1) Explication ultra-simple

Imagine que tu veux créer un magasin qui vend des articles.
Dans les anciens TP, tu devais :

- créer **un contrôleur**
- créer **un service**
- créer **un DAO**
- écrire **toutes les méthodes CRUD** à la main (GET, POST, PUT, DELETE)

👉 **Beaucoup de travail… répétitif !**

Spring Data REST arrive et te dit :

« Tu n'as pas besoin d'écrire les contrôleurs, je vais tout générer pour toi automatiquement ! »

Donc :

- Tu crées **juste les classes modèles (Article, Categorie)**
- Tu crées **des interfaces Repository**
- Et Spring Data REST fabrique **automatiquement les API REST**.

C'est comme si Spring te disait :

« Donne-moi ton modèle, et je m'occupe de créer toutes les routes REST tout seul. »

🎁 Bonus :
Spring Data REST envoie aussi des réponses jolies en **format HAL**, avec des **liens** pour naviguer entre les ressources.

A. Pourquoi Spring Data REST ? (Concept clé du TP)

Dans une application classique (TP1, TP2), l'architecture est :

1. Controller
2. Service
3. DAO (Repository)
4. Model

Tu devais coder **manuellement** toutes les API REST.

👉 **Problème : beaucoup de duplication**
Chaque entité = un contrôleur = plusieurs endpoints.

➞ **Spring Data REST élimine les contrôleurs et services**,
parce qu'il génère automatiquement **toutes les routes CRUD** en s'appuyant sur les Repository.

📌 Donc :
Le Repository devient un contrôleur REST automatique.

B. Architecture du projet

Elle devient très simple :

1) Model

- Article
- Categorie

2) Repository

- ArticleRepository
- CategorieRepository

3) Configuration Spring Boot

- application.properties
- MainApplication avec initDatabase()

👉 **Pas de couche service**
👉 **Pas de contrôleur**
👉 **Spring Data REST expose tout automatiquement**

C. Concepts super importants à retenir

1. HAL (Hypertext Application Language)

C'est un format JSON spécial qui ajoute :

- `_embedded` → les données
- `_links` → des liens pour naviguer d'une ressource à une autre

Exemple (page 15 du fichier)

3- TP 3- Spring Data Rest_V2

:
On voit bien que les articles ont des liens comme :
`self`
`categorie`

`projection`

HAL permet à ton API d'être « auto-documentée ».

2. Projections (@Projection)

Spring Data REST n'envoie pas automatiquement tous les champs.
Avec une projection, tu peux **choisir quels attributs sortir**.

Dans ce TP :

```
@Projection(name = "articleDTO", types = Article.class)
```

Cette projection expose :

- id
- desc
- quant
- cat

C'est un mini DTO automatique.

3. @RepositoryRestResource

Elle transforme une interface Repository en **endpoint REST**.

Exemple dans `ArticleRepository` :

```
@RepositoryRestResource(path = "ecommerce", excerptProjection =
ArticleDTO.class)
```

Donc Spring crée automatiquement les endpoints :

- GET /ecommerce
- POST /ecommerce
- GET /ecommerce/{id}
- PUT /ecommerce/{id}
- DELETE /ecommerce/{id}
- GET /ecommerce/search

Et même :

```
/ecommerce/search/byCategorie?categorie=...
```

Parce que tu as ajouté :

```
@RestResource(path = "byCategorie")
```

4. Méthodes dérivées Spring Data (Query Methods)

Spring crée automatiquement la requête SQL grâce à :

```
findByCategorie_Categorie(String categorie)
```

Donc pas besoin de JPQL ou SQL.

5. H2 Console

Tu vérifies que :

- les tables Article et Categorie ont été générées
- les données d'initDatabase ont été insérées automatiquement

Lien :

```
http://localhost:8080/h
```

6. Aucun controller n'a été écrit → tout est généré automatiquement

C'est LA particularité de ce TP.

D. Workflow du TP

1. Créer les modèles JPA
2. Créer les repositories
3. Configurer H2 dans application.properties
4. Initialiser la BD dans MainApplication
5. Lancer l'application
6. Tester les endpoints automatiques
7. Tester Swagger UI
8. Tester HAL Explorer

F. Swagger + OpenAPI

Tu ajoutes :

```
springdoc-openapi-starter-webmvc-ui
```

Ce qui t'offre :

- Documentation automatique OpenAPI
- Interface Swagger UI
- Test des endpoints facilement

Lien :

```
http://localhost:8080/swagger-ui/index.html
```

G. HAL Explore

(Un bonus dans ce TP)

Il permet :

- de naviguer dans l'API
- de voir les liens HAL
- de consulter les projections

Lien :

```
http://localhost:8080/
```

RÉSUMÉ

👉 Avant :
Je devais tout coder : Controller + Services + DAO.

👉 Avec Spring Data REST :
Je crée seulement :

- Entity
- Repository

Et Spring me donne automatiquement :

- toutes les routes REST CRUD
- la pagination
- les projections
- HAL + HATEOAS
- OpenAPI + Swagger
- HAL Explorer

C’est **le TP des services REST automatiques**.