HEALTHCARE SYSTEM ARCHITECTURE

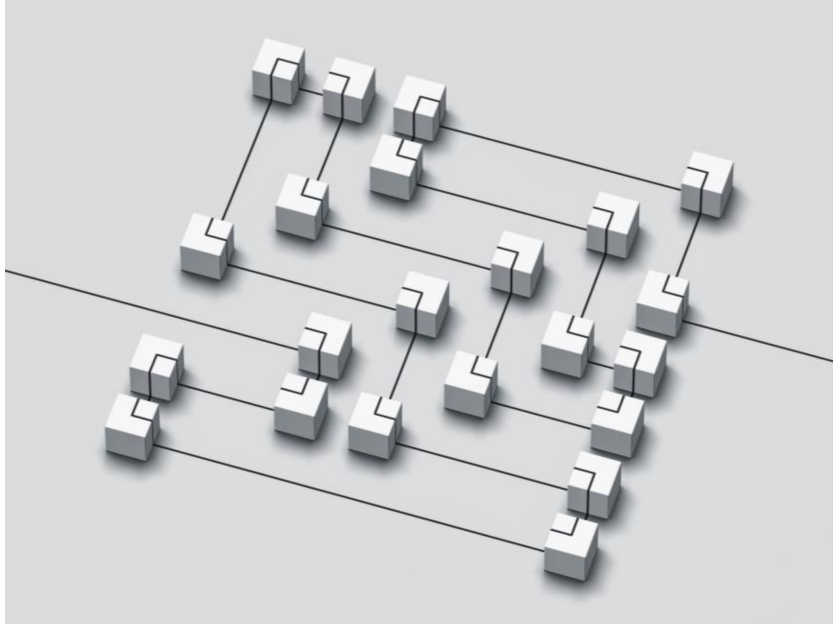
SWENG 837 - Software Design Final Project

Chastidy Joanem

PROJECT INTRODUCTION

- Project Goal: Design a secure, scalable healthcare platform
- Problem Statement:
  - Fragmented healthcare data
  - Delays in prescriptions & appointments
  - Limited interoperability → reduced patient safety

SOLUTION ARCHITECTURE OVERVIEW

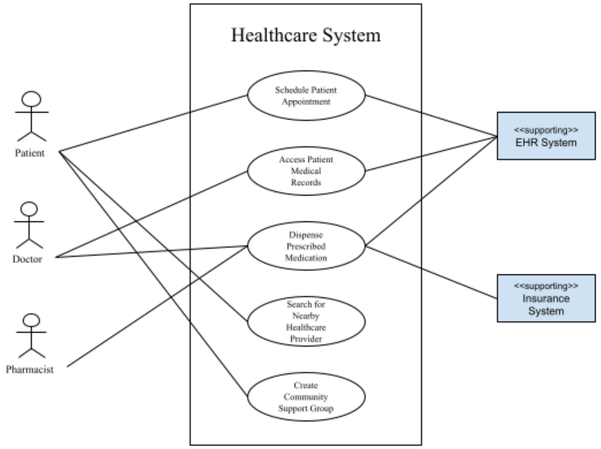


- **High-Level Approach:**
  - Cloud-based, modular healthcare system
- **Core Modules:**
  - Appointment Scheduling
  - Prescription Management
  - Patient Record Access
  - Provider Search
  - Community Support Groups
- **Design Principles:**
  - Modularity, scalability, compliance (HIPAA)
- **Architecture Style:**
  - Microservices + AWS Cloud

PROBLEM STATEMENT

Improve	Improve prescription accuracy & patient safety
Reduce	Reduce administrative overhead
Ensure	Ensure secure, HIPAA-compliant data handling
Enable	Enable multi-region, scalable deployment

SYSTEM USE CASES & ACTORS


Healthcare System
Patient
Doctor
Pharmacist
Schedule Patient Appointment
Access Patient Medical Records
Dispense Prescribed Medication
Search for Nearby Healthcare Provider
Create Community Support Group
<<supporting>> EHR System
<<supporting>> Insurance System

Actors & Interactions:

Patients → schedule appointments, access records, create groups

Doctors → review records, update treatments

Pharmacists → verify prescriptions, dispense medication

Supporting Systems → EHR, Insurance Provider

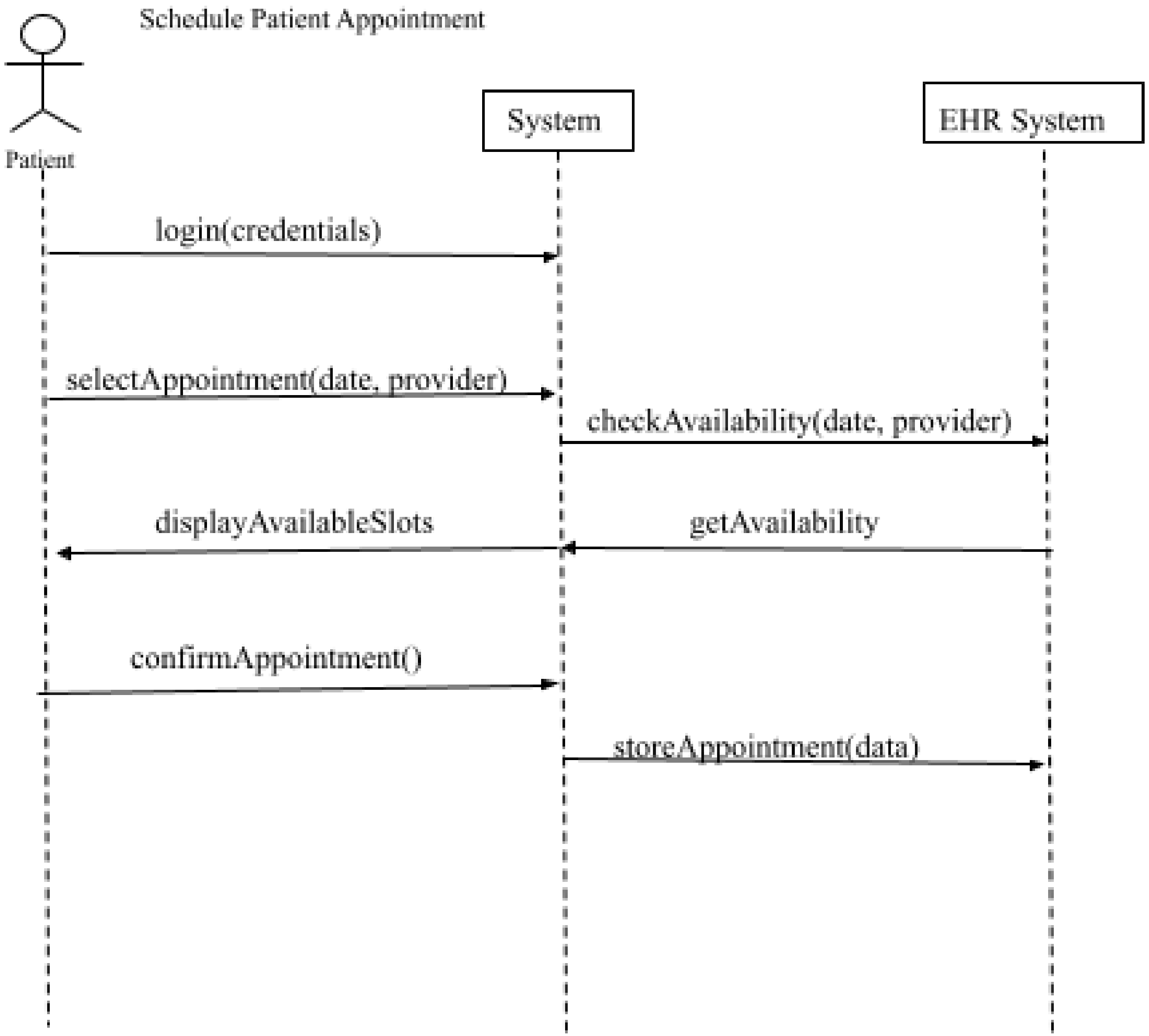
External/Offstage → Gov't Health Agencies, Caregivers, IT Auditors

KEY USE CASES

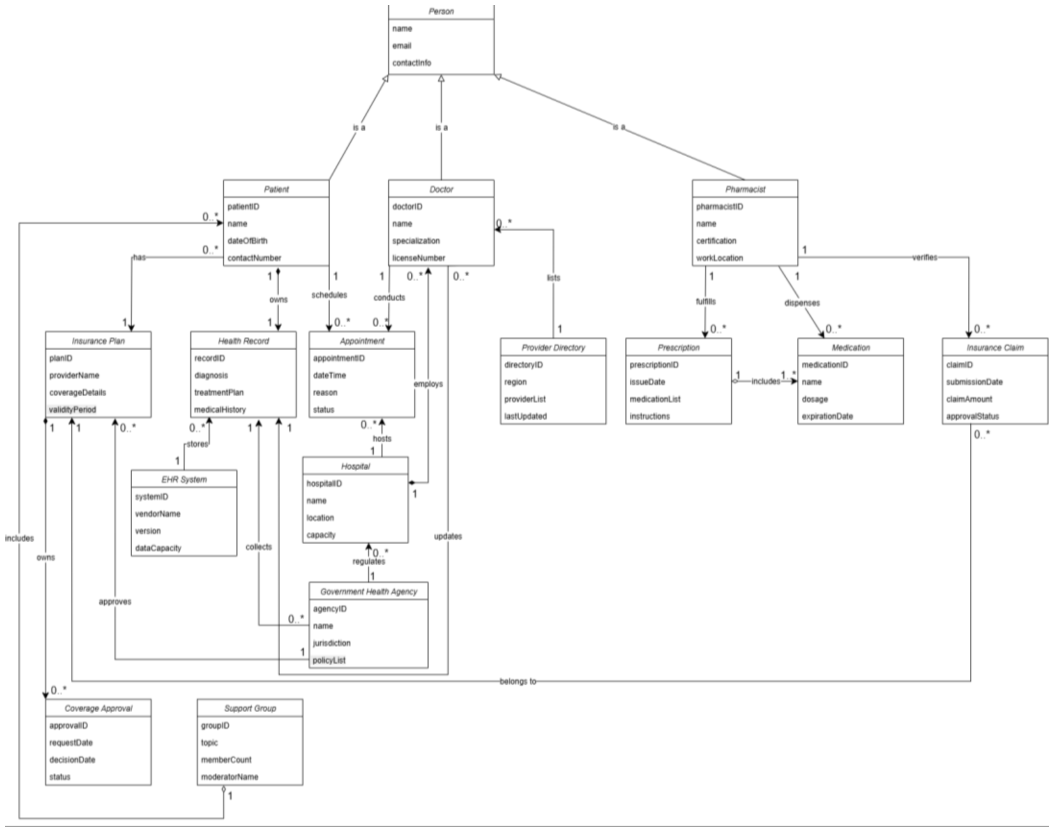
- Schedule Appointment (Patient ↔ EHR)
- Access Records (Doctor ↔ EHR)
- Dispense Medication (Pharmacist ↔ EHR & Insurance)
- Search Providers (Patient filters providers)
- Create Support Group (Patient ↔ Notification Service)

SEQUENCE/ACTIVITY DIAGRAMS

- Appointment Flow: Login → Select Slot → Confirm → Store in EHR
- Records Flow: Doctor Login → Search Patient → Retrieve Records
- Medication Flow: Pharmacist Login → Verify Coverage → Dispense → Log

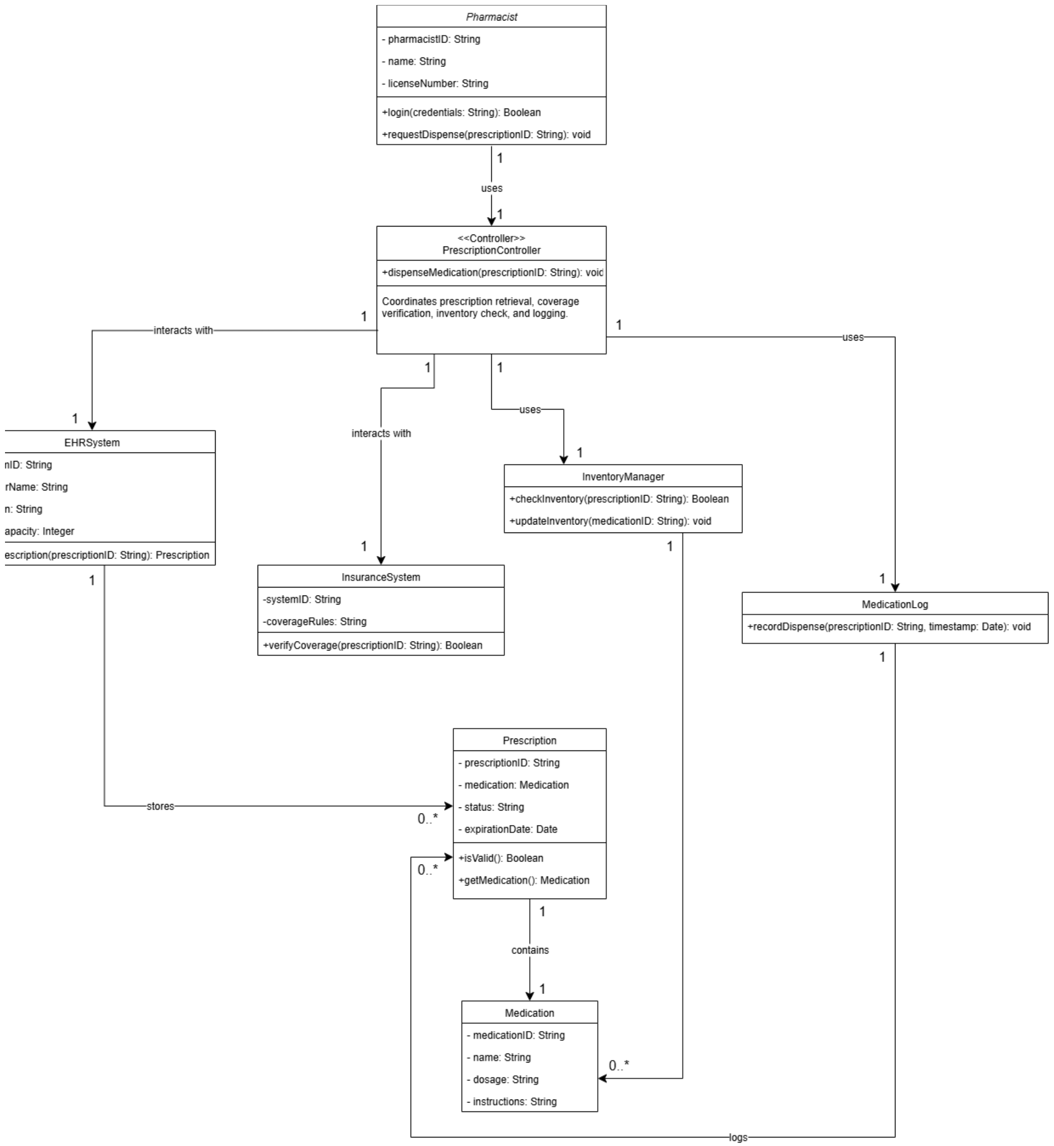


DOMAIN MODEL



- **Core Entities:** Patient, Doctor, Pharmacist, Appointment, Prescription, Health Record, Insurance Plan, Support Group
- **Relationships:**
  - Patient ↔ Appointment ↔ Doctor
  - Patient ↔ Prescription ↔ Pharmacist
  - Patient ↔ Support Groups

CLASS DIAGRAM



Pharmacist
- pharmacistID: String
- name: String
- licenseNumber: String
+login(credentials: String): Boolean
+requestDispense(prescriptionID: String): void

<<Controller>>
PrescriptionController
+dispenseMedication(prescriptionID: String): voic
Coordinates prescription retrieval, coverage verification, inventory check, and logging.

EHRSystem
nID: String
rName: String
n: String
apacity: Integer
escription(prescriptionID: String): Prescription

InsuranceSystem
-systemID: String
-coverageRules: String
+verifyCoverage(prescriptionID: String): Boolean

InventoryManager
+checkInventory(prescriptionID: String): Boolean
+updateInventory(medicationID: String): void

MedicationLog
+recordDispense(prescriptionID: String, timestamp: Date): void

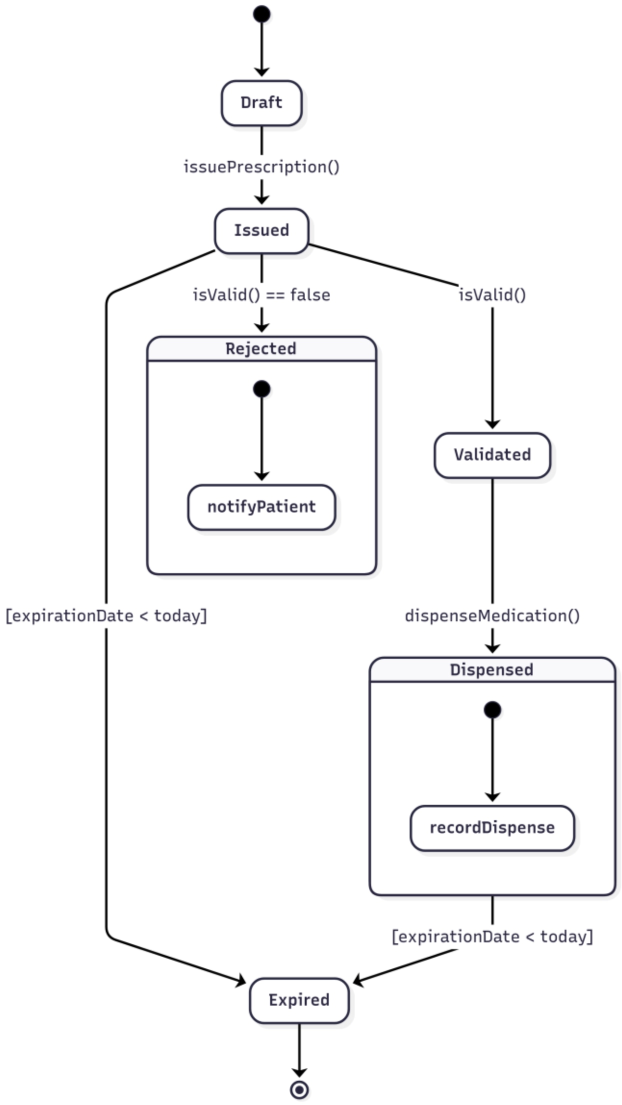
Prescription
- prescriptionID: String
- medication: Medication
- status: String
- expirationDate: Date
+isValid(): Boolean
+getMedication(): Medication

Medication
- medicationID: String
- name: String
- dosage: String
- instructions: String

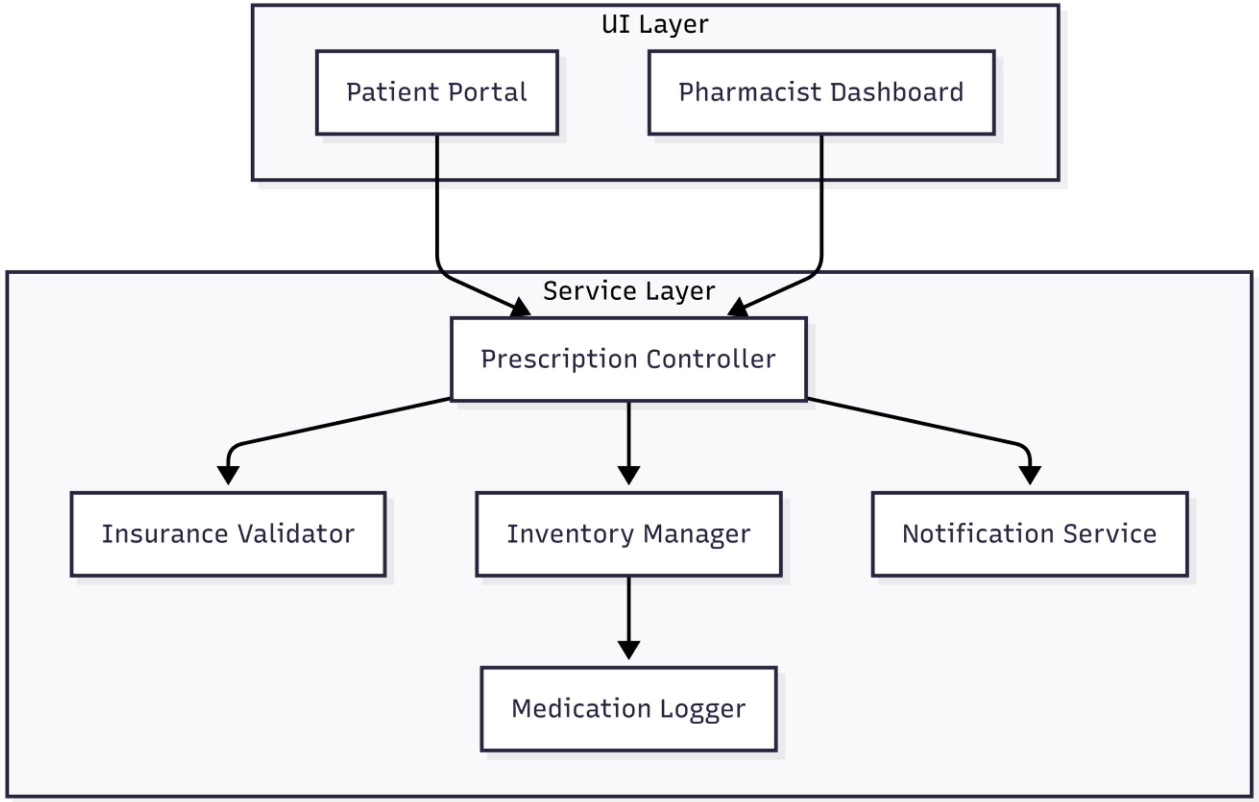
- **PrescriptionController:** Manages dispensing workflow
- **InventoryManager:** Stock updates
- **MedicationLog:** Transaction auditing
- **Person Superclass:** (Doctor, Patient, Pharmacist inherit)
- Design Principles Used: GRASP, Controller, Information Expert

PRESCRIPTION LIFECYCLE

- **States:**
  - Draft → Issued → Validated → Dispensed → Expired/Rejected
- **Transitions:**
  - Triggered by system actions and time conditions

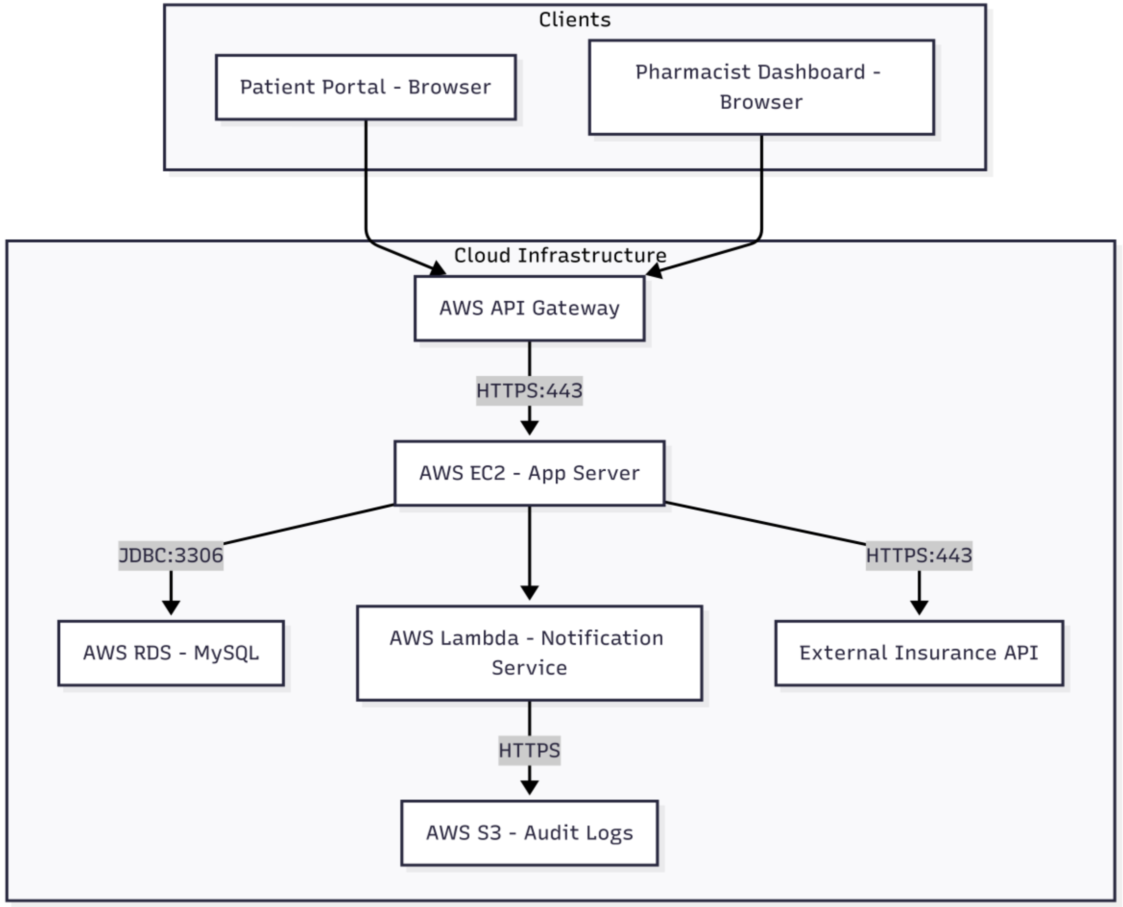


SYSTEM COMPONENTS & INTERFACES



- UI Layer (React.js on AWS S3)
- Service Layer (Spring Boot on EC2)
- DB Layer (MySQL on AWS RDS)
- Notification Service (AWS Lambda + SES)
- Logging (CloudWatch + S3)

AWS DEPLOYMENT STRATEGY



- AWS Infrastructure:
- EC2 for microservices
- RDS for data storage
- Lambda for events/notifications
- CloudFront for scaling & edge caching
- IaC: Docker + Terraform

ARCHITECTURE PATTERN

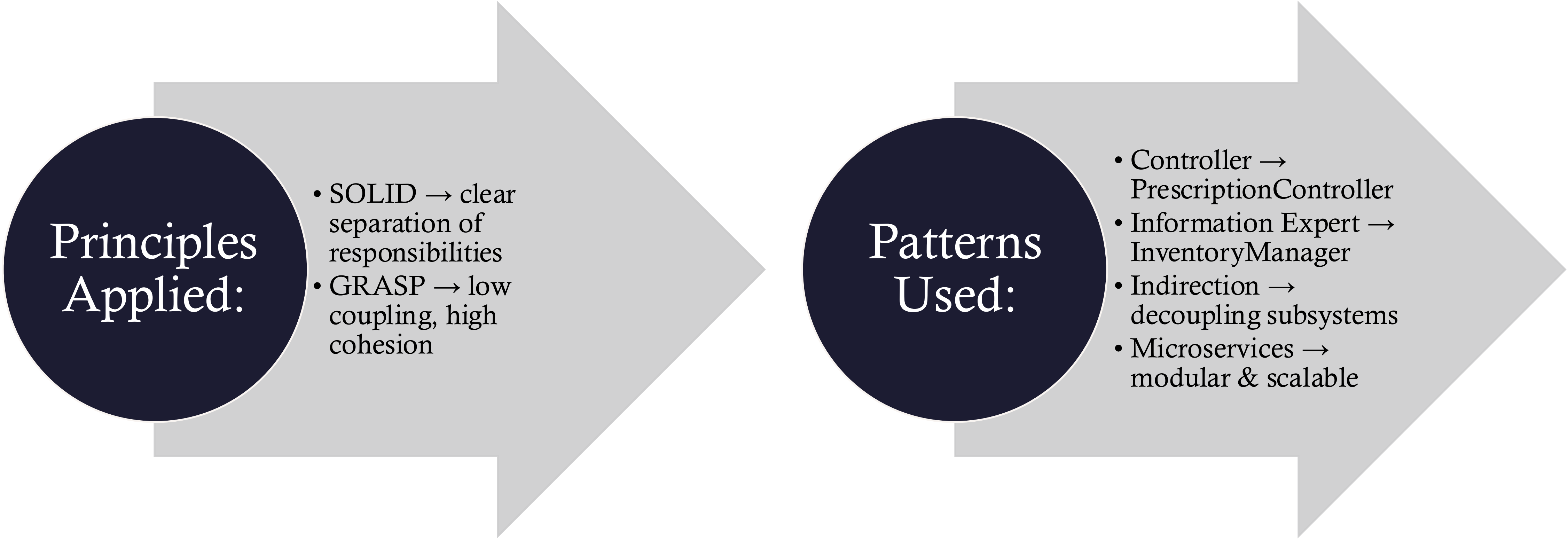
Pattern:

Microservices on Cloud

Why?

Scalability (auto-scaling + multi-region)
Modularity (independent deployment)
Compliance (secure boundaries per service)

APPLIED DESIGN PRINCIPLES & PATTERNS



Principles Applied:

- SOLID → clear separation of responsibilities
- GRASP → low coupling, high cohesion

Patterns Used:

- Controller → PrescriptionController
- Information Expert → InventoryManager
- Indirection → decoupling subsystems
- Microservices → modular & scalable

DESIGN LEARNINGS & INSIGHTS

Key Takeaways:

Designed secure, scalable, modular healthcare platform

Applied UML, design patterns, and architecture principles

Built on cloud-native infrastructure

Next Steps:

Expand to include analytics & AI for policy planning

Extend provider search with real-time geolocation

Apply learnings to future software engineering career

THANK YOU!