ChatGPT

Ontwikkelplan voor een moderne rekentool voor kinderopvangorganisaties

Inleiding

Dit document beschrijft een uitgebreid ontwikkelplan voor een **moderne rekentool** gericht op kinderopvangorganisaties. Deze tool zal ouders helpen om op basis van ingevoerde keuzes (per kind, opvangvorm, aantal dagen/uren, vast maandbedrag, etc.) te berekenen wat hun opvangkosten zijn. Tegelijk biedt het kinderopvangorganisaties een beheeromgeving om tarieven en opvangvormen in te stellen, en een **superuser**-omgeving voor support om alle organisaties te kunnen ondersteunen. De applicatie wordt opgebouwd met een **React** frontend (gebruikmakend van Chakra UI componenten) en een **Node.js** backend. Er is aandacht voor **multitenancy** (meerdere organisaties met afzonderlijke inlog en gescheiden data), embedding van de tool op externe websites en meertalige ondersteuning (Nederlands/Engels).

Belangrijkste functionaliteiten

- **Beheeromgeving voor organisaties:** elke kinderopvangorganisatie heeft een eigen login en kan binnen een afgeschermde omgeving eigen tarieven, opvangvormen (KDV, BSO, gastouder, etc.) en variaties instellen.
- **Superuser (support) omgeving:** een centrale beheerder kan als **support** inloggen en heeft toegang tot de data/configuratie van alle aangesloten organisaties (voor ondersteuning en monitoring).
- **Publieke rekenmodule voor ouders:** een voor ouders toegankelijke frontend waarmee per kind de kosten berekend kunnen worden op basis van gekozen opvangvorm, aantal dagen per week, uren per maand, eventuele vaste maandbedragen en andere relevante instellingen. Deze rekenmodule gebruikt de tarieven en instellingen van de betreffende organisatie.
- **Embedden op websites:** de rekenmodule moet eenvoudig **integreerbaar** zijn op de websites van kinderopvangorganisaties, bijvoorbeeld via een embed-code of widget. Hierdoor kunnen ouders de berekeningstool direct op de site van hun opvang bekijken.
- **Meertalige ondersteuning:** de interface zal beschikbaar zijn in **Nederlands en Engels**, met mogelijkheid om eenvoudig uit te breiden naar extra talen. Dit betekent dat zowel de oudermodule als de beheeromgeving meertalige tekstlabels en content ondersteunen.

Architectuur en technologieën

Frontend: We kiezen voor een frontend in **React** met **Chakra UI** als componentenbibliotheek. Chakra UI biedt een uitgebreide set herbruikbare React-componenten die het bouwen van gebruiksvriendelijke interfaces versnellen [1] [2]. Ontwikkelaars kunnen hiermee moderne, responsive interfaces creëren zonder vanaf nul alle componenten te schrijven, dankzij kant-en-klare elementen (knoppen, formulieren, navigatie, e.d.) die eenvoudig aanpasbaar en themable zijn. Bovendien voldoet Chakra UI aan WAI-ARIA standaarden, wat de toegankelijkheid ten goede komt.

Backend: De backend wordt gebouwd in **Node.js** (JavaScript runtime) met een framework als **Express** voor het opzetten van een RESTful API. Node.js is uitermate geschikt voor dit soort applicaties en kan ook op gangbare shared hosting-platformen draaien [3]. Veel moderne hostingproviders (met

bijvoorbeeld cPanel) bieden inmiddels ondersteuning voor Node.js applicaties, waarbij je via een GUI een Node-versie kunt kiezen en de applicatie kunt draaien [4]. De Node backend zal de logica bevatten voor authenticatie, business rules (bijv. de berekening van kosten), en datatoegang tot de database.

Communicatie & Hosting: De React frontend en Node backend communiceren via REST API calls (JSON). Dit maakt het mogelijk de frontend als single-page app te hosten (bijv. op een CDN of in dezelfde hostingomgeving als statische bestanden) en de backend als afzonderlijke service/API. Voor shared hosting kan de Node-app op dezelfde server draaien als waar de organisatie hun website host, mits die host Node ondersteunt. Anders kan gekozen worden voor een kleine cloud-VPS of Platform-as-a-Service die Node.js draait, terwijl de React-app statisch gehost wordt.

Chakra UI integratie: In de React-app wordt Chakra UI geïnitieerd via de ChakraProvider (in de root van de applicatie) om thema's en stijlen toe te passen. Chakra UI biedt een consistente design basis en handige utility props voor spacing, kleuren, etc., wat de ontwikkelsnelheid ten goede komt. Hierdoor kan de focus liggen op functionaliteit in plaats van op uitgebreide CSS-styling.

Overige libraries: Voor formulieren en state management in React kan gebruik worden gemaakt van bijv. React Hook Form of Formik (voor het beheren van formulierstatus en validatie) en eventueel React Context of Redux voor globale state (hoewel voor een relatief beperkte tool Context in combinatie met React hooks wellicht volstaat). Voor de backend kunnen hulpmiddelen als **jsonwebtoken** (JWT) of **express-session** worden ingezet voor sessiebeheer, en **bcrypt** voor veilige wachtwoordhashing. Verder wordt voor meertaligheid een i18n library ingezet (zie meertalige ondersteuning).

Databasekeuze en dataopslag

Database-oplossing: We adviseren het gebruik van een **relationele database** zoals **MySQL/MariaDB** of **PostgreSQL** in combinatie met Node.js. Dergelijke databases zijn breed ondersteund, draaien vaak al op gedeelde hosting omgevingen, en bieden betrouwbare dataopslag met transactieondersteuning. Er zijn ook ORM/ODM-oplossingen beschikbaar (bv. **Sequelize**, **Prisma** of **TypeORM** voor SQL databases, of **Mongoose** voor MongoDB) die integratie met Node vereenvoudigen. Gezien de multitenancy-eis (meerdere organisaties met gescheiden data) is een relationele database met gestructureerde tabellen per organisatie een robuuste keuze. Alternatief kan een NoSQL database als **MongoDB** gebruikt worden – Node.js werkt hier naadloos mee samen en data wordt in JSON-vorm opgeslagen – maar voor het gestructureerde karakter van tarieven en instellingen is relationeel zeer geschikt.

Multitenancy (gescheiden data): In een multi-tenant applicatie moet data van elke organisatie strikt gescheiden blijven van anderen [5]. We ontwerpen de database daarom zodanig dat elke relevante record is gekoppeld aan een **organisatie-ID** (tenant identifier). Deze aanpak – een *tenant discriminator* veld in gedeelde tabellen – is een veelgebruikte methode om meerdere klanten in één database te bedienen [6]. Concreet betekent dit dat we bijvoorbeeld tabellen/collecties hebben voor `Organisaties`, `Gebruikers`, `Tarieven`, `Opvangvormen`, etc., waarbij in de tarieven- en opvangvormen-tabellen een veld `organisatie_id` opgenomen is. Queries vanuit een organisatie-account filteren altijd op deze id, zodat een klant alleen zijn eigen data ziet. Dit model is relatief eenvoudig op te zetten en goed schaalbaar naar vele tenants, hoewel de ultieme isolatie wat lager is dan bij fysiek gescheiden databases.

> **Alternatieve data-segregatie:** Ter overweging: er bestaan meerdere benaderingen voor multitenant data-segregatie. Een optie is een **aparte database per organisatie**, of per organisatie een apart schema binnen dezelfde database [7]. Deze bieden maximale isolatie, maar brengen beheeroverhead mee (meerdere databases/schemas up-to-date

houden, migraties synchroniseren, etc.). Gezien het verwachte aantal tenants en het MVP-karakter kiezen we vooralsnog voor één database met logisch gescheiden data via organisatie-ID's. Deze aanpak minimaliseert complexiteit en is voldoende veilig mits er in de applicatielaag strict op tenant-id wordt gefilterd.

Database-structuur: We voorzien minimaal de volgende entiteiten:
- **Organisatie**: gegevens van de opvangorganisatie (naam, contactinfo, etc.) + instellingen zoals standaard openingstijden indien relevant.
- **Gebruiker**: inlogaccounts (gebruikersnaam/email, wachtwoord-hash, rol type) gekoppeld aan een organisatie. Hier onderscheiden we rollen zoals *organisatiebeheerder* en *superuser*.
- **Opvangvorm**: een type opvang binnen een organisatie (bv. *Kinderdagverblijf (KDV)*, *Buitenschoolse Opvang (BSO)*, *Gastouder*). Organisatiebeheerders kunnen deze aanmaken/configureren.
- **Tarieven/Varianten**: tarieven en eventuele variaties of pakketten behorend bij opvangvormen. Bijv. uurtarieven, dagtarieven of vaste maandbedragen per opvangvorm, mogelijk gedifferentieerd naar leeftijdsgroepen of aantal dagen. Deze records bevatten verwijzing naar de opvangvorm en de organisatie.

De Node.js backend zal via een ORM/Query-builder deze database aanspreken. Bijvoorbeeld bij PostgreSQL/MySQL zou **Prisma** of **Knex** gebruikt kunnen worden om typesafe queries te schrijven. Voor MongoDB zou **Mongoose** een optie zijn. De keuze van ORM is flexibel; belangrijk is dat het framework ondersteuning biedt voor de multi-tenant query filtering (dit kan ook handmatig worden afgedwongen door in elke query de `organisatie_id` mee te nemen).

Migratie & versiebeheer: Vanaf de start zetten we tooling in om databasewijzigingen beheersbaar door te voeren. Denk aan migratiescripts (bijv. met Knex migrations of Prisma migrate voor relationele DB, of handmatige migration scripts voor Mongo). Zo kan per MVP-stap nieuwe tabellen of velden worden toegevoegd zonder dat bestaande data/structuur breekt. Voor een MVP kan men dit eenvoudig houden, maar gezien de verwachte groei is het goed dit proces van meet af aan te structureren.

Gebruikersbeheer en rollen

Een degelijk gebruikersbeheer is essentieel om de verschillende typen gebruikers (organisaties vs. support vs. ouders) toegang te geven met de juiste privileges. We onderscheiden drie rollen/gebruikerstypen:

Rol	Beschrijving	Rechten
Organisatiebeheerder	Account van een kinderopvangorganisatie. Meestal één of enkele accounts per organisatie. Deze gebruiker logt in op de beheeromgeving van zijn eigen organisatie.	- Eigen organisatieprofiel beheren (naam, gegevens) - **Tarieven en opvangvormen beheren**: toevoegen/wijzigen van KDV, BSO, gastouder opties en bijbehorende tarieven/varianten - Eventueel inzage in gebruiksstatistieken van de eigen rekentool (optioneel voor later) - **Geen toegang tot** data van andere organisaties.

Rol	Beschrijving	Rechten
Superuser (Support)	Centrale beheerder(s) van de gehele applicatie, bijv. voor supportdoeleinden.	- **Toegang tot alle organisaties**: kan gegevens inzien en indien nodig aanpassen voor support - Beheer van organisaties (aanmaken nieuwe organisatieaccounts, resetten wachtwoorden organisatiebeheerder, etc.) - Monitoring van gebruik en logs.
Ouder (eindgebruiker)	Bezoeker/gebruiker van de rekentool via de website van een organisatie. Hoeft geen account aan te maken; gebruikt de tool anoniem.	- **Kosten berekenen** op basis van invoer (kindgegevens, opvangwensen) voor de gekozen organisatie. - **Geen toegang tot beheerfunctionaliteit** of gegevens van anderen.

Authenticatie: Organisatiebeheerders en superusers moeten inloggen om bij hun respectievelijke omgevingen te komen. We implementeren een login met gebruikersnaam/e-mail + wachtwoord. Wachtwoorden worden gehasht opgeslagen (bijv. met bcrypt). Bij succesvolle login genereert de backend een sessie of **JWT (JSON Web Token)**. Een JWT-benadering is handig in een SPA-context: de token wordt opgeslagen (liefst HttpOnly cookie of in memory) en meegestuurd bij vervolgaanvragen. In de token payload staan o.a. de `user_id`, `organisatie_id` (tenzij superuser) en rol. De backend valideert de token bij elke request en bepaalt zo toegang. Alternatief is het gebruik van server-side sessies met een sessie-cookie, maar JWT past goed bij een stateless API design.

Autorisatie & data-isolatie: Op basis van de rol en `organisatie_id` in de token wordt bepaald welke acties de gebruiker mag doen. Een organisatiebeheerder mag enkel resources van zijn eigen `organisatie_id` opvragen of muteren; de API endpoints zullen dit controleren (bijv. als een organisatiebeheerder probeert een tarief van een andere organisatie op te vragen, geeft de API niets terug of een 403 Forbidden). De superuser-rol kan een speciale route hebben om de context van een organisatie te kiezen (of endpoints die alle data teruggeven). Intern zou een superuser bijvoorbeeld een query zonder `organisatie_id` filter mogen uitvoeren of expliciet voor een bepaalde org.

Beheer van organisaties en gebruikers: We voorzien een mogelijkheid voor de superuser om nieuwe organisaties aan te maken. Dit omvat het creëren van een organisatiesrecord en een eerste gebruiker (beheerder) voor die organisatie. De superuser kan ook accounts van organisatiebeheerders beheren (activeren, deactiveren, wachtwoord resetten). Optioneel kan self-service registratie van organisaties in de toekomst worden toegevoegd, maar initieel houden we dit centraal via de superuser om kwaliteit en consistente inrichting te bewaken.

Logging en monitoring: In het kader van support is het nuttig dat de superuser algemene logbestanden of activiteiten kan inzien, om bij problemen (bijvoorbeeld een organisatie meldt een fout in de berekening) te kunnen achterhalen wat er is gebeurd. Dit kan door server-logs toegankelijk te maken of een eenvoudige auditlog van belangrijke wijzigingen (bijv. tarief aangepast door gebruiker X op tijdstip Y).

Ontwikkelstappen (MVP-fases)

Hieronder wordt het ontwikkeltraject opgedeeld in logische stappen. Elke stap levert een werkend **Minimal Viable Product (MVP)** op dat een subset van de uiteindelijke functionaliteit bevat, zodat er tussentijds getest en bijgestuurd kan worden. Per stap benoemen we wat er wordt ontwikkeld, de gebruikte technologieën, hoe te testen, en de oplevercriteria (wanneer de stap “af” is).

Stap 1: Basisopzet van de applicatie en authenticatie

- **Ontwikkeling:** In deze eerste fase richten we de projectstructuur in voor zowel frontend als backend. We initialiseren een React project (met Chakra UI) en een Node.js/Express project. Vervolgens ontwikkelen we een eenvoudige **login-functionaliteit** voor organisatiebeheerders (en eventueel alvast voor de superuser). Dit omvat het aanmaken van een **Gebruiker** model/tabel in de database, een registratie of seeding van een eerste gebruiker, en een inlogpagina in React. Na inloggen ziet de gebruiker een eenvoudige placeholder “dashboard” pagina. Hiermee leggen we de fundering voor beveiligde routes.
- **Technologieën:** Create React App of Vite voor het frontend scaffold; Chakra UI integratie voor snelle layout van het login-formulier (gebruik van Chakra `<Input>`, `<Button>` componenten etc.). Node.js met Express voor de backend API. We gebruiken een databaseverbinding via bijv. Knex/Objection (SQL) of Mongoose (als we MongoDB hadden gekozen) om de gebruikers in de DB te beheren. Verder integreren we bcrypt voor het hashen van wachtwoorden en JSON Web Tokens (via bijv. **jsonwebtoken** npm package) of Express-session voor het beheren van sessies.
- **Test:** Na deze stap testen we handmatig (of via unit tests) de login-flow. Bijvoorbeeld: probeer in te loggen met een juiste en onjuiste combinatie om te verifiëren dat authenticatie goed werkt en fouten netjes worden afgehandeld (onjuiste login geeft melding). Unit tests kunnen geschreven worden voor de password hash/verify functionaliteit en de API endpoint voor login (met tools als Jest/Supertest voor de backend). Daarnaast controleren we of na login de beschermde React-route (dashboard) alleen toegankelijk is als je ingelogd bent (bijv. door JWT in localStorage/cookie en een simpele auth guard in React).
- **Oplevercriteria:** Een werkende skeleton applicatie: gebruikers kunnen zich aanmelden op een loginpagina en worden bij correcte credentials doorgelaten naar een afgeschermde pagina. De frontend build draait lokaal (of op een testserver) en kan communiceren met de backend API. Beveiliging is in basis op orde (hashed wachtwoorden in DB, tokens/sessies bij login). Hoewel er inhoudelijk nog weinig te beheren is, staat de infrastructuur voor beveiligde toegang nu.

Stap 2: Beheeromgeving – Organisatiegegevens en opvangvormen/tarieven configureren

- **Ontwikkeling:** In deze stap wordt de daadwerkelijke **beheeromgeving** voor een organisatie ingevuld. Na het inloggen moet de organisatiebeheerder zijn eigen gegevens en tarieven kunnen beheren. We ontwikkelen schermen en API-endpoints voor: (a) **Organisatie-instellingen** (bv. naam, adres, contact, eventueel logo upload); (b) CRUD-functionaliteit voor **opvangvormen** (aanmaken van categorieën zoals KDV, BSO, gastouder, incl. benaming en eventueel omschrijving per opvangvorm); (c) CRUD-functionaliteit voor **tarieven/varianten** – per opvangvorm kan de beheerder één of meerdere tarieven instellen. Bijvoorbeeld een uurtarief of dagtarief, en varianten zoals hele dag vs halve dag, of verschillende pakketten (zoals 52-weken opvang vs 40-weken opvang) met elk hun eigen prijs. De precieze invulling hangt af van de behoefte, maar de structuur laat meerdere tarieven per opvangvorm toe. Alle via de React UI ingevoerde gegevens worden via API naar de Node backend gestuurd en in de database opgeslagen gekoppeld aan de organisatie.

- **Technologieën:** In React bouwen we formulieren voor het beheren van opvangvormen en tarieven. Chakra UI biedt hiervoor form-components (zoals `<FormControl>`, `<Input>`, `<Select>`, `<NumberInput>` etc.) die we inzetten voor consistente vormgeving. We gebruiken React state of a state management library om tijdelijk ingevoerde data te beheren voordat het opgeslagen wordt. Op de backend breiden we het datamodel uit: tabellen/collecties voor `Opvangvorm` en `Tarief` inclusief de relatie naar `Organisatie`. In Node/Express definiëren we routes zoals `POST /api/opvangvormen`, `GET/PUT/DELETE /api/opvangvormen/:id` en soortgelijk voor tarieven. We implementeren server-side validatie (bijv. verplichte velden, numerieke waarden voor tarieven) naast client-side validatie voor goede UX.
- **Test:** We voeren uitgebreide tests uit door in te loggen als organisatiebeheerder en verschillende scenario's te doorlopen: opvangvorm toevoegen (bv. "KDV") met een bepaald tarief, wijziging van een tarief, verwijderen van een opvangvorm etc. We controleren in de database of de gegevens correct worden opgeslagen en of ze alleen zichtbaar zijn voor de juiste organisatie. Belangrijk is ook test van validaties: probeer ongeldige inputs (lege verplichte velden, negatieve getallen voor tarieven) en kijk of de UI en API dit correct afvangen. Waar mogelijk schrijven we unit tests voor de backend logica (bijv. een functie die berekent of een bepaald tarief binnen grenzen valt) en integratietests voor de API endpoints. Eventueel kunnen we met een tool als **Cypress** een end-to-end test doen: invullen van een nieuw tarief in de browser en verifiëren dat het in het overzicht verschijnt.
- **Oplevercriteria:** De beheerder kan na deze stap *zelfstandig zijn organisatiegegevens en diensten configureren*. Concreet: als oplevering gelden een scherm voor opvangvormen-beheer (met lijst van bestaande vormen en mogelijkheid toe te voegen/bewerken/verwijderen) en een scherm voor tarieven (overzicht van tarieven per opvangvorm, met mogelijkheid toe te voegen/bewerken/verwijderen). Alle gegevensbewerkingen moeten persistent in de database worden opgeslagen en bij herladen van de app weer correct worden getoond. De UI is gebruiksvriendelijk genoeg dat een gemiddelde gebruiker hiermee overweg kan (denk aan labels in NL, helpteksten waar nodig). Tot slot is de data beveiligd: een ingelogde gebruiker van organisatie A kan via de API geen opvangvormen/tarieven van organisatie B opvragen (dit is afgedekt door de autorisatielaag).

Stap 3: Rekenmodule voor ouders (frontend)

- **Ontwikkeling:** Nu de organisaties hun tarieven hebben staan, bouwen we de **frontend rekenmodule** die ouders zullen gebruiken. Dit is feitelijk een aparte pagina of component in de React-app waar geen login voor vereist is, maar die wel gebonden is aan de context van een specifieke organisatie. We ontwikkelen een formulier waarin een ouder per kind keuzes kan maken: selectie van opvangvorm (bijv. dropdown met KDV/BSO/gastouder – geladen vanuit de organisatieconfig), aantal dagen per week (of specifieke weekdagen afhankelijk van gewenste detailniveau), aantal uren per maand (of per week, afhankelijk van hoe de organisatie rekent), eventuele keuze voor een pakket of variant (als de organisatie bijv. 52-weken contract vs flexibele afname aanbiedt), vaste maandbedragen vs uurtarieven, etc. Zodra de ouder de gegevens invult, kan hij/zij op "Bereken kosten" klikken, en de tool rekent het **maandbedrag** uit op basis van de ingevoerde keuzes en de tarieven. De berekening gebeurt ofwel volledig in de frontend (a.d.h.v. de tarieven die via een API-call zijn opgehaald), of gedeeltelijk in de backend (we kunnen ervoor kiezen een API-endpoint `/api/bereken` te maken dat de berekening doet zodat de logica gecentraliseerd is). Voor de MVP houden we de berekening simpel: bijvoorbeeld maandkosten = (uurtarief * aantal uren) of (dagtarief * aantal dagen) plus eventueel vaste toeslagen. Complexere zaken (zoals kinderopvangtoeslag berekenen) vallen buiten scope tenzij expliciet gewenst; de vraag lijkt te doelen op bruto kosten.
- **Technologieën:** We gebruiken in React Chakra UI componenten om de rekenmodule UI te bouwen, bv. meerdere `<Select>` voor keuzes en `<NumberInput>` voor aantallen. Voor de

berekening kan een eenvoudige functie geschreven worden in JavaScript. Als we de berekening server-side doen, gebruiken we Express om de logica in een route handler te implementeren. De tarieven en instellingen van de organisatie worden via de API opgehaald (bv. wanneer de ouderpagina laad, doet de frontend een GET request naar `/api/configuratie?organisatieId=X` om alle nodige info te krijgen). De frontend kan gebruik maken van React state om meerdere kinderen toe te voegen indien nodig (eventueel mogelijkheid "tweede kind toevoegen" wat een extra set keuzes toont). Echter, om MVP behapbaar te houden, starten we met berekening voor **één kind tegelijk**.

- **Test:** We testen de rekenmodule door deze te laden voor een gegeven organisatie (dit kan via een specifieke URL, bv. `/rekentool/{organisatieID}` of door als organisatiebeheerder een "open rekenmodule" knop te hebben). Vul verschillende combinaties in en vergelijk de output met handmatige berekeningen om zeker te weten dat de formule klopt. Bijvoorbeeld: stel KDV uurtarief is €8 en ouder kiest 2 dagen per week à 10 uur per week (ca. ~43 uur per maand); de tool zou ~€344 per maand moeten tonen (8*43). Test ook varianten als vaste maandbedragen: als een pakket "vast €500 p/maand" is geselecteerd, moet de uitkomst dat getal negeren of override zoals bedoeld. Eventuele randgevallen, zoals nul uren, extreme aantallen, combinaties van meerdere kinderen (in latere iteratie) worden ook geprobeerd. Eenheidstests kunnen de berekenfunctie valideren met bekende inputs/outputs. UI-tests controleren dat bij wijzigen van een inputveld de berekening realtime of na klik wordt geüpdatet.
- **Oplevercriteria:** De kernfunctionaliteit van de app – het berekenen van maandelijkse opvangkosten – is nu beschikbaar voor eindgebruikers. Criteria voor gereedheid zijn: ouders kunnen zonder in te loggen de rekentool-pagina openen (in de context van een specifieke organisatie), inputs invoeren en een duidelijk gepresenteerd resultaat zien (bijv. "Geschatte kosten per maand: €X"). Het resultaat moet overeenkomen met de verwachte waarde op basis van de ingevoerde gegevens en de in de beheeromgeving ingestelde tarieven. De oplossing is gebruiksvriendelijk (bijv. formuliervelden gelabeld, misschien dynamisch tonen van extra opties indien van toepassing) en performance is voldoende (berekening gebeurt vrijwel direct).

Stap 4: Ondersteuning voor meerdere organisaties (multitenancy uitbreiden)

- **Ontwikkeling:** Tot nu toe is de applicatie waarschijnlijk getest met één organisatie. In deze stap schalen we dat op naar **meerdere kinderopvangorganisaties** als volledig gescheiden tenants. Dit betekent concreet: (a) de mogelijkheid om meerdere organisatieaccounts te hebben in de database; (b) zorgen dat alle reeds gebouwde functionaliteit (beheer, berekening) correct gefilterd wordt op de huidige organisatie; (c) een inlogscherm dat onderscheid maakt tussen organisaties (dit kan impliciet via de gebruikersaccounts – elke gebruiker hoort al bij een organisatie, dus bij inloggen weten we bij welke organisatie iemand hoort). We moeten misschien voorzien in een manier om de ouder-rekenmodule voor de juiste organisatie te laden. Dit kan via een unieke URL per organisatie, bv. door subdomein (organisatieA.uwdomein.nl/rekentool) of een URL parameter/query (uwdomein.nl/rekentool?org=organisatieA). In een MVP is een simpele aanpak om bijvoorbeeld de organisatie een **code of ID** te geven dat in de embed-code of link zit.
  Daarnaast implementeren we het aanmaken van nieuwe organisaties. Dit kan via de superuser interface (die we in de volgende stap bouwen) of via een script. Voor nu zorgen we dat de structuur er is: nieuwe organisatie in `Organisatie` tabel, een gekoppelde gebruiker met beheerdersrol voor die organisatie, default instellingen (bv. lege tarievenlijst).
- **Technologieën:** Database aanpassingen zijn mogelijk niet nodig als we al vanaf stap 1 een organisatie-id gebruikten. Wel voegen we wellicht een **unieke identifier** of subdomeinveld toe aan de Organisatie tabel om makkelijker de juiste organisatie te vinden bij embed (bijv. een slug: "kinderopvang123"). De Node.js applicatie krijgt middleware die op basis van ingelogde user de `organisatie_id` toevoegt aan requests (zodat controllers niet per ongeluk data van

verkeerde org kunnen pakken). We testen onze queries of ze inderdaad allemaal al gefilterd zijn. Eventueel gebruiken we een library of middleware pattern om multitenancy automatisch af te handelen (sommige ORMs bieden zoiets als `setTenant(organisatieId)` context).

- **Test:** We maken in de testdatabase ten minste twee organisaties aan, elk met hun eigen tarieven en opvangvormen, en twee aparte inloggebruikers. Test 1: log in als beheerder van org A – verifieer dat je **alleen data van A ziet** (opvangvormen/tarieven van A) en dat acties data onder org A wegschrijven. Log in als beheerder van org B – controleer hetzelfde. Probeer bewust fout scenario: gebruiker van org A vraagt via API een resource op van org B (bijv. `/api/tarieven/{idVanB}`); de verwachting is dat de backend dit verhindert (geen toegang of lege response). Dit garandeert data-isolatie. Test ook de oudermodule voor beide organisaties: bijv. open de rekenmodule URL voor org A en org B en kijk of ze elk de juiste tarieven en berekeningen gebruiken. Deze stap verifieert dus het multitenant gedrag.
- **Oplevercriteria:** De applicatie is multitenant ready: meerdere organisaties kunnen er gebruik van maken zonder elkaar te storen. Een nieuw organisatieaccount kan toegevoegd worden en verschijnt als een volledig gescheiden entiteit met eigen beheeromgeving en eigen front-end gegevens. Succescriteria zijn onder andere dat bij N organisaties ingelogd (in verschillende browsers/incognito) elk hun eigen data zien, en dat ouders via een org-specifieke link de juiste rekentool krijgen. Belangrijk: er treden geen kruispollinatie van data op – d.w.z. geen enkel scenario waarbij data van de ene tenant zichtbaar of overschrijfbaar is bij een andere.

Stap 5: Superuser-omgeving voor support

- **Ontwikkeling:** We bouwen nu een **superuser dashboard** waarmee support-medewerkers alle organisaties kunnen beheren. Dit omvat functionaliteiten zoals: een overzichtslijst van alle organisaties; mogelijkheid om een specifieke organisatie te selecteren en in te zien welke opvangvormen/tarieven zij hebben ingesteld; eventueel de mogelijkheid om “in te loggen als” of de organisatieomgeving te simuleren voor support. Ook kan de superuser nieuwe organisaties toevoegen of verwijderen. Concreet ontwikkelen we een aparte login-route voor superusers (of hergebruiken dezelfde maar herkennen het rol-type), die naar een Support Dashboard leidt. In dat dashboard komt een lijst (table) met organisaties en acties per organisatie (bekijken, bewerken, organisatiebeheerder wachtwoord reset, etc.). Daarnaast komt er een form om een nieuwe organisatie aan te maken (wat ook meteen een eerste gebruikersaccount vereist). De backend krijgt corresponderende endpoints, bijv. `GET /api/organisaties` (lijst alle orgs), `POST /api/organisaties` (aanmaken met initial admin), etc. We zorgen dat alleen superusers deze kunnen aanroepen (auth-check op rol).
- **Technologieën:** In React kunnen we routering gebruiken om een apart gedeelte voor superusers te tonen (bijv. routes onder `/admin/*` of een conditie in App die op rol checkt). Chakra UI’s tabel componenten en layout components worden gebruikt om het organisaties-overzicht netjes vorm te geven. Backend: uitbreiden van de autorisatie middleware om superuser rechten toe te staan op bepaalde routes. Database: migratie om eventueel extra info op organisatie of gebruiker vast te leggen (bv. datum aangemaakt, actief/niet actief flags). Eventueel integreren we ook een mailingservice zodat wanneer een nieuwe organisatie wordt aangemaakt, automatisch een welkomstmail met inloggegevens naar de beheerder gestuurd kan worden (hoewel dit nice-to-have is buiten de core scope, maar wellicht handig als oplevercriterium voor een later fase).
- **Test:** We testen door in te loggen als superuser en te controleren of we alle organisaties zien. Probeer de functionaliteiten: maak een nieuwe organisatie aan via het superuser scherm, kijk in de DB of alles correct is toegevoegd (organisatie record, gebruiker, default waarden). Probeer een wijziging: pas een tarief van organisatie A aan via de superuser-interface (als die functionaliteit bestaat) en controleer als je later inlogt als org A dat de wijziging er is. Test de beveiliging: een gewone organisatiebeheerder zou niet via handmatige API calls de superuser

endpoints mogen bereiken (probeer als ingelogde org-admin een GET /organisaties en verwacht een 403). Dit garandeert dat de support tools echt afgeschermd zijn. Ook worden edge cases getest: wat als je een organisatie verwijdert? (Zorg dat gekoppelde data meeverwijderd wordt of gemarkeerd inactief; dit moet doordacht zijn om geen verweesde gebruiker te laten bijvoorbeeld).

- **Oplevercriteria:** De supportomgeving is beschikbaar en functioneel. Vanuit hier kan een beheerder alle organisaties overzien en beheren. Criteria: de superuser kan een nieuwe organisatie succesvol toevoegen (en die organisatie kan daarna inloggen en functioneren), en bestaande organisaties inzien. Alle acties die bedoeld zijn voor support werken (bv. reset wachtwoord zorgt dat de betreffende gebruiker met een nieuw wachtwoord kan inloggen). De superuser interface is simpel maar effectief: support medewerkers kunnen het zonder technische kennis bedienen. We beschouwen deze stap als afgerond wanneer het volledige beheer van tenants via de UI mogelijk is en de beveiliging dit uitsluitend aan superuser accounts toestaat.

Stap 6: Embedden van de rekentool op websites van organisaties

- **Ontwikkeling:** Deze fase richt zich op het **integreerbaar maken** van de ouder-rekenmodule in de websites van de kinderopvangorganisaties. We kiezen hier voor de eenvoudigste robuuste methode: het aanbieden van een embed via een **iframe**. Concreet betekent dit dat elke organisatie in zijn beheeromgeving een stukje HTML-code kan krijgen (een `<iframe>` tag) die verwijst naar de publieke rekentoolpagina voor die organisatie. Bijvoorbeeld: `<iframe src="https://www.ourtool.com/rekentool?org=XYZ" width="100%" height="600" frameborder="0"></iframe>` (waar `XYZ` een unieke code of ID van de organisatie is). Wanneer een organisatie deze code op zijn eigen website plaatst, zal daarin de React-app (alleen het onderdeel voor de rekentool) laden en kunnen ouders direct de berekening doen. We moeten zorgen dat de rekenmodule ook **stand-alone** bruikbaar is binnen een iframe context (bijv. styling die responsive is aan een meestal wat kleinere breedte). We voegen in de React app eventueel een route toe die minimalistisch is (alleen de rekentool component, zonder menu's van de beheerder uiteraard) en die op basis van de org-identificatie de juiste data ophaalt. Als extra detail zouden we de mogelijkheid kunnen bieden om de iframe hoogte dynamisch aan te passen (via postMessage scriptjes) aan de content, maar voor MVP mag een vaste hoogte volstaan als die groot genoeg is.
- **Technologieën:** Het embedden via iframe vereist verder geen extra libraries, alleen duidelijke documentatie aan de klant. In de beheerinterface kunnen we een sectie "Integratie" toevoegen waarin de iframe snippet getoond wordt die ze kunnen kopiëren. Indien we een meer geavanceerde integratie willen aanbieden, kan een **JavaScript embed script** gemaakt worden dat organisaties kunnen insluiten, vergelijkbaar met bijvoorbeeld chat-widgets. Een script snippet zou het voordeel hebben dat het dynamisch een DOM element injecteert. Echter, dit vergt meer ontwikkeltijd (bundelen als UMD module, cross-domain messaging), dus in deze MVP houden we het bij de iframe-oplossing, die **eenvoudig en betrouwbaar** is [8].
- **Test:** Om te testen maken we een simpele HTML-pagina (simulerend de website van een kinderopvangorganisatie) en plakken de aangeboden iframe-code daarin. Open dit in diverse browsers en schermformaten. Controleren: verschijnt de rekentool correct? Werkt de interactie binnen de iframe (invoer en berekening)? Test ook scrollgedrag (als de iframe hoger is dan de container, kan de gebruiker scrollen in iframe zonder problemen). We testen zowel via HTTP als HTTPS om mixed content issues te vermijden (de tool zal op HTTPS moeten draaien als de host site dat ook doet). Daarnaast controleren we dat het laden via iframe **alleen de publieke rekenmodule** toont en niet per ongeluk toegang geeft tot beheerfunctionaliteiten. Dit kan door te verifiëren dat de embed-route geen navigatie naar beheerpagina's bevat (of de React app detecteert dat het in embed-mode is en admin routes verbergt).

- **Oplevercriteria:** Organisaties kunnen de tool succesvol op hun eigen website tonen. Het criterium is dat iemand zonder technische achtergrond de iframe snippet kan plaatsen en daarmee de rekentool werkend ziet. De ingevoerde berekeningen in die embed werken hetzelfde als direct op de tool zelf. Belangrijk is dat er geen veiligheidsrisico's ontstaan: de communicatie is one-way (de iframe leest alleen publieke data van de API voor die org). De stap is klaar wanneer we gedocumenteerd hebben hoe een klant de embed moet opnemen, en dit getest is op een voorbeeldsite.

Stap 7: Meertalige ondersteuning (NL/EN)

- **Ontwikkeling:** In de laatste stap voegen we **internationalisatie (i18n)** toe zodat de tool door zowel Nederlands- als Engels-sprekenden gebruikt kan worden. We integreren een i18n-framework, bijvoorbeeld **react-i18next**, in de React applicatie. Hiermee kunnen we alle zichtbare tekst (labels, knopteksten, foutmeldingen, etc.) definiëren in JSON vertaalbestanden voor NL en EN. Concreet stappen: initialiseer i18next in React (meestal in een aparte `i18n.js` configuratiefile), laad vertaalbestanden voor beide talen, en markeer teksten in de code met vertaalfuncties (`t('sleutel')`). We zorgen dat er een taalkeuze mogelijkheid is: bijvoorbeeld een toggle of dropdown in de UI die taal wisselt. Voor de oudermodule kan de taal eventueel automatisch kiezen op basis van browserinstelling, met optie te override'en. Ook de beheeromgeving wordt tweetalig: dit vergt vertalingen voor alle menu-items, formulieren, etc. De inhoud die organisaties zelf invoeren (zoals namen van opvangvormen) blijft onvertaald of door henzelf te bepalen – mogelijk bieden we niet per se vertaling voor die content aan in MVP. De superuser omgeving (voor support) kan wellicht Engelstalig blijven als de supportmedewerkers Nederlands begrijpen, maar idealiter ook meertalig.
- **Technologieën: react-i18next** is de belangrijkste library hier. Deze detecteert de gebruikersvoorkeur (evt. via `navigator.language`) en laadt de passende resources [9]. We creëren JSON files, bv: `public/locales/nl/translation.json` en `public/locales/en/translation.json`, waarin key-value paren staan voor teksten. Chakra UI componenten zijn over het algemeen al meertalig ondersteund in de zin dat ze geen vaste Engelse labels hebben (behalve misschien date pickers of zo, maar die kunnen we zelf regelen). We controleren of eventueel specifieke componenten (bijv. `<AlertDialog>` knoppen "Cancel/OK") door Chakra of door ons moeten worden vertaald. Backend: foutmeldingen en validation messages die naar de frontend komen, kunnen ook van een taaltag worden voorzien zodat de frontend de juiste boodschap toont, of we implementeren simpele meertalige ondersteuning op de server (echter meestal houdt men de servermeldingen generiek en vertaalt in de client).
- **Test:** We testen de volledige applicatie in beide talen. Schakel de taal en controleer dat alle UI-elementen overschakelen (behalve natuurlijk data die niet vertaald is, zoals organisatienamen of tarieven). Test dat de default taal klopt (bv. browser op NL = Nederlandse interface, op EN = Engels). Test de switcher component. Controleer ook minder vaak bezochte stukken: foutmeldingen, validatieteksten, login scherm, alles moet in de gekozen taal verschijnen. Voor grondigheid laten we bij voorkeur een native speaker of vertaler de teksten nalopen om zeker te zijn dat de toon en terminologie kloppen in beide talen. Daarnaast testen we dat het toevoegen van een nieuwe taal technisch eenvoudig is (documenteer dit eventueel).
- **Oplevercriteria:** De applicatie is volledig bruikbaar in zowel het Nederlands als Engels. Bij oplevering leveren we twee complete sets van vertalingen aan voor alle interface-elementen. De gebruiker kan eenvoudig wisselen van taal, en deze voorkeur blijft behouden tijdens het gebruik (eventueel via locale in localStorage of via URL). Het eindproduct voldoet hiermee aan de taaleis. Belangrijk criterium: de meertalige implementatie mag geen breaking changes introduceren (d.w.z. alle eerdere functionaliteit werkt nog, alleen nu met vertaalde strings).

Aanbevelingen voor verdere architectuur en beheer

Tot slot nog enkele **aanvullende aanbevelingen** om de kwaliteit en toekomstbestendigheid van de rekentool te waarborgen:

- **Schaalbaarheid & Architectuur:** Hoewel de huidige opzet monolithisch is (frontend en backend als twee componenten van één applicatie), is deze modulair genoeg om uit te breiden. Bij groeiend gebruik kan men overwegen de frontend als **static build** te hosten (op bijv. Netlify) en de Node API als serverless functions of op een schaalbare service. De gekozen architectuur (REST API + SPA) maakt dit relatief eenvoudig. Multi-tenancy is al meegenomen in het ontwerp, en door gebruik van een *tenant discriminator* (organisatie-id) op alle data kunnen we opschalen tot vele organisaties in één database [10]. Mocht het aantal tenants zeer groot worden, dan kan men alsnog migreren naar bijv. separate databases per tenant of sharding, maar voor nu is dat niet nodig.

- **Beveiliging:** Blijf OWASP-principes volgen: valideer input zowel client- als server-side, gebruik HTTPS in productie, en zorg voor regelmatige security audits (met name rond de multi-tenant scheiding, omdat data-lek tussen tenants absoluut voorkomen moet worden). Implementeer eventueel Rate Limiting op de API (om misbruik van de rekenfunctie of brute-force login tegen te gaan). Sla gevoelige gegevens (wachtwoorden, tokens) veilig op en log niet onnodig PII.

- **Gebruikerservaring:** Overweeg in de toekomst een meer **dynamische embed** (bijv. via een kleine JavaScript snippet die de React-widget injecteert). Dit kan de integratie soepeler maken en meer custom styling toelaten op de klantsite. Echter, dit vereist aandacht voor CORS en cross-domain issues. De huidige iframe-oplossing is prima als MVP, maar voor hogere mate van integratie kan men de JS-widget route onderzoeken. Daarbij kan men inspiratie halen van hoe chat-widgets werken (een `<script>` tag die een extern script laadt en bij uitvoering een `<div>` in de pagina vult) [11].

- **Performance:** Houd de tool lichtgewicht. React + Chakra is relatief performant, maar zorg dat enkel benodigde componenten geladen worden. Gebruik code-splitting voor de admin vs public parts, zodat ouders die alleen de rekenmodule gebruiken niet de hele admin code binnenhalen. Cache veelvoorkomende data (bv. tarieven wijzigen niet vaak, dus die zou je bij eerste load kunnen cachen in sessionStorage, of via HTTP caching headers werken). Optimaliseer de berekening – die is nu simpel, maar indien er complexiteit bijkomt, monitor de responstijd.

- **Logging & Analytics:** Implementeer logging zowel client- als server-side om gebruik te monitoren en issues snel te spotten. Bijv. tel hoe vaak berekeningen worden gemaakt, of vang foutmeldingen met een service (Sentry voor frontend/backent). Dit helpt de superuser ook in support: eventueel een dashboard hoe vaak elke organisatie de tool gebruikt, of of er fouten optreden bij een specifieke organisatie.

- **Backups en data integriteit:** Regel periodieke backups van de database, zeker omdat meerdere organisaties er afhankelijk van zijn. In een shared hosting context is vaak een standaard MySQL backup voorzien, maar controleer dit. Test ook herstel van backups (disaster recovery scenario).

Samenvattend biedt dit ontwikkelplan een stapsgewijze aanpak om binnen afzienbare tijd een functionele rekentool op te leveren die aansluit bij de behoeften van kinderopvangorganisaties en ouders. Elke stap levert een werkend productincrement op – van de basis login tot de uiteindelijke meertalige, geïntegreerde kostencalculator. Door gefaseerd te werk te gaan, kunnen we na elke fase

feedback verzamelen en zo nodig bijsturen, zodat de eindoplossing gebruiksvriendelijk, betrouwbaar en goed onderhouden is. [8] [9]

---

[1] [2] Introduction to Chakra UI | Refine
https://refine.dev/blog/chakra-ui/

[3] [4] How to host your Node.js App on a shared hosting server | by Nweke Charles | Medium
https://nwekecharles5.medium.com/how-to-host-your-node-js-app-on-a-shared-hosting-server-71f8527aa59a

[5] [6] [10] Designing Your Postgres Database for Multi-tenancy | Crunchy Data Blog
https://www.crunchydata.com/blog/designing-your-postgres-database-for-multi-tenancy

[7] Multitenant Node.js Application with mongoose (MongoDB) | by Prakhar Jain | TechBlog | Medium
https://medium.com/brightlab-techblog/multitenant-node-js-application-with-mongoose-mongodb-f8841a285b4f

[8] How to Create Embedded React Widget
https://selleo.com/blog/how-to-create-embedded-react-widget

[9] Simplifying Multilingual Support in React with i18next: A Step-by-Step Guide | by Aakash Kumar | Medium
https://medium.com/@skysharinghisthoughts/simplifying-multilingual-support-in-react-with-i18next-a-step-by-step-guide-34621de40cbd

[11] Building Embeddable React Widgets: A Complete Guide
https://makerkit.dev/blog/tutorials/embeddable-widgets-react