Compte rendu XML - RSS25SB

Matisse SENECHAL

2024-2025

Encadrants :

— M. Émilien BONDU,
— M. Étienne REITH,
— M. Stéphane HÉRAUVILLE

Le projet est disponible sur GitHub : github.com/senechalmatisse/rss25SB

Table des matières

1 Introduction

L'objectif principal de ce projet est de concevoir, développer et déployer un service REST complet et robuste permettant de gérer des flux RSS personnalisés, conformes à la spécification `rss25SB`.

1.1 Objectifs du projet

Plus concrètement, les objectifs sont les suivants :

- **Développement d'un service RESTful complet** : le service expose différents endpoints pour l'affichage, l'insertion, la suppression et la consultation d'articles RSS. Chaque opération respecte les contraintes d'architecture REST (utilisation des verbes HTTP, formatage des réponses, traitement des statuts).
- **Validation des flux XML** : chaque flux RSS reçu est systématiquement validé à l'aide d'un schéma XSD (`rss25.xsd`) avant traitement. Garantissant la conformité structurelle des données et empêche toute insertion invalide.
- **Affichage multi-format (XML et HTML)** : afin de répondre à des cas d'usage variés, chaque ressource (article ou liste) peut être consultée soit au format XML, soit au format HTML.
- **Gestion robuste des erreurs** : selon le type de requête (API ou navigateur), le système retourne une réponse d'erreur qui doit être formatée soit en XML, soit en HTML.
- **Création d'outils clients** :
  - Un **outil de transfert**, avec interface simple, permet à l'utilisateur de sélectionner un fichier local au format rss25SB et de l'envoyer au serveur REST.
  - Un **outil de conversion** est capable de récupérer un flux externe (ex. `lemonde.fr`), de le convertir automatiquement au format rss25SB, puis de le transférer via l'outil précédent.
- **Déploiement en environnement réel** : le projet est hébergé sur la plateforme CleverCloud afin de simuler un déploiement professionnel.

En synthèse, ce projet vise à concevoir un service REST complet et modulaire, tout en assurant une interface utilisateur simple et fonctionnelle pour la manipulation/visualisation des données.

2 Architecture du projet

2.1 Package `client.config`

Ce package contient la classe `Rss25SBClientProperties`, chargée de centraliser les paramètres de connexion au service REST (`host`, `port`, `endpoint`, `fullUrl`) via injection de configuration. Il permet de construire dynamiquement l'URL cible et intègre un mécanisme de fallback avec journalisation en cas de configuration incomplète.

2.2 Package `controllers`

Ce package regroupe les différents contrôleurs REST exposant les fonctionnalités clés du service :

- `IndexController` : affiche la page d'accueil avec les métadonnées du projet.
- `HelpController` : génère dynamiquement la documentation de l'API REST.
- `ResumeController` : retourne une liste synthétique des articles en XML ou HTML.
- `ItemController` : affiche le détail d'un article au format HTML.
- `InsertController` : insère un flux XML, avec validation XSD et conversion automatique si nécessaire.
- `DeleteController` : supprime un article par ID et retourne une réponse XML structurée.

Chaque contrôleur est spécialisé, respectant ainsi le principe de responsabilité unique, et intègre la journalisation pour assurer la traçabilité des requêtes et erreurs.

2.3 Package `converter`

Le package `converter` est dédié à la conversion de flux RSS externes vers le format interne `rss25SB`. Il inclut :

- `FluxSourceSelector` : identifie dynamiquement la source d'un flux XML externe et sélectionne le convertisseur adapté.
- `sources.LeMondeFluxConverter` : convertisseur DOM spécifique au flux RSS du site *lemonde.fr*. Il extrait les métadonnées des balises XML (`title`, `guid`, `pubDate`, `media:content`, etc.) et les mappe dans le modèle `Feed`.

2.4 Package `dto`

Ce package contient l'ensemble des *Data Transfer Objects* (DTO) utilisés pour représenter les réponses XML échangées via l'API REST `rss25SB`. Tous les DTO sont annotés avec JAXB afin de garantir une sérialisation conforme au schéma XSD défini dans le projet.

On distingue notamment :

- `InsertResponseDTO`, `DeleteResponseDTO`, `XmlErrorResponseDTO` : réponses XML standardisées (succès, suppression, ou erreur).
- `ItemSummaryDTO` et `ItemSummaryListDTO` : représentations synthétiques des articles (id, guid, date).

2.5 Package `exception`

Ce package contient un contrôleur d'erreurs global, `CustomErrorController`, qui redéfinit la gestion des erreurs HTTP dans l'application Spring.

- Il intercepte les erreurs non gérées (404, 400, 500, etc.) via le point d'entrée `/error`.
- Il génère dynamiquement une réponse XML standardisée basée sur le DTO `XmlErrorResponseDTO`.
- Il utilise `XmlUtil` pour la sérialisation et `Slf4j` pour le logging.

2.6 Package `model`

Ce package contient les modèles de données structurants l'application, organisés comme suit :

- `model.db` : Entités JPA représentant les articles RSS (`ItemEntity`) et leurs composants persistés (auteurs, catégories, contenu, image...).
- `model.xml` : Modèles JAXB utilisés pour la sérialisation/désérialisation XML du flux `rss25SB` (`Item`, `Feed`, `Link`, etc.).
- `model.adapter` : Adaptateurs de type JAXB comme `OffsetDateTimeXmlAdapter` pour assurer la compatibilité avec les formats ISO/RFC.
- **racine** `model` : Classes utilitaires comme `ProjectInfo` et `OperationInfo`, utilisées respectivement sur les vues `/` et `/help`.

2.7 Package `repository`

Ce package contient l'interface `ItemRepository` qui étend `JpaRepository<ItemEntity, Long>`. Elle fournit les opérations de persistance classiques sur les articles RSS, ainsi qu'une méthode spécifique :

- `existsByGuid(String guid)` : permet de vérifier l'unicité d'un article en base avant insertion.

2.8 Package `service`

Le package `service` regroupe l'ensemble de la logique métier de l'application. Il est constitué de trois services principaux :

- `ItemService` : encapsule les opérations sur les articles RSS (CRUD, conversion entité $\leftrightarrow$ XML, suppression, validation).
- `HelpInfoService` : génère dynamiquement la documentation des endpoints disponibles sur l'API.
- `ProjectInfoService` : centralise les métadonnées statiques du projet (nom, version, développeur, logo).

Ces services exploitent un logging (`Slf4j`) pour faciliter la maintenance et le débogage.

2.9 Package `utils`

Le package `utils` regroupe les classes utilitaires génériques du projet. Il est structuré en modules spécialisés :

- `constants` : centralise les constantes globales comme les statuts XML ou le chemin XSD.
- `XmlUtil` : gère la sérialisation/désérialisation JAXB avec validation XSD.
- `ItemMapper` : assure la conversion bidirectionnelle entre entités JPA et objets XML JAXB.
- `HtmlRenderer` : génère des vues HTML via Thymeleaf à partir de modèles et variables.
- `DateTimeUtil` : propose un formatage des dates conforme à RFC 3339.
- `StringUtil` : fournit des méthodes de manipulation robuste des chaînes de caractères.

3 Fonctionnalités implémentées

3.1 Page d'accueil (I.1)

La page d'accueil, accessible via l'URL racine `/`, a été implémentée conformément aux exigences du cahier des charges.

— **Méthode HTTP** : `GET`
— **Retour** : Page au format **HTML** conforme

Cette page statique a été générée à l'aide du moteur de templates **Thymeleaf**, assurant ainsi un rendu proprement structuré et valide. Elle présente les informations attendues :

— Le nom du projet : `rss25SB`
— La version actuelle du projet
— Le nom complet du développeur
— Le logo officiel de l'Université de Rouen, centré sur la page

L'ensemble des données affichées sont injectées dynamiquement via le service `ProjectInfoService`.

3.2 Aide (I.2)

La page d'aide, accessible à l'URL `/help`, a été implémentée afin de fournir une vue synthétique des différentes fonctionnalités offertes par le service REST.

— **Méthode HTTP** : `GET`
— **Retour** : page **HTML** valide

Chaque opération REST exposée par le service est affichée dans un tableau HTML avec les informations suivantes :

— l'URL de l'endpoint ;
— la méthode HTTP utilisée ;
— une description condensée de l'opération.

Pour l'affichage les données sont injectées par le contrôleur via un objet `Model`, contenant une liste d'instances `OperationInfo` fournies par le service `HelpInfoService`.

Projet RSS25SB

Page d'Aide

Voici la liste des opérations disponibles :

URL	Méthode	Description
/	GET	Affiche la page d'accueil du service RSS25SB (format HTML).
/help	GET	Affiche la page d'aide avec la liste des opérations disponibles (format HTML).
/rss25SB/resume/xml	GET	Retourne la liste des articles disponibles sous forme synthétique (id, date, guid) au format XML.
/rss25SB/resume/html	GET	Retourne la liste synthétique des articles sous forme HTML (via transformation XSLT).
/rss25SB/resume/xml/{id}	GET	Affiche un article complet au format XML. L'identifiant doit être valide.
/rss25SB/html/{id}	GET	Affiche un article complet en HTML via transformation XSLT à partir du XML identifié.
/rss25SB/insert	POST	Insère un flux XML conforme au XSD rss25SB. Le flux doit être envoyé en format XML (Content-Type: application/xml). Retourne un statut XML indiquant le succès ou l'échec.
/rss25SB/insert	GET	Affiche un formulaire HTML pour téléverser un fichier XML local (multipart/form-data).
/rss25SB/insert/html	POST	Traite un fichier XML envoyé via formulaire (multipart/form-data), effectue une insertion après validation ou conversion, et affiche un retour HTML via transformation XSLT.
/rss25SB/delete/{id}	DELETE	Supprime l'article identifié par l'id fourni. Retourne un flux XML indiquant le statut de suppression (DELETED ou ERROR).

3.3 Liste synthétique des articles (I.3)

Cette fonctionnalité permet de consulter, au format XML ou HTML, une version résumée des articles stockés dans la base. L'objectif est de fournir une vue synthétique d'un article.

3.3.1 XML

— **URL** : `/rss25SB/resume/xml`
— **Méthode** : `GET`
— **Retour** : **Flux XML** conforme au schéma défini

Chaque article est représenté par un bloc XML contenant :
— `id` : identifiant unique généré par le serveur ;
— `date` : date de publication ou de mise à jour ;
— `guid` : identifiant global (souvent une URL).

Le flux est généré à partir d'objets `ItemSummaryDTO` encapsulés dans un `ItemSummaryListDTO`. La sérialisation en XML est assurée via `JAXB`.

```
<items>
    <item>
        <id>1</id>
        <title>Article minimal</title>
        <guid>
            https://example.com/323e4567-e89b-12d3-a456-426614174002
        </guid>
        <date>2025-05-18T06:00:00Z</date>
    </item>
    ...
</items>
```

3.3.2 HTML

— **URL** : `/rss25SB/resume/html`
— **Méthode** : `GET`
— **Retour** : page **HTML** valide

L'affichage HTML repose désormais sur une transformation **XSLT**, appliquée à un flux **XML** généré dynamiquement. Cette transformation permet de présenter les articles dans un tableau structuré en HTML, où chaque ligne correspond à un article synthétique.

— Les données sources proviennent du service `ItemService`, qui extrait les entités `ItemEntity` depuis la base et les convertit en objets `ItemSummaryDTO`.
— La liste est ensuite marshallée au format XML et transformée en HTML via le moteur `XsltTransformer` et la feuille `rss25-list.xslt`.
— Le format de date est normalisé (RFC 3339) via l'utilitaire `DateTimeUtil`.
— La colonne `guid` est un lien cliquable vers la page de détails d'un article (affichage complet).

Projet RSS25SB

Résumé des articles

ID	Titre	GUID	Date
1	En direct, guerre à Gaza : l'UE va réexaminer son accord d'association avec Israël	https://www.lemonde.fr/international/live/2025/05/20/en-direct-guerre-a-gaza-l-ue-va-reexaminer-son-accord-d-association-avec-israel_6606510_3210.html	2025-05-20T17:45:32Z
2	En Roumanie, George Simion demande l'annulation de l'élection présidentielle	https://www.lemonde.fr/international/article/2025/05/20/en-roumanie-george-simion-demande-l-annulation-de-l-election-presidentielle_6607405_3210.html	2025-05-20T16:39:44Z
3	La Californie ressent déjà les effets du « Trump slump », le ralentissement économique lié aux mesures chaotiques du président américain	https://www.lemonde.fr/economie/article/2025/05/20/la-californie-ressent-deja-les-effets-du-trump-slump-le-ralentissement-economique-lie-aux-mesures-chaotiques-du-president-americain_6607392_3234.html	2025-05-20T15:00:02Z

3.4 Détail d'un article (I.4)

Cette fonctionnalité permet d'afficher le contenu complet d'un article RSS à partir de son identifiant unique. Deux formats sont proposés : XML conforme au schéma `rss25.xsd` et HTML lisible par l'utilisateur final.

3.4.1 XML

— **URL** : `/rss25SB/resume/xml/{id}`
— **Méthode** : `GET`
— **Retour attendu** : Flux XML conforme au schéma XSD, représentant un seul article

Le contrôleur interroge le service métier `ItemService` qui effectue une recherche dans la base de données à partir de l'identifiant fourni. Deux cas peuvent survenir :

— Si l'article est trouvé, il est converti en objet JAXB `Item` et sérialisé en XML via `XmlUtil`.

— Si l'identifiant est invalide ou introuvable, une erreur structurée au format XML est retournée avec :
  — `id` : valeur fournie en entrée
  — `status` : `ERROR`
  — `description` : message expliquant la nature de l'erreur

Réponse en cas de succès :

```
<item>
    <guid>
        https://example.com/323e4567-e89b-12d3-a456-426614174002
    </guid>
    <title>Article minimal</title>
    <category term="tech"/>
    <published>2025-05-18T06:00:00Z</published>
    <content type="text"/>
    <author name="John Doe"/>
</item>
```

Réponse en cas d'erreur :

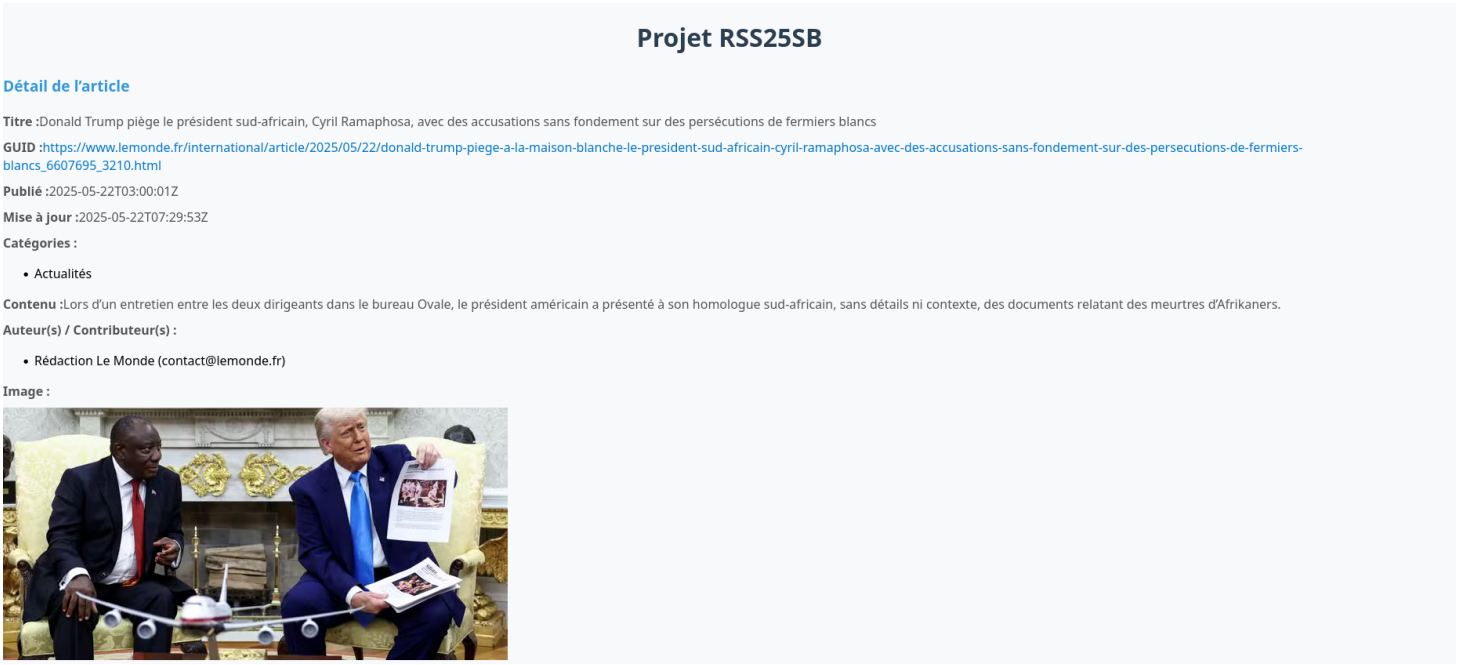
```
<error>
    <id>201</id>
    <status>ERROR</status>
    <description>
        Erreur lors de la récupération d'un flux rss25SB:
        L'article avec l'identifiant: 201 n'existe pas.
    </description>
</error>
```

3.4.2 HTML

— **URL** : `/rss25SB/html/{id}`
— **Méthode** : `GET`
— **Retour attendu** : Page HTML/XHTML contenant les informations détaillées de l'article

L'affichage HTML d'un article individuel repose sur une transformation **XSLT** appliquée à l'objet XML correspondant à l'article demandé. Le rendu est généré via la feuille de style `rss25.item.xslt`, à l'aide du moteur `XsltTransformer`.

— L'article est extrait depuis la base via le service `ItemService`, puis converti en objet `Item` (JAXB).
— L'objet est ensuite sérialisé en XML puis transformé en HTML via la feuille XSLT.
— Le rendu final présente de manière structurée :
  — Le titre, le `guid`, la date de publication et de mise à jour
  — Les auteurs et contributeurs
  — Le contenu principal
  — Les catégories associées
  — L'image, si disponible (avec ses attributs `href`, `alt`, `type`)



En cas d'erreur (article non trouvé), une page HTML dédiée est générée avec les éléments suivants :

- `id` de la requête
- `status` = `ERROR`
- Message explicatif

3.5 Insertion d'un article (I.5)

Cette fonctionnalité permet à un client de soumettre un flux XML conforme au format `rss25SB` pour insertion en base.

- **URL** : `/rss25SB/insert`
- **Méthode** : `POST`
- **Contenu envoyé** : Flux XML conforme au schéma `rss25.xsd`, représentant un flux complet de type `Feed`
- **Format du retour** : Flux XML

Fonctionnement interne Lorsqu'un flux XML est reçu :

1. Le contenu est désérialisé avec `JAXB` en objet `Feed`, après validation XSD stricte via `XmlUtil`.

2. Chaque élément `<item>` est ensuite converti en entité persistable via `ItemMapper`.
3. Pour chaque article, un contrôle d'unicité sur le champ `guid` est réalisé. Si le `guid` existe déjà en base, l'article est ignoré et une erreur est générée.
4. Si l'article est valide et unique, il est inséré en base et son identifiant auto-généré est conservé.

Structure de la réponse

— En cas de succès, la réponse XML est de la forme :

```
<inserted>
    <ids>
        <id>20</id>
    </ids>
    <status>INSERTED</status>
</inserted>
```

— En cas d'échec (flux invalide, duplicata détecté, erreur de parsing...), une réponse XML au format :

```
<inserted>
    <status>ERROR</status>
    <description>Le guid de l'article existe déjà.</description>
</inserted>
```

Contraintes respectées :

— Tous les flux entrants sont validés avec le schéma `rss25.xsd`
— Aucun article n'est persisté si une erreur est détectée
— L'attribution de l'`id` est automatique, unique, et gérée par la base
— L'opération est atomique : seuls les articles valides et uniques sont insérés

3.6 Suppression d'un article (I.5.1)

Cette opération permet de supprimer un article identifié par son identifiant unique depuis la base de données.

— **URL** : `/rss25SB/delete/{id}`
— **Méthode** : `DELETE`
— **Paramètre** : `id` (entier strictement positif représentant l'identifiant de l'article)
— **Format du retour** : Flux XML

Comportement attendu Lorsqu'un identifiant est fourni :

1. Le système vérifie si un article avec cet `id` existe en base.
2. Si oui, il est supprimé.
3. Une vérification est ensuite effectuée pour détecter si la base est vide :
   — Si aucun article ne reste, le flux complet est considéré comme vide et le fichier est supprimé (logique métier simulée).
4. Si l'article n'est pas trouvé, l'opération échoue.

Réponse en cas de succès :

```
<deleted>
    <id>42</id>
    <status>DELETED</status>
</deleted>
```

Réponse en cas d'échec :

```
<error>
    <id>122</id>
    <status>ERROR</status>
    <description>
        Erreur lors de la suppression d'un flux rss25SB:
        L'article avec l'identifiant: 122 n'a pas était trouvé
        ou a déjà été supprimé.
    </description>
</error>
```

Contraintes assurées

- La cohérence du flux RSS est maintenue : si aucun article n'est présent, le flux est considéré comme inexistant.
- Les erreurs sont retournées au format XML structuré via `XmlErrorResponseDTO`.

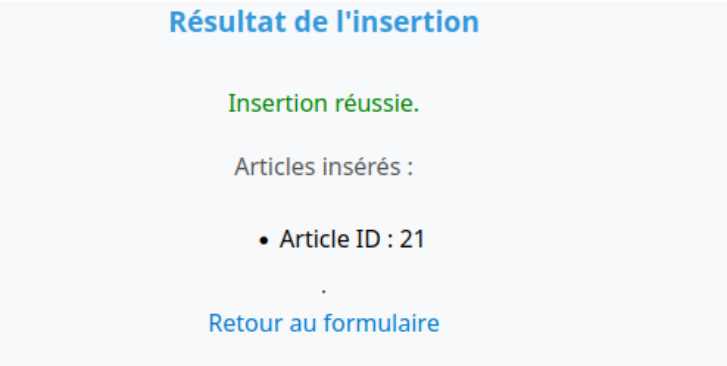
3.7 Application de transfert (III.1)

Afin de faciliter l'importation de flux XML au format `rss25SB` dans le service REST, une application cliente de transfert a été développée. Elle permet de simuler une interaction côté client et de tester facilement l'insertion de flux XML sans passer par des outils externes.

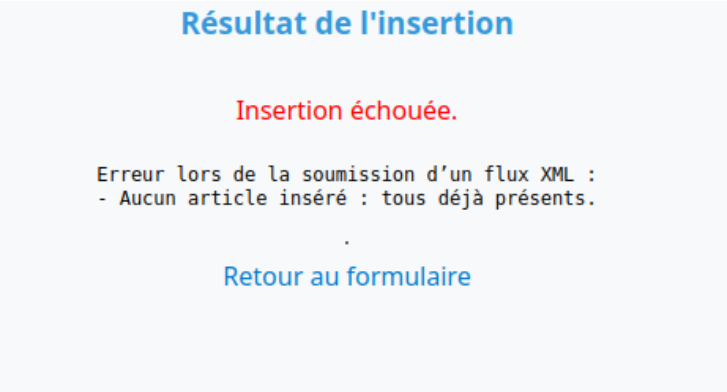
Fonctionnalités principales :

- **Interface ergonomique** : une interface graphique simple permet de sélectionner un fichier XML localement.

— **Paramétrage dynamique** : l'URL du serveur, le port, et le chemin de l'endpoint sont modifiables via des champs de saisie, ce qui rend l'outil adaptable à tous les environnements.
— **Transfert de fichier** : le fichier XML est transmis au serveur via une requête HTTP `POST` vers l'endpoint `/rss25SB/insert`.
— **Affichage du retour serveur** : une zone dédiée affiche la réponse retournée par le serveur.
  — En **cas de succès**, un message contenant un statut et la liste des identifiants insérés est affiché.



  — En **cas d'échec**, le statut `<status>ERROR</status>` est accompagné d'un élément `<description>` décrivant la cause de l'erreur (ex : flux invalide, doublons, erreur SQL...).



— **Validation optionnelle** : une case à cocher permet d'activer la validation du flux XML en local via un schéma XSD. Si activée, le fichier est validé avant tout envoi, et un message d'erreur s'affiche en cas de non-conformité.

3.8 Application de conversion (III.2)

Cette seconde version permet de récupérer un flux RSS externe (au format non conforme) depuis une source publique, de le convertir automatiquement au format `rss25SB`, puis de le transmettre au serveur via l'application de transfert décrite précédemment.

Fonctionnalités principales :

— **Sélection de la source** : l'utilisateur choisit une source parmi une liste déroulante prédéfinie (dans notre cas : `Le Monde`). Chaque source est liée à une stratégie de conversion spécifique.
— **Téléchargement automatique** : l'application récupère le flux distant via une requête HTTP, sans intervention manuelle.

— **Conversion au format** `rss25SB` : le contenu XML est transformé à l'aide d'une logique dédiée à la source sélectionnée, qui extrait les éléments pertinents et les reformate (JAXB). Par exemple :
  — les balises `<title>`, `<pubDate>`, `<guid>` sont conservées,
  — les images et auteurs sont reconstitués ou générés si absents,
  — la date de mise à jour est calculée si manquante.
— **Transfert vers le serveur** : après conversion, le fichier XML généré peut être immédiatement transféré au serveur à l'aide de l'application de transfert (réutilisation du module précédent).

Exemple de source prise en charge :
— `https://www.lemonde.fr/rss/plus-lus.xml`

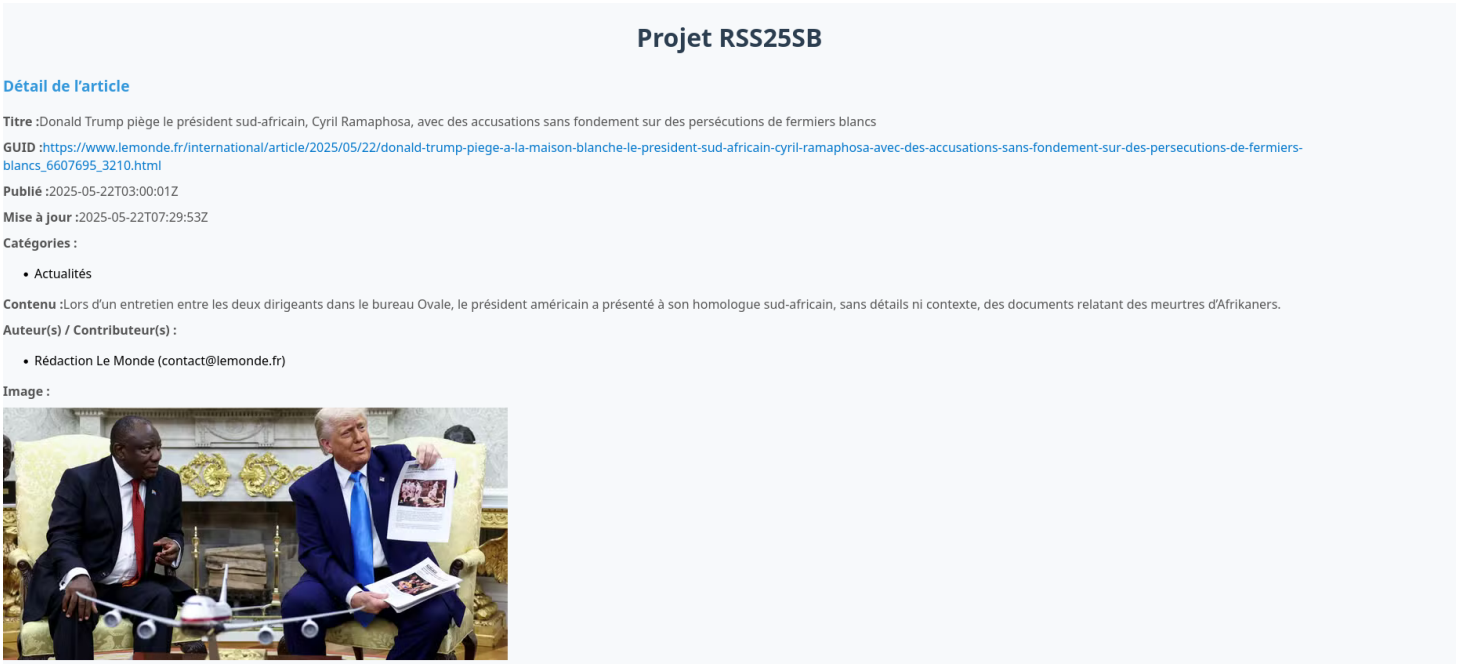


FIGURE 1 – Affichage HTML détaillé d'un article LeMonde convertit au format `rss25SB`

4 Fonctionnalités bonus

4.1 II.1 - Gestion des erreurs

Une gestion avancée des erreurs a été mise en place afin de respecter les exigences de robustesse du service et de faciliter le diagnostic des anomalies.

Objectif : Optimiser les retours XML en cas d'erreurs serveur ou client (404, 400, 500), en ajoutant une description explicite dans les réponses.

Mécanisme mis en place : Un contrôleur global personnalisé, `CustomErrorController`, implémente l'interface `ErrorController` de Spring Boot. Il intercepte les erreurs HTTP et les transforme en flux XML structurés.

Format du retour en cas d'erreur :

```
<error>
    <id>...</id>
    <status>ERROR</status>
    <description>Message détaillé de l'erreur</description>
</error>
```

Exemple d'erreur retournée :

```
<error>
    <status>ERROR</status>
    <description>
        Erreur 400 : ressource non trouvée ou invalide
        [/rss25SB/resume/xml/100a]. Message : Not Found
    </description>
</error>
```

A ANNEXE

A.1 Manuel d'utilisation (exécution locale)

Cette section décrit les étapes nécessaires pour exécuter localement le projet `rss25SB` en environnement de développement.

1. **Choix technologiques** :
   — Java 17,
   — Maven 3.8+,
   — PostgreSQL 15
2. **Lancer l'application en local** :
   (a) **Création du fichier** `.env` à la racine du projet :

Configuration

```
SPRING_DATASOURCE_URL=jdbc:postgresql://bjfnhbpjgb4ukxomwuee-
postgresql.services.clever-cloud.com:50013/bjfnhbpjgb4ukxomwuee
SPRING_DATASOURCE_USERNAME=uqqy8rc0y43dh80mkivd
SPRING_DATASOURCE_PASSWORD=KjxwfoS3IrgHReQuDQlBOeqAFwCXRD
PORT=... # Par défaut 8080
```

   (b) **Exécuter les commandes** :

Commande à exécuter

```
   export $(grep -v '^#' .env | xargs)

./mvnw spring-boot:run
↪  -Dspring-boot.run.arguments="--server.port=$PORT
↪  --spring.datasource.url=$SPRING_DATASOURCE_URL
↪  --spring.datasource.username=$SPRING_DATASOURCE_USERNAME
↪  --spring.datasource.password=$SPRING_DATASOURCE_PASSWORD"
```