Healthcare System

Architecture

Chastidy Joanem

SWENG 837

Final Project

August 15, 2025

[**Introduction 1**](#_brwl9pietvm2)

[**Use Case Diagram 3**](#_fah6x8mbbhfl)

[**Use Cases Analysis 3**](#_uircc2p8evbe)

[**Domain Model - Concept Categories 14**](#_ab29osu3j8iq)

[**Domain Model 16**](#_v055mkaabyu1)

[**System Operation Contracts 17**](#_2g6tm1epbm5l)

[**Class Diagram 23**](#_uwjpgg5rwb63)

[**State Diagram 25**](#_230j79260vo1)

[**Component Diagram 26**](#_apjunv9zc10i)

[**Deployment Diagram 27**](#_lui9bsjrhpsh)

[**Tech Stack & Cloud Service Provider 28**](#_690mvr9ac9x5)

[**Infrastructure as Code (IAC) 29**](#_2basrghftauu)

[Dockerfile (for PrescriptionController) 29](#_c46v3c9s6zfn)

[Terraform Snippet (EC2 + RDS + Lambda) 30](#_1glq3ter3bfo)

[**References 31**](#_kx9n0a3urpu)

# Introduction

Problem Statement

Modern healthcare systems face challenges in securely managing prescriptions, ensuring timely access to care, and facilitating collaboration among patients, providers, and pharmacists. Fragmented data, manual workflows, and limited interoperability between systems often lead to delays, errors, and reduced patient safety. This system aims to address these issues by providing a cloud-based, modular platform that streamlines appointment scheduling, prescription dispensing, medical record access, and community support, all while maintaining compliance with healthcare regulations.

Business Requirements

| **Category** | **Description** |
| --- | --- |
| **Problem to Solve** | Improve prescription accuracy, reduce delays in care, and enhance patient-provider-pharmacist coordination through a secure, scalable digital platform. |
| **Core Functionalities** | - Schedule appointments with providers  - Access and review patient medical records  - Dispense medication based on verified prescriptions  - Search for nearby healthcare providers  - Create and manage support groups for patients |
| **Target Users** | - **Patients**: Need access to records, providers, and peer support  - **Doctors**: Require secure access to patient history for diagnosis and treatment  - **Pharmacists**: Must verify prescriptions and insurance before dispensing  - **System Admins**: Oversee infrastructure, compliance, and data integrity |
| **Business Goals** | - Enhance patient safety and satisfaction  - Reduce administrative overhead for providers  - Ensure HIPAA-compliant data handling  - Enable scalable deployment across regions  - Support analytics and policy planning for health agencies |

Non-Functional Requirements

* Performance
  + **Scalability**: Support up to 100,000 users with auto-scaling infrastructure and multi-region deployment.
  + **Response Time**: API responses under 300ms for core operations (e.g., appointment booking, prescription retrieval).
  + **Throughput**: Handle up to 3,000 transactions per second (TPS) during peak load.
* Security
  + **Authentication**: OAuth2 via AWS Cognito for secure user login.
  + **Authorization**: Role-based access control for patients, doctors, pharmacists, and auditors.
  + **Data Encryption**: TLS for all API traffic; encrypted audit logs stored in AWS S3.
  + **Compliance**: Full alignment with HIPAA and healthcare data standards.
* Maintainability
  + **Modularity**: Microservices architecture with clear separation of UI, service, and data layers.
  + **Documentation**: Inline code comments, API specs, and architecture diagrams stored in GitHub.
  + **Testing Strategy**: Unit tests for controllers, integration tests for service workflows, and automated IaC validation.
* Other Requirements
  + **Availability**: 99.9% uptime via AWS infrastructure and failover strategies.
  + **Interoperability**: Integration with external EHR and insurance APIs.
  + **Auditability**: All access and transactions are logged for compliance review.

# Use Case Diagram



***Use Case Diagram – Healthcare System****; Shows core user interactions and external system support for scheduling, accessing records, dispensing medication, provider search, and community group creation.*

Actors Overview

| **Type** | **Actor** | **Goal Description** |
| --- | --- | --- |
| **Primary** | Patient | Access health records, search for providers, and participate in support communities |
| **Primary** | Doctor | Review patient data and update treatment information |
| **Primary** | Pharmacist | Retrieve prescriptions and provide medication guidance |
| **Supporting** | Electronic Health Record System | Store and secure patient data |
| **Supporting** | Insurance Provider | Validate coverage details and process treatment-related claims |
| **Offstage** | Government Health Agency | Monitor data trends for compliance and policy planning |
| **Offstage** | Caregiver or Family Member | Support patient well-being and care decisions |
| **Offstage** | IT Compliance Auditor | Ensure data handling aligns with security standards and HIPAA regulations |

# Use Cases Analysis

Use Case 1: Schedule Patient Appointment

****

***Sequence Diagram – Schedule Patient Appointment******Use Case:*** *Shows step-by-step interactions between Patient, System, and EHR System to verify availability and confirm appointment booking.*

**Format:** Fully-Dressed

**Goal in Context:** The patient successfully books an appointment with a preferred healthcare provider.

**Primary Actor:** Patient

**Supporting Actor:** EHR System

**Main Success Scenario:**

1. Patient logs into the healthcare portal.
2. Patient selects a preferred date, time, and provider.
3. System checks provider’s availability using the EHR system.
4. System confirms open slot and presents to patient.
5. Patient confirms booking.
6. System stores appointment in EHR.

| **Section** | **Details** |
| --- | --- |
| **Use Case Name** | Schedule Patient Appointment |
| **Scope** | Unified Patient Manager (Appointment Booking Module) |
| **Level** | User-goal |
| **Primary Actor** | Patient |
| **Stakeholders and Interests** | Patient wants quick access to care; Provider needs reliable scheduling |
| **Preconditions** | Patient is registered and has system access |
| **Success Guarantee** | Appointment is logged and confirmation is sent |
| **Main Success Scenario** | 1. Patient logs in 2. Selects provider, date/time  3. System checks availability  4. Patient confirms  5. System stores appointment  6. System sends confirmation |
| **Extensions** | 3a. No availability → suggest alternate times 5a. Network failure → retry or reschedule |
| **Special Requirements** | Real-time availability updates; HIPAA-compliant data storage |

Use Case 2: Access Patient Medical Records



***Sequence Diagram – Access Patient Medical Records Use Case:*** *Shows secure interactions between Doctor, System, and EHR System to retrieve and display patient health data for clinical review.*

**Format:** Fully-Dressed

**Goal in Context:** Doctor reviews current medical data for diagnosis or treatment.

**Primary Actor:** Doctor

**Supporting Actor:** EHR System

**Main Success Scenario:**

1. Doctor logs into the system.
2. Doctor searches for a patient using name or ID.
3. System queries EHR database.
4. System retrieves and displays lab results, diagnoses, and prescription history.
5. Doctor reviews the data.

| **Section** | **Details** |
| --- | --- |
| **Use Case Name** | Access Patient Medical Records |
| **Scope** | Unified Patient Manager (EHR Viewer) |
| **Level** | User-goal |
| **Primary Actor** | Doctor |
| **Stakeholders and Interests** | Doctor needs accurate patient history; Patient wants privacy |
| **Preconditions** | Doctor has login credentials and patient ID |
| **Success Guarantee** | Doctor views patient data securely |
| **Main Success Scenario** | 1. Doctor logs in 2. Searches for patient 3. System retrieves records 4. Displays allergies and history 5. Doctor reviews medical info |
| **Extensions** | 2a. Invalid ID → alert  3a. Record unavailable → error message |
| **Special Requirements** | Secure access control; audit trail |

Use Case 3: Dispense Medication Based on Prescription



***Sequence Diagram – Dispense Medication Use Case:*** *Shows pharmacist-driven interactions with System, EHR, and Insurance systems to retrieve prescriptions, verify coverage, and log medication dispensing.*

**Format:** Partially-Dressed

**Goal in Context:** Pharmacist dispenses medicine after validating prescription and coverage.

**Primary Actor:** Pharmacist

**Supporting Actor:** Insurance System, EHR System

**Main Success Scenario:**

1. Pharmacist logs into system.
2. System displays active prescriptions for patient.
3. System verifies insurance approval.
4. Pharmacist confirms medication inventory.
5. System logs transaction upon dispense.

| **Section** | **Details** |
| --- | --- |
| **Use Case Name** | Dispense Medication Based on Prescription |
| **Scope** | Pharmacy System |
| **Level** | User-goal |
| **Primary Actor** | Pharmacist |
| **Stakeholders and Interests** | Pharmacist needs verified orders; Patient expects correct medication |
| **Preconditions** | Valid prescription is stored in EHR |
| **Success Guarantee** | Medication is dispensed and logged |
| **Main Success Scenario** | 1. Pharmacist logs in 2. Retrieves prescription 3. System verifies insurance 4. Confirms inventory 5. Dispenses medication 6. Logs transaction |
| **Extensions** | 3a. Insurance denial → patient notified  4a. Out of stock → alert or request transfer |
| **Special Requirements** | Barcode verification; medication history tracking |

Use Case 4: Search for Nearby Healthcare Provider



***Sequence Diagram – Search for Nearby Healthcare Provider Use Case:*** *Shows patient-driven interaction with the System to filter providers by location, specialty, and insurance compatibility.*

**Format:** Partially-Dressed

**Goal in Context:** Patient finds a suitable provider based on location, specialty, and insurance.

**Primary Actor:** Patient

**Main Success Scenario:**

1. Patient accesses Qalb+ search page.
2. Patient enters desired specialty and insurance details.
3. System displays a list of nearby eligible providers.
4. Patient selects one for booking.

| **Section** | **Details** |
| --- | --- |
| **Use Case Name** | Search for Nearby Healthcare Provider |
| **Scope** | Qalb+ Provider Search Engine |
| **Level** | User-goal |
| **Primary Actor** | Patient |
| **Stakeholders and Interests** | Patient wants relevant options; System must show insurance-compatible results |
| **Preconditions** | Patient has system access and input filters |
| **Success Guarantee** | List of providers is displayed based on input |
| **Main Success Scenario** | 1. Patient accesses search tool 2. Inputs specialty and location 3. Filters by insurance accepted 4. System displays nearby providers |
| **Extensions** | 2a. No providers match → suggest broader filters |
| **Special Requirements** | Map integration; insurance database sync |

Use Case 5: Create Community Support Group



***Sequence Diagram – Create Community Support Group Use Case:*** *Shows patient-initiated workflow with System and Notification Service to validate group details, confirm creation, and notify relevant users.*

**Format:** Partially-Dressed

**Goal in Context:** Patient creates a new group on KLIK to connect with others facing similar conditions.

**Primary Actor:** Patient

**Main Success Scenario:**

1. Patient logs into KLIK portal.
2. Patient navigates to Support Groups section.
3. Patient clicks “Create Group” and adds description and condition tags.
4. System publishes the new group and notifies similar users.

| **Section** | **Details** |
| --- | --- |
| **Use Case Name** | Create Community Support Group |
| **Scope** | KLIK Support Portal |
| **Level** | User-goal |
| **Primary Actor** | Patient |
| **Stakeholders and Interests** | Patient wants peer support; System ensures moderation |
| **Preconditions** | Patient is logged in |
| **Success Guarantee** | Support group is published and discoverable |
| **Main Success Scenario** | 1. Patient logs in 2. Clicks "Create Group" 3. Enters group name and tags 4. System validates content 5. Publishes group 6. Sends notifications |
| **Extensions** | 4a. Inappropriate content → block creation |
| **Special Requirements** | Moderation tools; tagging system |

# Domain Model - Concept Categories

| **Conceptual Class Category** | **Examples** |
| --- | --- |
| **Physical or Tangible Objects** | Appointment, Medication, Prescription, Health Record, Support Group |
| **Specifications, Designs, Descriptions of Things** | Appointment Details, Prescription Info, Patient Profile, Group Tags |
| **Places** | Clinic Location, Pharmacy Address, Provider Facility |
| **Transactions** | Appointment Booking, Medication Dispense, Group Creation |
| **Transaction Line Items** | Medication Item, Insurance Approval |
| **Roles of People** | Patient, Doctor, Pharmacist, Caregiver, IT Auditor, Insurance Agent |
| **Containers of Other Things** | EHR System, Support Portal, Insurance System |
| **Things in a Container** | Patient Medical Records, Prescription History, Support Group Posts |
| **Other Computers/Systems (external)** | EHR System, Insurance System, Government Health Agency |
| **Abstract Noun Concepts** | Medical Condition, Support Need, Access Control, Availability Status |
| **Organizations** | Hospital, Government Health Agency, Insurance Provider |
| **Events** | Log In, Create Group, Confirm Appointment, Dispense Medication |
| **Processes** | Appointment Scheduling, Prescription Verification, Support Group Moderation |
| **Rules and Policies** | HIPAA Compliance, Coverage Rules, Moderation Guidelines |
| **Catalogs** | Provider Directory, Medication Catalog |
| **Records of Finance, Work, Contracts, Legal** | Insurance Claim, Audit Log, Medical Record Entry |
| **Financial Instruments and Services** | Insurance Plan, Coverage Approval |
| **Manuals, Documents, Reference Papers** | Privacy Notice, User Guidelines, Consent Form |

Class Refinement

| **Good Classes (Retained)** | **Bad Classes (Pruned)** |
| --- | --- |
| Appointment | Log In |
| Medication | Consent Form |
| Prescription | Privacy Notice |
| Health Record (merged with Patient Profile) | User Guidelines |
| Patient | Notification Service |
| Doctor | Audit Log |
| Pharmacist | Patient Profile |
| Support Group (with optional Support Group Post) | Prescription Info (already covered) |
| EHR System | Moderation Guidelines (rule, not class) |
| Provider Directory | Patient Profile (merged with Health Record) |
| Insurance Claim | Group Tags (implementation-specific) |
| Insurance Plan | Create Group |
| Coverage Approval | Confirm Appointment |
| Government Health Agency | Dispense Medication |
| Hospital |  |

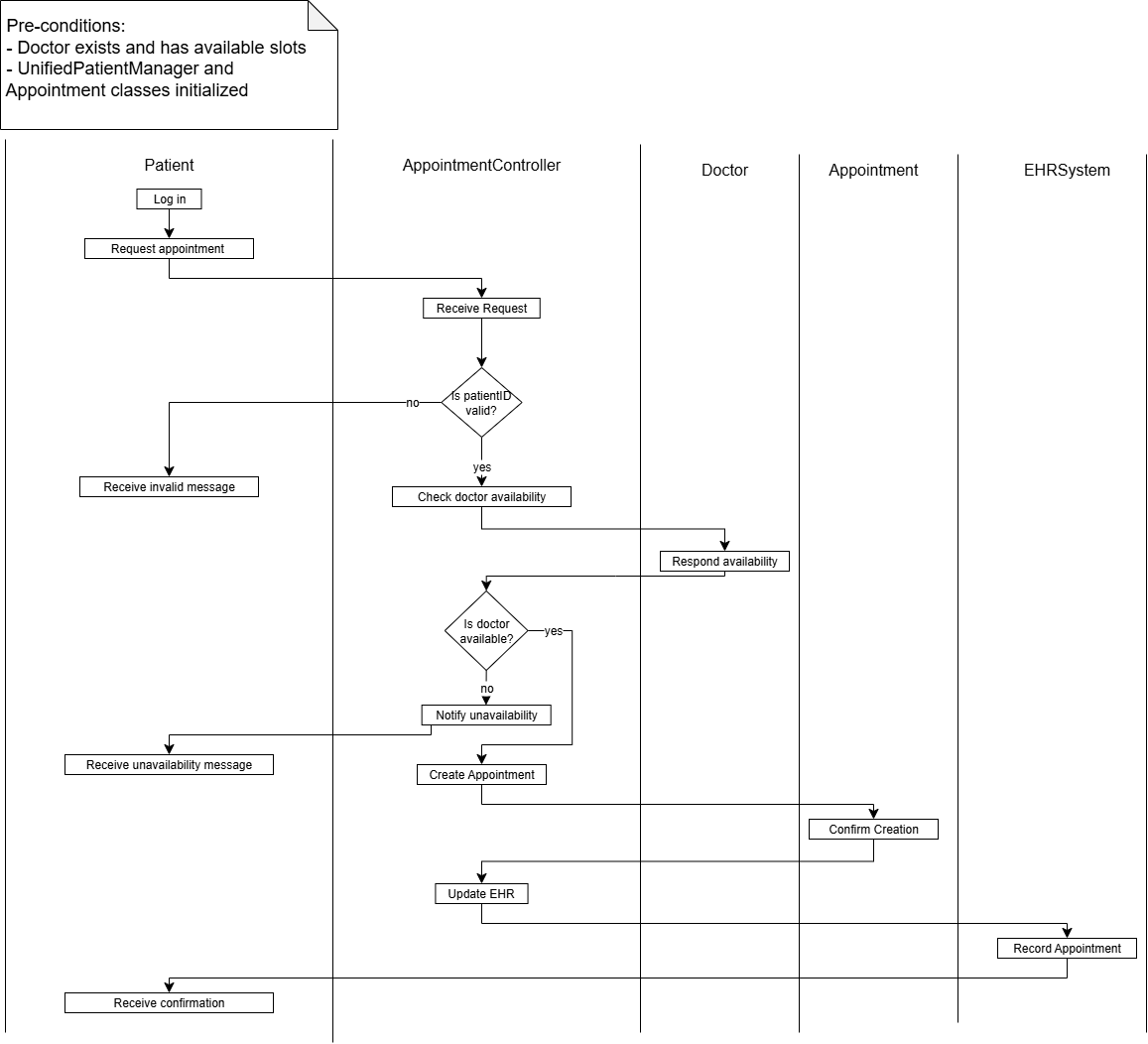
# Domain Model

***Domain Model – Healthcare System*** *defines core entities and their relationships across clinical, administrative, and regulatory domains, including patients, providers, prescriptions, claims, and health records.*

# System Operation Contracts

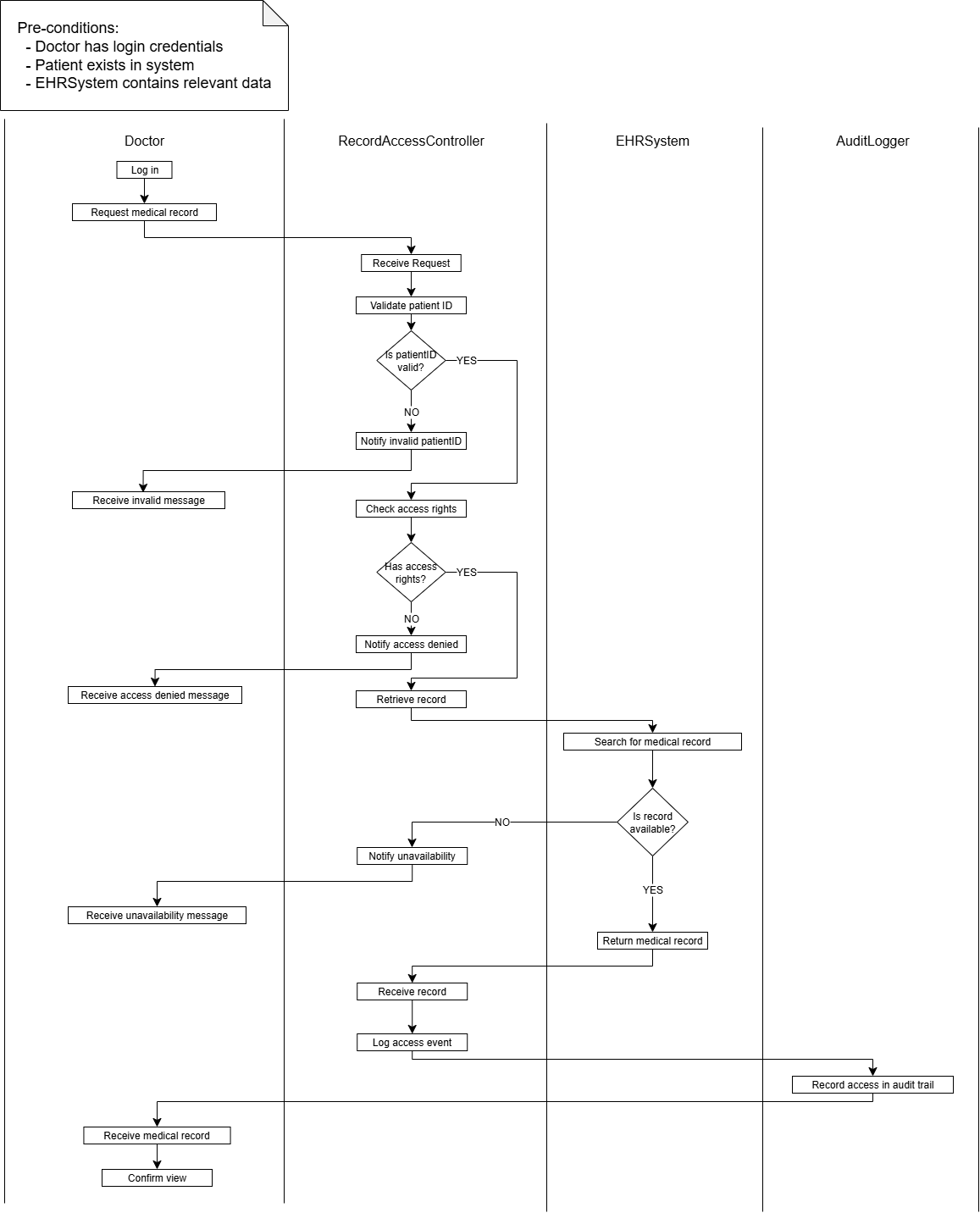
1. scheduleAppointment(patientID, doctorID, dateTime)

* **Responsibilities:** Book an appointment between a registered Patient and a Doctor at a requested time.
* **Type:** System Operation - Controller
* **Cross References:**
  + Use Case: Schedule Patient Appointment
  + SSD: scheduleAppointment
* **Exceptions:**
  + Doctor unavailable
  + Invalid patientID
* **Output:**
  + Confirmation message with appointment details
  + Updated EHR records
* **Pre-conditions:**
  + Patient is logged in
  + Doctor exists and has available slots
  + UnifiedPatientManager and Appointment classes initialized
* **Post-conditions**
  + A new Appointment object is created
  + Appointment.dateTime set to requested slot
  + Associated with Patient and Doctor
  + EHRSystem updated with appointment data

***Swimlane Diagram – Appointment Scheduling*** *models the scheduleAppointment operation across Patient, Controller, Doctor, Appointment, and EHRSystem, showing validation, availability checks, and EHR updates.*

2. retrieveMedicalRecords(doctorID, patientID)

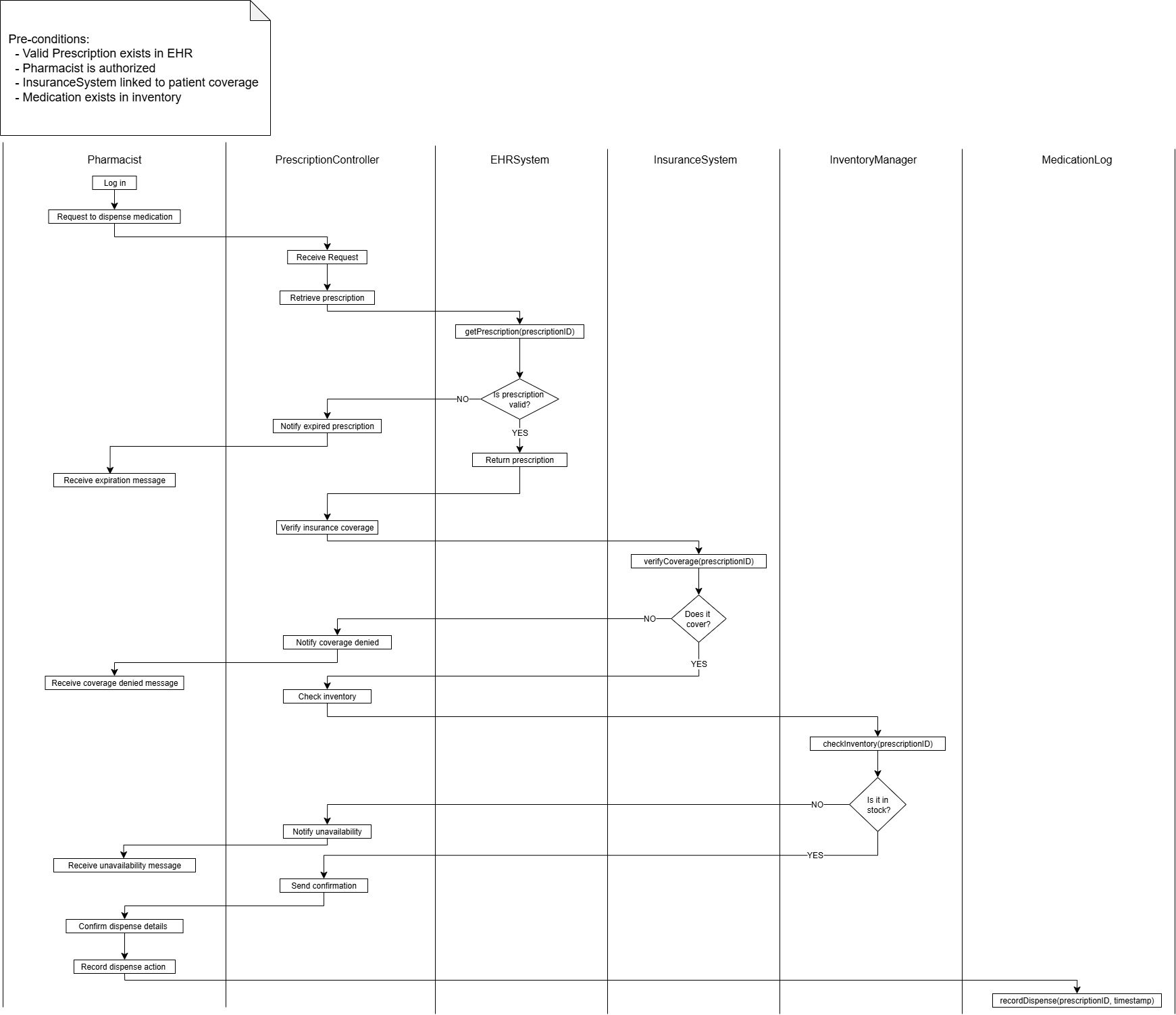
* **Responsibilities:** Allow Doctor to securely access a Patient’s health records for clinical review.
* **Type:** System Operation - Controller
* **Cross References:**
  + Use Case: Access Patient Medical Records
  + SSD: retrieveMedicalRecords
* **Exceptions:**
  + Invalid patient ID
  + EHR record unavailable
  + Access denied
* **Output:**
  + Detailed view of patient’s medical history
* **Pre-conditions:**
  + Doctor has login credentials
  + Patient exists in system
  + EHRSystem contains relevant data
* **Post-conditions:**
  + Doctor receives current diagnosis, prescription, and lab data
  + Access logged in audit trail



***Swimlane Diagram – Retrieve Medical Records*** *models the retrieveMedicalRecords operation across Doctor, Controller, EHRSystem, and AuditLogger, showing validation, access checks, data retrieval, and logging.*

3. dispenseMedication(prescriptionID, pharmacistID)

* **Responsibilities:** Verify prescription and insurance, then dispense appropriate medication.
* **Type:** System Operation - Controller
* **Cross References:**
  + Use Case: Dispense Medication Based on Prescription
  + SSD: dispenseMedication
* **Exceptions:**
  + Insurance denial
  + Medication out of stock
  + Prescription expired
* **Output:**
  + Medication dispensed
  + Logged transaction
* **Pre-conditions:**
  + Valid Prescription exists in EHR
  + Pharmacist is authorized
  + InsuranceSystem linked to patient coverage
  + Medication exists in inventory
* **Post-conditions:**
  + Prescription marked as dispensed
  + MedicationLog updated
  + Inventory adjusted



***Swimlane Diagram – Dispense Medication*** *models the dispenseMedication operation across six entities, showing prescription validation, insurance and inventory checks, and final logging.*

# Class Diagram

***Class Diagram – Prescription Dispensing Use Case*** *shows core entities and controller logic for verifying coverage, retrieving prescriptions, and updating inventory and logs during medication dispensing.*

RATIONALE

**1. Eliminated Classes and Associations**

* To streamline the model, some domain classes were removed:
  + InsuranceClaim, CoverageApproval, and AuditLogger were pruned.
  + Their responsibilities were either absorbed into broader system components like InsuranceSystem or merged into more focused classes like MedicationLog.

**2. Pure Fabrications Introduced**

* New classes were added to handle system logic:
  + PrescriptionController manages the dispensing workflow.
  + InventoryManager checks and updates stock.
  + MedicationLog records transactions for auditing.
* These are not part of the original domain but help keep the design modular and maintainable.

**3. Indirection Applied**

* The PrescriptionController acts as an intermediary between subsystems. This avoids tight coupling and keeps responsibilities clearly separated.

**4. Polymorphism Potential**

* Polymorphism isn’t used yet, but the Prescription class could be extended later. For example, it could support different types like digital or paper prescriptions.

**5. Generalization Used**

* The domain model includes a Person superclass. Pharmacist inherits from it, promoting reuse and reducing duplication across roles.

**6. Design Principles Followed**

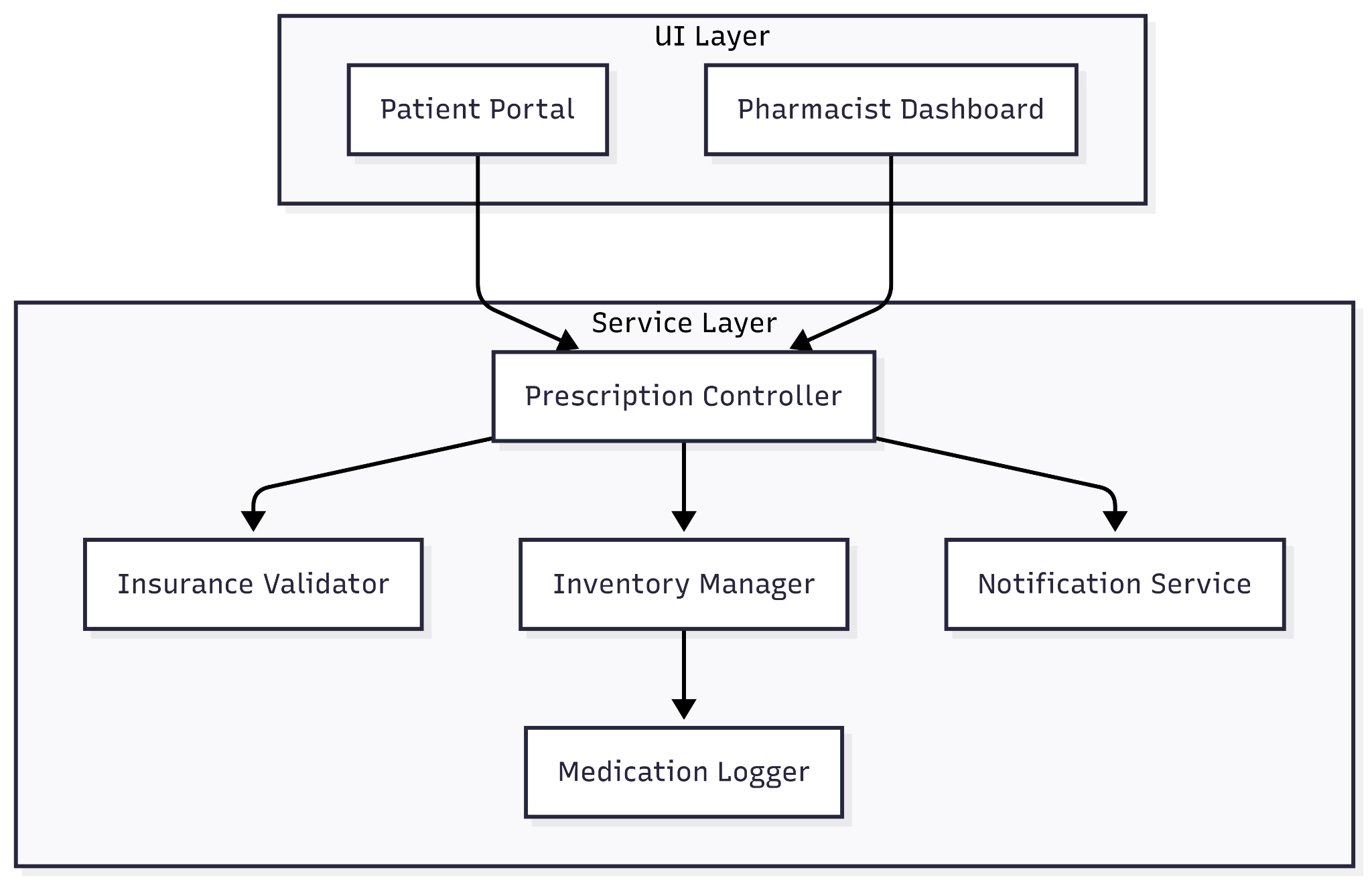
* The design follows GRASP principles:
  + Low coupling
  + High cohesion
  + Controller and Information Expert patterns
* This ensures the system is clear, modular, and easy to maintain.

# State Diagram

# 

***State Diagram – Prescription Lifecycle*** *models the progression of a prescription from creation to expiration, including validation, dispensing, and rejection paths. Transitions are triggered by system methods and time-based conditions, with entry actions for logging and notifications.*

# Component Diagram



***Component Diagram – Prescription Dispensing Use Case*** *shows modular components including UI, service, and integration layers for handling prescriptions.*

# Deployment Diagram

***Deployment Diagram – AWS Infrastructure Overview*** *illustrates cloud-based deployment of backend services, database, and notification system.*

# Tech Stack & Cloud Service Provider

| **Layer** | **Technology** | **Cloud Service** |
| --- | --- | --- |
| UI Layer | React.js | Hosted on AWS S3 |
| Service Layer | Kotlin + Spring Boot | AWS EC2 |
| DB Layer | MySQL | AWS RDS |
| Notification | AWS Lambda | AWS Lambda + SES |
| Logging | S3 + CloudWatch | AWS S3 |

Rationale for Architecture

* Modular components allow independent deployment and scaling.
* AWS services provide elasticity, security, and global availability.
* Clear separation of UI, service, and data layers improves maintainability.
* Event-driven notifications reduce latency and improve responsiveness.

Scaling Strategy

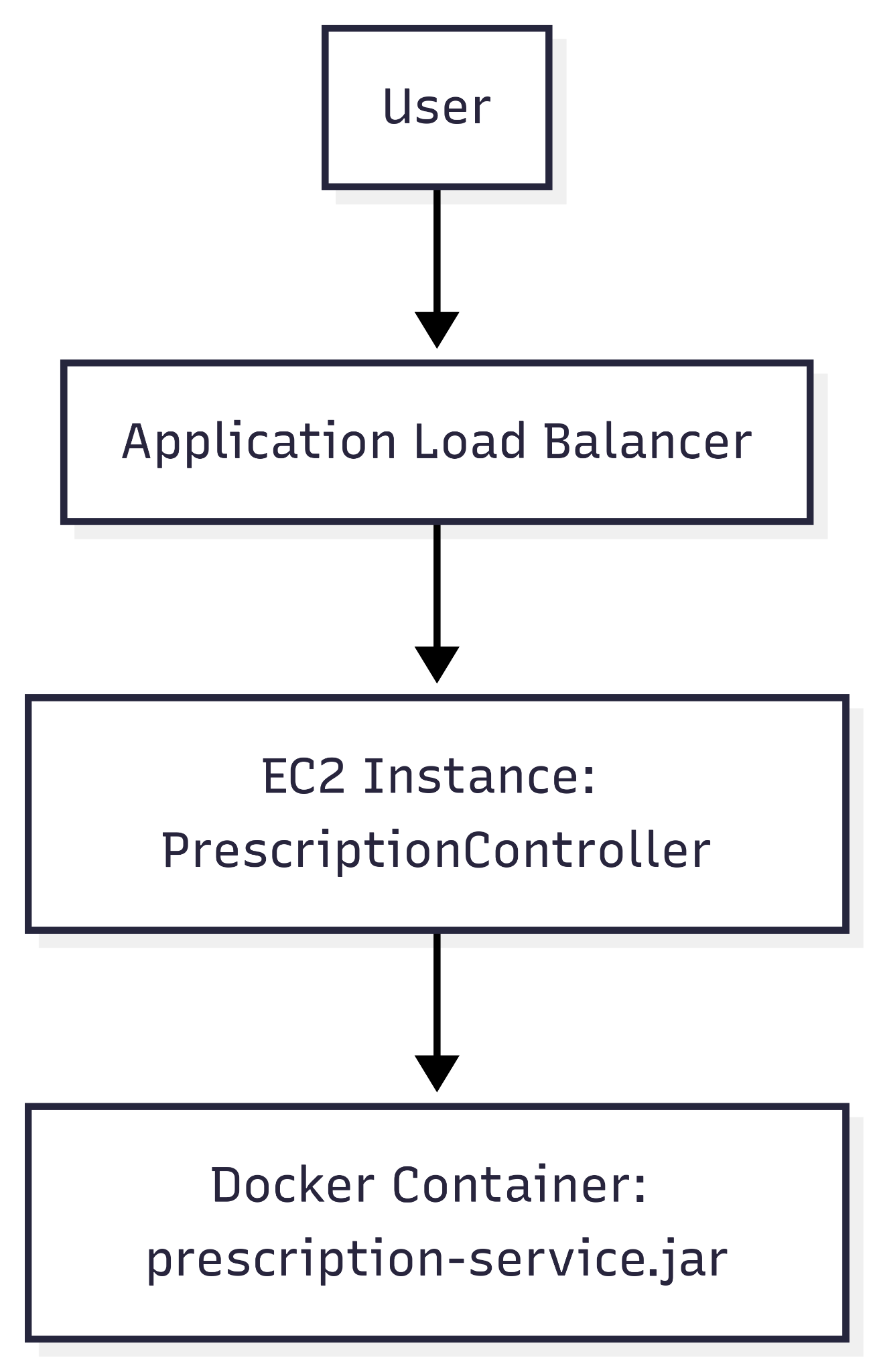
| **Load Scenario** | **Architecture Adjustments** |
| --- | --- |
| 100 users / 100 TPS | Single EC2 instance, basic RDS setup |
| 1,000 users / 1,000 TPS | Auto-scaling EC2, Redis cache for inventory reads |
| 10,000 users / 3,000 TPS | Microservices split, CDN for UI, async queues (e.g., SQS), load-balanced API Gateway |
| 100,000 users | Multi-region deployment, edge caching (CloudFront), event-driven architecture with Kafka |

Security Architecture

* HTTPS for all APIs via API Gateway
* TLS certificates on EC2 and RDS
* IAM roles for Lambda and EC2 access
* Private subnets for RDS and internal services
* OAuth2 authentication via AWS Cognito
* Encrypted audit logs stored in S3

# Infrastructure as Code (IAC)

### **Dockerfile (for PrescriptionController)**



**FROM** openjdk:17-jdk-slim

**WORKDIR** /app

**COPY** target/prescription-service.jar .

**EXPOSE** 8080

**CMD** ["java", "-jar", "prescription-service.jar"]

***Deployment Diagram – Dockerized PrescriptionController on EC2*** *depicts containerized microservice hosted on EC2 with load balancer and user access.*

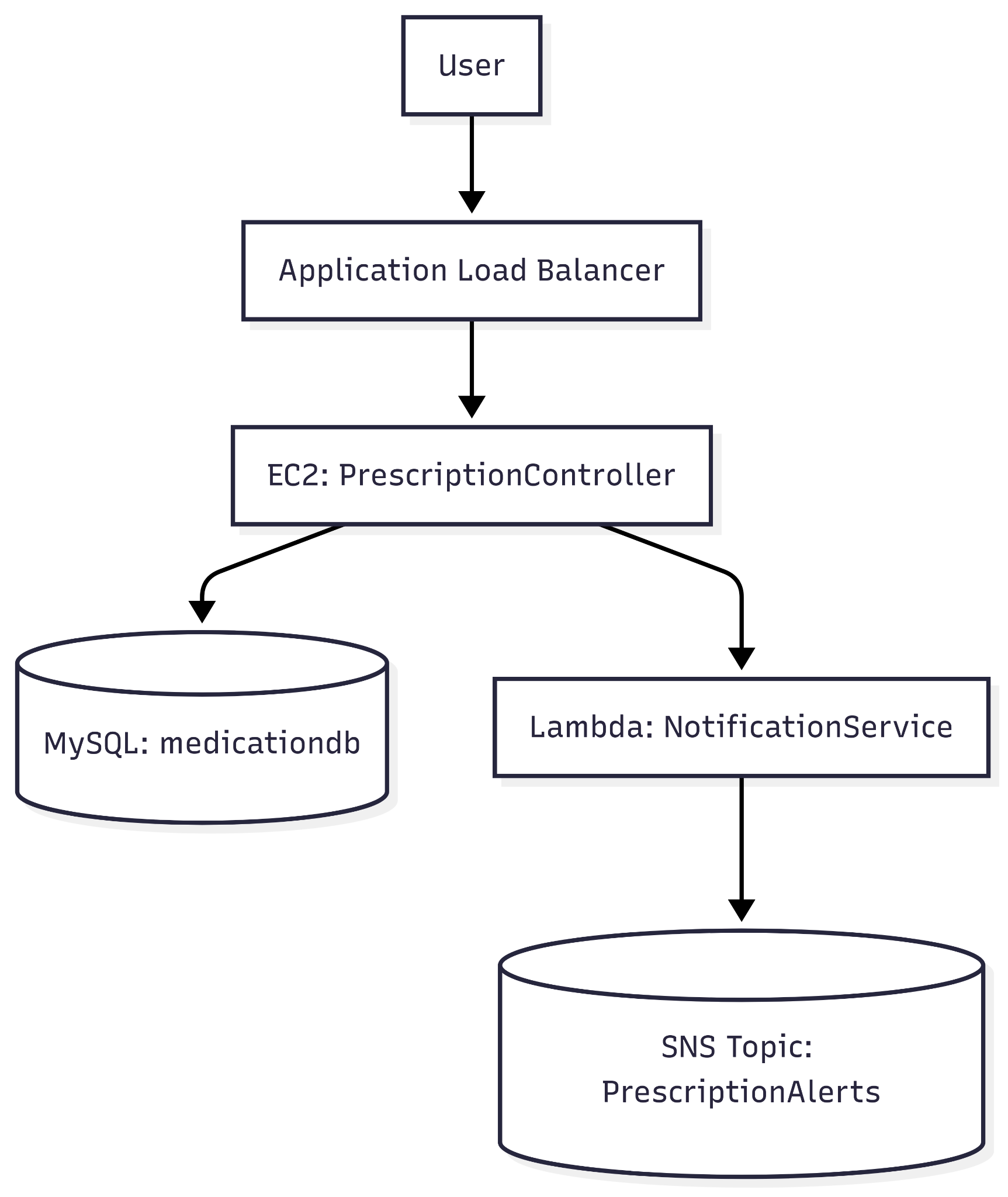
This Dockerfile packages the PrescriptionController microservice into a lightweight container:

* **Base Image:** Uses openjdk:17-jdk-slim for Java 17 support with minimal overhead
* **Working Directory:** Sets /app as the container’s working directory
* **Copy:** Adds the compiled JAR file into the container
* **Expose:** Opens port 8080 for incoming HTTP traffic
* **Command:** Runs the service using java -jar

This container can be deployed to EC2, ECS, or Kubernetes, and is ideal for modular microservices.

### 

### **Terraform Snippet (EC2 + RDS + Lambda)**



resource "aws\_instance" "app\_server" {

ami = "ami-0abcdef1234567890"

instance\_type = "t3.micro"

tags = {

Name = "PrescriptionAppServer"

}

}

resource "aws\_db\_instance" "rds" {

allocated\_storage = 20

engine = "mysql"

instance\_class = "db.t3.micro"

name = "medicationdb"

username = "admin"

password = "securepassword"

skip\_final\_snapshot = true

}

resource "aws\_lambda\_function" "notify" {

function\_name = "NotificationService"

handler = "com.example.NotifyHandler"

runtime = "java11"

role = aws\_iam\_role.lambda\_exec.arn

filename = "build/libs/notification-service.zip"

}

***Deployment Diagram – Terraform-Provisioned EC2, RDS, and Lambda*** *shows infrastructure provisioned via IaC for backend service, database, and event-driven notifications.*

This Terraform code provisions three key components:

1. **EC2 Instance (app\_server)**
   * Hosts the PrescriptionController container
   * Uses a t3.micro instance for cost-effective compute
   * Tagged for easy identification
2. **RDS Instance (rds)**
   * MySQL database named medicationdb
   * Stores prescription and medication data
   * Uses db.t3.micro with 20 GB storage
   * Credentials are hardcoded here (should be secured in production)
3. **Lambda Function (notify)**
   * Java-based notification handler
   * Triggered by events (e.g., new prescription added)
   * Uses a pre-built ZIP file for deployment
   * IAM role (lambda\_exec) must grant execution and logging permissions

# References

Object-Oriented Analysis & Design (OOAD)

* **Larman, C.** *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson, 2005.
* **Jacobson, Booch, Rumbaugh.** *The Unified Software Development Process*. Addison-Wesley, 1999.
* [GeeksforGeeks – Patterns and Frameworks in OOAD](https://www.geeksforgeeks.org/system-design/patterns-and-frameworks-in-ooad/)
* [Design Principles – University of Houston](https://www2.cs.uh.edu/~svenkat/SoftwareDesign/slides/ObjectOrientedDesignPrinciples.pdf)

UML and Domain Modeling

* [GeeksforGeeks – What is Domain Class in UML?](https://www.geeksforgeeks.org/system-design/what-is-domain-class-in-uml/)
* [Cerritos College – Chapter 4 Domain Modeling](https://www.cerritos.edu/dwhitney/SitePages/CIS201/Lectures/IM-7ed-Chapter04.pdf)
* **Fowler, M.** *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003.

Design Patterns

* **Gamma, Helm, Johnson, Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
* [Design Patterns in OOAD – MPP Indico](https://indico.mpp.mpg.de/event/2835/contributions/5798/attachments/4721/5217/patterns.pdf)
* [GeeksforGeeks – OOAD Design Patterns](https://www.geeksforgeeks.org/system-design/patterns-and-frameworks-in-ooad/)

Cloud Deployment & Microservices Architecture

* [AWS Microservices Architecture](https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html)
* [Azure Architecture Center – Microservices Style](https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices)
* [Azure Deployment Architecture](https://learn.microsoft.com/en-us/azure/architecture/web-apps/app-service-environment/architectures/ase-standard-deployment)
* [Google Cloud – Microservices Architecture](http://cloud.google.com/blog/topics/developers-practitioners/microservices-architecture-google-cloud)
* [Visual Paradigm – Component Diagrams](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/)
* [Visual Paradigm – Deployment Diagrams](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/)

System Design & Case Studies

* [System Design Primer – GitHub](http://github.com/donnemartin/system-design-primer)
* [Design Gurus Blog – System Design Guide](https://www.designgurus.io/blog/system-design-primer-the-ultimate-guide)
* [Grokking the OO Design Interview – GitHub](https://github.com/tssovi/grokking-the-object-oriented-design-interview/tree/master/object-oriented-design-case-studies)