

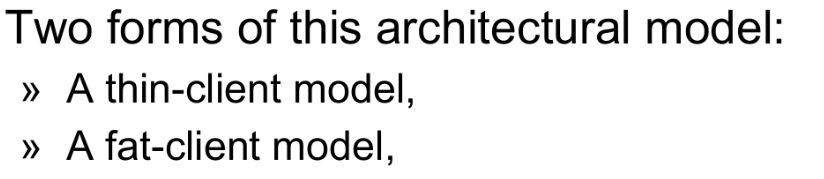
Architectural structure is a static model of a system (i.e. how the system is divided into components)

Architectural behavior is a dynamic model of a system (i.e. how the components interact with each other to perform some useful work)

**Architecture Design**

The major architectural components of any system are the software and the hardware.

The major software components of the system being developed have to be identified and then allocated to the various hardware components on which the system will operate.



A diagram of a server

AI-generated content may be incorrect.A diagram of a data server

AI-generated content may be incorrect.

**Levels/Layers of Architecture Design**

* **Presentation logic** (Client-Side): Web pages, reports
* **Application logic** (Server-Side): Programs, functional requirements realization
* **Data access logic**: SQL
* **Data storage** (Database): File, structured database, video, audio

Quality requirements usually considered during system architecture design:

**Performance:** Localize the operations to minimize subsystem communication.

**Security:** (CIA)

* **C**onfidentiality: Only authorized users can read the information.
* **I**ntegrity: Only authorized users can modify, edit or delete data.
* **A**vailability: Right information is available at the right time

**Safety:** Isolate safe-critical components.

**Availability:** Include redundant components in the architecture.

**Maintainability:** Use fine-grain, self-contained components.

**Microservices Architecture (Services Layer):**

**Fault Isolation:** If one microservice fails, it doesn't bring down the entire system.

**Technology Diversity:** Different microservices can use different technologies based on their specific requirements.

**Flexibility and Maintainability:** Microservices enable independent development, deployment, and updating of each component, making the system more flexible and easier to maintain.

**Complexity Management:** For a large and complex system like SHMS, microservices help break down the system into smaller, manageable parts.

**Agile development:** Microservices architecture is very compatible with agile development methodologies.

**📐 Steps to Draw a System Architecture Diagram**

**1. Identify the System Scope**

* Define the **system boundary**—what is **inside** vs. **outside** the system.
* Identify all **users**, **external systems**, and **major components**.

📌 *Example:* For a Smart Healthcare Management System (SHMS), you may have:

* **Actors:** Patients, Doctors, Receptionists, Pharmacists, Administrators.
* **External Systems:** Payment Gateway, Lab Test System.

**2. Identify Core Components**

Break down the system into **logical modules**:

* **Frontend/Client-side (User Interface)**: What users interact with (e.g., Web or Mobile App).
* **Backend/Server-side (Business Logic)**: Where the core processing happens (e.g., APIs, Microservices).
* **Database Layer**: Where data is stored (e.g., Patient Records, Medical History).
* **External Services**: Systems that the architecture interacts with i.e. third-party APIs (e.g., Payment Gateway, Inventory Tracker).

**3. Define Relationships and Data Flow**

* Identify how **components communicate**:
  + Direct API calls
  + Database queries
  + Message queues (asynchronous communication)
* Use **arrows** to show the flow of **data/information** between components.

📌 *Example Flow:*

1. Patient registers via **Web App** → Data stored in **User Database**.
2. Doctor accesses **Patient Records** from the **Medical Database**.
3. Pharmacy receives **Prescriptions** via the **Pharmacy Module**.

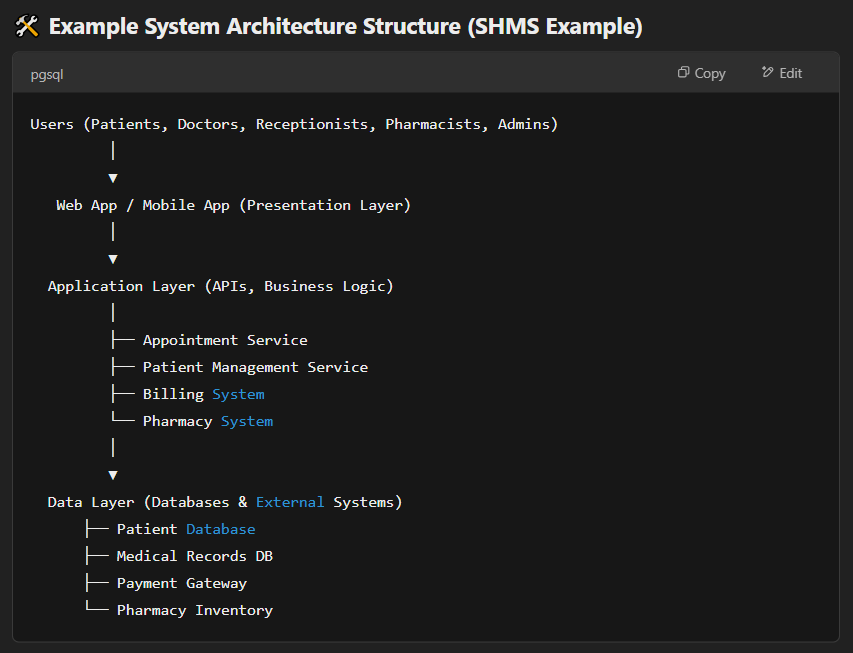
**4. Organize in Layers (3-Tier Model Example)**

1. **Presentation Layer** (Frontend): Handles **User Interfaces** (e.g., Web App, Mobile App).
2. **Application Layer** (Logic): Manages **Business Processes** (e.g., Appointment Scheduling).
3. **Data Layer** (Storage): Includes **Databases** and **File Storage**.

**5. Draw the Diagram**

Use a **diagram tool** (e.g., **Lucidchart**, **Draw.io**, **Microsoft Visio**) and follow this flow:

1. **Top**: Users and external systems (Actors).
2. **Middle**: Core system components (Application Layer).
3. **Bottom**: Databases and external services (Data Layer).



**System Architecture Design For Smart Healthcare Management System (SHMS)**

**1. System Architecture Type: Hybrid (Client-Server + Microservices Approach)**

**Justification:**  
A **Hybrid Architecture** combining **Client-Server** and **Microservices** is the best fit for the **Smart Healthcare Management System (SHMS)** due to the system’s complexity and need for scalability.

* **Client-Server**: Provides a centralized platform where clients (web/mobile interfaces) interact with the backend.
* **Microservices**: Divides core functionalities (e.g., patient management, appointment scheduling, billing, pharmacy) into separate, independently managed services for better scalability, maintenance, and flexibility.

**Why this choice?**  
✅ **Scalability** – Each microservice (e.g., Billing or Appointment) can scale independently.  
✅ **Security** – Access control and data are managed securely through a central **API Gateway**.  
✅ **Flexibility** – Easier to modify and add new features without affecting the entire system.  
✅ **Fault Tolerance** – Failure in one service (e.g., pharmacy) does not bring down the entire system.

**2. Key Components of the System**

**Client-Side (User Interface):**

* **Web Portal (Browser-based)** – For patients, doctors, receptionists, pharmacists, and administrators.
* **Mobile Application** – For patients to book appointments, view records, and receive notifications.
* **Features:**
  + User Authentication (Login/Logout).
  + Patient Registration and Appointment Scheduling.
  + Medical Record Access and Prescription Viewing.
  + Pharmacy Inventory Display and Billing Overview.

**Server-Side (Business Logic):**

**Core Microservices:**

1. **User Management Service** – Manages patient, doctor, and staff information, roles, and permissions.
2. **Appointment Service** – Handles appointment scheduling, approvals, and cancellations.
3. **Medical Record Service** – Stores and retrieves patient history, diagnoses, and lab requests.
4. **Billing Service** – Manages invoices, tracks payments, and integrates with payment gateways.
5. **Pharmacy Service** – Handles prescription validation, inventory management, and restocking alerts.
6. **Audit Log Service** – Logs and tracks user activities to ensure data integrity and compliance.

**Support Services:**

* **Authentication Service** – Manages user login and access control via secure authentication.
* **Notification Service** – Sends appointment reminders and prescription notifications.

**Database Layer:**

1. **Patient Database:** Stores personal information, medical records, and appointment history.
2. **Billing Database:** Tracks payment records, invoices, and transaction logs.
3. **Pharmacy Database:** Maintains prescription data, inventory levels, and restocking alerts.
4. **Audit Database:** Logs every modification for accountability and access tracking.

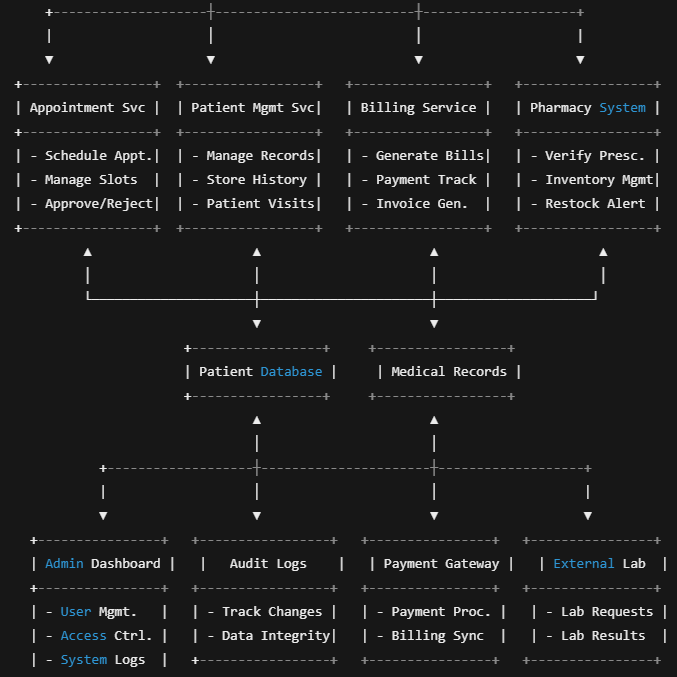
**External Integrations:**

1. **Payment Gateway** – Securely processes patient payments.
2. **External Lab System** – Interfaces with labs for test requests and reporting results.

**3. Data Flow Between Client, Server, and Database**

1. **User Interaction (Client Side to Server Side)**:
   * Patients, doctors, receptionists, and other users interact with the **Web Portal** or **Mobile App**.
   * The user interface sends requests via **HTTP (REST API)** to the **API Gateway**.
2. **Request Handling (Server Side - Microservices Layer):**
   * The **API Gateway** routes the request to the appropriate microservice (e.g., Patient Management for registration).
   * Each microservice processes the request and performs necessary business logic.
3. **Data Storage and Retrieval (Server Side to Database):**
   * Microservices interact with dedicated databases through **SQL queries**.
   * Data is stored securely, ensuring confidentiality and integrity.
4. **External Service Communication (Server to Third-Party):**
   * Payment transactions are sent to the **Payment Gateway**.
   * Lab requests and reports are communicated to the **External Lab System** via API calls.
5. **Response Flow (Database/External to Client):**
   * Processed data is sent back to the client through the **API Gateway**.
   * The **User Interface** updates to reflect changes (e.g., booking confirmation, prescription status).

**Example Data Flow (Appointment Booking Process):**

1. Patient books an appointment via the **Web Portal**.
2. The request is sent to the **API Gateway**.
3. **Appointment Service** validates the request and checks doctor availability.
4. If available, the system stores the appointment in the **Appointment Database**.
5. A screenshot of a computer screen

   AI-generated content may be incorrect.The patient receives a **confirmation message** via the **Notification Service**.

A computer screen shot of a program

AI-generated content may be incorrect.

**Data Flow:**

* Client devices send requests to the API Gateway.
* The API Gateway routes the requests to the relevant microservices.
* Microservices interact with their databases and external systems as needed.
* Microservices return responses to the API Gateway, which forwards them to the client devices.
* The Notification service is called by the microservices, and handles the sending of messages to the end user.
* All data modifications are logged in the Audit Log DB.

**Example of Hospital Management System**:

