Introduction

A hospital wants to either upgrade or freshly implement a Simple Patient Record System (SPRS) to keep track of patient names and unique ID's in a database. The system must come with an interface for the creation, retrieval, updating, and deletion of entries into the database, all while maintaining security for the data. The only true constraint for the system is the patient ID cannot be modified once made. The SPRS was created to fill this need. Living up to its first S, it is a very simple, menu-based system where the only inputs are the option of the menu, the name of the patient, and the ID to be updated or deleted.

Requirements Traceability

REQ-01 is simply that the system shall retrieve a patient's record using their unique Patient ID. This is a baseline system function with no constraints. This requirement is covered by the method register_patient(), which takes an inputted name and creates an entry in the database tied with a patient ID. This patient ID is unique to each patient due to this use of a counter that is incremented with every entry.

REQ-02 states that the system shall retrieve a patient's record using their unique Patient ID. This, again, is another baseline system function with no real constraints. This is covered by the get_patient() method, which takes an inputted patient ID and searches the database for the name that it's paired with.

REQ-03 is a constraint that states the system shall ensure data integrity by preventing the modification of a Patient ID once it has been assigned. As such, this requirement is something that must be maintained throughout the program. Thankfully it is relatively simple to implement: once the ID is created in register_patient(), it is never touched again by the rest of the program unless to simply read it.

REQ-04 is both a fact and a constraint, though really it only maintains the same constraint as REQ-03. It states the system shall allow updating a patient's name using their Patient ID, while keeping the Patient ID unchanged. This is mostly covered by update_patient_name(), which takes an inputted Patient ID and name, and replaces the original name paired with that ID with the new one. REQ-03 and the latter half of REQ-04 are maintained in that the Patient ID is never edited in the method.

Finally, REQ-05 is a fact that states the system shall allow deleting a patient record using their Patient ID. If the ID does not exist, the system shall display an appropriate error message. This requirement is covered by the method delete_patient(), which takes an inputted Patient ID and removes both the name and ID from the database. The condition is covered by a ValueError raised by the ID not being in the database.

Design Explanation

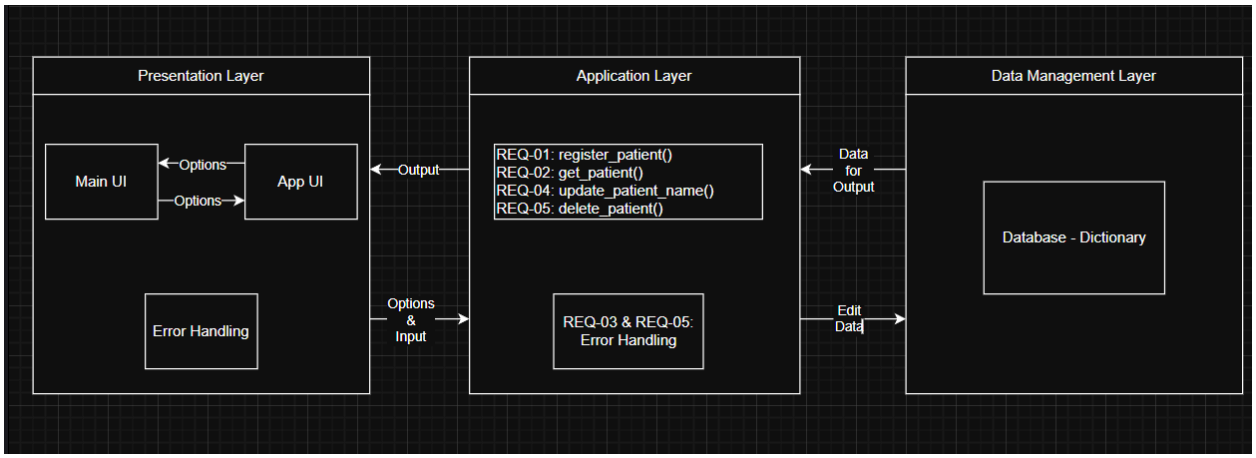
The design for the system was fairly easy due to the ability to limit it all to one class. The only major decision was the separation of main and patient_registry. This separation meant that the encapsulation of data could be maintained by having all data be handled and organized in an entirely different file from main. This way, even if someone were to see the source code of the file they were interacting with, the specifics are hidden behind the patient_registry.py file. For the most part, every requirement was its own method of the class, with the only exception being REQ-03. REQ-01, REQ-02, REQ-04, and REQ-04 are met with a fully functioning method simply meeting the requirements of input and output. REQ-03 is done throughout the implementation but is easily avoided by never editing the patient_id variable as it gets read.

A dictionary was used due to its built-in functions covering the majority of our needs for the system with data storage being a key-value pair. Patient IDs are tied to the names of their respective patients, making it perfect for this specific data type. Registration is covered by adding entries by tying a value (patient name) to a key (patient ID), retrieval is covered by printing out the patient name by inputting the patient ID, updating is by replacing the name stored at the entered patient ID, deletion is covered by the built-in del function, and listing all patients is covered by list and .items().

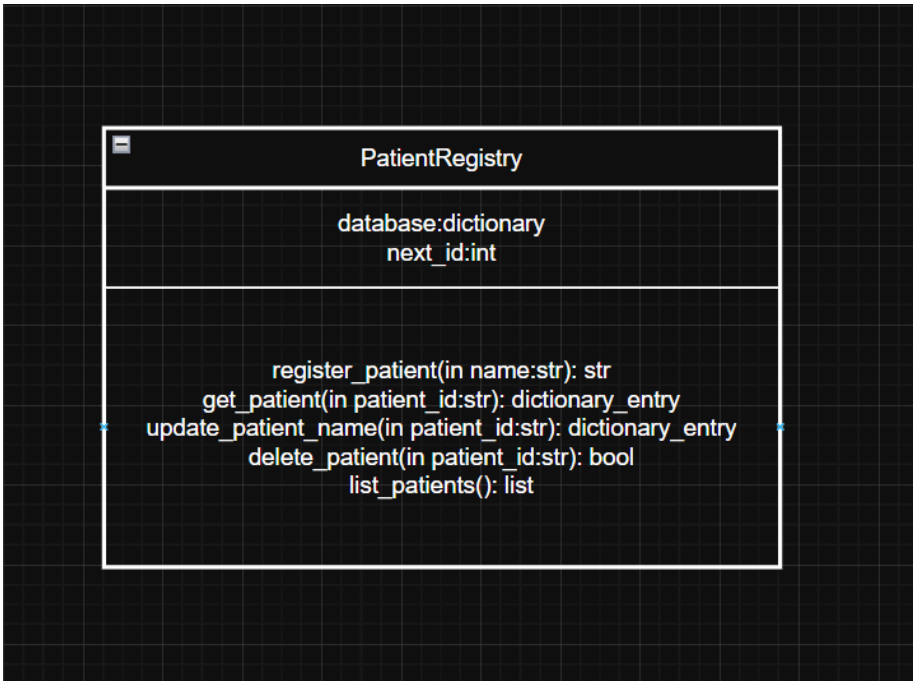
The immutability of the Patient ID is a simple matter, but one that must be maintained throughout the program. Every single time that Patient ID is used, which is several times, it must be ensured that it is never changed. Again, this is relatively simple as the variable only needs to be accessed, not interacted with.

Error handling must be done throughout the system but is mainly two different checks repeated. Anywhere there is input must be checked if the input is empty before it reaches the rest of the method. The only other different error that is checked is if the target of a search is present inside the dictionary. In both cases, either a ValueError or LookupError is raised and the program

stops. Not explicitly mentioned but part of my design is that for the menu input, if any entry other than one of the presented options is entered, a simple message is outputted to the console and you are moved back to the menu to enter it again.



High Level Architecture Diagram



Class Diagram for SPRS

Implementation Discussion

In truth, the implementation was fairly simple, revolving almost entirely around dictionaries and their built-in functions. The only implementation decision that deviated from the required methods was listing all out patients in the database. This was done through the use of a simple for loop and the items() function.

Assumptions were limited when compared to the directions, but there were lots of assumptions in relation to the requirements. Any actual UI implementation was entirely an assumption due to the lack of description from requirements. However, this is off-put by the directions of the assignment itself. Listing all entries in the database is also a full assumption from the requirements, since the only output mentioned is individual patients in REQ-02.

Team Contributions

I worked alone. All work on this project was done by me with the occasional help of Google for dictionary functions due to unfamiliarity with Python.

Screenshot Evidence

```
------------------------------------------------
Please select one of the following options:
        1. Register new patient
        2. Retrieve patient by ID
        3. Update patient name (ID cannot be changed)
        4. Delete patient by ID
        5. List all patients
        6. Go back to main
        7. Exit

Your Choice: 1
------------------------------------------------
Enter Pateint Name: Alice
Registered:  P-101
```

Output for register_patient()

```
------------------------------------------------
Please select one of the following options:
        1. Register new patient
        2. Retrieve patient by ID
        3. Update patient name (ID cannot be changed)
        4. Delete patient by ID
        5. List all patients
        6. Go back to main
        7. Exit

Your Choice: 2
------------------------------------------------
Enter ID of Patient: P-101
{'patient_id': 'P-101', 'name': 'Alice'}
```

Output for get_patient()

```
--------------------------------------------------
Please select one of the following options:
        1. Register new patient
        2. Retrieve patient by ID
        3. Update patient name (ID cannot be changed)
        4. Delete patient by ID
        5. List all patients
        6. Go back to main
        7. Exit

Your Choice: 3
--------------------------------------------------
Enter ID of Patient to be Changed: P-101
Enter New Name of P-101: Morgana
Updated:  Morgana
```

Output for update_patient_name()

```
--------------------------------------------------
Please select one of the following options:
        1. Register new patient
        2. Retrieve patient by ID
        3. Update patient name (ID cannot be changed)
        4. Delete patient by ID
        5. List all patients
        6. Go back to main
        7. Exit

Your Choice: 4
--------------------------------------------------
Enter ID of Patient to be Deleted: P-101
Deletion of P-101 successful.
```

Output for delete_patient()

```
--------------------------------------------------
Please select one of the following options:
        1. Register new patient
        2. Retrieve patient by ID
        3. Update patient name (ID cannot be changed)
        4. Delete patient by ID
        5. List all patients
        6. Go back to main
        7. Exit

Your Choice: 5
--------------------------------------------------
[{'patient_id': 'P-102', 'name': 'Bob'}]
```

Output for list_patients()

Source Code

```
class PatientRegistry:
    '''
    Manage patient names and IDs in a dictionary

    This class provides methods to register, retrieve, update, delete, and list patients.

    Data Fields:
        _database (dict[str, str]) - maps patient IDs to names
        _next_id (int) - tracks entries for unique patient IDs
    '''
    # Needs name and ID stored in simulated database
    def __init__(self):
        # Simulated Database
        self._database = {}
        # Unique ID - incremented for each user
        self._next_id = 101

    # Methods
    def register_patient(self, name:str) -> str:
        '''
        Creates a unique ID for a patient and creates a new entry in the database give name

        Requirements:
            REQ-01 - The system shall allow the creation of a new patient record with a name and
a unique Patient ID.
            REQ-03 - The system shall ensure data integrity by preventing the modification of a
Patient ID once it has been
                     assigned.

        Args:
            name (str) - name of patient

        Returns:
            patient_id (str) - unique ID of patient
        '''
        #Test input
        if not name:
            raise ValueError("Name cannot be empty")

        # Create unique patient ID
        patient_id = f"P-{self._next_id}"
        # Increcemnt ID so next one is unique
        self._next_id += 1

        #Store name and ID in database
        self._database[patient_id] = { "patient_id": patient_id, "name": name }

        return patient_id

    def get_patient(self, patient_id:str) -> dict:
        '''
```

```
        Retrieves patient name from database given ID

        Requirements:
            REQ-02 - The system shall retrieve a patient's record using their unique Patient ID.

        Args:
            patient_id - name of patient

        Returns:
            dictionary entry of patient
        '''
        #Test input
        if not patient_id:
            raise ValueError("Patient ID cannot be empty")
        elif patient_id not in self._database:
            raise LookupError(f"No patient found with ID \'{patient_id}\'")


        return self._database.get(patient_id, "ID Not Found")


    def update_patient_name(self, patient_id:str) -> dict:
        '''
        Updates the name of a patient in the database to given name

        Requirements:
            REQ-02 - The system shall allow updating a patient’s name using their Patient ID,
while keeping the Patient ID
                     unchanged.

        Parameters:
            patient_id (str) - id of patient to be changed

        Returns:
            dictionary entry of patient
        '''
        #Test input
        if not patient_id:
            raise ValueError("Patient ID cannot be empty")
        elif patient_id not in self._database:
            raise LookupError(f"No patient found with ID \'{patient_id}\'")

        new_name = input("Enter New Name of " + patient_id + ": ").strip()

        self._database[patient_id] = { "patient_id": patient_id, "name": new_name }

        return self._database.get(patient_id, "ID Not Found")


    def delete_patient(self, patient_id) -> bool:
        '''
        Delete an entry in the dataset given patient ID
```

```
        Requirements:
            REQ-05 - The system shall allow deleting a patient record using their Patient ID. If
the ID does not exist, the
                     system shall display an appropriate error message.

        Args:
            patient_id (str) - id of patient to be changed

        Returns:
            new name & old ID of patient
        '''
        #Test input
        if not patient_id:
            raise ValueError("Patient ID cannot be empty")
        elif patient_id not in self._database:
            raise LookupError(f"No patient found with ID \'{patient_id}\'")
        else:
            del self._database[patient_id]
            return True

    def list_patients(self) -> list:
        return list(self._database.values())
```

```
import sys
from patient_registry import PatientRegistry

MAIN_MENU = ("Welcome to the Simple Patient Record System (SPRS)\n"
             "-------------------------------------------------\n"
             "Please select one of the following options:\n"
             "\t 1. Run Application\n"
             "\t 2. Exit\n")

APP_MENU = ("Please select one of the following options:\n"
            "\t1. Register new patient\n"
            "\t2. Retrieve patient by ID\n"
            "\t3. Update patient name (ID cannot be changed)\n"
            "\t4. Delete patient by ID\n"
            "\t5. List all patients\n"
            "\t6. Go back to main\n"
            "\t7. Exit\n")

def get_menu_input(menu:str) -> str:
    '''
    Prints the menu and takes input for it

    Args:
        name (str) - patient's name

    Returns:
        option (str) - choice inputted by user
```

```
    '''
    print(menu)

    option = input("Your Choice: ").strip()

    return option

def run_application(registry:PatientRegistry):
    '''
    Runs the main Simple Patient Record System

    Args:
        registry (PatientRegistry) - class holding the data and methods for the program
    '''
    while True:

                print("-" * 50)
                app_input = get_menu_input(APP_MENU)
                print("-" * 50)

                # Exit the program
                if app_input == "7":
                    print("Exiting SPRS...")
                    sys.exit()
                # Register new patient
                elif app_input == "1":
                    name = input("Enter Pateint Name: ").strip()
                    pid = registry.register_patient(name)
                    print("Registered: ", pid)
                # Retrieve patient ID
                elif app_input == "2":
                    pid = input("Enter ID of Patient: ").strip()
                    print(registry.get_patient(pid))
                # Update patient name
                elif app_input == "3":
                    pid = input("Enter ID of Patient to be Changed: ").strip()
                    pid = registry.update_patient_name(pid)
                    print("Updated: ", pid)
                # Delete patient by ID
                elif app_input == "4":
                    pid = input("Enter ID of Patient to be Deleted: ").strip()
                    deleted = registry.delete_patient(pid)
                    if deleted:
                        print("Deletion of " + pid + " successful.")
                    else:
                        print("***ERROR*** - " + pid + " could not be deleted.")
                # List all patients
                elif app_input == "5":
                    print(registry.list_patients())
                # Go back to main
                elif app_input == "6":
                    break
```

```
                # Error catch for other input
                else:
                    print("***ERROR*** - Input must be a whole number between 1 and 7
inclusively.")

def main():
    '''Entry point of the program'''
    registry = PatientRegistry()
    while True:
        main_input = get_menu_input(MAIN_MENU)
        print("-" * 50)
        # Exit the program
        if main_input == "2":
            print("Exiting SPRS...")
            sys.exit()
        # Run the SPRS
        elif main_input == "1":
            run_application(registry)
        # Error catch for other input
        else:
            print("***ERROR*** - Input must be 1 or 2.")

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