Hoe we deze AI-schrijfomgeving geschikt maken voor gebruik in een schoolomgeving

Deze uitleg gaat over het veilig én betrouwbaar maken van de AI-schrijfomgeving, zodat docenten opdrachten kunnen klaarzetten en leerlingen op elke schoolcomputer met hun persoonlijke code kunnen werken. We verplaatsen de opslag van opdrachten van de browser naar een centrale server en zorgen ervoor dat de OpenRouter-API-sleutel nooit zichtbaar is voor leerlingen. Docenten gebruiken straks gewoon hun leerlinglinks, leerlingen krijgen overal toegang, en alle AI-functies draaien veilig op de achtergrond via de server.

Opdracht voor Lovable – centrale opslag van opdrachten (geen localStorage meer)

Probleem nu

Op dit moment worden opdrachten en beoordelingscriteria door de docentenpagina (`TeacherConfig`) opgeslagen in `localStorage` van de browser.
De leerlingpagina (`StudentWorkspace`) leest diezelfde data óók uit `localStorage` op basis van de code.

Gevolg:

- De link werkt alleen op **dezelfde computer + dezelfde browser** als waar de docent de opdracht heeft gemaakt.
- Op een andere computer bestaat `assignment_{code}` niet in `localStorage`, dus de leerling krijgt “link ongeldig / verlopen”.

Ik wil dat:

- Elke leerling **op elke computer binnen de school** met de code kan inloggen op zijn/haar opdracht.

- Eventueel later de app kan draaien binnen de **schoolomgeving / eigen server**, maar met dezelfde logica.

Gewenste architectuur

1. Centrale opslag van opdrachten

In plaats van `localStorage` wil ik een **centrale opslag**:

- Een database of key–value store (bijv. via Supabase of wat Lovable standaard aanbiedt).
- Daarin een tabel/collectie `assignments` met ongeveer deze velden:
- `code` (string, uniek, bijvoorbeeld de huidige 8-cijferige/lettercode)
- `level` (referentieniveau)
- `assignmentText` (de opdrachttekst)
- `criteria` (JSON met alle geselecteerde criteria)
- `createdAt`
- optioneel: `expiresAt` (bijv. voor tijdelijke opdrachten)
- optioneel: `teacherId` (als we later docenten-auth willen toevoegen)

---

2. Nieuwe backend-endpoints / edge functions

Ik wil twee basis–API-operaties:

1. **Aanmaken opdracht (docentkant)**
   Endpoint (bijv.): `POST /api/assignments`

Input:

- `level`
- `assignmentText`
- `criteria` (de gecombineerde lijst van geselecteerde criteria)

2. Werking:
   - Genereer een unieke `code` (zoals nu).
   - Sla alles op in de centrale database onder deze code.
   - Stuur de code terug naar de frontend.
3. Output:
   - `{ code: "3ZHP69WC" }`
4. **Ophalen opdracht (leerlingkant)**
   Endpoint (bijv.): `GET /api/assignments/{code}`

   Werking:
   - Zoek in de database naar `assignments` met deze `code`.
   - Als gevonden: stuur `level`, `assignmentText`, `criteria` terug.
   - Als niet gevonden: stuur een 404 of een duidelijke fout, zodat de UI het bericht "Deze link is ongeldig of verlopen" kan tonen.

---

3. Aanpassingen frontend – docentenpagina (TeacherConfig)

Nu:

- Bij het genereren van links wordt `assignmentData` in `localStorage` gezet met key `assignment_{code}`.

- De link zelf is alleen een URL met `/student/{code}`.

Gewenste wijziging:

- Op het moment dat de docent op “Genereer links” klikt:
    1. Maak eerst **één** opdrachtrecord aan via `POST /api/assignments` met:
        - `level`
        - `assignmentText`
        - `criteria` (alle geselecteerde criteria, dus AI-checklist + referentiekader + eigen criteria)
    2. Ontvang van de backend de `code`.
    3. Genereer de leerlinglinks alleen op basis van deze `code`, dus:
        - `https://…/student/{code}`
- `localStorage.setItem(...)` moet verdwijnen; opslag gebeurt alleen via de backend.

Belangrijk:
Als een docent meerdere leerlingen dezelfde opdracht geeft, mag je **één assignment-record + meerdere links** gebruiken (zelfde code) óf één code per leerling – dat mogen jullie zelf kiezen, als het maar consistent is. Voor nu mag één code per opdracht prima zijn.

---

4. Aanpassingen frontend – leerlingpagina (StudentWorkspace)

Nu:

- `StudentWorkspace` leest de `code` uit de URL.
- Maakt een key: `assignment_${code}`.
- Zoekt die in `localStorage`.

Gewenste wijziging:

- `StudentWorkspace` moet:
    1. De `code` uit de URL halen zoals nu.
    2. Een call doen naar `GET /api/assignments/{code}`.
    3. Bij succes:
        - de ontvangen data (`assignmentText`, `criteria`, `level`) in de state zetten als `assignmentData`.
    4. Bij fout / 404:
        - `isInvalidCode` op true zetten en de bestaande “Deze link is ongeldig of verlopen” melding tonen.
- Alle logica die nu met `localStorage.getItem` werkt, moet vervangen worden door deze API–call.

---

5. Optie: draaien binnen de schoolomgeving

Later wil ik de optie hebben om de app in de **schoolomgeving** te draaien (eigen server of eigen cloud). Architectuur blijft hetzelfde:

- Frontend (React / Vite) draait op een domein van de school.
- Backend (API endpoints) draait:
    - óf als edge functions via Supabase,
    - óf als eigen Node / Deno / server binnen de school.
- De database met `assignments` staat in de schoolomgeving.

Belangrijk hierbij:

- **Leerlingkant**:

    - geen persoonlijke login nodig; alleen de code.

- **Docentkant**:
    - later eventueel afschermen met school-SSO (Microsoft / Google / Magister), maar dat is voor een volgende fase.

- CORS en beveiliging zo instellen dat alleen de frontend van het schooldomein de API mag gebruiken.

---

6. Acceptatiecriteria

Deze wijziging is geslaagd als:

- Een docent op **computer A** een opdracht kan configureren en een link kan genereren.
- Een leerling op **computer B** (andere machine, andere browser) via `/student/{code}`:
    - de opdrachttekst ziet,
    - de criteria ziet / de AI-feedback kan gebruiken,
    - zonder dat er `localStorage`-data van de docent nodig is.
- De app blijft werken zoals nu qua UI/flow; alleen de opslag is verhuisd van `localStorage` naar een centrale backend.

Topvraag, dit is precies de plek waar veel projecten onbedoeld lek raken, dus goed dat je ‘m nu stelt.

Eerst het korte antwoord:

- **Nee, de API-code hoort *niet* “mee te gaan” met de leerlinglink.**
- De API-key moet **alleen op een veilige plek op de server** staan (of in de schoolomgeving), nooit in de link, nooit in de leerlingbrowser.

- De leerlinglink moet alleen de **opdrachtcode** bevatten.
  Als een leerling op “feedback” klikt, vraagt de backend:
  “Welke opdracht hoort bij deze code?” → pakt daarbij *zelf* de juiste API-key en model, en belt daarmee met OpenRouter.

Nu in twee delen: eerst simpel hoe het werkt, daarna een tekst die je later aan Lovable kunt geven.

---

1. Hoe zou het moeten werken? (simpel uitgelegd)

Nu (hackathon-setup)

- Jij typt een OpenRouter API-key in op de AI-instellingenpagina.
- Die wordt ergens opgeslagen (bijvoorbeeld lokaal of in Supabase).
- De leerlinglink zelf bevat **geen** API-key, alleen `/student/{code}`.
- Op dit moment is alles nog een beetje “plakkerig” omdat:
  - opdrachten in `localStorage` zitten,
  - de AI-key half front-end, half back-end geregeld is.

Straks (met centrale opslag én veilige AI-key)

Stel: je draait dit later in een schoolomgeving of netjes via Lovable met backend:

1. **Docent maakt opdracht**
   - Docent maakt opdracht + criteria.
   - De backend slaat dat op in de database onder een code, bijv. `3ZHP69WC`.
2. **AI-instellingen (API-key + model)**
   Twee varianten:

   Variant A – Eén centrale AI-key voor de hele school / installatie

- Er is een *beheerderspagina* (niet voor leerlingen, niet voor normale docenten).
- Daar zet iemand (ICT / beheer) één keer de OpenRouter-API-key in.
- De backend bewaart die als **server-secret** (env var / secrets-store).
- Docenten zien de key nooit, leerlingen al helemaal niet.

3. **Variant B – Per docent eigen AI-key (meer gedoe, maar flexibel)**
    - Docenten loggen in (SSO / schoolaccount).
    - Op een AI-instellingenpagina vullen ze hun eigen OpenRouter-key in.
    - De frontend stuurt die key één keer versleuteld naar de backend.
    - De backend slaat die versleuteld op, gekoppeld aan `teacherId`.
    - Bij feedback op een opdracht kijkt de backend:
      "Van welke docent is deze opdracht?" → gebruikt bijbehorende key.
4. **Leerling gebruikt de link**
    - Leerling opent `/student/3ZHP69WC` op *een willekeurige computer*.
    - Frontend stuurt die code naar de backend: "Geef opdracht + criteria."
    - Als de leerling op "Vraag feedback" klikt:
        - De backend haalt:
            - de opdracht en criteria (uit DB),
            - de juiste AI-config (API-key + model) → **alleen op de server**,
        - doet de OpenRouter-call,
        - stuurt alleen de **feedback** terug naar de leerling.
    - De leerling **krijgt nooit de API-key te zien**, die blijft achter de schermen.

Belangrijk:
De **link** bevat alleen een code.
De **API-key reist nooit mee** met die link.

---

2. Tekst voor later aan Lovable (inclusief schoolomgeving-scenario)

Hier een tekst die je later zo aan Lovable kunt geven:

Opdracht voor Lovable – veilige afhandeling van de OpenRouter API-key in combinatie met centrale opdrachten

Ik wil dat de manier waarop de OpenRouter API-key wordt gebruikt volledig veilig wordt ingericht en past bij een schoolomgeving.

Huidige situatie (globaal)

- Docenten kunnen een OpenRouter API-key invoeren op een AI-instellingenpagina.
- Leerlingen gebruiken een link zoals `/student/{code}` om hun opdracht en AI-feedback te krijgen.
- Op dit moment is de opslag van opdrachten en de AI-key nog deels in de frontend (localStorage / browser).

Dit moet veiliger en centraal.

---

1. Belangrijkste eisen

1. **De OpenRouter API-key mag nooit zichtbaar zijn voor leerlingen.**
    - Niet in de HTML, niet in JavaScript, niet in de network-tab, niet in localStorage, niet in de URL.
2. **De API-key mag niet als tekst in de database bij opdrachten opgeslagen worden.**

    - Opdrachtrecords (`assignments`) mogen alleen functionele data bevatten (code, opdrachttekst, criteria, etc.), geen geheimen.

3. **API-aanroepen naar OpenRouter moeten altijd vanuit de backend gebeuren.**
    - Frontend stuurt: tekst + context (criteria, code, etc.).
    - Backend voegt de API-key toe en stuurt de call naar OpenRouter.
    - Frontend ziet alleen de feedback, nooit de key.

---

2. Gewenste architectuur AI-config

Ik zie twee scenario's; ik wil dat de code zo geschreven wordt dat we makkelijk kunnen kiezen of omschakelen.

Scenario A – Eén centrale AI-config per installatie (aanbevolen voor schoolomgeving)

- Er komt een **admin-instellingenpagina** (niet publiek, niet voor leerlingen) waar een beheerder:
    - één OpenRouter API-key kan invoeren,
    - een standaardmodel kan kiezen (bijv. `openrouter/…`).
- De backend slaat deze API-key en model **niet in een tabel op**, maar gebruikt:
    - environment variables / secrets (bijv. `OPENROUTER_API_KEY`, `OPENROUTER_MODEL`).
- AI-gerelateerde backend-functies (bijv. voor:
    - analyseren van de opdracht (docentenpagina),
    - genereren van criteria,
    - feedback op leerlingtekst,

    - AI-reflectie voor de PDF,
) lezen de key **altijd** uit de server-secret (`Deno.env` of equivalent).

- De docent-zijde kan hooguit het model kunnen selecteren uit een lijst, maar nooit de key lezen.

Scenario B – Per docent een eigen AI-key (optioneel, later)

- Voeg later eventueel docenten-auth toe (bijv. via Supabase Auth / SSO).
- Elke docent krijgt een eigen record in een `teacher_ai_config`-tabel:
    - `teacher_id`
    - `encrypted_api_key`
    - `preferred_model`
- De AI-instellingenpagina stuurt de API-key via HTTPS naar de backend.
- De backend:
    - versleutelt de key,
    - slaat hem op bij `teacher_id`,
    - geeft de key nooit meer terug naar de frontend.
- Als een opdracht AI gebruikt, wordt het `teacher_id`-veld in `assignments` mee opgeslagen.
- Bij een AI-call (docent of leerling):
    - De backend kijkt naar `teacher_id` bij de opdracht,
    - haalt de juiste key uit `teacher_ai_config`,
    - gebruikt die server-side om OpenRouter aan te roepen.

Voor nu is **Scenario A** voldoende; Scenario B is voor later.

---

3. Relatie met leerlinglinks en centrale opdrachten

Ik wil dat leerlinglinks **nooit** afhankelijk zijn van waar of hoe de API-key is opgeslagen.

Flow:

1. Docent maakt een opdracht en gebruikt AI om criteria/checklist te genereren.
2. De backend slaat de opdracht centraal op in een `assignments`-tabel:
    - `code`, `assignmentText`, `criteria`, `level`, evt. `teacherId`.
3. Leerlinglink: `/student/{code}`
    - De frontend stuurt `code` naar een backend-endpoint.
    - De backend geeft opdracht + criteria terug (maar geen secrets).
4. Als de leerling op “Vraag feedback” klikt:
    - Frontend stuurt alleen:
        - `code`
        - leerlingtekst
        - context (bijv. ronde, eerdere feedback, etc.).
    - Backend:
        - zoekt de opdracht bij `code`,
        - bepaalt welke AI-config hoort bij deze opdracht (centrale key of per docent),
        - gebruikt de API-key server-side,
        - stuurt alleen de AI-feedback terug.

Conclusie:
De API-key “gaat niet mee” met de leerlinglink, maar wordt **automatisch gekozen en gebruikt op de server** wanneer er feedback of analyse nodig is.

---

4. Acceptatiecriteria

Deze wijziging is correct geïmplementeerd als:

- In de frontend **nooit** een OpenRouter API-key te vinden is:
    - niet in code,
    - niet in responses,
    - niet in localStorage,
    - niet in de URL.
- Alle AI-calls vanaf de leerlingpagina en docentenpagina:
    - gaan via backend-endpoints,
    - gebruiken daar de server-side secret(s).
- Een leerlinglink (`/student/{code}`):
    - werkt op elke computer binnen of buiten de school,
    - zonder dat er lokaal een API-key hoeft te bestaan.
- Bij een toekomstige schooldeploy:
    - hoeft alleen de server-omgeving ingesteld te worden met environment variables / secrets;
    - leerlingen en docenten hoeven nooit direct met de OpenRouter-key te werken.

Dit is precies wat ik wil:
De **opdrachtcode** gaat met de leerling mee,
de **API-key blijft altijd opgesloten in de (school)server / backend**.