### 🧱 Solution Blueprint – Prior Authorization Intake & Review System

This blueprint outlines the architecture and key building blocks of a scalable, secure, microservices-based \*\*Prior Authorization Intake and Review Platform\*\* for the healthcare domain (e.g., for eviCore).

---

## 🔹 1. Architecture Overview

| Layer | Technology | Responsibilities |

|-------|------------|------------------|

| \*\*Frontend\*\* | React.js, TypeScript, MSAL.js | Intake Portal, Review Dashboard, Forms, Secure B2C login |

| \*\*Backend APIs\*\* | ASP.NET Core Web API | Patient, Provider, Request, Clinical Review services |

| \*\*Auth\*\* | Azure AD B2C | OAuth2 login, Role-based access (RBAC) |

| \*\*Communication\*\* | Azure Service Bus | Async events between services (Event-Driven) |

| \*\*Data Store\*\* | Azure SQL, Azure Blob, Redis | Normalized tables for requests, file storage, caching rules |

| \*\*DevOps\*\* | Azure DevOps, Bicep/Terraform | CI/CD, infra as code, deployment automation |

| \*\*Monitoring\*\* | Azure Monitor, App Insights, Log Analytics | Logs, telemetry, distributed tracing |

---

## 🔹 2. Microservice Components

### 🧩 Intake Microservices:

- \*\*Patient Service\*\* – Search, validate eligibility, fetch history

- \*\*Provider Service\*\* – Lookup requesting/rendering/ordering providers

- \*\*Procedure Service\*\* – CPT/HCPCS validation, basket selection

- \*\*Diagnosis Service\*\* – Code lookup and eligibility validation

- \*\*Request Intake Service\*\* – Accept request shell, emit domain events

- \*\*Attachment Service\*\* – Secure upload via SAS token to Blob Storage

### 🧩 Review & Workflow Microservices:

- \*\*Clinical Review Service\*\* – MD/Nurse decisioning, rationale capture

- \*\*Notes & Activity Service\*\* – Full audit history, user notes

- \*\*Determination Service\*\* – Approval/denial/partial status mgmt

- \*\*Delegation Service\*\* – Routes request to Cigna (external partner)

### 🧩 Integration Services:

- \*\*Kafka Gateway Adapter\*\* – Consumes Cigna EDI results → Updates request

- \*\*Notification Service\*\* – Emails to patients, providers

- \*\*Outbox Processor\*\* – Eventual consistency using transactional outbox

---

## 🔹 3. C4 Model – Component View

\*\*System Context:\*\*

```

[Patient] ──> [React Portal] ──> [API Gateway (Azure APIM)] ──> [Microservices]

[Provider] ──> [Intake Portal] ──> [B2C Auth] → [Backend Services]

↓

[Blob Storage, SQL, Kafka]

```

\*\*Containers:\*\*

- Frontend: React SPA

- API Gateway: Azure API Management

- Services: Dockerized microservices

- Storage: Azure SQL + Blob + Redis

- Async Messaging: Azure Service Bus + Kafka adapter

---

## 🔹 4. Domain-Driven Design (DDD)

### 🧠 Core Aggregates

- `AuthorizationRequest` (root) → Has `Patient`, `ProcedureBasket`, `DiagnosisList`, `ProviderLinks`, `ClinicalAttachments`, `ReviewHistory`

- Rich domain models with business validation (eligibility, delegation, time-window restrictions)

- Uses Domain Events like `RequestCreated`, `ReviewCompleted`, `DelegatedToPayer`

---

## 🔹 5. API Gateway Design

- API published via \*\*Azure API Management (APIM)\*\*

- Authenticated with \*\*Azure AD B2C JWT\*\*

- Rate limiting, request size, validation policies in APIM

---

## 🔹 6. Security Model

| Layer | Control |

|-------|---------|

| UI | MSAL + B2C + Role Guards |

| API | JWT validation, `[Authorize]` by role/scope |

| Data | Encryption-at-rest, SQL RBAC, Blob SAS tokens |

| Web | CSP headers, X-Frame-Options, Anti-XSS |

---

## 🔹 7. Deployment Strategy

- Use \*\*Azure DevOps Pipelines\*\*

- Bicep/Terraform to provision:

- App Services

- APIM + Azure SQL

- Blob Storage + Redis

- Use \*\*blue/green deployment\*\* with slot swaps

---

## 🔹 8. Observability

| Tool | Purpose |

|------|---------|

| Application Insights | Logs, metrics, exceptions, traces |

| Azure Monitor | Alerts, health probes |

| Log Analytics | Query history, failures |

---

## 🔹 9. Database Design (Azure SQL)

- Normalized relational model

- Indexed by `RequestId`, `PatientId`, `CreatedAt`

- Partitioned tables by month (for `AuthorizationRequests`)

- Separate `ClinicalAttachments` in Blob

---

## 🔹 10. Data Flow Summary

1. Agent logs in → Authenticated via B2C

2. Patient searched → Eligibility verified

3. Provider, Procedure, Diagnosis collected via micro-forms

4. Attachments uploaded (blob)

5. Final request submitted → Domain Event `RequestCreated`

6. Notification, Review, Delegation processed asynchronously

7. Status updated, user notified, audit logged

---

Would you like a \*\*UML package\*\*, \*\*architecture diagram image\*\*, or an \*\*exportable Word or PDF version\*\* of this solution blueprint?

✅ The solution blueprint has been generated and added to your document, covering:

* Microservices architecture
* C4 model (component-level)
* DDD aggregates
* API gateway strategy
* Security model
* Database design
* Full data flow & deployment model

Would you like a **Word/PDF export**, **UML diagrams**, or a **C4 visual rendering** next?

This is a **C4 Component-Level Diagram** for the **Prior Authorization Intake System**. It visualizes:

### 🧩 **Key Elements in the Architecture:**

#### 👥 **External Actors:**

* **Provider** – Initiates requests on behalf of patients
* **Intake Agent** – Internal user entering request details
* **Payer** – Health plan reviewing and sending back determination (via events)

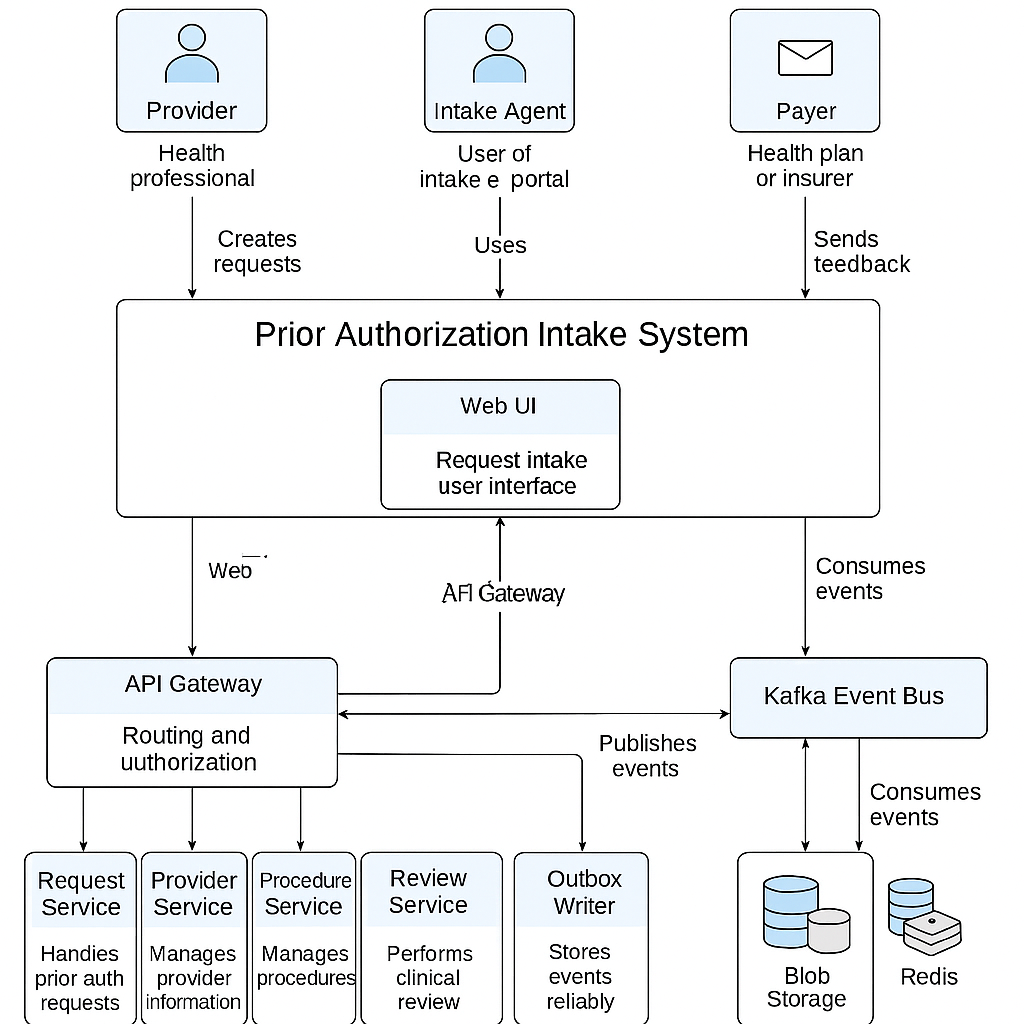
#### 🧱 **Core Components:**

* **Web UI** – React-based intake portal used by agents and providers
* **API Gateway (APIM)** – Routes requests, applies policies (auth, rate-limiting)
* **Microservices:**
  + Request Service – Creates and tracks requests
  + Provider Service – Provider validation and details
  + Procedure Service – CPT/HCPCS logic and grouping
  + Review Service – Handles MD/Nurse review actions
  + Outbox Writer – Ensures reliable domain events are pushed

#### 🌀 **Event-Driven Infra:**

* **Kafka Event Bus** – Publishes/consumes review and delegation events
* **Blob Storage** – For clinical document uploads
* **Redis** – Caching for eligibility rules or lookup acceleration

Would you like:

* The **UML Class Diagram** next?
* A **Sequence Diagram** for request intake or review flow?
* Or a **Word export** of this C4 explanation?

This is a **UML Class Diagram** representing the **Domain Model** for your **Healthcare Prior Authorization System**. It includes all key entities and their relationships according to **DDD principles**.

### 🧩 Key Classes & Descriptions:

#### 📌 AuthorizationRequest (Aggregate Root)

* Central entity for the intake request.
* Holds references to:
  + Patient
  + RequesterInformation
  + Program, Created
  + List of ProcedureBasket, Activities, Attachments
* Includes a Validate() method for business rules.

#### 👤 Patient

* Contains demographic details like Name, DOB.

#### 🧑‍⚕️ RequesterInformation

* Requesting party (e.g., provider staff or intake agent).

#### 🧾 ProcedureBasket

* Holds selected Procedure with a flag (IsAdditional).
* Can include multiple CPT codes.

#### 🔢 DiagnosisCode

* ICD-10 codes linked to the request.

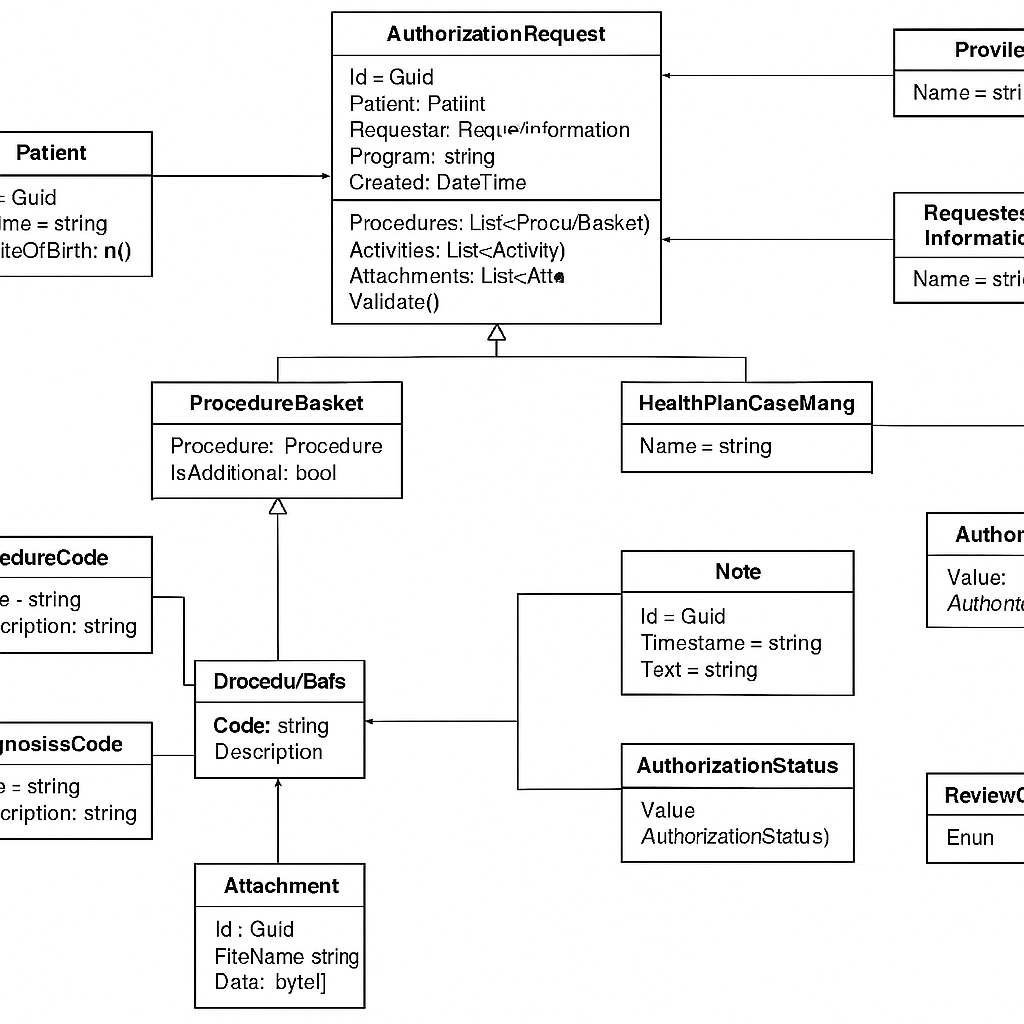
#### 🗃️ Attachment

* Medical documents uploaded by providers.

#### 🏥 HealthPlanCaseManager

* Assigned case manager per request or plan.

#### 🧠 Note



This is a **High-Level Architecture Diagram** for your **Healthcare Prior Authorization Intake & Review System**.

### 📌 Key Elements Explained:

#### 👥 **Actors (Left - Clients):**

* **Providers** and **Intake Agents** interact with the system via the portal.

#### 🧩 **Frontend Application:**

* **Prior Authorization Intake & Review Portal**: A React-based web interface used by agents/providers to create and manage requests.

#### 🌐 **API Gateway Layer:**

* **Azure API Gateway (APIM)** routes requests securely to backend microservices.
* Handles JWT auth, rate limiting, and routing.

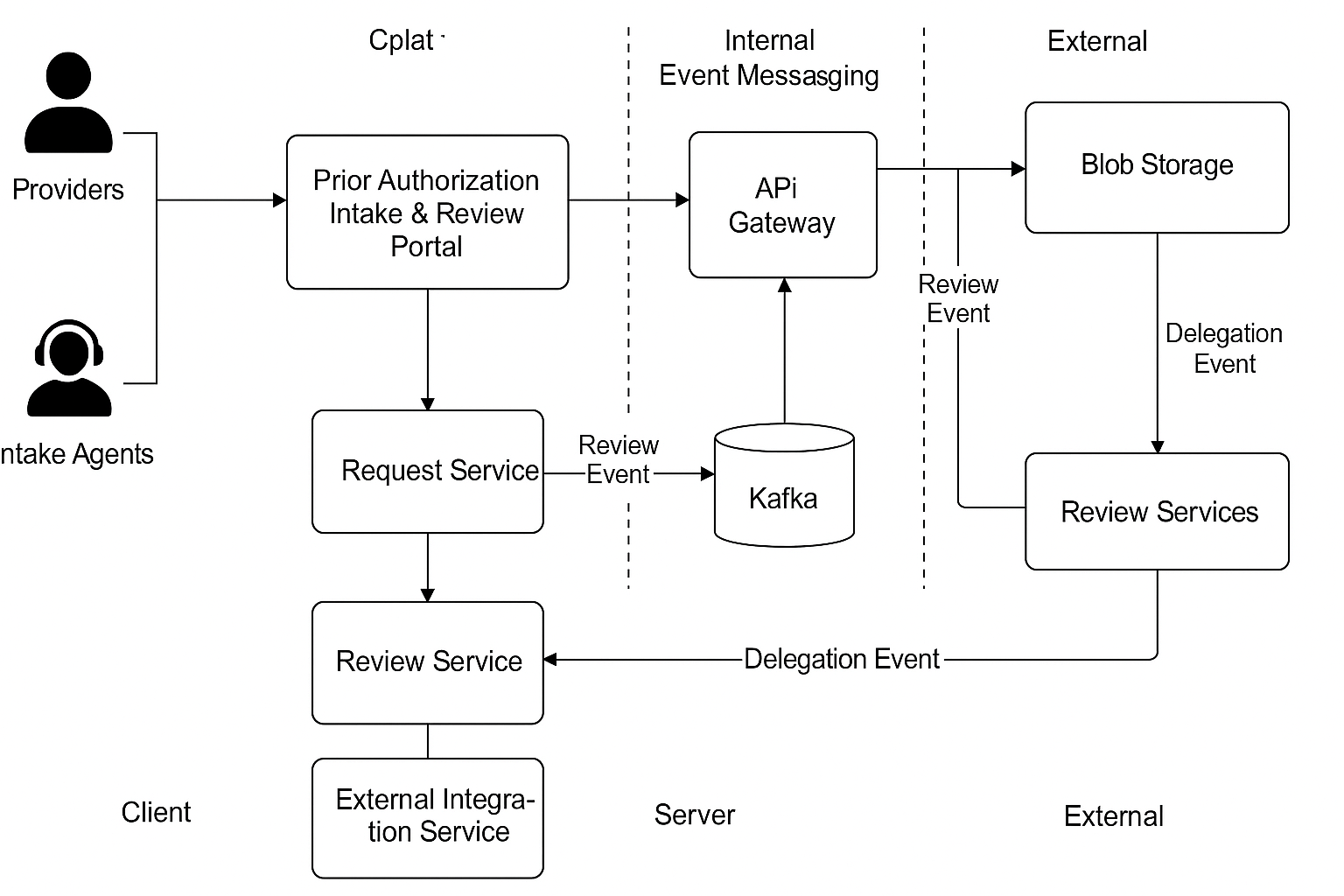
#### 🔧 **Core Microservices (Server):**

* **Request Service**: Captures procedure code, patient, provider, eligibility details.
* **Review Service**: Manages medical reviews, approvals/denials.
* **External Integration Service**: Sends outbound requests or events (e.g., to Cigna).
* **Kafka**: Event streaming between services (e.g., ReviewEvent, DelegationEvent).

#### ☁️ **External Systems:**

* **Blob Storage**: Stores clinical documents securely (e.g., PDFs, reports).
* **Review Services (e.g., Cigna)**: Processes delegated cases, responds via Kafka.

Would you like a version with **color-coded zones**, or a **sequence diagram** next to explain request flow from portal to determination?



Excellent question! Understanding the **differences** between **Solution Blueprint**, **High-Level Architecture Diagram**, **Low-Level Design (LLD)**, and **Architecture Decision Records (ADR)**—and knowing when to create each—is essential for software architects and tech leads.

## 📘 1. ✅ **Solution Blueprint**

### 🔹 What It Is:

A **holistic document** that provides a **strategic overview** of the system being built. It outlines:

* Business context and scope
* System goals and constraints
* Key components (frontend/backend, APIs, databases)
* Technology choices
* Deployment, integration, and observability plans
* Security strategy

### 📌 Example Content:

* Microservices list and responsibilities
* C4 diagrams (Context, Container, Component)
* Data flow between major modules
* Message queues, external integrations

### 🕓 When Created:

✅ **Very early**, usually after the requirements gathering phase, before actual design or development.

## 🧱 2. ✅ **High-Level Architecture Diagram**

### 🔹 What It Is:

A **visual representation** of your system’s major building blocks and their interactions.

### 📌 Shows:

* Clients, APIs, services
* Communication protocols (REST, gRPC, Kafka, etc.)
* Cloud services used (e.g., Azure App Services, APIM, SQL, Blob Storage)
* Authentication flows (Azure B2C, OAuth)

### 🕓 When Created:

✅ After the **Solution Blueprint** is reviewed and finalized.

It's typically included as part of the solution blueprint but can also stand alone in architecture documentation or executive presentations.

## 🧪 3. ✅ **Low-Level Design (LLD)**

### 🔹 What It Is:

A **detailed technical design document** focused on **individual components or services** in the system. It is consumed by developers for actual implementation.

### 📌 Includes:

* Class diagrams (UML), sequence diagrams
* API specs (endpoints, methods, models)
* Database schema (tables, relationships)
* Design patterns applied (e.g., Factory, Strategy)
* Interface contracts and service boundaries
* Validation logic, error handling plans

### 🕓 When Created:

✅ Once HLD is finalized, **just before or during** development of each module/service.

## 📜 4. ✅ **Architecture Decision Records (ADR)**

### 🔹 What It Is:

A **log of architecture decisions**, written in a consistent format, documenting:

* A particular design choice
* Alternatives considered
* Justification
* Impacts and consequences

### 📌 Example Decisions:

* Why choose Azure Service Bus over Kafka
* Why go with Microservices instead of Monolith
* Why use Event Sourcing or CQRS
* Why Azure AD B2C for auth

### 🕓 When Created:

✅ **Along the way**, during HLD, LLD, or even development—whenever **a key decision is made** that could affect architecture, scalability, or maintainability.

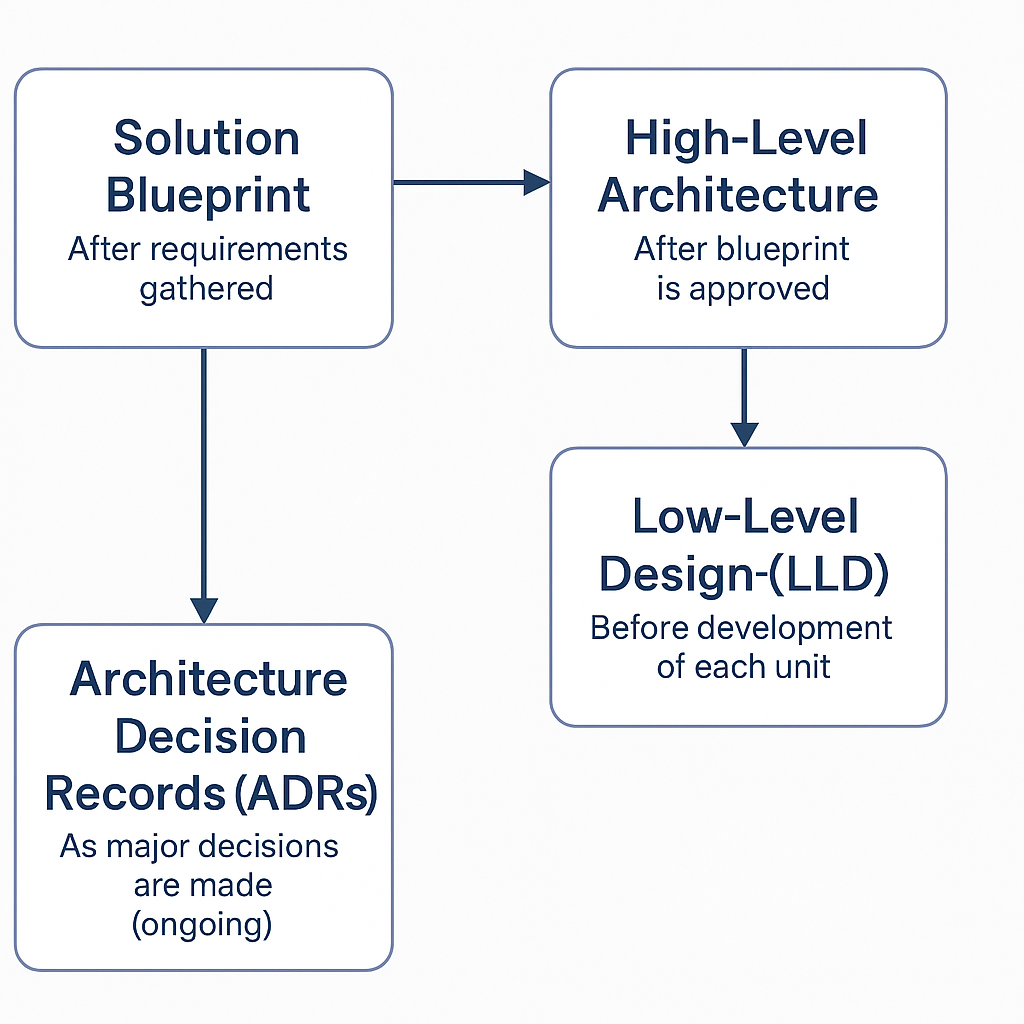
## 🧭 Summary: Order of Creation

| **Artifact** | **Created When?** | **Created By** | **Output Format** |
| --- | --- | --- | --- |
| ✅ Solution Blueprint | After requirements gathered | Solution/Enterprise Architect | Doc / Confluence page |
| ✅ High-Level Architecture | After blueprint is approved | Technical Architect | Diagram (Lucidchart, Draw.io) |
| ✅ ADRs (ongoing) | As major decisions are made | Architects / Tech Leads | Markdown or .adr format |
| ✅ Low-Level Design (LLD) | Before development of each unit | Developers / Leads | Detailed docs + UML diagrams |

## 🧠 Quick Analogy

| **Term** | **Analogy** |
| --- | --- |
| Solution Blueprint | City master plan (zoning, roads) |
| HLD Diagram | Highway map of the city |
| LLD | Blueprint of each building |
| ADR | Council meeting minutes on decisions like bridge vs tunnel, water system choices etc. |

Would you like a **visual timeline** showing when each document fits into a project lifecycle? Or a **template** (Word/Markdown) for any of these artifacts?



Here's a detailed explanation and purpose of each item from the visual timeline:

### 1. ✅ **Solution Blueprint**

#### 📘 Description:

A **comprehensive strategic document** that defines the overall approach to building the system. It outlines system objectives, functional boundaries, architectural patterns, technology stack, deployment considerations, and integration touchpoints.

#### 🎯 Purpose:

* Align technical vision with business goals
* Define scope, constraints, assumptions
* Establish shared understanding between stakeholders
* Set the foundation for high-level and low-level design
* Plan for scalability, security, and maintainability

### 2. ✅ **High-Level Architecture Diagram**

#### 🧱 Description:

A **visual overview** of the system architecture showing:

* Major components (e.g., microservices, UI, DB, external systems)
* Communication paths (REST, message bus, etc.)
* Authentication flows
* Cloud services used

#### 🎯 Purpose:

* Communicate the structure of the system clearly
* Help in identifying separation of concerns
* Assist infra & dev teams in setting up environments
* Useful in onboarding, reviews, and audits

### 3. ✅ **Low-Level Design (LLD)**

#### 🧪 Description:

A **detailed design document** that focuses on the **internals** of each module or service. It includes:

* Class diagrams, sequence diagrams
* Data models and validation rules
* API endpoint contracts
* Component interactions
* Error handling and retry logic

#### 🎯 Purpose:

* Serve as a blueprint for developers to implement the system
* Ensure consistency across modules
* Handle edge cases and validation up front
* Reduce rework by clarifying design early

### 4. ✅ **Architecture Decision Records (ADRs)**

#### 📜 Description:

ADRs are **lightweight documents** that record **important architectural decisions** made during the project, including:

* The problem being solved
* Alternatives considered
* Chosen solution
* Consequences of the decision

#### 🎯 Purpose:

* Maintain a history of architectural thinking
* Provide traceability for future changes or audits
* Encourage thoughtful evaluation of trade-offs
* Help new team members understand why choices were made

### 🧭 Timeline Flow:

1. **Solution Blueprint** – Created first after requirements.
2. **High-Level Architecture** – Visual design derived from the blueprint.
3. **Low-Level Design (LLD)** – Done after HLD for each component or service.
4. **ADR** – Created any time a major decision is made (ongoing).

Would you like templates or examples for any of these (Blueprint, LLD, ADR)?