GenAI System Building Challenge

Lab Hackathon

Duration: 3 Days (show us what you got on Monday)

Goal: Build your first cloud-native, microservices-style, GenAI-powered NLP system

---

Challenge Overview

Your task is to **design and build a small NLP-focused system** using GenAI tools and modern system design practices. You'll go from **idea and research** to a **working GenAI application**, gradually evolving from simple terminal-based components to a fully functional, cloud-native microservices setup.

Groups of 2 — but you're free to seek help from and use any public tools/resources available.

Key Concepts (in simple terms)

What are Microservices?

A **microservices architecture** breaks an app into **small, independent components** (services), each responsible for a single task. These services communicate with each other through well-defined interfaces (like APIs or messages).

For this project, your microservices can be:

- Separate Python files or functions at first
- Later exposed as separate APIs (e.g., using FastAPI or Flask)

What is Cloud-Native?

A **cloud-native application** is:

- Built to run in the cloud

- Easily deployable using platforms like Render.com or Streamlit Cloud
- Designed to be modular, scalable, and low-maintenance

We'll keep this simple:

- Build locally first (build an application that runs)
- Then deploy parts (like APIs or frontends) to a **free cloud platform**

Guidelines for GenAI Use

You're expected to actively use GenAI tools (like ChatGPT, Claude, Gemini, etc.) throughout every stage of your project — from brainstorming and design to coding, testing, and documentation.
As part of your final submission, reflect on your experience:

- What worked well when using GenAI?
- Where did it fall short or need human intervention?
- How did GenAI impact your development process overall?

Phase	**Use GenAI to…**
Requirements	Brainstorm features, understand domain, define user needs, create a **Requirements Specification Document**
Design	Draft architectures, break system into components
Coding	Generate boilerplate code, implement microservices
Testing	Write unit tests, test prompts, debug errors
Documentation	Auto-generate **READMEs**, user guides, **API docs**

Use any GenAI platform (ChatGPT, Claude, Gemini, etc.).

Detailed Task Steps

Phase 0 – Idea & Domain Research

- Come up with an **NLP-based application idea**.

Example: “A debate coach bot” – Research what makes a good argument, how to rate a debate performance, what formats are used (e.g., Oxford style), etc.

- Use GenAI to research:
    - How the domain works (e.g., rules of debate, characteristics of good poetry, etc.)
    - What kind of inputs/outputs would be helpful for users
    - Key features or flows that your system could support

Phase 1 – Terminal-Based Modular System (Microservices Style)

- Split your system into **independent Python modules/functions**.
    - For example: `input_handler.py`, `argument_generator.py`, `response_critic.py`
- Use **function calls or command-line I/O** to pass data between them.
- Each component should be small, focused, and ideally use GenAI.

Goal: You now have a working pipeline — even if it only runs in the terminal.

Phase 2 – Add a Minimal Frontend

- Build a **simple interface** using:
    - **Streamlit** (preferred) OR
    - CLI menus with `argparse` OR
    - A basic HTML page
- Let users interact with the GenAI pipeline without needing to modify code.

Goal: You now have an interactive app others can use and demo.

Phase 3 – Add Data Layer (Optional)

- Add basic **state storage** for:
    - User inputs
    - Generated content
    - Logs/summaries
- Use local files, SQLite, or Firebase/Supabase (if you’re comfortable)

Goal: System now supports persistence or history. Think in terms of improving customer experience using this added context.

Phase 4 – Cloud-Native Deployment

- Wrap key components as **APIs** using **FastAPI** or **Flask**
- Deploy to **Render.com** or **Streamlit Cloud**
- Connect services to simulate microservices (even if hosted in the same app)

Goal: Your GenAI app runs in the cloud, accessible via URL or endpoint.

Example Ideas (Use as Inspiration)

Example 1: Debate Coach Bot

Build an assistant that helps students prepare for debates.

- User enters a topic and their stance
- Bot generates both pro and con arguments
- Another module scores and critiques the arguments
- Bonus: Generate rebuttals or structure speech in debate format

Example 2: Poetry Style Assistant

An app that helps users write poems in specific styles.

- Choose a poet or form (e.g., haiku, Shakespearean sonnet)
- AI generates a poem or improves one provided by the user
- A critique module rates the style, rhythm, and originality
- Optional: Save and display a gallery of poems

What You Must Submit

1. **GitHub Repo** with:
    - All code and prompts
    - Requirement Specification Document
    - README with setup and usage instructions
    - Any and all system design diagram (draw.io, Excalidraw, or hand-drawn OK)
    - Unit Tests and Testing Methodology
2. **Working Demo** (even if local)
3. **5-min Walkthrough or Notes** on:

- Architecture
- GenAI usage
- What worked and what didn’t

Tips & Tools

Task	**Tools**
GenAI	ChatGPT, Claude, Gemini, Poe
Backend APIs	FastAPI, Flask
Frontend	Streamlit, Replit, HTML+JS
Deployment	Render.com (FastAPI), Streamlit Cloud
Storage	JSON, SQLite, Supabase (free)
Docs	Notion, GitHub README, Excalidraw
Bonus	Docker, GitHub Actions, Postman, LangChain