**06 - Plan voor Ontwikkelomgeving & Tooling**

**Status:** DEFINITIEF (voor Fase 0)

**Inhoudsopgave:**

1. [Code Repository](#1-code-repository)
2. [Lokale Ontwikkelomgeving](#2-lokale-ontwikkelomgeving)
3. [Kerntechnologieën (Herhaling)](#3-kerntechnologieen-herhaling)
4. [Code Editor & Extensies](#4-code-editor--extensies)
5. [Code Kwaliteit & Consistentie](#5-code-kwaliteit--consistentie)
6. [Testing Frameworks](#6-testing-frameworks)
7. [CI/CD](#7-cicd)
8. [Project Management](#8-project-management)

**1. Code Repository**

* **Versiebeheer:** Git.
* **Hosting:** GitHub (of GitLab/Bitbucket, nader te bepalen - GitHub aanbevolen vanwege integraties zoals Actions).
* **Repository Structuur:** Monorepo wordt aanbevolen, beheerd met PNPM workspaces (of alternatief NPM/Yarn workspaces). Dit houdt frontend en backend code bij elkaar, wat dependency management en cross-component wijzigingen vereenvoudigt.

/nexa-platform/

├── apps/

│ ├── frontend/ # React app

│ └── supabase/ # Supabase project (migrations, functions)

├── packages/ # Gedeelde code/types (optioneel)

├── docs/ # Documentatie (zoals dit document)

├── pnpm-workspace.yaml

├── package.json

* **Branching Strategie:** Gitflow (of een variant zoals GitHub Flow) wordt aanbevolen:
  + main: Productie code, stabiele releases.
  + develop: Integratiebranch voor volgende release, basis voor staging deployments.
  + feature/xxx: Branches voor nieuwe features, afgetakt van develop. Worden gemerged naar develop via Pull Requests (PRs).
  + bugfix/xxx: Branches voor fixes op develop.
  + hotfix/xxx: Branches voor urgente fixes op main, worden gemerged naar main én develop.
* **Pull Requests:** Alle merges naar main en develop gaan via Pull Requests, die minimaal één review vereisen en succesvolle CI checks (linting, tests).

**2. Lokale Ontwikkelomgeving**

* **Vereiste Software:**
  + Node.js (LTS versie, zie TAD)
  + PNPM (Package Manager, zie TAD)
  + Git
  + Docker Desktop (voor lokaal draaien Neo4j, indien niet direct met cloud dev instance verbonden)
  + Supabase CLI (npm install -g supabase)
* **Setup Stappen (per ontwikkelaar):**

1. Clone de Git repository.
2. Installeer project dependencies (pnpm install in de root).
3. Zet lokale Supabase stack op:
4. Login bij Supabase CLI (supabase login).
5. Start de lokale Supabase services (supabase start vanuit apps/supabase). Dit start Postgres, GoTrue (Auth), Storage, en de Functions runtime.
6. Pas initiële migraties toe (supabase db reset of supabase migration up).
7. Zet lokale Neo4j op (indien nodig):
8. Start Neo4j container via docker-compose up -d (vereist docker-compose.yml bestand).
9. Configureer environment variabelen: Maak .env bestanden aan in apps/frontend en apps/supabase/functions op basis van .env.example bestanden (die wel in Git staan). Vul hier lokale database URLs, Supabase anon key, service role key, Neo4j credentials, etc. in.
10. Start Frontend dev server (pnpm --filter frontend dev).
11. Start Supabase functions lokaal (gebeurt via supabase start).

**3. Kerntechnologieën (Herhaling)**

* **Frontend:** React, TypeScript, Tailwind CSS, Zustand/RTK, React Router, TanStack Query, Cytoscape.js.
* **Backend (API):** Node.js, TypeScript, Express.js, Supabase JS Client, Neo4j JS Driver, JSforce.
* **Database:** PostgreSQL (via Supabase), Neo4j (via AuraDB voor staging/prod).

**4. Code Editor & Extensies**

* **Editor:** VS Code (sterk aanbevolen).
* **Essentiële Extensies:**
  + ESLint
  + Prettier - Code formatter
  + Tailwind CSS IntelliSense
  + Docker
  + Supabase (Officiële VS Code extensie)
  + GitLens — Git supercharged
  + Debugger for Node.js / Deno
  + Import Cost (optioneel, voor frontend bundle size)
  + YAML (voor OpenAPI specs)
  + Markdown All in One (voor documentatie)

**5. Code Kwaliteit & Consistentie**

* **Linter:** ESLint met aanbevolen configuraties voor TypeScript, React, Node.js. Zorgt voor het afdwingen van codeerstijl en het vinden van potentiële bugs.
* **Formatter:** Prettier met een gedeelde configuratie (.prettierrc). Zorgt voor consistente code opmaak.
* **Git Hooks:** Gebruik van husky en lint-staged om automatisch linting en formatting uit te voeren vóór elke commit, zodat slechte code de repository niet bereikt.

**6. Testing Frameworks**

* **Unit Tests:** Vitest (aanbevolen vanwege snelheid en moderne API, compatibel met Vite voor frontend) of Jest. Voor het testen van individuele functies en componenten in isolatie. React Testing Library voor frontend component tests.
* **Integration Tests:** Vitest/Jest kunnen ook gebruikt worden voor tests die de interactie tussen meerdere modules binnen de backend testen (bijv. API route -> service -> DB call).
* **End-to-End (E2E) Tests:** Playwright (aanbevolen vanwege snelheid en betrouwbaarheid) of Cypress. Voor het testen van volledige user flows via de browser op een draaiende (staging) omgeving.

**7. CI/CD**

* **Platform:** GitHub Actions (zie TAD Sectie 7.2 voor pipeline details).
* **Configuratie:** Workflow bestanden (.github/workflows/\*.yaml) in de repository definiëren de build, test en deploy stappen voor verschillende branches/events.

**8. Project Management**

* **Issue Tracking:** GitHub Issues (eenvoudig, geïntegreerd) of een externe tool zoals Jira, Linear, of Trello. Keuze afhankelijk van teamgrootte en procesvoorkeur.
* **Communicatie:** Slack of Microsoft Teams voor dagelijkse communicatie.
* **Documentatie:** Google Drive (voor deze planningsdocumenten) en de docs/ map in de repository (voor technische/API documentatie).