AeroAspire - SDE Intern

Gokul Krishna S

Week 3 – Day 4 (October 10)

Questions/Reflections :

1. What's OpenAPI / Swagger? Why document APIs?

    - OpenAPI is a machine-readable format—usually written in YAML or JSON - that describes the structure of an API, including endpoints, request/response formats, authentication, and more.
    - Swagger is best known for tools like Swagger UI (interactive docs) and Swagger Codegen (which generates client/server code), making API development much faster and more consistent.
    - These tools make it easy for both humans and machines to "see" how an API works, letting teams share, test, and even auto-generate parts of the API.

    Why Document APIs?

    - Good documentation makes it much easier for others to use the API, reducing miscommunication, mistakes, and the time it takes for teams or external clients to integrate.
    - It boosts adoption—people are far more likely to use a well-documented API than one that's opaque.
    - It also lowers the ongoing support burden since well-made docs answer common questions, and reduces costs by speeding up developer onboarding and troubleshooting.

2. How versioning helps backward compatibility; what versioning strategies exist (URI versioning, header versioning etc.).

    - Versioning helps manage changes safely—old clients can still use v1, while new features (or breaking changes) roll out in v2 or later.

- This means that when an API needs a redesign or major updates, existing users don't get their apps unexpectedly broken, ensuring smooth transitions and happy users.
- It allows teams to innovate and add features freely without forcing instant upgrades or risking backward incompatibility.
- Versioning Strategies
- URI Path Versioning: Add version in the endpoint's URL, e.g. /api/v1/users—this is the most common and visible approach, especially for public APIs.
- Query Parameter Versioning: Add a version query string, e.g. /api/users?version=1.0, which is easy to test but can complicate routing.
- Header Versioning: Specify the API version in a custom HTTP header, e.g. api-version: 1, making URLs cleaner but requires header handling on both client and server.
- Accept Header Versioning: Use the Accept header to indicate versioned media types; allows for fine-grained/content negotiation, but is harder to test in browsers and less obvious to users.

3. If deploying to cloud (or staging), what environment configs will you need?

- When deploying to the cloud or a staging environment, typical environment configs you will need include:
- **Environment Variables:** For database connection info, API keys, secrets, third-party credentials, and custom settings per environment (use .env files or equivalent in your cloud platform).
- **Service URLs:** Backend, frontend, and other resource URLs should point to staging endpoints, not production.
- **Logging/Monitoring:** Enable suitable levels of logging—errors or verbose logging for staging, minimal for production.
- **Debug/Release Flags:** Feature toggles or debug options to control release or testing features per environment.
- **Storage Configurations:** Bucket paths, CDN endpoints, or external storage keys tailored for staging or cloud.

- **OAuth or Auth Configs:** Use different client IDs/secrets or redirect URIs for staging versus production.
- **Deployment Variables:** Resource allocation—VM size, container image tags, scaling settings, etc.
- **Network Settings:** Allowed IPs, network security groups, or firewall rules may differ between environments.