**Documentation for the Code**

**Overview**

This program facilitates a basic hospital management system where employees and patients can interact with the system to access or manage data. It includes modules for doctors, pharmacies, vitals, insurance, and patient information access.

The system includes three primary functionalities:

1. **Employee Portal (E):** Allows employees to log in and manage their respective department data.
2. **Patient Portal (P):** Enables patients to access their medical information securely through security questions or OTPs.
3. **New User Creation:** Permits the creation of new employee accounts under various roles (Doctor, Pharmacy, Insurance, etc.).

The program interacts with several files like employee.json, insurance\_payment\_report.json, eye\_patient\_info.csv, and health\_data.json for data storage and retrieval.

**Code Breakdown**

1. **Global Data:**
   * reg: A dictionary containing appointment and metadata information.
   * dic: Patient IDs as keys with placeholder values.
   * sq: Security questions for validating patient identity.
2. **Utility Functions:**
   * **grant\_access(x)**: Displays patient details, diagnosis, prescriptions, payment, and rejection information from insurance\_payment\_report.json.
   * **diagnostics(id)**: Placeholder for patient diagnostic data.
   * **insurance(id)**: Reads patient insurance information from eye\_patient\_info.csv.
   * **doctor(id)**: Retrieves health data for a patient from health\_data.json.
   * **pharmacy(id)**: Reads diagnostic results for a patient from diagnosis\_results.csv.
   * **send\_email(mail, num)**: Sends an OTP to a patient's email address for identity verification.
3. **Main Workflow:**
   * **Option E:** Employee login to access department-specific functions based on ID and password validation.
   * **Option P:** Patient portal requiring authentication via security questions or OTP.
   * **Option new:** Creation of new users for different departments after ensuring the uniqueness of ID.

**Explanation of Methods Used in the Code**

**1. grant\_access(x)**

* **Purpose: Grants access to patient details and displays their diagnosis, prescribed drugs, and payment details based on an external JSON file.**
* **Key Functionality:**
  + **Opens a JSON file insurance\_payment\_report.json and reads its content.**
  + **Displays structured information about the patient, diagnosis, drugs, and payment details.**
  + **Iterates through lists and dictionaries to extract and display relevant details.**

**2. diagnostics(id)**

* **Purpose: Displays a portion of the patient’s diagnostic information.**
* **Key Functionality:**
  + **Checks the input id to determine which diagnostic information to retrieve.**
  + **Prints part of the diagnostics data from the reg variable.**

**3. insuarance(id)**

* **Purpose: Displays patient insurance information from a CSV file.**
* **Key Functionality:**
  + **Opens the file eye\_patient\_info.csv and reads its lines.**
  + **Splits lines into columns using commas.**
  + **Retrieves and displays information for the specified patient ID.**
  + **Handles header and row processing for tabular data.**

**4. doctor(id)**

* **Purpose: Displays health data for a patient based on their ID.**
* **Key Functionality:**
  + **Reads a JSON file health\_data.json.**
  + **Retrieves the health data for the specified patient ID.**
  + **Iterates through key-value pairs in the health data dictionary and formats the output for display.**

**5. pharmacy(id)**

* **Purpose: Retrieves and displays diagnostic results for a patient from a CSV file.**
* **Key Functionality:**
  + **Reads data from diagnosis\_results.csv.**
  + **Uses a csv.DictReader to parse the file.**
  + **Retrieves diagnostic results for a specific patient based on ID.**

**6. send\_email(mail, num)**

* **Purpose: Sends an email containing a One-Time Password (OTP) to a specified email address.**
* **Key Functionality:**
  + **Uses the smtplib library to send emails via SMTP.**
  + **Creates an email with a subject and body using MIMEMultipart.**
  + **Sends the email securely by connecting to an SMTP server and logging in.**
  + **Handles errors during the email-sending process.**

**7. Main Logic**

* **Purpose: Determines the flow of the program based on user input.**
* **Key Functionality:**
  + **Accepts input to determine whether the user is an employee (E), a patient (P), or creating a new user (new).**

**Employee Logic:**

* + **Reads a JSON file employee.json to verify login credentials.**
  + **Determines the employee's department and retrieves appropriate patient information using relevant functions (doctor, pharmacy, insuarance, or diagnostics).**

**Patient Logic:**

* + **Validates the patient ID and allows authentication through security questions or OTP.**
  + **Uses the grant\_access method to display patient details if authentication is successful.**

**New User Logic:**

* + **Reads the employee.json file to ensure a new ID is unique.**
  + **Adds the new user credentials to the file and saves it back.**

**Additional Notes:**

1. **Variables:**
   * **reg: Stores mock appointment data.**
   * **dic: Maps patient IDs to validity states.**
   * **sq: Maps patient IDs to their security question answers.**
2. **File Handling:**
   * **Multiple files are used: insurance\_payment\_report.json, eye\_patient\_info.csv, health\_data.json, and diagnosis\_results.csv.**
   * **The code frequently opens, reads, and processes files to retrieve data.**
3. **Email Functionality:**
   * **Uses smtplib and email.mime for sending OTPs securely.**
   * **Requires proper email setup (e.g., SMTP server and valid credentials).**
4. **Error Handling:**
   * **Limited error handling; some cases, like invalid file paths or malformed data, may cause runtime errors.**

**Suggested Improvements:**

* **Refactor repetitive code in the new user logic to a single reusable function.**
* **Implement proper error handling for file operations and invalid user input.**
* **Validate email credentials and enhance security measures for sensitive data (e.g., environment variables for storing email credentials).**

**File Descriptions**

1. **employee.json:**
   * Stores employee IDs and passwords.
   * Example format:

[

{"ID": "D123", "Password": "pass123"},

{"ID": "P456", "Password": "pass456"}

]

1. **insurance\_payment\_report.json:**
   * Contains detailed insurance and diagnosis data for patients.
   * Example format:

{

"Patient\_ID": "1",

"Patient\_Name": "Chinyemba Kalenga",

"Doctor\_ID": "D123",

"Doctor\_Name": "Dr. Smith",

"Diagnosis": [

{

"CPT\_Code": "12345",

"CPT\_Description": "Routine Checkup",

"Condition\_Name": "Hypertension",

"ICD10\_Code": "I10",

"ICD10\_Description": "Primary hypertension",

"Price": 150.00

}

],

"Prescribed\_Drugs": ["Amlodipine", "Losartan"],

"Payment\_Details": {"Insurance Covered": 100.00, "Out-of-pocket": 50.00},

"Rejections": ["Insurance claim rejected for lab tests."]

}

1. **eye\_patient\_info.csv:**
   * Stores patient insurance information.
   * Example format:

csv

Copy code

ID,Name,Age,Gender,Insurance

1,Chinyemba Kalenga,29,F,Yes

2,Ali,34,M,No

1. **health\_data.json:**
   * Contains health data per patient.
   * Example format:

json

Copy code

[

{"health\_data": {"Blood Pressure": ["120/80"], "Allergies": ["Penicillin"]}},

{"health\_data": {"Blood Pressure": ["140/90"], "Allergies": ["None"]}}

]

**Team Member Contributions**

1. **Dingani Freddie Kandiwo: Backend Developer**
   * **Contributions:**
     + Implemented the main functionality for the employee and patient portals.
     + Developed functions for data retrieval (grant\_access, doctor, insurance, pharmacy).
     + Worked on OTP generation and email integration in send\_email.
2. **Josep Boateng: Data Specialist**
   * **Contributions:**
     + Created and formatted JSON and CSV files for storage.
     + Ensured compatibility of file formats with the parsing logic.
     + Debugged issues related to file handling and data validation.
3. **Khloe Edwards: Security and Testing**
   * **Contributions:**
     + Designed and implemented the patient authentication system using OTP and security questions.
     + Validated login processes for employees and patients.
     + Performed rigorous testing to ensure system robustness and handled error cases (e.g., invalid inputs).

**Future Enhancements**

* Add a database backend for scalable and robust data storage.
* Implement encryption for sensitive data like passwords.
* Develop a GUI or web-based front-end for better usability.
* Integrate advanced security measures like two-factor authentication.

This modular structure and division of responsibilities ensured the project was completed efficiently while maintaining code quality.