Assignment: Scalable URL Shortener Service

Objective

Design and implement a scalable URL shortener service. A user interface (UI) is optional but will be considered a plus. Your implementation should be suitable for handling a high volume of traffic and storing large datasets efficiently.

Core Functional Requirements

The service must be able to:

1. **Shorten a URL** – Accept a long URL and generate a unique shortened version.
2. **Expand a URL** – Resolve a shortened URL to its original form and redirect the user.
3. **Manage Expiry** – Shortened URLs should automatically expire after a configurable retention period.

Technical & Architectural Requirements

C4 Architecture (Container Level)

Provide a hand-drawn or modeled C4 **Container** diagram that clearly illustrates:

1. System context
2. Main containers (e.g., web/API server, database, caching layer)
3. Interactions and responsibilities between containers
4. Key technologies used within each container

Containerized Deployment

Your application should be fully containerized and able to run in either:

1. A **Docker** environment with appropriate Dockerfile and docker-compose.yml (if needed), or
2. A **Kubernetes** environment with deployment manifests and service configurations.

Implementation Expectations

A satisfactory solution should demonstrate:

1. **High Load Tolerance** – Efficient request handling with consideration for performance under scale.

2. **Security Measures** – Appropriate security precautions must be implemented for all shortened URLs. You are responsible for determining and justifying the specific measures based on your design decisions.
3. **Optimized Storage** – Think how you can optimize and minimize the storage footprint.
4. **Scalability Plan** – Be ready to explain how your solution would scale horizontally.
5. **[Optional] Monetization Strategy** – Describe how the service could generate revenue.

Development Guidelines

For .NET Developers

1. **Language & Framework:** Use **C# with ASP.NET Core**.
2. **Recommended Tools:**
3. .NET 6 or later

For Java Developers

1. **Language & Framework:** Use **Java with Spring Boot**.
2. **Recommended Tools:**
3. Java 17 or later

Additional Notes

- **Testing & Coverage:** Full unit test coverage for all critical components is expected. Test cases should cover core logic, edge cases, and failure scenarios.
- **Documentation:** Please provide accompanying documentation that includes:
    - Project overview and features
    - Setup instructions (local, Docker, and/or Kubernetes)
    - API endpoints with sample requests and responses
    - Expiry and storage behavior
    - Design decisions, especially regarding scalability and security
- **Versioning:** Follow Semantic Versioning (MAJOR.MINOR.PATCH) for your codebase. Clearly indicate the current version of your implementation in the documentation or project metadata.
- **Git Commit Standards:** Use Conventional Commits to maintain consistent and meaningful commit messages. This includes commit prefixes such as feat:, fix:, chore:, refactor:, docs:, etc., to improve readability and changelog generation.
- **Live Demo:** A hosted or locally runnable demo is appreciated but not mandatory. Include demo instructions if applicable.