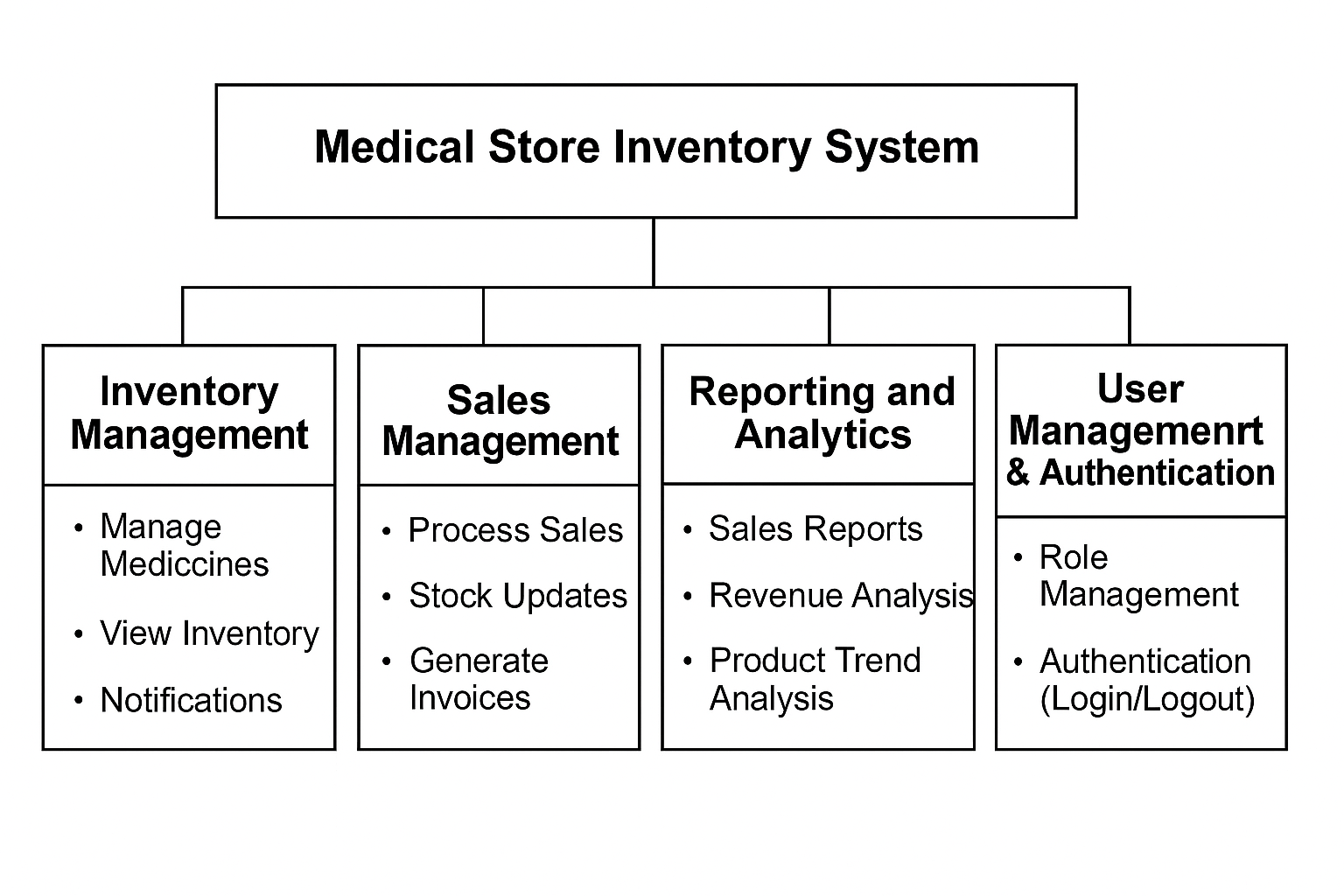
System Architecture

1. UML package diagram



2. Architecture Styles Used

1. Layered (N-Tier) Architecture

The system follows a clear Layered Architecture, dividing responsibilities into logical layers:

Presentation Layer: The user interface developed using Flask templates (HTML/CSS).

Business Logic Layer: Flask routes and Python code handle the core business processes (inventory management, sales processing).

Data Access Layer: SQLite database is accessed through Python to store and retrieve data (medicines, users, sales, etc.).

Advantages:

High separation of concerns.

Easier to maintain and scale.

Each layer can be modified independently.

2. Client-Server Architecture

The system is designed with a clear Client-Server structure:

Client: Users interact with the system through web browsers.

Server: The Flask server processes client requests, interacts with the database, and returns appropriate responses.

Advantages:

Centralized control over the application.

Clients are lightweight and only require a web browser.

3. Model-View-Controller (MVC) Pattern (Lightweight, within Flask)

Though Flask is lightweight, the project internally follows an MVC-inspired separation:

Model: Python classes/methods that interact with the database.

View: HTML templates display data to the user.

Controller: Flask routes handle user requests, perform logic, and return responses.

Advantages:

Better organization of code.

Easy to update the UI or backend independently.

4. Repository Pattern (For database operations)

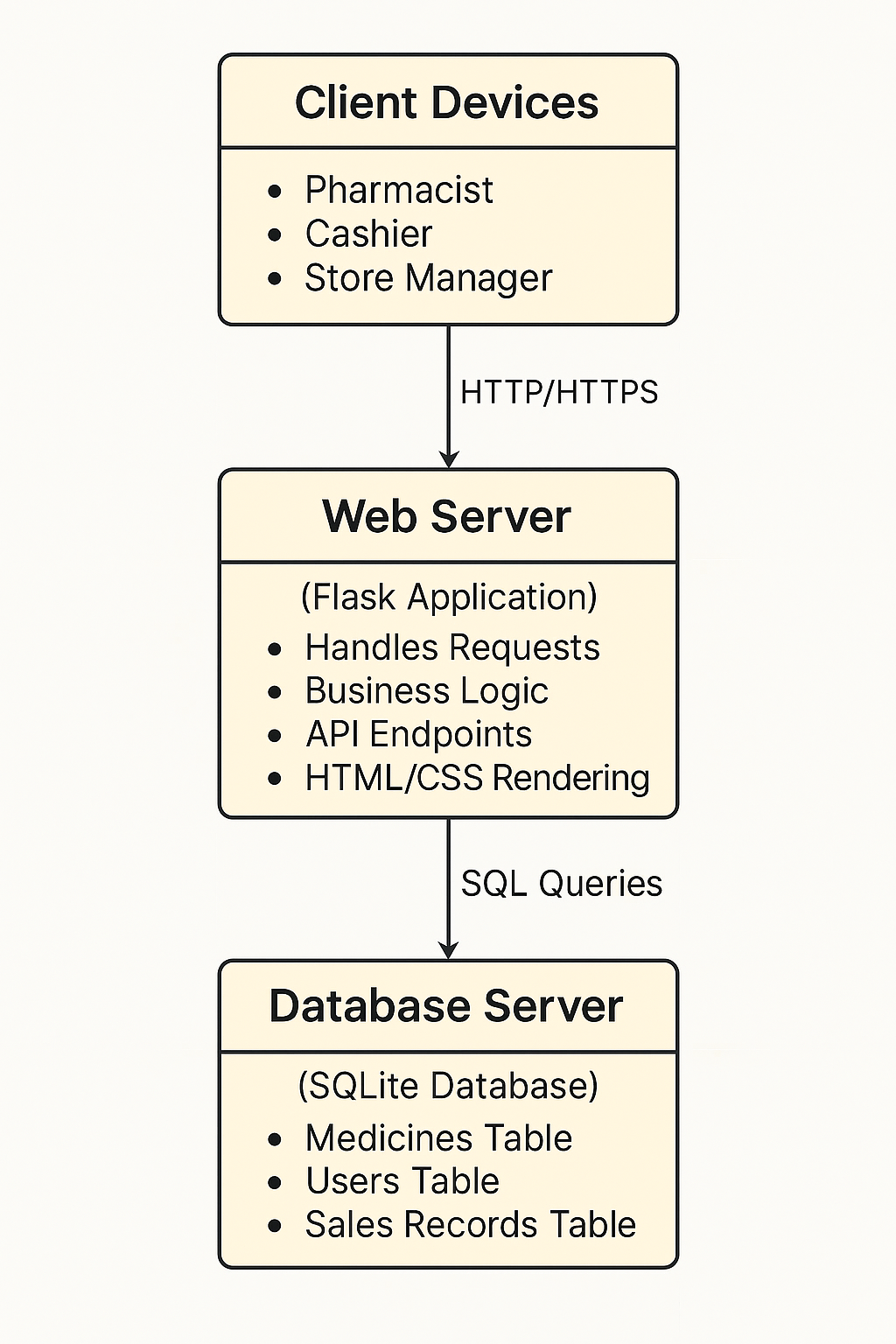
The codebase likely isolates database operations (CRUD) into dedicated Python functions/modules, behaving like a Repository.

Advantages:

Centralized access to database logic.

Easier to change the database (e.g., from SQLite to PostgreSQL) without major app rewrites.

3 Deployment diagram for client deployments



4. Component diagrams

