URL checker

Overview

The goal is to develop a software component that manages a simple database, essentially consisting in a registry of URLs, and which regularly check these URL to asses if they are still accessible. This module will be used in a larger setup in which other components will register URLs to this component and will be notified when their registered URL is dead (through an asynchronous messaging system, such as Kafka)

Main features summary

It should offer the following external API:

- Registering a new URL along with an associated object ID reference
- Unregistering a previously registered URL and associated ID
- Retrieve the IDs associated with an URL
- Get the current stats

And it should have the following features:

- Manage possibly multiple reference IDs for each URL and remove the URL only when all references have been unregistered
- Have a scheduled system that ensure that each URL is checked every predefined time periods of several days (check period)
- Publish events to a Kafka topic each time a URL goes to the “dead” state, the event should contains the URL, the associated IDs and some customisable informations about the error
- The condition used to check an URL should be conf (it can be hard-coded but should be easy to change)
- When a URL returns a failure, the system must recheck it every “recheck period” until it reaches a “recheck threshold”, moment at which it becomes “dead” and a corresponding event is published

See also URL_check.png for system overview diagram and URL_state.png for the state transition diagram associated to each URL.

Detailed processes description

The module will keep a state associated with each URL with at least three value (“alive”, “potentially dead” or “dead”) as well as additional informations such as the time of the last checks or the detailed error(s) contents of a “potentially dead” or a “dead” URL.
The module checks each registered URLs periodically with a configurable “check period”. Ideally, the time at which an URL must be checked again should be managed for each URL independently. Hence, if we have a “check period” of 10 days and URL-1 has been checked on the 2.11 and URL-2 checked on the 5.11, then the system must recheck URL-1 on the 12.11 and URL-2 on the 5.11

When the module checks an URL:

- if it returns a successful HTTP response (200), then the URL goes or stay in the “alive” state and nothing else is done (except that the URL will be rechecked when the “check period” has passed)
- If it returns a bad HTTP response (code != 200), then the URL becomes “potentially dead” accompanied with a counter indicating the number of failures (initialised to 1)

Every “recheck period”, all “potentially dead” URL are checked again and either their counter get incremented, or they come back in the “alive” state.
If a “potentially dead” counter reach a custom “recheck threshold”, then the URL becomes “dead” and an event including the URL, the related IDs and the HTTP error details, is published on Kafka. “Dead” URL are not checked anymore (unless it gets fully unregistered and then re-registered).

Technology

Ideally, the component should be designed as independent micro-service with its own REST API and should be built using Scala along with the Akka, Play & Lagom frameworks. The external components identified for this system are a database to store the URL and associated data (preferably PostgreSQL) and a asynchronous messaging system (preferably Kafka)
Other solution using potentially different technologies could also be considered if they cover the needs correctly. In any case, it should run on a JVM.

Please details in your quote which technology you plan to use and a general description of how you plan to manage the automated periodical checking. I also consider solutions that slightly vary from the given description as long as the main features idea is present.