**Think of Your Project in 3 Layers of Design**

| **Layer** | **Purpose** | **UML Diagrams** |
| --- | --- | --- |
| **High-Level (System Architecture)** | What the system does and how services interact | Use Case, Context, Component |
| **Medium-Level (Behavior & Process)** | How data and control flow through the system | Sequence, Activity |
| **Low-Level (Implementation & Deployment)** | How classes, controllers, and runtime environments are structured | Class, Deployment |

Let’s break it down.

**HIGH-LEVEL — “What and Who?”**

**Use Case Diagram**

* Defines all **functional requirements**.
* Clearly identifies all **actors (users, admins, recruiters)** and **major system functions**.
* Helps you write **user stories** and **test cases** later.

**You understand:** *What the system should do.*

**Context Diagram**

* Defines the **system boundaries**.
* Shows **external systems** (e.g., external APIs) and **how your services interact**.
* Defines the **input/output flow** of your entire system.

**You understand:** *How your system fits in a bigger ecosystem.*

**Component Diagram**

* Breaks your system into **microservices and modules**.
* Shows **dependencies** (like which service depends on common-lib).
* Clarifies **service interfaces** and how data moves between them.

**You understand:** *How to structure the codebase and service communication.*

**MEDIUM-LEVEL — “How does it behave?”**

**Sequence Diagram**

* Shows **request-response flow** between components for a specific use case.
* Example: “User uploads resume → resume parsed → job matcher finds jobs → response returned.”
* Helps in designing your **controller → service → repository** chain properly.

**You understand:** *How your system behaves dynamically.*

**Activity Diagram**

* Represents the **workflow logic** within a single process.
* Example: “Resume upload → extract text → NLP skill extraction → store → send to matcher.”
* Excellent for designing your **service-layer methods** or **business logic pipelines**.

**You understand:** *What happens inside a single workflow step-by-step.*

**LOW-LEVEL — “How is it implemented and deployed?”**

**Class Diagram**

* Shows the **internal structure** of each service:
  + Entities
  + DTOs
  + Controllers
  + Services
  + Repositories
  + Relationships between them
* Each microservice can have its own class diagram.
* You can even add **method signatures**, **attributes**, and **inheritance** for precision.

**You understand:** *How to implement the code.*

**Deployment Diagram**

* Represents the **runtime structure**:
  + Containers (Docker)
  + Networks
  + Databases
  + Ports
  + Cloud setup (AWS, etc.)
* Used for **DevOps and deployment planning**.

**You understand:** *How to run and scale your system.*

**Together — These 7 UML Diagrams Cover 95% of Your System Design**

But, to reach **100% coverage** (including testing, CI/CD, and detailed class internals), here are **optional UMLs** that can complement your design:

| **Extra Diagram** | **When to Use** | **Purpose** |
| --- | --- | --- |
| **State Machine Diagram** | When an entity has multiple states (e.g., JobApplication → Applied → Shortlisted → Hired → Rejected) | Captures transitions and conditions |
| **Package Diagram** | When your codebase grows large | Groups related classes/modules for maintainability |
| **Object Diagram** | For debugging or documentation | Shows instance-level data (snapshot of Class Diagram) |
| **Communication Diagram** | Alternative to Sequence | Emphasizes object relationships instead of time flow |
| **Test Case Mapping Diagram** | Optional custom diagram | Links Use Cases ↔ Unit Tests ↔ Integration Tests |
| **CI/CD Flow Diagram** | Optional DevOps design | Visualizes build, test, and deployment pipeline |

**Summary — Coverage Map**

| **Concern** | **Covered By** |
| --- | --- |
| System Features / Users | Use Case |
| External Integrations | Context |
| Service Architecture | Component |
| Data Model / Relationships | Class |
| Execution Flow | Sequence |
| Workflow Logic | Activity |
| Runtime Infrastructure | Deployment |
| State Transitions (optional) | State Machine |
| Packaging / Module Grouping (optional) | Package |
| Testing & CI/CD (optional) | Test Flow / CI-CD Diagram |

## Recommended Path for You

To make your documentation and understanding complete for **Career Nexus**:

1. **Create the 7 core UML diagrams** (Use Case → Deployment)
2. For **Job and Resume workflow**, add:
   * a **State Machine Diagram** (optional but insightful)
3. Later, when you do **Docker + Jenkins setup**, create:
   * a **CI/CD Flow Diagram**
4. Combine everything into a **System Design Document (SDD)** or **Architecture Doc**.