Serengo

Projectdocumentatie

Zias van Nes

Toegepaste Informatica, Odisee Hogeschool, Brussel

2025-12-26

Samenvatting

Serengo is een locatie-gebaseerde sociale webapplicatie waarmee gebruikers zogenaamde ‘finds’ (plaatsen en momenten) op een kaart kunnen vastleggen, verrijken met media en delen met een vertrouwde kring. Deze paper beschrijft de doelstelling en doelgroep van het project, de onderliggende data- en software-architectuur, de gebruikte tools en ontwikkelworkflow, de deploymentaanpak en de belangrijkste optimalisaties die tijdens de ontwikkeling zijn uitgevoerd.

Inhoudstafel

1. Wat is Serengo?

Serengo is een locatie-gebaseerde sociale webapplicatie, gebouwd met SvelteKit [(Svelte Core Team, 2025)], waarin gebruikers zogenaamde 'finds' kunnen vastleggen en delen. Een find is een combinatie van een plaats op de kaart, een kort verhaal en bijhorende media (foto's en video's). Gebruikers kunnen zo hun favoriete plekken documenteren, waarderen en met anderen delen.

Het project combineert interactieve kaarten, moderne webtechnologie en sociale functionaliteit tot één samenhorende ervaring:

- **Kaart-centrische interface** – de startpagina toont een fullscreen kaart (MapLibre GL JS [(MapLibre Community, 2025)]) waarop finds als markers verschijnen. Via een zijbalk kunnen gebruikers finds verkennen, filteren en selecteren.
- **Finds met rijke media** – bij elke find horen een titel, beschrijving, locatie en één of meerdere media-items (beelden en video's). Media worden opgeslagen in Cloudflare R2 [(Cloudflare, Inc., 2025)] en veilig ontsloten via signed URLs en een lokale mediaproxy.
- **Sociale interacties** – gebruikers kunnen finds liken, commenten en beoordelen (ratings). Vriendschapsrelaties bepalen wie welke finds mag zien: publiek, alleen vrienden of enkel de eigenaar.
- **Realtime en gebruiksvriendelijke updates** – door een sync-service en een centrale `api-sync` store worden wijzigingen eerst optimistisch in de UI toegepast en daarna met de database gesynchroniseerd. Bij fouten volgt automatisch een rollback.
- **Notificaties en PWA** – via een service worker en Web Push [(Barnes & Others, 2016)] worden gebruikers op de hoogte gebracht van gebeurtenissen zoals likes, comments en vriendschapsverzoeken. De applicatie is ingericht als Progressive Web App met caching en offline-ondersteuning voor kernfunctionaliteit.

In essentie is Serengo een digitale kaart vol persoonlijke verhalen. Het biedt de technische infrastructuur om plaatsen vast te leggen, te verrijken met context en die ervaringen gecontroleerd te (her)delen binnen een sociale omgeving.

2. Voor wie is Serengo?

Serengo is ontworpen voor kleine tot middelgrote groepen gebruikers die waarde hechten aan het gezamenlijk ontdekken, bewaren en herbeleven van interessante plaatsen. In plaats van een volledig open, anoniem social media-platform, richt Serengo zich op contexten waar onderling vertrouwen en controle over zichtbaarheid belangrijk zijn.

2.1. Primaire doelgroep

- **Vriendengroepen en kennissenkringen** die samen een kaart willen opbouwen van favoriete cafés, restaurants, uitzichtpunten, wandelroutes of andere "hidden gems".
- **Lokale communities** (bijvoorbeeld studentenverenigingen, hobbygroepen of buurtinitiatieven) die interessante locaties in hun omgeving willen cureren en onderling delen.
- **Reizigers en explorers** die hun mooiste ontdekkingen op reis willen vastleggen, documenteren met media en nadien eenvoudig terugvinden op een kaart.

2.2. Secundaire doelgroep

- **Individuele gebruikers** die een persoonlijke kaart van favoriete plaatsen willen beheren zonder dit publiek op een grote social media-site te delen.

- **Early adopters en ontwikkelaars** die geïnteresseerd zijn in moderne webtechnologie (SvelteKit, Drizzle ORM, Web Push, MapLibre) en een concreet voorbeeldproject zoeken.

2.3. Behoeften en use-cases

De belangrijkste behoeften die Serengo adresseert zijn:

- **Gecontroleerde zichtbaarheid** – via het vriendensysteem en privacyfilters (All/Public/Friends/Mine) kan de eigenaar per find bepalen wie wat te zien krijgt.
- **Rijke context rond locaties** – naast coördinaten en naam biedt Serengo ruimte voor verhalen, foto's, video's, likes, comments en ratings.
- **Gezamenlijk geheugen** – meerdere gebruikers kunnen finds aan dezelfde locatie koppelen zodat er een gedeelde geschiedenis per plaats ontstaat.
- **Eenvoudig delen** – via deelbare URL's en de Web Share API kunnen finds snel met anderen worden gedeeld.

2.4. Link met functionaliteit

De ontwerpkeuzes in de applicatie sluiten direct aan bij deze doelgroep:

- Het **vriendensysteem** en de **privacy-bewuste filtering** maken het veilig om persoonlijke plaatsen te delen binnen een beperkte kring.
- **Ratings en comments** maken het mogelijk om samen te reflecteren op plaatsen en aanbevelingen te doen.
- De **fullscreen kaartinterface** met slimme centrering en clustering ondersteunt het ontdekken van nieuwe locaties op een intuïtieve, visuele manier.

3. ERD

In dit hoofdstuk wordt het Entity-Relationship Diagram (ERD) van de PostgreSQL-database toegelicht. Het ERD is automatisch gegenereerd op basis van het Drizzle-schema (`src/lib/server/db/schema.ts`) en is als `erd.svg` toegevoegd aan de repository.

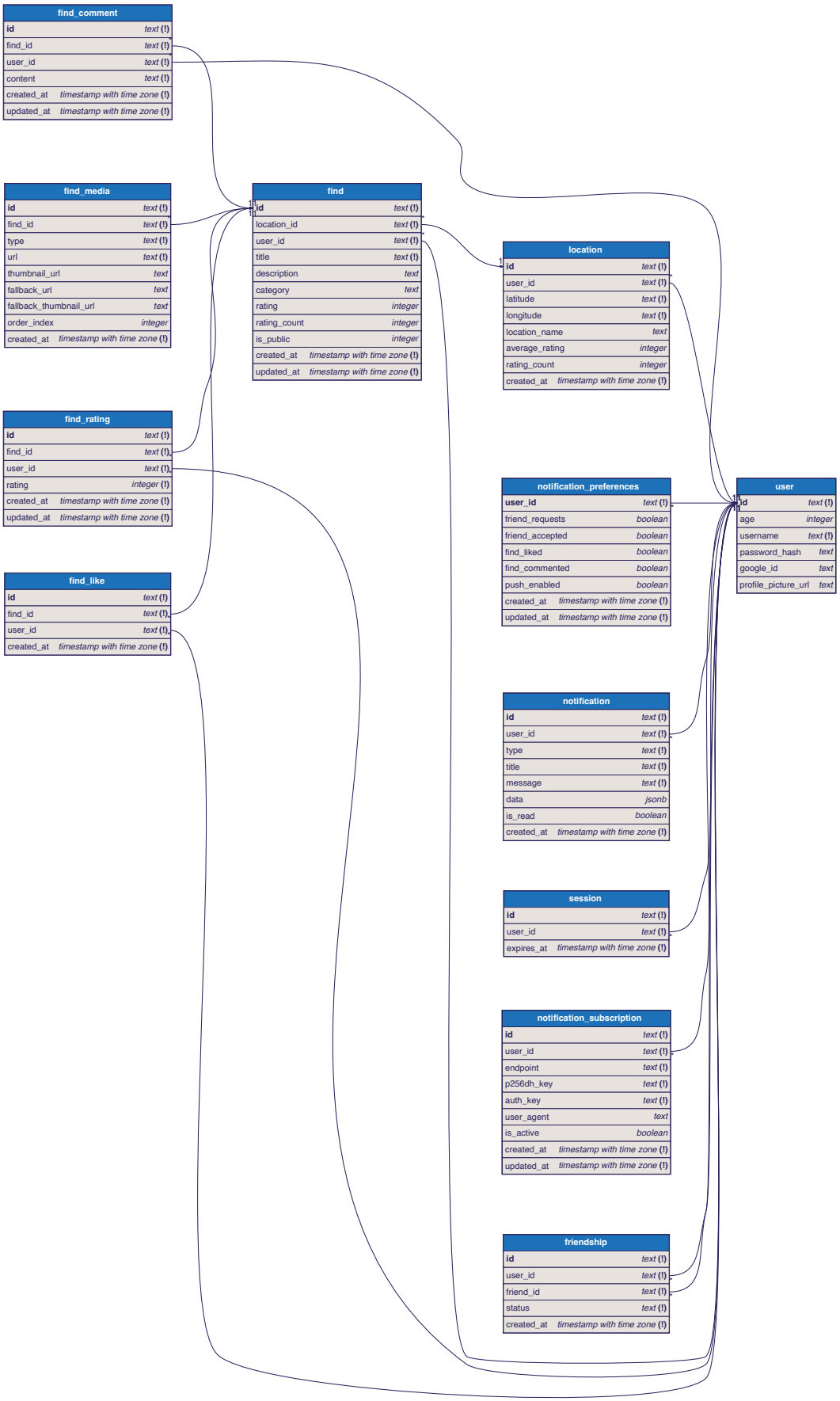

Figure 1: Globale ERD van de Serengo-database

De database is ontworpen rond een aantal kernentiteiten die samen het sociale, locatie-gebaseerde karakter van Serengo ondersteunen.

3.1. Kernentiteiten

De belangrijkste tabellen zijn:

- **users** – bevat gebruikersaccounts, inclusief referenties naar authenticatiegegevens (Lucia) [(Lucia Contributors, 2025)] en profielinformatie.
- **sessions** – beheert actieve sessies voor gebruikers.
- **oauth_accounts** – slaat externe OAuth-accounts op (zoals Google) die aan een user gekoppeld zijn.
- **locations** – beschrijft gedeelde locaties met o.a. coördinaten, naam en type.
- **finds** – posts die aan een locatie gekoppeld zijn, met titel, beschrijving, privacy-instelling en metadata.
- **media** (impliciet in het schema afhankelijk van de definitie) – verwijzingen naar geüploade afbeeldingen en video's in Cloudflare R2.
- **likes** – registreren welke gebruiker welke find leuk vindt.
- **comments** – reacties van gebruikers bij finds.
- **friendships** – vriendschapsrelaties en hun status (bijv. pending, accepted, rejected).
- **ratings** – numerieke beoordelingen (1–5 sterren) die gebruikers aan finds geven.
- **notification_subscriptions** – Web Push-abonnementen (endpoint + keys) per gebruiker/apparaat.
- **notification_preferences** – instellingen per gebruiker voor welke notificaties gewenst zijn.

3.2. Belangrijkste relaties

Een aantal relaties zijn cruciaal voor het gedrag van de applicatie:

- Eén **user** heeft veel **finds**; elke find hoort bij precies één user (de eigenaar).
- Eén **location** heeft veel **finds**; meerdere users kunnen finds aan dezelfde locatie koppelen.
- Eén **find** heeft veel **likes**, **comments** en **ratings**.
- Eén **user** heeft veel **likes**, **comments** en **ratings** verspreid over verschillende finds.
- **friendships** modelleren wederzijdse relaties tussen twee users; hieruit volgt welke finds zichtbaar zijn bij privacyfilters.
- **notification_subscriptions** en **notification_preferences** zijn aan users gekoppeld en bepalen hoe en wanneer pushnotificaties verstuurd worden.

Samen vormen deze relaties het fundament voor de sociale interacties rond locaties: het ERD maakt zichtbaar hoe gebruikers, locaties, finds en interacties logisch met elkaar verbonden zijn.

3.3. Normalisatie en evolutie van het datamodel

Tijdens de ontwikkeling is het datamodel geëvolueerd van een eenvoudiger structuur naar een meer genormaliseerde, location-gecentreerde architectuur (zie ook het hoofdstuk Architectuur). Belangrijke stappen hierin waren:

- Het verplaatsen van locatienamen en -informatie uit de **finds**-tabel naar een aparte **locations**-tabel, zodat meerdere finds dezelfde locatie kunnen delen.
- Het toevoegen van aparte tabellen voor **comments**, **likes** en **ratings**, in plaats van alles in één "finds"-structuur te stoppen.
- Het uitbreiden van notificatie- en privacygerelateerde tabellen om Web Push en friend-based zichtbaarheid te ondersteunen.

Deze normalisatie heeft als effect dat:

- data-integriteit verbeterd is (minder duplicatie, duidelijke foreign keys);
- queries efficiënter en expressiever worden (bijv. alle finds op een locatie, alle activiteiten van een user);
- nieuwe features (zoals locatiepopulariteit of uitgebreidere statistieken) eenvoudiger in te passen zijn op basis van het bestaande schema.

4. Klassediagram

In dit hoofdstuk wordt een logisch klassediagram gepresenteerd dat de belangrijkste domeinmodellen (typen) van Serengo samenvat. Hoewel de implementatie in TypeScript + Drizzle ORM gebeurt in plaats van klassieke OOP-klassen, kunnen we de structuur voorstellen als een verzameling klassen met velden en relaties. Dit klassediagram sluit nauw aan op het ERD, maar benadrukt vooral de verantwoordelijkheid per type in de applicatielogica.

4.1. Overzicht van domeinmodellen

De kernmodellen zijn:

- **User**
  - Velden: id, naam, e-mailadres, avatar/profielfoto, aanmaakdatum, OAuth-gegevens.
  - Verantwoordelijkheid: vertegenwoordigt een eindgebruiker; koppelt naar sessies, find-activiteit en sociale relaties.
- **Location**
  - Velden: id, naam, beschrijving (optioneel), coördinaten (latitude, longitude), type/categorie.
  - Verantwoordelijkheid: beschrijft een fysieke plaats op de kaart waarop meerdere finds kunnen worden geprojecteerd.
- **Find**
  - Velden: id, eigenaar (user-id), location-id, titel, beschrijving, privacy-instelling, timestamps, verwijzingen naar media.
  - Verantwoordelijkheid: vormt de kern-“post” in de applicatie: een verhaal + media gekoppeld aan een locatie.
- **MediaItem**
  - Velden: id, find-id, type (afbeelding/video), url (naar R2), fallback-url, metadata (bijv. formaat, grootte).
  - Verantwoordelijkheid: encapsuleert één mediabestand; zorgt voor scheiding tussen inhoud (find) en opslag (R2).
- **Comment**
  - Velden: id, find-id, user-id, inhoud, aanmaakdatum.
  - Verantwoordelijkheid: laat gebruikers reageren op finds; gekoppeld aan de API-sync-laag voor realtime updates.
- **Like**
  - Velden: id, find-id, user-id, timestamp.
  - Verantwoordelijkheid: registreert een “vind-ik-leuk”-actie op een find.
- **Rating**
  - Velden: id, find-id, user-id, waarde (1–5), timestamp.

  - Verantwoordelijkheid: legt een expliciete beoordeling van een find vast, los van likes.
- **Friendship**
  - Velden: id, requester-id, addressee-id, status (pending, accepted, rejected), timestamps.
  - Verantwoordelijkheid: modelleert de vriendrelatie en vormt de basis voor privacy-bewuste filtering.
- **Notification** (logisch model bovenop meerdere DB-tabellen)
  - Velden: id, user-id, type (like, comment, friend-request, …), payload, status (gelezen/ongelezen).
  - Verantwoordelijkheid: representeert een gebeurtenis die aan de gebruiker wordt gecommuniceerd; gebruikt samen met subscription- en preference-modellen voor Web Push.

4.2. Relatie met frontend en stores

In de frontend worden deze modellen gebruikt in Svelte-componenten en Svelte stores:

- In `src/lib/stores/api-sync.ts` wordt een typesafe representatie van finds, comments, likes en ratings bijgehouden.
- Components in `src/lib/components/finds/` en `src/lib/components/map/` ontvangen deze modellen via props en renderen ze in de UI.
- Door overal TypeScript-typen te gebruiken, blijven de klassediagram-concepten consistent tussen database, serverlogica en frontend.

Het klassediagram helpt zo om de logische structuur van de applicatie in één oogopslag te begrijpen en vormt een brug tussen het relationele ERD en de concrete implementatie in TypeScript en Svelte.

5. Architectuur van het project

De architectuur van Serengo is opgebouwd rond SvelteKit als full-stack framework, Drizzle ORM voor de PostgreSQL-database en een aantal gespecialiseerde services voor storage, kaarten en notificaties. De frontend en backend leven in dezelfde codebase en delen type-informatie, zodat de volledige stack strongly typed is.

5.1. Opbouw van de applicatie

Op hoog niveau bestaat de applicatie uit de volgende lagen:

- **Presentatielaag (UI)** – Svelte 5-componenten in `src/lib/components` en pagina's in `src/routes`. Componenten zijn per domein gegroepeerd (auth, finds, map, media, notifications, profile, …).
- **Domein- en servicelaag** – Svelte stores en hulpfuncties in `src/lib/stores` en `src/lib/utils`, plus server-side services in `src/lib/server` (auth, db, media-processor, push, R2, oauth).
- **API-laag** – SvelteKit-endpoints in `src/routes/api/*` die CRUD- en acties aanbieden voor finds, friends, users, notifications, profile pictures, places en media.
- **Datalaag** – een PostgreSQL-database aangestuurd via Drizzle ORM. De schema-definitie staat in `src/lib/server/db/schema.ts`, migraties in de map `drizzle/`.

Deze lagen zijn zodanig opgebouwd dat UI-componenten alleen via duidelijke interfaces praten met stores en services, en dat alle persistente data via de Drizzle-laag loopt.

5.2. Projectstructuur

De belangrijkste mappen van het project zijn: .

- `src/lib/components/` – herbruikbare UI-componenten, gegroepeerd per domein:
  - `auth/` – login-form en gerelateerde auth-componenten.
  - `finds/` – componenten rond finds (overzichten, detailweergave, editmodals, comment-UI, …).
  - `map/` – kaartcomponenten (Map, LocationManager, POI-zoekfunctionaliteit).
  - `media/` – video- en mediacomponenten.
  - `notifications/` – notificatie-UI en beheer.
  - `profile/` – profielpaneel en profielfoto's.
  - plus een set shadcn-achtige UI-primitieven [(Delaney & Contributors, 2025)] (button, card, dropdown-menu, sheet, skeleton, sonner, …).
- `src/lib/server/` – server-side logica:
  - `db/` – databaseconfiguratie en Drizzle-schema.
  - `auth.ts` – Lucia-authenticatie. [(Lucia Contributors, 2025)]
  - `oauth.ts` – Google OAuth-integratie.
  - `push.ts` – Web Push-notificaties.
  - `r2.ts` – integratie met Cloudflare R2.
  - `media-processor.ts` – beeld- en videobewerking (o.a. WebP/JPEG-pipeline).
- `src/lib/stores/` – Svelte stores:
  - `api-sync.ts` – centrale sync-service voor optimistic updates.
  - `location.ts` – tracking van de huidige gebruikerslocatie.
- `src/lib/utils/` – hulpfuncties, o.a. `geolocation.ts` en `places.ts` (Google Places API).
- `src/routes/` – pagina's en API-endpoints:
  - `+page.svelte` – homepage met de kaart en finds.
  - `finds/`, `friends/`, `login/`, … – feature-specifieke pagina's.
  - `api/` – submappen voor finds, friends, users, notifications, profile-picture, places, media, …
- `drizzle/` – migratiebestanden en metadata.
- `static/` – statische assets (fonts, map-styles, afbeeldingen, manifest, robots.txt, …).
- `scripts/` – hulpscripts (zoals het genereren van VAPID-keys).

5.3. Location-gecentreerde architectuur

Eén van de belangrijkste architecturale keuzes is de overstap naar een **location-gecentreerd architectuurmodel**. In plaats van elke find zijn eigen, dupliceerende locatiedata te laten opslaan, worden locaties in een aparte `locations`-tabel beheerd. Meerdere finds kunnen aan dezelfde locatie gekoppeld worden. Dit heeft meerdere voordelen:

- **Data-normalisatie** – locatiegegevens (coördinaten, naam, type) worden op één plaats beheerd.
- **Betere performance** – queries kunnen efficiënter filteren en groeperen op locatie.
- **Functionaliteit per locatie** – het wordt eenvoudiger om alle finds op één plek te combineren en hier extra functionaliteit rond te bouwen (bijv. populariteitsmetrieken in de toekomst).

De grote logic overhaul (Phase 6–7 in het logboek) beschrijft hoe de bestaande finds-architectuur werd omgevormd naar dit model, inclusief migraties en aanpassingen op zowel API als frontend.

5.4. Sync-service en api-sync store

Om de gebruikerservaring soepel te houden, maakt Serengo gebruik van een **sync-service** en een centrale `api-sync` store. In plaats van bij elke wijziging te wachten op een serverrespons, wordt het volgende patroon gebruikt:

1. De gebruiker voert een actie uit (bijvoorbeeld een find aanpassen of een comment toevoegen).
2. De wijziging wordt onmiddellijk lokaal toegepast (optimistic update) zodat de UI instant reageert.
3. De sync-service stuurt de wijziging naar de API.
4. Bij succes wordt de lokale staat bevestigd; bij een fout wordt de wijziging teruggedraaid (rollback) en ziet de gebruiker een duidelijke foutmelding.

Deze architectuur zorgt voor een responsieve interface, zelfs bij netwerkvertraging, en centraliseert state management voor finds, comments, likes en ratings.

In de toekomst zou ik hier eventueel WebSockets aan kunnen toevoegen voor real-time updates tussen gebruikers. Waarschijnlijk zou het ook interessant zijn om hiervoor te kijken naar een externe service. Ik kijk hierbij naar Convex [(Convex Team, 2025)] die real-time sync en offline-first functionaliteit biedt zodat dit niet allemaal zelf gebouwd en onderhouden hoeft te worden.

5.5. Integratie met externe diensten

De architectuur integreert verschillende externe diensten op een consistente manier:

- **Cloudflare R2** voor opslag van media, benaderd via de `r2.ts`-service en beveiligde signed URLs.
- **Google OAuth** voor authenticatie, afgehandeld via `oauth.ts` in combinatie met Lucia [(Lucia Contributors, 2025)].
- **Google Places API** voor POI-zoekfunctionaliteit, gebruikt door de map- en locatiecomponenten.
- **Web Push** voor notificaties, met VAPID-keys en een service worker die push-events verwerkt.

Elke integratie is ondergebracht in een eigen module of service, zodat de invloed op de rest van het systeem beperkt en overzichtelijk blijft.

6. Buildtools en ontwikkelcommando's

Serengo maakt gebruik van een moderne JavaScript-toolchain gebaseerd op SvelteKit en Vite, aangevuld met TypeScript, ESLint/Prettier en Drizzle voor databasebeheer [(Svelte Core Team, 2025); (You & Contributors, 2025); (Drizzle Team, 2025)]. In dit hoofdstuk worden de belangrijkste tools en commando's samengevat.

6.1. Bundling en framework

- **Framework**: SvelteKit 2 met Svelte 5-componenten en runes (`$props`, `$derived`, `$effect`).
- **Bundler/dev-server**: Vite 7, geconfigureerd in `vite.config.ts`.
- **Adapter**: `@sveltejs/adapter-node` voor een Node.js-productieserver (`svelte.config.js`).

De bundling gebeurt via Vite: tijdens ontwikkeling draait een snelle dev-server met hot module replacement; voor productie wordt een geoptimaliseerde bundel gebouwd met code-splitting en tree-shaking.

6.2. Ontwikkelcommando's (pnpm scripts)

De belangrijkste commando's (zie `package.json`) zijn:

- `pnpm run dev`
  - Start de Vite dev-server met SvelteKit in ontwikkelmodus.
  - Handig tijdens actieve ontwikkeling; herlaadt automatisch bij codewijzigingen.
- `pnpm run build`
  - Maakt een productiebuild van de applicatie.
  - Gebruikt Vite + SvelteKit om geoptimaliseerde JavaScript en CSS te genereren.
- `pnpm run preview`
  - Start een lokale server bovenop de productiebuild.
  - Wordt gebruikt om de uiteindelijke build te testen voor deployment.
- `pnpm run check`
  - Voert `svelte-kit sync` en vervolgens `svelte-check` uit met de `tsconfig.json`.
  - Controleert TypeScript-typen en Svelte-componenten op fouten.
- `pnpm run lint`
  - Voert eerst `prettier --check .` en daarna `eslint .` uit.
  - Zorgt voor consistente code-stijl en detecteert mogelijke probleemgevallen.
- `pnpm run format`
  - Draait Prettier in schrijfmodus (`--write`) over het project.
  - Past de afgesproken formatteringsregels (tabs, single quotes, max 100 chars) toe.

6.3. Databasetools (Drizzle ORM)

Voor het beheer van de PostgreSQL-database wordt Drizzle ORM gebruikt, met bijhorende CLI-commando's:

- `pnpm run db:start`
  - Start de PostgreSQL-database via `docker-compose` (zie `docker-compose.yml`).
- `pnpm run db:generate`
  - Genereert migratiebestanden op basis van wijzigingen in het Drizzle-schema (`src/lib/server/db/schema.ts`).
- `pnpm run db:migrate`
  - Voert de gegenereerde migraties uit op de database.
  - Wordt zowel lokaal als in de Docker-deployment gebruikt (entrypoint script).
- `pnpm run db:push`
  - Pusht de huidige schema-definitie naar de database (handig in vroege ontwikkelfase).
- `pnpm run db:studio`
  - Start Drizzle Studio voor het visueel inspecteren van de database.
- `pnpm run db:generate-erd`
  - Gebruikt `drizzle-erd` om op basis van het schema een ERD (`erd.svg`) te genereren.

6.4. ESLint, Prettier en typechecking

- **ESLint**: bewaakt codekwaliteit en dwingt consistente patterns af, met ondersteuning voor Svelte (`eslint-plugin-svelte`) en moderne JavaScript/TypeScript-regels.

- **Prettier**: zorgt voor automatische formattering met de in `AGENTS.md` vastgelegde stijl.
- **TypeScript + svelte-check**: zorgen voor statische typeveiligheid doorheen de hele stack.

Samen vormen deze tools een sterke ontwikkelbasis: fouten worden vroegtijdig opgespoord, de code blijft leesbaar en het buildproces is voorspelbaar en reproduceerbaar.

7. Deployment stappen

Serengo ondersteunt meerdere deployment-scenario's, waaronder Vercel en een self-hosted Docker-omgeving [(Docker, Inc., 2025)]. In dit hoofdstuk worden de belangrijkste concepten, vereiste configuratie en een concreet stappenplan beschreven.

7.1. Vereiste environment-variabelen

Voor een werkende productieomgeving zijn minimaal de volgende variabelen nodig (zie ook `logs/logboek.md`):

```
DATABASE_URL=           # PostgreSQL-verbinding
GOOGLE_CLIENT_ID=       # Google OAuth client ID
GOOGLE_CLIENT_SECRET=   # Google OAuth secret
R2_ACCOUNT_ID=          # Cloudflare R2 account ID
R2_ACCESS_KEY_ID=       # Cloudflare R2 access key
R2_SECRET_ACCESS_KEY=   # Cloudflare R2 secret key
R2_BUCKET_NAME=         # Cloudflare R2 bucket-naam
VAPID_PUBLIC_KEY=       # Web Push public key
VAPID_PRIVATE_KEY=      # Web Push private key
GOOGLE_MAPS_API_KEY=    # Google Places API key
```

Deze worden tijdens build en runtime ingelezen door de SvelteKit-app en de server-side services (`auth`, `r2`, `push`, `places`).

7.2. Vercel-deployment (cloud)

In een Vercel-scenario wordt de app gedeployed met behulp van `@sveltejs/adapter-vercel` (aanwezig als dependency). Belangrijke aandachtspunten:

- De SvelteKit-app wordt als serverless of edge-functies uitgerold, afhankelijk van de Vercel-configuratie.
- `DATABASE_URL` moet verwijzen naar een publiek bereikbare PostgreSQL-instantie (bijv. een managed database).
- Cloudflare R2 en Web Push blijven extern; de nodige secrets worden als Vercel environment-variabelen ingesteld.

7.3. Self-hosted Docker-deployment

Voor een zelfgehoste omgeving is een uitgebreide Docker-setup voorzien (Phase 7 in het logboek):

- Multi-stage Docker build om een compacte productie-image te maken.
- Health checks om de beschikbaarheid van de container te monitoren.
- Een entrypoint-script dat bij het starten van de container automatisch Drizzle-migraties uitvoert.
- Drizzle als production dependency, zodat migraties ook in de container kunnen draaien.

Typische stappen voor een self-hosted deployment zijn:

1. Zorg voor een draaiende PostgreSQL-instantie (lokaal, via `docker-compose` of extern) en vul de `DATABASE_URL` correct in.
2. Stel alle vereiste environment-variabelen in voor de app (zie boven).
3. Bouw de Docker-image, bijvoorbeeld met:

   ```
   docker build -t serengo:latest .
   ```

4. Start de container met de juiste environment-variabelen en netwerkconfiguratie, bijvoorbeeld via `docker-compose`:

   ```
   docker-compose up -d
   ```

5. Het entrypoint-script in `docker-entrypoint.sh` zorgt ervoor dat `pnpm run db:migrate` (of een equivalent via `drizzle-kit`) uitgevoerd wordt bij het opstarten, zodat de database-schema's up-to-date zijn.
6. Controleer de logs en health checks om na te gaan of de app correct draait.

7.4. Van ontwikkeling naar productie

Samengevat ziet de route van lokale ontwikkeling naar productie er als volgt uit:

1. Lokaal ontwikkelen met `pnpm run dev`, frequent linten en typen checken (`pnpm run lint`, `pnpm run check`).
2. Database- en schemawijzigingen doorvoeren met Drizzle (`db:generate`, `db:migrate`, `db:generate-erd`).
3. De productiebuild genereren met `pnpm run build` en lokaal testen via `pnpm run preview`.
4. Deployen naar Vercel of bouwen en uitrollen van de Docker-image in een self-hosted omgeving.

Door de geautomatiseerde migraties en consistente configuratie is het deploymentproces herhaalbaar en betrouwbaar.

7.5. Aanvullende documentatie en hosting

Naast deze paper is er een aparte API-referentie beschikbaar, opgebouwd met Mintlify en gehost op mijn eigen domein:

- **API-documentatie**: `https://docs.zias.be` – Mintlify-documentatie [(Mintlify, Inc., 2025)] voor de Serengo-API.

De volledige broncode van Serengo wordt self-hosted op mijn eigen Git-server:

- **Broncode-repository**: `https://git.zias.be/zias/serengo` – privé Git-instantie met de volledige Serengo broncode.

De productieversie van de Serengo-applicatie draait op mijn eigen Hetzner VPS [(Hetzner Online GmbH, 2025)], uitgerold met Docker [(Docker, Inc., 2025)]:

- **Live-applicatie**: `https://serengo.zias.be` – SvelteKit-app in Docker-container op een Hetzner VPS.

Deze deployment zorgt ervoor dat zowel de code als de infrastructuur volledig in eigen beheer zijn, zonder afhankelijkheid van publieke Git-hosting of PaaS-platformen.

8. Optimalisaties en resultaten

Tijdens de ontwikkeling van Serengo zijn meerdere optimalisatiegolven doorgevoerd op het gebied van performance, UX, architectuur en deployment. Dit hoofdstuk vat de belangrijkste stappen en hun impact samen.

8.1. Performance- en SEO-optimalisaties

Op 7 oktober is een grote optimalisatieronde uitgevoerd gericht op SEO, PWA en laadtijden:

- **SEO-verbeteringen**
  - Toevoegen en bijwerken van meta-tags en Open Graph-data.
  - Automatische generatie van `sitemap.xml` voor betere indexeerbaarheid.
- **PWA-verbeteringen**
  - Uitbreiding van de service worker voor caching van statische assets en kernroutes.
  - Optimalisatie van `manifest.json` (iconen, naamgeving, start-URL).
- **Performance**
  - Compressie van de achtergrondafbeelding (± 50% kleiner; van 4.2MB naar 2.1MB).
  - Vite-configoptimalisaties en fixen van build-issues.

Gemeten resultaten (op basis van Lighthouse-logs):

- Homepageload verbeterd van ongeveer 2.5s naar 1.2s.
- Largest Contentful Paint gedaald van ±1.8s naar ±0.9s.
- Betere stabiliteit door caching en service worker-ondersteuning.

8.2. Architecturale verbeteringen

In Phase 6 en 7 is de interne architectuur grondig herzien:

- **Overgang naar location-gecentreerd datamodel**
  - Scheiding tussen `locations` en `finds`, met meerdere finds per locatie.
  - Normalisatie van locatiedata (minder duplicatie, betere querymogelijkheden).
  - Transactieve migraties en data-integriteitscontroles tijdens de overgang.
- **Sync-service en api-sync-laag**
  - Implementatie van een centrale sync-service (`api-sync.ts`) voor optimistic updates.
  - Automatische rollback bij falende API-calls, waardoor inconsistenties worden voorkomen.
  - Vermindering van dubbele logica in componenten en eenduidige bron van waarheid voor find-, comment-, like- en ratingdata.

Deze ingrepen verhogen zowel de schaalbaarheid (efficiëntere queries, minder redundantie) als de gebruikerservaring (snellere, soepelere UI).

8.3. UX- en mapoptimalisaties

Een belangrijke focus lag op de kaart- en navigatie-ervaring:

- **Fullscreen map + sidebar**
  - Introductie van een fullscreen kaart met een uitschuifbare zijbalk voor finds.
  - Oplossing van overscroll- en overflowproblemen in lijsten.
- **Dynamische mapcentrering**
  - Intelligente centrering van de kaart afhankelijk van sidebar-positie (desktop vs. mobiel).
  - Voorkomen van storende autozoom tijdens het volgen van locaties.

- **Marker- en layoutverbeteringen**
  - Toevoegen van location markers, clustering en betere positionering.
  - UI-finetuning voor FindPreview, CommentsList en andere componenten.

Resultaat: gebruikers zien hun find altijd in het zichtbare deel van de kaart, en navigeren vloeiend tussen locaties en finds.

8.4. Mediaperformance en veiligheid

Om zowel performance als security te verbeteren, zijn verschillende stappen gezet rond media:

- Invoering van een lokale mediaproxy (`/api/media/[...path]`) in plaats van directe Cloudflare R2-URL's.
- CSP-fixes om externe resources gecontroleerd toe te laten.
- WebP/JPEG-pipeline voor efficiëntere afbeeldingen met fallback-ondersteuning.

Dit zorgt voor snellere laadtijden, minder kans op CSP-gerelateerde fouten en betere controle over de media-aflevering.

8.5. Deployment- en operationele optimalisaties

Om deployments robuust en reproduceerbaar te maken zijn onder meer de volgende verbeteringen geïntroduceerd:

- Multi-stage Docker build voor kleinere, efficiëntere images.
- Verplaatsing van Drizzle naar production dependencies zodat migraties binnen de container kunnen draaien.
- Een entrypoint-script dat bij het starten van de container automatisch database-migraties uitvoert.
- Health checks en verbeterde logging voor snellere foutdetectie.

Samen maken deze optimalisaties Serengo geschikt voor langdurige, stabiele hosting met goede gebruikerservaring en onderhoudbaarheid.

9. Screenshots en gebruiksscenario's

In dit hoofdstuk worden enkele representatieve screenshots van de applicatie opgenomen. Om het Typst-document compileerbaar te houden, zijn er placeholder-afbeeldingen voorzien in `docs/screenshots/`. Vervang deze placeholders later door echte screenshots (bijvoorbeeld PNG) en pas de bestandsnamen/paden hieronder aan.

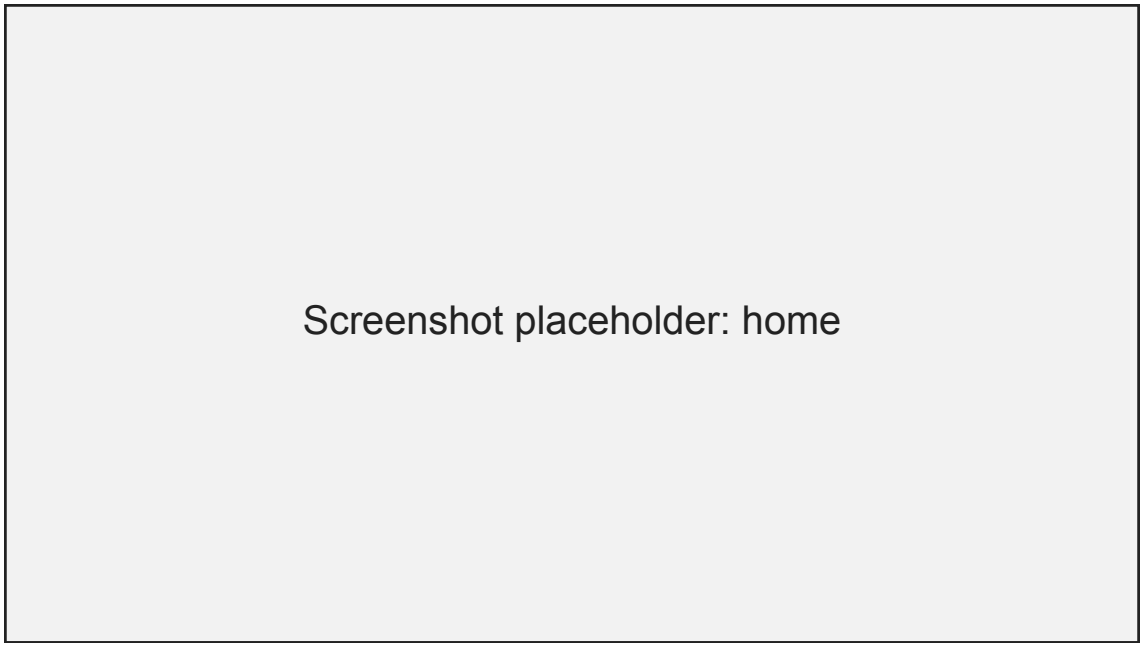

Figure 2: Startscherm met fullscreen kaart en zijbalk met finds

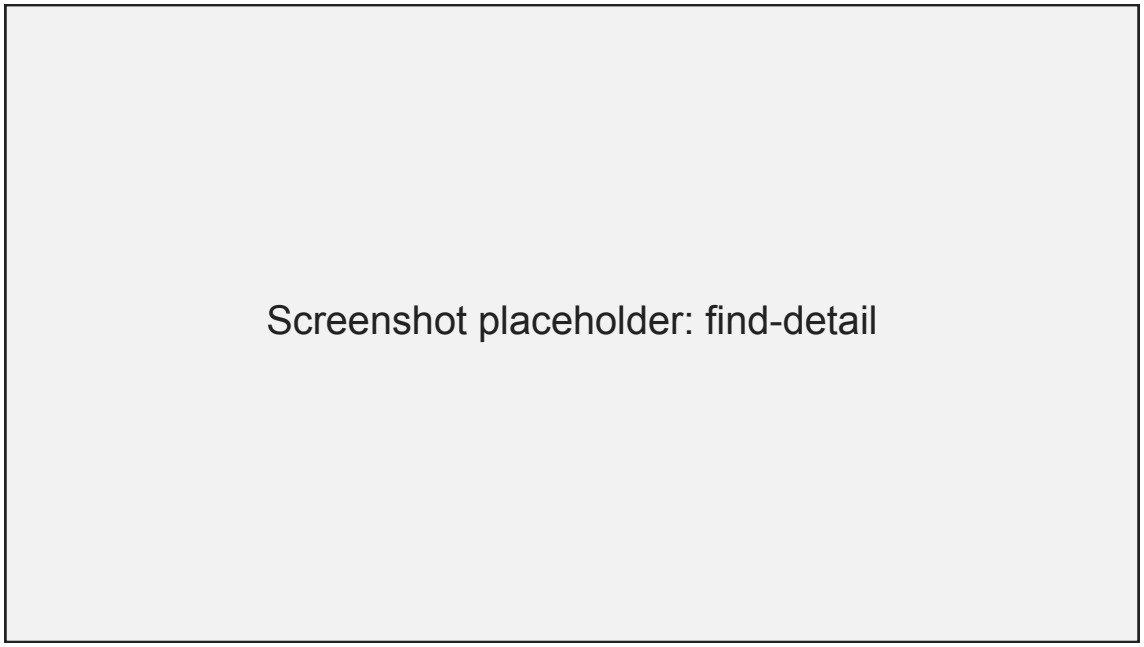

Figure 3: Detailpagina van een find met media, likes, comments en rating

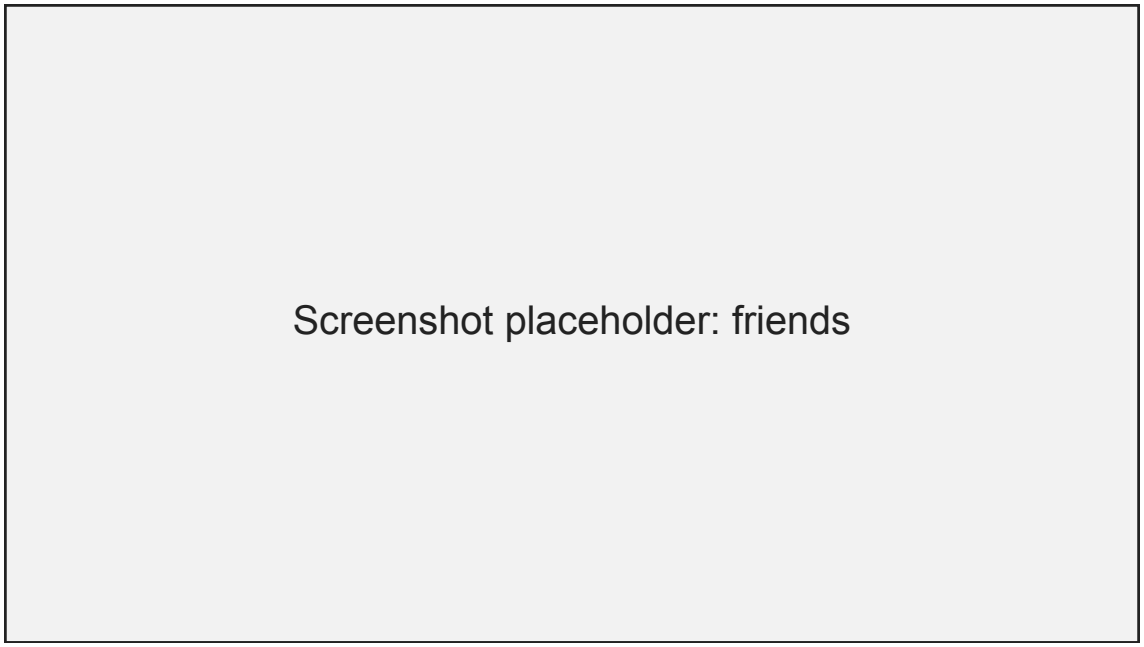

Figure 4: Vriendenoverzicht en privacy-bewuste filtering van finds

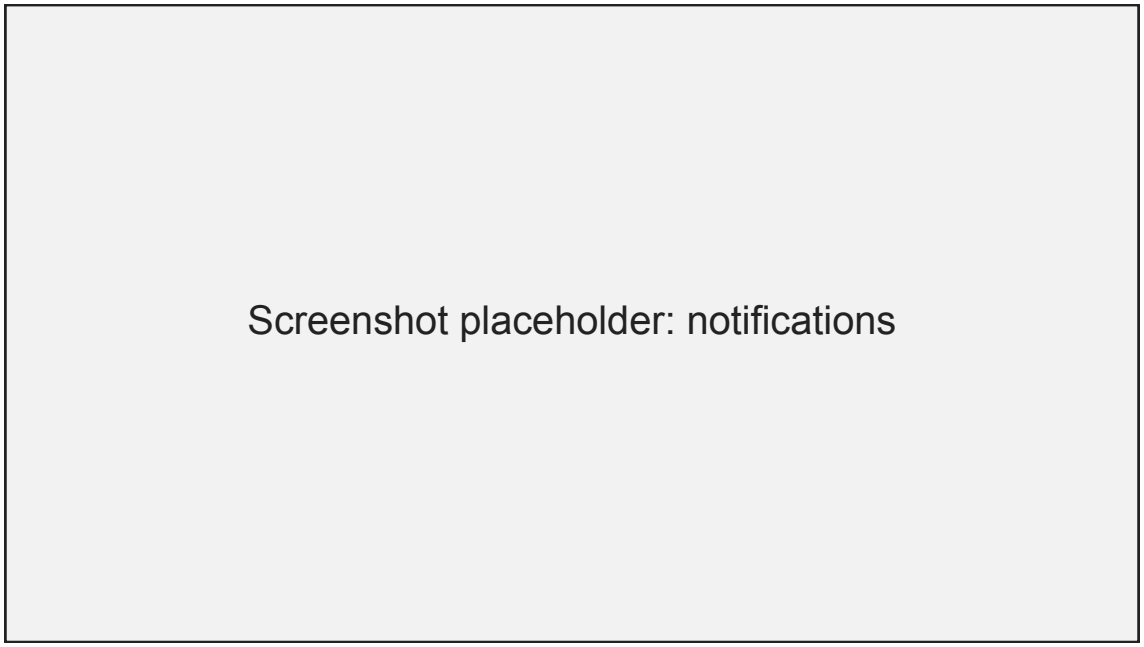

Figure 5: In-app notificaties en Web Push-permissies

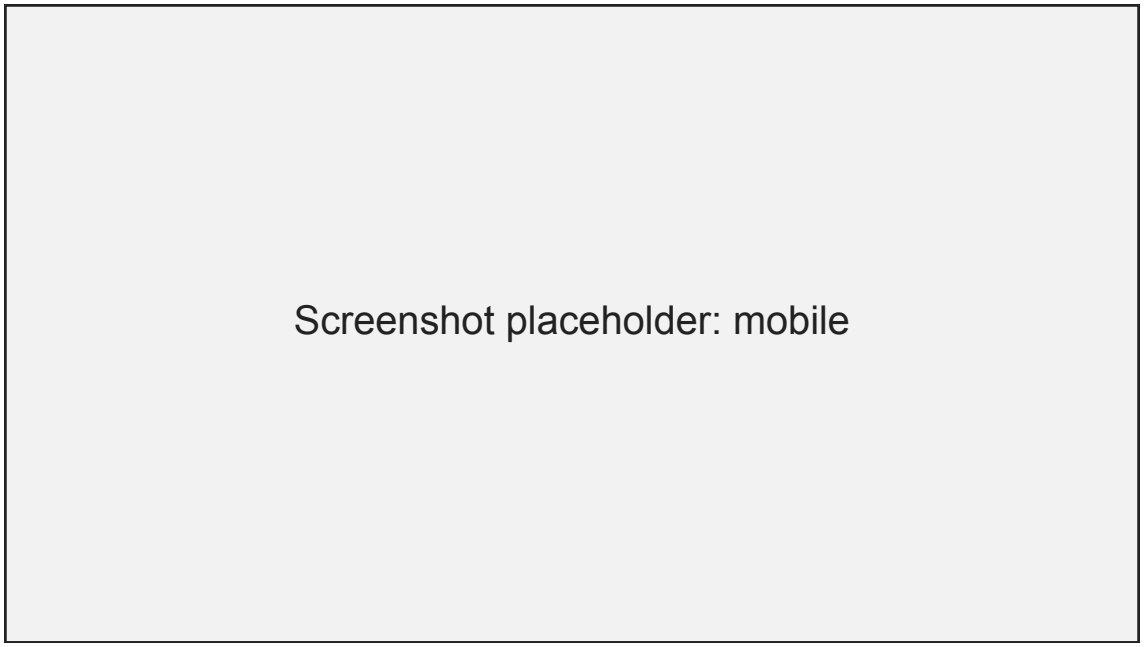

Figure 6: Mobiele weergave met zijbalk en dynamische mapcentrering

Referenties

Barnes, M., & Others. (2016,). *The Web Push Protocol*. https://datatracker.ietf.org/doc/html/rfc8030

Cloudflare, Inc. (2025,). *Cloudflare R2 Object Storage Documentation*. https://developers.cloudflare.com/r2/

Convex Team. (2025,). *Convex Documentation*. https://convex.dev/docs

Delaney, h., & Contributors. (2025,). *shadcn-svelte: Re-usable Components*. https://www.shadcn-svelte.com/

Docker, Inc. (2025,). *Docker Documentation*. https://docs.docker.com/

Drizzle Team. (2025,). *Drizzle ORM Documentation*. https://orm.drizzle.team/docs

Hetzner Online GmbH. (2025,). *Hetzner Cloud Documentation*. https://docs.hetzner.com/cloud/

Lucia Contributors. (2025,). *Lucia Auth Documentation*. https://lucia-auth.com/

MapLibre Community. (2025,). *MapLibre GL JS Documentation*. https://maplibre.org/maplibre-gl-js/docs/

Mintlify, Inc. (2025,). *Mintlify Documentation Platform*. https://mintlify.com/

Svelte Core Team. (2025,). *SvelteKit Documentation*. https://kit.svelte.dev/docs

You, E., & Contributors, V. (2025,). *Vite – Next Generation Frontend Tooling*. https://vitejs.dev/