

Mode REST standalone — Architecture et configuration

Les widgets EMILE fonctionnent normalement dans un **iframe Grist** (mode plugin). Le mode REST permet d'y accéder depuis une **URL publique**, sans iframe — pour partage externe (email, SMS) ou formulaires autonomes.

Widgets accessibles en mode REST

Widget	URL directe	Usage
fiche-candidat	/widgets/emile/fiche-candidat?rowId=42	Consultation/édition d'un dossier candidat via magic link
ajout-etablissement	/widgets/emile/ajout-etablissement	Formulaire d'ajout d'un établissement
creation-compte-orienteur	/widgets/emile/creation-compte-orienteur	Formulaire de création d'un compte orienteur

Architecture

```
Navigateur (GitHub Pages, CORS bloqué vers Grist)
|
| GET ?table=CANDIDATS&filter={"id":[42]}
▼
https://n8n.incubateur.../webhook/grist ← proxy
|
| Bearer <clé API> (jamais exposée au navigateur)
| GET https://grist.incubateur.dnum.din.developpement-durable.gouv.fr
|     /api/docs/75GHATRaKvHSmx3FRqCi4f/tables/CANDIDATS/records?filter=...
▼
Grist (instance interne)
| {"records": [{id: 42, fields: {...}}]}
▼
n8n → réponse avec Access-Control-Allow-Origin: *
▼
Widget → affiche / enregistre les données
```

Pourquoi un proxy ?

CORS — Les widgets tournent sur `stiiig.github.io`. Le navigateur bloque par sécurité tout appel JS vers un autre domaine (`grist.incubateur.dnum...`) sauf si ce domaine répond avec `Access-Control-Allow-Origin: *`. Grist ne le fait pas. n8n appelle Grist côté serveur (pas de CORS), et renvoie la réponse avec le bon header.

Clé API — Grist demande un Bearer token pour toute requête. Si on appelait Grist directement depuis le browser, la clé serait visible dans le JS. Avec n8n, elle reste côté serveur, jamais exposée.

Activation du mode REST

Contrôlé par la variable d'environnement `NEXT_PUBLIC_GRIST_PROXY_URL` (baked dans le bundle au build Next.js).

Variable	Valeur
<code>NEXT_PUBLIC_GRIST_PROXY_URL</code>	<code>https://n8n.incubateur.dnum.din.developpement-durable.gouv.fr/webhook/grist</code>

Déclarée dans `.github/workflows/deploy.yml` :

```
env:  
  NEXT_PUBLIC_GRIST_PROXY_URL: ${ secrets.NEXT_PUBLIC_GRIST_PROXY_URL }
```

Quand cette variable est définie, `src/lib/grist/init.ts` bascule automatiquement en mode `rest` et utilise `createRestDocApi()` au lieu du plugin Grist.

Fichiers clés

`src/lib/grist/rest.ts`

Implémente `GristDocAPI` via le proxy n8n :

Méthode	Requête vers n8n	Traduite par n8n en	Workflow
<code>fetchTable(tableId)</code>	<code>GET ?table=TABLE</code>	<code>GET /tables/TABLE/records</code>	GET
<code>fetchSingleRowRest(tableId, rowId)</code>	<code>GET ?table=TABLE&filter={"id":[rowId]}</code>	<code>GET /tables/TABLE/records?filter=...</code>	GET
<code>getAttachmentDownloadUrl(id)</code>	<code>GET ?attachId=ID</code>	<code>GET /attachments/ID/download</code>	GET
<code>uploadAttachments(files)</code>	<code>POST multipart/form-data (champ upload)</code>	<code>POST /attachments</code>	POST
<code>applyUserActions(["AddRecord", ...])</code>	<code>POST ?table=TABLE + JSON</code>	<code>POST /tables/TABLE/records</code>	POST*
<code>applyUserActions(["UpdateRecord", ...])</code>	<code>PATCH ?table=TABLE + JSON</code>	<code>PATCH /tables/TABLE/records</code>	POST*

** Les actions `AddRecord` / `UpdateRecord` envoient du `Content-Type: application/json`, ce qui déclenche un preflight `OPTIONS`. Le workflow `POST` devra gérer ce cas si ces actions sont utilisées en mode `REST` (voir *Limitations*).*

`src/lib/grist/meta.ts` — `loadColumnsMetaFor`

Charge les métadonnées des colonnes (types, options, choices) via les tables internes Grist :

- `fetchTable("_grist_Tables")` → trouve le rowId de la table cible
- `fetchTable("_grist_Tables_column")` → récupère les colonnes de cette table

Ces deux tables sont accessibles via le proxy n8n comme n'importe quelle table normale.

src/lib/grist/hooks.ts

- `useGristInit` : charge `grist-plugin-api.js` uniquement si on est dans un iframe. En mode standalone, ce script n'est jamais chargé.
- `useDepartementOptions` : charge `DPTS_REGIONS` via `fetchTable`, retourne `{deptOptions, dptsLoading, dptsError}`.

Configuration n8n

Deux workflows séparés

Le widget utilise **deux workflows n8n distincts** au même path `grist` :

- **Workflow GET** — gère les lectures (records, métadonnées, téléchargement de pièces jointes)
- **Workflow POST** — gère l'upload de pièces jointes

***Pourquoi deux workflows ?** La version n8n installée ne propose pas "Any Method" sur les webhooks. L'upload `multipart/form-data` sans header custom est une [« simple CORS request »](#) : le navigateur ne fait pas de preflight OPTIONS. On peut donc utiliser un webhook POST dédié sans avoir à gérer OPTIONS.*

Workflow GET — records et téléchargement

URL (production) :

```
https://n8n.incubateur.dnum.din.developpement-durable.gouv.fr/webhook/grist
```

Nœud 1 — Webhook (GET)

Paramètre	Valeur
HTTP Method	GET
Path	<code>grist</code>
Response Mode	Using Respond to Webhook Node

Nœud 2 — IF Attachment download

Paramètre	Valeur
Condition	<code>{{ \$json.query.attachId }}</code> is not empty
True →	branche téléchargement pièce jointe
False →	branche records

⚠ Piège : n8n lève "Wrong type: '9' is a string but was expecting a boolean". Activer l'option **"Convert types where required"** dans le nœud IF.

Branche download (True)

Nœud 3a — HTTP Request (fichier binaire)

Paramètre	Valeur
Method	GET
URL	https://grist.incubateur.dnum.din.developpement-durable.gouv.fr/api/docs/75GHATRaKvHSmx3FRqCi4f/attachments/{{ \$json.query.attachId }}/download
Authentication	Generic Credential Type → Bearer Auth → Bearer Auth Grist
Response Format	File

⚠ **Piège** : sans `Response Format: File`, n8n tente de parser la réponse comme JSON et échoue.

Nœud 4a — Respond to Webhook (binaire)

Paramètre	Valeur
Respond With	Binary Data
Response Data Property Name	data
Response Code	200
Send Headers	Using Fields Below
Header	Access-Control-Allow-Origin: *

ℹ n8n forward automatiquement les headers `Content-Type` et `Content-Disposition` de la réponse Grist.

⚠ **Piège navigateur** : `window.open(url)` affiche les PDFs au lieu de les télécharger. Le code utilise `fetch + blob + <a download>` pour forcer le téléchargement avec le bon nom de fichier.

Branche records (False)

Nœud 3b — HTTP Request (appel Grist — records)

Paramètre	Valeur
Method	GET
Authentication	Generic Credential Type → Bearer Auth → Bearer Auth Grist
Query Parameters	Vide

URL (sans activer `fx`) :

```
https://grist.incubateur.dnum.din.developpement-durable.gouv.fr/api/docs/75GHATRaKvHSmx3FRqCi4f/tables/{{ $json.query.table }}/records{{ $json.query.filter ? '?filter=' + $json.query.filter : '' }}
```

⚠ Ne jamais ajouter de param `filter` dans "Using Fields Below" — utiliser uniquement l'expression dans l'URL.

Cas couverts :

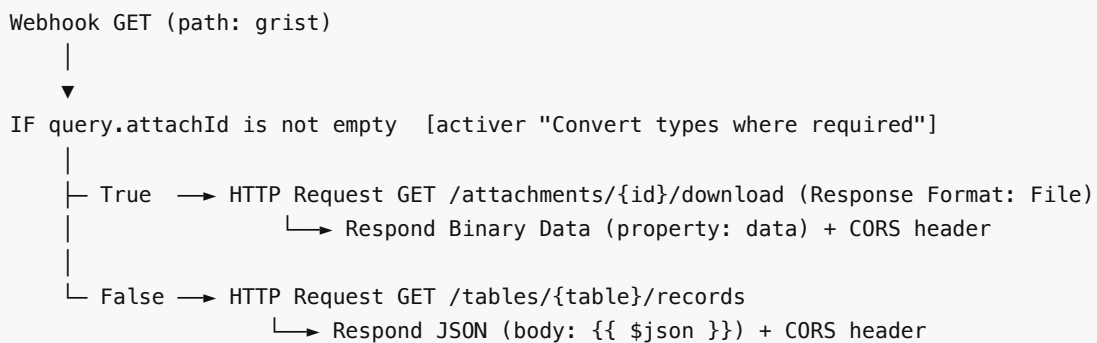
Requête entrante	URL envoyée à Grist
?table=ETABLISSEMENTS	.../tables/ETABLISSEMENTS/records
?table=CANDIDATS&filter={"id": [42]}	.../tables/CANDIDATS/records?filter=...
?table=_grist_Tables	.../tables/_grist_Tables/records

Nœud 4b — Respond to Webhook (JSON)

Paramètre	Valeur
Respond With	JSON
Response Code	200
Response Body	{{ \$json }}
Send Headers	Using Fields Below
Header	Access-Control-Allow-Origin: *

⚠ **Piège critique** : écrire `{{ $json }}` et **non** `={{ $json }}`. Le `=` est la syntaxe d'expression n8n dans les champs texte — dans le champ Response Body du Respond to Webhook, il ne faut pas le préfixe `=`.

Schéma — Workflow GET




Workflow POST — upload de pièces jointes

URL (production) :

`https://n8n.incubateur.dnum.din.developpement-durable.gouv.fr/webhook/grist`

(même path que le workflow GET — n8n route par méthode HTTP)


 Le widget envoie un `POST multipart/form-data` sans header custom → pas de preflight CORS OPTIONS. Le webhook POST n'a donc pas besoin de gérer OPTIONS.


Nœud 1 — Webhook (POST)

Paramètre	Valeur
HTTP Method	POST
Path	grist
Response Mode	Using Respond to Webhook Node

Nœud 2 — HTTP Request (upload vers Grist)

Paramètre	Valeur
Method	POST
URL	https://grist.incubateur.dnum.din.developpement-durable.gouv.fr/api/docs/75GHATRaKvHSmx3FRqCi4f/attachments
Authentication	Generic Credential Type → Bearer Auth → Bearer Auth Grist
Body Content Type	Form-Data
Body / Name	upload
Body / Type	n8n Binary File
Body / Input Data Field Name	upload

 **Piège** : le champ **Name** doit être `upload` (pas `data`). C'est le nom du champ multipart envoyé à Grist. Le champ **Input Data Field Name** est le nom de la propriété binaire dans l'item n8n entrant (vérifiable dans l'onglet **Binary** du Webhook en mode test — il s'appelle `upload` car c'est le nom du champ `fd.append("upload", ...)` envoyé par le browser).

 **Piège** : sans l'**Authentication** configurée sur ce nœud, Grist renvoie une page HTML 403 — n8n lève "Invalid JSON in response body".

Nœud 3 — Respond to Webhook (IDs des pièces jointes)

Paramètre	Valeur
Respond With	JSON
Response Body	{{ \$json }}
Response Code	200
Send Headers	Using Fields Below
Header	Access-Control-Allow-Origin: *


 Grist renvoie `[42]` (tableau d'entiers). `n8n` le stocke en string dans `$json.data = "[42]"`. Le code `rest.ts` gère cette sérialisation et tous les autres formats possibles selon la version `n8n`.

Schéma — Workflow POST

```
Webhook POST (path: grist)
|
▼
HTTP Request POST /attachments
  Body: Form-Data, Name=upload, Type=n8n Binary File, Input Data Field Name=upload
  Auth: Bearer Auth Grist
  |
  ▼
Respond to Webhook
  body: {{ $json }} ← PAS ={{ $json }}
  header: Access-Control-Allow-Origin: *
```

Pièges rencontrés — référence complète

Récapitulatif de tous les problèmes rencontrés lors de la mise en place.

n8n — Webhook

Problème	Cause	Solution
Pas de "Any Method" dans le dropdown	Version <code>n8n</code> limitée	Deux workflows séparés GET et POST
Condition IF lève "Wrong type: '9' is a string but was expecting a boolean"	Le query param est une string, le comparateur attend un booléen	Activer " Convert types where required " dans le nœud IF

n8n — HTTP Request

Problème	Cause	Solution
"Invalid JSON in response body" sur l'upload	Authentication manquante → Grist renvoie du HTML 403	Configurer Bearer Auth Grist sur ce nœud spécifiquement
"Invalid JSON in response body" sur le download	Response Format = JSON alors que Grist renvoie du binaire	Passer Response Format à File
Champ Name = data au lieu de upload	Confusion entre le nom du champ Grist (upload) et le nom de la propriété <code>n8n</code>	Name = upload, Input Data Field Name = upload

n8n — Respond to Webhook

Problème	Cause	Solution
----------	-------	----------

Réponse vide ou non-JSON reçue par le browser	={{ \$json }} (syntaxe expression) au lieu de {{ \$json }}	Écrire {{ \$json }} sans le = préfixe
Le fichier s'ouvre dans un onglet au lieu de se télécharger	n8n ne renvoie pas Content-Disposition: attachment systématiquement	Le code utilise fetch + blob + <a download> côté browser

n8n — Format de réponse imprévisible

Grist renvoie [42] (tableau JSON). n8n transforme cette réponse de manière non déterministe selon sa version et sa config :

Format reçu par le browser	Quand
{"data": "[42]"}	HTTP Request sans Code node — n8n stocke le body brut en string dans .data
{"ids": [{"json": 42, "pairedItem": {...}]}	Avec Code node return [{ json: { ids: items } }] — items contient les objets n8n complets, pas juste les valeurs
[42]	Dans certaines versions de n8n

Le code `uploadAttachmentsRest` dans `rest.ts` gère tous ces formats avec un parsing exhaustif — **ne pas simplifier**.

Code — React

Problème	Cause	Solution
"Application error: a client-side exception" sur les pages avec attachments	return null conditionnel placé avant un useCallback → violation des Rules of Hooks	Déplacer tous les return conditionnels après tous les hooks

Table IDs Grist

⚠️ Le **Table ID** Grist (utilisé dans l'API et dans le code) est différent du **nom d'affichage** visible dans l'onglet. En cas de renommage d'une table, le Table ID ne change pas automatiquement.

Pour vérifier ou modifier le Table ID d'une table : clic droit sur l'onglet → **Table Settings** → champ **Table ID**.

Tables utilisées par les widgets EMILE :

Table ID (API)	Utilisée par
CANDIDATS	fiche-candidat, inscription-candidat
ETABLISSEMENTS	ajout-etablissement, creation-compte-orienteur
ACCOMPAGNANTS	creation-compte-orienteur
DPTS_REGIONS	ajout-etablissement
_grist_Tables	loadColumnsMetaFor (méta interne)

`_grist_Tables_column`

`loadColumnsMetaFor` (méta interne)

Mettre à jour la clé API Grist

1. Aller sur Grist → Profile Settings → API Key → copier la clé
2. Dans n8n : **Credentials** → **Bearer Auth Grist** → coller la nouvelle clé
3. Aucun redéploiement nécessaire

Générer un magic link (fiche candidat)

```
https://stiiig.github.io/grist-widgets/widgets/emile/fiche-candidat?rowId=<ID_GRIST>
```

`ID_GRIST` = rowId de l'enregistrement dans la table `CANDIDATS` (colonne `id` dans Grist).

Limitations connues

AddRecord / UpdateRecord — preflight CORS bloquant

Les actions `AddRecord` et `UpdateRecord` (sauvegarde de fiche, soumission de formulaire) envoient un `Content-Type: application/json`, ce qui déclenche un **preflight CORS OPTIONS**. Le workflow POST actuel ne gère que le `multipart/form-data` de l'upload.

Pour débloquer ces actions en mode REST, il faudra soit :

- Ajouter un webhook **PATCH** dédié et répondre aux OPTIONS avec les bons headers CORS
- Ou exposer un endpoint côté serveur (Next.js API route) qui fait le proxy sans contrainte CORS

Jusqu'à ce que ce soit configuré, les sauvegardes en mode REST échoueront.

Pièces jointes (AttachmentField)

Support en mode REST selon configuration n8n :

Fonctionnalité	Disponible	Condition
Affichage des noms	✓	Automatique (<code>_grist_Attachments</code> proxifié via workflow GET)
Téléchargement	✓	Branche <code>?attachId=X</code> dans le workflow GET configurée
Upload	✓	Workflow POST configuré
Suppression	✓	Upload opérationnel (ré-enregistre la liste sans l'ID supprimé)

Fiche candidat — onglets non mappés

Seul l'onglet "Administratif" de `fiche-candidat` est mappé sur des colonnes Grist. Les autres onglets affichent "non mappé" hors iframe.

Sécurité du magic link

Le magic link est **public** — toute personne ayant l'URL peut consulter le dossier. À sécuriser si nécessaire (token signé, expiration, authentification).