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?token=ID.HMAC`	Consultation/édition d'un dossier candidat via magic link signé
`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&token=ID.HMAC
    v
https://n8n.incubateur.../webhook/grist   ← proxy
    |  vérifie HMAC-SHA256 du token
    |  Bearer <clé API>  (jamais exposée au navigateur)
    |  GET https://grist.incubateur.dnum.din.developpement-durable.gouv.fr
    |       /api/docs/75GHATRaKvHSmx3FRqCi4f/tables/CANDIDATS/records?filter={"id":[rowId]}
    v
Grist (instance interne)
    |  {"records": [{id: 42, fields: {...}}]}
    v
n8n -> réponse avec Access-Control-Allow-Origin: *
    v
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.

Vérification du magic link — Le secret HMAC ne doit jamais apparaître dans le bundle JS. La vérification se fait entièrement dans n8n.

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
NEXT_PUBLIC_GRIST_PROXY_URL	https://n8n.incubateur.dnum.din.developpement-durabl

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
fetchTable(tableId)	GET ?table=TABLE	GET /tables/TABLE/records	GET
fetchSingleRowRest(tableId, rowId, token)	GET ?table=TABLE&token=ID.HMAC	Vérif HMAC -> GET /tables/TABLE/records?filter={"id":[rowId]}	GET
fetchSingleRowRest(tableId, rowId)	GET ?table=TABLE&filter={"id":[rowId]}	GET /tables/TABLE/records?filter=... (dev fallback, sans vérif)	GET
getAttachmentDownloadUrl(id)	GET ?attachmentId=ID	GET /attachments/ID/download	GET
uploadAttachments(files)	POST multipart/form-data (champ upload)	POST /attachments	POST
applyUserActions([['AddRecord', tableId, ...]])	POST ?table=TABLE + JSON	POST /tables/TABLE/records	POST*
applyUserActions([['UpdateRecord', tableId, ...]])	PATCH ?table=TABLE + JSON	PATCH /tables/TABLE/records	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 : 1. fetchTable("_grist_Tables") -> trouve le rowId de la table cible 2. 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 (pas de token nécessaire).

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

Trois workflows

Workflow	Méthode	Path	Usage
GET	GET	`/webhook/grist`	Lecture records, métadonnées, téléchargement PJ, vérification magic link
POST	POST	`/webhook/grist`	Upload pièces jointes
GENERATE	POST	`/webhook/grist-generate`	Génération de magic links signés (Basic Auth)

Workflow GET — magic link + records + téléchargement

URL (production) :

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

Nœud 1 — Webhook (GET)

Paramètre	Valeur
HTTP Method	**GET**
Path	`grist`
Response Mode	Using Respond to Webhook Node

Nœud 2 — IF token présent (magic link)

Paramètre	Valeur
Condition	`{{ $json.query.token }}` **is not empty**
True ->	branche vérification HMAC
False ->	**IF attachId** (chemin classique, pas de 403 !)

> [!] **Piège critique** : la branche **faux** doit aller vers **IF attachId**, jamais vers un Respond 403. Les requêtes de métadonnées (`_grist_Tables`, `_grist_Tables_column`, etc.) n'ont pas de token — les rejeter avec 403 empêche le widget de charger les colonnes et les dropdowns.

Branche token (True)

Nœud 3a — Code (extraction rowId + sig)

```
const token = $json.query.token;        // ex: "42.a3f9b2..."
const parts = token.split(".");
const rowId = parts[0];
const sig   = parts[1];
if (!rowId || !sig || isNaN(Number(rowId))) throw new Error("Token malformé");
return [{ json: { rowId: Number(rowId), sig, token } }];
```

Nœud 4a — Crypto (HMAC-SHA256)

Paramètre	Valeur
Action	**HMAC**
Type	**SHA256**
Value	`{{ $json.rowId.toString() }}`
Credential	**EMILE HMAC Secret** (credential Crypto n8n)
Encoding	**HEX**
Property Name	`data`

Nœud 5a — IF signature valide

Paramètre	Valeur
Value 1	`{{ $json.data }}` (HMAC calculé)
Operation	**equals**
Value 2	`{{ $json.sig }}` (sig extrait du token)
True ->	IF attachId (lecture normale)
False ->	Respond to Webhook 403 `{ "error": "Token invalide" }`

Branche classique (False du IF token) + branche token valide

Les deux chemins convergent vers :

Nœud — IF attachId

Paramètre	Valeur
Condition	`{{ $json.query.attachId }}` **is not empty**
True ->	branche téléchargement pièce jointe
False ->	branche records

[!] Activer **"Convert types where required"** dans ce nœud IF.

Branche download (True)

HTTP Request (fichier binaire)

Paramètre	Valeur
Method	GET
URL	`https://grist.incubateur.dnum.din.developpement-du` `$json.query.attachId }}/download`
Authentication	Generic Credential Type -> Bearer Auth -> **Bearer Auth Grist**
Response Format	**File**

[!] Sans `Response Format: File`, n8n tente de parser la réponse comme JSON et échoue.

Respond to Webhook (binaire)

Paramètre	Valeur
Respond With	**Binary Data**
Response Data Property Name	`data`
Response Code	200
Header	`Access-Control-Allow-Origin: *`

Branche records (False)

HTTP Request (appel Grist — records)

Paramètre	Valeur
Method	GET
Authentication	Generic Credential Type -> Bearer Auth -> **Bearer Auth Grist**

URL (expression, sans activer `fx` séparément) :

`https://grist.incubateur.dnum.din.developpement-durable.gouv.fr/api/docs/75GHATRaKvHSmx3FRqCi4f/tables/{{ `

Cette expression gère les deux cas : - **Token path** (`$json.rowId` défini) -> table = `CANDIDATS`, filter = `{"id":[rowId]}` - **Chemin classique** (`$json.query` intact) -> table et filter viennent de la query

> [!] 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`
`?table=CANDIDATS&token=42.abc...`	`.../tables/CANDIDATS/records?filter={"id":[42]}`

Respond to Webhook (JSON)

Paramètre	Valeur
Respond With	JSON
Response Code	200
Response Body	`{{ $json }}`
Header	`Access-Control-Allow-Origin: *`

> [!] Écrire `{{ $json }}` et **non** `={{ $json }}`.

Schéma — Workflow GET

```
Webhook GET (path: grist)
    |
    v
IF query.token is not empty
    |
    +- True --> Code (extrait rowId + sig)
```

```
|           +--> Crypto HMAC-SHA256 (credential EMILE HMAC Secret)
|                +--> IF data === sig
|                      +- True  --> IF attachId (voir ci-dessous)
|                      +- False --> Respond 403 { "error": "Token invalide" }
|
+- False --> IF query.attachId is not empty  [Convert types: ON]
              +- True  --> HTTP GET /attachments/{id}/download (Format: File)
              |            +--> Respond Binary Data + CORS header
              +- False --> HTTP GET /tables/{table}/records (URL expression)
                           +--> Respond JSON (body: {{ $json }}) + CORS header
```

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)

[i] 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-du`
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`

[!] Le champ **Name** doit être `upload` (pas `data`). C'est le nom du champ multipart envoyé à Grist.

[!] 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

Paramètre	Valeur
Respond With	JSON
Response Body	`{{ $json }}`
Response Code	200
Header	`Access-Control-Allow-Origin: *`

Workflow GENERATE — génération de magic links

URL (production) :

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

Protégé par **Basic Auth** — à appeler depuis une automation Grist ou manuellement.

Nœud 1 — Webhook (POST, Basic Auth)

Paramètre	Valeur
HTTP Method	**POST**
Path	`grist-generate`
Authentication	Basic Auth -> credential **Emile generate token**
Respond	**Using Respond to Webhook Node**

Nœud 2 — Code (extraire rowId)

```
const rowId = $json.body.rowId;
if (!rowId || isNaN(Number(rowId))) throw new Error("rowId requis");
return [{ json: { rowId: Number(rowId), rowIdStr: rowId.toString() } }];
```

Nœud 3 — Crypto (HMAC-SHA256)

Paramètre	Valeur
Action	**HMAC**
Type	**SHA256**
Value	`{{ $json.rowIdStr }}`
Credential	**EMILE HMAC Secret** (même credential que le workflow GET !)
Encoding	**HEX**
Property Name	`data`

Nœud 4 — Code (construire token + URL)

```
const rowId = $json.rowId;
const sig   = $json.data;
const token = `${rowId}.${sig}`;
const url   = `https://stiiig.github.io/grist-widgets/widgets/emile/fiche-candidat?token=${token}`;
return [{ json: { rowId, token, url } }];
```

Nœud 5 — Respond to Webhook

Paramètre	Valeur
Respond With	**First Incoming Item**

Retourne `{ rowId, token, url }`.

Exemple d'appel

```
curl -X POST "https://n8n.incubateur.dnum.din.developpement-durable.gouv.fr/webhook/grist-generate" \
  -u "user:password" \
  -H "Content-Type: application/json" \
  -d '{"rowId": 42}'
# -> { "rowId": 42, "token": "42.a3f9b2...", "url": "https://stiiig.github.io/...?token=42.a3f9b2..." }
```

Pièges rencontrés — référence complète

n8n — Webhook

Problème	Cause	Solution
Pas de "Any Method" dans le dropdown	Version n8n limitée	Trois workflows séparés GET, POST, GENERATE
Condition IF lève "Wrong type: '9' is a string but was expecting a boolean"	Le query param est une string	Activer **"Convert types where required"** dans le nœud IF
`{"code":0,"message":"Unused Respond to Webhook node found in the workflow"}`	Le nœud Respond to Webhook n'est pas câblé	Le connecter au dernier nœud, ou passer Respond à "When Last Node Finishes"

n8n — IF et routing

Problème	Cause	Solution
Toutes les requêtes de métadonnées retournent 403	Branche faux du IF token connectée au Respond 403	La brancher sur **IF attachId** — les métadonnées n'ont jamais de token
Token invalide alors que le secret est correct	Les deux Crypto nodes utilisent des credentials différents	Vérifier que GENERATE et GET utilisent **le même credential**
`sig` tronqué dans n8n (ex: `887512` au lieu de 64 chars)	URL du magic link copiée/collée tronquée	Copier l'URL complète depuis la réponse JSON du workflow GENERATE

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 et la propriété n8n	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 (upload)

Grist renvoie `[42]` (tableau JSON). n8n transforme cette réponse de manière non déterministe :

Format reçu par le browser	Quand
`{"data": "[42]"}`	HTTP Request sans Code node
`{"ids": [{"json": 42, "pairedItem": {...}}]}`	Avec Code node retournant les items n8n complets
`[42]`	Certaines versions de n8n
`42`	n8n unwrappe le tableau d'un seul élément

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

Code — React

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

---

Générer un magic link (fiche candidat)

```bash
# Générer un token signé pour le rowId 42
curl -X POST "https://n8n.incubateur.dnum.din.developpement-durable.gouv.fr/webhook/grist-generate" \
  -u "user:password" \
  -H "Content-Type: application/json" \
  -d '{"rowId": 42}'
```

Réponse :

```json
{
  "rowId": 42,
  "token": "42.a3f9b2c4...",
  "url": "https://stiiig.github.io/grist-widgets/widgets/emile/fiche-candidat?token=42.a3f9b2c4..."
}
```

Partager le champ `url` par email ou SMS au candidat.

> [!] Le fallback `?rowId=42` (sans token) reste fonctionnel pour le développement local — ne pas l'utiliser en production car il ne nécessite aucune authentification.

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.

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

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

Pièces jointes (AttachmentField)

Fonctionnalité	Disponible	Condition
Affichage des noms	Oui	Automatique (`_grist_Attachments` proxifié via workflow GET)
Téléchargement	Oui	Branche `?attachId=X` dans le workflow GET configurée
Upload	Oui	Workflow POST configuré
Suppression	Oui	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.

Expiration des magic links

Les tokens sont permanents (pas d'expiration). Pour révoquer un lien, il faudrait changer le secret HMAC (invalide tous les tokens existants). Une expiration par date pourrait être ajoutée ultérieurement.