

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** (`stiig.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 `stiig.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&filter=...
| (pas de clé API, pas de contrainte CORS)
```

```

▼
n8n (accessible publiquement)
  |   Authorization: Bearer <clé API>   ← jamais vue par le browser
  |   GET
https://grist.incubateur.dnum.../api/docs/{docId}/tables/CANDIDATS/records
▼
Grist (réseau interne, inaccessible depuis internet)
  |   {"records": [...]}
▼
n8n → ajoute Access-Control-Allow-Origin: *
▼
Browser → données disponibles dans React

```

Le proxy joue trois 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

Architecture actuelle — deux workflows n8n

La version n8n déployée ne supporte pas "Any Method" sur les webhooks (uniquement GET, POST, PATCH...). On a donc deux workflows distincts au même path `/webhook/grist` , routés par méthode HTTP :

Workflow GET — lecture + téléchargement de pièces jointes

```

Webhook GET (?table=X ou ?attachId=Y)
|
▼
IF query.attachId est présent
  |— True  → GET /attachments/{id}/download → Respond Binary (Content-Type
forwarded)
  |— False → GET /tables/{table}/records   → Respond JSON

```

Couvre : chargement de toutes les tables, métadonnées colonnes (`_grist_Tables` , `_grist_Tables_column`), noms des pièces jointes (`_grist_Attachments`), téléchargement de fichiers.

Workflow POST — upload de pièces jointes

```

Webhook POST (multipart/form-data)
|
▼
HTTP Request POST /attachments (Form-Data, champ upload, Bearer Auth)
|
▼
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: ...}`).

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 email	<code>GET ?table=CANDIDATS&filter={"id":42}</code>
Charger les listes déroulantes (départements, établissements...)	<code>GET ?table=DPTS_REGIONS</code> etc.
Lire les types/options de colonnes	<code>GET ?table=_grist_Tables + _grist_Tables_column</code>
Afficher les pièces jointes (noms)	<code>GET ?table=_grist_Attachments</code>
Télécharger une pièce jointe	<code>GET ?attachId=42</code> → binaire
Uploader une pièce jointe	<code>POST multipart/form-data</code>

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.

Sécurité du magic link

L'URL `?rowId=42` est publique — quiconque l'a peut consulter le dossier. À sécuriser : token signé (HMAC ou JWT), expiration, éventuellement vérification par email/SMS. Non prioritaire tant que les liens sont distribués à la demande.

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