# System Design Lab – Project Plan

A hands-on, end-to-end framework to master large-scale system design through practical builds, simulation, and iterative refinement. This document defines structure, rules, and workflows to maximize learning and interview readiness (E5+ level).

## 1. Goals of System Design Lab

The goal of this project is to convert theoretical system design knowledge into deep intuition by building, running, and stress-testing realistic distributed systems. This repository is designed to simulate how senior engineers reason about architecture, tradeoffs, and failure modes.

1 Develop E5-level system design thinking and narration

2 Practice translating abstract designs into running systems

3 Build intuition for scale, bottlenecks, and failure handling

4 Create a credible, interview-safe public portfolio

## 2. Repository Structure

The repository is organized around multiple independent systems. Each system follows the same artifact structure to enforce discipline and repeatability.

system-design-lab/ ■■■ README.md ■■■ principles/ ■ ■■■ design-invariants.md ■■■ systems/ ■ ■■■ top-k-user-aggregation/ ■ ■ ■■■ design/ ■ ■ ■ ■■■ excalidraw.png ■ ■ ■ ■■■ design-doc.md ■ ■ ■■■ implementation/ ■ ■ ■ ■■■ docker-compose.yml ■ ■ ■ ■■■ services/ ■ ■ ■■■ load-test/ ■ ■ ■■■ lessons-learned.md ■ ■■■ ...

## 3. The Four Required Artifacts (Per System)

### 3.1 Excalidraw Diagram

This is the interviewer-facing whiteboard. It must clearly show read paths, write paths, async processing, and state ownership. The diagram should be explainable end-to-end in under 10 minutes.

### 3.2 Design Document

A concise but complete design document (5–7 pages max) that explains assumptions, APIs, data models, core flows, scaling strategy, and tradeoffs. This is where E5-level reasoning must be explicit.

### 3.3 Dockerized Implementation

A minimal but realistic implementation using open-source equivalents. The goal is not feature completeness but architectural fidelity: correct boundaries, state placement, and async behavior.

### 3.4 Load & Failure Simulation

Synthetic load and failure injection to validate assumptions. Results must feed back into updates to the design diagram and document.

## 4. Do’s and Don’ts

Do:

1 Always state assumptions explicitly before designing

2 Design read paths as carefully as write paths

3 Name and justify tradeoffs (latency vs cost, freshness vs accuracy)

4. Treat failures as first-class design inputs
5. Keep systems small but conceptually complete

Don’t:

1. Do not over-engineer or chase Meta-internal details
2. Do not add tools without a clear problem they solve
3. Do not let implementation drift from the design doc
4. Do not optimize algorithms before fixing system boundaries
5. Do not hide behind tool names instead of explaining principles

## 5. How to Use AI Editors Effectively

AI editors are accelerators, not decision-makers. They should compress iteration time while you retain ownership of architectural choices.

AI Do:

1. Use AI to generate boilerplate (Dockerfiles, configs, CRUD APIs)
2. Use AI to refactor code after design changes
3. Use AI to generate load-testing and failure-injection scripts
4. Review AI-generated code like a senior reviewer

AI Don’t:

1. Do not let AI choose data models or shard keys implicitly
2. Do not accept retry or consistency logic without scrutiny
3. Do not skip design docs because AI can write code faster

## 6. Recommended Execution Plan

Each system should take approximately 2–3 weeks. Depth is more important than the number of systems. After 3–4 systems, system design intuition improves dramatically.

1. Week 1: Excalidraw + Design Doc
2. Week 2: Dockerized implementation
3. Week 3: Load testing, failures, and design revision

## 7. Final Note

This lab is not about building impressive demos. It is about developing correct instincts under constraints. Treat every system as a learning loop: design, build, break, refine.