Case Study: Health Monitoring System (CLI Application)

Detailed Problem Statement

You are tasked with developing a Health Monitoring System that tracks patient health metrics, allows doctors to monitor patients, and provides alerts for abnormal readings. The system will be a Command Line Interface (CLI) application with a robust backend.

System Requirements

Functional Requirements

1. **User Roles**:
    - **Patient**: Can register, log in, record health metrics, and view health history.
    - **Doctor**: Can register, log in, monitor assigned patients, view health metrics, and respond to alerts.
    - **Admin**: Can manage user accounts and configure alert thresholds.
2. **Health Metrics**:
    - The system should track various health metrics such as blood pressure, heart rate, and blood sugar levels.
    - Patients should be able to input their health metrics through the CLI.
    - Health metrics should be timestamped and stored in the database.
3. **Alerts**:
    - The system should generate alerts for abnormal health readings based on predefined thresholds.
    - Alerts should be prioritized based on severity (e.g., critical, high, medium, low).
    - Doctors should be notified of these alerts and able to view them in a dashboard.
4. **Reports**:
    - The system should generate reports for patients' health metrics over time.
    - Patients and doctors should be able to download these reports as PDF files.

Non-Functional Requirements

1. **Scalability**: The system should be able to handle a growing number of users and data entries without performance degradation.
2. **Security**: Sensitive data should be encrypted, and access should be controlled using authentication and authorization mechanisms.
3. **Usability**: The user interface should be intuitive and accessible to users with varying levels of technical expertise.

Technical Requirements

1. **RDBMS**:
    - Design a relational database schema to store users, health metrics, and alerts.
    - Implement SQL queries for retrieving and manipulating data.
2. **Core Java**:
    - Define classes for users (patients and doctors), health metrics, and alerts.
    - Implement business logic to handle user actions such as recording health metrics and responding to alerts.
3. **JPA**:
    - Use JPA annotations to map Java classes to database tables.
    - Implement repository classes for database operations.
4. **Spring Boot (REST APIs)**:
    - Develop RESTful APIs for user registration, login, recording health metrics, viewing health history, and managing alerts.
    - Implement controllers to handle API requests and responses.
    - Secure the APIs using Spring Security (JWT or OAuth2).
5. **Data Structures and Algorithms**:
    - Use data structures like PriorityQueue to manage alerts based on severity.
    - Implement algorithms to detect abnormal health readings and trigger alerts.
    - Use appropriate data structures to store and manage health metrics and user information.

CLI Menu System

The application should present a menu system to navigate through different functionalities. Below is a suggested structure for the CLI menu. Feel free to improvise.

Main Menu

1. Login
2. Register
3. Exit

Login Menu (After successful login)

1. **Patient Menu**
    - Record Health Metrics
    - View Health History
    - Generate Health Report

  - Logout

2. **Doctor Menu**
   - View Patients
   - View Alerts
   - Respond to Alerts
   - View Patient Health History
   - Generate Patient Health Report
   - Logout
3. **Admin Menu**
   - Manage Users
   - Configure Alert Thresholds
   - Logout

Detailed CLI Menu and Submenu System

Main Menu

```
1. Login
2. Register
3. Exit
```

Registration Menu

```
Enter User Role (Patient/Doctor/Admin):
Enter Name:
Enter Email:
Enter Password:
```

Login Menu

```
Enter Email:
Enter Password:
```

Upon successful login, display the appropriate submenu based on the user role.

Patient Menu

```
1. Record Health Metrics
2. View Health History
3. Generate Health Report
4. Logout
```

Record Health Metrics Submenu

```
Enter Metric Type (Blood Pressure/Heart Rate/Blood Sugar):
Enter Value:
Timestamp (auto-generated):
```

View Health History Submenu

```
Enter Time Period (Last Week/Last Month/All Time):
```

Generate Health Report Submenu

```
Enter Time Period (Last Week/Last Month/All Time):
Report generated and saved as PDF.
```

Doctor Menu

```
1. View Patients
2. View Alerts
3. Respond to Alerts
4. View Patient Health History
5. Generate Patient Health Report
6. Logout
```

View Patients Submenu

```
List of Patients:
[Patient ID] - [Patient Name]
```

View Alerts Submenu

```
List of Alerts:
[Alert ID] - [Patient Name] - [Metric Type] - [Severity]
```

Respond to Alerts Submenu

```
Enter Alert ID to Respond:
Alert details displayed. Enter response:
```

View Patient Health History Submenu

```
Enter Patient ID:
Enter Time Period (Last Week/Last Month/All Time):
```

Generate Patient Health Report Submenu

```
Enter Patient ID:
Enter Time Period (Last Week/Last Month/All Time):
Report generated and saved as PDF.
```

Admin Menu

```
1. Manage Users
2. Configure Alert Thresholds
3. Logout
```

Manage Users Submenu

```
List of Users:
[User ID] - [User Role] - [User Name]
Options: Add User, Remove User, Update User
```

Configure Alert Thresholds Submenu

```
Enter Metric Type (Blood Pressure/Heart Rate/Blood Sugar):
Enter Threshold Value:
Threshold updated.
```

Evaluation Criteria

1. **Database Design (20%)**:
    - Correctness and normalization of the relational schema.
    - Use of appropriate data types and constraints.

  - Efficiency of SQL queries.

2. **Core Java Implementation (20%)**:
   - Correctness and completeness of the Java classes.
   - Adherence to object-oriented principles.
   - Implementation of business logic.
3. **JPA Integration (20%))**:
   - Proper use of JPA annotations for ORM.
   - Implementation of repository interfaces.
   - Efficiency of database operations using JPA.
4. **CLI Application (20%)**:
   - Correctness and completeness of the CLI application.
   - User-friendly navigation through the menu and submenus.
   - Proper handling of user inputs and outputs.
5. **Data Structures and Algorithms (20%)**:
   - Use of appropriate data structures.
   - Efficiency of algorithms for detecting and managing alerts.
   - Overall system performance and scalability.

Please pay attention to details:

- Additional features like role-based access control, data visualization, and advanced reporting.
- Code quality, including readability, documentation, and testing.

Be innovative and creative. Happy coding 😃