INFOM451 Conception d'applications mobile

Projet Gastro

Partie 2 : futurs acteurs gustatifs exceptionnels

Valentin Arioti

15 novembre 2022

1 Principe général

Ce nouveau générateur propose à l'utilisateur un **menu équilibré** (environ 25% de protéines, 40% de glucides et 35% de lipides) en fonction du **nombre de kilocalories** dont il a besoin. Le programme affiche d'abord les informations utiles à l'utilisateur, ensuite l'utilisateur peut demander autant de menus qu'il le souhaite en indiquant un nombre. Il n'a pas besoin d'attendre qu'un menu précédent soit prêt pour en demander un nouveau. Et si un problème a lieu durant la préparation, le programme prévient l'utilisateur mais ne s'arrête pas, l'utilisateur peut donc continuer à demander des menus (dans certains cas il devra régler le problème pour obtenir des menus).

2 Acteurs

2.1 Main

Main contient le programme principal. Il affiche les informations et recueille l'input de l'utilisateur. À chaque input reçu de l'utilisateur, il prévient le **Coq** avec le nombre de kilocalories souhaitées pour que ce dernier prépare un menu. Après 100 ms, il est de nouveau prêt à recevoir un input. Et sinon, il est toujours à l'écoute d'un message qui contient un menu pour pouvoir l'afficher à l'utilisateur. La communication entre le **Coq** et **Main** se fait via un *tell pattern* (ils se communiquent les informations grâce à la méthode *tell*)

2.2 Coq

Lorsque le **Coq** reçoit une commande, il commence un *ask pattern* avec l'**Intendant** et 3 **Dispensers**. Il vérifie d'abord que l'input de l'utilisateur est bien un entier et si oui, au dessus de 200. En dessous de cette valeur, il est difficile pour le générateur de proposer des menus équilibrés et avec plusieurs aliments car il peut trouver moins de produits. Si l'input ne respecte pas cette condition, alors une valeur par défaut (500) est utilisée.

Le **Coq** demande alors à l'**Intendant** un aliment principal respectant ce nombre de kilocalories. Il attend sa réponse avant de continuer car si l'**Intendant** ne parvient pas à trouver un aliment, c'est que probablement il y a un problème qui rendra impossible la génération de menus.

Une fois que le produit principal est choisi, le **Coq** demande aux 3 **Dispensers** une liste de produits dans lesquels ils sont spécialisés. Ceux-ci travaillent en même temps, et comme le **Coq** reçoit des *Future*, il peut attendre l'étape suivante (où il a besoin de la liste complète) avant de bloquer.

Dans cette étape, le **Coq** essaye de combiner les produits secondaires avec le produit de base de manière à respecter le nombre de calories (avec 75 de marge) et la répartition décrite à la section 1 (avec 8% de marge). Si il n'y arrive pas alors, le **Coq** ne sélectionne aucun produit secondaire et prévient l'utilisateur.

Il demande ensuite à l'**Intendant** les différentes portions pour les produits finalement sélectionnés.

Et pour finir, les produits et les portions sont mis dans la *case class* **Menu** et le **Coq** renvoie à **Main** le menu final sous forme de String, encapsulé dans un *Option*.

2.3 Intendant

L'**Intendant** doit d'abord extraire un produit au hasard dans le fichier csv. Pour être capable de réaliser un menu composé de plusieurs aliments et respectant la contrainte de calories, le produit choisi doit avoir maximum 75% du nombre de kilocalories désiré par l'utilisateur.

La deuxième fonction de l'**Intendant** est de donner une quantité pour chaque produit sélectionné dans le menu. Pour faire cela, il choisi une quantité au hasard pour chaque id de produit présent dans le fichier portions. Lorsqu'il ne trouve pas de quantité pour un produit, il indique 1 comme valeur par défaut. Une fois toutes les quantités définies, l'**Intendant** sélectionne celles correspondant au menu.

2.4 Dispensers

Il y a un **Dispenser** pour chaque type de nutriment important : protéines, glucides et lipides. Chacun sélectionne 50 produits au hasard qui respectent la contrainte suivante : le nutriment dont il est spécialisé doit être plus élevé que la somme des 2 autres.

3 Gestion des erreurs

Dans ce programme, la gestion des erreur a 2 objectifs. Le premier est de faire en sorte que le programme ne s'arrête jamais. Si le **Coq** rencontre un problème majeur, il renvoie *None* au programme principal, qui lui, continue à fonctionner. Le second objectif est d'informer l'utilisateur qu'il y a un problème le plus précisément possible grâce à des messages.

3.1 Problèmes importants

Voici les potentiels problèmes qui mènent à l'arrêt de la génération du menu :

1. Problème lors du choix d'un produit principal : comme le reste de la génération se repose là dessus, il vaut mieux prévenir et ne pas continuer.
2. Exception lors de l'extraction des portions : le programme indique quel est le problème pour que l'utilisateur puisse le corriger.
3. Dans chaque pattern matching (en rapport avec des manipulations du menu), un *case* par défaut va envoyer *None* dans le cas où un problème non envisagé venait à se produire.

3.2 Problèmes mineurs

Certains problèmes sont jugés mineurs et n'entraînent pas un arrêt de la génération de menu :

1. Si l'utilisateur entre un mauvais input : utilisation d'une valeur par défaut (500)
2. Si un **Dispenser** ne parvient pas à extraire ses produits, il envoie une liste vide.
3. Si le **Coq** ne parvient pas à réaliser un menu complet avec le produit principal choisi : il prévient l'utilisateur, mais renvoie tout de même un menu "raté" composé uniquement du produit principal.

3.3 Exemple

Ci-dessous, voici un exemple d'utilisation où le programme a tenté d'extraire les produits alors que le fichier était corrompu. Une fois que le fichier était correct, le programme a réussi a extraire un produit principal. Cependant, il n'a pas pu réaliser un menu complet respectant les contraintes. Il affiche un message pour informer l'utilisateur de ce soucis, et lui envoie quand même le menu "raté". Le programme ne s'arrête jamais.

3.3.1 Output extraction fichier corrompu suivi d'un menu raté

```
I am listening to your order.500
Order received for menu with 500 kcal. Your menu will be ready soon.
AH ! Something wrong happened while extracting products from the file :
java.lang.NumberFormatException: For input string: "Human milk"
AH ! Something wrong happened while extracting products from the file :
java.lang.NumberFormatException: For input string: "Human milk"
Please retry with correct products.csv file.
Sorry I could't make a menu for your order. You can retry.
I am listening to your order.400
Order received for menu with 400 kcal. Your menu will be ready soon.
I am listening to your order.No side products respecting the constraints found


----- Menu of 1 ingredients:
       - 1 of Ground meat, NFS [20000200] [260 kcal]
----- Total energy: 260.0 kcal
```