Examen Scala - Semestre 2 - *Rattrapage*

Dans cet examen, nous allons utiliser une des APIs de wikipedia pour faire des statistiques sur les articles : https://en.wikipedia.org/wiki/Special:ApiSandbox#action=query&format=json&prop=&list=search&srsearch=Scala&srlimit=10.

En allant sur le lien ci-dessus, dans l'onglet à gauche sur `list=search`, on peut choisir un mot clef à chercher sur wikipedia (par exemple "Scala"), et en appuyant sur "Make Request", on voit le résultat de l'appel API.

Dans cet examen, nous allons utiliser l'API pour récupérer les pages associées à un *mot clef*, traiter le résultat pour le transformer en un objet scala, et analyser les informations des différentes pages.

CONSIGNES :

- **Les conventions de styles Scala vu en cours sont obligatoires (sous peines de pénalités si non respectées)**
- **Si vous n'arrivez pas à faire une fonction, pensez à commenter le code associée pour ne pas faire planter le reste du projet**
- **Le projet est à rendre sous forme d'archive .zip et à envoyer par message teams à Quentin Nambot et Duy Nguyen**

L'examen est constitué de deux parties :

A. Code Source : Création du cœur de l'algorithme et du projet
B. Tests Unitaires : Ajout de tests unitaires sur le projet (**même si la section A n'est pas terminé, la B peut être faite en partie**)

A. Code Source

Une base existe déjà dans l'archive wikipages.zip

1. **Ajouter *scalafmt* sur le projet**. (Ne pas oublier de bien formater le code une dernière fois avant de le rendre)

---

2. Pour ce projet, nous aurons besoin des librairies suivantes. **Ajouter ces librairies au projet**

groupId	artifactId	revision
org.scalaj	scalaj-http	2.4.2
com.typesafe.play	play-json	2.9.3
com.github.scopt	scopt	4.1.0

---

3. Le projet prend deux arguments d'entrées :

- La limite du nombre de pages renvoyés par l'API (optionnel, valant 10 par défaut)
- Le mot clef à rechercher (argument obligatoire)

Mettre à jour le parser pour récupérer ces deux arguments, en utilisant des options de la libraire scopt. (La case class Config a déjà été implémenté)

Cette question peut être passé, la suite peut se faire sans cette question, en mettant une valeur par défaut `keyword = "Scala"`

---

3. L'url de l'API est faite de la façon suivante, avec {keyword} représentant le mot clef, et {limit} la limite de résultat :
https://en.wikipedia.org/w/api.php?action=query&format=json&prop=&sroffset=0&list=search&srsearch=**{keyword}**&srlimit=**{limit}**

Créer une fonction `formatUrl(keyword: String, limit: Int): String` qui formate l'url de l'API en fonction du mot clef et de la limite

---

4. Avec la librairie `scalaj.http` on peut faire des requêtes Http, exemple :

```
import scalaj.http.Http

val result = Http("https://example.com/").asString
println(result.body) // <!doctype html> <html> <head> <title>Example Domain</title> ..
println(result.code) // 200
```

Créer une fonction `getPages(url: String): Either[Int, String]` qui prend une URL est renvoie un Either avec deux possibilités :

- **Left : Si le code de la réponse est différent de 200, avec comme valeur ce code d'erreur**
- **Right : Si le code de la réponse vaut 200, avec comme valeur le corps de la page**

---

5. **Dans la fonction `run`, appeler `formatUrl` puis `getPages` pour renvoyer le résultat de l'API.**

- **S'il y a eu une erreur (left), afficher un message d'erreur contenant le code d'erreur**
- **Sinon, afficher le corps du résultat**

---

6. L'API renvoie un résultat sous forme de JSON, nous allons utiliser la librairie play-json pour parser ce résultat.

```
{"query": {"search": [...]}}
```

La librairie s'utilise de la façon suivante :

```
import play.api.libs.json.{Json, JsArray}

// Parse un json sous forme de chaîne de caractère
val json = Json.parse("{\"name\": \"Luke\", \"jobs\": [\"farmer\", \"pilot\", \"jedi\"]}")

// Récupère une valeur associée à une clef
val name = (json \ "name").as[String] // Luke
// Récupère une liste de json
val jobs = (json \ "jobs").as[JsArray].value // ["farmer", "pilot", "jedi"]
```

Créer une case class `WikiPage` qui contient deux paramètres `title` (titre de la page) et `words` (nombre de mot dans la page)

Créer une fonction `parseJson(rawJson: String): Seq[WikiPage]` qui prend en argument un json brut au format de l'API, et qui renvoie une list de `WikiPage` (on garde donc uniquement le titre d'une page, et son nombre de mots, le reste des résultats en nous intéresse pas)

Astuce : La méthode `.map` permet de facilement transformer une liste de json en une liste de WikiPage

---

7. **Mettre à jour la fonction run pour :**
   - **Appeler la méthode `parseJson` et parser les résultats**
   - **Afficher le nombres de pages trouvées**
   - **Afficher chaque page une à une**

---

8. On peut désormais faire des statistiques sur les pages !

Créer une fonction `def totalWords(pages: Seq[WikiPage]): Int` qui compte le nombre de mots à travers toutes les pages

Astuce : Utiliser la méthode `.foldLeft`

Mettre à jour la fonction run pour afficher le nombre total de mots parmis toutes les pages *et* le nombre moyen de mots par page

B. Tests Unitaires

Cette section peut être partiellement faite même si la section A n'est pas fini. Les questions 2 à 6 de cette section sont indépendantes

Afin d'avoir un projet robuste, nous allons ajouter des tests unitaires sur le projet.

1. **Ajouter la librairie suivante pour les tests uniquement :**

groupId	artifactId	revision
org.scalatest	scalatest	3.2.11

Créer une classe `MainSpec` pour les tests unitaires

---

2. **Ajouter un test unitaire pour la méthode `formatUrl`**

---

3. **Ajouter un test unitaire pour la fonction `parseJson`**

---

4. **Ajouter deux test unitaire pour la fonction totalWords**
   - **Un avec une liste vide en entrée**
   - **Un avec une liste non vide**

---

5. **Ajouter trois tests unitaires pour la fonction `parseArguments`**
   - **Un avec des arguments non parsable**
   - **Un avec un mot clef**
   - **Un avec mot clef et limite**

---

6. **Ajouter un test unitaire pour la fonction `getPages`**

Conseil : Pour ce test, il va falloir faire un mock de la classe Http, qui renvoie un objet HttpRequest

- *Créer un trait HttpUtils, avec une méthode parse(url: String): HttpRequest*
- *Créer un object étendant HttpUtils, qui implémente la méthode parse*
- *Modifier la méthode getPages pour qu'elle prenne en argument un objet de type HttpUtils*
- *Dans les tests, créer un object MockHttpUtils qui implémente une méthode parse ne faisant pas d'appelle à internet*