🌐 AI-Driven Development 30-Day Challenge Task -2

🗀Part A — Theory (Short Questions)

1. Nine Pillars Understanding

Q.1: Why is using AI Development Agents (like Gemini CLI) for repetitive setup tasks better for your growth as a system architect?

Answer: Using AI agents for repetitive tasks (setting up projects, configuring environments, writing boilerplate, etc.) frees up your mental energy and time. Instead of spending hours on things that don't require deep thinking, you can focus on high-level decisions: system design, architecture choices, trade-offs, and how different parts connect. This is exactly what a system architect does. The more you practice making big-picture decisions and less time you waste on mechanical work, the faster you train your brain to think like an architect.

Q.2: Explain how the Nine Pillars of AIDD help a developer grow into an M-Shaped Developer.

Answer: The Nine Pillars give a single developer superpowers that used to require an entire team (tester, DevOps, architect, product owner, etc.). When you have AI handling testing (TDD pillar), deployment, documentation, code generation from specs, and multi-step orchestration, you no longer stay narrow . You are forced to understand and control many domains deeply: testing strategies, infrastructure, clean architecture, precise requirements, agent orchestration, etc.

2. Vibe Coding vs Specification-Driven Development

Q3: Why does Vibe Coding usually create problems after one week?

Answer: Vibe coding feels fast at the beginning because you jump straight into writing code based on intuition and "what feels right." After a week (or when the project grows), you forget why certain decisions were made, variable names become confusing, hidden assumptions break, and adding new features becomes painful because there is no clear map of how everything is supposed to work. The code becomes a big ball of mud that only the original "viber" half-understands.

Q4: How would Specification-Driven Development prevent those problems?

Answer: In SDD you first write clear, executable specifications (tests or formal specs) that describe exactly what the system must do before touching implementation. These specs act as living documentation and a safety net. When requirements change or you come back after a week, the specifications tell you precisely what the intended behavior is. Any change that breaks the original intent fails instantly, so the codebase stays consistent, understandable, and maintainable for much longer.

3. Architecture Thinking

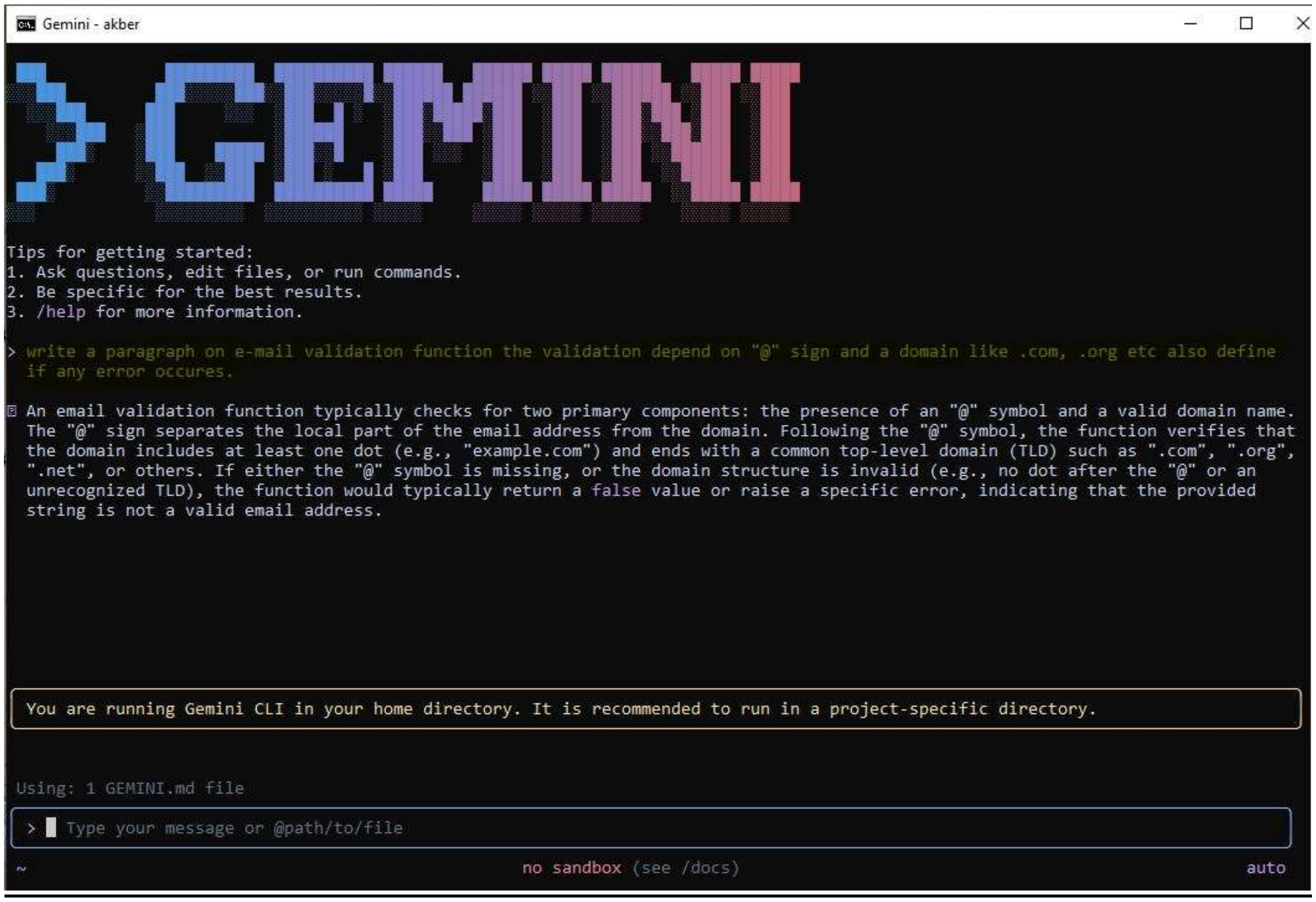
Q5: How does architecture-first thinking change the role of a developer in AIDD?

Answer: In traditional coding, a developer's job is mostly "write working code fast." In AIDD with architecture-first thinking, the developer becomes the conductor of an orchestra. You decide the overall structure (layers, modules, boundaries), write the high-level specifications, and then let AI agents fill in lower-level details. Your main job shifts from typing code to designing systems, defining contracts, and orchestrating agents exactly like a real software architect or tech lead.

Q6: Explain why developers must think in layers and systems instead of raw code.

Answer: Raw code is just the final artifact. If you only think in raw code, you easily create tightly coupled, untestable, hard-to-change systems. Thinking in layers (Model → IDE/workspace → Agents) and systems forces you to separate concerns: intelligence (core logic), workspace (tools and environment), and orchestration (how agents cooperate). This separation makes the project scalable, replaceable, and maintainable even when AI writes 80 to 90 percent of the actual lines. In the AI era, the human's value is in designing the system, not in typing the most lines.

🗀 Part B — Practical Task



```
Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. /help for more information.

> write a paragraph on e-mail validation function the validation depend on "@" sign and a domain like .com, .org etc also define
  if any error occures.

✦ An email validation function typically checks for two primary components: the presence of an "@" symbol and a valid domain name.
  The "@" sign separates the local part of the email address from the domain. Following the "@" symbol, the function verifies that
  the domain includes at least one dot (e.g., "example.com") and ends with a common top-level domain (TLD) such as ".com", ".org",
  ".net", or others. If either the "@" symbol is missing, or the domain structure is invalid (e.g., no dot after the "@" or an
  unrecognized TLD), the function would typically return a false value or raise a specific error, indicating that the provided
  string is not a valid email address.

 You are running Gemini CLI in your home directory. It is recommended to run in a project-specific directory.

 Using: 1 GEMINI.md file
 > ▌ Type your message or @path/to/file
 ~                         no sandbox (see /docs)                         auto
```

🗀Part C — Multiple Choice Questions

Correct Answers

1. ✅ B. Clear requirements before coding begins
2. ✅ B. Thinking in systems and clear instructions
3. ✅ B. Architecture becomes hard to extend
4. ✅ B. Handle repetitive tasks so dev focuses on design & problem-solving
5. ✅ C. Deep skills in multiple related domains

🗀Task completed by

Ghulam Akber

Role No: 00231794

Class Slot: Friday 6:00 PM to 9:00 PM