Objet : Présentation des annotations @InputData / @OutputData pour le mapping des fichiers de données

Bonjour [Nom du destinataire],

Je me permets de vous partager un bref aperçu du fonctionnement des annotations personnalisées @InputData et @OutputData, utilisées dans notre projet pour mapper des classes Java vers des fichiers de données (CSV, Excel, Parquet, Avro, JSON, etc.).

Ces annotations sont définies au niveau de la classe ou des champs (@Target({ElementType.TYPE, ElementType.FIELD})) et permettent :

- De spécifier le chemin d'entrée/sortie via l'attribut path
- De définir le format du fichier (via l'enum FormatEnum) : CSV, PARQUET, JSON, XML, AVRO, etc.
- De configurer différents paramètres liés à la lecture/écriture : séparateurs (delimiter, columnDelimiter), noms de table, de colonnes, gestion des valeurs nulles (nullValue, treatEmptyValuesAsNulls), partitionnement, etc.

Ces annotations servent de point d'entrée unique pour injecter dynamiquement les données dans les classes Java (ex: FiltersBloomTW) en définissant :
- le schéma via les annotations @Column
- les partitions via @PartitionColumn

Le système est générique et extensible pour supporter plusieurs types de sources et de

destinations, en assurant une cohérence entre les fichiers et les modèles Java.

N'hésitez pas si vous avez besoin de plus de détails ou d'un exemple d'implémentation complet.

Bien cordialement,

[Votre Prénom Nom]

Subject: Overview of @InputData / @OutputData annotations for file-to-Java class mapping

Hi [Recipient's Name],

I would like to share a quick overview of how the custom annotations @InputData and @OutputData work in our project, enabling seamless mapping of Java classes to data sources such as CSV, Excel, Parquet, Avro, JSON, and more.

These annotations are applied at the class or field level (@Target({ElementType.TYPE, ElementType.FIELD})) and allow us to:

- Specify the file path using the path attribute
- Indicate the file format via the FormatEnum enum: CSV, PARQUET, JSON, XML, AVRO, etc.
- Configure various read/write parameters such as delimiter, columnDelimiter, nullValue, treatEmptyValuesAsNulls, and partitioning rules

For example, classes like FiltersBloomTW are annotated to:
- Define the data schema using @Column
- Specify partitioning via @PartitionColumn

This system is designed to be generic and extensible, providing a unified way to ingest or export data from different formats while keeping the Java model consistent.

Let me know if you need further details or a complete working example.

Best regards,

[Your First Name Last Name]

InputData : OutputData
(Annotation)

FormatEnum

FiberBloomTW
(Java Model)

Fichier source
(CSV, PARQUET, etc.)

Fichier cible
(CSV, PARQUET, etc.)