**02 - Technisch Architectuur Document (TAD) voor NEXA**

**Status:** IN PROGRESS

**Inhoudsopgave:**

1. Architectuuroverzicht
2. Componenten en Technologiekeuzes
3. Dataflow Beschrijvingen (TODO)
4. Dataopslag Architectuur (TODO)
5. API Architectuur (TODO)
6. Security Architectuur (TODO)
7. Deployment Architectuur (TODO)

**1. Architectuuroverzicht**

NEXA volgt een moderne webapplicatie-architectuur, gecentreerd rond **Supabase** als Backend-as-a-Service (BaaS) platform en **Neo4j** als gespecialiseerde graph database. De architectuur bestaat uit de volgende hoofdlagen:

1. **Frontend Laag (React):** De gebruikersinterface, draaiend in de browser van de gebruiker. Verantwoordelijk voor weergave, gebruikersinteractie en communicatie met de API Laag.
2. **API Laag (Supabase Edge Functions):** De backend logica, gehost als serverless functions binnen Supabase. Verantwoordelijk voor het afhandelen van requests van de frontend, orkestreren van de dataflow, authenticatie/autorisatie en communicatie met de datalagen en externe services.
3. **Dataopslag Laag:**
   * **Supabase Postgres DB:** Relationele database voor persistente opslag van gebruikersdata, authenticatie-info, applicatieconfiguraties, metadata snapshots (geparsed), analyseresultaten en logs. Maakt gebruik van Row Level Security (RLS) voor data-scheiding.
   * **Neo4j Graph DB:** Gespecialiseerde graph database (bij voorkeur cloud-hosted via Neo4j AuraDB) voor opslag van de knowledge graph (metadata componenten en relaties per snapshot). Wordt gebruikt voor complexe dependency tracing en impactanalyse queries.
4. **Externe Services Laag:**
   * **Salesforce Platform:** Bron van de metadata, benaderd via OAuth 2.0 en de Metadata/Tooling API's.
   * **AI Engine (LLM - OpenAI/Claude):** Externe service voor het genereren van natuurlijke taal uitleg en remediatiesuggesties, aangeroepen vanuit de API Laag.

**Hoog-niveau Diagram (Beschrijvend):**

* De **Gebruiker** interageert via de browser met de **React Frontend**.
* De **React Frontend** stuurt HTTPS Requests naar de **API Laag (Supabase Edge Functions)** binnen het **Supabase Platform**.
* Binnen het **Supabase Platform**:
  + De **API Laag** communiceert via SQL/RPC met de **Supabase Postgres DB**.
  + De **API Laag** communiceert via HTTPS/Bolt met de **Neo4j Graph DB (AuraDB)**, die zich als externe service buiten Supabase bevindt.
  + De **API Laag** doet API Calls naar de **AI Engine (LLM)** (externe service).
  + De **API Laag** doet OAuth/API Calls naar het **Salesforce Platform** (externe service).
  + De **Supabase Postgres DB** wordt beveiligd met **Row Level Security Policies**.
  + De **API Laag** voert **Authenticatie Checks** uit.
* De **Externe Services** omvatten het **Salesforce Platform**, **Neo4j Graph DB**, en de **AI Engine (LLM)**.

**Kernprincipes:**

* **Supabase-Centric:** Maximale benutting van Supabase features (Auth, DB, Functions, Storage) om complexiteit te minimaliseren.
* **Specialized Datastores:** Gebruik van Postgres voor gestructureerde data en Neo4j voor graph-specifieke queries.
* **Serverless:** API laag draait op schaalbare serverless functions.
* **API-Driven:** Communicatie tussen frontend en backend verloopt via een goed gedefinieerde API.
* **Snapshot-Based:** Analyses zijn gebaseerd op point-in-time metadata snapshots.

**2. Componenten en Technologiekeuzes**

Hieronder een gedetailleerd overzicht van de technologieën per component.

**2.1 Frontend (React)**

* **Framework:** React (v18+)
* **Taal:** TypeScript
* **UI Bibliotheek:** Tailwind CSS + Headless UI componenten (of een componentenbibliotheek zoals Shadcn/UI, Material UI, Mantine - nader te bepalen in UI/UX fase)
* **State Management:** Zustand (of Redux Toolkit, nader te bepalen) - voorkeur voor lichtgewicht oplossing.
* **Routing:** React Router
* **Data Fetching:** React Query (TanStack Query) voor caching, achtergrond updates, etc. i.c.m. Supabase JS Client.
* **Graph Visualisatie:** Cytoscape.js (aanbevolen vanwege flexibiliteit en performance) of D3.js (meer low-level controle).
* **Hosting:** Vercel (aanbevolen vanwege naadloze integratie met Next.js/React) of Netlify.

**2.2 API Laag (Supabase Edge Functions)**

* **Runtime:** **Node.js** (LTS versie, bijv. v18 of v20)
* **Taal:** TypeScript
* **Framework:** **Express.js** (voor robuuste routing, middleware en request/response handling)
* **Supabase Interactie:** Supabase JS Client (@supabase/supabase-js)
* **Neo4j Interactie:** Officiële Neo4j JavaScript Driver (neo4j-driver)
* **Salesforce Interactie:** JSforce (robuuste library voor diverse Salesforce API's)
* **LLM Interactie:** Officiële SDK's van OpenAI/Anthropic of directe fetch calls.
* **Build Proces:** TypeScript compilatie naar JavaScript (bijv. met tsc of esbuild) als onderdeel van het deployment proces voor Edge Functions.

**2.3 Dataopslag Laag**

* **Supabase Postgres DB:**
  + Versie: Laatste stabiele versie aangeboden door Supabase.
  + Schema Beheer: Supabase Migrations (SQL-bestanden beheerd in Git).
  + Toegang: Via Supabase JS Client (RPC en REST-like API) en directe SQL vanuit Edge Functions.
  + Beveiliging: Row Level Security (RLS) policies, Network restrictions (indien mogelijk/nodig).
* **Neo4j Graph DB:**
  + Hosting: **Neo4j AuraDB (Managed Cloud Service - Sterk Aanbevolen)**. Biedt schaalbaarheid, backups, security en beheergemak. Alternatief: Self-hosted (Docker).
  + Versie: Laatste stabiele versie (bijv. Neo4j 5.x).
  + Query Taal: Cypher.
  + Toegang: Via Neo4j JavaScript Driver (Bolt protocol).
  + Beveiliging: Authenticatie via database user/password, netwerk policies (AuraDB).

**2.4 Externe Services Laag**

* **Salesforce Platform:**
  + Authenticatie: OAuth 2.0 PKCE Flow.
  + API's: Metadata API (voor ophalen componenten), Tooling API (mogelijk voor snellere queries of specifieke info), REST/SOAP API (indien nodig voor andere info).
* **AI Engine (LLM):**
  + Provider: OpenAI (GPT-4 / GPT-3.5-Turbo) of Anthropic (Claude 2/3). Keuze kan configureerbaar zijn.
  + Integratie: Via officiële SDK's of directe REST API calls vanuit Supabase Functions.
  + Beveiliging: API keys veilig beheerd als secrets in Supabase Functions environment.

**2.5 Ontwikkeling & Tooling**

* **Versiebeheer:** Git (Repository gehost op GitHub, GitLab, etc.)
* **Code Editor:** VS Code (met relevante extensies voor TS, React, Tailwind, Docker, etc.)
* **Package Manager:** PNPM (aanbevolen voor efficiënt diskgebruik in monorepo) of NPM/Yarn.
* **Lokale Ontwikkeling:**
  + Supabase CLI (voor lokaal draaien Supabase stack, incl. DB, Auth, Functions).
  + Docker Compose (voor optioneel lokaal draaien Neo4j).
* **CI/CD:** GitHub Actions (aanbevolen) of GitLab CI.

**3. Dataflow Beschrijvingen**

Deze sectie beschrijft de sequenties van interacties tussen componenten voor kernfunctionaliteiten.

**3.1 Dataflow: Metadata Pipeline (Fetch, Parse, Store, Build Graph)**

Dit proces wordt getriggerd door de gebruiker vanuit de frontend om Salesforce metadata te importeren en de knowledge graph te bouwen/updaten.

1. **Trigger (Frontend):**
   * Gebruiker selecteert een gekoppelde Salesforce Org en klikt op "Fetch Metadata" (of een vergelijkbare knop voor de volledige pipeline).
   * De React Frontend stuurt een authenticated HTTPS POST request naar een specifieke API endpoint op de **Supabase Edge Function (Node.js/Express)**, bijv. /api/pipeline/start. De request bevat het ID van de gekoppelde org.
2. **Start Pipeline (API Function):**
   * De Edge Function valideert de request en de autorisatie van de gebruiker (via Supabase Auth context).
   * Maakt een nieuw metadata\_snapshot record aan in de **Supabase Postgres DB** met status 'PENDING' of 'FETCHING', gekoppeld aan de org en gebruiker. Retourneert het snapshot\_id naar de frontend (of start het proces asynchroon).
3. **Fetch Metadata (API Function):**
   * De Edge Function haalt de benodigde Salesforce access token (veilig opgeslagen in de **Supabase DB**) op voor de geselecteerde org.
   * Gebruikt JSforce om verbinding te maken met de **Salesforce Platform** Metadata API.
   * Vraagt de benodigde metadata types op (Objecten, Fields, Flows, etc., zoals gedefinieerd in FR-PIPE-02). Dit kan meerdere API calls vereisen.
   * *Status Update:* Werkt de status van de metadata\_snapshot in de **Supabase DB** bij naar 'FETCHING'.
4. **Parse Metadata (API Function):**
   * Na succesvolle ontvangst van de metadata (vaak XML/ZIP), parsed de Edge Function deze data.
   * De parsing logica zet de ruwe metadata om naar de **vooraf gedefinieerde, consistente structuur** per component type (zie FR-PIPE-03).
   * *Status Update:* Werkt de status van de metadata\_snapshot in de **Supabase DB** bij naar 'PARSING'.
5. **Store Metadata (API Function):**
   * De Edge Function schrijft de gestructureerde, geparste metadata weg naar de relevante tabellen (bijv. parsed\_components, parsed\_relationships) in de **Supabase Postgres DB**, gekoppeld aan het snapshot\_id.
   * *Status Update:* Werkt de status van de metadata\_snapshot in de **Supabase DB** bij naar 'STORING' of 'STORED'.
6. **Start Graph Population (API Function - mogelijk aparte trigger/functie):**
   * Nadat de metadata succesvol is opgeslagen in Supabase, wordt de graph populatie getriggerd (ofwel direct aansluitend, of via een aparte gebruikersactie/endpoint die het snapshot\_id meegeeft).
   * *Status Update:* Werkt de status van de metadata\_snapshot in de **Supabase DB** bij naar 'BUILDING\_GRAPH'.
7. **Populate Graph (API Function):**
   * De Edge Function leest de gestructureerde metadata voor het betreffende snapshot\_id uit de **Supabase DB**.
   * Vertaalt deze data naar Cypher CREATE of MERGE statements voor nodes en relaties.
   * Maakt verbinding met de **Neo4j Graph DB (AuraDB)** via de Neo4j JavaScript Driver.
   * Voert de Cypher queries uit om de graph voor dit snapshot\_id te bouwen of bij te werken.
8. **Voltooiing (API Function & Frontend):**
   * Na succesvolle graph populatie, werkt de Edge Function de status van de metadata\_snapshot in de **Supabase DB** bij naar 'COMPLETED'.
   * De **Frontend** kan periodiek de status van de snapshot opvragen via een API endpoint (bijv. /api/pipeline/status/{snapshot\_id}) en toont de 'Completed' status aan de gebruiker, waarna de visualisatie en analyse beschikbaar worden.
   * Bij fouten in eender welke stap wordt de status bijgewerkt naar 'FAILED' met een foutmelding, die ook zichtbaar is in de frontend.

**3.2 Dataflow: AI Impact Analyse Vraag**

Dit proces beschrijft hoe een vraag van de gebruiker in de AI Assistant wordt afgehandeld.

1. **Vraag Stellen (Frontend):**
   * Gebruiker selecteert een actieve snapshot en typt een vraag in de AI Assistant UI (bijv. "Wat gebeurt er als ik Flow X aanpas?").
   * De React Frontend stuurt een authenticated HTTPS POST request naar een API endpoint op de **Supabase Edge Function (Node.js/Express)**, bijv. /api/ai/ask. De request bevat de vraagtekst en het snapshot\_id.
2. **Vraag Verwerking (API Function):**
   * De Edge Function valideert de input en autorisatie.
   * Interpreteert de gebruikersvraag (FR-AI-02). Dit kan simpele patroonherkenning zijn of een complexere NLU-stap (eventueel zelfs een eerste LLM call voor intentieherkenning). Doel is om de *intentie* (bijv. impactanalyse, relatie zoeken) en de *entiteiten* (bijv. Flow X, Veld Y) te extraheren.
3. **Graph Query (API Function):**
   * Op basis van de geïnterpreteerde vraag, genereert de Edge Function een of meerdere Cypher queries om de relevante informatie uit de **Neo4j Graph DB** te halen voor het opgegeven snapshot\_id (FR-AI-02, AC-AI-02-01).
   * Voorbeeld: Voor een impactanalyse query, zoekt het naar alle paden die uitgaan van de node die Flow X representeert.
   * Voert de Cypher query(s) uit op de **Neo4j DB**.
4. **Context Voorbereiding (API Function):**
   * De Edge Function verzamelt de resultaten van de Neo4j query (bijv. een lijst van afhankelijke componenten en hun types).
   * Bereidt de context voor die naar de LLM gestuurd moet worden. Dit omvat minimaal: de oorspronkelijke vraag, de query resultaten, en eventuele instructies voor de LLM over hoe het antwoord geformuleerd moet worden (FR-AI-03, AC-AI-03-01).
5. **LLM Aanroep (API Function):**
   * De Edge Function stuurt de voorbereide prompt naar de geconfigureerde **AI Engine (LLM - OpenAI/Claude)** via diens API.
   * Wacht op het antwoord van de LLM.
6. **Antwoord Verwerking & Opslag (API Function):**
   * De Edge Function ontvangt het gegenereerde antwoord (de uitleg) van de LLM.
   * Verwerkt het antwoord indien nodig (bijv. opmaak, extractie van kernpunten).
   * Slaat de interactie (vraag, Neo4j resultaat (samengevat), LLM antwoord, feedback placeholder) eventueel op in de **Supabase DB** voor logging en de feedback loop (FR-AI-07).
   * Stuurt het verwerkte antwoord terug naar de frontend in de API response.
7. **Antwoord Tonen (Frontend):**
   * De React Frontend ontvangt het antwoord van de API.
   * Toont het AI-gegenereerde antwoord aan de gebruiker in de AI Assistant UI (FR-AI-04).
   * Toont eventuele feedbackknoppen (FR-AI-06).
8. **(Optioneel) Feedback Verwerking:**
   * Als de gebruiker feedback geeft (👍/👎/commentaar), stuurt de Frontend dit naar een apart API endpoint (bijv. /api/ai/feedback).
   * De betreffende Edge Function slaat deze feedback op in de **Supabase DB**, gekoppeld aan de oorspronkelijke interactie (FR-AI-07).

**4. Dataopslag Architectuur**

Deze sectie beschrijft de structuur en modellen voor dataopslag in Supabase Postgres en Neo4j.

4.1 Supabase (PostgreSQL) Datamodel

De Supabase database wordt gebruikt voor gestructureerde data, gebruikersbeheer, en het persisteren van de geparste metadata voordat deze naar Neo4j gaat.

Kern Tabellen (Voorbeelden - niet uitputtend):

* users (Beheerd door Supabase Auth):
  + id (uuid, primary key) - Standaard Supabase user ID.
  + email (text) - Standaard Supabase.
  + ... (Andere standaard Supabase Auth velden).
* organizations (Als gebruikers meerdere orgs kunnen beheren):
  + id (uuid, primary key) - Uniek ID voor de tenant/organisatie in NEXA.
  + name (text) - Naam van de organisatie.
  + owner\_id (uuid, foreign key naar users.id) - Optioneel, wie de org heeft aangemaakt.
  + created\_at (timestampz).
* organization\_members (Koppelt users aan organizations):
  + organization\_id (uuid, foreign key naar organizations.id).
  + user\_id (uuid, foreign key naar users.id).
  + role (text, bijv. 'admin', 'member').
  + created\_at (timestampz).
  + *(Primary key op organization\_id, user\_id)*.
* salesforce\_connections:
  + id (uuid, primary key).
  + organization\_id (uuid, foreign key naar organizations.id) - Aan welke NEXA org deze SF connectie behoort.
  + sf\_org\_id (text, 18-char) - Unieke Salesforce Org ID.
  + sf\_user\_id (text) - Salesforce User ID van de authenticerende gebruiker.
  + instance\_url (text) - Salesforce instance URL.
  + access\_token (text, encrypted) - Versleuteld access token.
  + refresh\_token (text, encrypted) - Versleuteld refresh token.
  + status (text, bijv. 'active', 'inactive', 'revoked').
  + last\_connected\_at (timestampz).
  + created\_at (timestampz).
  + user\_id (uuid, foreign key naar users.id) - Wie de koppeling heeft gelegd.
* metadata\_snapshots:
  + id (uuid, primary key) - Uniek ID voor deze snapshot run.
  + sf\_connection\_id (uuid, foreign key naar salesforce\_connections.id) - Welke SF org.
  + status (text, 'PENDING', 'FETCHING', 'PARSING', 'STORING', 'STORED', 'BUILDING\_GRAPH', 'COMPLETED', 'FAILED').
  + error\_message (text, nullable) - Details bij status 'FAILED'.
  + triggered\_by (uuid, foreign key naar users.id).
  + created\_at (timestampz).
  + completed\_at (timestampz, nullable).
* parsed\_components (Voor opslag geparste metadata - Kan opgesplitst worden per type indien nodig):
  + id (uuid, primary key).
  + snapshot\_id (uuid, foreign key naar metadata\_snapshots.id, index).
  + component\_type (text, bijv. 'CustomObject', 'CustomField', 'Flow', 'ApexTrigger', 'Report', index).
  + api\_name (text, index) - Volledige API naam (bijv. 'Account', 'Account.Name', 'MyFlow\_\_c').
  + label (text, nullable).
  + sf\_id (text, nullable) - Salesforce ID indien beschikbaar.
  + attributes (jsonb) - Opslag voor type-specifieke attributen (bijv. voor Field: datatype, isRequired; voor Flow: status, processType).
  + raw\_definition (text, nullable) - Optioneel, opslag van de ruwe XML/JSON definitie.
  + created\_at (timestampz).
* parsed\_relationships (Expliciete relaties uit parsing):
  + id (uuid, primary key).
  + snapshot\_id (uuid, foreign key naar metadata\_snapshots.id, index).
  + source\_component\_id (uuid, foreign key naar parsed\_components.id).
  + target\_component\_id (uuid, foreign key naar parsed\_components.id).
  + relationship\_type (text, bijv. 'FIELD\_USED\_IN\_FLOW', 'FLOW\_REFERENCES\_FIELD', 'FIELD\_BELONGS\_TO\_OBJECT', index).
  + context (jsonb, nullable) - Extra context over de relatie (bijv. specifiek flow element waar veld gebruikt wordt).
  + created\_at (timestampz).
* analysis\_results:
  + id (uuid, primary key).
  + snapshot\_id (uuid, foreign key naar metadata\_snapshots.id, index).
  + analysis\_type (text, bijv. 'Flow Inconsistency Check', 'Unused Component', index).
  + component\_id (uuid, foreign key naar parsed\_components.id, nullable) - Component waar het resultaat over gaat.
  + severity (text, bijv. 'Error', 'Warning', 'Info').
  + description (text) - Beschrijving van het gevonden probleem/resultaat.
  + details (jsonb, nullable) - Extra gestructureerde details.
  + created\_at (timestampz).
* ai\_interactions (Voor logging en feedback loop):
  + id (uuid, primary key).
  + snapshot\_id (uuid, foreign key naar metadata\_snapshots.id).
  + user\_id (uuid, foreign key naar users.id).
  + prompt (text) - De vraag van de gebruiker.
  + neo4j\_query\_summary (text, nullable) - Samenvatting/type van uitgevoerde query.
  + llm\_response (text) - Het antwoord van de LLM.
  + feedback\_rating (integer, nullable, bijv. 1=👍, -1=👎).
  + feedback\_comment (text, nullable).
  + created\_at (timestampz).

Beveiliging: Row Level Security (RLS) policies moeten worden geïmplementeerd om ervoor te zorgen dat gebruikers alleen toegang hebben tot data (connections, snapshots, components, results) die behoren tot de organizations waarvan zij lid zijn.

4.2 Neo4j Graph Model

De Neo4j database bevat de knowledge graph, geoptimaliseerd voor het traceren van afhankelijkheden. Elke node en relatie bevat een snapshotId property om data van verschillende snapshots te scheiden.

Node Labels (Voorbeelden):

* :MetadataComponent (Abstract label, alle componenten erven dit)
  + Properties: snapshotId (string, index), apiName (string, index), label (string), sfId (string, nullable), type (string), createdAt (datetime)
* :CustomObject (Erft van :MetadataComponent)
  + Properties: sharingModel (string), ...
* :CustomField (Erft van :MetadataComponent)
  + Properties: dataType (string), isRequired (boolean), formula (string, nullable), ...
* :Flow (Erft van :MetadataComponent)
  + Properties: processType (string), status (string), ...
* :ApexTrigger (Erft van :MetadataComponent)
  + Properties: events (list<string>, e.g., ['before insert', 'after update']), ...
* :Report (Erft van :MetadataComponent)
  + Properties: reportType (string), ...
* ... (Andere labels voor PermissionSet, Profile, ApexClass, LWC, etc. naarmate ondersteuning wordt toegevoegd)

Relationship Types (Voorbeelden):

* BELONGS\_TO
  + (f:CustomField)-[:BELONGS\_TO]->(o:CustomObject)
* REFERENCES (Algemene verwijzing)
  + (flow:Flow)-[:REFERENCES]->(field:CustomField)
  + (flow:Flow)-[:REFERENCES]->(obj:CustomObject)
  + (trigger:ApexTrigger)-[:REFERENCES]->(field:CustomField)
* USED\_IN (Specifieker dan REFERENCES, kan context bevatten)
  + (field:CustomField)-[:USED\_IN {elementType: 'Decision', flowElementApiName: 'Check\_Status'}]->(flow:Flow)
  + (obj:CustomObject)-[:USED\_IN]->(report:Report)
* TRIGGERS
  + (trigger:ApexTrigger)-[:TRIGGERS {onEvents: ['before insert']}]->(obj:CustomObject) (Of andersom, afhankelijk van modelkeuze)
* IMPACTS (Mogelijk afgeleid/hoger niveau relatie)
  + Kan gebruikt worden om complexere, indirecte impactpaden samen te vatten.

Indexering: Er moeten indexen worden aangemaakt op :MetadataComponent(snapshotId) en :MetadataComponent(apiName) voor snelle lookups. Afhankelijk van de queries kunnen meer specifieke indexen nodig zijn (bijv. op :CustomField(dataType)).

Constraint: Een UNIQUE constraint op (:MetadataComponent {snapshotId, apiName}) zorgt ervoor dat elk component uniek is binnen een snapshot.

Model Evolutie: Dit model is een startpunt. Tijdens ontwikkeling en op basis van performance analyse kunnen labels, properties en relatietypes worden verfijnd.

**5. API Architectuur**

Deze sectie definieert de architectuur van de API-laag die draait op Supabase Edge Functions en dient als de interface tussen de frontend en de backend logica/data.

**5.1 API Stijl en Format**

* **Stijl:** RESTful API. We gebruiken standaard HTTP-methoden (GET, POST, PUT, DELETE) en resource-gebaseerde URL's.
* **Format:** JSON voor zowel request bodies als response bodies.
* **Content Type:** application/json.

**5.2 Hosting en Framework**

* **Hosting:** Supabase Edge Functions. Elke (of een groep gerelateerde) endpoint(s) wordt geïmplementeerd als een aparte Edge Function.
* **Framework:** Node.js runtime met het Express.js framework voor routing, request/response handling en middleware.

**5.3 Basispad en Versioning**

* **Basispad:** Alle API-endpoints zullen beschikbaar zijn onder een basispad, bijvoorbeeld /api/v1.
* **Versioning:** API-versies worden beheerd via het URL-pad (bijv. /api/v1, /api/v2). Dit maakt het mogelijk om toekomstige wijzigingen door te voeren zonder bestaande integraties (zoals de frontend) te breken.

**5.4 Authenticatie en Autorisatie**

* **Authenticatie:** Alle API-endpoints (behalve eventuele publieke endpoints zoals login/registratie) vereisen authenticatie. Dit wordt afgehandeld via JSON Web Tokens (JWT) die worden uitgegeven door Supabase Auth. De frontend stuurt het JWT mee in de Authorization: Bearer <token> header bij elke request. Een Express middleware valideert het token bij elke beschermde request.
* **Autorisatie:** Na succesvolle authenticatie bepaalt de API-logica of de gebruiker gemachtigd is om de gevraagde actie uit te voeren op de betreffende resource. Dit gebeurt op basis van:
  + De user\_id uit het gevalideerde JWT.
  + Lidmaatschap en rol binnen organizations (indien van toepassing).
  + Controles op eigendom van resources (bijv. mag gebruiker X snapshot Y zien?).
  + Supabase Row Level Security (RLS) wordt ingezet op database-niveau als extra beveiligingslaag.

**5.5 Endpoint Definitie Overzicht (Conceptueel)**

Hieronder volgt een conceptueel overzicht van de belangrijkste API-resources en endpoints. De exacte specificaties (request/response details) worden verder uitgewerkt in het aparte "API Specificaties" document.

* **Authenticatie (Grotendeels afgehandeld door Supabase Client SDK):**
  + Standaard Supabase endpoints voor login, registratie, wachtwoord reset, etc.
  + POST /api/v1/auth/sfdc/initiate: Start de OAuth flow om een Salesforce org te koppelen (FR-AUTH-03).
  + GET /api/v1/auth/sfdc/callback: Callback endpoint voor Salesforce OAuth flow.
* **Salesforce Connections:**
  + GET /api/v1/connections: Haalt de lijst van gekoppelde Salesforce orgs op voor de ingelogde gebruiker/organisatie (Relateert aan FR-AUTH-03).
  + GET /api/v1/connections/{connectionId}: Haalt details van een specifieke koppeling op.
  + DELETE /api/v1/connections/{connectionId}: Verwijdert een Salesforce koppeling.
* **Metadata Pipeline & Snapshots:**
  + POST /api/v1/pipeline/start: Start de volledige metadata pipeline (Fetch, Parse, Store, Build Graph) voor een specifieke connectionId (Relateert aan FR-PIPE-01, FR-GRAPH-01). Request body: { "connectionId": "..." }. Response: { "snapshotId": "...", "initialStatus": "PENDING" }.
  + GET /api/v1/snapshots?connectionId={connectionId}: Haalt een lijst van metadata snapshots op voor een connectie, inclusief hun status (Relateert aan FR-PIPE-07).
  + GET /api/v1/snapshots/{snapshotId}/status: Haalt de actuele status van een specifieke snapshot run op (Relateert aan FR-PIPE-05).
  + GET /api/v1/snapshots/{snapshotId}: Haalt details van een specifieke snapshot op (na voltooiing).
* **Graph & Analyse:**
  + GET /api/v1/graph/{snapshotId}/visualize?depth={depth}&focus={nodeId}: Haalt data op voor de graph visualisatie van een specifieke snapshot (Nodes & Edges). Parameters voor initiële focus en diepte (Relateert aan FR-UI-03).
  + GET /api/v1/graph/{snapshotId}/component/{componentApiName}: Haalt details en directe relaties van een specifiek component op uit de graph.
  + POST /api/v1/analysis/{snapshotId}/start: Start het analyse proces (bijv. Flow Checks) voor een snapshot (Relateert aan FR-GRAPH-06).
  + GET /api/v1/analysis/{snapshotId}/results: Haalt de opgeslagen analyse resultaten op voor een snapshot (Relateert aan FR-RSLT-01, FR-UI-08).
* **AI Assistant:**
  + POST /api/v1/ai/{snapshotId}/ask: Stuurt een vraag naar de AI assistant voor een specifieke snapshot (Relateert aan FR-AI-01). Request body: { "question": "..." }. Response: { "answer": "...", "interactionId": "..." }.
  + POST /api/v1/ai/feedback: Stuurt gebruikersfeedback over een AI interactie (Relateert aan FR-AI-06, FR-AI-07). Request body: { "interactionId": "...", "rating": 1, "comment": "..." }.

**5.6 Error Handling**

* API-fouten worden geretourneerd met een passende HTTP status code (bijv. 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error).
* De response body bij een fout bevat een gestructureerd JSON-object met minimaal een errorCode (een interne, herkenbare code) en een errorMessage (een mens-leesbare beschrijving).

Voorbeeld:

{

"errorCode": "SNAPSHOT\_NOT\_FOUND",

"errorMessage": "Metadata snapshot with the specified ID does not exist or you don't have permission to access it."

}

**5.7 CORS (Cross-Origin Resource Sharing)**

* Supabase Edge Functions moeten correct geconfigureerd worden (via Access-Control-Allow-Origin headers, etc.) om requests van de gehoste frontend (Vercel/Netlify domein) toe te staan.

**6. Security Architectuur**

Deze sectie beschrijft de beveiligingsmaatregelen die worden geïmplementeerd om de vertrouwelijkheid, integriteit en beschikbaarheid van het NEXA platform en de data van gebruikers te waarborgen.

**6.1 Authenticatie**

* **Mechanisme:** Supabase Auth wordt gebruikt voor gebruikersauthenticatie. Ondersteuning voor e-mail/wachtwoord en mogelijk OAuth providers (Google, GitHub) voor registratie en login (FR-AUTH-01, FR-AUTH-02).
* **JWT:** Na succesvolle login geeft Supabase Auth een JSON Web Token (JWT) uit. Dit token wordt door de frontend opgeslagen (bijv. in localStorage of sessionStorage - afweging nodig m.b.t. XSS risico) en meegestuurd in de Authorization: Bearer <token> header bij elke API-request naar de Supabase Edge Functions.
* **Token Validatie:** Een Express middleware in de API-laag valideert de handtekening en geldigheid (vervaldatum) van het JWT bij elke beschermde request. De user\_id wordt uit het gevalideerde token geëxtraheerd voor autorisatiechecks.
* **Wachtwoordbeleid:** Supabase Auth standaardinstellingen voor wachtwoordcomplexiteit en eventuele MFA (Multi-Factor Authentication) worden overwogen.

**6.2 Autorisatie**

* **API Autorisatie:** De API-laag (Edge Functions) voert autorisatiechecks uit na authenticatie. Deze checks verifiëren of de geauthenticeerde gebruiker (user\_id uit JWT) de gevraagde actie mag uitvoeren op de specifieke resource (bijv. toegang tot een snapshotId of connectionId). Dit gebeurt op basis van lidmaatschap van organizations en eventuele rollen (organization\_members tabel).
* **Database Autorisatie (RLS):** Supabase Row Level Security (RLS) wordt geconfigureerd op kritieke tabellen (zoals salesforce\_connections, metadata\_snapshots, parsed\_components, analysis\_results). RLS policies zorgen ervoor dat database queries (ook die via de Supabase client library of RPC calls) alleen data retourneren die behoort tot de organisatie(s) van de geauthenticeerde gebruiker. Dit biedt een diepe verdedigingslaag.

**6.3 Data Security**

* **Encryptie in Transit:** Alle communicatie tussen de gebruiker's browser en de frontend hosting (Vercel/Netlify), tussen de frontend en de Supabase API/DB, en tussen Supabase Functions en externe services (Salesforce, Neo4j AuraDB, LLM) verloopt via HTTPS/TLS, waardoor data tijdens het transport versleuteld is.
* **Encryptie at Rest (Supabase):**
  + Supabase versleutelt data at rest standaard op infrastructuurniveau.
  + **Specifieke Kolom Encryptie:** Gevoelige data, met name access\_token en refresh\_token in de salesforce\_connections tabel, moeten *expliciet applicatief versleuteld* worden voordat ze naar de database worden geschreven. Dit kan met Supabase's pgsodium extensie of door encryptie/decryptie logica in de Edge Functions (met sleutels veilig beheerd als Supabase secrets). Dit zorgt ervoor dat zelfs bij een volledige database dump de tokens niet direct leesbaar zijn.
* **Encryptie at Rest (Neo4j AuraDB):** Neo4j AuraDB biedt standaard encryptie at rest voor de graph data.
* **Data Masking/Minimization:** Er wordt geen onnodig gevoelige Salesforce data opgeslagen. Focus ligt op metadata structuren. Er worden geen Salesforce record *data* opgeslagen.

**6.4 API Security**

* **Input Validatie:** Alle data die via API-requests binnenkomt (request bodies, query parameters, path parameters) wordt gevalideerd op type, formaat en toegestane waarden (bijv. met een library zoals zod of express-validator) om injection attacks en onverwachte fouten te voorkomen.
* **Rate Limiting:** Implementatie van rate limiting op API-endpoints (mogelijk via Supabase ingebouwde features of Express middleware) om misbruik en Denial-of-Service (DoS) aanvallen te beperken.
* **Security Headers:** Gebruik van standaard security headers (zoals Strict-Transport-Security, Content-Security-Policy, X-Content-Type-Options, X-Frame-Options) op zowel de frontend hosting als de API responses om browser-gebaseerde aanvallen (XSS, clickjacking) te mitigeren.
* **CORS:** Strikte configuratie van Cross-Origin Resource Sharing (CORS) headers om alleen requests van de toegestane frontend domeinen toe te staan.

**6.5 Salesforce Connectie Security**

* **OAuth 2.0 PKCE:** De Proof Key for Code Exchange (PKCE) extensie wordt gebruikt tijdens de OAuth flow. Dit is de best practice voor publieke clients (zoals een backend die namens een browser handelt) en voorkomt "authorization code interception" aanvallen.
* **Veilige Token Opslag:** Refresh en access tokens worden versleuteld opgeslagen in de Supabase database (zie 6.3). Tokens worden nooit direct naar de frontend gestuurd.
* **Scoped Permissions:** Tijdens de OAuth flow worden alleen de minimaal benodigde Salesforce permissies aangevraagd (bijv. toegang tot metadata API's, refresh token scope).

**6.6 Infrastructuur Security**

* **Supabase:** Vertrouwt op de beveiligingsmaatregelen van het Supabase platform (infrastructuur beveiligd door AWS/cloud provider, firewalling, etc.). Toegang tot het Supabase project dashboard wordt beperkt en beveiligd met MFA.
* **Neo4j AuraDB:** Vertrouwt op de beveiligingsmaatregelen van Neo4j's cloud service. Toegang tot de database wordt beveiligd met sterke credentials en netwerk policies. Database credentials worden veilig opgeslagen als secrets in de Supabase Functions environment.
* **LLM Provider:** API keys voor de LLM provider worden veilig opgeslagen als secrets in de Supabase Functions environment en nooit blootgesteld in de frontend of versiebeheer.

**6.7 Dependency Management**

* **Scanning:** Regelmatig scannen van project dependencies (NPM packages) op bekende kwetsbaarheden met tools zoals npm audit, Snyk, of GitHub Dependabot.
* **Updates:** Tijdig updaten van dependencies om bekende kwetsbaarheden te patchen.

**6.8 Logging en Monitoring**

* **Audit Logging:** Belangrijke security-gerelateerde events (succesvolle/mislukte logins, Salesforce connectie pogingen, wijzigingen in permissies, API fouten met status 4xx/5xx) worden gelogd (bijv. in een aparte Supabase tabel of via Supabase's ingebouwde logging).
* **Monitoring:** Monitoring op ongebruikelijke activiteiten of error rates kan helpen bij het detecteren van beveiligingsincidenten.

**7. Deployment Architectuur**

Deze sectie beschrijft de strategie en processen voor het bouwen, testen en deployen van de NEXA applicatie naar de verschillende omgevingen.

**7.1 Omgevingen**

Er worden minimaal drie omgevingen onderhouden:

1. **local (Ontwikkelomgeving):**
   * **Doel:** Individuele ontwikkeling en unit testing.
   * **Setup:** Draait lokaal op de machine van de ontwikkelaar.
   * **Componenten:**
     + Frontend (React): Draait lokaal via npm run dev (of Vercel CLI).
     + Supabase Stack: Draait lokaal via de supabase start CLI commando (inclusief Postgres DB, Auth, Storage, Edge Functions runtime).
     + Neo4j: Draait lokaal via Docker Compose (of verbindt met een gedeelde dev/test instantie op AuraDB).
     + Salesforce: Ontwikkelaars gebruiken hun eigen Salesforce Developer Edition orgs of Sandboxes voor testen. OAuth callback URL's moeten lokaal werken (bijv. via localhost of een tunnel zoals ngrok).
   * **Secrets:** Lokale .env bestanden voor API keys (niet in Git).
2. **staging (Test-/Acceptatieomgeving):**
   * **Doel:** Integratietesten, end-to-end testen, User Acceptance Testing (UAT) voordat features naar productie gaan. Stabiliteit waarborgen.
   * **Setup:** Gehost in de cloud, qua architectuur identiek aan productie.
   * **Componenten:**
     + Frontend (React): Gehost op Vercel/Netlify (bijv. staging.nexa-app.com).
     + Supabase Project: Een *apart* Supabase project specifiek voor staging.
     + Neo4j Instance: Een *aparte* Neo4j AuraDB instance (of namespace) voor staging.
     + Salesforce: Koppelingen naar Salesforce Sandboxes of test-orgs.
   * **Data:** Bevat testdata, eventueel periodiek gesynchroniseerd/geanonimiseerd vanuit productie of handmatig aangemaakt.
   * **Deployment:** Automatisch gedeployed vanuit een specifieke Git branch (bijv. develop of staging) via CI/CD.
3. **production (Productieomgeving):**
   * **Doel:** Live omgeving voor eindgebruikers.
   * **Setup:** Gehost in de cloud met focus op beschikbaarheid, performance en security.
   * **Componenten:**
     + Frontend (React): Gehost op Vercel/Netlify (bijv. app.nexa-app.com).
     + Supabase Project: Een *apart* Supabase project specifiek voor productie.
     + Neo4j Instance: Een *aparte*, geschaalde Neo4j AuraDB instance voor productie.
     + Salesforce: Koppelingen naar de productie Salesforce orgs van klanten.
   * **Data:** Live klantdata. Backups en monitoring zijn essentieel.
   * **Deployment:** Automatisch of handmatig (na goedkeuring) gedeployed vanuit de hoofd Git branch (bijv. main of master) via CI/CD.

**7.2 CI/CD Pipeline (Continuous Integration / Continuous Deployment)**

* **Tool:** GitHub Actions (aanbevolen) of GitLab CI.
* **Trigger:** Pipeline wordt getriggerd bij pushes naar specifieke branches (main, develop) of bij het aanmaken van Pull Requests.

**Pipeline Stappen (Voorbeeld voor push naar develop -> deploy naar staging):**

1. **Checkout Code:** Haalt de laatste code op uit de develop branch.
2. **Setup Environment:** Installeert Node.js, PNPM/NPM/Yarn, en eventuele andere benodigde tools.
3. **Install Dependencies:** Installeert project dependencies (pnpm install).
4. **Lint & Format Check:** Voert linters (ESLint) en formatters (Prettier) uit om codekwaliteit en consistentie te waarborgen. Mislukt bij fouten.
5. **Unit Tests:** Voert unit tests uit voor frontend componenten en backend logica (API functions). Mislukt bij falende tests.
6. **Build Frontend:** Bouwt de React applicatie voor productie (npm run build).
7. **Build/Bundle Functions:** Compileert TypeScript Edge Functions naar JavaScript (indien nodig) en bundelt ze.
8. **(Optioneel) Integration Tests:** Voert integratietests uit die de interactie tussen componenten testen (bijv. API calls naar een test-database).
9. **Deploy Supabase Migrations:** Past database schema wijzigingen (uit de supabase/migrations map) toe op de *staging* Supabase database via de Supabase CLI (vereist Supabase Access Token als secret).
10. **Deploy Supabase Edge Functions:** Deployt de gebouwde Edge Functions naar het *staging* Supabase project via de Supabase CLI.
11. **Deploy Frontend:** Deployt de gebouwde React applicatie naar Vercel/Netlify voor de *staging* omgeving (vereist Vercel/Netlify Access Token als secret).
12. **(Optioneel) End-to-End Tests:** Voert geautomatiseerde end-to-end tests uit op de gedeployde staging omgeving (bijv. met Cypress of Playwright).
13. **Notificatie:** Stuurt een notificatie (bijv. naar Slack) over het succes of falen van de deployment.

**Deployment naar Productie:** Volgt een vergelijkbaar proces, maar getriggerd door de main branch, en deployt naar de productie Supabase/Neo4j/Vercel omgevingen. Dit kan een handmatige goedkeuringsstap vereisen.

**7.3 Database Schema Beheer**

* **Supabase Migrations:** Schema wijzigingen voor de Supabase Postgres database worden beheerd via SQL-bestanden in de supabase/migrations map in de Git repository. De Supabase CLI wordt gebruikt om migraties lokaal te genereren en toe te passen, en de CI/CD pipeline past ze toe op staging/productie.
* **Neo4j Schema:** Indexen en constraints voor Neo4j worden beheerd via Cypher scripts. Deze kunnen handmatig worden toegepast of via een script dat wordt uitgevoerd als onderdeel van de deployment (bijv. via een aparte Edge Function of CI/CD stap).

**7.4 Secrets Management**

* API keys, database credentials, JWT secrets, en andere gevoelige configuratiewaardes worden **nooit** in de Git repository opgeslagen.
* **Lokaal:** Gebruik van .env bestanden (toegevoegd aan .gitignore).
* **Staging/Productie:** Gebruik van de secrets management features van de hosting platforms:
  + Supabase Edge Functions: Environment Variables geconfigureerd in het Supabase project dashboard.
  + Vercel/Netlify: Environment Variables geconfigureerd in de projectinstellingen.
  + CI/CD (GitHub Actions): Encrypted Secrets geconfigureerd in de repository instellingen.