VRIJE
UNIVERSITEIT
BRUSSEL

SOFTWARE ARCHITECTURES

Report assignment 1

Christophe Van den Eede

December 28, 2021

Sciences and Bio-Engineering Sciences

1 Introduction

In this assignment we were instructed to request and process the Npm packages listed in a given text file contained in an archive. We should do this using Akka Streams. In this report I will cover the design decisions made for this project.

2 Program flow

In this section I will explain the separate parts of the program flow, as they do not 100% conform to the specification.

- First a combined flow is created following an example we saw in the lab to unzip the tarball and make a stream of the contained strings.
- Each of these strings is mapped to a NpmPackage object, which for now only contains the name of the package and an empty list of versions.
- A backpressure buffer of size 10 is implemented here as per specification.
- This is followed by the flow being throttled to one package per 3 seconds. This is so that the Npm website will not start refusing our requests.
- for every package, the fetchDependencies method is called. This method will, internally in the package object, send a request to retrieve all different versions of that package. After this it will fill its versionlist with new Version objects for each version listed in the json result. These Version objects in turn unpack all of the runtime and dev dependencies they have and put them into a single list of Depencency objects. These objects only containt the NpmPackage they belong to, the version of NpmPackage they belong to and their own type of dependency (runtime/dev).
- After this the flow will be mapped to contain all of the versions inside of each NpmPackage. Note that these Version objects containt context as to which NpmPackage they belong to.
- Another backpressure buffer of size 12 is implemented here as per specification.
- Here the Versions enter the custom flow graph where all Versions are balanced out over 2 instances of *another* custom flow graph.
- This other custom flow graph broadcasts the Version to two filters, one creates a DependencyCount object of the runtime dependencies and the other of the dev dependencies. These DependencyCount objects contain the NpmPackage name, the version of this package and the amount of runtime and dev dependencies. Though in the output of these filters only the runtime or dev dependencies are filled, depending on which filter outputs them. The two flows are zipped together at the end of this custom flow graph.
- Right after this, the zipped DependencyCounts are merged to a single DependencyCount containing the counts from both sides.
- Then the results from the 2 flows are merged and we exit the bigger custom flow graph.
- Now we have a flow containing a lot of DependencyCounts, we could print them out now, however to create the same output as in the specification this means we need to somehow guarantee the order in which the packages reach the Sink. To solve this problem, I implemented a fold procedure to accumulate every single DependencyCount and add them to a map of maps. These maps allow us to Search for a specific package, followed by looking for all their versions and receive their dependency counts.
- now in the sink this allows us to loop over all NpmPackages, followed by looping over all their Versions and printing their Dependencies.