

Projet Spark Scala SBT sous IntelliJ

Mr DIATTARA Ibrahima

Contexte Projet

Le service de vente de l'entreprise **LatDior Data** cherche un **Data Engineer** pour mettre en place une application Spark/Scala/SBT pour la gestion des contrats

L'application doit être capable de traiter les types de fichiers suivants:

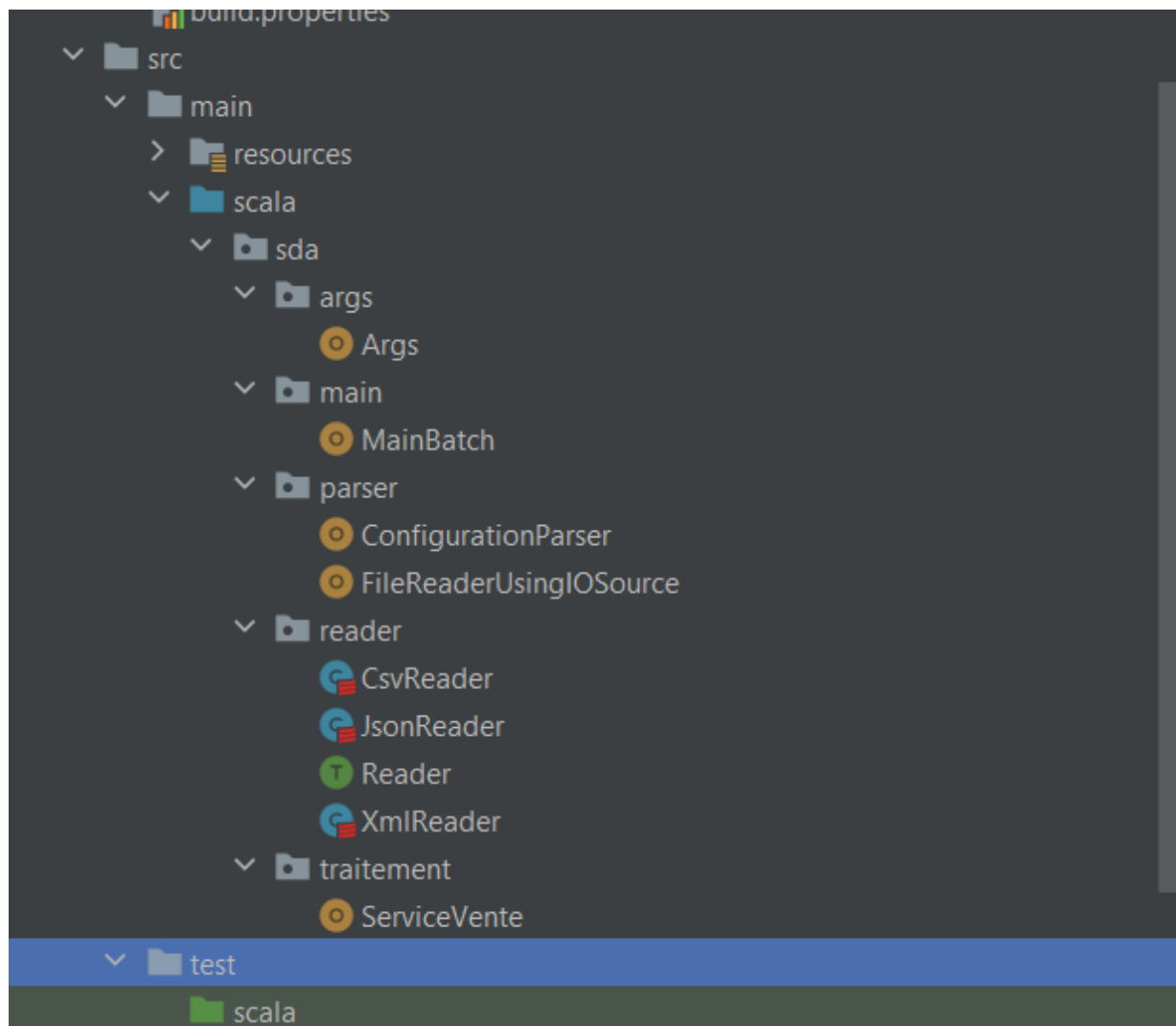
- JSON
- CSV
- XML

Pour les fichiers CSV et JSON, le modèle de données est déjà défini, mais pour les fichiers XML, vous devez le définir

Contraintes Techniques

- ❖ Vous serez obligé de réaliser les développements avec
 - IntelliJ IDEA
 - Sbt
 - Scala 2.12.15
 - Java 1.8
 - Spark
- ❖ Le respect du découpage du code est impératif, sans le respect du découpage défini, vos développements ne seront pas acceptés
- ❖ Un passage de connaissance sera organisé dès votre arrivée

Découpage du code



sbt.build

https://github.com/idiattara/sda_2024/blob/main/sbt.build

```
name := "projet_sda_2024"  
version := "1.0"  
scalaVersion := "2.12.15"  
libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.3.1"  
libraryDependencies += "com.beust" % "jcommander" % "1.48"
```

File Configuration

https://github.com/idiattara/sda_2024/blob/main/resources.zip

Type CSV

```
{  
  "path": "src/main/resources/DataforTest/data.csv",  
  "delimiter": "#",  
  "header": true  
}
```

Type SON

```
{  
  "path": "src/main/resources/DataforTest/data.json",  
  "multiline": true  
}
```

Pour le XML vous devez créer la conf

Présentation des données : Exemple de CVS avec séparateur

```
Id_Client#HTT_TVA#MetaData
1#100,5|0,19#{ "MetaTransaction": [{"Ville": "Paris", "Date_End_contrat": "2020-12-23 00:00:00"}, {"TypeProd": "Laitier", "produit": ["yaourt", "laitcoco"]}]}
2#120,546|0,20#{ "MetaTransaction": [{"Ville": "Alger", "Date_End_contrat": "2023-12-23 00:00:00"}, {"TypeProd": "Boisson", "produit": ["coca", "fanta"]}]}
3#123,6|0,201#{ "MetaTransaction": [{"Ville": "Dakar", "Date_End_contrat": "2020-12-23 00:00:00"}, {"TypeProd": "Laitier", "produit": ["yaourt"]}]}
4#5,546|0,15#{ "MetaTransaction": [{"Ville": "Abidjan", "Date_End_contrat": "2024-12-23 00:00:00"}, {"TypeProd": "Laitier", "produit": ["laitcoco"]}]}

```

Présentation des données : Exemple de JSON

```
[
  {
    "Id_Client": "1",
    "HTT_TVA": "100,5|0,19",
    "MetaData": "{ \"MetaTransaction\": [{ \"Ville\": \"Paris\", \"Date_End_contrat\": \"2024-12-23 00:00:00\" }, { \"TypeProd\": \"Laitier\", \"produit\": [ \"yaourt\", \"laitcoco\" ] } ] }"
  },
  {
    "Id_Client": "2",
    "HTT_TVA": "120,546|0,20",
    "MetaData": "{ \"MetaTransaction\": [{ \"Ville\": \"Alger\", \"Date_End_contrat\": \"2024-12-23 00:00:00\" }, { \"TypeProd\": \"Boisson\", \"produit\": [ \"coca\", \"fanta\" ] } ] }"
  },
  {
    "Id_Client": "3",
    "HTT_TVA": "123,6|0,201",
    "MetaData": "{ \"MetaTransaction\": [{ \"Ville\": \"Dakar\", \"Date_End_contrat\": \"2020-12-23 00:00:00\" }, { \"TypeProd\": \"Laitier\", \"produit\": [ \"yaourt\" ] } ] }"
  },
  {
    "Id_Client": "4",
    "HTT_TVA": "5,546|0,15",
    "MetaData": "{ \"MetaTransaction\": [{ \"Ville\": \"Abidjan\", \"Date_End_contrat\": \"2022-12-23 00:00:00\" }, { \"TypeProd\": \"Laitier\", \"produit\": [ \"laitcoco\" ] } ] }"
  }
]
```

Pour les données XML vous devez créer votre propre jeu de données

Présentation de l'existante: Prepare Appli Args

https://github.com/idiattara/sda_2024/blob/main/Args.scala

```
package sda.args
import com.beust.jcommander.{JCommander, Parameter}
object Args {

  @Parameter(
    names = Array("-rc", "--reader-file-path"),
    description = "Reader configuration file path",
    required = true)
  var readerConfigurationFile: String = _

  @Parameter(
    names = Array("-rt", "--reader-type"),
    description = "Reader type ",
    required = true)
  var readertype: String = _

  def parseArguments(args: Array[String]): JCommander = {
    new JCommander(object = this, args.toArray: _*)
  }
}
```


Présentation de l'existant:Parser FileConf

https://github.com/idiattara/sda_2024/blob/main/FileReaderUsingIOSource.scala

```
package sda.parser
import scala.io.Source
object FileReaderUsingIOSource {

  def getContent(file: String): String = {
    Source.fromFile(file).getLines().mkString
  }

}
```

https://github.com/idiattara/sda_2024/blob/main/ConfigurationParser.scala

```
import org.json4s.DefaultFormats
import org.json4s.jackson.JsonMethods
import sda.reader._

object ConfigurationParser {

  implicit val format = DefaultFormats

  def getCsvReaderConfigurationFromJson(jsonString: String): CsvReader = {
    JsonMethods.parse(FileReaderUsingIOSource.getContent(jsonString)).extract[CsvReader]
  }

  /* Complétez cette fonction. Elle prend un String en argument et renvoie un objet JsonReader.
   Elle doit être codée de la même manière que la fonction getCsvReaderConfigurationFromJson */

  def getJsonReaderConfigurationFromJson(jsonString: String): JsonReader = {
    .....
  }

  coder aussi la partie XML |
  */
```

Présentation de l'existant:Objet CsvReader

https://github.com/idiattara/sda_2024/blob/main/Reader.scala

```
package sda.reader

import org.apache.spark.sql.{DataFrame, SparkSession}

trait Reader {
  val format: String
  def read()(implicit spark: SparkSession): DataFrame
}
```

https://github.com/idiattara/sda_2024/blob/main/CsvReader.scala

```
package sda.reader

import org.apache.spark.sql.{DataFrame, SparkSession}
case class CsvReader(path: String,
                     delimiter: Option[String] = None,
                     header: Option[Boolean] = None
                     )

extends Reader {
  val format = "csv"

  def read()(implicit spark: SparkSession): DataFrame = {
    spark.read.format(format)
      .option("delimiter", delimiter.getOrElse(","))
      .option("header", header.getOrElse(true))
      .load(path)
  }
}
```

JsonReader

- ❑ Coder la Class JsonReader de la même façon que la Class CsvReader mais avec ses propres attributs
- ❑ La Class héritera le trait Reader
- ❑ Elle doit contenir une fonction read qui permet de lire un fichier JSON et retourne un DataFrame en se basant sur les attributs de class JsonReader provenant du fichier de conf reader_json.json

XmlReader

- ❑ Coder la Class Xml Reader de la même façon que la Class CsvReader mais avec ses propres attributs
- ❑ La Class héritera le trait Reader
- ❑ Elle doit contenir une fonction read qui permet de lire un fichier Xml et retourne un DataFrame en se basant sur les attributs de class XmlReader provenant du fichier de conf reader_xml.json **que vous allez définir**

Objet Service Vente

https://github.com/idiattara/sda_2024/blob/main/ServiceVente.scala

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.{DataFrame}
import org.apache.spark.sql.types._
object ServiceVente {

  implicit class DataFrameUtils(dataFrame: DataFrame) {

    def formatter()= {
      dataFrame.withColumn( colName = "HTT", split(col( colName = "HTT_TVA"), pattern = "\\|")(0))
      .withColumn( colName = "TVA", split(col( colName = "HTT_TVA"), pattern = "\\|")(1))
    }

    def calculTTC () : DataFrame ={
      /*.....Coder ici.....*/
    } : Unit

    def extractDateEndContratVille(): DataFrame = {
      val schema_MetaTransaction = new StructType()
      .add( name = "Ville", StringType, nullable = false)
      .add( name = "Date_End_contrat", StringType, nullable = false)
      val schema = new StructType()
      .add( name = "MetaTransaction", ArrayType(schema_MetaTransaction), nullable = true)
      /* Coder ici*/
    } : Unit

    def contratStatus(): DataFrame = {
      /*.....Coder ici.....*/
    } : Unit
  }
}
```

CalculTTC

def calculTTC () : DataFrame

- ✓ Calcule le TTC => $HTT + TVA * HTT$, le TTC doit être arrondi à 2 chiffres après la virgule
- ✓ Supprime la colonne TVA, HTT

Id_Client	HTT	TVA	MetaData	TTC
1	100,5	0,19	{"MetaTransaction..."}	119.6
2	120,546	0,20	{"MetaTransaction..."}	144.66
3	123,6	0,201	{"MetaTransaction..."}	148.44
4	5,546	0,15	{"MetaTransaction..."}	6.38

Extract_Date_End_Contrat_Ville

✓ Créer une nouvelle colonne Date_End_contrat et Ville en utilisant la méthode **select from_json** et **regexp_extract** pour extraire YYYY-MM-DD

✓ Supprime la colonnemetaData

Id_Client	HTT_TVA	TTC	Ville	Date_End_contrat
1	100,5 0,19	119.6	Paris	2020-12-23
2	120,546 0,20	144.66	Alger	2023-12-23
3	123,6 0,201	148.44	Dakar	2020-12-23
4	5,546 0,15	6.38	Abidjan	2024-12-23

Contrat_Status

Créer un nouvelle colonne Contrat_Status avec "Expired" si le contrat a expiré et sinon "Actif"

Id_Client	HTT_TVA	TTC	Ville	Date_End_contrat	Contrat_Status	
	1	100,5 0,19	119.6	Paris	2020-12-23	Expired
	2	120,546 0,20	144.66	Alger	2023-12-23	Expired
	3	123,6 0,20	148.44	Dakar	2020-12-23	Expired
	4	5,546 0,15	6.38	Abidjan	2024-12-23	Active

Main Configuration

Type CSV

Name: ☐ Store as project file 

Build and run Modify options ▾ Alt+M






 

Press Alt for field hints

Type JSON

Build and run Modify options ▾ Alt+M



Press Alt for field hints

MainBatch

https://github.com/idiattara/sda_2024/blob/main/MainBatch.scala

```
package sda.main
import ...
object MainBatch {
  def main(args: Array[String]): Unit = {
    implicit val spark: SparkSession = SparkSession
      .builder
      .appName(name = "SDA")
      .config("spark.master", "local")
      .getOrCreate()
    Args.parseArguments(args)
    val reader = Args.readertype match {
      case "csv" => {
        ConfigurationParser.getCsvReaderConfigurationFromJson(Args.readerConfigurationFile)
      }
      case "json" => {
        ConfigurationParser.getJsonReaderConfigurationFromJson(Args.readerConfigurationFile)
      }
      /*Rajouter le fait de prendre en compte un fichier XML*/
      case _ => throw new Exception("Invalid reader type. Supported reader format : csv, json and xml in feature")
    }
    val df=reader.read().formatter()
    println("*****Resultat Question1*****")
    df.show( numRows = 20)
    println("*****Resultat Question2*****")
    df.calculTTC().show( numRows = 20)
    println("*****Resultat Question3*****")
    df.calculTTC.extractDateEndContratVille.show
    println("*****Resultat Question4*****")
    df.calculTTC.extractDateEndContratVille.contratStatus.show( numRows = 20)
  }
}
```

Dans le main rajouter seulement le fait de prendre en compte un fichier XML

Test Unitaire

Dans votre code, implémentez des tests unitaires pour les fonctions de l'objet **ServiceVente**.

Dans le cours, nous n'avons pas abordé la partie sur les tests unitaires, donc vous devez effectuer des recherches à ce sujet.