Spécification Technique - Évolution NOMABIP pour la Facturation Électronique France

Author: *Franck Blettner*

Revision

Version	Date	Author	Description
1.0	02-12-2025	Franck Blettner	Initial
1.1	15-12-2025	Franck Blettner	Fonctionnalités UBL, Amélioration des l'interface graphique

Projet : bip-nomaubl **Version :** 1.0.0
Date : Décembre 2025
Statut : Draft
Repository : https://github.com/fblettner/bip-nomabip

1. Table des Matières

2. Nouvelles Fonctionnalités (v1.1 - Décembre 2025)

2.1. Interface de Validation GUI (MainModern.java)

Onglet "✓ UBL Validation" avec deux modes de validation :

Mode	Description	Prérequis
XML Source	Transformation XML → UBL puis validation	Template requis
UBL Source	Validation directe d'un fichier UBL existant	Aucun template

Affichage des résultats :

- Table multi-colonnes : Severity, Source, Rule ID, Message
- Couleurs : Rouge (ERROR/FATAL), Orange (WARNING), Vert (SUCCESS)
- Hauteur de ligne dynamique pour messages longs
- Logs formatés : `** {SEVERITY} ** {SOURCE} ** {RULE_ID} : {MESSAGE}`

Méthodes clés :

- `validateUbl()` : Orchestration validation avec choix du mode
- `validateUblDirectly()` : Validation directe sans transformation
- `createUblValidationPanel()` : Construction de l'interface

2.2. Conversion XML → UBL

Fonctionnalité : Transformation des spools XML source vers format UBL 2.1 Invoice

Configuration : Utilisation d'un paramètre dans la ligne de commande pour déclencher un UBL

Pipeline de transformation :

```
XML Spool → XSLT 2.0 (Saxon) → UBL 2.1 Invoice → Validation → PA
```

Propriétés template :

Propriété	Description	Exemple
`ublXslt`	Template XSLT transformation	jde_to_ubl_2_1.xsl
`ublOutputDir`	Répertoire sortie UBL	%PROCESS_HOME%/ubl/

Méthode : `CustomUBL.convertToUBL()`

- Processeur : Saxon TransformerFactory (support XSLT 2.0)
- Input : Document XML parsé depuis spool
- Output : Fichier UBL 2.1 XML structuré
- Nommage : `{activité}_{typePiece}_{docID}_{typeJDE}_{societeJDE}.xml`

Templates XSLT personnalisables : Chaque template métier (`invoice`, `invoice2`, etc.) peut référencer un XSLT différent pour adapter la transformation selon le type de document source.

Gestion erreurs : En cas d'échec de transformation, insertion dans `F564230_ERR` avec code erreur et arrêt du traitement pour le document.

2.3. Embedding PDF dans UBL

Fonctionnalité : Intégration du PDF facture en Base64 dans le XML UBL

Configuration : Propriété `attachment` dans template

- `create` : Embedding PDF en base64
- `attach` : Référence externe (futur)
- `none` : Pas d'attachement

Structure UBL générée :

```
<cac:AdditionalDocumentReference>
    <cbc:ID>PDF_Invoice</cbc:ID>
    <cbc:DocumentType>PDF</cbc:DocumentType>
    <cac:Attachment>
        <cbc:EmbeddedDocumentBinaryObject mimeCode="application/pdf"
filename="{docName}.pdf">
            {Base64 encoded PDF}
        </cbc:EmbeddedDocumentBinaryObject>
    </cac:Attachment>
</cac:AdditionalDocumentReference>
```

Méthode : `embedPdfInUBL(String ublFile, String pdfFile, String fileName)`

Position d'insertion : Après `AdditionalDocumentReference` existants ou avant `AccountingSupplierParty`

2.4. Optimisations Architecture

CustomUBL Constructor :

- Ajout paramètre `UBLValidator` pour partage d'instance entre threads
- Élimination de multiples instanciations coûteuses
- Signature : `CustomUBL(..., UBLValidator validator)`

Gestion InputStream :

- Recréation systématique avant `convertToUBL()` pour éviter consommation
- Protection null pointer sur `pAttachment`
- Compatibilité Oracle XML Parser (suppression propriété non-standard)

3. Contexte et Objectifs

3.1. Contexte Réglementaire

La réforme de la facturation électronique en France impose à toutes les entreprises assujetties à la TVA d'adopter la facturation électronique pour leurs transactions B2B.

Échéance	Obligation
1er septembre 2026	Réception obligatoire pour toutes les entreprises + Émission pour GE/ETI
1er septembre 2027	Émission obligatoire étendue aux PME et micro-entreprises

Les factures doivent être transmises via une **Plateforme Agréée (PA)** dans l'un des trois formats structurés autorisés : UBL 2.1, CII, ou Factur-X. Cette spécification se concentre sur la production de fichier **UBL 2.1**.

3.2. Références

- **OASIS UBL 2.1** : https://docs.oasis-open.org/ubl/UBL-2.1.html
- **Norme EN 16931** : Norme sémantique européenne de la facture électronique
- **Schematron EN 16931** : référentiel officiel de validation sémantique.
- **CIUS-FR / Extension FR** : adaptation française de l'EN 16931 (règles spécifiques France).
- **Normes AFNOR << Facture électronique >>** : https://fnfe-mpe.org/
- **GitHub NomaBip** : https://github.com/fblettner/bip-nomabip
- **Documentation NomaBip** : https://docs.nomana-it.fr/api/bip-api/nomabip/
- **DGFiP Facturation électronique** : https://www.impots.gouv.fr/facturation-electronique
- **Github Repository XSL to manage schematron** : https://github.com/Schematron/schematron/tree/master/trunk/schematron/code
- **Règles de gestion BR-FR en application de la Norme XP Z12-012** : https://fnfe-mpe.org/wp-content/uploads/2025/12/2025_12_04_FNFE_SCHEMATRONS_FR_CTC_V1.2.0.zip

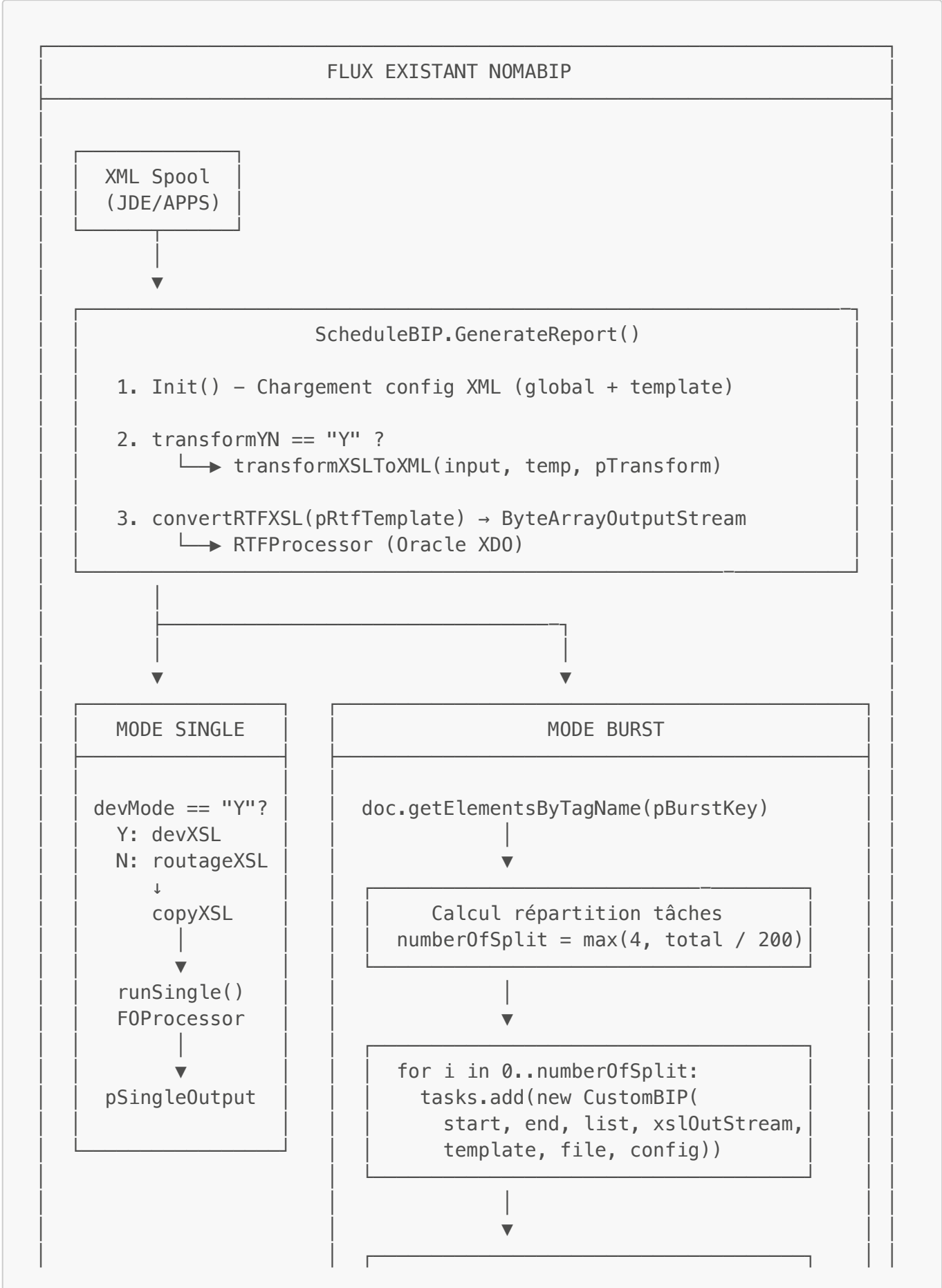
3.3. Objectifs de l'Évolution

NOMABIP est une API Java développée pour intégrer JD Edwards avec Oracle BI Publisher. Elle offre déjà des fonctionnalités clés qui seront réutilisées. L'évolution proposée vise à étendre NOMABIP pour supporter la facturation électronique française :

1. **Réutiliser l'infrastructure existante** (`ScheduleBIP`, `CustomBIP`) pour la parallélisation
2. **Étendre le système de transformation XSL** existant pour générer du UBL 2.1
3. **Valider les factures UBL** avant envoi à la Plateforme Agréée
4. **Étendre le modèle de données Oracle** existant pour le cycle de vie
5. **S'interfacer avec les PA** pour l'envoi et le suivi des statuts

4. Architecture Existante

4.1. Diagramme de Flux



4.2. Classes Principales

L'architecture actuelle repose sur deux classes Java principales :

Classe	Package	Responsabilité
`ScheduleBIP`	`nomabip`	Orchestration, configuration, modes SINGLE/BURST
`CustomBIP`	`custom.bip`	Tâche parallélisable (`Callable<Integer>`), traitement unitaire

Le principe de fonctionnement est le suivant :

- Lit des **spools XML** générés par JDE ou d'autres sources.
- Transforme ces spools en **PDF** via BI Publisher (FOProcessor) + template RTF/XSL-FO.
- Génère un **XML d'index** via XSLT.
- Gère une **base Oracle** pour journaliser les traitements (log, erreurs, éventuellement BLOB du spool).
- S'appuie sur un **système de templates** de configuration, combinant :
  - un template `global` (paramètres communs),
  - un ou plusieurs templates fonctionnels (`invoice`, `invoice_alt`, etc.).
- Effectue le traitement en **multi-thread** via :
  - une classe orchestratrice `ScheduleBIP`,
  - une classe worker `CustomBIP` qui implémente `Callable<Integer>`.

4.3. Fichier de Configuration XML

Principe :

- Le template `global` fournit les **propriétés communes**.
- Chaque template métier (`invoice`, `invoice_alt`, etc.) fournit :
  - les tags XML utilisés (`burstKey`, `docID`, `societeJDE`, etc.),
  - les styles de sortie (`rtf`, `xsl`, `transform`),
  - des paramètres spécifiques (nombre de threads, routage, etc.)

4.4. Ligne de commande

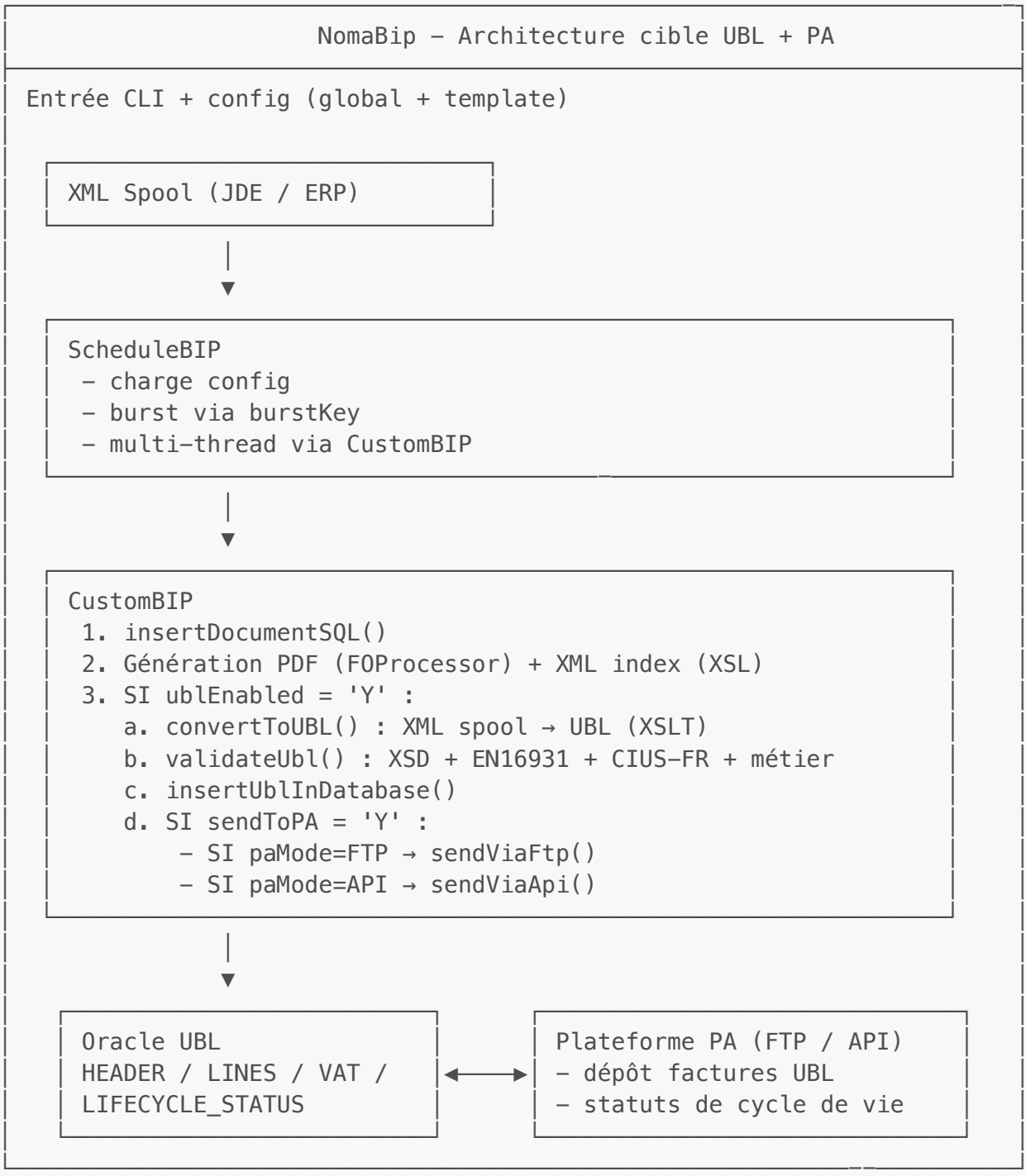
Exemple :

```
java -jar ../dist/nomabip.jar -run ./config/config.properties invoice
R42565_FBL0001_FR_1 BURST 1
```

- `config.properties` : fichier de configuration.
- `invoice` : nom du template métier à utiliser.
- `R42565_FBL0001_FR_1` : nom du fichier spool (ou identifiant de traitement).
- `BURST` : mode burst (1 spool → N documents).
- `1` : identifiant supplémentaire (job id / run id).

5. Architecture cible avec UBL et Plateforme Agréée

5.1. Schéma cible – Vue globale



5.2. Principes

L'architecture cible :

- **ajoute** un pipeline UBL aux traitements existants,
- **n'altère** pas la génération PDF + XML d'index,

- est **paramétrable par template** (tous les flux ne seront pas UBL),
- se charge aussi de l'**envoi** vers la Plateforme Agréée :
  - par **FTP/SFTP** (mode "fichiers déposés"),
  - par **API REST**,
- gère la **récupération des statuts** et leur stockage Oracle.

L'évolution s'appuie donc sur l'architecture existante **sans la modifier** :

1. **Nouveau XSL** `ublTransform` : Ajouté dans la configuration template, appelé après les XSL existants
2. **Nouvelle méthode** `convertToUBL()` : Ajoutée dans `CustomBIP`, similaire à `convertToXML()`
3. **Nouvelles classes** : `UBLValidator`, `InvoiceRepository`, `PAConnector` dans un nouveau package

6. Évolutions de configuration

> **Ajouter des propriétés UBL & PA dans le template `global` et dans les templates métier existants.**

6.1. Ajouts dans `template name="global"`

```
<template name="global">
   ...
   <!-- UBL global -->
    <property name="ublXsdPath" value="%APP_HOME%/xsd/main/UBL-Invoice-
2.1.xsd"/>
    <property name="ublSchematronPath" value="%APP_HOME%/schematron"/>

   <!-- Paramètres Plateforme Agréée -->
   <property name="paMode" value="FTP"/>  <!-- FTP | API -->

   <!-- FTP -->
   <property name="paFtpHost" value="sftp.pa.com"/>
   <property name="paFtpPort" value="22"/>
   <property name="paFtpUser" value=""/>
   <property name="paFtpPassword" value=""/>
   <property name="paFtpOutboundDir" value="/out/invoices/"/>
   <property name="paFtpInboundDir" value="/in/status/"/>

   <!-- API -->
   <property name="paApiBaseUrl" value="https://api.com/v1"/>
   <property name="paApiTokenUrl" value="https://api.com/oauth2/token"/>
   <property name="paApiClientId" value=""/>
   <property name="paApiClientSecret" value=""/>
   <property name="paApiInvoiceEndpoint" value="/invoices"/>
   <property name="paApiStatusEndpoint" value="/status"/>
   <property name="paApiTimeout" value="30"/>

</template>
```

6.2. Ajouts dans les templates métier (`invoice`, `invoice2`, etc.)

```
<template name="invoice">
   ...
   <!-- XSLT spool → UBL -->
   <property name="ublXslt" value="%UBL_XSLT_HOME%/xml_to_ubl_2_1.xsl"/>
   <!-- PDF Attachment in UBL: create = embed PDF in UBL, attach =
reference only, none = no attachment -->
   <property name="attachment" value="create"/>
</template>
```

7. Évolutions `CustomUBL` (pipeline détaillé)

7.1. Propriétés Java UBL

Classe `CustomUBL` (package `custom.ubl`) :

```
// UBL
private String pUblEnabled;           // "Y" / "N"
private String pUblXsltPath;         // chemin XSLT UBL pour ce template
private String pUblOutputDir;        // répertoire sortie UBL
private String pUblXsdMain;          // XSD principal
private String pUblValidationMode;   // FULL / XSD_ONLY / NONE
private boolean ublValidationXsd;
private boolean ublValidationBusiness;
private UBLValidator pUBLValidator;      // Instance validateur (passée en
paramètre)
private String pAttachment;          // "create" / "attach" / "none" – PDF
embedding in UBL

// Plateforme Agréée
private String pSendToPA;            // "Y" / "N"
private String pPaMode;              // "FTP" / "API"
private String pPaName;              // "PA"

private String pPaFtpHost;
private int    pPaFtpPort;
private String pPaFtpUser;
private String pPaFtpPassword;
private String pPaFtpOutboundDir;
private String pPaFtpInboundDir;

private String pPaApiBaseUrl;
private String pPaApiTokenUrl;
private String pPaApiClientId;
private String pPaApiClientSecret;
private String pPaApiInvoiceEndpoint;
private String pPaApiStatusEndpoint;
private int    pPaApiTimeout;
```

7.2. Configuration et Initialisation

Propriétés UBL (template XML) :

Propriété	Description	Valeurs
`ublXsdPath`	Répertoire des schémas XSD UBL 2.1	Chemin absolu
`ublSchematronPath`	Répertoire des fichiers Schematron	Chemin absolu
`ublXsltPath`	Transformation XML → UBL	Chemin vers .xsl

Propriété	Description	Valeurs
`ublOutputDir`	Répertoire de sortie UBL	Chemin absolu
`attachment`	Embedding PDF	create/attach/none
`sendToPA`	Envoi à la plateforme	Y/N

Propriétés Plateforme Agréée :

Propriété	Description
`paMode`	Mode de transmission : FTP/API
`paFtp*`	Paramètres FTP (host, port, user, pass, dir)
`paApi*`	Paramètres API (baseUrl, endpoints, auth)

Architecture : Le validateur `UBLValidator` est instancié une fois dans `ScheduleUBL` et partagé entre threads pour optimisation.

7.3. Pipeline de Traitement

Flux CustomUBL.call() :

```
1. Init() + Connexion Oracle (si updateDB=Y)
   ↓
2. Pour chaque document (startInvoice → endInvoice) :
   ↓
   a) Extraction métadonnées
      - docID, activité, typePiece, typeJDE, societeJDE
      - Nom fichier :
{activité}_{typePiece}_{docID}_{typeJDE}_{societeJDE}
   ↓
   b) Mode BURST/BOTH :
      → FOProcessor (PDF)
      → Ghostscript (compression si configuré)
      → XSLT (XML index)
   ↓
   c) Mode UBL/BOTH :
      → convertToUBL() : XSLT 2.0 (Saxon)
      → embedPdfInUBL() : si attachment=create/attach
      → Parser UBL
      → Validation (pUBLValidator.validateUbl())
      →┬─ ✓ Succès : insertUblInDatabase() + sendToPlatform()
       └─ ✗ Échec : Logs erreurs formatées
   ↓
   d) Gestion erreurs : insertErrorSQL()
   ↓
3. Fermeture connexion Oracle
```

Transformation XML → UBL :

- Processeur : Saxon TransformerFactory (XSLT 2.0)
- Template : `pUblXsltPath` configuré par template
- Multi-templates : Chaque template peut utiliser un XSLT différent

Embedding PDF Base64 :

- Position d'insertion : Après `AdditionalDocumentReference` existants ou avant `AccountingSupplierParty`
- Structure UBL conforme : cf. section 1.1.3
- Activation : Propriété `attachment=create` ou `attach`

Chaque template `invoice*` peut pointer vers un XSLT différent :

- `xml_to_ubl.xsl`
- `xml_to_ubl2.xsl`
- etc.

7.4. Architecture XSLT - Modules Réutilisables

7.4.1. Vue d'ensemble

L'architecture XSLT s'appuie sur des **modules réutilisables** pour simplifier le développement et garantir la conformité UBL 2.1 / EN16931 / CIUS-FR.

Principe de composition :

```
Template métier (vrac_pro_ubl.xsl, isc_facture_ubl.xsl, etc.)
    ↓ import
├── ubl-common.xsl      : Fonctions utilitaires
└── ubl-defaults.xsl    : Templates par défaut et règles métier
```

Processeur : Saxon HE 9.x (support XSLT 2.0 + XPath 2.0 + fonctions personnalisées)

7.4.2. Module ubl-common.xsl

Responsabilité : Bibliothèque de fonctions utilitaires pour manipulation de données

Namespace : `xmlns:ubl="urn:nomana:ubl:common"`

Fonctions principales :

Fonction	Signature	Description
`ubl:is-not-empty`	`xs:string? → xs:boolean`	Teste si chaîne non vide après normalisation
`ubl:normalize-amount`	`xs:string? → xs:string`	Convertit montant FR "1 160,55 €" → "1160.55"
`ubl:normalize-percent`	`xs:string? → xs:string`	Convertit taux "20,00%" → "20.00"
`ubl:normalize-quantity`	`xs:string? → xs:string`	Normalise quantité (4 décimales max)
`ubl:extract-tax-rate`	`xs:string? → xs:string`	Extrait taux depuis texte descriptif
`ubl:unit-code`	`xs:string? → xs:string`	Convertit UM (T→TNE, L→LTR, KG→KGM, UN→C62)
`ubl:validate-tax-rate`	`xs:string?, xs:string → xs:string`	Valide taux TVA avec marqueur erreur

Templates nommés :

Template	Paramètres	Description

Template	Paramètres	Description
`ubl:emit`	qname, value	Émet élément XML si valeur non vide
`ubl:emit-scheme`	qname, value, code, codeValue	Émet élément avec attribut schemeID

Exemple d'utilisation :

```
<!-- Normalisation montant -->
<xsl:variable name="normalizedAmount"
    select="ubl:normalize-amount('1 234,56 €')" />
<!-- Résultat : "1234.56" -->

<!-- Émission conditionnelle -->
<xsl:call-template name="ubl:emit">
    <xsl:with-param name="qname" select="'cbc:ID'" />
    <xsl:with-param name="value" select="$documentNumber" />
</xsl:call-template>
<!-- Émet <cbc:ID>FAC123</cbc:ID> uniquement si $documentNumber non vide -
->
```

7.4.3. Module ubl-defaults.xsl

Responsabilité : Templates prédéfinis conformes EN16931 et CIUS-FR

Namespace : `xmlns:ubl="urn:nomana:ubl:common"`

Variables globales :

Variable	Valeur	Description
`$defaultCurrency`	EUR	Devise par défaut

Templates de structure :

Template	Paramètres	Description
`ubl:invoice-header`	ublVersion, customizationID, profileID	En-tête UBL standard
`ubl:invoice-type-code`	code, default	Code type facture (défaut: 380)
`ubl:currency-code`	code, default	Devise document (défaut: EUR)

Templates d'identification :

Template	Paramètres	Description
`ubl:party-siren`	siren, missingMarker	PartyIdentification avec schemeID=0002
`ubl:company-siren`	siren, missingMarker, invalidMarker	CompanyID avec validation longueur=9

Template	Paramètres	Description
`ubl:party-siret`	siret, missingMarker	PartyIdentification avec schemeID=0009
`ubl:party-gln`	gln, missingMarker	PartyIdentification avec schemeID=0088
`ubl:company-vat`	vat, missingMarker, invalidMarker	CompanyID TVA avec validation préfixe FR
`ubl:party-tax-vat`	vat, missingMarker, invalidMarker	PartyTaxScheme VAT complet
`ubl:endpoint-id`	id, schemeID, missingMarker	EndpointID (défaut: 0225)
`ubl:country`	code, default	Country avec IdentificationCode (défaut: FR)

Templates de paiement :

Fonction	Paramètres	Description
`ubl:payment-code`	code → xs:string	Mapping codes paiement (S→49, 4→30)

Templates légaux français :

Template	Paramètres	Description
`ubl:note-payment-delay`	-	Note pénalités retard (3× taux légal)
`ubl:note-recovery-fee`	-	Note indemnité forfaitaire (40€)
`ubl:notes-french-legal`	AAB	Ensemble notes légales + CGV optionnelles

Validation et marqueurs d'erreur :

- Tous les templates d'identification acceptent des `missingMarker` et `invalidMarker`
- Valeurs par défaut : `**MISSING_XXX**`, `**INVALID_XXX_FORMAT**`
- Génère des erreurs détectables par validation Schematron

Exemple d'utilisation :

```
<!-- En-tête standard -->
<xsl:call-template name="ubl:invoice-header" />
<!-- Génère :
<cbc:UBLVersionID>2.1</cbc:UBLVersionID>
<cbc:CustomizationID>urn:cen.eu:en16931:2017</cbc:CustomizationID>
<cbc:ProfileID>M1</cbc:ProfileID>
-->

<!-- SIREN fournisseur avec validation -->
<xsl:call-template name="ubl:company-siren">
    <xsl:with-param name="siren" select="$supplierSiren" />
</xsl:call-template>
<!-- Génère : <cbc:CompanyID schemeID="0002">123456789</cbc:CompanyID>
     ou : <cbc:CompanyID schemeID="0002">**INVALID_SIREN_LENGTH**
```

```
</cbc:CompanyID> si longueur ≠ 9
-->

<!-- Notes légales françaises -->
<xsl:call-template name="ubl:notes-french-legal">
    <xsl:with-param name="AAB" select="$conditionsGenerales" />
</xsl:call-template>
```

7.4.4. Template métier - Exemple vrac_pro_ubl.xsl

Responsabilité : Transformation spécifique d'un format XML source vers UBL 2.1

Structure :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="urn:oasis:names:specification:ubl:schema:xsd:Invoice-2"

xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateCom
ponents-2"

xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicCompone
nts-2"
    xmlns:ubl="urn:nomana:ubl:common"
    exclude-result-prefixes="xsl ubl">

    <!-- Import des modules réutilisables -->
    <xsl:import href="ubl-common.xsl" />
    <xsl:import href="ubl-defaults.xsl" />

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
/>

    <!-- Template racine -->
    <xsl:template match="/">
        <!-- Extraction variables depuis XML source -->
        <xsl:variable name="invoiceGroup"
            select="/*/*[local-name()='Numero_de_Facture_LBH_DOC_S6']" />
        <xsl:variable name="documentNumber"
            select="$invoiceGroup/*[local-name()='DocumentNumber']" />
        <xsl:variable name="issueDate"
            select="$invoiceGroup/*[local-name()='DateFacture']" />
        <xsl:variable name="CGV"
            select="$invoiceGroup/*[local-
name()='Message_Condition_générale_de__ID47']" />

        <!-- Document UBL -->
        <Invoice>
            <!-- En-tête standard via template réutilisable -->
            <xsl:call-template name="ubl:invoice-header" />
```

```
            <!-- Identifiants avec validation -->
            <xsl:call-template name="ubl:emit">
                <xsl:with-param name="qname" select="'cbc:ID'" />
                <xsl:with-param name="value" select="$documentNumber" />
            </xsl:call-template>

            <!-- Dates -->
            <xsl:call-template name="ubl:emit">
                <xsl:with-param name="qname" select="'cbc:IssueDate'" />
                <xsl:with-param name="value" select="$issueDate" />
            </xsl:call-template>

            <!-- Type de facture -->
            <xsl:call-template name="ubl:invoice-type-code">
                <xsl:with-param name="code" select="'380'" />
            </xsl:call-template>

            <!-- Notes légales françaises -->
            <xsl:call-template name="ubl:notes-french-legal">
                <xsl:with-param name="AAB" select="$CGV" />
            </xsl:call-template>

            <!-- Devise -->
            <xsl:call-template name="ubl:currency-code" />

            <!-- Fournisseur, Client, Lignes... -->
            <!-- ... reste de la transformation ... -->
        </Invoice>
    </xsl:template>
</xsl:stylesheet>
```

Avantages de l'architecture modulaire :

- **Maintenabilité** : Correction centralisée dans `ubl-common.xsl` ou `ubl-defaults.xsl`
- **Conformité** : Templates garantissent respect EN16931 et CIUS-FR
- **Productivité** : Templates métier courts (focus sur mapping source→UBL)
- **Validation** : Marqueurs d'erreur standardisés détectables par Schematron
- **Évolutivité** : Ajout de nouveaux templates métier sans duplication de code

Bonnes pratiques :

1. Toujours importer `ubl-common.xsl` et `ubl-defaults.xsl`
2. Utiliser `ubl:emit` pour éléments optionnels (évite éléments vides)
3. Normaliser montants/quantités via fonctions `ubl:normalize-*`
4. Utiliser templates d'identification avec marqueurs d'erreur
5. Inclure notes légales françaises via `ubl:notes-french-legal`

8. Module de Validation UBL

8.1. Classe UBLValidator

Package : `custom.ubl`

Responsabilité : Validation multi-niveaux des documents UBL

Construction :

- Paramètres : `xsdPath`, `schematronPath`
- Initialisation du schéma XSD UBL 2.1
- Compilation des fichiers Schematron (.sch) en XSLT via transformations ISO
- Création d'un cache de transformations compilées pour optimisation

M\u00e9thodes principales :

Méthode	Description	Retour
`validateUbl(Document)`	Validation complète : XSD + EN16931 + CIUS-FR	`ValidationResult`
`validateXSD(Document)`	Validation structure XSD uniquement	`ValidationResult`
`validateSchematron(Document, String profile)`	Validation Schematron spécifique (EN16931 ou CIUS-FR)	`ValidationResult`

Niveaux de validation :

1. **XSD** : Validation structurelle UBL 2.1
2. **EN16931** : Règles sémantiques européennes
3. **CIUS-FR** : Règles spécifiques France (SIREN, TVA, etc.)

Technologie : Saxon pour XSLT 2.0/Schematron

8.2. Classes de Support

ValidationResult (package `custom.ubl`)

- Conteneur pour résultats de validation
- Collections : `errors` (List), `warnings` (List)
- Méthodes : `isValid()`, `addError()`, `merge()`, `getErrorsSummary()`

ValidationError (package `custom.ubl`)

- Représente une erreur/warning de validation
- Attributs :
  - `source` : XSD, EN16931, CIUS-FR, METIER
  - `severity` : FATAL, ERROR, WARNING
  - `message` : Description de l'erreur
  - `location` : XPath ou ligne/colonne

- `ruleId` : Identifiant de la règle Schematron

ValidationErrorHandler (package `custom.ubl`)

- Implémente `org.xml.sax.ErrorHandler`
- Capture les erreurs XSD lors du parsing
- Convertit SAXParseException en ValidationError

Format de sortie console :

```
** {SEVERITY} ** {SOURCE} ** {RULE_ID} : {MESSAGE}
```

Exemple :

```
** ERROR ** CIUS-FR ** FR-01 : Le SIREN du fournisseur est obligatoire
** WARNING ** EN16931 ** BR-45 : La devise devrait être EUR pour la France
** SUCCESS ** UBL ** FACTURE : validation successful for ISC_FACTURE_12345
```

9. Chargement UBL en base Oracle

9.1. Tables

> Table structure is an example and have to be adapted because they must be created into JD Edwards

```
CREATE TABLE UBL_INVOICE_HEADER (
  ID                NUMBER(18)      PRIMARY KEY,
  DOC_ID            VARCHAR2(50)    NOT NULL,
  SOURCE_SYSTEM     VARCHAR2(30),
  ISSUE_DATE        DATE,
  DUE_DATE          DATE,
  SUPPLIER_ID       VARCHAR2(50),
  SUPPLIER_NAME     VARCHAR2(200),
  CUSTOMER_ID       VARCHAR2(50),
  CUSTOMER_NAME     VARCHAR2(200),
  CURRENCY          VARCHAR2(3),
  TOTAL_WITHOUT_TAX NUMBER(18, 2),
  TOTAL_TAX         NUMBER(18, 2),
  TOTAL_WITH_TAX    NUMBER(18, 2),
  UBL_VERSION       VARCHAR2(10),
  PROFILE_ID        VARCHAR2(100),
  CREATION_DATE     DATE DEFAULT SYSDATE
);

CREATE TABLE UBL_INVOICE_LINE (
  ID                NUMBER(18)      PRIMARY KEY,
  HEADER_ID         NUMBER(18)      REFERENCES UBL_INVOICE_HEADER(ID),
  LINE_NUMBER       NUMBER(9),
  ITEM_ID           VARCHAR2(50),
  ITEM_DESCRIPTION  VARCHAR2(2000),
  QUANTITY          NUMBER(18, 4),
  UNIT_PRICE        NUMBER(18, 6),
  LINE_AMOUNT       NUMBER(18, 2),
  TAX_CATEGORY      VARCHAR2(10),
  TAX_PERCENT       NUMBER(5, 2)
);

CREATE TABLE UBL_VAT_SUMMARY (
  ID                NUMBER(18)      PRIMARY KEY,
  HEADER_ID         NUMBER(18)      REFERENCES UBL_INVOICE_HEADER(ID),
  TAX_CATEGORY      VARCHAR2(10),
  TAX_PERCENT       NUMBER(5, 2),
  TAXABLE_AMOUNT    NUMBER(18, 2),
  TAX_AMOUNT        NUMBER(18, 2)
);

CREATE TABLE UBL_LIFECYCLE_STATUS (
  ID                NUMBER(18)      PRIMARY KEY,
  HEADER_ID         NUMBER(18)      REFERENCES UBL_INVOICE_HEADER(ID),
```

```
  STATUS_CODE       VARCHAR2(40),
  STATUS_DATE       DATE,
  SOURCE_PLATFORM   VARCHAR2(50),
  RAW_STATUS        VARCHAR2(4000),
  COMMENT           VARCHAR2(4000)
);
```

9.2. Méthode d'insertion (pseudo-code)

```
private boolean insertUblInDatabase(Connection conn,
                                    String ublFilePath,
                                    String docID,
                                    String activite,
                                    String typePiece,
                                    String typeJDE,
                                    String societeJDE) {
    if (!"Y".equalsIgnoreCase(pUpdateDB)) return true;

    try {
        // parse UBL
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(ublFilePath));

        long headerId = getNextSequenceValue(conn,
"SEQ_UBL_INVOICE_HEADER");

        // extractions diverses (ID, dates, montants...)
        // INSERT HEADER
        // INSERT LINES
        // INSERT VAT_SUMMARY

        conn.commit();
        return true;

    } catch (Exception e) {
        return false;
    }
}
```

10. Intégration Plateforme Agréée

10.1. Configuration API

Propriétés globales (config.properties) :

Propriété	Description	Exemple
`sendToPA`	Activation envoi PA	Y/N/F
`paMode`	Mode de transmission	API/FTP
`paApiBaseUrl`	URL de base API PA	https://url_to_pa
`paApiLoginEndpoint`	Endpoint authentification	/api/login_check
`paApiImportEndpoint`	Endpoint import UBL	/api/v1/sale/invoices/import
`paApiUsername`	Identifiant API	-
`paApiPassword`	Mot de passe API	-
`paApiTimeout`	Timeout requêtes (ms)	30000

Valeurs sendToPA :

- `Y` : Envoi si validation réussie (pas d'erreur, pas de warning)
- `F` : Force envoi si validation réussie OU seulement warnings (utile pour CIUS-FR)
- `N` : Pas d'envoi

10.2. Gestion Token (TokenManager)

Classe : `custom.ubl.TokenManager`

Responsabilité : Gestion centralisée et thread-safe du token JWT

Caractéristiques :

- Token partagé entre tous les workers CustomUBL
- Refresh automatique après 55 minutes (avant expiration 60min)
- Retry automatique en cas de 401 Unauthorized

Méthodes principales :

Méthode	Description
`getToken()`	Retourne token valide, refresh si expiré
`refreshToken()`	Force refresh (après 401)
`invalidateToken()`	Invalide token pour forcer refresh

Initialisation : Dans `ScheduleUBL.GenerateReport()`, une instance unique est créée avant le lancement des workers :

```
TokenManager tokenManager = new TokenManager(
    pPaApiBaseUrl,
    pPaApiLoginEndpoint,
    pPaApiUsername,
    pPaApiPassword,
    pPaApiTimeout
);

// Pre-fetch pour fail-fast si credentials invalides
String initialToken = tokenManager.getToken();
```

10.3. Envoi API

Flux d'authentification :

1. **POST** `{baseUrl}/api/login_check`
   - Body : `{"username":"...", "password":"..."}`
   - Response : `{"token":"JWT_TOKEN"}`
   - Token stocké avec expiry time (55min)

Flux d'import UBL :

1. **Obtention token** via `TokenManager.getToken()`
2. **Lecture UBL** et encodage Base64
3. **POST** `{baseUrl}/api/v1/sale/invoices/import`
   - Header : `Authorization: Bearer {token}`
   - Body : `{"format":"xml_ubl", "content":"{base64}", "postActions":[]}`
4. **Gestion réponse :**
   - 200-299 : Succès
   - 401 : Token expiré → refresh et retry automatique
   - Autre : Erreur → log et insertion error SQL

Retry Logic :

```
Tentative 1 → 401 → TokenManager.refreshToken() → Tentative 2
```

10.4. Modes de Traitement

Mode	UBL Generation	Validation	PA Sending
`UBL`	✓	✓	✓ (si sendToPA=Y/F)
`BOTH`	✓	✓	✓ (si sendToPA=Y/F)
`UBL_VALIDATE`	✓	✓	✗ (GUI validation)
`BURST`	✗	✗	✗

Mode	UBL Generation	Validation	PA Sending
`SINGLE`	×	×	×

Note : Le mode `UBL_VALIDATE` est utilisé par l'interface graphique pour valider sans envoyer à la PA

11. Tests

11.1. Tests unitaires

Objectif : sécuriser chaque brique technique isolément.

- **Tests XSLT spool → UBL** :
  - Donnée spool d'exemple JDE → UBL d'exemple attendu (XMLUnit).
- **Tests `UblValidator`** :
  - UBL valide → `isValid() = true`.
  - UBL volontairement invalide XSD → erreur identifiée.
  - UBL avec incohérence montants → erreur métier.
- **Tests `insertUblInDatabase`** :
  - Vérifier l'insertion d'un UBL simple dans HEADER / LINES / VAT.
- **Tests de mapping statuts** :
  - Input JSON/ XML → ligne `UBL_LIFECYCLE_STATUS` correcte.

11.2. Tests d'intégration

- **Scénario complet PDF + UBL + Oracle** :
  - Spool → PDF + XML index + UBL + insertion DB.
- **Scénario FTP** :
  - UBL écrit dans `paFtpOutboundDir`.
  - Fichier de statut fictif dans `paFtpInboundDir`.
  - Job StatusFetcher → mise à jour `UBL_LIFECYCLE_STATUS`.
- **Scénario API** :
  - Mock serveur (WireMock ou équivalent).
  - POST /invoices → réponse JSON reçue.
  - GET /status/{id} → mise à jour des statuts.

11.3. Tests de non-régression

- Templates `ublEnabled = "N"` :
  - PDF + XML index strictement identiques au comportement historique.
  - Temps d'exécution et charge conformes.

11.4. Tests de charge

- Exécution avec :
  - `numProc` élevé,
  - spool volumineux,
  - activation UBL + envoi PA.
- Vérifier :
  - temps moyen par facture,
  - absence de fuite mémoire,
  - stabilité des connexions Oracle / réseau.

12. Compatibilité ascendante

- La ligne de commande ne change pas.
- Le pipeline PDF / XML index ne change pas.

13. Dépendances et Technologies

13.1. Dépendances Existantes (inchangées)

Bibliothèque	Version	Usage
Oracle XDO	-	RTFProcessor, FOProcessor
SimpleXML	2.x	Désérialisation config
Apache Commons IO	2.x	FileUtils
Oracle JDBC	19+	Connexion DB

13.2. Nouvelles Dépendances

```
<!-- Validation Schematron -->
<dependency>
    <groupId>com.helger.schematron</groupId>
    <artifactId>ph-schematron-pure</artifactId>
    <version>7.1.2</version>
</dependency>

<!-- HTTP Client pour API PA -->
<dependency>
    <groupId>org.apache.httpcomponents.client5</groupId>
    <artifactId>httpclient5</artifactId>
    <version>5.3</version>
</dependency>

<!-- JSON -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.16.0</version>
</dependency>
```

14. Annexes

14.1. Codes TVA France

Code	Catégorie	Description
S	Standard	TVA 20%
AA	Réduit	TVA 5.5% / 10%
Z	Zéro	Exportations
E	Exonéré	Exonération TVA
AE	Autoliquidation	TVA due par preneur

14.2. Statuts PA Obligatoires

Statut	Description
DEPOSEE	Facture déposée sur la PA
REJETEE	Rejetée par la PA
MISE_A_DISPOSITION	Disponible pour le client
REFUSEE	Refusée par le client
ENCAISSEE	Paiement confirmé