In software architecture and system design interviews, **High-Level Design (HLD)** and **Low-Level Design (LLD)** are two crucial stages. Each covers different depths of design decisions. Here's a structured breakdown of the **topics to be covered** in both:

### ✅ **High-Level Design (HLD) Topics**

Focus: **System architecture, components, communication, scalability, and technology stack**

#### 1. **Requirement Analysis**

* Functional and non-functional requirements
* Use case identification
* SLAs (availability, scalability, latency)

#### 2. **System Architecture**

* Monolithic vs Microservices
* Layered architecture (e.g., presentation, business, data)
* Cloud-native vs on-premise
* Hybrid architectures

#### 3. **Component Identification**

* Services/modules/subsystems
* Bounded Contexts (in DDD)
* API Gateway, Identity Service, Notification Service, etc.

#### 4. **Communication**

* REST, gRPC, GraphQL
* Sync vs Async
* Messaging queues (e.g., Azure Service Bus, Kafka)

#### 5. **Technology Stack**

* Programming languages
* Databases (SQL vs NoSQL)
* Frontend framework
* CI/CD tools
* DevOps tools (Docker, Kubernetes)

#### 6. **Deployment Architecture**

* Cloud architecture (Azure, AWS, GCP)
* Load balancers, App Services, Containers, Function Apps
* Multi-region/multi-AZ setups

#### 7. **Data Management Strategy**

* Caching (Redis, CDN)
* Database sharding, replication
* Data warehousing

#### 8. **Scalability & Availability**

* Horizontal vs Vertical scaling
* Fault-tolerance mechanisms
* Auto-scaling

#### 9. **Security**

* Authentication/Authorization (OAuth2, Azure AD B2C)
* API security
* Network security (firewalls, VNETs)

#### 10. **Observability**

* Logging, Monitoring, Tracing
* Alerting systems
* Dashboards (e.g., Azure Monitor, App Insights, Grafana)

#### 11. **Trade-off Analysis**

* CAP Theorem
* Consistency vs Availability vs Partition Tolerance
* Cost vs Performance

#### 12. **UML Diagrams (HLD-level)**

* Component Diagram
* Deployment Diagram
* C4 Model (Context + Container)

### ✅ **Low-Level Design (LLD) Topics**

Focus: **Class design, object interactions, design patterns, and coding structure**

#### 1. **Class Diagrams**

* Class structure, responsibilities
* Attributes, methods
* Associations, inheritance, composition

#### 2. **Object-Oriented Design Principles**

* SOLID principles
* DRY, KISS, YAGNI
* Encapsulation, Abstraction, Inheritance, Polymorphism

#### 3. **Design Patterns (with real examples)**

* **Creational:** Singleton, Factory, Abstract Factory, Builder
* **Structural:** Adapter, Decorator, Composite, Proxy
* **Behavioral:** Strategy, Observer, Command, State, Chain of Responsibility

#### 4. **Database Design**

* Table structure
* Indexing strategy
* Relationships (1:1, 1:N, M:N)
* Entity-Relationship Diagrams

#### 5. **Interface Design**

* Interfaces and abstractions
* Service contracts
* API contract definition

#### 6. **Data Structures and Algorithms**

* Custom implementations (queues, caches)
* Trees, Graphs, Maps
* Efficient querying

#### 7. **Sequence Diagrams**

* Object interactions
* Request/response flow
* Service-to-service call flows

#### 8. **Error Handling & Validation**

* Exception handling strategy
* Retry mechanisms
* Input validation strategy

#### 9. **Concurrency & Thread Safety**

* Multi-threaded architecture
* Locks, semaphores
* Event-driven patterns

#### 10. **Coding Conventions**

* Code modularity and readability
* Naming conventions
* Comments, documentation

#### 11. **Testability**

* Unit testing strategy
* Mocking and Dependency Injection
* Test coverage tools (xUnit, Moq, etc.)

#### 12. **UML Diagrams (LLD-level)**

* Class Diagrams
* Sequence Diagrams
* Activity Diagrams
* State Diagrams

### 🧠 Summary

| **Area** | **High-Level Design** | **Low-Level Design** |
| --- | --- | --- |
| Focus | Architecture, components, scalability | Class design, patterns, logic flow |
| Output | Architecture diagrams, tech stack | UML, class & sequence diagrams |
| Audience | Architects, Tech Leads, Stakeholders | Developers, Code Reviewers |
| Granularity | Coarse | Fine |

Would you like a checklist format for interviews or documentation templates (like HLD & LLD templates)?

### ✅ High-Level Design (HLD) and Low-Level Design (LLD) Interview Prep

#### Includes Topics, Real-World Use Cases, Key Questions, and Expected Answers

## ✅ High-Level Design (HLD)

### 1. System Requirements and Functional Overview

**Use Case:** Design a healthcare appointment booking system for hospitals.

**Questions & Answers:**

1. **What are the functional and non-functional requirements?**
   * **A:** Functional: Book appointment, view schedule, cancel. Non-functional: Scalability, security, availability.
2. **How do you gather and analyze system requirements?**
   * **A:** Through stakeholder interviews, document analysis, and use case modeling.
3. **What are system constraints?**
   * **A:** Legacy systems, response time under 2s, regulatory compliance (e.g., HIPAA).
4. **How do you ensure scalability and extensibility?**
   * **A:** Through microservices, horizontal scaling, and modular design.
5. **What are the expected SLA/SLOs?**
   * **A:** 99.9% uptime, 1-second average response time, < 1% error rate.

### 2. Component-Level Design (Modules and Services)

**Use Case:** E-commerce platform with product, cart, and order services.

**Questions & Answers:**

1. **What are the key components in your design?**
   * **A:** Product catalog, shopping cart, order processing, user profile, payment.
2. **How do services communicate?**
   * **A:** Synchronously via REST; async via message queue (e.g., RabbitMQ).
3. **How do you design for statelessness?**
   * **A:** No session in service; use tokens for identity.
4. **What design principles do you follow?**
   * **A:** SOLID, DRY, KISS, YAGNI, high cohesion, low coupling.
5. **How do you break monolith into services?**
   * **A:** Identify bounded contexts, define service boundaries, migrate incrementally.

### 3. Data Management & Storage Strategy

**Use Case:** Customer order tracking system.

**Questions & Answers:**

1. **How do you handle data consistency?**
   * **A:** Eventual consistency across services using event sourcing.
2. **What DB types do you use?**
   * **A:** Relational (SQL Server) for orders, NoSQL (CosmosDB) for audit logs.
3. **How do you ensure data security?**
   * **A:** Encryption at rest and transit, RBAC, audit logs.
4. **How do you handle schema evolution?**
   * **A:** Use versioning, migrations, backward compatibility.
5. **What is your strategy for multi-region data?**
   * **A:** Data replication, geo-partitioning, consistency trade-offs.

### 4. API and Integration Design

**Use Case:** Insurance claim processing platform.

**Questions & Answers:**

1. **What API design approach do you follow?**
   * **A:** RESTful, versioned APIs with Swagger/OpenAPI specs.
2. **How do you handle integration with third-party systems?**
   * **A:** Use adapters, API gateways, retry policies, timeouts.
3. **How do you secure APIs?**
   * **A:** OAuth2, Azure AD B2C, rate limiting, input validation.
4. **What are some common integration patterns?**
   * **A:** Sync request-response, async event-driven, polling.
5. **How do you manage backward compatibility?**
   * **A:** API versioning, feature toggles, deprecation policy.

### 5. Deployment, Scaling, and Infra View

**Use Case:** Global web application for ride-sharing.

**Questions & Answers:**

1. **What is your deployment strategy?**
   * **A:** CI/CD pipelines, blue-green or canary releases.
2. **How do you scale the system?**
   * **A:** Auto-scaling groups, load balancers, service replication.
3. **What infrastructure do you use?**
   * **A:** Azure App Services, AKS, CosmosDB, Redis.
4. **How do you ensure high availability?**
   * **A:** Multi-AZ deployment, failover clusters, redundancy.
5. **How do you monitor and alert?**
   * **A:** Azure Monitor, Application Insights, custom alerts.

## ✅ Low-Level Design (LLD)

### 1. Class Design and Object Modeling

**Use Case:** Library management system.

**Questions & Answers:**

1. **How do you identify classes in the system?**
   * **A:** Use noun extraction from requirements and responsibilities.
2. **What principles do you apply in class design?**
   * **A:** SOLID, high cohesion, low coupling.
3. **How do you model relationships between classes?**
   * **A:** Use UML diagrams – association, aggregation, composition.
4. **How do you ensure flexibility in class design?**
   * **A:** Use interfaces, design patterns like Strategy, Factory.
5. **How do you represent domain objects vs DTOs?**
   * **A:** Domain objects hold business logic; DTOs are data carriers.

### 2. Design Patterns Usage

**Use Case:** Payment gateway integration module.

**Questions & Answers:**

1. **What design pattern would you use to switch payment providers?**
   * **A:** Strategy pattern.
2. **When would you use a Factory pattern?**
   * **A:** To create objects without exposing creation logic.
3. **What’s the benefit of Repository pattern?**
   * **A:** Abstracts data access, promotes testability.
4. **What’s the difference between Adapter and Facade?**
   * **A:** Adapter converts interface; Facade simplifies interface.
5. **When is Singleton useful?**
   * **A:** Shared config/service where one instance is required.

### 3. Data Structures and Algorithms

**Use Case:** Real-time leaderboard for a game.

**Questions & Answers:**

1. **Which data structure would you use?**
   * **A:** Balanced BST or heap for ranking.
2. **How do you handle millions of entries efficiently?**
   * **A:** Use distributed cache like Redis with sorted sets.
3. **What sorting algorithm is optimal for near-sorted data?**
   * **A:** Insertion sort.
4. **How do you ensure time-efficient access?**
   * **A:** HashMap + LinkedList for O(1) operations.
5. **How would you paginate large results?**
   * **A:** Use offset-limit or keyset pagination.

### 4. Exception Handling and Logging

**Use Case:** Finance app requiring audit logs.

**Questions & Answers:**

1. **What’s your strategy for error handling?**
   * **A:** Centralized try-catch, meaningful messages, custom exceptions.
2. **How do you log sensitive operations?**
   * **A:** Use secure logging with masking, logging libraries.
3. **How do you handle third-party API failures?**
   * **A:** Retry pattern, fallback, circuit breaker.
4. **What tools do you use for logging?**
   * **A:** Serilog, Application Insights, Log4Net.
5. **How do you handle global exceptions in .NET?**
   * **A:** Use middleware or filters for exception logging.

### 5. Validation, Configuration, and Security

**Use Case:** User onboarding and profile management.

**Questions & Answers:**

1. **How do you validate user input?**
   * **A:** Data annotations, FluentValidation, manual checks.
2. **Where do you store app config?**
   * **A:** appsettings.json, Azure Key Vault.
3. **How do you manage secrets?**
   * **A:** Azure Key Vault, environment variables.
4. **How do you secure internal APIs?**
   * **A:** Use internal VNETs, client certificate, IP whitelisting.
5. **What authentication flows have you implemented?**
   * **A:** OAuth2 with PKCE, JWT-based, Azure AD B2C login.

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

## 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

### 1. Introduction & Scope

This High-Level Design (HLD) document outlines the architecture for a **Healthcare Prior Authorization System** for eviCore. The system enables intake agents, call center representatives, and providers to submit, track, and review prior authorization requests for various health plans and supported programs.

### 2. Functional Requirements

* Login-based access for agents and providers
* Create prior authorization requests for patients
* Patient and provider lookup
* Eligibility checks for patients and procedures
* Selection of CPT/Procedure and Diagnosis codes
* Attach clinical documents and requester details
* Track historical and current requests
* Review workflow by MDs and nurses
* Final review statuses (Approved, Denied with rationale, Partial Approval, Withdrawn, On Hold)
* Auto-approval logic for eligible cases
* Export requests to payers like Cigna and handle response via EDI
* Kafka-based internal event communication for updates
* Request reassignment and dashboard for team collaboration
* Activity audit and note-tracking within request
* Prevention of duplicate/similar requests

### 3. Non-Functional Requirements

* High availability (99.9%)
* Low-latency APIs (< 500ms)
* Scalable to thousands of concurrent users
* HIPAA compliance
* End-to-end audit logging and traceability

### 4. Architecture Overview

* Microservices-based architecture deployed in Azure
* Web portal built with React (desktop and mobile responsive)
* Backend developed in ASP.NET Core
* RESTful API communication with asynchronous messaging using Azure Service Bus
* Authentication and authorization via Azure AD B2C

### 5. Component Diagram

+------------------+ +------------------------+

| Provider Portal | <---> | API Gateway (APIM) |

+------------------+ +------------------------+

| |

+------+ +--------+

| Auth |<---->| Azure |

| Svc | | AD B2C |

+------+ +--------+

|

+------------------+ +-----------------------+ +------------------+

| PA Submission Svc|<--> | Validation & Routing |<--> | Payer Decision UI|

+------------------+ +-----------------------+ +------------------+

|

+----------------+

| Notification Svc|

+----------------+

|

+------------------+

| Audit Logging DB |

+------------------+

### 6. Technology Stack

* **Frontend:** React, TypeScript
* **Backend:** ASP.NET Core (C#)
* **Authentication:** Azure AD B2C
* **Messaging:** Azure Service Bus
* **Storage:** Azure SQL, Azure Blob Storage (for clinical docs)
* **Deployment:** Azure App Service, Azure DevOps Pipelines

### 7. Module Responsibilities

* **PA Submission Service:** Captures and validates new authorization requests
* **Validation & Routing Engine:** Applies health plan rules, handles delegation, eligibility, and auto-approval
* **Notification Service:** Sends email/SMS alerts to providers and members
* **Audit Logging:** Maintains secure logs for all user activities
* **Admin Portal:** Allows internal configuration of payer programs and routing logic

### 8. Communication Flow

* REST APIs for portal interactions
* Azure Service Bus for event-driven internal workflows
* EDI file exchange for external payer integration (e.g., Cigna)
* Kafka events to sync decisions from UM system into intake portal

### 9. Data Management Strategy

* **Relational DB:** Azure SQL for structured request data
* **Blob Storage:** Stores clinical documents and scanned attachments
* **Caching:** Azure Redis for fast access to provider/patient lookups
* **Backups:** Geo-redundant and automated snapshotting

### 10. Security Considerations

* OAuth2-based authentication
* Role-based access control: Agent, Reviewer, Provider, Admin
* Encryption at rest and in transit
* HIPAA-compliant logging and access control

### 11. Scalability & Availability

* Auto-scale App Services
* Distributed event handling through Kafka and Service Bus
* Global load balancing using Azure Front Door or App Gateway
* Multi-region deployment support

### 12. Deployment Strategy

* Infrastructure as Code (Bicep/Terraform)
* CI/CD pipelines using Azure DevOps
* Blue-green deployment model
* Pre-production integration test environment

### 13. Observability

* Azure Application Insights for telemetry
* Dashboards for monitoring latency, throughput, failures
* Alert rules configured via Azure Monitor

### 14. Trade-Off Analysis

* Chose Azure Service Bus over Kafka for external integration simplicity, but Kafka for internal event chaining
* Selected Azure Blob over relational DB for large unstructured document storage
* Microservices architecture preferred over monolithic design for better scalability and maintainability

### 15. Diagrams

* **Component Diagram** (included above)
* **Deployment Diagram:** Azure App Services, API Gateway, Azure SQL, Azure Storage, Azure DevOps
* **Sequence Diagram:** Submit Request → Validate → Route → Review → Notify

Let me know if you'd like this polished version exported to **Word or PDF**, or if you'd like to move forward with the **Low-Level Design (LLD)** for this healthcare intake system.