Compte rendu du projet

Réalisation d'un code bar scanner pour des produits alimentaires

A l'attention de :

M.Franck BARBIER

Réalisé par :

YAHI Lotfi

ARFA Mouad

I. Problématique

Le but du projet consiste à réaliser un code bar scanner permettant de capter le code bar d'un produit alimentaire figurant sur une photographie et de retourner ensuite des informations concernant le produit en les affichants sur l'écran. Pour ce, nous devons utiliser des librairies JavaScript tel Quagga afin de permettre de gérer le coté client notamment pour construire le scanner, nous devons aussi utiliser Java pour gérer le coté serveur afin de permettre de récupérer les informations recherchées sous format JSON ou XML sur le site openfoodfact, et de les envoyer ensuite à JavaScript pour les afficher.
Afin de permettre la communication entre le JAVA et le JS, nous devons utiliser la technologie des websockets.

II. Réalisation

• Coté client :

Pour gérer le coté client, nous avons choisis d'utiliser la librairie JavaScript « Quagga » qui permet la localisation et le décodage de divers types de codes-barres en temps réel La bibliothèque est aussi capable d'obtenir un accès direct à camera de l'utilisateur

En suivant les étapes et les orientations spécifiés sur le site de la librairie nous obtenant ce code qui permet la construction de notre code-barre scanner

```
Quagga.init({
    inputStream : {
        name : "Live",
        type : "LiveStream",
        target: document.querySelector('#camera')    // Or '#yourElement' (optional)
    },
    decoder : {
        readers : ["ean_reader"] //code_128_reader
    }
}, function(err) {
    if (err) {
        console.log(err);
        return
    }
    console.log("Initialization finished. Ready to start");
    Quagga.start();
});
```

Initialisation de la bibliothèque

Les fonctions Onprocessed et OnDetected qui permettent la détection d'un code barre et de le retourner

Apres la récupération du code barre sous format d'une chaine de caractère, nous l'envoyant à l'aide des websockets (illustré dans la partie serveur) à Java afin de pouvoir récupérer les informations et de les renvoyer à JavaScript sous forme d'une chaine de caractère. Apres réception de cette chaine, à l'aide de javascript nous allons extraire les informations une par une et de les insérer sur notre page html afin de les afficher.

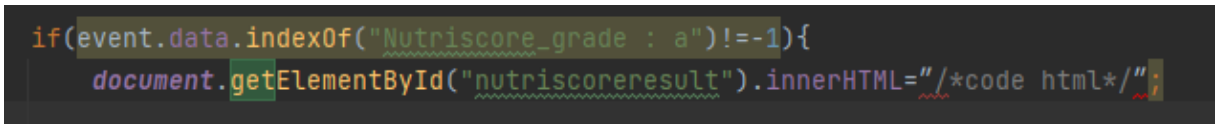
Exemple :

Sachant que Java nous renvoie la chaine sous le format suivant : ''nutriscore_grade : nutriscore Ecoscore : ecoscore''

Nous allons vérifier si cette chaine contient la sous-chaine

'' nutriscore_grade : nutriscore '', si oui, cela veut dire que nous avons trouvé un nutriscore pour le produit et on affiche ensuite selon le nutriscore.

Supposons que nous avons la chaine suivante : ''nutriscore_grade : a Ecoscore : b'' qui contient la sous chaine ''nutriscore_grade : a'', nous affichons donc nutriscore A sur notre page web.



Pour ce projet nous avons choisis de rechercher et d'afficher le code, nutri score ainsi que l'ecoscore.

Afin d'optimiser l'affichage, nous avons aussi inclus un fichier CSS.

- # Coté serveur :

Pour la création du serveur Java websockets nous avons choisis d'utiliser la Bibliothèque spark.

Pour pouvoir utiliser cette bibliothèque nous devons d'abord ajouter certaines dépendance dans notre fichier pom.xml

Ensuite, Nous crions deux fichier java, main et Echowebsocket.

WebSockets ne fonctionne qu'avec le serveur Jetty intégré et doit être défini avant les itinéraires HTTP habituelles.

Le fichier Echowebsocket est donc pour créer un itinéraires websocket et une classe gestionnaire (Handler class).

```java
public class EchoWebSocket {

    // Store sessions if you want to, for example, broadcast a message to all users
    2 usages
    private static final Queue<Session> sessions = new ConcurrentLinkedQueue<>();

    no usages  new *
    @OnWebSocketConnect
    public void connected(Session session) { sessions.add(session); }

    no usages  new *
    @OnWebSocketClose
    public void closed(Session session, int statusCode, String reason) { sessions.remove(session); }

    no usages  new *
    @OnWebSocketMessage
    public void message(Session session, String message) throws IOException {
```

Fichier EchoWebsocket

```java
public class Main {

    no usages  new *
    public static void main(String[] args) throws IOException {

        Spark.webSocket( path: "/websocket/echo", EchoWebSocket.class);
        Spark.get( path: "/", (request, response) -> {

            return "hello world";

                });
        Spark.get( path: "/echo", (request, response) -> {

            HashMap<String, Object> model = new HashMap<>();
            return new ThymeleafTemplateEngine().render (new ModelAndView(model,  viewName: "echoview"));
        });

        Spark.init();



    }
```

Fichier Main

Le serveur est automatiquement démarré lorsqu'on quelque chose qui nécessite le démarrage du serveur (c'est-à-dire déclarer une route ou définir le port).

Mais on peut également le démarrer en appelant la fonction init.

• Interroger le site « openfoodfact » afin de récupérer le fichier JSON contenants les informations du produit :

Dans la fonction message du fichier Echowebsockets, on reçoit le code-bar envoyé par Javascript, c'est donc dans cette fonction que nous allons interroger le site « openfoodfact ».

Pour ce, nous utilisons l'url suivant :
https://world.openfoodfacts.org/api/v0/product/ *[barcode]*.json

Cet url permet d'accéder à l'API json du site et de récupérer le fichier JSON.

Nous allons ensuite parcourir le JSON et de récupérer le contenu dans une variable de type StrinBuffer qui est « responseContent ». Apres la visualisation du fichier JSON on constate que c un fichier qui contient des objets de type JSONobject. Nous allons donc créer avec notre variable « responseContent » un objet JSON de type JSONObject.

```
JSONObject GS = new JSONObject(responseContent.toString());
JSONObject GG = GS.getJSONObject( key: "product");
```

On constate que le nutriscore et l'ecoscore sonc dans un objet de clé ''product'', nous récupérons donc cet objet et on accède au nutriscore et à l'ecoscore s'ils existent.

```
if (GG.has( key: "nutriscore_grade")) {
    // score= GG.getString("nutriscore_grade");
    nutriscore = GG.optString( key: "nutriscore_grade");
} else {

    nutriscore = "non renseigné ";
}
```

Exemple pour le nutriscore

Nous faisons de même pour l'écoscore, Puis on envoie à Javascript une chaine de caractère contenant le nutriscore et l'ecoscore.

```
session.getRemote().sendString( s: "Nutriscore_grade : " + nutriscore + " Ecoscore_grade : " + ecoscore);
```

- Les exception :

Lorsqu'on interroge l'API JSON du site openfoodfact, si le code-bar scanné n'est pas un code-bar d'un produit alimentaire, l'exception JSONexception surgit. Et donc pour la gérer, nous mettons en place un :

try {/*interroger le site*/}catch (JSONException e){}

III. Résultat : Nous essayons de scanner le code bar d'une bouteille de lai LACTEL

Ce que la console renvoie :

```
cbscanner.html?_ijt=…n7qb39o2jss14ol:113
▸ {codeResult: {…}, line: Array(2), angle: 0.0958347894527765, pattern: Array(315), threshold: undefined, …}
cbscanner.html?_ijt=…n7qb39o2jss14ol:130
▸ MessageEvent {isTrusted: true, data: 'Nutriscore_grade : a Ecoscore_grade : c', origin: 'ws://localhost:4567', lastEventId: '', source: null, …}
```