



Projet, partie 1 : compositeur de menus simplistes

Thibaut Septon

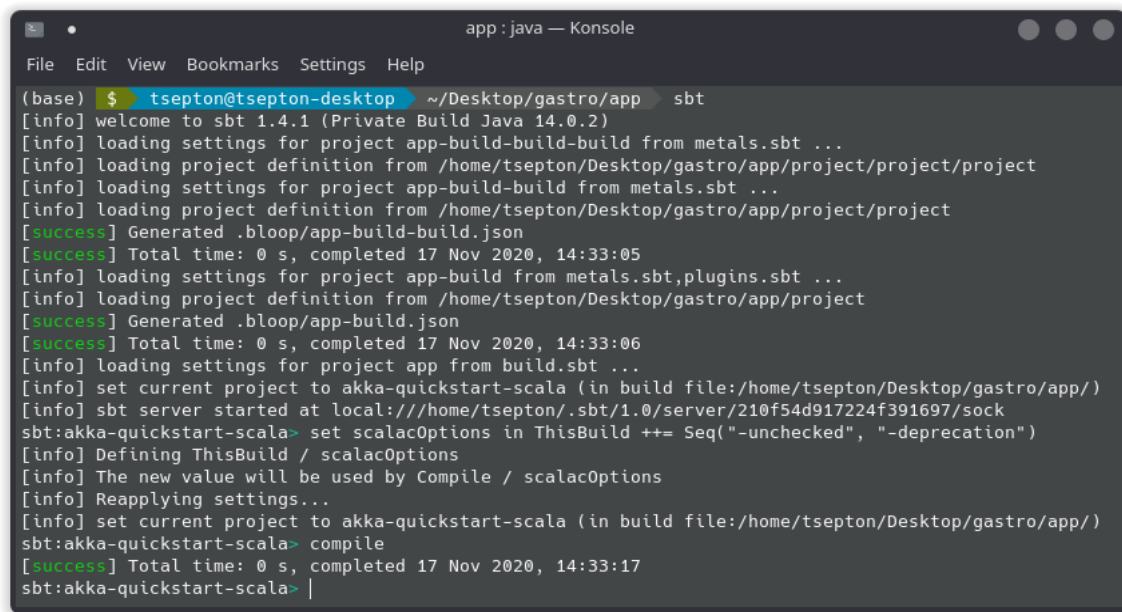
[INFOM451] Conception d'applications mobiles

November 17, 2020

Jacquet Jean-Marie, Vanhoof Wim

Table des matières

Introduction	2
Logique	2
Division	2
Communication entre les acteurs	3
Améliorations de l'algorithme	3
Remarques	3



```
app:java — Konsole
File Edit View Bookmarks Settings Help
(base) $ tsepton@tsepton-desktop ~/Desktop/gastro/app sbt
[info] welcome to sbt 1.4.1 (Private Build Java 14.0.2)
[info] loading settings for project app-build-build from metals.sbt ...
[info] loading project definition from /home/tsepton/Desktop/gastro/app/project/project/project
[info] loading settings for project app-build-build from metals.sbt ...
[info] loading project definition from /home/tsepton/Desktop/gastro/app/project/project/project
[success] Generated .bloop/app-build-build.json
[success] Total time: 0 s, completed 17 Nov 2020, 14:33:05
[info] loading settings for project app-build from metals.sbt,plugins.sbt ...
[info] loading project definition from /home/tsepton/Desktop/gastro/app/project
[success] Generated .bloop/app-build.json
[success] Total time: 0 s, completed 17 Nov 2020, 14:33:06
[info] loading settings for project app from build.sbt ...
[info] set current project to akka-quickstart-scala (in build file:/home/tsepton/Desktop/gastro/app/)
[info] sbt server started at local:///home/tsepton/.sbt/1.0/server/210f54d917224f391697/sock
sbt:akka-quickstart-scala> set scalacOptions in ThisBuild += Seq("-unchecked", "-deprecation")
[info] Defining ThisBuild / scalacOptions
[info] The new value will be used by Compile / scalacOptions
[info] Reapplying settings...
[info] set current project to akka-quickstart-scala (in build file:/home/tsepton/Desktop/gastro/app/)
sbt:akka-quickstart-scala> compile
[success] Total time: 0 s, completed 17 Nov 2020, 14:33:17
sbt:akka-quickstart-scala> |
```

Figure 1: La compilation n'émet aucun *Warning*.

Introduction

Etant donné que je ne suis pas un expert en alimentation, et que l'objectif transversale du projet était de montrer la compréhension des concepts du langage Scala (à savoir un mélange de fonctionnel et d'orienté objet), mon script pourra vous paraître simpliste.

En effet, il ne se base que sur le nombre de calories présent dans le repas afin de déterminer si celui-ci convient ou pas. Cependant, j'ai préféré garder un programme simple afin de mettre en oeuvre les deux paradigmes de Scala et ainsi mieux appréhender l'apprentissage de ce langage, tout en appliquant les divers consignes du mieux que je le pouvais.

Le code étant lisible et commenté, je ne m'attarderai pas sur l'aspect technique du programme dans ce rapport, mais seulement sur la partie logique.

Logique

Division

Afin de garder au maximum un code propre et lisible, et ainsi d'éviter d'avoir un fichier de 1000 lignes, j'ai décidé de le scinder en plusieurs parties. Ainsi les différents packages sont :

- Main
- Menu (contient tout ce qui est relatif à un menu)
- Waiter (est ce que vous avez appelé "master" au sein des consignes)
- Coq
- Intendant
- Dispensers

La logique respecte vos consignes, cependant, afin de mieux schématiser la situation, j'ai réalisé le schéma de la figure 2.

Communication entre les acteurs

Je suis parti du principe que nous allions reconstituer un restaurant classique. Ainsi, nous avons le serveur (*Waiter*) qui prend les commandes auprès des clients (nous donc) et qui transmet ces commandes à la cuisine, c'est à dire au chef. Ensuite le chef se charge de préparer chaque plat. C'est pourquoi nous avons la classe *Meal*, qui représente, pour chaque commande, sa préparation en cuisine. Le chef demande dans un premier temps un ingrédient de base à son intendant (*Intendant*), puis demande aux commis de cuisines (*Dispensers*) de lui ramener des ingrédients afin de compléter le plat. Toutes les communications entre le chef et ses délégués se font via une demande *ask* de Akka. Seuls les communication entre le chef et son serveur se font via un *tell*.

Améliorations de l'algorithme

L'algorithme de sélection d'ingrédients (décrit dans le premier rapport, avec un système de calories et en fonction du type de repas et du sexe) n'est pas réellement amélioré. En effet, je préférais réaliser une implémentation propre du système d'acteur en gardant une architecture proche d'une cuisine plutôt que de m'attarder sur la sélection d'ingrédient puisque, selon moi, cette partie était moins intéressante (de plus que je ne savais pas sur quoi me baser pour réaliser ma sélection d'ingrédients). Cependant, avec cette nouvelle implémentation, nous retrouverons dans les plats préparés, des ingrédients contenant un minimum de sucre, de protéines et de gras tout en s'assurant que les calories totales ne sont pas dépassées en fonction du repas (déjeuner, dîner ou souper).

Concernant les quantités, le fichier *.csv* n'était pas fort fourni. C'est la raison pour laquelle, si vous décidez de lancer 100 plats, vous n'aurez des quantités indiquées que pour très peu d'ingrédients (à 2%).

Remarques

Je n'ai pas rédigé de commentaires (autre ceux indiquant les différentes utilisations des méthodes propre à scala demandés) puisque un code se doit d'être compréhensible en le lisant via les différents choix de nom de variables, de fonctions, etc. De plus, pour un code susceptible d'évoluer dans le temps, les commentaires ne sont que trop rarement modifiés pour respecter ce qu'ils documentent. Ainsi, j'espère que vous apprécierez mon effort de rendre le code le plus lisible et compréhensible possible (j'ai notamment tenté de respecter les conventions de nommage propre à Scala).

J'ai nottament implémenté un moyen simple de tester plusieurs commandes à la fois (générées de façon aléatoire) afin de prouver que la coordination entre les différents acteurs fonctionnent. Il vous suffira de taper 2 au premier input demandé par le programme. Pour avoir testé pour 25.000 plats à la fois, je n'ai pas relevé de problème. Cependant, pour une raison que je ne parviens pas à comprendre, si vous demander trop de commande, le chef commencera à recevoir de mauvaises réponses de la part de ses commis (sans pour autant crasher, puisque la gestion des erreurs se fait correctement).

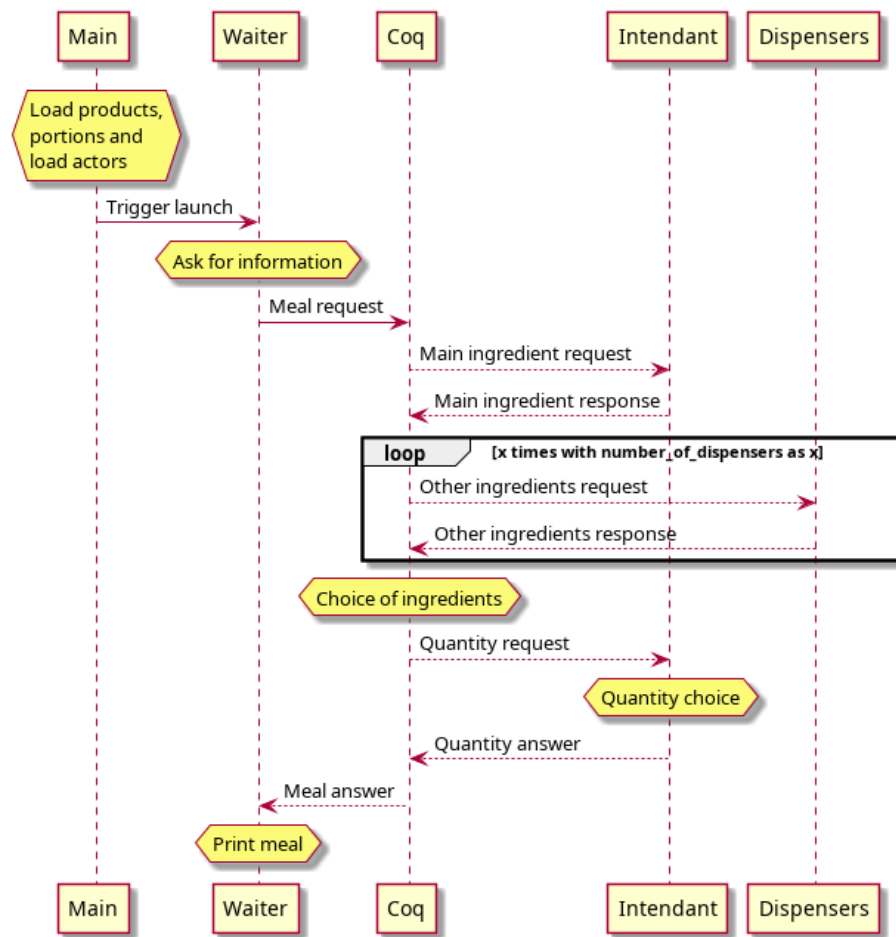


Figure 2: Diagramme de séquence