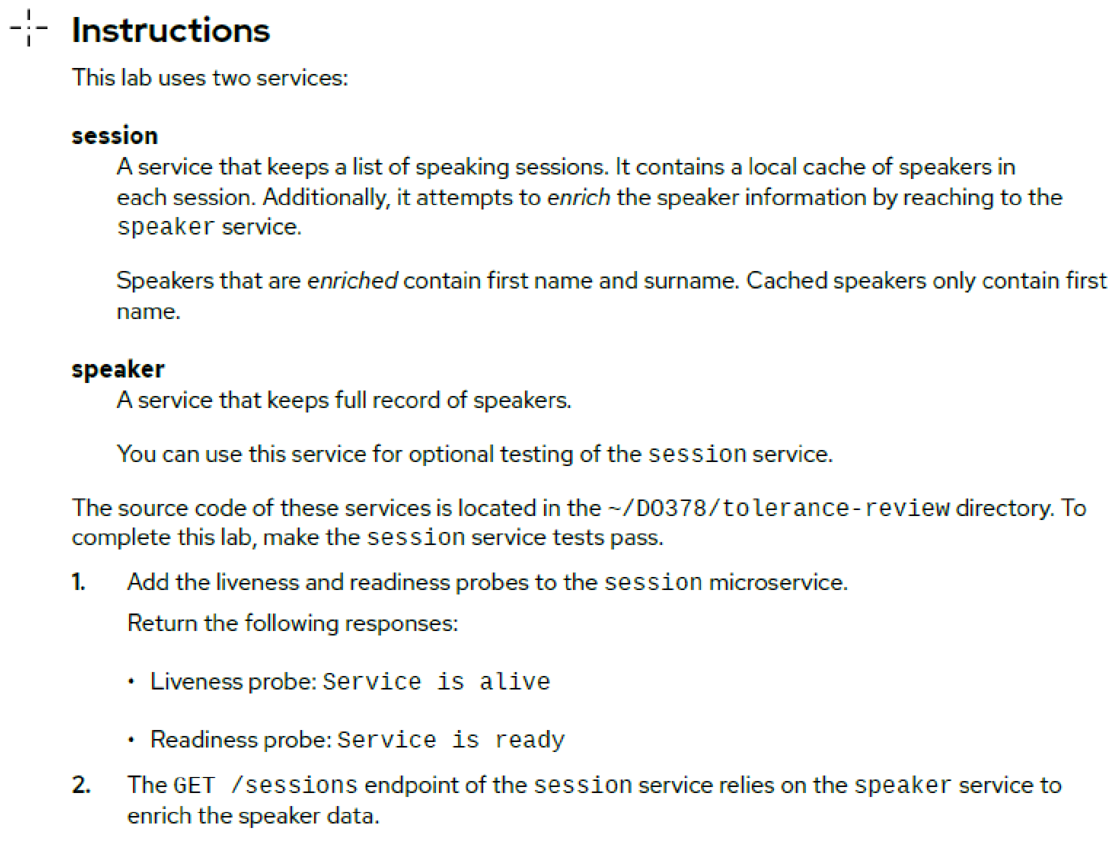
LAB 20: QUARKUS TOLERANCE REVIEW

Autor: José Díaz

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

Abre el proyecto **tolerance-review**



Instructions

This lab uses two services:

session

A service that keeps a list of speaking sessions. It contains a local cache of speakers in each session. Additionally, it attempts to *enrich* the speaker information by reaching to the `speaker` service.

Speakers that are *enriched* contain first name and surname. Cached speakers only contain first name.

speaker

A service that keeps full record of speakers.

You can use this service for optional testing of the `session` service.

The source code of these services is located in the `~/DO378/tolerance-review` directory. To complete this lab, make the `session` service tests pass.

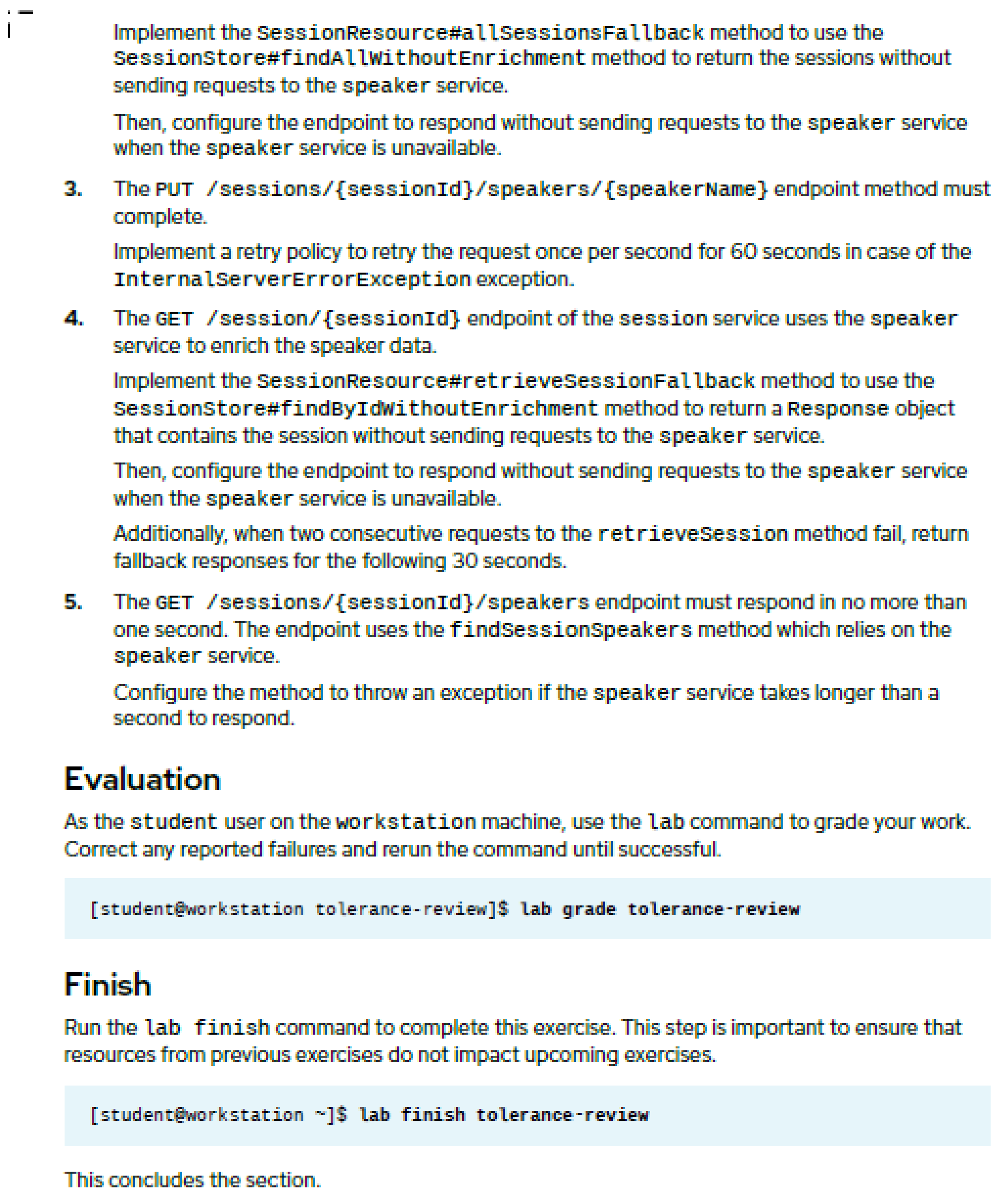
1. Add the liveness and readiness probes to the `session` microservice.

   Return the following responses:

   - Liveness probe: `Service is alive`
   - Readiness probe: `Service is ready`

2. The `GET /sessions` endpoint of the `session` service relies on the `speaker` service to enrich the speaker data.

.



Implement the `SessionResource#allSessionsFallback` method to use the `SessionStore#findAllWithoutEnrichment` method to return the sessions without sending requests to the `speaker` service.

Then, configure the endpoint to respond without sending requests to the `speaker` service when the `speaker` service is unavailable.

3. The `PUT /sessions/{sessionId}/speakers/{speakerName}` endpoint method must complete.

   Implement a retry policy to retry the request once per second for 60 seconds in case of the `InternalServerErrorException` exception.

4. The `GET /session/{sessionId}` endpoint of the `session` service uses the `speaker` service to enrich the speaker data.

   Implement the `SessionResource#retrieveSessionFallback` method to use the `SessionStore#findByIdWithoutEnrichment` method to return a `Response` object that contains the session without sending requests to the `speaker` service.

   Then, configure the endpoint to respond without sending requests to the `speaker` service when the `speaker` service is unavailable.

   Additionally, when two consecutive requests to the `retrieveSession` method fail, return fallback responses for the following 30 seconds.

5. The `GET /sessions/{sessionId}/speakers` endpoint must respond in no more than one second. The endpoint uses the `findSessionSpeakers` method which relies on the `speaker` service.

   Configure the method to throw an exception if the `speaker` service takes longer than a second to respond.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation tolerance-review]$ lab grade tolerance-review
```

Finish

Run the `lab finish` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish tolerance-review
```

This concludes the section.

SOLUCIÓN:

Instructions

This lab uses two services:

session
: A service that keeps a list of speaking sessions. It contains a local cache of speakers in each session. Additionally, it attempts to *enrich* the speaker information by reaching to the `speaker` service.

  Speakers that are *enriched* contain first name and surname. Cached speakers only contain first name.

speaker
: A service that keeps full record of speakers.

  You can use this service for optional testing of the `session` service.

The source code of these services is located in the `~/DO378/tolerance-review` directory. To complete this lab, make the `session` service tests pass.

1. Add the liveness and readiness probes to the `session` microservice.

   Return the following responses:

   - Liveness probe: `Service is alive`
   - Readiness probe: `Service is ready`

   1.1. Change into the `~/DO378/tolerance-review/session` directory.

```
[student@workstation ~]$ cd ~/DO378/tolerance-review/session
```

1.2. In an IDE of your choice, open the `session` project. Then, in the `src/main/java/com/redhat/training/conference/session/LivenessCheck.java` file, implement the `HealthCheck` interface.

Finally, add the `@Liveness` annotation.

```
@Liveness
@ApplicationScoped
public class LivenessCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        return HealthCheckResponse.up("Service is alive");
    }
}
```

1.3. In the `src/main/java/com/redhat/training/conference/session/ReadinessCheck.java` file, implement the `HealthCheck` interface.

Finally, add the `@Readiness` annotation.

```
@Readiness
@ApplicationScoped
public class ReadinessCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        return HealthCheckResponse.up("Service is ready");
    }
}
```

1.4. Verify that the `testLivenessProbe` and `testReadinessProbe` tests pass.

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testLivenessProbe,SessionResourceTest#testReadinessProbe
...output omitted...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

2. The GET `/sessions` endpoint of the `session` service relies on the `speaker` service to enrich the speaker data.

Implement the `SessionResource#allSessionsFallback` method to use the `SessionStore#findAllWithoutEnrichment` method to return the sessions without sending requests to the `speaker` service.

Then, configure the endpoint to respond without sending requests to the `speaker` service when the `speaker` service is unavailable.

2.1. Open the `src/main/java/com/redhat/training/conference/session/SessionResource.java` file. Then, implement the `allSessionsFallback` method.

```
public Collection<Session> allSessionsFallback() throws Exception {
    logger.warn("Fallback for GET /sessions");
    return sessionStore.findAllWithoutEnrichment();
}
```

2.2. Use the `@Fallback` annotation to configure the `allSessions` method to use the `allSessionsFallback` method during failures.

```
@GET
@Fallback(fallbackMethod="allSessionsFallback")
public Collection<Session> allSessions() throws Exception {
  return sessionStore.findAll();
}
```

2.3. Verify that the `testAllSessionsFallback` test passes.

```
[student@workstation session]$ mvn clean test \
  -Dtest=SessionResourceTest#testAllSessionsFallback
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

3. The PUT `/sessions/{sessionId}/speakers/{speakerName}` endpoint method must complete.

Implement a retry policy to retry the request once per second for 60 seconds in case of the `InternalServerErrorException` exception.

3.1. Add the `@Retry` annotation to both endpoint methods. Use the `maxRetries` and `delay` options to configure a retry per second for 60 seconds.

```
...class omitted...

@PUT
@Path("/{sessionId}/speakers/{speakerId}")
@Transactional
@Retry(maxRetries=60, delay=1_000, retryOn=InternalServerErrorException.class)
public Response addSessionSpeaker(@PathParam("sessionId") final String sessionId,
 @PathParam("speakerName") final String speakerName) {
...class omitted...
```

3.2. Verify that the `testAddSpeakerToSession` test passes.

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testAddSpeakerToSession
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

4. The GET `/session/{sessionId}` endpoint of the `session` service uses the `speaker` service to enrich the speaker data.

Implement the `SessionResource#retrieveSessionFallback` method to use the `SessionStore#findByIdWithoutEnrichment` method to return a `Response` object that contains the session without sending requests to the `speaker` service.

Then, configure the endpoint to respond without sending requests to the `speaker` service when the `speaker` service is unavailable.

Additionally, when two consecutive requests to the `retrieveSession` method fail, return fallback responses for the following 30 seconds.

4.1. Implement the `retrieveSessionFallback` method.

```
public Response retrieveSessionFallback(final String sessionId) {
  logger.warn("Fallback for GET /sessions/"+sessionId);
  return sessionStore.findByIdWithoutEnrichment(sessionId)
    .map(s -> Response.ok(s).build())
    .orElseThrow(NotFoundException::new);
}
```

4.2. Use the `@Fallback` annotation to configure the `retrieveSession` method to use the `allSessionsFallback` method during failures.

Additionally, use the `@CircuitBreaker` annotation to use the fallback method after two failures.

```
@GET
@Path("/{sessionId}")
@Fallback(fallbackMethod="retrieveSessionFallback")
@CircuitBreaker(requestVolumeThreshold = 2, failureRatio = 1, delay = 30_000)
public Response retrieveSession(@PathParam("sessionId") final String sessionId) {
```

4.3. Verify that the `testSessionCircuitBreaker` test passes.

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testSessionCircuitBreaker
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

5. The `GET /sessions/{sessionId}/speakers` endpoint must respond in no more than one second. The endpoint uses the `findSessionSpeakers` method which relies on the `speaker` service.

Configure the method to throw an exception if the `speaker` service takes longer than a second to respond.

5.1. Use the `@Timeout` annotation on the `findSessionSpeakers` method. Use a parameter value of 1000 milliseconds.

```
@Timeout(1000)
public Optional<Session> findSessionSpeakers(String sessionId) {
```

5.2. Verify that the `testSessionSpeakerFallback` test passes.

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testSessionSpeakerFallback
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation tolerance-review]$ lab grade tolerance-review
```

Finish

Run the `lab finish` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish tolerance-review
```

This concludes the section.

enjoy!

Jose