Spring Boot Caching and Exception Handling Lab

Objectives:

- Implement caching strategies to improve application performance.
- Use **Redis** as a distributed caching solution.
- Integrate **Caffeine** as an in-memory cache.
- Handle exceptions effectively in a Spring Boot application.

---

1. Caching in Java

In-Memory Caching using Caffeine:

Caffeine has been integrated to provide in-memory caching with a **10-minute expiration** and **maximum size of 1000 entries**. This cache is suitable for frequently accessed data.

Example Usage in ChefService:

- @Cacheable: Fetches data from the cache or stores it if not present.
- @CachePut: Updates the cache after data modification.
- @CacheEvict: Removes the entry from the cache when data is deleted.

---

2. Redis Integration

Redis as Distributed Cache:

Redis is used as a distributed cache for data persistence across different services. The Spring Boot project integrates Redis for efficient caching across multiple instances.

Redis Configuration:

- Redis configurations such as host, port, and timeouts have been externalized in the application.properties file, ensuring flexibility for different environments.

---

3. Caching Strategies

Implemented Strategies:

- **Write-Through**: When a new chef or task is saved, it's written directly to the cache and database.
- **Read-Through**: When retrieving chefs or tasks, the cache is checked first to avoid hitting the database.
- **Write-Behind** (optional): Future enhancements can allow background writing to Redis.

---

4. Advanced Exception Handling

Custom Exception Handling:

- **Custom Exceptions**: Handled using @ControllerAdvice to provide clean and consistent error messages.
- **Global Exception Handling**: Unhandled exceptions are caught and returned as 500 Internal Server Error responses.

Example:

The ChefNotFoundException is thrown if a chef is not found in the database. The GlobalExceptionHandler catches this and sends a 404 Not Found response.