Circuit Breaker Case Study

Case Study 2: User Profile Service with Caching 💾

The Scenario: A social media platform, "Connectly," has a **User Profile Service** that fetches user

data from a database. Another service, the **Timeline Service**, constantly calls the User Profile

Service to fetch data for displaying posts. The database is a single point of failure and can

sometimes become slow or unreachable. If the database goes down, the User Profile Service will

fail, causing the Timeline Service to fail, and so on.

The Solution with a Circuit Breaker and Fallback Cache: Connectly implements a circuit

breaker to protect against database failures.

• **Closed State**: The circuit breaker is **closed**, and the User Profile Service fetches data

directly from the database.

• **Threshold Trigger**: The database experiences a period of high load, causing requests to the

User Profile Service to timeout. The circuit breaker detects this based on its configured

threshold (e.g., **90%** of calls failing).

• **Open State**: The circuit breaker opens, preventing any more requests from hitting the

database.

• **Fallback with a Cache**: In the fallbackMethod, the User Profile Service does not

return a generic error. Instead, it attempts to retrieve the requested user data from a **local**

cache (e.g., Redis or an in-memory cache).

◦ If the data is found in the cache, it's returned to the Timeline Service. This allows the

Timeline Service to continue displaying timelines with slightly outdated but still

usable data.

- If the data is not in the cache, it returns a generic fallback response.

• **Half-Open State**: After a set time, the circuit breaker transitions to the **half-open state**,

allowing a few requests to test the database connection.

• **Recovery**: If the database has recovered, the test requests succeed, the circuit breaker closes,

and the User Profile Service returns to fetching fresh data from the database.

Complete Code Implementation

1. UserProfileServiceApplication.java

```
package com.example.userProfile;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UserProfileServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserProfileServiceApplication.class, args);
    }
}
```

2. UserProfile.java

```
package com.example.userProfile.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class UserProfile {
    @Id
    private Long id;
    private String username;
    private String fullName;
    private String bio;

    // Constructors, Getters, Setters
    public UserProfile() {}

    public UserProfile(Long id, String username, String fullName, String bio) {
        this.id = id;
        this.username = username;
        this.fullName = fullName;
        this.bio = bio;
    }

    // Getters and Setters
}
```

3. UserProfileRepository.java

```
package com.example.userProfile.repository;

import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.userProfile.entity.UserProfile;

public interface UserProfileRepository extends JpaRepository<UserProfile, Long>
{
    Optional<UserProfile> findByUsername(String username);
}
```

4. UserCache.java

```
package com.example.userProfile.cache;

import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Component;
import com.example.userProfile.entity.UserProfile;

@Component
public class UserCache {
    private Map<String, UserProfile> cache = new HashMap<>();

    public void save(UserProfile profile) {
        cache.put(profile.getUsername(), profile);
    }

    public UserProfile get(String username) {
        return cache.get(username);
    }
}
```

5. UserProfileService.java

```
package com.example.userProfile.service;

import com.example.userProfile.cache.UserCache;
import com.example.userProfile.entity.UserProfile;
import com.example.userProfile.repository.UserProfileRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserProfileService {

    @Autowired
    private UserProfileRepository repository;

    @Autowired
    private UserCache cache;

    @CircuitBreaker(name = "userProfileCB", fallbackMethod =
```

```
"getUserProfileFromCache")
    public UserProfile getUserProfile(String username) {
        simulateDatabaseIssue(username);
        UserProfile profile = repository.findByUsername(username)
                .orElseThrow(() -> new RuntimeException("User not found"));
        cache.save(profile);
        return profile;
    }

    @PostConstruct
    public void init() {
        repository.save(new UserProfile(1L, "mina", "Minakshi Patel", "Java
developer from Pune"));
    }

    public UserProfile getUserProfileFromCache(String username, Throwable ex) {
        UserProfile cached = cache.get(username);
        if (cached != null) return cached;

        UserProfile fallback = new UserProfile();
        fallback.setUsername(username);
        fallback.setFullName("Unavailable");
        fallback.setBio("Data temporarily unavailable");
        return fallback;
    }

    private void simulateDatabaseIssue(String username) {
        if (username.equalsIgnoreCase("error")) {
            throw new RuntimeException("Simulated DB failure");
        }
    }
}
```

6. UserProfileController.java

```
package com.example.userProfile.controller;

import com.example.userProfile.entity.UserProfile;
import com.example.userProfile.service.UserProfileService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
public class UserProfileController {

    @Autowired
    private UserProfileService service;

    @GetMapping("/{username}")
    public UserProfile getUserProfile(@PathVariable String username) {
        return service.getUserProfile(username);
    }
}
```

7. application.properties

```
server.port=8080

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

management.endpoints.web.exposure.include=*
```

8. pom.xml (Dependencies)

```
<!-- Add these inside <dependencies> -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-spring-boot3</artifactId>
    <version>2.1.0</version>
</dependency>
```