16/12/2022

Rapport Projet TOO-CAI

Projet 2022-2023 « Better Food »

Christoph Samuel – Mouche Valentin

Rapport Projet TOO-CAI

Projet 2022-2023 « Better Food »

I. Présentation du sujet.

Ce projet consiste à développer un site Web qui délivre à l'utilisateur, suite au scan d'un produit alimentaire via la caméra de son appareil, des informations si elle les a trouvées, comme par exemple, dans notre cas le nutri-score de ce produit. Le but est d'utiliser la technologie Websocket pour communiquer entre le site web et le serveur où se font les requêtes vers les serveurs DNS à propos d'un site web donné, ici openfoodfacts.org.

II. Explications et choix faits dans l'application.

Pour développer cette application nous avons donc dû mettre en œuvre trois grandes parties, d'une part un côté client (bar_code_reader.ts), d'autre part un côté un côté serveur (OpenFoodFacts.java) et une dernière partie permettant de communiquer entre ses deux précédentes. Le but ici, est alors de faire corréler les deux côtés (client et serveur), afin d'obtenir, en premier lieu les informations scannées par le client, pour aller les chercher côté serveur à l'aide de l'API d'openfoodfacts pour ensuite renvoyer côté client les informations si elles sont trouvées.

L'ensemble des fichier .java et .ts possède des commentaires afin de détailler ou d'expliquer les axes principaux des programmes. Nous aurions aimé rajouter la gestion d'erreurs à notre programme au niveau du fichier WebSocket_Server.java, cependant nous ne nous s'y sommes pas attelé.

1° Détails sur la partie client

Pour répondre aux attentes de cette partie, nous nous sommes inspirés du code (Selfie.ts) disponible directement depuis la page web du sujet, auquel nous avons apporté quelques petites modifications, comme par exemple, la suppression des éventements liés au son et autres parties qui nous paraissait superflue pour le but réel de notre application.

Une fois ce code allégé, une grande partie était déjà réalisée, en effet, il nous manquait plus qu'à récupérer le code-barres présent sur la photo enregistrée, pour cela, nous avons donc décidé d'utiliser la librairie quaggaJS plus précisément la fonction Quagga.decodeSingle() de celle-ci qui comme son nom l'indique permet de décoder un code bar directement via une photo. Après avoir ajouté quagga dans nos dépendances (fichier package.json) ainsi que le site référençant cette même librairie dans le script HTML (fichier index.html) nous avons repris et modifié l'exemple disponible sur le site Web de la librairie quagga afin d'obtenir la fonction suivante.

```
93              /* Decoding of the obtained image using quagga */
94              Quagga.decodeSingle( {
95                  locate: true,
96                  frequency: 10,
97                  decoder: {
98                      readers: ["ean_reader", {
99                          format: "ean_reader",
100                         config: {
101                             supplements: [
102                                 'ean_5_reader', 'ean_2_reader'
103                             ]
104                         }
105                     }],
106                 },
107                 locator: {
108                     patchSize: "medium",
109                     halfSample: true
110                 },
111                 src: _canvas.toDataURL("image/png")
112             },
113             /* End of decoding */
114             function(result: any) {
115                 /* If a barcode is detected it is sent to the WebSockets */
116                 if (result?.codeResult) {
117                     let cb = window.document.getElementById('cb') as HTMLDivElement;
118                     console.log("Bar code dectected is:", result.codeResult.code);
119                     _bar_code = result.codeResult.code;
120                     cb.textContent = result.codeResult.code
121                     send(_bar_code);
122                 } else {
123                     /* Otherwise nothing */
124                     console.log("Bar code is not detected please retry");
125                 }
126             } );
```

Fonction Quagga .decodeSingle()

Cette fonction contient plusieurs propriétés, parmi lesquelles on retrouve « locate » and « locator », qui permettent d'une part d'activer le processus de localisation du code-barres mais également de contrôle le comportement de celui-ci, de plus le paramètre « decoder » qui, comme son nom l'indique décode le code-barres trouvé si celui-ci se situe dans la liste « readers », pour nous, il est cas de lecture de code-barres alimentaire ce qui justifie le choix du code [ean-reader] en tant que « readers » ainsi que ses extensions [ean_5_reader] et [ean_2_reader], src renvoie l'image à décoder, pour finir la fonction envoie le code-barres, s'il a été trouvé en direction de la technologie Websocket.

2° Détails sur la partie serveur

La partie serveur est gérée en Java, elle va permettre la recherche du nutri-score via le code barre et le site OpenFoodFacts. Tout ceci se trouve dans la classe Java OpenFoodFacts. On y trouve une fonction nutri-score, qui prend en paramètre un String, le bar code, qui est récupéré via quagga lors de la prise de la photo et envoyé dans la fonction via la fonction OnMessage des Websockets.

On va alors, stocker dans une variable url, l'url du site OpenFoodFacts avec le code-barres insérée dans celle-ci pour pouvoir accéder au nutri-score du produit souhaité. Après cela, nous ouvrons une connexion avec le site via la fonction openConnection() comme représenté ci-dessous.

```
/* Open the connection to acess the API */
java.net.URL url = new java.net.URL("https://world.openfoodfacts.org/api/v0/product/" + bar_code + ".json");
java.net.URLConnection connection = (java.net.URLConnection) url.openConnection();
```

Ouverture de la connexion vers OpenFoodFacts

Une fois que l'on s'est assurée que la connexion était bien établie (via le if), on utilise un parser de java pour parcourir le site et récupérer la valeur du nutri-score qui a pour clé nutriscore_grade.

Lorsque celui-ci trouve la clé nutriscore_grade, la valeur du nutri-score est récupérée dans une variable _nutriscore qui sera return à la fin du programme pour être affichée dans le main.

```
/* Recovery of the desired element, in our case the nutri-score */
if (connection != null) {
    javax.json.stream.JsonParserFactory factory = javax.json.Json.createParserFactory( config:null);
    /* Effective connection with the service */
    javax.json.stream.JsonParser parser = factory.createParser( in:connection.getInputStream());
    /* Return the JSON object */
    while (parser.hasNext()) {
        javax.json.stream.JsonParser.Event event = parser.next();
        if (event == javax.json.stream.JsonParser.Event.KEY_NAME && parser.getString().equals( anObject:"nutriscore_grade")) {
            while (parser.hasNext()) {
                event = parser.next();
                if (event == javax.json.stream.JsonParser.Event.VALUE_STRING) {
                        _nutri_score = parser.getString();
                    break;
                }
            }
        }
    }
}
/* Returns the nutriscore found or not */
return _nutri_score;
```

récupération du nutri-score()

3° Détails sur la partie communication via WebSocket

Cette partie permet la communication entre les deux parties précédentes, en effet, une fois le fichier exécuté, celui-ci lance une connexion à la technologie WebSocket à partir de la fonction connect() du fichier (bar_code_reader.ts) qui réunit l'ensemble des fonctions en relation avec les WebSocket (onopen, onerror, onmessage, onclose). Lorsque cette partie (client) détecte un code-barres à l'aide de quagga celui-ci est redirigé à la fonction send(), qui envoi le code-barres coté serveur (fichier WebSocket_Server.java).

Arrivé côté serveur, le code-barre est récupéré et utiliser dans la fonction OnMessage(). La variable message récupère la valeur envoyée par la fonction send(), on se retrouve donc avec le code-barres stocké dans la variable message que l'on va pouvoir exploiter côté serveur. On appelle pour ce faire, la fonction nutriscore de la classe Java OpenFoodFacts avec en paramètre le code-barres (stockée dans la variable message). On retrouve également, tout comme dans notre fichier TypeScript (bar_code_reader) les fonctions par défaut Onclose, OnError, OnMessage et OnOpen, dans lesquelles on a simplement des affichages dans la console.

Par exemple, la fonction OnClose affiche dans la console « onClose: WebSocket connection closed » qui permet d'avoir un visuel via la console de la fermeture de la connection des WebSocket. OnError permet, elle d'afficher l'erreur dans la console pour pouvoir potentiellement l'identifier plus facilement et la traiter dans le futur. La fonction main, elle permet le lancement du serveur WebSocket via server.start() et le lancement de la page index.html.

```
*** Please press any key to stop the server... ***

onOpen... Christoph Samuel and Mouche
onClose: WebSocket connection closed
onOpen... Christoph Samuel and Mouche
Bar code is: 3017620424403
Nutriscore of this product is: e
onClose: WebSocket connection closed
```

Suivi des informations depuis la console d'apache

Une fois cette valeur retournée, la fonction onmessage() permet l'affichage de cette valeur dans la partie web et remplace la valeur par défaut undefined. Les autres fonctions de cette partie sont, dans notre cas seulement présentes pour le bon fonctionnement de l'application. De plus une fois la page web fermé, le serveur envoie une demande de fermeture de la WebSocket exécutée par la fonction closeSocket() sur notre fichier TypeScript.

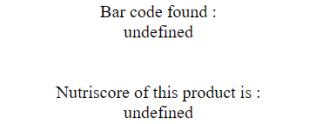
```
20    /* WebSocket message */
21    WebS.onmessage = (event: any) => {
22        let nutriscore = window.document.getElementById('nutriscore') as HTMLDivElement;
23        if (nutriscore && event?.data) {
24            nutriscore.textContent = event.data;
25            console.log("Nutriscore of this product is : " + event.data);
26        } else {
27            nutriscore.textContent = "undefined";
28        }
29    }
```

Fonction onmessage()

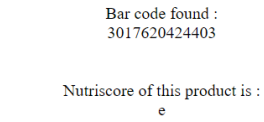
III . Utilisation de l'application.

Codé en TypeScript d'une part et en java de l'autre, l'application fonctionne très simplement, après avoir ouvert le fichier (better_food) sur Apache NetBeans IDE, il suffit de construire et d'exécuter le programme, le serveur se crée, la liaison s'effectue et la page web est chargée.

Une fois sur la page web il suffit de cliquer sur l'image centrale afin de faire apparaitre une vidéo qui se trouve être le flux en direct de la caméra choisit « environnement » par défaut (caméra extérieure de l'appareil s'il en a une). Ensuite, on dispose le code-barres que l'on veut scanner sur ce flux on confirme la capture de code-barres en cliquant une nouvelle fois sur la vidéo, en cas de succès le code-barres scanné et le nutri-score apparaissent en dessous du flux, en cas d'échec les valeurs restent par défaut undefined. Dans le cas ou un produit ne possède pas de nutri-score seul son code-barres est affiché.

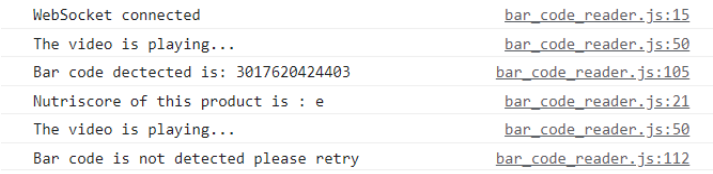


Cas d'échec



Cas de succès

Nous pouvons également voir apparaitre dans la console de la page web ainsi que sur celle d'apache le suivi des informations. De plus nous avons adapté la page HTLM (index.html et style.css), tout en restant assez sobre, dans le thème de ce projet « Better Food », png, icône de code-barres…



Suivi des informations depuis la console web

IV. Conclusion.

Ce projet nous a permis de manipuler des librairies via TypeScript ainsi que de comprendre en profondeur l'utilisation, la manipulation de données via l'API et l'utilisation de la technologie Websocket. En outre, nous avons bien compris le fonctionnement de l'interaction entre l'utilisateur et un serveur. Cela pourra nous être utile dans une future poursuite d'études.

Ce projet a été réaliser à l'aide d'Apache NetBeans IDE 16, Visual Studio Code et Google.