Architecture proxy n8n — Gestion déportée de Grist

Document technique — contexte, choix d'architecture, état actuel

---

Contexte

Les widgets EMILE sont des interfaces React hébergées sur **GitHub Pages** (`stiiig.github.io`). En mode normal, ils tournent dans un **iframe embarqué dans Grist** : c'est le plugin Grist qui leur fournit l'accès aux données via `grist.docApi` (une API JavaScript injectée dans l'iframe).

Ce mode fonctionne bien pour les utilisateurs qui ouvrent Grist. Il ne couvre pas deux besoins clés :

- **Magic links** — envoyer un lien par email ou SMS à un candidat pour qu'il consulte/complète son dossier, sans qu'il ait accès à Grist
- **Formulaires publics** — `ajout-etablissement`, `creation-compte-orienteur` accessibles sans compte Grist

L'objectif : faire fonctionner ces widgets **hors iframe**, en appelant directement l'API Grist depuis le navigateur.

Pourquoi on ne peut pas appeler Grist directement depuis le browser

Deux obstacles rendent l'appel direct impossible :

1. CORS

Le navigateur applique la politique *Same-Origin* : un script chargé depuis `stiiig.github.io` ne peut pas faire de requête vers `grist.incubateur.dnum.din.developpement-durable.gouv.fr` sauf si ce dernier répond avec le header `Access-Control-Allow-Origin: *`. L'instance Grist interne ne le fait pas — les appels sont bloqués avant même d'atteindre le serveur.

2. Clé API exposée

L'API Grist REST exige un `Authorization: Bearer <clé>` sur chaque requête. Si on appelle Grist depuis le browser, la clé est visible dans le JS du bundle ou dans les DevTools — n'importe qui peut la récupérer et lire ou modifier toute la base de données.

> **Note** : ce problème sera partiellement résolu par une **clé de service** (compte applicatif Grist avec permissions restreintes à lecture/écriture sur les tables EMILE uniquement, sans accès admin). En attente de provisionnement côté infra. Même avec une clé de service, le problème CORS reste entier — le proxy reste nécessaire.

Solution : proxy n8n

n8n est la plateforme d'automatisation déjà déployée sur l'infra. Elle est accessible depuis internet et gère le CORS côté serveur.

```
Browser (GitHub Pages)
      |
      |  GET https://n8n.incubateur.dnum.../webhook/grist?table=CANDIDATS&token=ID.HMAC
      |  (pas de clé API, pas de contrainte CORS)
      v
n8n (accessible publiquement)
      |  vérifie HMAC-SHA256 du token
      |  Authorization: Bearer <clé API>   ← jamais vue par le browser
      |  GET https://grist.incubateur.dnum.../api/docs/{docId}/tables/CANDIDATS/records
      v
Grist (réseau interne, inaccessible depuis internet)
```

```
      |  {"records": [...]}
      v
n8n -> ajoute Access-Control-Allow-Origin: *
      v
Browser -> données disponibles dans React
```

Le proxy joue quatre rôles : 1. **Contournement CORS** — il appelle Grist server-side, sans contrainte cross-origin 2. **Isolation de la clé API** — elle reste dans les credentials n8n, jamais dans le bundle JS 3. **Passerelle réseau** — Grist est sur le réseau interne, n8n peut l'atteindre, le browser non 4. **Vérification du magic link** — HMAC-SHA256 vérifié côté n8n, secret jamais exposé

Architecture actuelle — trois workflows n8n

Workflow GET — lecture + téléchargement + vérification magic link

```
Webhook GET (?table=X&token=ID.HMAC  ou  ?table=X&filter=JSON  ou  ?attachId=Y)
    |
    v
IF query.token est présent
    +- True  -> Code (extrait rowId + sig)
    |          -> Crypto HMAC-SHA256 (même secret que GENERATE)
    |          -> IF sig === HMAC calculé
    |              +- True  -> IF query.attachId est présent
    |              |           +- True  -> GET /attachments/{id}/download -> Respond Binary
    |              |           +- False -> GET /tables/CANDIDATS/records?filter={"id":[rowId]}
    |              +- False -> Respond 403 { "error": "Token invalide" }
    |
    +- False -> IF query.attachId est présent
                +- True  -> GET /attachments/{id}/download -> Respond Binary
                +- False -> GET /tables/{table}/records    -> Respond JSON
```

La branche **faux** (pas de token) couvre toutes les requêtes de métadonnées et de tables de référence qui ne nécessitent pas de vérification (`_grist_Tables`, `_grist_Tables_column`, `DPTS_REGIONS`, etc.).

> [!] **Piège critique** : la branche **faux** du IF token doit pointer vers **IF attachId**, pas vers un Respond 403. Les requêtes de métadonnées (`_grist_Tables`, etc.) n'ont jamais de token — si elles tombent sur un 403, le widget ne peut plus charger les types de colonnes et les dropdowns.

Workflow POST — upload de pièces jointes

```
Webhook POST (multipart/form-data)
    |
    v
HTTP Request POST /attachments (Form-Data, champ upload, Bearer Auth)
    |
    v
Respond JSON { data: "[42]" } + CORS header
```

L'upload utilise `multipart/form-data` sans header custom — c'est ce qu'on appelle une *simple CORS request* (spec WHATWG Fetch) : le navigateur ne fait pas de preflight `OPTIONS` avant l'envoi. On peut donc utiliser un webhook POST simple sans avoir à gérer le handshake CORS OPTIONS.

Le code `rest.ts` parse la réponse de manière défensive car n8n sérialise les tableaux JSON de manière non déterministe selon sa version (`{"data":"[42]"}`, `[42]`, ou des objets items `{json: 42, pairedItem: ...}`).

Workflow GENERATE — génération de magic links

```
Webhook POST /webhook/grist-generate  (Basic Auth)
    |
```

```
    v
Code (extrait rowId du body)
    |
    v
Crypto HMAC-SHA256 (même secret que le workflow GET)
    |
    v
Code (construit token = rowId.HMAC et URL complète)
    |
    v
Respond to Webhook { rowId, token, url }
```

Génère un token signé `rowId.HMAC` pour un candidat donné. Appelé manuellement (curl) ou via une automation Grist à la création d'un enregistrement.

Ce que ça permet concrètement

Sans modifier Grist, sans plugin, sans compte Grist côté utilisateur :

Fonctionnalité	Mécanisme
Afficher la fiche d'un candidat via lien signé	`GET ?table=CANDIDATS&token=ID.HMAC`
Charger les listes déroulantes (départements, établissements…)	`GET ?table=DPTS_REGIONS` etc.
Lire les types/options de colonnes	`GET ?table=_grist_Tables` + `_grist_Tables_column`
Afficher les pièces jointes (noms)	`GET ?table=_grist_Attachments`
Télécharger une pièce jointe	`GET ?attachId=42` -> binaire
Uploader une pièce jointe	`POST multipart/form-data`
Générer un magic link signé	`POST /webhook/grist-generate` (Basic Auth)

Sécurité du magic link

Token format

```
token = rowId + "." + HMAC-SHA256(rowId.toString(), SECRET)
```

- **rowId** — identifiant de l'enregistrement Grist (entier)
- **HMAC-SHA256** — signature cryptographique avec un secret partagé entre les deux workflows n8n
- **SECRET** — stocké uniquement dans les credentials n8n (type Crypto), jamais dans le code ni dans le repo

Propriétés

Propriété	Valeur
Forgeable sans le secret	Non Non
Expiration	Non Permanent (pas d'expiration)
Révocable	[!] Uniquement en changeant le secret (invalide tous les tokens)
Lié à un candidat spécifique	Oui Oui (rowId dans le token)

Fallback dev

Le paramètre `?rowId=123` (sans signature) est conservé comme **fallback de développement** uniquement — le workflow GET n'exige pas de token pour les requêtes sans `?token=`. Ne pas utiliser en production.

Limitations et chantiers ouverts

Sauvegarde et soumission de formulaires

Les requêtes `POST` et `PATCH` avec `Content-Type: application/json` déclenchent un preflight `OPTIONS`. n8n doit répondre aux OPTIONS avec les bons headers CORS avant que le browser n'envoie la vraie requête — ce n'est pas encore configuré.

En pratique : lecture et upload de fichiers fonctionnent, mais le bouton "Enregistrer" de la fiche candidat et la soumission des formulaires d'ajout échoueront en mode REST jusqu'à résolution.

Options envisagées : - Ajouter des webhooks PATCH et OPTIONS dans n8n - Ou créer une **Next.js API Route** côté serveur (dans le repo) qui proxifie sans contrainte CORS — plus propre, pas de dépendance à n8n pour les écritures

Clé de service Grist

Actuellement la clé API configurée dans n8n est une clé personnelle. Une **clé de service** (compte applicatif, permissions minimales, rotation sans impact humain) est nécessaire pour la production. En attente de provisionnement côté infra Grist.

Expiration des tokens

Les magic links sont permanents. Pour des dossiers sensibles, une expiration (date butoir dans le token, vérifiée par n8n) pourrait être ajoutée ultérieurement.

Fichiers concernés dans le repo

Fichier	Rôle
`src/lib/grist/rest.ts`	Client REST — implémente `GristDocAPI` via proxy n8n
`src/lib/grist/init.ts`	Détecte `NEXT_PUBLIC_GRIST_PROXY_URL` et bascule en mode REST
`src/lib/grist/meta.ts`	Charge métadonnées colonnes via `_grist_Tables`
`src/components/AttachmentField.tsx`	Gestion pièces jointes (affichage, download fetch+blob, upload)
`docs/rest-mode.md`	Config n8n pas-à-pas avec tous les pièges rencontrés