ATIH

Nos données
au service
de la Santé

API e-Satis 2.0

Guide intégrateur

Juin 2026

Table des matières

1. Introduction 4
2. Principe général 4
3. Diagramme de fonctionnement 5
4. Présentation de l'intégrateur 6
5. Authentification 7
   - 5.1. Header 7
   - 5.2. Payload (Json) 7
6. Dépôt de fichier 7
   - 6.1. Endpoint 7
   - 6.2. Exemple CURL 7
   - 6.3. Réponse en Json 7
7. Polling de l'état (IMPORTANT) 8
   - 7.1. Endpoint 8
   - 7.2. Exemple CURL 8
   - 7.3. États possibles 8
   - 7.4. Exemple de réponse : 8
8. Stratégie de polling 9
   - 8.1. Recommandation 9
   - 8.2. Pseudo-code polling 9
9. Récupération du log 9
   - 9.1. Endpoint 9
   - 9.2. CURL 9
10. Gestion des erreurs 10
11. Gestion des retry (client) 10
   - 11.1. Recommandation 10
   - 11.2. Pseudo-code retry 10
12. Exemple complet PHP 11
13. Exemple Java (HttpClient complet) 12
14. Exemples complets – Python / NodeJS 13
   - 14.1. Exemple Python complet (requests) 13

14.2. Exemple NodeJS (axios) 14

15. Collection Postman 15

15.1. Variables environnement 15

15.2. Requête 1 – Dépôt 15

15.3. Script Postman (très important) 15

15.4. Requête 2 – Etat 16

15.5. Requête 3 – Log 16

16. Bonnes pratiques clés 17

17. Conclusion 17

1. Introduction

Ce document a pour objectif d'accompagner les établissements et les intégrateurs dans la mise en œuvre de l'API e-Satis 2.0.
Il décrit de manière structurée les étapes nécessaires pour :

- envoyer des données à l'API,
- suivre leur traitement,
- récupérer les résultats,
- et gérer les éventuels cas d'erreur.

L'API e-Satis 2.0 intègre la solution Delta Dromeus, contribuant à renforcer la sécurisation des échanges (notamment au niveau des logs et de la gestion des données) et à améliorer la robustesse des traitements.

Elle repose sur un fonctionnement asynchrone, nécessitant la mise en place d'un mécanisme de suivi (polling) basé sur un identifiant technique (tracking_id).

Une bonne compréhension de ce fonctionnement est essentielle pour garantir une intégration fiable.
Ce guide propose :

- des explications fonctionnelles,
- des exemples de requêtes (CURL, PHP, Java, Python, NodeJS, Postman),
- des recommandations techniques et bonnes pratiques,
- ainsi qu'une checklist de validation.

Les exemples fournis sont indicatifs et doivent être adaptés à l'environnement technique de chaque établissement.

2. Principe général

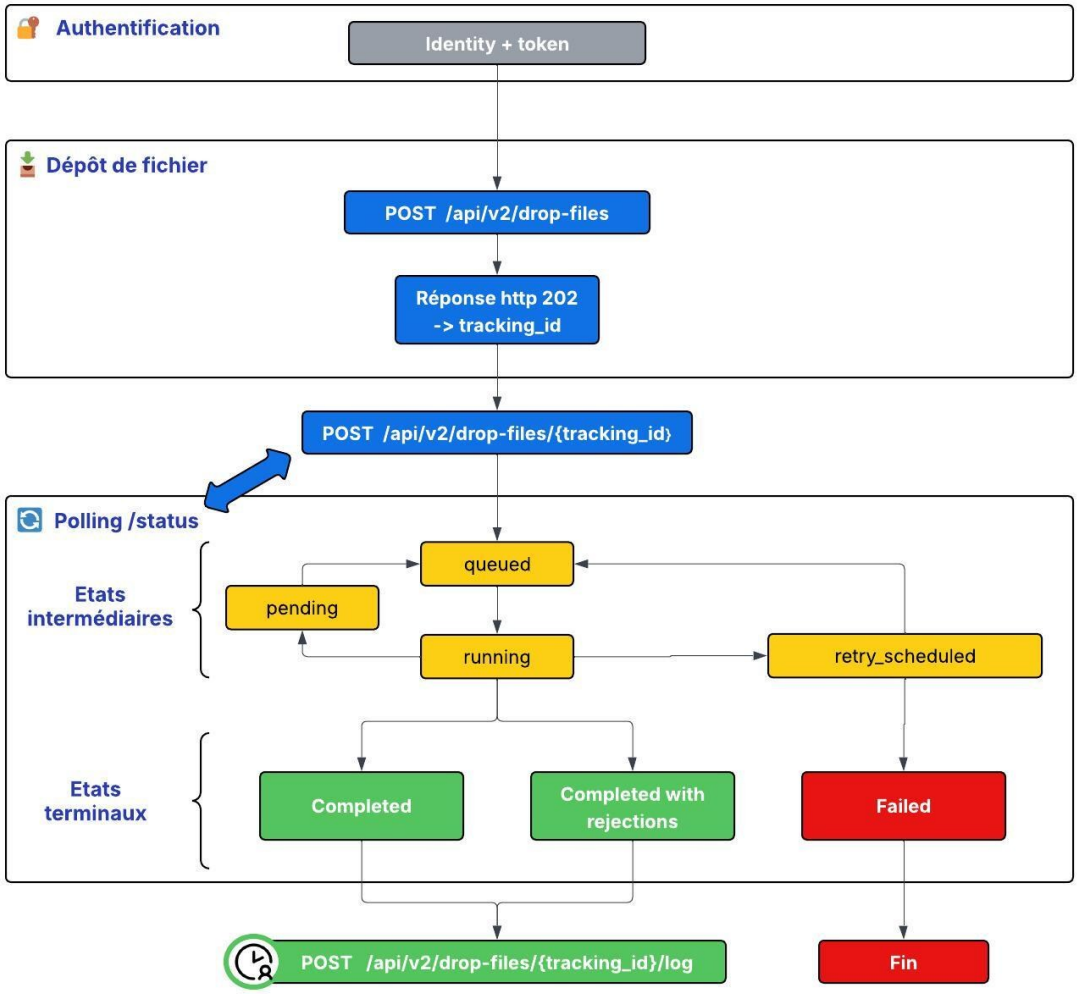
L'API e-Satis repose sur un fonctionnement **asynchrone** :
Cela signifie que le traitement des données n'est pas immédiat : une fois le dépôt effectué, il y a une réponse immédiate, mais pour savoir si les adresses mails contenues dans le fichier ont bien été importées, il faut suivre son avancement et télécharger le fichier log.

1. Envoi du fichier → POST /api/v2/drop-files
2. Réponse immédiate → tracking_id
3. Suivi du traitement → POST /api/v2/drop-files/{tracking_id}
4. Téléchargement du log → POST /api/v2/drop-files/{tracking_id}/log

⚠️ IMPORTANT
Un dépôt accepté (HTTP 202) signifie uniquement que la demande est prise en compte.
Le traitement métier est effectué de manière asynchrone et doit être suivi via le tracking_id.

3. Diagramme de fonctionnement

Le diagramme ci-dessous illustre les différentes étapes et états du traitement :



- **Bleu** → Appels API (entrée)
- **Jaune** → Phase de supervision (polling) : Traitement en cours
- **Rouge** → Erreur
- **Vert** → Succès + récupération des résultats

4. Présentation de l'intégrateur

Un client complet doit implémenter toutes les étapes suivantes :

Étape	Description	Objectif / Résultat attendu
Authentification	Le client est capable de s'authentifier avec succès auprès de l'API, en utilisant à la fois le token (Authorization Bearer) et les informations présentes dans le bloc identity (user_id, password, structure).	Garantir que les appels sont autorisés et correctement associés à une structure métier.
Dépôt	Le client envoie correctement un dépôt via l'endpoint /api/v2/drop-files, avec un payload conforme (identity + data + éventuellement out_filename).	Le dépôt est accepté par l'API et retourne un tracking_id.
tracking_id stocké	Le tracking_id retourné par l'API est immédiatement récupéré et stocké côté client (en mémoire ou en base).	Permettre le suivi du traitement, reprendre le traitement en cas d'interruption et éviter les doublons.
Polling de l'état	Le client interroge régulièrement l'API via l'endpoint /api/v2/drop-files/{tracking_id} pour suivre l'état du traitement.	Suivre l'évolution du traitement jusqu'à son état final (completed, failed, etc.).
Gestion des états	Le client interprète correctement les états retournés (queued, running, pending, completed, completed_with_rejections, failed, retry_scheduled).	Adapter le comportement (attente, succès, correction, alerte) en fonction de l'état
Téléchargement du log	Une fois le traitement terminé, le client appelle l'endpoint /log pour récupérer le journal d'import.	Permettre l'analyse des résultats du dépôt et des éventuelles lignes rejetées.
Gestion des erreurs (retry)	Le client implémente une stratégie de retry en cas d'échec technique (erreur réseau ou serveur), sans recréer de dépôt inutile.	Garantir la robustesse de l'intégration sans générer de doublons ou surcharge côté serveur

5. Authentification

Chaque requête doit contenir des informations permettant l'authentification :

5.1. Header

```
Authorization: Bearer <token>
Content-Type: application/json
```

5.2. Payload (Json)

```
"identity":
{
        "user_id": "xxx",
        "password": "xxx",
        "structure": "xxx"
}
```

6. Dépôt de fichier

6.1. Endpoint

```
POST /api/v2/drop-files
```

6.2. Exemple CURL

Ce code permet d'envoyer un dépôt à l'API avec les données à traiter :

```
curl -X POST https://<base-url>/api/v2/drop-files
\
-H "Authorization: Bearer TOKEN" \
-H "Content-Type: application/json" \
-d '{
"identity": {
"user_id": "563773",
"password": "123456",
"structure": "000000001"
},
"data": [
"000000001;patient@test.fr;1931;2;10/01/2026;17/01/2026;MCO48H"
],
"out_filename": "IMPORT_TEST"
}'
```

Le champ data contient les lignes à importer.

6.3. Réponse en Json

```
{
"success": true,
"tracking_id": "uuid",
"processing": {
"state": "queued"
}
}
```

Le tracking_id est essentiel : il permet de suivre le traitement.

7. Polling de l'état (IMPORTANT)

7.1. Endpoint

```
POST /api/v2/drop-files/{tracking_id}
```

7.2. Exemple CURL

Cet appel permet de vérifier l'état du traitement côté serveur :

```
curl -X POST https://<base-url>/api/v2/drop-files/{tracking_id} \
-H "Authorization: Bearer TOKEN" \
-H "Content-Type: application/json" \
-d '{
"identity": {...},
"tracking_id": "uuid"
}'
```

Le tracking_id doit être strictement identique dans l'URL et dans le payload.

7.3. États possibles

État	Description	Action recommandée
queued	Dépôt accepté, en attente de traitement.	Continuer le polling
running	Traitement en cours.	Continuer le polling
pending	Traitement temporairement suspendu	Continuer le polling jusqu'à reprise du traitement.
completed	Traitement terminé sans rejet.	Fin du processus
completed_with_rejections	Traitement terminé avec rejets.	Télécharger le log et corriger
failed	Échec technique global.	Analyser / contacter support
retry_scheduled	Nouvelle tentative planifiée automatiquement.	Attendre puis continuer polling

7.4. Exemple de réponse :

Cette réponse donne l'état actuel et éventuellement un résumé du traitement.

```
{
  "tracking_id": "uuid",
  "processing": {
    "state": "running"
  },
  "summary": {
    "imported_count": 120,
    "rejected_count": 3
  }
}
```

8. Stratégie de polling

8.1. Recommandation

- 0 → 60s : toutes les 5 secondes
- > 60s : toutes les 30 secondes
- Timeout recommandé : 15 minutes
- Timeout HTTP recommandé côté client : 30 secondes par requête.

8.2. Pseudo-code polling

Ce pseudo-code décrit la logique minimale à implémenter pour suivre un traitement.

```
while not finished:
status = call_status(tracking_id)

if status in [completed, completed_with_rejections]:
break

if status == failed:
log_error()
break

wait(delay)
```

Le polling doit continuer tant que l'état n'est pas terminal :
- completed
- completed_with_rejections
- failed

9. Récupération du log

9.1. Endpoint

```
POST /api/v2/drop-files/{tracking_id}/log
```

9.2. CURL

Cet appel permet de récupérer le fichier de log du traitement.

```
curl -X POST https://<base-url>/api/v2/drop-files/{tracking_id}/log \
-H "Authorization: Bearer TOKEN" \
-H "Accept: text/plain" \
-OJ \
-d '{ "identity": {...}, "tracking_id": "uuid" }'
```

Le log doit être récupéré uniquement lorsque le traitement est terminé :
- completed
- completed_with_rejections

Aucun log n'est disponible lorsque l'état est "failed".

10. Gestion des erreurs

Voici la liste des erreurs que vous pourrez rencontrer et l'action correspondante pour chaque code d'erreur :

HTTP	Code	Action
400	INVALID_PAYLOAD	Corriger payload
401	AUTHENTICATION_FAILED	Vérifier credentials
403	FORBIDDEN	Vérifier structure
404	NOT_FOUND	Mauvais tracking_id
409	DUPLICATE	Ne pas renvoyer
500	INTERNAL_ERROR	Retry

Un même dépôt ne doit jamais être renvoyé sans vérification préalable de son état via le tracking_id (status = failed).

11. Gestion des retry (client)

11.1. Recommandation

Retry uniquement si status = « failed » :

- erreur réseau
- code erreur 500

11.2. Pseudo-code retry

Ce code illustre une stratégie simple avec backoff.

```
max_retry = 3

for i in range(max_retry):
response = call_api()

if success:
break

wait(2^i seconds)
```

Le retry ne doit jamais entraîner la création d'un nouveau dépôt sans vérification préalable du l'état.

12. Exemple complet PHP

Ce code montre une implémentation complète : dépôt → polling → récupération du log.

```php
<?php

function callApi($url, $data) {
$ch = curl_init($url);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, [
"Content-Type: application/json"
]);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));

$response = curl_exec($ch);
curl_close($ch);

return json_decode($response, true);
}

// 1. dépôt
$deposit = callApi("https://<base-url>/api/v2/drop-files", [
"identity" => [
"user_id" => "563773",
"password" => "123456",
"structure" => "000000001"
],
"data" => ["ligne"]
]);

$trackingId = $deposit["tracking_id"];

// 2. polling
do {
sleep(5);

$status = callApi("https://<base-url>/api/v2/drop-files/$trackingId", [
"identity" => [...],
"tracking_id" => $trackingId
]);

} while(in_array($status["processing"]["state"], ["queued","running"]));

// 3. récupération log
if ($status["processing"]["state"] === "completed") {

$ch = curl_init("https://<base-url>/api/v2/drop-files/$trackingId/log");

curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode([
"identity" => [...],
"tracking_id" => $trackingId
]));

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$log = curl_exec($ch);
file_put_contents("result.log", $log);
}
```

Exemple de code généré avec l'assistance d'une intelligence artificielle, fourni à titre indicatif. À adapter selon votre environnement.

13. Exemple Java (HttpClient complet)

Cet exemple montre comment envoyer un dépôt avec le client HTTP natif de Java

```
HttpClient client = HttpClient.newHttpClient();

String body = """
{
"identity": {
"user_id": "563773",
"password": "123456",
"structure": "000000001"
},
"data": ["ligne"]
}
""";

HttpRequest request = HttpRequest.newBuilder()
.uri(URI.create("https://<base-url>/api/v2/drop-files"))
.header("Content-Type", "application/json")
.POST(HttpRequest.BodyPublishers.ofString(body))
.build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
```

Exemple de code généré avec l'assistance d'une intelligence artificielle, fourni à titre indicatif. À adapter selon votre environnement.

14. Exemples complets – Python / NodeJS

Les exemples ci-dessous montre une implémentation complète : dépôt → polling → récupération du log avec différents langages.

14.1. Exemple Python complet (requests)

```
import requests
import time

BASE_URL = "https://<base-url>"
TOKEN = "YOUR_TOKEN"

identity = {
"user_id": "563773",
"password": "123456",
"structure": "000000001"
}

headers = {
"Authorization": f"Bearer {TOKEN}",
"Content-Type": "application/json"
}

# 1. DEPOT
deposit_payload = {
"identity": identity,
"data": [
"000000001;patient@test.fr;1931;2;10/01/2026;17/01/2026;MCO48H"
],
"out_filename": "IMPORT_TEST"
}

response = requests.post(f"{BASE_URL}/api/v2/drop-files", json=deposit_payload, headers=headers)
data = response.json()

tracking_id = data["tracking_id"]
print("Tracking ID:", tracking_id)

# 2. POLLING
state = "queued"

while state in ["queued", "running",” pending“]:
time.sleep(5)

status_payload = {
"identity": identity,
"tracking_id": tracking_id
}

response = requests.post(f"{BASE_URL}/api/v2/drop-files/{tracking_id}", json=status_payload,
headers=headers)
data = response.json()

state = data["processing"]["state"]
print("Status:", state)

# 3. DOWNLOAD LOG
if state in ["completed", "completed_with_rejections"]:
log_payload = {
"identity": identity,
"tracking_id": tracking_id
}

response = requests.post(f"{BASE_URL}/api/v2/drop-files/{tracking_id}/log",
json=log_payload,
headers=headers)

with open("result.log", "wb") as f:
f.write(response.content)

print("Log téléchargé")
```

Exemple de code généré avec l'assistance d'une intelligence artificielle, fourni à titre indicatif. À adapter selon votre environnement.

14.2. Exemple NodeJS (axios)

```
const axios = require("axios");

const BASE_URL = "https://<base-url>";
const TOKEN = "YOUR_TOKEN";

const identity = {
user_id: "563773",
password: "123456",
structure: "000000001"
};

const client = axios.create({
baseURL: BASE_URL,
headers: {
"Authorization": `Bearer ${TOKEN}`,
"Content-Type": "application/json"
}
});

async function main() {

// 1. DEPOT
const deposit = await client.post("/api/v2/drop-files", {
identity,
data: ["000000001;patient@test.fr;1931;2;10/01/2026;17/01/2026;MCO48H"],
out_filename: "IMPORT_TEST"
});

const trackingId = deposit.data.tracking_id;
console.log("Tracking:", trackingId);

// 2. POLLING
let state = "queued";

while (state === "queued" || state === "running") {
await new Promise(r => setTimeout(r, 5000));

const status = await client.post(`/api/v2/drop-files/${trackingId}`, {
identity,
tracking_id: trackingId
});

state = status.data.processing.state;
console.log("Status:", state);
}

// 3. DOWNLOAD LOG
if (["completed", "completed_with_rejections"].includes(state)) {

const log = await client.post(
`/api/v2/drop-files/${trackingId}/log`,
{
identity,
tracking_id: trackingId
},
{ responseType: "arraybuffer" }
);

const fs = require("fs");
fs.writeFileSync("result.log", log.data);

console.log("Log téléchargé");
}
}

main();
```

Exemple de code fourni à titre indicatif, généré avec l'assistance d'une intelligence artificielle. À adapter selon votre environnement.

15. Collection Postman

Postman est un outil permettant de tester et d'exécuter des appels à l'API sans écrire de code. Il est particulièrement utile pour vérifier les échanges HTTP, valider les paramètres et comprendre le fonctionnement de l'API avant d'implémenter une intégration automatisée.

15.1. Variables environnement

Ces variables permettent de centraliser les paramètres utilisés dans les différentes requêtes (URL, identifiants, tracking_id). Elles évitent de dupliquer les informations et facilitent la maintenance.

```
{
"baseUrl": "https://<base-url>",
"user_id": "563773",
"password": "123456",
"structure": "000000001",
"tracking_id": ""
}
```

La variable tracking_id sera automatiquement renseignée après le dépôt (voir script ci-dessous).

15.2. Requête 1 – Dépôt

Cette requête permet d'envoyer un fichier à traiter à l'API. Elle correspond à la première étape du processus.

POST

```
{{baseUrl}}/api/v2/drop-files
```

Body

```
{
"identity": {
"user_id": "{{user_id}}",
"password": "{{password}}",
"structure": "{{structure}}"
},
"data": [
"000000001;patient@test.fr;1931;2;10/01/2026;17/01/2026;MCO48H"
],
"out_filename": "IMPORT_TEST" // nom de fichier de sortie -- facultatif
}
```

15.3. Script Postman (très important)

Ce script est exécuté automatiquement après la requête. Il permet de récupérer le tracking_id retourné par l'API et de le stocker dans les variables d'environnement.

```
const response = pm.response.json();

if (response.tracking_id) {
pm.environment.set("tracking_id", response.tracking_id);
}
```

Vérifier que le header Authorization: Bearer <token> est bien présent sur chaque requête.

15.4. Requête 2 – Etat

Cette requête permet de suivre l'avancement du traitement en interrogeant l'API avec le tracking_id.

POST

```
{{baseUrl}}/api/v2/drop-files/{{tracking_id}}
```

Body

```
{
"identity": {
"user_id": "{{user_id}}",
"password": "{{password}}",
"structure": "{{structure}}"
},
"tracking_id": "{{tracking_id}}"
}
```

15.5. Requête 3 – Log

Cette requête permet de récupérer le log de traitement une fois celui-ci terminé.

POST

```
{{baseUrl}}/api/v2/drop-files/{{tracking_id}}/log
```

Body

```
{
"identity": {
"user_id": "{{user_id}}",
"password": "{{password}}",
"structure": "{{structure}}"
},
"tracking_id": "{{tracking_id}}"
}
```

Pour télécharger automatiquement le fichier retourné par l'API, cocher :

```
Save response → file
```

16. Bonnes pratiques clés

Les bonnes pratiques suivantes permettent de sécuriser l'intégration et d'éviter les erreurs fréquentes.

- Toujours conserver le tracking_id
- Ne jamais relancer un dépôt immédiatement
- Gérer les états completed_with_rejections
- Télécharger le log généré en cas de succès à la fin du traitement
- Ne jamais stocker ni tracer les mots de passe dans les logs
- Le statut "pending" ne nécessite aucune action côté client autre que la poursuite du polling.

17. Conclusion

Avec cette documentation, vous disposez de tous les éléments nécessaires pour implémenter une intégration complète avec l'API e-Satis. Elle permet notamment de :

- Développer un client complet conforme aux attentes de l'API
- Suivre le traitement des dépôts (polling)
- Gérer les erreurs et les rejets
- Sécuriser son intégration
- Éviter 90% des problèmes support

Une intégration respectant ces principes garantit un fonctionnement fiable, robuste et conforme aux bonnes pratiques.