**B.M.S COLLEGE OF ENGINEERING BENGALURU**

Autonomous Institute, Affiliated to VTU



**Python** **AAT**

**Report on**

**“Patient Record App”**

*Submitted in partial fulfillment of the requirements for AAT*

Bachelor of Engineering

in

Computer Science and Engineering

*Submitted by:*

Ayush G

(1BM22CS063)

Pranav R Hegde

(1BM22CS202)

Department of Computer Science and Engineering

B.M.S College of Engineering

Bull Temple Road, Basavanagudi, Bangalore 560 019

2022-2023

**B.M.S College of EngineerinG**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

****

***DECLARATION***

We, Ayush G(1BM22CS063), Pranav R Hegde (1BM22CS202) students of 2nd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this AAT Project entitled " Patient Record App " has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester June - August 2023. I also declare that to the best of our knowledge and belief, the AAT Project report is not from part of any other report by any other students.

**Signature of the Candidate**

Ayush G(1BM22CS063)

Pranav R Hegde (1BM22CS202)

**BMS College of EngineerinG**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

****

***CERTIFICATE***

This is to certify that the AAT Project titled “Patient Record App**”** has been carried out by Ayush G(1BM22CS063), Pranav R Hegde (1BM22CS202)during the academic year 2022-2023.

Signature of the Faculty in Charge

Table of Contents

|  |  |  |
| --- | --- | --- |
| Sl. No | Title | Page no |
| 1 | Introduction | 5 |
| 1.1 | Problem Definition | 5 |
| 1.2 | Scope | 5 |
| 1.3 | Motivation | 5 |
| 1.4 | Abstract | 6 |
| 2 | Hardware and Software requirement | 6 |
| 3 | Design | 6 |
| 3.1 | Project Flow | 7 |
| 3.2 | Algorithm |  |
| 4 | Implementation |  |
| 4.1 | Source code |  |
| 4.2 | Experimental Analysis and Results |  |
| 5 | Conclusion |  |
| 6 | References |  |

1 INTRODUCTION

The provided Python program implements a simple patient record management system. This system is designed to help healthcare professionals keep track of patient information and medical records in a command-line interface. It allows users to add patient details, add medical records to existing patients, and view the medical records of specific patients.

* 1. PROBLEM DEFINITION

In the fast-paced world of healthcare, maintaining accurate patient records is crucial for providing effective medical care. The "Patient Record App" presented here is a lightweight tool that assists healthcare professionals in managing patient information and medical records efficiently. This command-line application enables users to input patient data, add medical records, and retrieve patient-specific medical history.

* 1. SCOPE

Data Validation: Enhance the system by adding input validation for fields like patient IDs, dates, and other user inputs. Ensure that the data entered adheres to appropriate formats and constraints.

Error Handling: Implement comprehensive error handling to gracefully manage scenarios such as invalid patient IDs, incorrect dates, or unexpected inputs.

Data Persistence: Integrate a data storage mechanism such as a database or a file system to store patient records persistently between application runs. This ensures data is not lost when the program terminates

Search and Filtering: Extend the application to allow users to search and filter records based on various criteria, such as date range, diagnosis, or specific keywords.

Editing and Deleting Records: Include functionality to edit or delete existing patient records. This is important for maintaining accurate and up-to-date medical histories.

User Authentication: If the application is used in a secure environment, you can add user authentication to restrict access to authorized medical professionals

Data Privacy and Security: Implement data encryption and security measures to ensure patient data remains confidential and compliant with privacy regulations.

Reporting: Provide the ability to generate reports summarizing patient records, medical history, and trends.

* 1. MOTIVATION

1.4 ABSTRACT

Some of the Features:

* Add Patient: Users can input essential patient details, such as patient ID, name, date of birth, and diagnosis. This information is stored in the system, creating a unique patient profile.
* Add Record: For existing patients, medical practitioners can add medical records, including the date of the record and relevant notes. These records are associated with the respective patient's profile.
* Display Records: The application allows users to retrieve and view the medical records of a particular patient. This feature is helpful for medical professionals when evaluating a patient's medical history.
* User-Friendly Interface: The program's command-line interface guides users through the available options and provides prompts for necessary information.

Exit: Users can exit the application when they have completed their tasks.

2 HARDWARE AND SOFTWARE REQUIREMENT

Hardware Requirements:

• Personal Computer or Laptop

• Keyboard and Mouse

Software Requirements:

• Operating System: Windows, macOS, or Linux

• Code Editor/IDE: Google Colab

3DESIGN**:**

The program follows a simple design and flow.

Program stars with declaring all neccessary variables.

Program just demands numbers from customers for name,Dob and digits for further their record

Conditional statements (if-elif)are used.

3.1 PROJECT FLOW:

Certainly, here's the flow of the project based on the provided code:

1. The `Patient Record App` class is defined to manage patient records.

2. The constructor `\_\_init\_\_` initializes an empty dictionary called `patients` to store patient information and records.

3. The `add\_patient` method is used to add a new patient to the records. It takes parameters like `patient\_id`, `name`, `dob` (date of birth), and `diagnosis`. It creates a dictionary entry for the patient with their details and an empty list for records.

4. The `add\_record` method is used to add medical records to an existing patient. It takes parameters like `patient\_id`, `date`, and `notes`. It checks if the provided `patient\_id` exists in the records and appends a new record (date and notes) to the patient's record list.

5. The `display\_records` method is used to display all the records of a specific patient. It takes `patient\_id` as a parameter and checks if the patient exists. If found, it prints the patient's name and then iterates through their records, printing the date and notes for each record.

6. The `run` method implements the main loop of the application, which continues until the user decides to exit. It displays a menu with options: adding a patient, adding a record, displaying records, and exiting.

7. Inside the loop, the user's choice is obtained using the `input` function. Depending on the choice, the respective method (add\_patient, add\_record, or display\_records) is called.

8. If the user chooses to exit, the loop breaks, and the application ends.

3.2 ALGORITHM:

1. Initialize the Patient RecordApp class:

• Initialize an empty dictionary called patients in the constructor.

2. Add Patient:

• Prompt the user for patient details: patient\_id, name, dob, and diagnosis.

• Create a dictionary with the patient's information and an empty list for records.

• Add the patient information dictionary to the patients dictionary, using patient\_id as the key.

3. Add Record:

• Prompt the user for patient\_id, date, and notes.

• Check if the provided patient\_id exists in the patients dictionary.

• If found, append the (date, notes) tuple to the records list of the patient.

• If not found, display a message indicating that the patient was not found.

4. Display Records: • Prompt the user for patient\_id.

• Check if the provided patient\_id exists in the patients dictionary.

• If found, print the patient's name and iterate through the records list:

• Print each record's date and notes.

• If not found, display a message indicating that the patient was not found.

5. Run the Application:

• Enter a loop that runs until the user chooses to exit.

• Display a menu with options:

• 1. Add Patient •

• Add Record

• 3. Display Records

• 4. Exit

• Prompt the user for their choice.

• Based on the user's choice:

• If 1: Call add\_patient method.

• If 2: Call add\_record method.

• If 3: Call display\_records method.

• If 4: Display "Exiting the app" message and break out of the loop.

• For any other input: Display "Invalid choice" message.

2. Main Execution:

• Check if the script is being run directly (not imported as a module).

• Create an instance of PatientRecordApp called app.

• Call the run method on the app instance to start the app's main loop

4 IMPLEMENTATION:

Please note that this code is meant to be run in a command-line environment. To interact with the application, you'll need to run it in a terminal or command prompt. The program will guide you through the process of adding patients, adding records, displaying records, and exiting the application.

4.1 SOURCE CODE:

class PatientRecordApp:

def \_init\_(self):

self.patients = {}

def add\_patient(self, patient\_id, name, dob, diagnosis):

self.patients[patient\_id] = {

'name': name,

'dob': dob,

'diagnosis': diagnosis,

'records': []

}

def add\_record(self, patient\_id, date, notes):

if patient\_id in self.patients:

self.patients[patient\_id]['records'].append((date, notes))

else:

print("Patient not found!")

def display\_records(self, patient\_id):

if patient\_id in self.patients:

print(f"Patient Name: {self.patients[patient\_id]['name']}")

print("Records:")

for date, notes in self.patients[patient\_id]['records']:

print(f"Date: {date}, Notes: {notes}")

else:

print("Patient not found!")

def run(self):

while True:

print("\nPatient Record App")

print("1. Add Patient")

print("2. Add Record")

print("3. Display Records")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == '1':

patient\_id = input("Enter Patient ID: ")

name = input("Enter Name: ")

dob = input("Enter Date of Birth: ")

diagnosis = input("Enter Diagnosis: ")

self.add\_patient(patient\_id, name, dob, diagnosis)

print("Patient added successfully!")

elif choice == '2':

patient\_id = input("Enter Patient ID: ")

date = input("Enter Date: ")

notes = input("Enter Notes: ")

self.add\_record(patient\_id, date, notes)

print("Record added successfully!")

elif choice == '3':

patient\_id = input("Enter Patient ID: ")

self.display\_records(patient\_id)

elif choice == '4':

print("Exiting the app.")

break

else:

print("Invalid choice. Please select again.")

if \_name\_ == "\_main\_":

app = PatientRecordApp()

app.run()

4.2 EXPERIMENTAL ANALYSIS AND RESULTS

1) Here are some of the outputs.

5. CONCLUSION

The Patient Record Management System code provides a foundational platform for healthcare professionals to efficiently manage patient information and medical records. This code offers a command-line interface that facilitates the addition of patient details, recording medical information, viewing patient records, and exiting the application. While the code serves as a starting point, there are various ways to enhance and expand its functionality.

By implementing data validation, error handling, and data persistence mechanisms, the system can become more robust and reliable. Integrating features like search and filtering, editing and deleting records, and user authentication can significantly improve user experience and data security.

6. REFERENCES:

1. Google Colab- https://research.google.com/colaboratory/
2. Jupyter - https://jupyter.org/
3. GDB Online Debugger- https://www.onlinegdb.com/online\_python\_compiler
4. Idle Python Compiler-