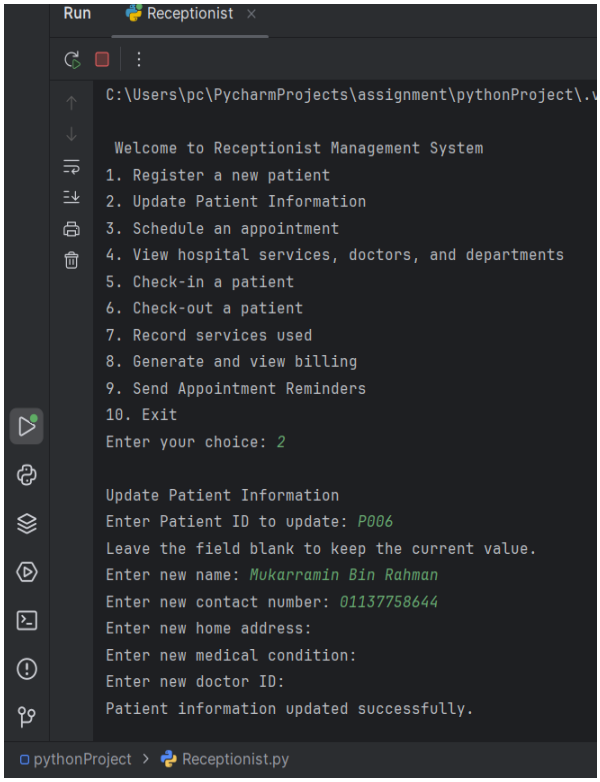
Users can register new patients in the system using the register_patient function. It does so by collecting important information about the patient such as the PatientID, name, age, gender, contact details, address, medical condition, and Doctor ID assigned to the patient. It also validates numeric inputs such as age and contact number and appends the details to the **Patients. txt** file. This feature provides a great way to manage a growing database of patients effectively. It provides you opportunity to collect all required information in an organized and safe manner.

Updating an existing patient

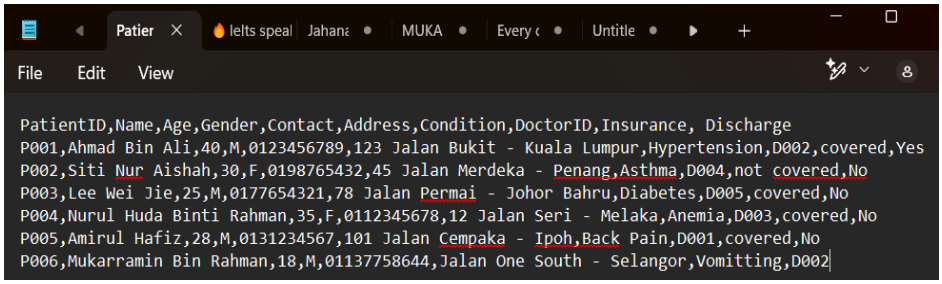
- File before updating an existing patient



INPUT:



OUTPUT:

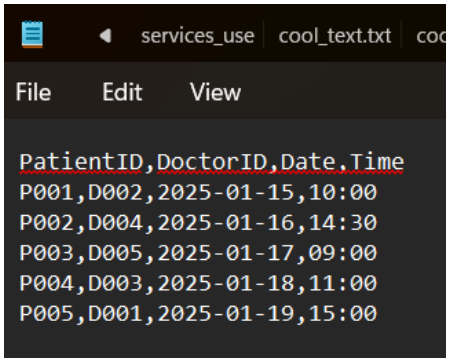


EXPLANATION:

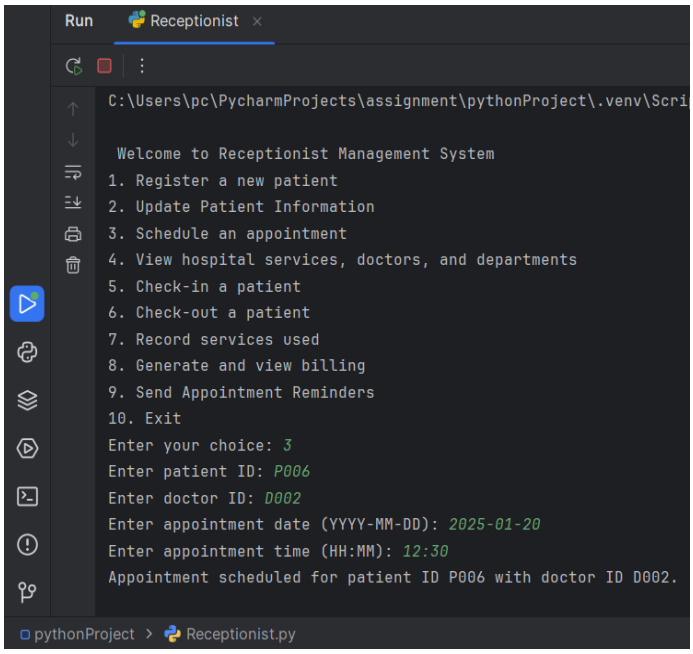
The update_patient_info function allows the users to make changes to the information of an existing patient. The user provides the patient ID, and the function gets the relevant record from the **Patients. txt** file. This means users can change individual fields — like name, contact , address, condition and doctor ID without overwriting the existing values for any fields left empty. This updates the record in place, which means that the latest information that needs to be implemented is available back to the file. This feature contributes to the flexibility of the system, as well as the fact that the patients' records stay accurate and up to date.

2. Schedule and manage appointments for doctors.

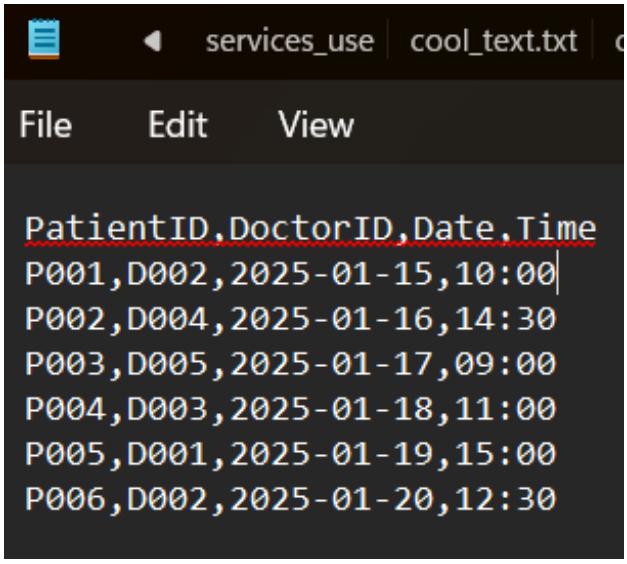
- File before Scheduling an appointment



INPUT:



OUTPUT:

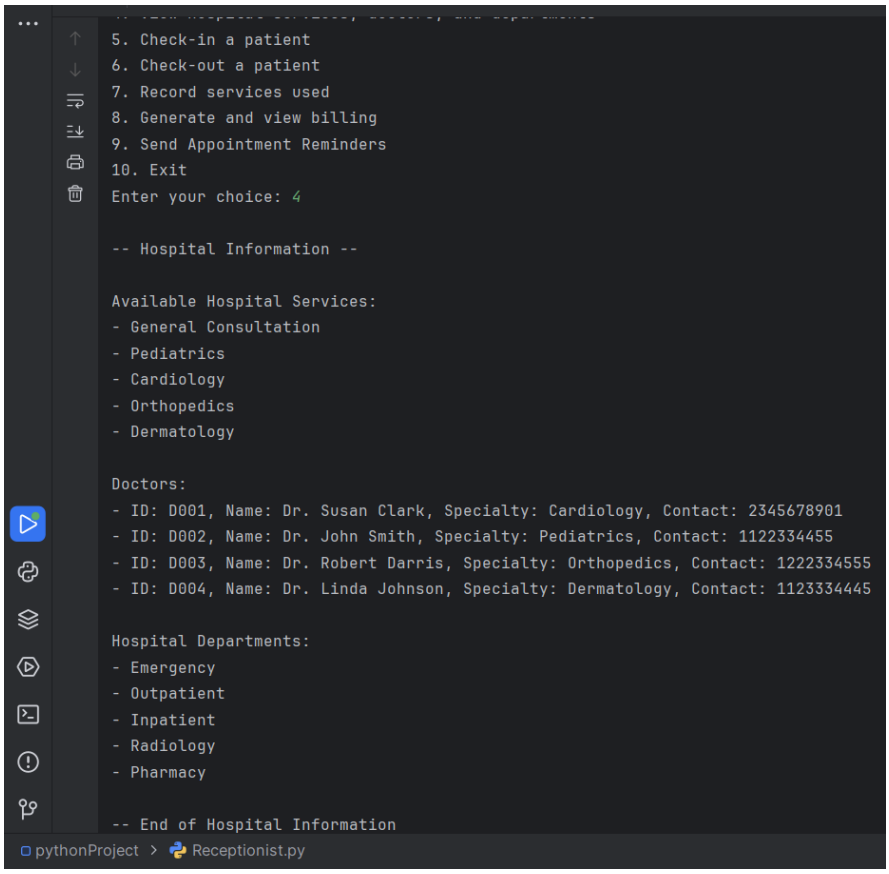


EXPLANATION:

It allows patients to make appointments with doctors. The users need to input the patient id, doctor id, appointment date, and time as text values, and there are validations in place in order to ensure that the date and time field are in the correct format. The appointment is then saved to the **Appointments. txt** file. This function is vital for effective schedule keeping and contributes to the seamlessness of patient-doctor interactions. It also sets the stage for sending appointment reminders in a later phase.

3. **Provide patients with information on hospital services, doctors, and departments.**

INPUT AND OUTPUT:



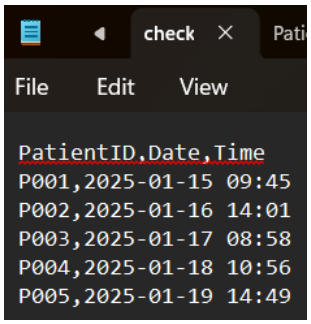
EXPLANATION:

The view_hospital_info function displays information about the hospital's services, doctors, and departments. It provides structured data, such as a list of services, including cardiology, dermatology etc and doctor profiles with specialties and contact information. The function also lists the different departments in the hospital; emergency, radiology etc. This feature allows users to receive all key information about the hospital's offerings.

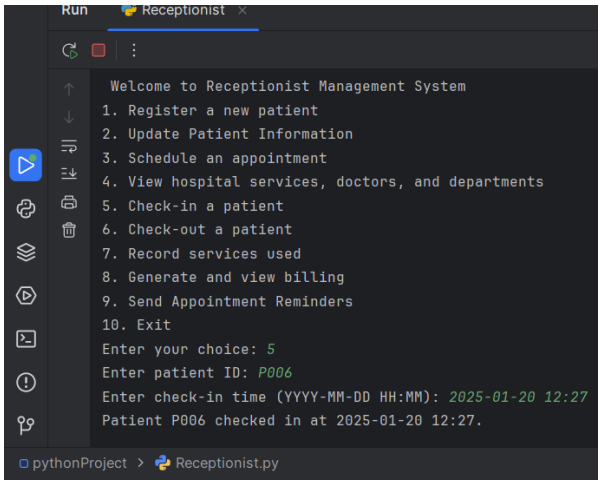
4. **Handle patient check-in and check-out processes.**

Check-In:

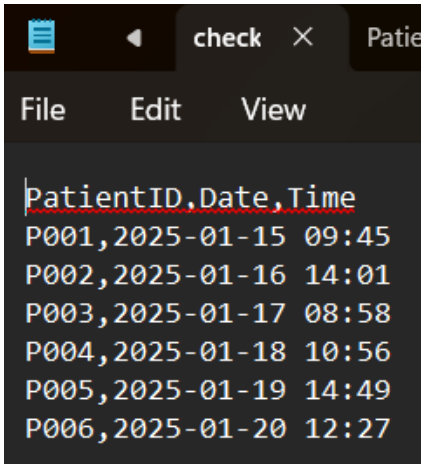
- File before checking-in the patient



INPUT:



OUTPUT:

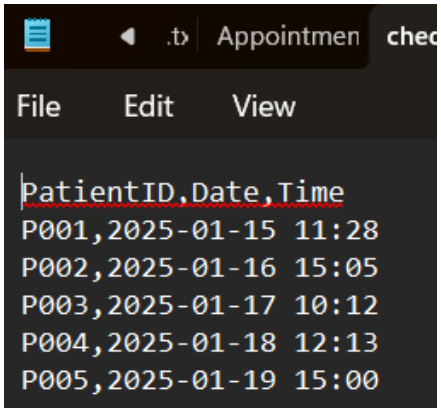


EXPLANATION:

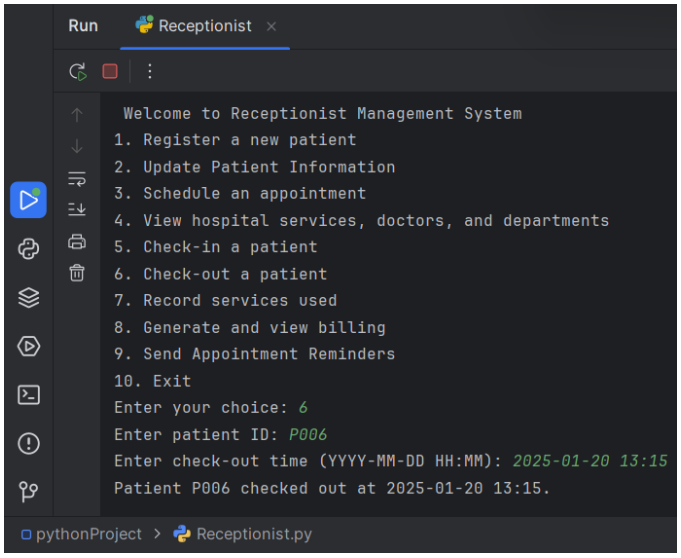
The check_in_patient function stores the timestamp for the patient's arrival at the hospital. Then, the program will ask the user to input the patient ID as well as the check-in time which will be stored in the **check_in.txt** file. One function uses this information to track which patients are in attendance and allowing for services, It is the first part of a patient journey through the hospital services.

Check-Out:

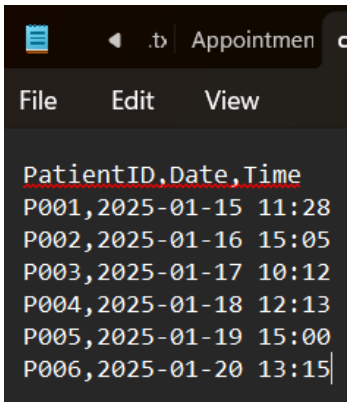
File Before Checking-Out



INPUT:



OUTPUT:



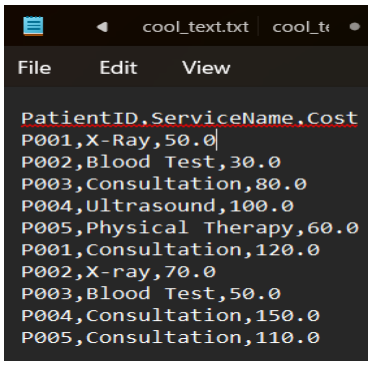
EXPLANATION:

It gets the check-out time for a patient, much like the previous check-in method. The time of checkout is stored in the check_out when the user provides the patient ID and the time of **check_out. txt** file. This is important for monitoring the length of a patient's stay and can be helpful in determining billing or peak operational hours. It is used in collaboration with the check-in feature to maintain a full attendance record.

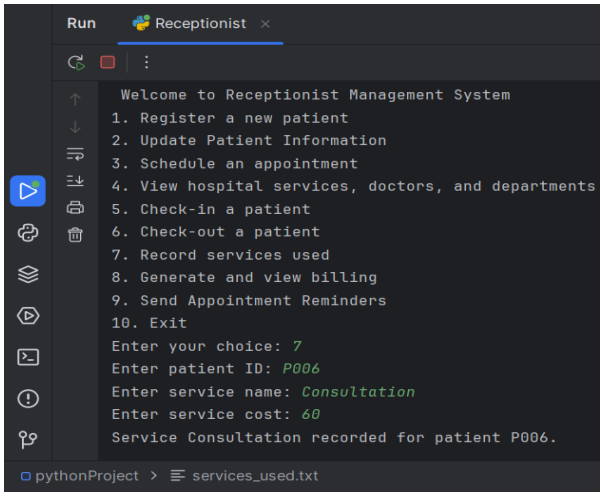
5. Generate billing details based on services used by the patient.

- **Services used:**

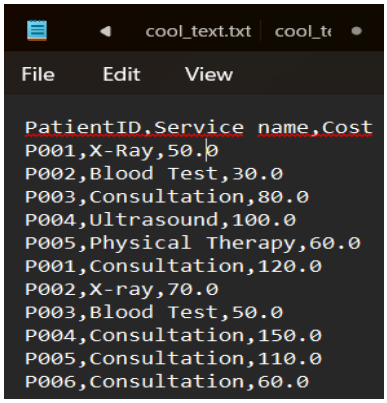
File before services used entry:



INPUT:



OUTPUT:



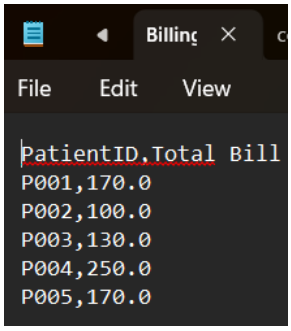
EXPLANATION:

The record_services function can be used to log services provided to a patient when they visit. It takes the patient ID, the name of the service, and the price of that service but only if the price is a valid number.This data is stored **services_used . txt** file for all performed services to keep the

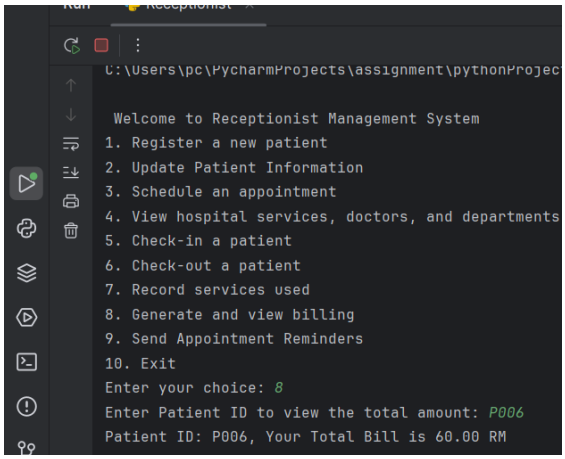
system history. This feature is critical for generating accurate billing and tracking of service utilization.

- **Generate Billing:**

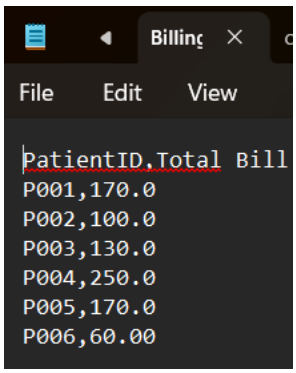
File before Generating Bill:



INPUT:



OUTPUT:



EXPLANATION:

It is the function that calculates and prints total bill for a patient according to the services entered into the system. It reads the **services_used. txt** file, group by patient ID, and print the total bill in a user-friendly way. If a record is found of the patient, then the bill also gets saved in the **Billing. txt file** to

use for reference later. This function is important in keeping an open billing and also adds a clear view to their expenses.

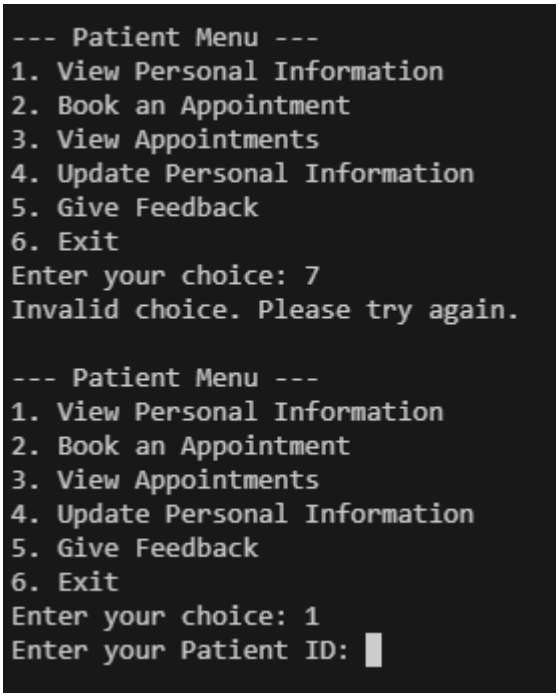
Input/Output & Explaination of Patient's Role:

This program is designed to allow a patient to manage their interaction with a hospital system. It includes functionalities like viewing personal information, booking appointments, updating personal details, and providing feedback.

It uses the following txt files and the following menus

Txt files	Menu
• Users • Patients • Feedback • Appointments	• View Personal Information • Book an Appointment • View Appointments • Update Personal Information • Give Feedback

The Patient Menu

Input and Output


When the program is runned it prompts the user with the following options on what they would like to proceed with, Depending on the choice of the user
It will either redirect the user to the selected page or promt the user to insert the correct number

Function: View Personal Information

File Interaction:

patients.txt

Input and Output

```
--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: 1
Enter your Patient ID: P004

--- Personal Information ---
P004,Nurul Huda Binti Rahman,35,F,0112345678,12 Jalan Seri - Melaka,Anemia,D003,covered, No

--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: P0000
Invalid choice. Please try again
```

The “View Personal Information” Allows the patient to view their stored personal information.

The file stores patient records in the format:

PatientID, Name, Age, Gender, Contact, Address, Condition, DoctorID, Insurance, Discharge

Process:

The program begins by checking the existence and content of the patients.txt file, making sure that it is not empty.
If the file is empty or missing, it displays the message: "No patient records found."

If the file contains data, the program proceeds to prompt the patient to input their specidic Patient ID. It then searches the file for a matching Patient ID.

If a match is found, the program retrieves and displays the corresponding line from the file as the patient's personal information.
If no match is found, it informs the user with the message: "Invalid Choice." This ensures that only valid records are accessed while providing appropriate feedback to the user.

Function: book_appointment()

File Interaction:
appointments.txt (to store booked appointments)

Users.txt (to retrieve the list of available doctors)

Input and Output with error handling

```
--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: 2
Enter your Patient ID: P004
Enter the doctor's name: Will
Invalid doctor name. Available doctors:
- Susan Clark
- John Smith
- Robert Darris
- Linda Johnson
Enter the doctor's name: John Smith
Enter the appointment date (YYYY-MM-DD): 2003-02-04
Enter the appointment time (HH:MM): 17:09
The appointment date and time must be in the future.
Enter the appointment date (YYYY-MM-DD): 2025-01-19
Enter the appointment time (HH:MM): 17:08
Appointment booked successfully.
```

Purpose:
Allows a patient to book an appointment with a doctor.

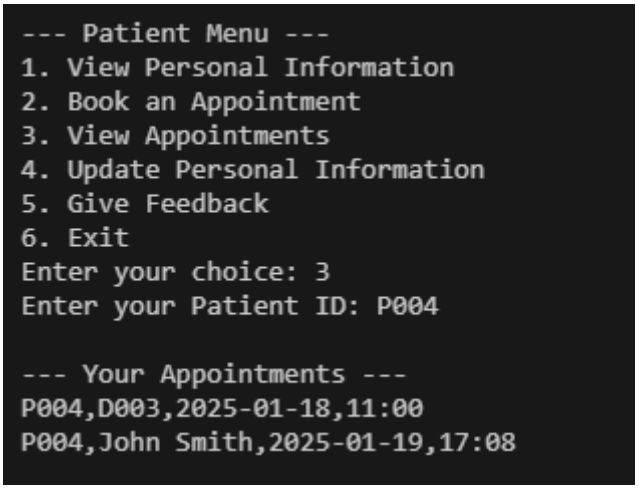
Process:

The program begins by reading a file named “Users.txt”, which contains a list of doctor and admins but only allows the patient to interact with the booking of the doctor entries stored in the format: DoctorID,Name,Role,Department.
It filters this list to include only those entries where the role is specified as "doctor".

After generating the filtered list of doctors, the program prompts the patient for the following inputs: their unique Patient ID, the selection of a doctor's name from the available list, and their desired appointment date and time. To ensure the input is valid, the program performs several checks. It validates the date format as YYYY-MM-DD using a function validate_date() and the time format as HH:MM using validate_time(). Additionally, it confirms that the appointment is scheduled for a future date and time using a function is_future_appointment(). Once all validations are successfully passed, the program writes the confirmed appointment details in the appointments.txt file in the format PatientID,DoctorName,Date,Time. Finally, it displays a confirmation message to the patient: "Appointment booked successfully."

Function: view_appointments()

File Interaction:
appointments.txt

Input and Output


Stores appointments in the format:

PatientID,DoctorName,Date,Time

Purpose:
Displays all appointments for a specific patient.

Process:

The program begins by verifying whether the `appointments.txt` file exists and contains data. If the file is empty or does not exist, it displays the message: "No appointments found." If data is present, the program prompts the patient to input their unique Patient ID. It then searches the file for any lines that begin with the provided Patient ID. If matches are found, the program displays all corresponding lines as a list of the patient's appointments. If no matches are found, it informs the user with the message: "No appointments found for the given Patient ID." This process ensures patients can access their appointment details while providing clear feedback when no relevant records exist.

Function: update_personal_information()

File Interaction:

patients.txt

Input and Output

```
--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: 4
Enter your Patient ID: P004
Current Information:
Name: Nurul Huda Binti Rahman
Contact: 0112345678
Address: 12 Jalan Seri - Melaka
Name (current: Nurul Huda Binti Rahman): Keep same? (y/n): y
Contact (current: 0112345678): Keep same? (y/n): n
Enter new Contact: 0179082264
Address (current: 12 Jalan Seri - Melaka): Keep same? (y/n): y
Information updated successfully.
```

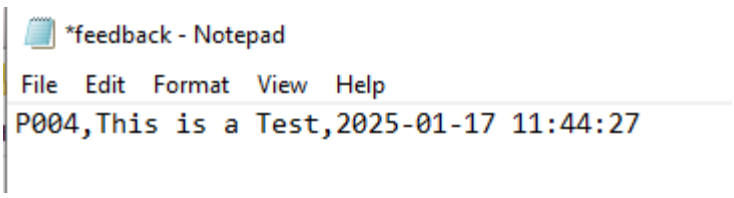
```
--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: 1
Enter your Patient ID: P004

--- Personal Information ---
P004,Nurul Huda Binti Rahman,35,F,0179082264,12 Jalan Seri - Melaka,Anemia,D003,covered, No
```

Process:
The program starts by checking if the patients.txt file exists and contains data. If the file is missing or empty, it displays the message: "No patient records found." If data is available, the program prompts the patient to enter their unique Patient ID. It searches the file for a matching record associated with the given ID. If a match is found, the program displays the current patient information and allows the user to update each field one by one, offering the option to retain the current value for any field. If no matching record is found, it informs the user with the message: "Patient ID not found." After updating the record, the program rewrites it into the file, ensuring that all other records remain unchanged. Finally, it displays a confirmation message: "Information updated successfully." This ensures efficient and accurate updates to patient records while maintaining data integrity.

Function: give_feedback()
File Interaction: feedback.txt

Input and Output

```
--- Patient Menu ---
1. View Personal Information
2. Book an Appointment
3. View Appointments
4. Update Personal Information
5. Give Feedback
6. Exit
Enter your choice: 5
Enter your Patient ID: P004
Enter your feedback: This is a test
Feedback submitted successfully.
```



Purpose:
Allows the patient to submit feedback about the hospital services.

Stores feedback in the format:

PatientID,Feedback,DateTime

Process:

The program begins by prompting the patient to input their unique Patient ID and provide their feedback. It then captures the current date and time using the `datetime.now()` function to record when the feedback was submitted. The program formats the feedback entry, including the Patient ID, feedback, and timestamp, and writes this information into the `feedback.txt` file. Finally, it displays a success message: "Feedback submitted successfully." This ensures that patient feedback is recorded accurately with a timestamp for reference.

“Conclusion”

The Hospital Management System (HMS) successfully addresses core hospital operations by providing role-based functionalities for administrators, doctors, nurses, receptionists, and patients. Developed using Python with a modular design and text-based data storage, the system improves efficiency and ensures role-specific access to relevant functionalities.

Advantages

- Efficiency: Streamlines workflows and reduces administrative burdens with role-specific tasks.
- User-Friendly: The menu-driven interface simplifies navigation for users.
- Data Integrity: Consistent file handling and validations ensure accurate record-keeping.
- Scalability: The modular approach allows easy additions of new features or roles.
- Security: Role-based access limits functionalities to authorized users.

Limitations:

- The system uses text files, limiting real-time data access and multi-user functionality.
- The command-line interface may not be intuitive for all users.
- Error reporting and validation could be enhanced for complex scenarios.

Suggestions for Improvement

- Database Integration: Replacing text files with a database (e.g., SQLite) for better performance.
- Graphical User Interface: Adding a GUI to make the system more user-friendly.
- Real-Time Access: Supporting multi-user, concurrent access for larger hospitals.
- Enhanced Reporting: Adding advanced reports and visualizations for better insights.

Overall , the HMS provides a functional and efficient solution for hospital management, addressing key operational challenges. While it meets the current requirements, further improvements in interface design, data handling, and scalability can enhance its usability and impact, making it a robust tool for modern healthcare institutions.

"Appendix"

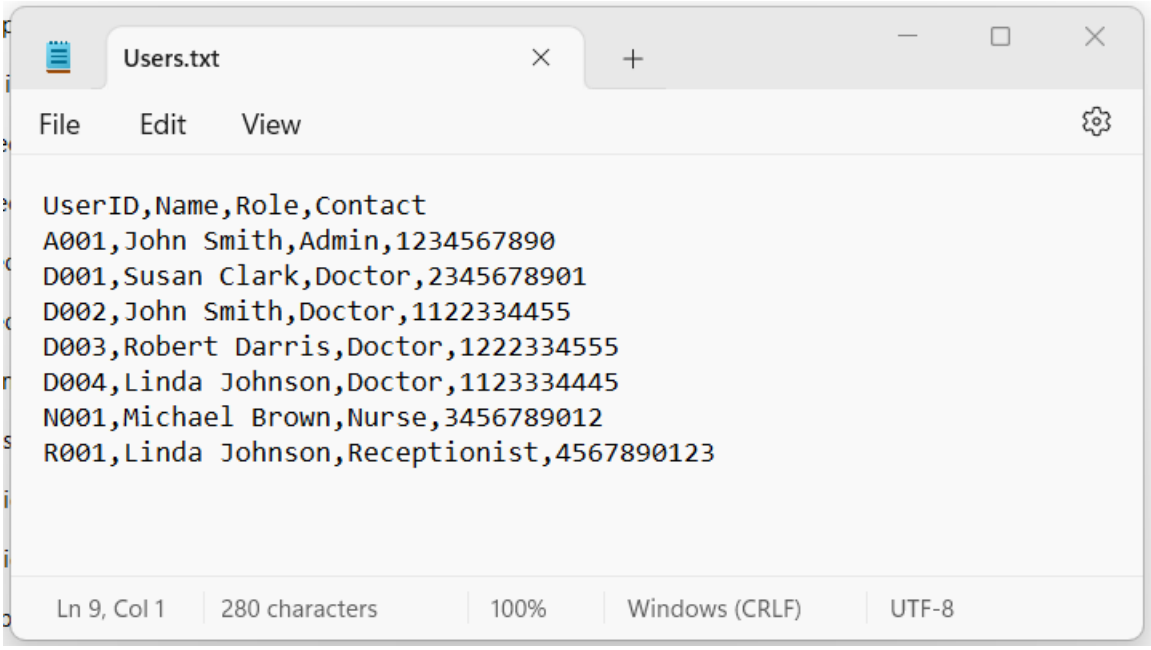
Workload Matrix:

Name	Role	Contribution	Individual Task
Syed Kazim **(Group Leader)**	Administrator	• Compiling codes • Documentaion and formatting • Assigning Tasks among Team • Conducting Meetings • Additional Feature • Application of Validation • Flow of Generating Reports • Screenshots of Text files • References	• Screenshots of input/output with explanation of **Administrator** • Pseudocode of **Administrator** • Python Code for Administrator
Pawan	Doctor	• Introduction & Assumption • Flow of Delete Function	• Screenshots of input/output with explanation of **Doctor** • Pseudocode of **Doctor** • Python Code for Doctor's Role
Nour	Nurse	• Application of storage types • Application of Control Structure	• Screenshots of input/output with explanation of **Receptionist** • Pseudocode of **Receptionist** • Python Code for Nurse's Role
Mukarramin	Receptionist	• Conclusion • Application of Try & Except • Additional Feature	• Screenshots of input/output with explanation of **Receptionist** • Pseudocode of **Receptionist** • Python Code for Receptionist's Role

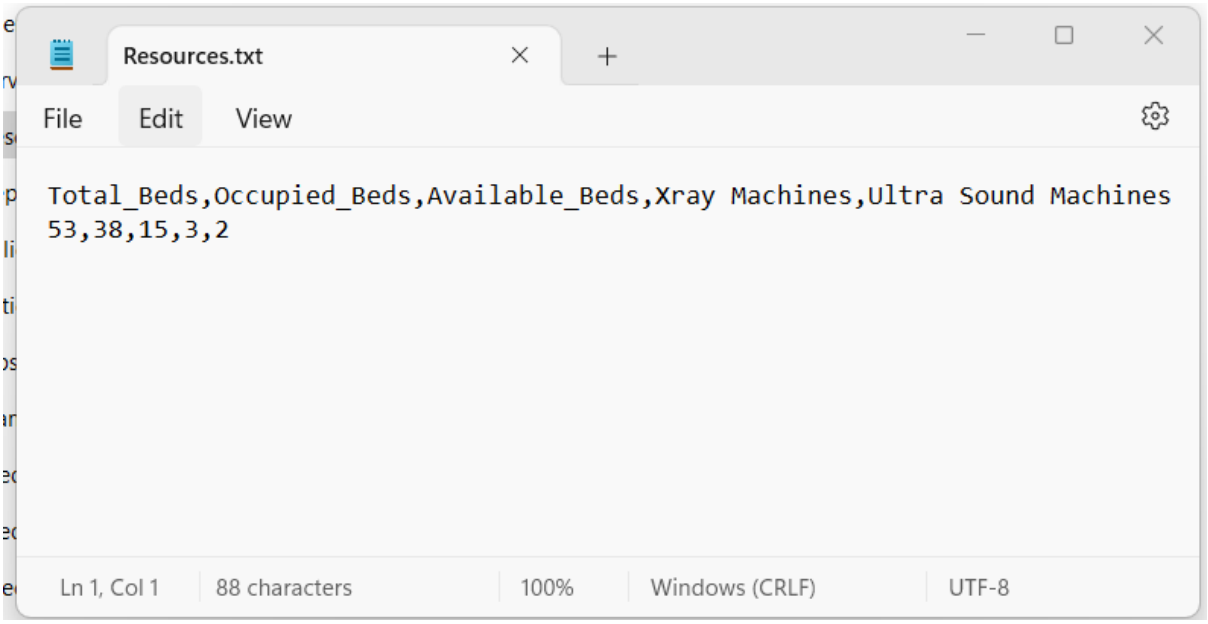
Yahya	Patient	• Flow of Add Function • Flow of update/modify/edit function	• Screenshots of input/output with explanation of **Receptionist** • Pseudocode of **Receptionist** • Python's Code for Patient's Role & **(Helped in fixing bugs and logical mistakes in Nurse's Role)**

"Screen shot Text files"

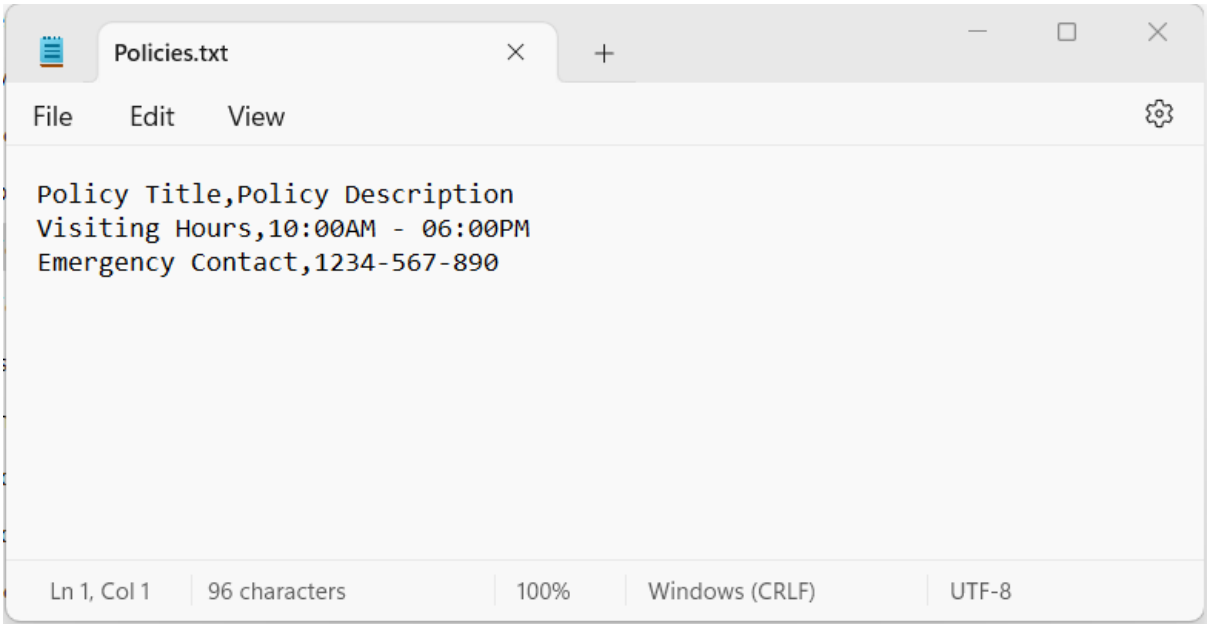
- **Users.txt**
  - Stores hospital staff information.



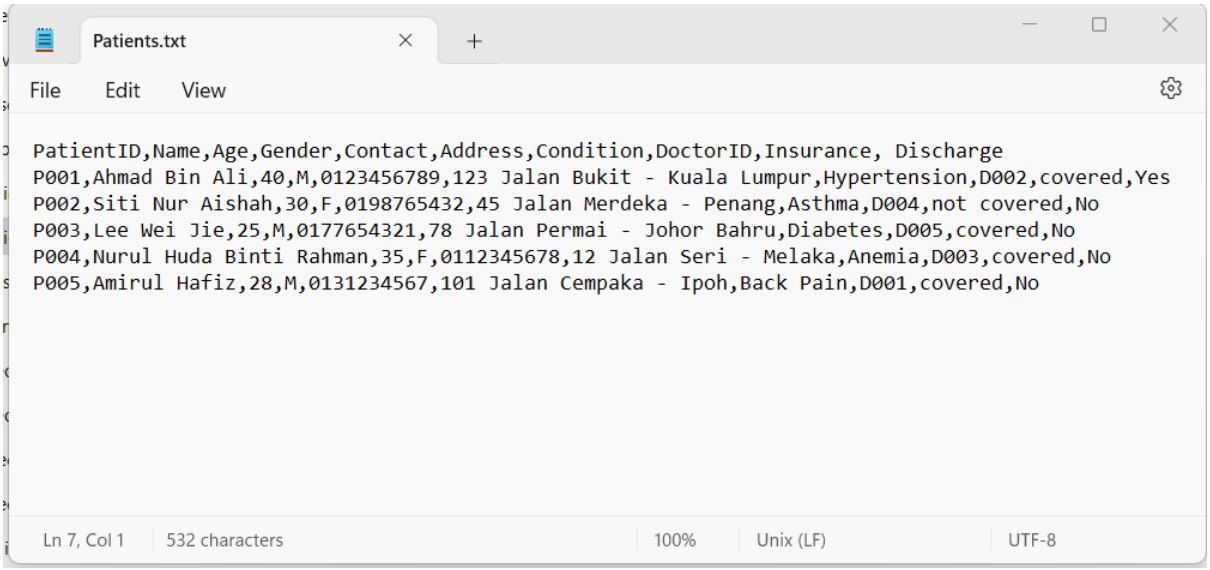
- **Resources.txt**
  - Stores information about Hospital resources.



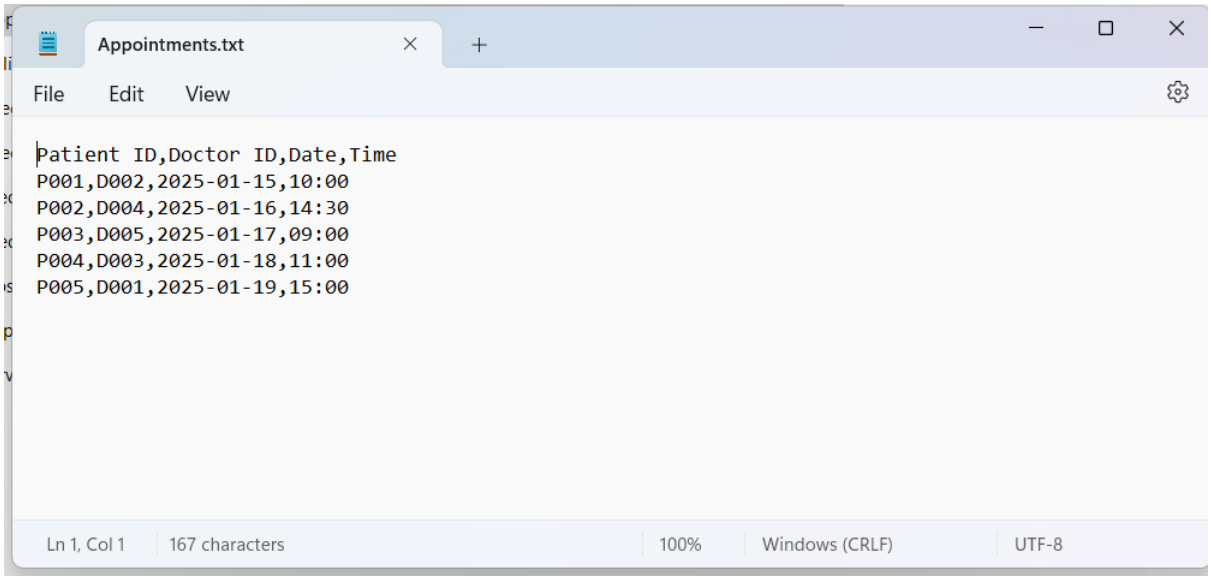
- **Policies.txt**
  - Stores hospital policies



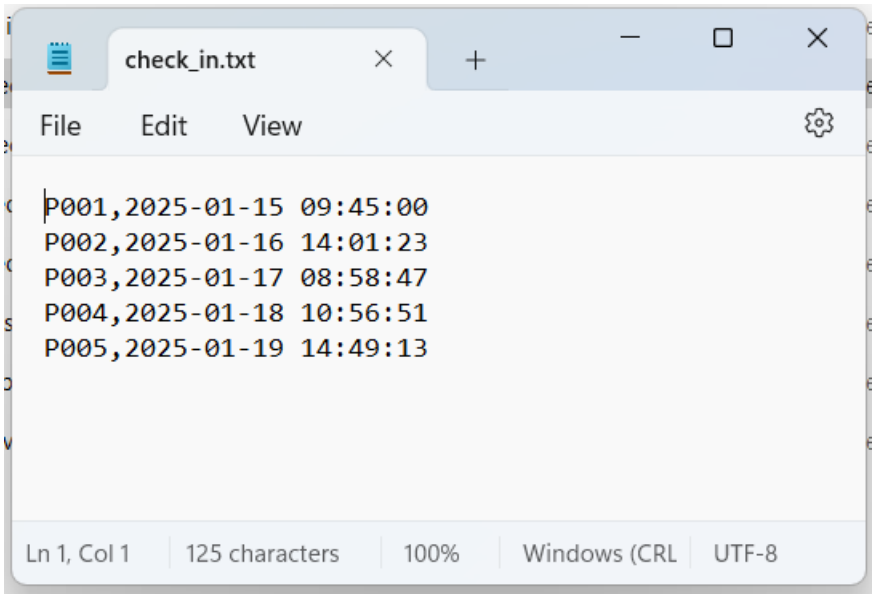
- **Patients.txt**
  - **Stores patients details.**



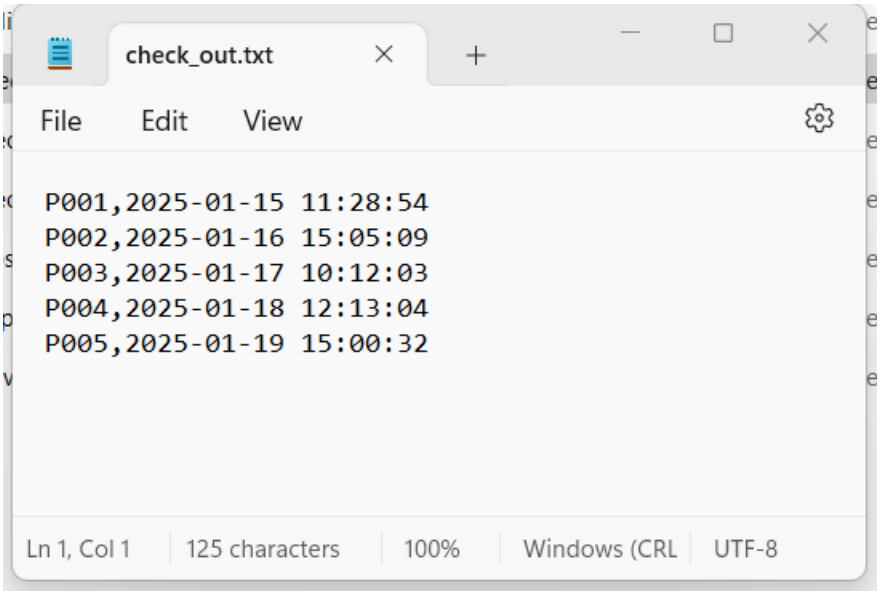
- **Appointments.txt**
  - Stores Appointments details of patients with doctors.



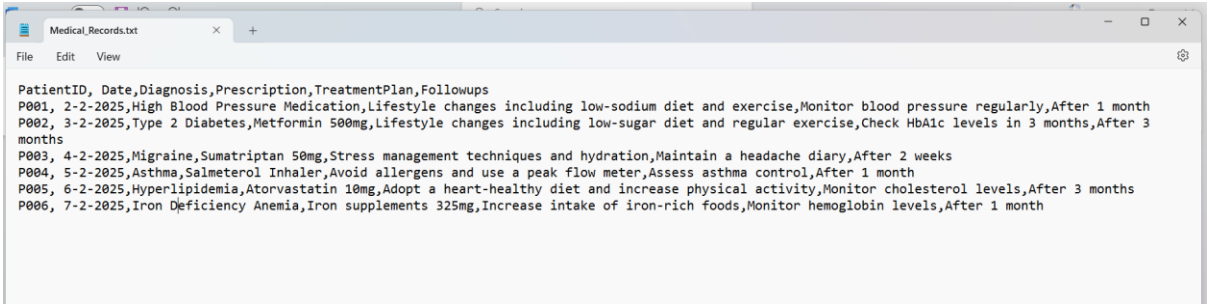
- **check_in.txt**
  - Stores check in details of patients.



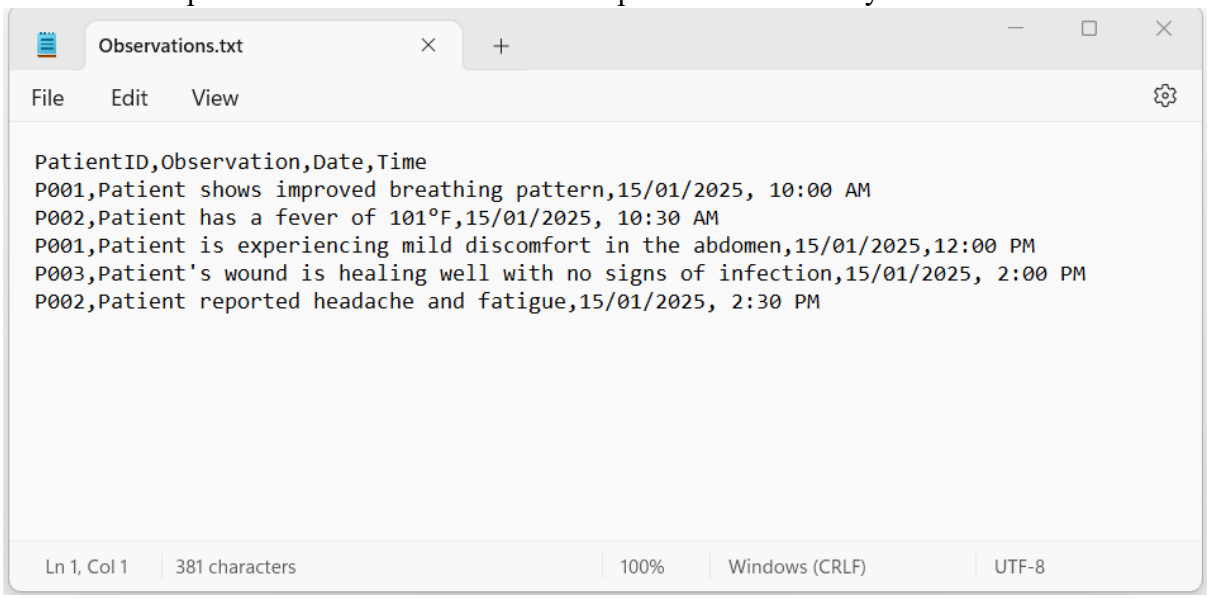
- **check_out.txt**
- Stores check out (discharge) details of patient



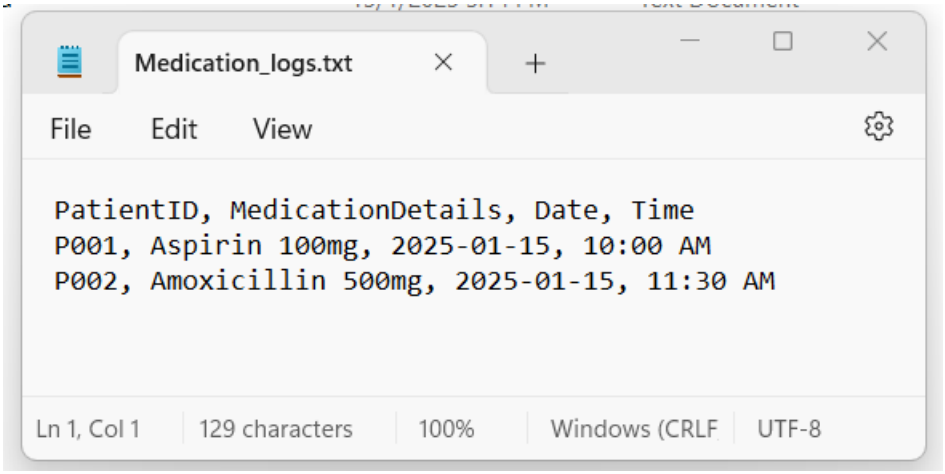
- **Medical_Records.txt**
  - Stores prescriptions, Diagnosis, Treatment plans and followups of patients given by doctors



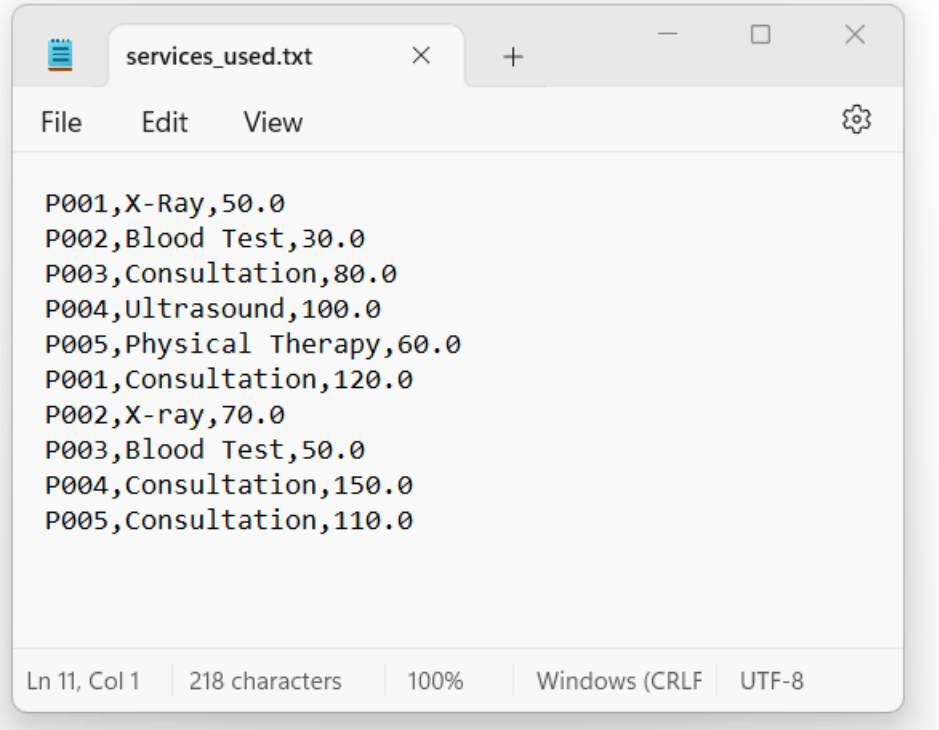
- **Observations.txt**
  - Stores patient vitals and observations of patients observed by Nurse.



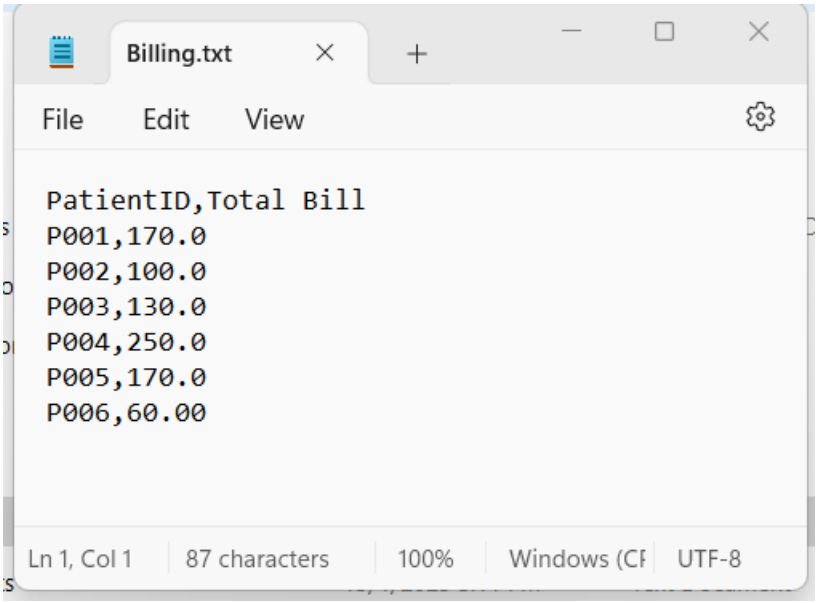
- **Medication_logs.txt**
  - Stores medication timings.



- **Services_used.txt**
  - Stores details about hospital services used by patients.



- **Billing.txt**
  - Stores billing details of patients.



- **Report.txt**
  - Stores details of hospital resources usage/availability and staff strength.