Formation Scala - Projet

Réalisation d'une API REST

Sujet :

Une streamer vient de faire le buzz et a des milliers de nouveaux sub'. Cependant, il n'a pas eu le temps de mettre un place un service de tips / giveaways.
Pour répondre à ce besoin, il vous a demandé de créer un API REST qui contient plusieurs endpoints :

- Liés aux Tips
    - Récupérer la liste de tous les donateurs (liste de users)
    - Réaliser un don
    - Annuler un don
    - Faire la somme de tous les dons
    - Faire la somme de tous les dons par utilisateur
    - Faire la somme de tous les dons d'un utilisateur
- Liés aux Sub
    - Récupérer la liste de tous les abonnées (liste de users)
- Liés aux giveaways
    - Créer un giveaway
    - S'inscrire à un giveaway pour un utilisateur
    - Tirage au sort du gagnant (aléatoirement pondéré par le montant donné par l'utilisateur…, s'assurer que l'utilisateur n'est pas ban)
- Liés à la blacklist
    - Pouvoir blacklister un utilisateur (ne peut plus participer aux giveaway)
- Liés aux sondages
    - Créer un sondage (2 choix possibles à chaque fois)
    - Participer au sondage pour un utilisateur
    - Avoir le résultat final du sondage

Libre à vous d'ajouter de nouvelles fonctionnalitées (elles devront documentées).

LA "LOGIQUE" NE DOIT PAS ÊTRE RÉALISÉE DANS LES REQUÊTES SQL mais par des fonctions en scala.

Votre application doit être réaliser en scala en y appliquant les principes de la programmation fonctionnelle vu en cours (**fonctions pures**, etc). Elle peut utiliser les librairies vu en cours (**akka-http** et **scalatest**); si vous souhaitez intégrer d'autres librairies, envoyez moi un message pour que je m'assure de la pertinence. Il est important que votre code soit structuré en plusieurs fichiers/packages tout en gardant une certain logique (package util/db etc.). Chaque fonction/méthode/etc devra être testé via la librairie Scalatest (vous pouvez choisir le style de test qui vous convient le mieux).

En ce qui concerne la base de données, vous utiliserez un SQLite (la partie modèle de données, gestion des tables, etc ne feront pas parties de la notation). Pour peupler la base de données, vous avez le choix entre un mode “manuel” via la CLI sqlite3 ou directement via l’application scala via une option de lancement (point supplémentaire). Dans le premier cas, le script de création/population est à fournir dans le livrable final.

Lors de la notation, pour m’assurer du bon fonctionnement de vos API, j’utiliserai Postman (https://www.getpostman.com). Vous pouvez fournir un export de votre Postman pour simplifier les tests; sinon, dans le README, mettez un exemple pour chaque point de l’API. Pour les versions de scala, vous utiliserez la version 2.11 ou 2.12.

Pour le livrable, au choix, vous pouvez rendre un archive zip (sans mot de passe) contenant votre code, le jar ainsi que les instructions pour le lancer, ou alors un repo Github (accessible) contant votre code avec un README expliquant comment build et run le projet.

Conseils :

- Séparez vous le travail : partie API REST / partie échange avec la base de données
- N’hésitez pas à poser des questions, ne restez pas bloquer
- Ouvrez et fermez la connection à la base à chaque nouvelle requête

Annexes :

SQLite

Ajouter sqlite-jdbc à vos dépendances
(https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc)
exemple :

```
package utils

import java.sql.{Connection, DriverManager, ResultSet}

import scala.util.{Failure, Success, Try}

object SQLiteHelpers{

  def request(url: String,
              sql: String,
              cols: Seq[String],
              driver: String = "org.sqlite.JDBC"):
Option[Vector[Map[String, Object]]] = {
    Class.forName(driver)
    val conn: Connection = DriverManager.getConnection(url)
    val statement = conn.createStatement()
```

```
    val result: Option[ResultSet] = Try {
      statement.executeQuery(sql)
    } match {
      case Success(output) => Some(output)
      case Failure(_) => None
    }

    val resultFormat = result match{
      case Some(rs) => Some(Iterator.continually(buildMap(rs,
cols)).takeWhile(!_.isEmpty).map(_.get).toVector)
      case None => None
    }

    if(conn != null) conn.close()
    resultFormat
  }

  def buildMap(queryResult: ResultSet, colNames: Seq[String]):
Option[Map[String, Object]] =
    if (queryResult.next()){
      Some(colNames.map(n => n -> queryResult.getObject(n)).toMap)
    }
    else{
      None
    }
}
```

Slick (point en plus):

Ajouter un fichier de conf dans les resources décrivant la connection à votre base de données.