ChatGPT

Technisch Specificatiedocument – Skillmatching AI Vacatureplatform

Dit document beschrijft de volledige technische specificatie voor het **Cursor App Skillmatching Platform**, een multi-tenant vacatureplatform met geïntegreerde AI-chatbot voor skill-matching. Het systeem is opgezet op basis van het projectplan **Nexa Skillmatching AI** en omvat de volgende technologieën en componenten:

- **Backend:** Laravel 12 (PHP) – inclusief RESTful API en geavanceerd rechtenbeheer (Spatie Roles & Permissions).
- **Frontend:** Angular 20 – moderne single-page application met Material Design (light/dark thema) en responsive layout.
- **Database:** PostgreSQL (host: `mdb.tosun.nl`, poort: 5432, database: `nexa`) – relationele database voor alle data, inclusief chatgeheugen.
- **AI Chatbot:** n8n automation server met een **AI Agent** (Ollama Large Language Model) en **PostgreSQL Chat Memory** voor conversatiegeschiedenis.
- **Containerisatie:** Docker + docker-compose – alle componenten (frontend, backend, database, n8n, AI model) draaien in containers.
- **Authenticatie:** Laravel Sanctum + 2-factor authenticatie (keuze SMS of e-mail).
- **Betalingen:** Mollie Payments API – €15.000 fee bij succesvolle match (hire) tussen kandidaat en vacature.
- **Meertaligheid:** Nederlands (default) en Engels, uitbreidbaar via JSON-vertaalbestanden.
- **Admin UI Template:** Angular Material Admin Dashboard voor beheerdersinterface (inclusief light/dark modus toggle).
- **Multi-Tenancy:** Ondersteuning voor meerdere tenants (bedrijven) met strikte scheiding van data per tenant.
- **SEO Optimalisatie:** Vacatures publiek SEO-vriendelijk gepubliceerd met relevante meta-tags.

De verdere secties werken alle onderdelen uit, inclusief database-ontwerp, configuratie, API-endpoints, frontend en integraties (chatbot, videochat, agenda, notificaties, e-mail en betalingen).

1. Database Schema (PostgreSQL)

Onderstaand is het volledige databaseschema voor het platform, inclusief alle benodigde tabellen (met primaire/foreign keys), kolommen en relaties. De SQL-DDL definieert de structuur, gevolgd door een toelichting per tabel.

```
-- Tabel: companies (tenants)
CREATE TABLE companies (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    slug VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    department VARCHAR(255),
    -- Adresgegevens
```

```
    street VARCHAR(255),
    house_number VARCHAR(10),
    house_number_extension VARCHAR(10),
    postal_code VARCHAR(20),
    city VARCHAR(100),
    country VARCHAR(100),
    -- Contactgegevens
    website VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(50),
    contact_first_name VARCHAR(100),
    contact_middle_name VARCHAR(100),
    contact_last_name VARCHAR(100),
    contact_email VARCHAR(255),
    -- Status
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: users
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(100),
    middle_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    date_of_birth DATE,
    email_verified_at TIMESTAMP,      -- voor 2FA per e-mail
    phone_verified_at TIMESTAMP,      -- voor 2FA per SMS
    company_id INT REFERENCES companies(id),   -- Tenant koppelingskolom
    role_id INT REFERENCES roles(id),
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: roles (gebruikersrollen)
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    description TEXT,
    company_id INT REFERENCES companies(id) DEFAULT NULL,  -- NULL betekent
globale rol (geldt voor alle tenants)
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: permissions (individuele rechten)
CREATE TABLE permissions (
    id SERIAL PRIMARY KEY,
```

```
    name VARCHAR(50) NOT NULL  -- bijv. 'read', 'write', 'delete' of
specifieke permissiecode
);

-- Tabel: role_permission (koppeltabel tussen roles en permissions)
CREATE TABLE role_permission (
    id SERIAL PRIMARY KEY,
    role_id INT REFERENCES roles(id) ON DELETE CASCADE,
    permission_id INT REFERENCES permissions(id) ON DELETE CASCADE
);

-- Tabel: categories (vacaturecategorieën/branches)
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    slug VARCHAR(100) NOT NULL UNIQUE,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: job_configurations (configuraties voor vacatures, bv.
dienstverband-types, werktijden, status)
CREATE TABLE job_configurations (
    id SERIAL PRIMARY KEY,
    type VARCHAR(50) NOT NULL,
-- bijv. 'employment_type', 'working_hours', 'status'
    value VARCHAR(100) NOT NULL,
    company_id INT REFERENCES companies(id) DEFAULT NULL,  -- NULL = globale
config, anders tenant-specifiek
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: vacancies (vacatures)
CREATE TABLE vacancies (
    id SERIAL PRIMARY KEY,
    company_id INT REFERENCES companies(id) ON DELETE CASCADE,  -- de tenant/
bedrijf die de vacature plaatst
    title VARCHAR(255) NOT NULL,
    location VARCHAR(255),
    employment_type VARCHAR(50),     -- bijv. 'Fulltime', 'Parttime', etc
(kan ook FK naar job_configurations)
    description TEXT,
    requirements TEXT,
    offer TEXT,
    application_instructions TEXT,
    category_id INT REFERENCES categories(id),
    reference_number VARCHAR(100),
    logo VARCHAR(255),
    salary_range VARCHAR(100),
    start_date DATE,
```

```
    working_hours VARCHAR(50),      -- bijv. '09:00 - 17:00'
    travel_expenses BOOLEAN DEFAULT FALSE,
    remote_work BOOLEAN DEFAULT FALSE,
    status VARCHAR(20) DEFAULT 'Open',   -- 'Open', 'Gesloten', 'In
behandeling'
    language VARCHAR(20) DEFAULT 'Nederlands',  -- taal van de vacature
    publication_date TIMESTAMP WITHOUT TIME ZONE,
    closing_date TIMESTAMP WITHOUT TIME ZONE,
    meta_title VARCHAR(255),
    meta_description TEXT,
    meta_keywords TEXT,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: matches (koppeling kandidaat <-> vacature, incl. status van
sollicitatie)
CREATE TABLE matches (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE CASCADE,       -- kandidaat
die solliciteert
    vacancy_id INT REFERENCES vacancies(id) ON DELETE CASCADE, -- vacature
waarop gesolliciteerd is
    score NUMERIC(5,2),            -- matchscore (bijv. % overeenstemming,
eventueel door AI bepaald)
    status VARCHAR(20) DEFAULT 'matched',  -- 'matched' (sollicitatie
ontvangen), 'interview', 'hired', 'rejected'
    ai_feedback TEXT,              -- optioneel: feedback/toelichting van AI
over de match
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
    -- evt. extra velden: payment_status, payment_date (voor Mollie betaling
status bij 'hired')
);

-- Tabel: interviews (afspraken voor sollicitatiegesprekken)
CREATE TABLE interviews (
    id SERIAL PRIMARY KEY,
    match_id INT REFERENCES matches(id) ON DELETE CASCADE,
    company_id INT REFERENCES companies(id),   -- redundantie: handig om op
tenant te filteren
    scheduled_at TIMESTAMP WITHOUT TIME ZONE,  -- geplande datum+tijd van
gesprek
    location VARCHAR(255),         -- fysieke locatie (optioneel)
    video_chat_url VARCHAR(255),
-- unieke URL voor videochat sessie (indien remote)
    notes TEXT,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: notifications (in-app notificaties)
```

```sql
CREATE TABLE notifications (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
-- ontvanger van de notificatie
    content TEXT NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- Tabel: email_templates (beheerbare e-mailtemplates voor het CMS)
CREATE TABLE email_templates (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,         -- interne naam van de template,
bijv. 'invite_interview'
    subject VARCHAR(255) NOT NULL,
    body TEXT NOT NULL,                -- HTML/tekst met placeholders
    language VARCHAR(20) DEFAULT 'Nederlands',
    company_id INT REFERENCES companies(id) DEFAULT NULL,  -- NULL = globale
template, anders tenant-specifiek
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);

-- (Optioneel) Tabel: chat_history (chatgeheugen voor AI chatbot, indien niet
automatisch gecreëerd)
CREATE TABLE chat_history (
    id SERIAL PRIMARY KEY,
    session_id VARCHAR(50) NOT NULL,   -- unieke ID per chat sessie (bijv.
conversation ID, of userID/timestamp combo)
    role VARCHAR(20) NOT NULL,        -- 'user' of 'assistant'
    message TEXT NOT NULL,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW()
);
```

Toelichting Tabellen & Relaties:

- **companies** – Bevat alle tenants (bedrijven) die het platform gebruiken. Hierin staan bedrijfsnaam, eventuele afdelingen, adres- en contactgegevens (inclusief contactpersoon) en een activatie-status. Alle data in het systeem is gekoppeld aan een company via `company_id`, wat tenant-isolatie garandeert (elke tenant kan alleen zijn eigen vacatures, gebruikers, etc. zien). Alleen de Super Admin beheert deze tabel (aanmaken/wijzigen/verwijderen van tenants).

- **users** – Gebruikers van het platform. Elke gebruiker is gekoppeld aan één `company` (tenzij het een Super Admin is, die globaal toegang heeft – in dat geval kan `company_id` eventueel null zijn of wijzen naar een speciale global company entry). In de gebruikersgegevens staan naam, email (uniek), versleuteld wachtwoord, geboortedatum en verificatie-tijdstempels voor 2FA (zowel e-mail als telefoon mogelijk). `role_id` verwijst naar de rol van de gebruiker binnen zijn tenant. Relaties: *User* behoort tot een Company en heeft één Rol.

- **roles** – Rollen voor het rechtenbeheer. Dit gebruikt **Laravel Spatie Roles & Permissions** onder de motorkap. Elke rol heeft een naam (bijv. *Super Admin*, *Admin*, *Manager*, *Editor*, etc.) en optioneel een beschrijving. De kolom `company_id` is *null* voor globale rollen (aangemaakt door Super Admin, toepasbaar systeem-breed) of gevuld met een specifieke company_id voor tenant-specifieke rollen (aangemaakt door een tenant Admin, alleen binnen die tenant zichtbaar). Hierdoor kunnen Super Admin-rollen in alle tenants worden gebruikt, en tenant-specifieke rollen alleen binnen de eigen tenant. Voorbeeld: de rol *Super Admin* heeft company_id NULL (globaal), rol *Manager* van bedrijf X heeft company_id = X. Relaties: kan gekoppeld zijn aan een Company, en heeft koppeling naar permissies.

- **permissions** – Overzicht van individuele permissies/rechten (zoals `read`, `write`, `delete` of specifiekere acties). In combinatie met rollen kan zo fijnmazig rechtenbeheer plaatsvinden. Dit is voornamelijk ondersteunend voor Spatie's rollen-permissies systeem.

- **role_permission** – Pivot-tabel (join table) tussen roles en permissions. Hiermee worden permissies aan rollen toegekend. Bijvoorbeeld: rol *Editor* krijgt permissies `read` en `write` maar niet `delete`. Deze tabel wordt door Spatie gebruikt om checkes zoals `user->can('delete')` af te handelen via zijn rol.

- **categories** – Lijst van categorieën of branches waarin vacatures kunnen vallen (bv. IT, Zorg, Techniek, etc.). Elke vacature verwijst naar 0 of 1 categorie (`category_id`). Super Admin beheert de categorielijst (toevoegen/bijwerken), of dit kan desgewenst ook tenant-specifiek uitbreidbaar zijn. In deze tabel staan categorienaam en een URL-vriendelijke slug.

- **job_configurations** – Deze tabel bevat configuratielijsten voor verschillende vacaturegerelateerde keuzelijsten, namelijk *dienstverband types* (employment_type zoals Fulltime, Parttime, Freelance, Stage, Traineeship, etc.), *werkuren* (working_hours zoals '08:00-16:00', '09:00-17:00', etc.), en *vacature statussen* ('Open', 'Gesloten', 'In behandeling'). In plaats van drie losse tabellen is hier gekozen voor één generieke tabel met een `type` veld om het soort config aan te duiden en een `value` voor de waarde. Deze waarden zijn beheerbaar en uitbreidbaar via het admin panel door de **Super Admin** (of per tenant indien `company_id` niet null, maar doorgaans zullen deze globaal zijn zodat elke tenant dezelfde soorten keuzes heeft). De Super Admin heeft een configuratiepagina waar nieuwe dienstverbandtypes, werktijden, statusopties etc. toegevoegd of aangepast kunnen worden. Vacatures refereren naar deze waarden (bv. vacancy.employment_type kan overeenkomen met een entry in job_configurations of direct de string opslaan).

- **vacancies** – De kern tabel voor vacatures. Hierin staan alle vacatures die bedrijven (tenants) aanmaken. Belangrijke velden:

- `company_id`: refereert aan het bedrijf dat de vacature plaatst. Via deze koppeling en tenant-scope zal alleen dat bedrijf (en Super Admin) de vacature kunnen beheren.
- `title`: titel van de functie.
- `location`: locatie (adres, plaats of bijvoorbeeld "Remote").
- `employment_type`: type dienstverband (tekst of als FK naar job_configurations als men dat dynamisch wil beheren).
- `description`, `requirements`, `offer`: uitgebreide omschrijvingen (functieomschrijving, eisen/kandidaatprofiel, aanbod/arbeidsvoorwaarden).
- `application_instructions`: eventuele extra instructies voor kandidaten om te solliciteren.
- `category_id`: optionele verwijzing naar een categorie/branche.

- `reference_number`: eventueel intern referentienummer van de vacature.
- `logo`: pad/URL naar een logo of afbeelding voor de vacature (bijv. bedrijfslogo of vacature-afbeelding).
- `salary_range`: indicatie van salaris (bijv. "€3000-€4000" of "€15 per uur").
- `start_date`: gewenste startdatum.
- `working_hours`: welk schema of uren (e.g. '32-40 uur', of werktijden zoals uit config).
- `travel_expenses` (boolean): of reiskosten vergoed worden.
- `remote_work` (boolean): of (deels) remote werken mogelijk is.
- `status`: status van de vacature: 'Open' (standaard nieuw), 'Gesloten' (niet meer zichtbaar voor kandidaten) of 'In behandeling' (bijvoorbeeld als er al kandidaten in procedure zijn). Deze statuswaarden zijn ook via configuratie te beheren, maar in principe intern gebruikt om de lifecycle van een vacature te managen. De status is zichtbaar met kleurcodering in het dashboard: **Open = lichtgroen**, **Gesloten = lichtrood**, **In behandeling = oranje**.
- `language`: taal van de vacaturetekst (Nederlands of Engels bijvoorbeeld), zodat dit voor SEO en weergave kan worden gebruikt.
- SEO-velden: `meta_title`, `meta_description`, `meta_keywords` – zodat elke vacature haar eigen meta tags heeft voor zoekmachines. Deze worden door de applicatie gebruikt in de pagina `<head>` voor betere vindbaarheid. Het is belangrijk dat bij het publiceren van een vacature deze meta-velden worden ingevuld (handmatig of automatisch gegenereerd) zodat Google de vacature goed kan indexeren. **SEO**: Alle relevante vacature-informatie (titel, locatie, categorie, etc.) wordt ook op de vacaturepagina getoond en via meta-tags gemarkeerd, zodat vacatures snel en hoog rankend gevonden kunnen worden via zoekmachines.
- `publication_date` en `closing_date`: geven aan wanneer de vacature live is gegaan en tot wanneer reageren mogelijk is. Dit kan gebruikt worden om automatisch vacatures op "Gesloten" te zetten na de sluitingsdatum.
- Timestamp velden `created_at` / `updated_at` voor audit.

Relaties & Indexering: Vacatures horen bij één Company (bedrijf) en kunnen een Category hebben. Belangrijke kolommen zoals status, category_id, etc. zouden voorzien worden van indexen om filtering in het frontend (bv. filter op categorie, status) snel te laten verlopen. In het tenant-backend ziet men alleen eigen vacatures. In de frontend (publieke vacatureoverzicht) worden alleen vacatures met status 'Open' getoond aan kandidaten.

- **matches** – Deze tabel vertegenwoordigt een **sollicitatie of match** tussen een kandidaat (user) en een vacature. Telkens als een kandidaat reageert op een vacature (na doorlopen van de chatbot-vragen), wordt hier een record aangemaakt. Velden:
- `user_id`: de kandidaat die solliciteert.
- `vacancy_id`: de vacature waarop gesolliciteerd is.
- `score`: een matchscore (0-100%) die aangeeft hoe goed de kandidaat op basis van skills/ voorkeuren bij de vacature past. Deze kan bv. door de AI-chatbot of matching algoritme worden berekend aan de hand van de gegeven antwoorden en vacature-eisen. In eerste instantie kan dit veld leeg of optioneel zijn, of een eenvoudige berekende waarde (bijv. aantal overeenkomende criteria).
- `status`: de fase van deze match in het sollicitatieproces. Start op 'matched' zodra de kandidaat gereageerd heeft. Vervolgens kan het bedrijf de status wijzigen:
    - **matched** (of *applied*): sollicitatie ontvangen, nog geen actie.
    - **interview**: kandidaat is uitgenodigd voor een gesprek (in behandeling).
    - **hired**: kandidaat is aangenomen (match succesvol).
    - **rejected**: kandidaat is afgewezen.

- `ai_feedback`: optioneel tekstveld waar de AI of recruiters feedback/opmerkingen over de match kunnen zetten, bijvoorbeeld een samenvatting van de motivatie antwoorden of reden van selectie/afwijzing.
- `created_at`: moment van aanmaken (sollicitatie ingediend).

Elke `match` vormt de basis voor verdere acties: bij *matched* ontvangt de vacature-eigenaar een notificatie en e-mail, bij *interview* wordt een interview gepland, bij *hired* wordt een betalingstrigger geactiveerd. **Betaling:** Bij status *hired* hoort het bedrijf een fee te betalen (via Mollie). Hiervoor kan ofwel in `matches` een extra veld komen zoals `payment_status` (open/paid) en bijvoorbeeld `paid_at`, of een aparte betaling/factuur tabel (zie hieronder bij betalingen). In deze specificatie houden we het simpel door betalingen af te leiden van de match status en Mollie transactie logs.

- **interviews** – Deze tabel beheert de geplande sollicitatiegesprekken (video meetings of fysieke afspraken) tussen een bedrijf en een kandidaat:
- `match_id`: verwijst naar de bijbehorende match (kandidaat + vacature) waarvoor dit gesprek is.
- `company_id`: redundant opgeslagen voor makkelijke queries per tenant (dit is dezelfde als via de vacancy->company, maar direct erbij voor indexering per bedrijf).
- `scheduled_at`: datum en tijd van het gesprek.
- `location`: locatie van het gesprek indien fysiek (adres of plaatsnaam, of evt. “Online”).
- `video_chat_url`: de unieke URL voor de videochat sessie indien het een online gesprek is. Beide partijen kunnen via deze link deelnemen aan de call. Deze link wordt gegenereerd bij het inplannen (zie Videochat integratie sectie).
- `notes`: eventueel notities of opmerkingen (bijv. agendapunten of voorbereidende info).
- Timestamps voor aangemaakt/bijgewerkt.

Een bedrijf (via de backend) kan voor elke ontvangen sollicitatie één of meerdere interviewafspraken plannen. De kandidaat krijgt hiervan een notificatie en e-mail (met de link en details). Zodra ingepland, kan het systeem ook automatische herinneringen sturen (bijv. 1 dag van tevoren).

- **notifications** – Generieke tabel voor in-app notificaties (melding icon in de frontend/backoffice). Wanneer bepaalde events gebeuren (nieuwe sollicitatie, uitnodiging voor gesprek, herinnering betaling, etc.), wordt hier een record gemaakt voor de betreffende gebruiker. Velden:
- `user_id`: de gebruiker (ontvanger) voor wie de melding is.
- `content`: de tekst/informatie van de notificatie (bijv: *"Nieuwe sollicitatie van Jan Jansen op vacature X"*, *"Betaling voor match Y is nog niet voldaan"*, *"Uitnodiging: gesprek gepland op [datum]"*, etc.). Dit kan eventueel in de voorkeurstaal van de gebruiker worden gegenereerd.
- `is_read`: boolean of de gebruiker de notificatie al gezien/geopend heeft. Ongelezen notificaties tellen op tot een rood badge-nummer bij het notificatiebelletje in de UI.
- `created_at`: aanmaakdatum (gebruikt om sortering te doen, nieuwste eerst).

Notificaties worden getoond in zowel de bedrijf (tenant) omgeving als voor kandidaten (in hun accountpagina) waar relevant. Bijvoorbeeld: een kandidaat ziet notificatie wanneer een bedrijf hem uitnodigt voor interview; een bedrijf ziet notificatie wanneer een nieuwe sollicitatie binnen is of als er een herinnering voor betaling is verstuurd. Super Admin zou notificaties kunnen krijgen voor bijv. nieuwe tenant aangemaakt, maar dat is optioneel.

- **email_templates** – Deze tabel beheert de inhoud van diverse e-mails die het systeem verstuurt, via een kleine CMS-interface. Denk aan: bevestigingsmail bij registratie, notificatie mail naar bedrijf bij nieuwe sollicitatie, uitnodiging voor interview naar kandidaat, wachtwoord-reset mail, betalingsherinnering, etc. Velden:

- `name`: logische naam van de template, gebruikt door het systeem om de juiste template te laden (bv. *"new_application_company"*, *"interview_invite_candidate"*, *"payment_reminder_company"*).
- `subject`: het onderwerp van de e-mail.
- `body`: de HTML- of tekstinhoud van de e-mail, met placeholders/variabelen die tijdens verzending vervangen worden (bijv. `%USERNAME%`, `%JOB_TITLE%`, `%COMPANY_NAME%`, `%PAYMENT_LINK%`, etc.). Het CMS-beheer geeft een lijst van beschikbare variabelen per template.
- `language`: taal van de template (Nederlands of Engels). Er kunnen dus meerdere records van hetzelfde `name` bestaan voor verschillenden talen. Bij het verzenden kiest het systeem de juiste taalversie op basis van ontvanger of tenant voorkeur.
- `company_id`: eventueel om tenant-specifieke e-mailtemplates toe te staan. Standaard zullen templates globaal zijn (company_id = NULL) zodat alle tenants dezelfde basis e-mails gebruiken. Maar een tenant zou bijv. hun eigen branding/tekst willen voor bepaalde mails; dan kan een variant met company_id worden aangemaakt die het systeem verkiest boven de globale. Dit is een uitbreidbaarheid-optie.
- `created_at`, `updated_at`: voor versiebeheer/logging.

De Super Admin (en eventueel tenant Admins, beperkt tot hun eigen templates) kunnen via een rijke teksteditor (WYSIWYG) deze e-mails aanpassen. Het platform zorgt ervoor dat de variabelen goed worden vervangen bij versturen en dat er geen onbevoegde HTML/js kan worden ingevoerd (beveiliging).

- **chat_history** – (*optioneel*, kan door n8n automatisch worden aangemaakt indien geconfigureerd*.) Deze tabel is bedoeld voor het PostgreSQL Chat Memory** van de AI-chatbot. Het n8n-systeem kan conversatiehistorie opslaan in een Postgres tabel om de context te bewaren tussen berichten. Als we deze tabel zelf definiëren (met naam `chat_history` of soortgelijk), kunnen we die koppelen. Velden:
- `session_id`: een sleutel die een chatgesprek identificeert. Dit kan bijvoorbeeld een unieke ID per gebruiker sessie zijn of per vacature-chat. Bij een ingelogde kandidaat kan dit bijv. de `match_id` of `user_id` zijn; voor een anonieme bezoeker op de homepage (die de chatbot gebruikt om vacatures te zoeken) kan een random sessie-ID worden gebruikt (opgeslagen in de browser localStorage/cookie).
- `role`: geeft aan wie het bericht stuurde – `'user'` (kandidaat/gebruikersvraag) of `'assistant'` (AI-chatbot antwoord).
- `message`: de inhoud van het bericht.
- `created_at`: tijdstip van het bericht.

De n8n **Postgres Chat Memory** node zal bij configuratie de tabelnaam verwachten en zo nodig zelf aanmaken. Hier hebben we alvast een passend schema neergezet. Deze geschiedenis is belangrijk zodat de AI bij vervolgvragen of een dialoog weet wat eerder is gezegd/gevraagd. De Laravel-app zelf hoeft deze tabel niet direct te gebruiken; het wordt voornamelijk door de chatbot-flow beheerd. Wel moeten de juiste database-credentials in n8n worden ingesteld om deze tabel te kunnen benaderen (zie .env configuratie).

Database relaties & constraints: Naast de reeds benoemde foreign keys is in Laravel een **Global Scope** of middleware voorzien om tenant-isolatie af te dwingen: elke query op bijvoorbeeld users/ vacancies wordt automatisch gefilterd op `company_id` = van de huidige gebruikerscontext (tenzij de gebruiker een Super Admin is, die over alle tenants heen kan kijken). Hiermee wordt gegarandeerd dat tenants elkaars gegevens niet zien. Migraties in Laravel zullen deze tabellen aanmaken. Ook worden pivot-tabellen voor Spatie (bijv. `model_has_roles`, `model_has_permissions`, etc.) door het Spatie package zelf ingericht – die kunnen vergelijkbaar zijn met role_permission maar specifiek voor

polymorfe relaties. In het schema hierboven is een vereenvoudigd model gegeven; in de implementatie kunnen de Spatie-tabellen gebruikt worden (met model_type kolommen) in plaats van `role_permission`. Het principe blijft hetzelfde.

2. Configuratiebestanden (.env)

De toepassing maakt gebruik van een `.env` configuratiebestand in de Laravel backend (en indien nodig, environment-specific config voor Angular) om alle gevoelige gegevens en endpoints te beheren. Hierin worden ook de koppelingen naar de database en AI-chatbot vastgelegd. Belangrijke configuratieparameters zijn onder andere:

- **Database connectie:**
  `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD` – Instellingen voor de PostgreSQL database `nexa`. Hiermee krijgt Laravel toegang tot de hoofddatabase. Ook de n8n-chatbot zal via deze database werken, dus dezelfde credentials (of een aparte db-user) moeten in de n8n instance bekend zijn. Denk aan een aparte `.env` voor n8n als dat vereist is, of configuratie via de n8n UI (referenties naar deze DB).

- **App URL en domein instellingen:**
  `APP_URL` – basis-URL waarop de Laravel backend draait (nodig voor correcte URL-generatie, bijv. in e-mails). Voor multi-tenant kan eventueel gebruik worden gemaakt van subdomeinen per tenant (bijv. `{tenant}.platform.nl`), maar in deze opzet is tenant-scheiding op applicatieniveau; het domein is generiek. Indien subdomeinen gebruikt worden, moet `APP_URL` dynamisch zijn of multi-tenancy middelware de juiste vinden.

- **Authenticatie & Security:**
  `APP_KEY` – de Laravel sleutel voor encryptie (al gegenereerd via artisan).
  `SANCTUM_STATEFUL_DOMAINS` – bij gebruik van Sanctum voor SPA-auth kan hier het front-end domein worden opgegeven zodat sessiecookies werken.
  Configuratie voor 2FA: er is niet één standaard env-var, maar mogelijk keys voor SMS service. Bijvoorbeeld:

- `SMS_API_KEY`, `SMS_PROVIDER` – API-sleutel en provider voor SMS-verificatie (als we bijv. Twilio, MessageBird o.i.d. integreren voor versturen van SMS-codes).

- `2FA_ENABLED=true` – om 2-staps-authenticatie te activeren.
  Laravel zelf zal 2FA logica custom nodig hebben (bijv. via een package of eigen implementatie), waarbij in `.env` ingesteld wordt welke methode default is of hoe lang codes geldig zijn.

- **Mail instellingen:**
  `MAIL_MAILER` (bijv. SMTP), `MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, `MAIL_PASSWORD`, `MAIL_FROM_ADDRESS`, `MAIL_FROM_NAME` – voor het versturen van e-mails (bevestigingsmails, notificaties, etc.). Deze worden gebruikt door Laravel's mail functionaliteit en moeten correct ingevuld zijn (bijv. een transactional email service of eigen SMTP). Omdat e-mails een belangrijk onderdeel zijn (notificaties, uitnodigingen, etc.), moeten deze instellingen kloppen. Voor ontwikkel/test kan Mailhog in docker-compose worden opgenomen.

- **AI Chatbot & n8n integratie:**
  `N8N_URL` – URL van de n8n instance (bijv. `https://n8n.tosun.nl` zoals opgegeven).
  `N8N_CHAT_WEBHOOK` – Endpoint of workflow ID voor de chatbot in n8n, indien de frontend via

API calls communiceert. (Bijvoorbeeld een webhook URL die de Angular app aanroept voor elke chat-bericht). We kunnen een aparte webhook workflow in n8n maken die gekoppeld is aan de chat AI-agent; deze webhook-URL (met token) wordt hier geconfigureerd.
`AI_MODEL` – Naam of endpoint van het AI model (bijv. een parameter voor n8n om het juiste Ollama model te laden, mocht dat nodig zijn). Als de AI Agent node dit intern regelt, is dit wellicht niet nodig, maar als bijvoorbeeld n8n via een HTTP Request node een eigen AI-service moet aanroepen, dan komt hier de URL.
`AI_API_KEY` – (Optioneel) als er een API sleutel nodig is voor het AI model (bij externe modelproviders, niet van toepassing bij lokale Ollama tenzij beveiligd).

Omdat we Ollama lokaal draaien, is er eigenlijk geen externe API key nodig, maar mogelijk wel een instelling voor modelnaam. De **Ollama** container serveert op poort 11434; n8n moet weten hoe te connecten (doorgaans `http://ollama:11434` binnen Docker netwerk). In n8n's eigen configuratie kunnen we aangeven dat het Ollama model gebruikt moet worden. In `.env` van Laravel zou eventueel alleen de `N8N_URL` en eventueel een auth key voor n8n komen als de webhook beveiligd is.

- **Betalingsprovider (Mollie) instellingen:**
  `PAYMENT_PROVIDER` – op "MOLLIE" zetten om de code te laten weten welke provider gebruikt wordt.
  `MOLLIE_API_KEY` – de live/test API key die je van Mollie krijgt om betalingen te kunnen initiëren.
  `MOLLIE_WEBHOOK_URL` – URL waar Mollie betalingsupdates naartoe moet sturen (naar een Laravel API endpoint, zie API sectie). Deze kan door Laravel dynamisch gegenereerd worden, maar in .env kan je bijv. domein zetten en code bouwt `'/api/payment/webhook'` eraan.
  `MATCH_FEE_AMOUNT` – eventueel het bedrag (15000) configurabel houden via .env, zodat dit makkelijk aanpasbaar is zonder codewijziging. Hier zou dan `15000` ingesteld worden (bedrag in cents of euros afhankelijk van implementatie).

- **Meertaligheid / Locale:**
  `APP_LOCALE` – standaardtaal, bv. `nl`.
  `APP_FALLBACK_LOCALE` – fallback taal, bv. `en`.
  Eventueel flags voor beschikbare talen. Angular gebruikt zijn eigen config voor i18n, maar we kunnen via API ook de beschikbare talen aan front-end doorgeven indien nodig. In Laravel kan een lijst van ondersteunde locales in config files staan; .env niet strikt nodig, maar men kan toch bv. `AVAILABLE_LOCALES=nl,en` opnemen om in de beheerinterface te tonen welke keuzes er zijn.

- **Overige integraties:**
  Als later bijvoorbeeld agenda-integratie met externe calendar (Google Calendar API) zou gebeuren, kunnen client IDs/secrets ook in .env. Idem voor een eventuele integratie met videochat service (als niet volledig in-house).
  Voor nu is videochat eigen implementatie (zie sectie 7), dus geen externe API keys nodig daarvoor.

Retrieval van .env in Laravel: Laravel biedt het config() systeem aan om deze variabelen op te halen. De koppeling naar n8n-chatbot gebeurt bijvoorbeeld via een environment var `N8N_CHAT_WEBHOOK`; de applicatie (of direct de Angular frontend) gebruikt deze om berichten door te sturen. We zorgen ervoor dat alle gevoelige keys veilig op de server staan en niet naar de frontend gebundeld worden (behalve eventuele openbare info zoals een public token of de base URL als dat nodig is).

Configuratie n8n zijde: Aangezien n8n als aparte service draait, moet daar ook de databaseverbinding ingesteld worden voor de Postgres Chat Memory node. Dit gebeurt binnen n8n door een **Credentials** aan te maken voor Postgres (host, database, user, password – overeenkomstig de hierboven genoemde DB-config) en die toe te wijzen aan de Chat Memory node. Verder moet in n8n de verbinding met het **Ollama** model worden ingesteld. Omdat Ollama in Docker draait, is het aan te raden n8n en Ollama in hetzelfde docker netwerk te plaatsen; in n8n AI Agent node kan dan als model-endpoint `http://ollama:11434` gebruikt worden (zie Docker Compose). In de n8n configuratie kan eventueel `N8N_COMMUNITY_PACKAGES_ALLOW_TOOL_USAGE=true` gezet worden (via environment) om de AI agent toe te staan extra tools te gebruiken, zoals een database query.

Samengevat: alle belangrijke koppelingen (DB, AI, payment) zijn via configuratie instelbaar, waardoor de applicatie flexibel is en de chatbot direct inzetbaar is zodra de containers starten en de env correct is ingevuld.

3. Docker Compose Configuratie

Het volledige platform wordt gecontaineriseerd met Docker. Met een **docker-compose.yml** file kunnen alle onderdelen opgezet en gelinkt worden, wat ontwikkeling en uitrol vereenvoudigt. Hieronder een overzicht van de Docker Compose configuratie en hoe de services samenwerken:

```
version: "3.8"
services:
  # Backend service (Laravel PHP-FPM container)
  backend:
    image: php:8.2-fpm  # PHP-FPM base image, Laravel zal via volume worden
toegevoegd
    container_name: laravel_app
    volumes:
      - ./backend:/var/www/html    # aanname: backend code in ./backend
directory
    working_dir: /var/www/html
    environment:
      - APP_ENV=local
      - APP_DEBUG=1
      - APP_URL=http://localhost    # tijdens ontwikkeling
      - DB_HOST=db
      - DB_PORT=5432
      - DB_DATABASE=nexa
      - DB_USERNAME=nexa_user
      - DB_PASSWORD=nexa_pass
      - BROADCAST_DRIVER=redis
      - CACHE_DRIVER=redis
      - QUEUE_CONNECTION=redis
      - SESSION_DRIVER=redis
      - MAIL_HOST=mailhog
      - MAIL_PORT=1025
      - MAIL_USERNAME=
      - MAIL_PASSWORD=
      - MAIL_FROM_ADDRESS=no-reply@platform.nl
      - MAIL_FROM_NAME="Skillmatch Platform"
```

```
      - N8N_URL=http://n8n:5678
      - N8N_CHAT_WEBHOOK=https://n8n.tosun.nl/webhook/ai-chat   # voorbeeld
webhook URL voor chatbot
      - MOLLIE_API_KEY=test_xxxxxxxxxx    # Mollie test key
      - PAYMENT_PROVIDER=MOLLIE
      - MATCH_FEE_AMOUNT=15000
    depends_on:
      - db
      - redis
    networks:
      - app_net

  # Frontend service (Angular app in dev server or static server)
  frontend:
    image: node:20-alpine   # NodeJS for building Angular (or use nginx for
served build)
    container_name: angular_app
    volumes:
      - ./frontend:/usr/src/app
    working_dir: /usr/src/app
    command: sh -c "npm install && npm run start -- --host 0.0.0.0"
    ports:
      - "4200:4200"   # Angular dev server port
    depends_on:
      - backend
    networks:
      - app_net

  # Database service (PostgreSQL)
  db:
    image: postgres:15-alpine
    container_name: postgres_db
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=nexa
      - POSTGRES_USER=nexa_user
      - POSTGRES_PASSWORD=nexa_pass
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - app_net

  # Cache/Broker service (Redis) for queues, caching, real-time
notifications
  redis:
    image: redis:7-alpine
    container_name: redis_cache
    networks:
      - app_net
    ports:
```

```
      - "6379:6379"
    volumes:
      - redis_data:/data

  # n8n service (Automation + AI chatbot)
  n8n:
    image: n8nio/n8n:latest   # official n8n image
    container_name: n8n
    ports:
      - "5678:5678"
    environment:
      - N8N_BASIC_AUTH_ACTIVE=true        # beveilig de n8n instance
      - N8N_BASIC_AUTH_USER=admin
      - N8N_BASIC_AUTH_PASSWORD=adminpass
      - WEBHOOK_URL=http://localhost:5678  # in dev: adjust for production
domain
      - N8N_PUBLIC_API_HOST=n8n
      - N8N_PUBLIC_API_PORT=5678
      - DB_TYPE=postgresdb
      - DB_POSTGRESDB_HOST=db
      - DB_POSTGRESDB_PORT=5432
      - DB_POSTGRESDB_DATABASE=nexa
      - DB_POSTGRESDB_USER=nexa_user
      - DB_POSTGRESDB_PASSWORD=nexa_pass
      - N8N_COMMUNITY_PACKAGES_ALLOW_TOOL_USAGE=true   # nodig voor AI agent
tools (DB query)
    depends_on:
      - db
      - ollama
    networks:
      - app_net

  # Ollama AI Model service (serving LLM for chatbot)
  ollama:
    image: ollama/ollama:latest
    container_name: ollama
    ports:
      - "11434:11434"   # port for Ollama's API
    volumes:
      - ollama_data:/root/.ollama   # to persist models
    command: ollama serve
    networks:
      - app_net

  # Mailhog (for local email testing)
  mailhog:
    image: mailhog/mailhog
    container_name: mailhog
    ports:
      - "8025:8025"   # web interface for mailhog
      - "1025:1025"   # SMTP port
```

```
    networks:
      - app_net

networks:
  app_net:
    driver: bridge

volumes:
  db_data:
  redis_data:
  ollama_data:
  n8n_data:
```

Toelichting Docker Compose:

- **Backend (Laravel):** Gebaseerd op een PHP-FPM image. De Laravel code wordt via een volume gemount. In productie zou men een op PHP-FPM + Nginx configuratie hebben (bijv. twee containers: één php-fpm, één Nginx die static files serveert en doorverwijst naar php-fpm). Hier is voor eenvoud alleen de PHP container getoond; voor een volledige setup kan een aparte Nginx container aan `services` worden toegevoegd die `depends_on: backend` en poort 80:80 mapt, met site config om php via `backend:9000` aan te spreken. De environment sectie toont belangrijke .env variabelen voor de container, waaronder de databaseverbinding en integratiekeys. Redis wordt gebruikt voor caching, sessies en queues (Mail en notificaties via queue bijv.). De backend container hangt in `app_net` netwerk zodat hij de db, redis, etc. bij naam kan vinden.

- **Frontend (Angular):** Voor ontwikkelmodus is hier Node gebruikt die de Angular app serveert op poort 4200. In productie zou men de Angular app builden (`npm run build`) en de statische output door een Nginx server laten hosten (of via Laravel's blades, maar hier gaan we uit van een aparte SPA). In dev is hot-reload handig, dus Node met Angular CLI dev-server. Het volume mount de Angular project directory. In het compose voorbeeld is geen specifieke environment voor Angular, maar als de Angular app bepaalde environment config nodig heeft (bv API base URL), kan men ofwel een environment.ts instellen of via een environment variable. We kunnen bijv. `API_BASE_URL=http://localhost` meegeven en in Angular bij build inlines replacen. De frontend container hangt ook op hetzelfde netwerk, maar eigenlijk communiceert de Angular app direct via de browser naar de API (via een reverse proxy/Nginx or direct to backend's exposed port). In dev, de backend isn't exposed in this file on host ports, one might either expose port (e.g. 8000:80 for Nginx) or call host from Angular. This detail kan aangepast worden; uiteindelijk zal voor productie een front-end build in Nginx container draaien die via docker networking de backend kan bereiken.

- **Database (PostgreSQL):** Standaard Postgres container, storing data in `db_data` volume for persistence. Credentials zijn zoals in .env van backend. De hostnaam binnen docker network is `db` (matching what backend and n8n use). Port 5432 is ook aan de host gebonden voor toegang (optioneel; in productie kan men db niet openzetten). **Initialisatie:** Men kan een init script of migrate command gebruiken om tabellen aan te maken (Laravel migrations). Tijdens development kan `docker-compose exec backend php artisan migrate --seed` uitgevoerd worden om schema in te richten volgens bovenstaande.

- **Redis:** Simpele Redis cache, ook op netwerk. Laravel gebruikt dit voor snelle session/queue. Geen persistent volume nodig voor cache, maar aangemaakt voor volledigheid.

- **n8n (AI Chatbot service):** n8n draait op poort 5678 (ook ge-exposed voor toegang tot de editor UI en webhook consumption). De `WEBHOOK_URL` environment moet in productie gezet worden naar de publieke URL waar webhooks bereikbaar zijn (hier in dev op localhost). We hebben basic auth aangezet op n8n's editor voor veiligheid (in productie wellicht achter VPN of disabled UI). Belangrijk:

- Database voor n8n zelf is hier ook ingesteld op dezelfde Postgres. n8n kan zijn eigen tabellen maken (prefixed). Dit is optioneel; men kan ook SQLite of aparte DB gebruiken, maar hergebruik is oké mits naming conflicts voorkomen (n8n prefixes tables als ik me goed herinner).
- `N8N_COMMUNITY_PACKAGES_ALLOW_TOOL_USAGE` is true gezet. Dit laat de AI Agent node extra tools gebruiken, zoals een DB-query tool of HTTP calls, wat in onze chatbot workflow nodig kan zijn om bijv. vacatures op te zoeken.
- We set `depends_on: ollama` en in the environment did not explicitly set the model, but in the workflow JSON we will specify usage of Ollama. Because n8n and ollama share `app_net`, de AI agent kan straks `http://ollama:11434` benaderen om het model te gebruiken.
- Volume `n8n_data` kan worden toegevoegd als mount if we want to persist workflows credentials etc., but in snippet we've included the volumes: section at bottom (though not attached to n8n in example, could be missing a mount line; ideally mount a local folder or named volume to `/home/node/.n8n` as shown commented, for persistence of config and workflows).

- **Ollama (AI model):** De Ollama container draait de AI model server. We expose port 11434. Op start laadt deze container standaard geen model; men zou via een command of at runtime een model moeten laden (via `ollama pull <model>` and then queries). In een development setup kan men vooraf het gewenste model (bijv. Llama 2 7B chat or Mistral etc.) in de volume `ollama_data` plaatsen of via an init script. Voor eenvoud nemen we aan dat het model al geconfigureerd is of dat n8n via de AI Agent node het model-commando doorgeeft (n8n's Ollama integration zal de modelnaam nodig hebben, bijv. `ollama: Llama2`). De container laat toe dat n8n verbinding maakt via de hostnaam `ollama` op poort 11434. (Belangrijk: als Docker op Linux draait kan een extra_hosts configuratie nodig zijn om `host.docker.internal` te laten resolven, maar hier bypassen we dat door samenvoeging in compose network).

- **Mailhog:** Dit is een optionele service voor development om e-mails te onderscheppen. In de Laravel .env staat MAIL_HOST=mailhog en poort 1025, zodat alle uitgaande mails hierheen gaan. Via poort 8025 kan je de mailbox in de browser bekijken. In productie wordt Mailhog niet gebruikt; in plaats daarvan stel je echte SMTP/ mailservice in.

Met deze opzet kan men het hele systeem starten met één commando (`docker-compose up`) en zijn alle componenten beschikbaar en verbonden. **Scaling:** Men kan eventueel meerdere instantie van backend of frontend schalen als load het vereist (met een load balancer voor frontend). Ook n8n kan schaalbaar zijn maar dat is geavanceerd gebruik (voor ons doel volstaat 1 instance). De multi-tenant structuur zit in de software, dus meerdere tenants draaien op dezelfde containers/DB.

4. API Architectuur (Routes & Endpoints)

De Laravel backend biedt een RESTful API die door de Angular frontend en de AI-chatbot (via webhooks) wordt gebruikt. Authenticatie gebeurt voornamelijk met **Laravel Sanctum** (voor SPA login sessions of API tokens) in combinatie met de inlog- en 2FA-flow. Hieronder een overzicht van de belangrijkste routes/endpoints, gegroepeerd per functionaliteit, inclusief authenticatievereisten en een indicatie van request/response.

Authenticatie & Gebruikers: (prefix: `/api/auth/`) - `POST /api/auth/register` – Registreert een nieuwe gebruiker. Er zijn twee varianten: - **Kandidaat registreert zich** (via frontend): levert naam, email, wachtwoord, evt. aanvullende info. Standaard wordt zo iemand als *Kandidaat* (bijbehorende rol) aangemaakt binnen een algemene "candidates" tenant of standalone zonder tenant? *(Opmerking:* Aangezien het platform multi-tenant is, moeten we beslissen hoe kandidaten bestaan. Waarschijnlijk zijn kandidaten niet gekoppeld aan een specifiek bedrijf, tenzij men kiest dat elk kandidaat ook een tenant is – dat is niet logisch. Waarschijnlijk hebben we een globale tenant of company "Candidates" of we laten company_id toe NULL voor kandidaten accounts die geen eigen bedrijf hebben. We kunnen voor nu aannemen dat kandidaten geen eigen tenant hebben en company_id NULL of een speciale waarde krijgen.) - **Bedrijf registreert zich**: een bedrijf kan ook een account aanmaken (inclusief tenant). Dit kan via een ander endpoint of parameter. Mogelijk wordt dit door de Super Admin gedaan i.p.v. self-service, om wildgroei te beperken. In veel SaaS scenario's maakt de Super Admin tenants aan en krijgen die een Admin user. Maar als men open registratie voor bedrijven wil, kan `POST /api/auth/register?type=company` voorzien worden: dan wordt naast user ook een nieuwe company record aangemaakt en de gebruiker wordt Admin daarvan. - Validatie en beveiliging: Sterk wachtwoord vereisen, uniek e-mail. Na registratie eventueel e-mail verificatie sturen.

- `POST /api/auth/login` – Logt een gebruiker in. Verwacht email en wachtwoord, en eventueel 2FA code als 2FA vereist is. Flow:
- **Eerste factor:** Laravel valideert credentials. Bij succes:
  - Als 2FA voor deze gebruiker actief is, genereer een eenmalige code (6-cijferig) en stuur via gekozen kanaal (SMS of email). Sla code (hashed) tijdelijk op (bijv. in cache of in een `two_factor_codes` tabel met expiratie).
  - Retourneer een response dat aangeeft dat 2FA code vereist is (HTTP 200 met payload `{"2fa_required": true}`).
  - **Zonder 2FA:** Sanctum maakt direct een sessie cookie of token aan en we geven success + user info terug.
- **Tweede factor:** Endpoint: `POST /api/auth/verify-2fa` – gebruiker vult de code in, wij valideren tegen verstuurde code. Bij correct: markeer gebruiker als verified (set email_verified_at of phone_verified_at) en maak de daadwerkelijke login sessie/token aan (Sanctum). Frontend krijgt login success en kan doorgaan.

- Alternatief: Men kan ook implementeren dat *na login credentials direct token geven maar markeren als 2FA_pending en API weigert echte data tot code geverifieerd.* Simpler is bovenstaand twee-staps model via aparte endpoints.

- `POST /api/auth/logout` – Logt de huidige gebruiker uit (invalideren token of sessie).

- `GET /api/auth/user` – Geeft profielinfo van ingelogde gebruiker (naam, rol, tenant info) zodat frontend context heeft (bijv. om menu-opties op rollen te baseren).

Tenants & Beheer: (prefix: `/api/tenants/` – alleen voor Super Admin) - `GET /api/tenants` – **(Auth: Super Admin)** Lijst van alle companies/tenants, met basisinfo en stats (bv aantal users, vacatures). - `POST /api/tenants` – **(Auth: Super Admin)** Nieuwe tenant aanmaken (bedrijf registreren). In de payload bedrijfsgegevens en eventueel direct een admin gebruiker. Alternatief: dit kan ook via een GUI in admin dashboard gebeuren en dan de API wordt intern aangeroepen. - `GET /api/tenants/{id}` – **(Auth: Super Admin)** Details van één tenant (inclusief lijst van gebruikers, vacatures, etc. afhankelijk van of we dat willen embedden of via andere endpoints). - `PUT /api/tenants/{id}` – **(Auth: Super Admin)** Bewerkt tenant informatie (naam, contactpersoon, actief ja/nee). - `DELETE /api/tenants/{id}` – **(Auth: Super Admin)** Verwijdert een tenant (optioneel, wellicht logisch om eerst data over te hevelen of soft delete).

- **Tenant context switch:** Super Admin moet kunnen "impersonaten" of wisselen van tenant om hun data te bekijken. Dit kan via een endpoint als `POST /api/tenants/{id}/impersonate` dat een token teruggeeft waarmee frontend als die tenant kan handelen. Of simpeler, binnen de admin frontend, super admin kan een dropdown selecteren om tenant X te managen, en de API kan een query parameter of header verwachten (bijv. `X-Tenant-ID: {id}`) die Laravel scopes. Dit vereist maatwerk in middleware: detecteer als ingelogde user superadmin en er is een tenant header, dan override de default scope voor die requests. Dit is een detail in implementatie; in documentatie volstaat notitie dat Super Admin via het admin dashboard tenants kan selecteren en hun data inzien.

Gebruikers & Rollen: (prefix: `/api/users/`) - `GET /api/users` – Geeft lijst van gebruikers **binnen de eigen tenant**. - Auth: Tenant Admins kunnen dit voor hun tenant doen. Super Admin kan een query-param `?tenant_id=X` meegeven om die tenant's users te krijgen of via impersonation zoals boven. - `POST /api/users` – **(Auth: Admin)** Nieuwe gebruiker aanmaken binnen huidig tenant. Wordt gebruikt door bijvoorbeeld een Admin om collega's uit te nodigen (manager/editor accounts). - `GET /api/users/{id}` – Detail van user (indien binnen eigen tenant of als SuperAdmin). - `PUT /api/users/{id}` – Bewerk gebruiker (rollen toekennen, activa status, etc.). - `DELETE /api/users/{id}` – Verwijder gebruiker. - Eventueel specifieke endpoints: - `POST /api/users/{id}/resend-invite` – om opnieuw een uitnodigingsmail te sturen. - `GET /api/roles` – lijst van beschikbare rollen (binnen tenant + globale). Bijvoorbeeld nodig bij user aanmaken (dropdown rol). - `POST /api/roles` – **(Auth: Admin)** nieuwe rol binnen eigen tenant aanmaken (beperkt tot niet hoger dan zichzelf). Super Admin kan globale rollen aanmaken via wellicht andere route of same with company_id null. - `GET /api/permissions` – lijst alle mogelijke permissies (alleen Super Admin relevant wellicht). - `POST /api/roles/{id}/permissions` – rechten koppelen aan rol (Super Admin of tenant Admin als het eigen tenant rol betreft).

Vacatures (Jobs): (prefix: `/api/vacancies/`) - **Publieke endpoints (geen auth vereist) voor vacatures bekijken:**
- `GET /api/vacancies` – Geeft een openbare lijst van **Open** vacatures (status = Open) voor weergave op de front-end website. Ondersteunt query parameters voor filters:
- `?category_id=`, `?employment_type=`, `?location=`, `?q=` (zoekterm) etc. Deze filteren respectievelijk op categorie, dienstverband, locatie (substring of postcode) en vrije zoekterm (die t.o.v. titel/omschrijving zoekt).
- Paginering parameters `?page=` en `?per_page=` voor lijst.
Standaard sortering: nieuwste eerst (publication_date desc). Eventueel kan `?sort=` param verschillende sorteeropties bieden (bv op relevatie als AI matchingscore meegegeven wordt bij zoekterm, anders op datum). - `GET /api/vacancies/{id}` – Publieke detailweergave van een vacature. Bevat alle info uit de vacature (title, description, requirements, etc. inclusief misschien de company naam info), behalve interne dingen. Deze data wordt gebruikt om de vacature detailpagina te

tonen. Ook zitten hier de SEO meta velden bij, die de frontend in de `<head>` kan stoppen (Angular kan via router resolver de data laden en meta service invullen).
Als een vacature niet `Open` is of niet bestaat, retourneert dit 404 (tenzij een ingelogde eigenaren bedrijf het opvraagt, zie backend endpoint hieronder). - Eventueel `GET /api/vacancies/filters` – voor populatie van filtermogelijkheden (lijst van categorieën, lijst van alle employment_types, etc.). Dit kan ook ingebakken in frontend of via aparte endpoints: - categorieën endpoint is er al (`GET /api/categories`). - employment types, status etc. kunnen via `GET /api/job-configs?type=employment_type` etc. of we sturen één object: `{categories: [...], types: [...], locations: [...]}`. Locaties zijn vrij veld, maar men zou top 10 steden met open jobs kunnen laten zien etc., niet strikt noodzakelijk via API te leveren. - SEO: eventueel een `GET /sitemap.xml` route die alle open vacatures lijnt voor zoekmachines.

- **Beveiligde endpoints (voor bedrijven/tenants) voor vacaturebeheer:**
  (Auth vereist: role Editor, Manager, Admin of Super Admin met juiste rights)
- `GET /api/tenant/vacancies` – Lijst van vacatures van de ingelogde gebruiker's tenant, ongeacht status. Filterbaar op status eventueel (`?status=Open/Gesloten`). Hierin kunnen ook concepten of gearchiveerde zitten indien men dat ondersteunt. Default sortering: nieuwste eerst. Dit endpoint wordt gebruikt in het bedrijfsportal om overzicht te tonen met kolommen incl. status met kleur.
- `POST /api/tenant/vacancies` – Nieuwe vacature aanmaken. Vereist alle nodige velden (tenzij sommige optioneel). Server-side worden eventueel defaults ingesteld (status default Open, publication_date default nu). Na aanmaken kan de vacature direct gepubliceerd zijn (Open) of men kan concept status implementeren; niet genoemd hier, dus aangenomen dat aanmaken = Open.
- `GET /api/tenant/vacancies/{id}` – Detail van een vacature voor bewerken. Dit is vrijwel dezelfde data als publieke detail maar aangevuld met eventueel interne notities indien die zouden bestaan.
- `PUT /api/tenant/vacancies/{id}` – Bewerkt een bestaande vacature. Zaken als titel, beschrijving updaten, of status wijzigen (bijv. vacature sluiten als de positie vervuld is). Als status op 'Gesloten' gezet wordt, zal de frontend hem niet meer tonen publiek.

- `DELETE /api/tenant/vacancies/{id}` – Vacature verwijderen. Optioneel: men kan ook soft-deleten en status 'Gesloten' gebruiken in plaats van echt fysiek verwijderen, om data te bewaren. In een eenvoudig scenario mag verwijderen, met CASCADE zorgt dat matches en interviews van die vacature ook weggaan.

- **Bulk acties:** Wellicht om meerdere vacatures te sluiten of te openen, maar niet per se nodig voor MVP.

- **Sollicitaties & Matching (Matches):** (prefix: `/api/matches/`)

- `POST /api/vacancies/{id}/apply` – Endpoint voor kandidaten om te solliciteren op een vacature. **Auth:** de kandidaat moet ingelogd zijn (als user). Als niet, kan de frontend hem redirecten naar login/registreer en daarna terug. Dit endpoint triggert de AI-chatbot flow:

  - Mogelijk ontwerp: de frontend doet niet direct POST /apply, maar opent eerst de chatbot UI (zie integratie sectie). De chatbot zelf verzamelt alle antwoorden en **aan het einde** van de chatsequence zal via de webhook alsnog de sollicitatie daadwerkelijk indienen. Dus de API call kan intern vanuit n8n komen naar dit endpoint om de match te creëren zodra chatbot klaar is.

    - Een alternatief is dat de frontend meteen een match record aanmaakt met status 'matched' wanneer de gebruiker op "Reageer" klikt, en daarna de chat gebruikt om details op te halen en via een PUT de match bijwerkt. Maar netter is wachten tot de chat klaar is en dan één call.
    - Hier kiezen we: de chatbot workflow zal uiteindelijk een POST doen naar (bijvoorbeeld) `/api/matches` of specifieke apply endpoint, met alle verzamelde info (vacature-id, user-id (uit context), antwoorden op vragen). De backend maakt de match record aan, eventueel berekent een `score` en slaat `ai_feedback` (bijvoorbeeld de geformuleerde motivatie of een samenvatting) op. Vervolgens stuurt de backend een e-mail naar de vacature-eigenaar en een notificatie.
    - Response naar frontend: success of failure. De chatbot UI kan dit melden ("Bedankt voor je sollicitatie!") of de frontend route kan daarna melding tonen.

- `GET /api/matches` – **(Auth: Admin/Manager)** Lijst van alle matches (sollicitaties) binnen de eigen tenant (dus op vacatures van dat bedrijf). Dit geeft recruiters een overzicht van alle inkomende kandidaten. Mogelijk filters: `?vacancy_id=` om per vacature te filteren, `?status=matched` etc. In de Super Admin context, `GET /api/matches?tenant_id=X` zou alle matches van tenant X tonen (of via impersonation).

- `GET /api/matches/{id}` – Detail van een specifieke sollicitatie. Dit kan o.a. de antwoorden van de kandidaat bevatten (op de vragen) als die bij `ai_feedback` of elders zijn opgeslagen, plus gegevens van de kandidaat (naam, CV als die was toegevoegd, etc.). Hier kan men beslissen of men kandidaatprofiel ook deels opslaat of alles via de chat deed. Wellicht is een CV optioneel upload, dat dan bij match gekoppeld kan zijn (we zouden een Documenten tabel kunnen hebben, maar dat is nu niet genoemd – wellicht later als uitbreiding).

- `PUT /api/matches/{id}` – Bewerken van de match status door het bedrijf. Dit wordt gebruikt om de kandidaat door het proces te bewegen:

    - Als een bedrijf een kandidaat wil uitnodigen: men zet status op 'interview'. Dit zou ook trigger kunnen zijn om een interview te plannen, maar planning is beter via aparte endpoint hieronder. We kunnen het splitsen:
    - Een `PUT` om status=interview te zetten, en tegelijk meegeven interview details of
    - Beter: via interviews endpoint een interview creëren, en dat laat de status vanzelf upgraden.
    - Als bedrijf besluit kandidaat aan te nemen: `status = hired` via PUT. Dit triggert het betalingssysteem: de backend maakt een Mollie betaling aan van €15.000 en geeft de betaallink terug (of stuurt email). De match record kan tijdelijk een `payment_status = pending` markeren.
    - Als bedrijf de kandidaat afwijst: `status = rejected`. Dit kan ervoor zorgen dat kandidaat een afwijs-mail krijgt en eventueel de vacature weer Open blijft voor anderen.
    - Elke statusupdate kan notificaties genereren: bij interview -> notificatie naar kandidaat ("Je bent uitgenodigd voor een gesprek"), bij hired -> notificatie naar Super Admin (zodat die weet dat betaling moet plaatsvinden, of we kunnen direct een betaalnotificatie naar bedrijf sturen?), bij rejected -> notificatie naar kandidaat ("Helaas, u bent niet geselecteerd").

- `DELETE /api/matches/{id}` – Eventueel om een sollicitatie te verwijderen/terugtrekken. Een kandidaat zou misschien zijn sollicitatie willen intrekken; in dat geval zou hij dit endpoint kunnen aanroepen als we dat toestaan (auth: user is eigenaar van match). Of een bedrijf kan een record verwijderen (maar beter afwijzen i.p.v. verwijderen). Dit is optioneel.

- **Interviews (afspraken):** (prefix: `/api/interviews/`)

- `POST /api/matches/{id}/interview` – Plant een nieuw interview in voor de betreffende sollicitatie/match. **Auth:** Recruiter (Admin/Manager of Editor met rechten). Payload bevat `scheduled_at` (datum+tijd), `location` (of een indicator voor remote), eventueel `notes`. De backend maakt een nieuw record in `interviews` aan, genereert een `video_chat_url` (zie sectie 7) als het remote is, en:
    - Stuurt een e-mail naar de kandidaat met de uitnodiging (met datum, tijd, locatie of link) en plaatst een notificatie voor de kandidaat.
    - Eventueel stuurt een kalender-uitnodiging (.ics bijlage) mee zodat kandidaat het in eigen agenda kan zetten.
    - Update de match status naar 'interview' (als dat nog niet was gedaan). Dit kan hier automatisch gebeuren, zodat men niet twee aparte calls hoeft te doen.
- `GET /api/interviews?vacancy_id=X` – Lijst alle interviews gepland voor vacature X (voor recruiters handig om planning te zien).
- `GET /api/interviews?candidate_id=Y` – (Of zonder param voor ingelogde kandidaat) – zodat een kandidaat de lijst van zijn geplande gesprekken kan zien (in zijn profiel).
- `PUT /api/interviews/{id}` – Bewerkt een geplande afspraak (bijvoorbeeld tijd verzetten of locatie aanpassen). Mogelijk alleen bedrijf kan dit doen, of kandidaat kan een voorstel doen via communicatie buiten om; als we geavanceerd wilden, kunnen we reschedule functionaliteit bouwen. Voor nu is het vooral recruiter-gedreven.

- `DELETE /api/interviews/{id}` – Interview annuleren. Dit zou notificatie naar kandidaat sturen dat het gesprek is geannuleerd of verplaatst indien meteen herpland.

- **Notificaties:** (prefix: `/api/notifications/`)

- `GET /api/notifications` – Haalt alle notificaties voor de ingelogde gebruiker op, ongelezen eerst. Dit wordt gebruikt om het meldingen-menu in de frontend te vullen. Kan eventueel alleen ongelezen teruggeven of parameter `?unread_only=true`.
- `POST /api/notifications/mark-read` – Markeert één of meerdere notificaties als gelezen (client stuurt IDs in body). Alternatief: `PUT /api/notifications/{id}` to mark individually.

- Notificaties worden normaliter ook realtime gepusht (zie notificatiesectie), maar de API is er als fallback en initial load.

- **E-mail templates:** (prefix: `/api/email-templates/` – Auth: Super Admin of Tenant Admin indien tenant-specifieke templates)

- `GET /api/email-templates` – Lijst van e-mailtemplates (mogelijk filter op taal of tenant).
- `GET /api/email-templates/{id}` – Ophalen inhoud van een template om te bewerken.
- `PUT /api/email-templates/{id}` – Bijwerken van subject/body (en evt. naam of taal). We zorgen dat de body veilig opgeslagen wordt (strip ongewenste tags).
- `POST /api/email-templates` – Nieuw template toevoegen (waarschijnlijk niet veel nodig, meeste templates zijn vooraf gedefinieerd in seeds).

- Deze endpoints stellen de beheerder in staat de content te wijzigen, maar de triggers welk template wanneer gebruikt wordt, liggen vast in de backend code (of configuratie).

- **Configuraties (Job configurations):**

- `GET /api/job-configs?type=employment_type` (of `working_hours`, `status`) – Geeft lijst van waarden voor dat type, eventueel tenant-specifiek als Admin van tenant opvraagt (dan combineer globale en tenant overrides). Dit is nodig om dropdowns te vullen in UI.
- `POST /api/job-configs` – **(Auth: Super Admin)** nieuwe config waarde toevoegen (kan via admin UI).
- `PUT /api/job-configs/{id}` – aanpassen van naam (bijv. hernoem “Tijdelijk/ZZP” naar “Tijdelijk”).
- `DELETE /api/job-configs/{id}` – verwijderen indien niet in gebruik (cascade regels bedenken).

- **Zoekfunctionaliteit (via Chatbot of direct):**
  Naast het standaard filteren via `/api/vacancies` endpoints, is er integratie met de AI chatbot. De chatbot kan door user input criteria verzamelen en direct een zoekopdracht uitvoeren. Dit kan gerealiseerd worden via:

- Een speciaal endpoint `POST /api/vacancies/search-by-answers` dat een lijst van vacatures teruggeeft op basis van door de chatbot verzamelde antwoorden (bijv. een JSON payload met {functie: "Developer", regio: "Utrecht", dienstverband: "Fulltime", salaris: 3000, remote: false, sector: "IT", beschikbaar_vanaf: "per direct"}). De backend vertaalt dit naar een query (met bijvoorbeeld functies voor fuzzy search op titel/beschrijving voor functie, filter op locatie radius voor regio, etc.).
- De n8n workflow zou dit endpoint kunnen aanroepen als *tool* tijdens het gesprek om direct resultaten te krijgen. De respons kan dan door de AI aan de gebruiker gecommuniceerd worden ("Ik heb 5 vacatures gevonden die passen bij uw voorkeuren. Zie de lijst op de achtergrond."). De frontend kan tegelijkertijd de normale lijst bijwerken (via directe API call of via data die de chatbot teruggeeft).
- Dit endpoint vereist geen login als het alleen algemene zoek doet. Het gebruikt alleen publieke vacatures. Het is voornamelijk een convenience voor AI-gestuurde zoeken. (NB: Dit is een uitbreiding; initieel kan de chatbot ook volstaan met de instructie dat de user de filters moet gebruiken. Maar gezien de vraag, implementeren we het intelligent.)

API Authenticatie & Beveiliging:
- We maken gebruik van **Laravel Sanctum** om de API te beveiligen. In een SPA scenario kan Sanctum cookies gebruiken (stateful) of we kunnen Personal Access Tokens aanmaken. Aangezien onze frontend en backend op dezelfde domain (of subdomain) zitten, kunnen we stateful cookies gebruiken: bij login via `/auth/login`, Sanctum issues een cookie (HttpOnly, Secure, SameSite) that the browser sends on subsequent calls. Dit vereenvoudigt de setup (geen expliciete token handling in JS, minder gevoelig voor XSS). We moeten dan wel `SANCTUM_STATEFUL_DOMAINS` instellen en CORS goed configuren. - Alle gevoelige endpoints vereisen Auth. We definiëren middleware groepen: - `auth:sanctum` voor ingelogde gebruikers (kandidaten, bedrijven). - Extra middleware voor role permissions: bv. `role:admin` voor tenant admin-only endpoints, `role:super` voor super admin. Spatie faciliteert een `->middleware('role:Admin')` etc. - Multi-tenant scope: een middleware `TenantScope` die bij ingelogde user de `company_id` opvangt en een global scope zet zodat alle queries `WHERE company_id = {id}` bevatten. Voor Super Admin kan die scope uit staan of overschreven bij specifieke endpoints (via een header of impersonation mechanisme zoals genoemd).

- **Error handling:** API responses geven nette foutboodschappen in JSON (met codes en messages), bijvoorbeeld 401 Unauthorized voor niet ingelogd toegang, 403 Forbidden voor gebrek aan permissie, 422 Validation errors met details per veld, 404 Not Found als resource niet bestaat of niet tot jouw tenant behoort, etc.

- **Versionering:** Eventueel prefix `/api/v1/` in URL voor versioning. In deze spec hebben we het niet expliciet vermeld maar dat is aan te raden zodat later API veranderingen beheersbaar zijn.

Deze API-architectuur zorgt ervoor dat de Angular frontend alle benodigde acties kan uitvoeren via AJAX calls. De integratie met de chatbot gebeurt via webhooks of speciale endpoints (zoals zoekfunctie of apply), wat in de volgende sectie verder wordt toegelicht.

5. Frontend Structuur (Angular 20)

De frontend applicatie is gebouwd in **Angular 20** en volgt de best-practices voor modulariteit, routing en responsive design. Er wordt gebruikgemaakt van Angular Material (voor UI componenten) en een Material Admin Dashboard template voor de beheerkant. Hieronder de belangrijkste aspecten van de frontend structuur:

Architectuur & Modules:
- We delen de applicatie op in duidelijk gescheiden modules, die **lazy loading** gebruiken via Angular routing. Bijvoorbeeld: - **Public module** – voor niet-ingelogde gebruikers: landingspagina, vacature-overzicht, vacature-detail, registreren/inloggen, etc. - **Candidate module** – voor ingelogde kandidaten: profielpagina, overzicht eigen sollicitaties, notificaties, etc. - **Company/Portal module** – voor ingelogde bedrijf gebruikers (Admin/Manager/Editor): bevat dashboard, vacaturebeheer, kandidaten/matches overzicht, planning, etc. - **Admin module** – voor Super Admin interface: tenant beheer, globale instellingen, statistieken.

Deze modules laden alleen in wanneer de betreffende route wordt bezocht (lazy loading), wat de initial load van de app snel houdt.

- **Routing:** In `app-routing.module.ts` definiëren we route-segmenten:
- `'/'` (empty path) – Landingspagina (bijv. redirect naar `/vacatures` of een homepage met uitleg en een zoekbalk/chat).
- `'/vacatures'` – Vacaturelijstpagina (component toont filteropties en lijst van vacatures).
- `'/vacatures/:id'` – Vacature detailpagina (component die details ophaalt en toont, plus een "Reageer" knop).
- `'/login'` & `'/register'` – Auth pages (met eventueel child routes of separate for candidate vs company register).
- `'/dashboard'` – Bedrijfsdashboard (ingelogde omgeving home).
- … Binnen `/dashboard` of een `'/portal'` prefix kunnen we subroutes maken voor verschillende admin functies:
    - `'/dashboard/vacatures'` – lijst van eigen vacatures.
    - `'/dashboard/vacatures/nieuw'` – vacature aanmaken.
    - `'/dashboard/vacatures/:id/bewerken'` – vacature aanpassen.
    - `'/dashboard/sollicitaties'` – lijst van binnengekomen matches.
    - `'/dashboard/sollicitaties/:id'` – detail van sollicitatie + actieknoppen (uitnodigen, afwijzen).
    - `'/dashboard/agenda'` – kalenderweergave van geplande interviews (met optie doorklikken).
    - `'/dashboard/instellingen'` – voor tenant admin: beheer van gebruikers, rollen, en evt. eigen email templates.
    - `'/admin'` – (alleen Super Admin rol) parent route voor super admin panel:
    - `'/admin/tenants'`, `'/admin/tenants/:id'` – beheersschermen voor tenants.

    - `'/admin/categories'`, `'/admin/configurations'` – beheer globale categorieën en vacatureconfig.
    - `'/admin/statistieken'` – overkoepelende statistieken, filters per tenant.

Route Guards: We zetten Angular **route guards** in om bescherming te bieden: - AuthGuard: voorkomt toegang tot ingelogde routes voor anonieme gebruikers (checkt via een AuthService of token bestaat). - RoleGuard: controleert of gebruiker de juiste rol heeft voor een route. Bijvoorbeeld SuperAdminGuard voor `/admin` routes, CompanyUserGuard voor `/dashboard` (Admin/Manager/Editor), CandidateGuard voor kandidaatprofiel routes. Deze guards krijgen info van JWT token claims of een globale user service (die door API `/auth/user` was gevuld bij app load). - Additionally, within company portal, we might restrict certain subpages via permission (like only Admin can go to instellingen). We can either handle that in component (if unauthorized, hide or redirect) or fine-tune guards if needed.

- **Layout & Navigation:** We hebben naar doelgroep gescheiden navigaties:
- **Public layout:** Bovenaan een navigatiebar (bv. logo, knop Inloggen/Registreren, taal-switch). Homepage kan marketing info tonen plus direct de vacature zoeklijst.
- **Candidate logged-in layout:** Navigatiebar met misschien link naar vacatures, en een dropdown/icoon voor profiel & notificaties.
- **Company portal layout:** Sidebar menu (Material sidenav) with sections: Vacatures, Sollicitaties, Agenda, etc., and topbar with toggles for theme, notifications, user menu (log out). We use the Material Admin Dashboard styling, which likely comes with a responsive drawer menu.
- **Admin layout:** Kan grotendeels hetzelfde als company portal, maar met extra menu-items (Tenants, Configuraties, Statistieken).

Het Material Admin template zorgt voor een consistente, professionele uitstraling. Inclusief toggling tussen licht/donker thema (we integreren dit via Angular Material theming; wellicht een service die de CSS class toggelt, triggered by a button in the header). Dit thema voorkeur kan per gebruiker opgeslagen worden (in een User preference field of localStorage).

- **Componenten & UX:** We gebruiken Angular Material components voor formulieren (mat-form-field, inputs), tabellen (mat-table) voor lijsten, modals/dialogs voor bevestigingen of input (bv. bij uitnodigen gesprek, een dialoog vragen om datum/tijd), en SnackBar/toasts voor feedback (zoals "Vacature opgeslagen", "E-mail verstuurd"). We implementeren **responsive design** zodat op mobiel en tablet alles bruikbaar is:
- De vacaturelijst schakelt bijvoorbeeld naar kaartweergave op mobiel (onder elkaar).
- Het admin dashboard menu colapst tot icon-only of hamburger on narrow screens.

- De videochat component moet full-screen of in een responsive modal werken voor goed gebruik. Testen op verschillende schermgroottes is onderdeel van development.

- **Filtering & Zoekfunctionaliteit:** Op de vacature overzichtspagina (public) zijn diverse filters mogelijk:

- Branche/categorie (dropdown of lijst met checkboxes).
- Vacaturetype / dienstverband (multi-select of checkboxes voor fulltime/parttime etc.).
- Locatie (input veld voor stad of postcode, eventueel auto-aanvullen of radius filter als uitbreiding).
- Salaris (een slider voor salarisrange of dropdown ranges).
- Thuiswerkmogelijkheid (checkbox of dropdown: Op locatie / Hybride / Remote).
- Trefwoord zoeken (vrij tekstveld, zoekt in titel/beschrijving).

- Ervaring/Opleidingsniveau (mogelijk als filter als we die gegevens van kandidaten of vacatures hebben; in vacature staat niet direct een niveau, maar we zouden "seniority" of opleidingsniveau als veld kunnen toevoegen aan vacature of eisen, wat matcht met kandidaatprofiel).

In de UI zorgen we dat deze filters duidelijk zichtbaar zijn (zijbalk of boven de lijst). De gebruiker kan combinaties kiezen. Bij elke verandering van filter zal Angular de lijst opnieuw ophalen via de API met de filters als query params, of we kunnen alle vacatures binnenhalen en client-side filteren als dat weinig data is; maar bij potentieel veel vacatures is server-side filter beter. We kiezen server-side filtering via API calls, zodat ook de AI zoek via chatbot consistent die filters gebruikt.

- **Vacature detailpagina:** Toont alle informatie van de vacature overzichtelijk:
- Titel, locatie, bedrijf (tenzij anoniem, maar lijkt niet zo), categorie, dienstverband, salarisindicatie e.d. in een info blok.
- De beschrijving, eisen en aanbod in tabs of secties.
- Eventueel bedrijf informatie (bedrijfsprofiel: deze info kan uit companies tabel komen, bv. description/website).
- **Solliciteer knop:** opvallende knop die de chatbot start als gebruiker ingelogd is. Als niet ingelogd, de knop stuurt naar login/registreer – na succesvol inloggen keert terug naar deze vacaturedetail en kan opnieuw op Solliciteer klikken om chatbot te starten.

- SEO: Angular Universal (Server-side rendering) zou ideaal zijn om deze pagina voor Google te laten indexeren met meta tags. Mocht SSR buiten scope zijn initieel, dan zorgen we dat Angular via Meta service de meta_title/description van vacature zet bij navigatie. Google's bots kunnen tegenwoordig ook JS renderen, maar SSR geeft betere resultaten. Dit kan als toekomstige optimalisatie benoemd worden (zie sectie Toekomstige Uitbreidingen wellicht).

- **Inloggen & Registratie flows:**

- Registratiepagina: laat keuze toe voor *"Ik zoek werk (kandidaat)"* vs *"Ik ben een werkgever (bedrijf)"*. Op basis daarvan tonen we extra velden:
    - Kandidaat: enkel persoonlijke info.
    - Werkgever: bedrijfsnaam, adres, contactpersoon etc. (of we beperken registratie van bedrijven via aparte procedure).
- Na registratie: eventueel e-mail verificatie, dan login.

- Loginpagina: vraagt email & wachtwoord. Bij submit:

    - Als 2FA niet vereist voor die user, direct inloggen (Sanctum cookie krijgt).
    - Als 2FA vereist, tonen we een volgende stap (zelfde component of navigatie naar /verify-code). Daar kan men kiezen om code via SMS of e-mail te ontvangen als we die keuze bieden. In ons systeem zou de keuze vooraf ingesteld zijn (de gebruiker heeft wellicht bij registratie of in profiel aangegeven voorkeur sms of email). We tonen dus "We hebben een verificatiecode gestuurd naar uw [e-mail/telefoon]. Voer de code in: [input]." plus knop "Herstel code verzenden".
    - Bij code correct -> inloggen compleet, navigeren naar juiste dashboard:
    - Kandidaten gaan naar bijv. `/vacatures` of eigen profiel page.
    - Bedrijf users gaan naar `/dashboard` (portal home).
    - Super Admin naar `/admin`.
    - *Uitloggen* verwijdert sessie en cookie; navigatie naar homepage.

- **Meertaligheid (i18n):** Alle tekst in de frontend UI wordt uit externe JSON-labelbestanden geladen, zodat eenvoudig gewisseld kan worden tussen Nederlands en Engels.

- We gebruiken bijvoorbeeld **ngx-translate** of Angular's i18n. In deze context, JSON label-bestanden per taal zijn genoemd, wat duidt op gebruik van ngx-translate (Angular's eigen i18n werkt met compile-time XLIFF, maar een runtime JSON approach is flexibeler voor dynamische omschakeling).
- Structuur: `/src/assets/i18n/nl.json` en `en.json` etc., waarin key-value paren staan voor alle UI labels en berichten. Bijvoorbeeld:

```
{
  "LOGIN_TITLE": "Inloggen",
  "EMAIL": "E-mailadres",
  "PASSWORD": "Wachtwoord",
  "LOGIN_BUTTON": "Inloggen",
  "LOGOUT": "Uitloggen",
  "FILTER_LOCATION": "Locatie",
  ...
}
```

- Er is een taal-switch component (bijv. een knop met vlag-icoon) in de UI, die via de translate service de actieve taal zet. Standaard taal is Nederlands.
- In zowel frontend als backend is Nederlands de default (foutmeldingen aan gebruiker ook in NL tenzij user voorkeur EN kiest).
- Voor content (vacature teksten) wordt geen automatische vertaling gedaan, die verschijnen in de taal waarin ze zijn ingevoerd (bedrijven kunnen vacatures in NL of EN plaatsen; de `language` veld geeft dit aan zodat de UI bv. een vlag kan tonen of filter op taal mogelijk is).
- Voor beheer van de labels: Super Admin zou een pagina kunnen hebben waar hij keys kan toevoegen of bestaande vertalingen kan aanpassen. Dit is lastiger direct in JSON te schrijven via web. Een aanpak is om deze vertalingen ook in de database te beheren (bv. een `translations` tabel met key, nl_text, en_text, etc.), en bij wijziging de JSON bestanden opnieuw te genereren of de frontend via API te laten ophalen. Echter, men heeft expliciet JSON bestanden genoemd; wellicht is het idee dat developers die bijwerken. Als onderdeel van platform kan een eenvoudige interface gemaakt worden die de JSON overschrijft (de container moet schrijfrechten hebben) of een export-knop.

- In het kader van dit project beschrijven we dat er *een beheerpagina is om de vertalingen in te voeren* (zoals ook in projectplan stond). Dat kan een simpel formulier zijn die direct de DB of file bijwerkt. De implementatie daarvan kan bedrijfsspecifiek zijn.

- **Responsive Design:** We zorgen dat de applicatie er goed uitziet en bruikbaar is op verschillende devices:

- Gebruik van flex-layout of CSS Grid voor gridviews. Material components zijn grotendeels responsive.
- Vacaturelijst: op desktop als tabel of cards in grid (3-col), op mobiel 1-col stacked.
- Formulieren: op mobiel full-width fields, op breed scherm naast elkaar waar logisch.
- Admin dashboard: Material sidenav in push mode for mobile (hamburger menu), normal mode for desktop.

- Videochat: moet ook op mobiel werken (full screen weergave als mogelijk, en chat/discussie perhaps not needed as separate if one can speak).

- **Login/Registratie**: Ook mobile-friendly. Eventueel social login could be future idea (LinkedIn sign-in to import CV, etc.), not in scope now but worth noting as future.

- **Notificatie weergave:** Rechtsboven is er een bell icon. Als `notifications` API ongelezen > 0, toont een rode badge met aantal. Bij klikken opent een dropdown of sidebar met lijst van laatste notificaties (titel + tijd). Onclick van een notificatie:

- Markeer als gelezen (badge count update).
- Voer bijbehorende actie: bijv. navigeren naar de betreffende sollicitatie of interview detail. We kunnen in de content of separate field de type/id verstoppen om te weten waarheen: bijvoorbeeld content: "Nieuwe sollicitatie voor *[Vacature X]* door Jan Jansen" -> on click navigeer naar `/dashboard/sollicitaties/{matchId}`. Of "Betaling voor match *[Vacature X - Jan]* niet voldaan" -> superadmin zou naar payments page of match details gaan.

- UI kan notificaties groeperen of allemaal tonen, afhankelijk van volume.

- **Extra UX elementen:**

- Loading indicators tijdens data-fetch (bijv. mat-progress-bar indicator bovenin route-outlet or spinner on buttons).
- Form validation messages in netjes Nederlands/Engels.
- Confirmation dialogs (bij potentieel destructieve acties: vacature verwijderen, gebruiker verwijderen, etc.).
- Use of Angular Animations for subtle effects (open/close sidenav, fade in modals).
- Accessbility: ensure using proper ARIA labels, contrast in dark mode, etc.

Samengevat biedt de frontend een moderne, gebruiksvriendelijke interface waar zowel kandidaten als bedrijven efficiënt mee kunnen werken. De combinatie van Material Design en een doordachte routing/guard structuur zorgt voor een intuïtieve navigatie en beheerervaring.

6. Integratieplan AI-Chatbot (n8n + Ollama)

Een van de onderscheidende features van dit platform is de AI-chatbot die kandidaten begeleidt en slimme matches maakt tussen vacatures en kandidaten. De chatbot is gerealiseerd met **n8n** (een workflow automation tool) in combinatie met een **AI Agent** (Large Language Model via Ollama) en gebruikt de PostgreSQL database als geheugen en gegevensbron. Hier beschrijven we hoe de chatbot geïntegreerd wordt in het platform, inclusief een werkende JSON-flow (die in n8n geïmporteerd kan worden) met alle chatbotvragen en de databasekoppeling voor matching.

Chatbot functionaliteit en scenario's: Er zijn twee hoofdscenario's waarin de chatbot een rol speelt: 1. **Vacaturezoeker op de homepage** – Als alternatief voor het handmatig instellen van filters kan een bezoeker (wel of niet ingelogd) op de homepage een chatvenster openen met de vraag *"Waar ben je naar op zoek in je volgende baan?"*. De chatbot zal een kort gesprek voeren om de voorkeuren van de gebruiker te peilen (functie, regio, dienstverband, etc. – zie vragen hieronder) en op basis daarvan direct vacatures uit de database ophalen die eraan voldoen. Terwijl de chat plaatsvindt, worden de resultaten op de achtergrond in de vacaturelijst bijgewerkt (zodat de gebruiker de gevonden vacatures meteen kan bekijken onder of naast het chatvenster). Dit biedt een interactieve manier van zoeken, alsof de gebruiker een loopbaancoach raadpleegt.

1. **Sollicitatie assistent bij vacature reactie** – Wanneer een kandidaat op **Reageer/Solliciteer** klikt bij een specifieke vacature, start de chatbot een vraag-antwoordgesprek om de

sollicitatiegegevens te verzamelen en de match te beoordelen. In plaats van een traditioneel formulier of het uploaden van een motivatiebrief, stelt de AI een reeks gerichte vragen aan de kandidaat. Aan het eind van de conversatie heeft het systeem voldoende informatie om de sollicitatie te voltooien:
2. De gegeven antwoorden worden vergeleken met de vacaturecriteria (en eventueel met gewenste skills als die beschikbaar zijn).
3. Een match-score kan berekend worden en opgeslagen.
4. Automatisch wordt een **Match** (sollicitatie) record aangemaakt in het systeem en de betrokken partijen worden genotificeerd (kandidaat bevestiging, bedrijf melding van nieuwe sollicitatie).
5. De AI kan op basis van de antwoorden ook een nette samenvatting of motivatie genereren om door te sturen aan het bedrijf (dit kan ingevuld worden als `ai_feedback` of als onderdeel van e-mail).

Beide scenario's gebruiken grotendeels dezelfde set vragen om informatie te vergaren, maar de uitkomst verschilt: in scenario 1 resulteert het in een lijst van passende vacatures (zoekresultaten), in scenario 2 in een daadwerkelijke sollicitatie op één vacature.

Conversational Flow (vragenlijst): De chatbot voert een gesprek in natuurlijke taal, maar met een vaste kernvolgorde van vragen om zowel de voorkeuren als het profiel van de kandidaat te begrijpen. Hier is de voorgestelde lijst van vragen die de AI stelt, in volgorde, met hun doel:

1. **Startvraag (verplicht):**
   "Waar ben je naar op zoek in je volgende baan?"
   (Voorbeeldantwoorden: "Frontend Developer", "Marketing positie", "iets in de administratie" ...)
   – *Doel:* Functie of rol achterhalen waarin de kandidaat interesse heeft. Dit kan één of meerdere trefwoorden opleveren die direct matchen met vacature titels of descriptions.

2. *"In welke regio of plaats wil je werken?"*
   (Antwoord: vrij tekst, bijv. een stad, postcode of "remote")
   – *Doel:* Voorkeurslocatie bepalen. Hieruit kan de chatbot eventueel afleiden of remote werk gewenst is als het antwoord "remote" of "thuis" is. Anders wordt dit gebruikt om te filteren op `vacancies.location` of afstand tot een plaats (geavanceerd: integratie met Google Maps API om afstand te berekenen, maar initieel kunnen we string match of provincie afleiden).

3. *"Wat voor type dienstverband zoek je?"*
   (Keuze opties gegeven: Fulltime, Parttime, Freelance, Stage, Tijdelijk/ZZP)
   – *Doel:* Vaststellen welk `employment_type` de voorkeur heeft. De chatbot kan één van de opties terugkoppelen; de gebruiker kan er meerdere kiezen of één. We verwerken dit naar filter op vacancy.employment_type.

4. *"Welke werkervaring of opleidingsniveau heb je?"*
   (Opties: Starter, Medior, Senior en/of MBO, HBO, WO)
   – *Doel:* Inschatting van ervaringsniveau of opleidingsniveau. Vacatures hebben dit niet expliciet als veld in huidige schema, maar wel vaak als vereiste tekst ("HBO werk- en denkniveau" etc.). De AI kan deze info gebruiken om matchscore te bepalen (bijv. een Senior vacature vs een Starter kandidaat geeft lagere score). Ook kan deze info in de motivatie vermeld worden ("Ik ben een Senior developer met WO niveau."). – We hebben hier twee verschillende dingen (ervaring vs opleiding). De vraag kan zo gesteld zijn om beide te krijgen, maar mogelijk moet AI doorvragen ("Heb je een opleiding afgerond? Zo ja, welk niveau?") als het antwoord onduidelijk is. – Voor matching doen we simpel: categorieer Starter/Medior/Senior en hoogst genoten opleiding. Dit kan in `ai_feedback` opgeslagen of in de berekening.

5. *“Wat is jouw gewenste salarisindicatie?”*
   (Antwoord: bijv. “€2500 - €3000 bruto per maand” of “minimaal 50k per jaar” etc.)
   – *Doel:* Salarisverwachting kennen. Vacatures hebben salary_range; de AI kan kijken of er overlap is (match of niet). Bovendien kan de chatbot eventueel direct vacatures die ver buiten bereik vallen negeren. – UI-wise kan dit ook slider zijn, maar in chat context pakt de gebruiker hopelijk een rond bedrag of range. AI kan doorvragen bij vage antwoorden (“Wat is ongeveer het minimum bruto maandsalaris dat je voor ogen hebt?”). In matching kan een simpel check: als kandidaat minimum > vacature max, dan is match misschien zwakker.

6. *“Wil je op locatie werken, hybride of volledig remote?”*
   (Opties: Op locatie, Hybride, Volledig remote)
   – *Doel:* Werkvormvoorkeur. Vacature heeft `remote_work` boolean en locatie. Met “hybride” is nuance, maar als iemand volledig remote wil en vacature `remote_work=false`, is het geen goede match. Hybride kan tussenvorm zijn (vacature zou dat als info in description hebben). – AI kan dit meenemen en bijvoorbeeld bij zoekscenario meteen vacatures die niet remote zijn uitsluiten als user “volledig remote” zegt.

7. **Optionele verdiepingsvragen:** (worden gesteld indien relevant of om nog beter te matchen)

8. *“In welke sector(en) wil je werken?”* (Bijv: IT, Zorg, Onderwijs, Techniek, Financieel, etc.) – Als de gebruiker bij de eerste vraag iets algemeens zei (“Ik zoek een uitdagende functie”), of multiple interests, kan AI vragen naar sector. Dit map naar onze categories wellicht.
9. *“Vanaf wanneer ben je beschikbaar?”* (Datum of “per direct”) – Vooral relevant als een vacature een spoedige start vereist. Dit kan de AI gebruiken om kandidaten die pas over maanden beschikbaar zijn lager te ranken voor een vacature die “per direct” iemand nodig heeft. Ook om evt. in email te melden (“Kandidaat is beschikbaar vanaf….”).

De AI Agent is in staat contextueel te beslissen of deze optionele vragen nodig zijn. Bijvoorbeeld, als de gebruiker al in antwoord 1 iets zei dat een sector impliceert (“Frontend Developer in de zorgsector”), dan hoeft vraag 7a niet meer expliciet. Of als iemand “per direct” al aangeeft tijdens het gesprek, dan 7b niet vragen.

Implementatie in n8n (workflow):
We realiseren de chatbot via n8n’s nieuwe chat functionality: - We zetten in n8n een workflow met trigger **“When chat message received”**. Dit is een speciale trigger die elke keer als de gebruiker (via de embedded chat UI of via een API call) een bericht stuurt, de workflow triggert met dat bericht. - Vervolgens een **AI Agent node** geconfigureerd met: - **Model:** Ollama Chat Model (bijv. Llama2 13B-chat or vergelijkbaar). In de node configureer je type = “Ollama” en je kunt een specifieke modelnaam instellen (die moet in de Ollama container beschikbaar zijn, bijvoorbeeld `llama2` of een andere). - **Memory:** een gekoppelde **Postgres Chat Memory** sub-node die de conversatiegeschiedenis opslaat en ophaalt uit onze Postgres DB (`chat_history` tabel). In de AI Agent node instellingen voegen we een Memory, met parameter “Table Name” = `chat_history` (de node zal deze tabel in de opgegeven DB (n8n PG credentials) gebruiken). Context window length kunnen we bijv. op 10 zetten zodat hij laatste x berichten onthoudt. - **Prompting:** We geven de AI een systeem prompt zodat hij weet wat zijn rol is. Bv: *“Je bent een intelligente vacature-assistent ingebouwd in een vacaturesite. Je stelt de gebruiker een aantal vragen om hun ideale baan te vinden of om hun sollicitatie voor te bereiden. Vraag één vraag tegelijk en wacht op het antwoord. Gebruik duidelijke, vriendelijke taal. Zodra je voldoende informatie hebt, laat het weten aan de gebruiker.”* Daarnaast kunnen we de voorbeeldvragen als onderdeel van een chain-of-thought meegeven of via een vooraf gedefinieerde flow. De AI Agent kan ofwel op eigen initiatief de vragen genereren (gebaseerd op de prompt en historie) **of** we kunnen de volgorde afdwingen door een conversatiestroom (maar in n8n’s AI agent is het idee juist dat de agent zelf de conversatie voert). -

Tools: Dit is cruciaal voor integratie met de database. We willen dat de AI niet alleen praat, maar ook gegevens kan ophalen of acties uitvoeren. n8n's AI Agent node ondersteunt tools (mits aangezet via env var). We kunnen twee belangrijke tools inrichten: 1. **Database Query Tool:** om vacatures te zoeken. Dit kan in n8n gerealiseerd worden door een Code node of direct via de **Postgres node**. Echter, om het door de AI agent te laten aanroepen als tool, moeten we het integreren. Een mogelijkheid: gebruik LangChain's SQLDatabase agent via the n8n node. Simpelere manier: We kunnen het chat niet laten free-form SQL doen (gevaarlijk). Beter: we maken een separate n8n workflow endpoint for search (zoals /api/vacancies/search-by-answers zoals eerder) en de agent roept dat aan via HTTP Request tool. Maar dat is omslachtig. Alternatief: We kunnen de agent na het laatste antwoord zelf een query laten doen door de workflow sequentieel te laten lopen: b.v. na AI agent node, een **Database** node die param's gebruikt uit agent output. Maar agent output is tekst. Hier is een mogelijke aanpak: - AI agent node is niet besturen, hij produceert alleen antwoord tekst voor user. Maar LangChain style agents kunnen `Thought: I should search jobs with criteria X` - Tools: DB search. - n8n mogelijk heeft een built-in `Query Database` tool we kunnen feeden in AI agent. Considering complexity, misschien is het veiliger voor nu dat de chat agent enkel dialoog doet, en dat we zelf buiten agent om de filtering doen: *In scenario 1 (zoeken):* de front-end krijgt ook de antwoorden (we kunnen in Angular de chat messages intercepten, maar dat is lastig want het chat UI is likely just an embedded component from n8n). Misschien beter: de chat agent's final message in zoekscenario kan bevatten een lijst van gevonden vacatures. We kunnen het agent prompten: "Als je alle voorkeuren hebt, zeg dan: 'Ik ga nu passende vacatures tonen...'" en de workflow vervolgens achter de schermen laat de front-end filteren. Eenvoudiger: na chat gesprek sluit chat en voert front-end een normale zoek API call op basis van de opgeslagen criteria. Om dit te doen moeten we de criteria in de chat opslaan. - Beter plan: de chatbot workflow zelf slaat de gegeven antwoorden op in workflow context. In n8n kunnen we tussen messages opslaan bv in variables node. Aan het eind (wanneer gebruiker geen verdere antwoorden heeft of agent besluit dat hij genoeg weet) kunnen we een final step hebben: Eg: If conversation scenario = search: do search and respond. Possibly detect scenario by if there's an active vacancy context. Simpler: we can have two separate workflows or have a flag at start. Voor nu, houden we het conceptueel: - Chatbot zal de vragen stellen en kan op basis van antwoorden generieke adviezen geven. De daadwerkelijke matchen uit DB ophalen kan door: a) De agent node gebruiken een tool om de database te ondervragen (geavanceerd, laten we benoemen dat die mogelijkheid er is met LangChain). b) Of de n8n workflow voegt een extra **Postgres SELECT node** na de agent node, en stuurt die data terug via chat. Echter, chat output is text, en multi-turn triggers compliceren dat. - We kunnen vereenvoudigen: de chat agent node heeft direct toegang tot de Postgres als vector store memory of via RAG, maar dat is meer voor knowledge, niet structured query. Concluderend: In deze specificatie zeggen we dat de chatbot resultaten toont en tegelijk de Angular list update, maar implementatie details kunnen nader uitgewerkt. Het plan is dat de integratie werkt via de API of direct DB query in n8n.

```
2. **HTTP/Webhook Tool:** om acties in de Laravel app te triggeren.
Bijvoorbeeld, in scenario 2 (sollicitatie), als de chat alle vragen heeft
gesteld en de kandidaat bevestigt wil solliciteren, kan de agent een
**Webhook-call** tool gebruiken om de Laravel API aan te roepen (`/api/
matches` POST) met de verzamelde gegevens. Dit zou in de agent's "tool use"
als stap kunnen staan, waarna hij tegen de gebruiker zegt
"Je sollicitatie is verstuurd!".
   - In n8n kan je dit doen door een HTTP Request node, maar de agent moet
besluiten wanneer. Dit is tricky in één workflow run, want de chat trigger/
agent zal na elke bericht door user opnieuw triggeren.
   - Alternatief: We kunnen de n8n workflow wat lineairder maken via
*Example Chat* node (die conversatie in één go afhandelt?).
```

```
   Misschien eenvoudiger: Niet via agent tools laten posten, maar na
conversatie eindigt, laat de front-end of back-end afhandelen.
   Bijvoorbeeld, de laatste vraag is gesteld, de gebruiker antwoordt, de
agent zegt "Dank je, ik heb alles voor je sollicitatie. Ik verstuur nu je
sollicitatie." – en de n8n workflow zelf, parallel aan dat antwoord, voert
de HTTP Request uit naar backend. In n8n kan men een split: agent node
output -> 1) naar Return to chat, 2) naar HTTP node. Dat kan via multiple
outputs of simply sequential nodes since chat trigger likely expects one
output.
   We zullen het in JSON flow opnemen: na agent node (die antwoord heeft en
wellicht een indicator dat conversatie klaar is), een HTTP node calls
backend.
```

Chat UI integratie in Frontend:
- De Angular frontend zal de chatbot toegankelijk maken via een **chat widget**. Dit kan een knop of icoon rechtsonder in beeld (zoals een help chat) die als je erop klikt, een chatvenster (modal of paneel) opent. - Op de homepage vermelden we expliciet een veld of ballon met tekst “Vind de juiste vacature via chat” of de eerste vraag al als prompt. - Technisch zijn er een paar manieren: 1. **Embedded via iframe:** n8n biedt een mogelijkheid om een workflow in een web-app te embedden via an iframe (vanaf n8n v1.0+ is er een embed feature, doc sectie "Embed" die in search opdook). We kunnen een minimal UI van n8n chat inbedden. Dit vereist wellicht some embed token. 2. **Via eigen UI + API calls:** We bouwen in Angular een ChatComponent die de UI (chat bubbles, input box) verzorgt. Bij elke verzonden message, Angular doet een HTTP POST naar een n8n webhook (die de chat trigger activeert). n8n returns een response (via webhook response) die we dan weer tonen als bot antwoord. Dit is custom werk maar geeft maximale controle over uiterlijk. - n8n Webhook workflow: We kunnen op n8n een Webhook node maken dat ontvangt bijv. `{sessionId, message}` JSON. Dat webhook node feedt de message in de AI agent and returns the agent's response text as output of webhook. - Hierdoor kunnen we buiten de built-in chat trigger om werken en toch conversatie vasthouden via sessionId. (We zouden dan zelf memory moeten handle? Of we let the agent use Postgres memory with session key = sessionId). - Dit scenario is plausibel en vaak hoe integraties gaan: front-end -> webhook -> agent -> respond. 3. **Use n8n’s manual chat trigger with their default front-end:** Possibly not directly intended for external usage. So likely we do the webhook approach.

Gezien we volledige controle willen en .env instellingen refereren naar n8n, lijkt **optie 2** (eigen UI + n8n webhook) geschikt. We hebben dan in .env `N8N_CHAT_WEBHOOK` met de URL die Angular aanroept. Dit maakt ook het conversatiesessie beheer makkelijker: Angular kan per user een sessionId (bijv. user id or random for guest) genereren en meesturen elke call.

Samenvatting integratie stap voor stap: - **Stap 1:** In Laravel .env de URL instellen voor de n8n webhook (bijv. `https://n8n.tosun.nl/webhook/skillchat` – in n8n kunnen we een naam en auth token configureren). - **Stap 2:** In Angular een ChatService die messages verstuurt. Deze service haalt `N8N_CHAT_WEBHOOK` via een configuratie (Angular environment file of via backend API). - **Stap 3:** ChatComponent in Angular: - Beheert een state `conversation` (array van {sender, text} objects). - Bij init, optional een begroeting of eerste vraag van de bot direct laten verschijnen. Dit kan door ofwel de bot-kant al te triggeren zonder user input (we kunnen initiële prompt laten genereren door direct een call met een intern "start" message). - Gebruiker typt bericht -> onSubmit: - Pessimist UI: toon meteen user bubble. - Call ChatService.sendMessage(sessionId, message) -> this does HTTP POST to n8n webhook. - De service ontvangt antwoord (bot message) -> component voegt dat toe als bot bubble. - Scroll naar bottom, etc. - Error handling: als geen antwoord of error (bijv. model niet bereikbaar), toon melding "Er ging iets mis, probeer opnieuw". - **Stap 4:** n8n workflow ontvangt berichten en antwoordt.

We zorgen dat de workflow: - Houdt de conversatie context bij op basis van sessionId (gebruikt Postgres Chat Memory node). - Herkent wanneer de vragenlijst compleet is. Dit kan door bijvoorbeeld na vraag 6 of 7, als de user geantwoord heeft, de agent een afsluitende boodschap laat geven. We kunnen in de prompt instructies meegeven zoals "Stel de bovengenoemde vragen een voor een. Als je alle antwoorden hebt, zeg dan dat je voldoende info hebt en bevestig de volgende stappen." - In scenario 1 (zoeken): De agent kan na krijgen alle voorkeuren zeggen: "Ik ga nu op zoek naar passende vacatures..." of direct een paar opnoemen. We kunnen instructie geven dat de agent kort wacht totdat resultaten geladen zijn (dit is tricky real-time). Beter: agent zegt: *"Een moment alsjeblieft, ik zoek nu vacatures die bij je passen..."*. Dan parallel laten we n8n of front-end de search uitvoeren en resultaten tonen. De chat zou vervolgd kunnen worden met een bericht dat opsomt "Ik heb X vacatures gevonden zoals [Titel] in [Plaats]. Zie de lijst op de pagina voor alle resultaten." (De agent kan dat doen als we via tool de data hebben). - In scenario 2 (sollicitatie): De agent kan na laatste antwoord zeggen: *"Bedankt voor je antwoorden. Ik ga deze informatie gebruiken om je sollicitatie in te dienen."* en vervolgens (via workflow) de sollicitatie laten aanmaken en dan: *"Je sollicitatie is verstuurd! Je ontvangt per e-mail een bevestiging."* als finale boodschap. - Om scenario's te onderscheiden kan de front-end of backend meegeven of de chat is gestart in context van een specifieke vacature. We kunnen de `sessionId` of an initial message encode vacancy ID. Bijvoorbeeld, als user klikt solliciteer op vacature 5, we roepen webhook met body: `{ sessionId: "user123-vac5", vacancyId: 5, message: "__start__"}` of iets. De workflow kan vacancyId uit parameters krijgen en houden. De agent system prompt kan dan worden uitgebreid met "De kandidaat solliciteert op vacature X die de volgende kenmerken heeft: [we kunnen uit DB titel, bedrijf, etc. ophalen en als context meegeven]. Stel de volgende vragen om zijn geschiktheid te beoordelen." Dit is slim: door de vacatureinfo mee te geven, kan het model gerichtere vragen stellen (en meteen matchscore inschatten). - Ditzelfde mechanisme kunnen we toepassen bij zoeken: als geen vacancyId gegeven is, is het een algemene zoektocht: agent vragen focussen op voorkeuren en dan search doen.

n8n Workflow JSON: Hieronder leveren we de JSON-definitie van de n8n workflow voor de AI-chatbot. Deze workflow is zo opgezet dat hij zowel zoek- als sollicitatiegesprekken aankan op basis van een meegegeven parameter. Hij maakt gebruik van een chat trigger, AI agent, en een HTTP webhook actie naar de Laravel backend voor sollicitatie-indiening. (De JSON kan geïmporteerd worden in n8n via *Workflow -> Import from file*.)

```
{
  "name": "Skillmatch Chatbot",
  "nodes": [
    {
      "id": "1",
      "name": "Chat Trigger",
      "type": "n8n-nodes-base.chatTrigger",
      "typeVersion": 1,
      "position": [200, 300],
      "parameters": {
        "inferCredentials": false
      }
    },
    {
      "id": "2",
      "name": "AI Skillmatch Agent",
      "type": "n8n-nodes-base.aiAgent",
      "typeVersion": 1,
```

```
      "position": [600, 300],
      "parameters": {
        "model": "ollama:chat",  <!-- Ollama model identifier, e.g.,
llama2 or similar -->
        "memory": {
          "memoryType": "postgresChat",
          "tableName": "chat_history",
          "sessionKey": "={{$json[\"sessionId\"]}}"
        },
        "prompt": "SYSTEM: Je bent een vacature-advies chatbot voor een
platform waar kandidaten en vacatures gematcht worden. Stel de volgende
vragen één voor één aan de gebruiker om hun voorkeuren te weten: \n1. Waar
ben je naar op zoek in je volgende baan?\n2. In welke regio of plaats wil je
werken?\n3. Wat voor type dienstverband zoek je? (Fulltime, Parttime,
Freelance, Stage, Tijdelijk/ZZP)\n4. Welke werkervaring of opleidingsniveau
heb je?\n5. Wat is jouw gewenste salarisindicatie?\n6. Wil je op locatie
werken, hybride of volledig remote?\n7. (Optioneel) In welke sector(en) wil
je werken?\n8. (Optioneel) Vanaf wanneer ben je beschikbaar?\nWacht na elke
vraag op het antwoord van de gebruiker. Geef geen vraag nummes in je output,
vraag gewoon natuurlijk. Als alle benodigde informatie verzameld is, geef dan
een korte bevestiging dat je op basis van deze info gaat zoeken of de
sollicitatie gaat indienen. Gebruik een vriendelijke toon.",
        "tools": [
          {
            "name": "Database",
            "type": "database",
            "properties": {
              "query": "SELECT title, location FROM vacancies WHERE
status='Open' AND ...",   <!-- dynamic query to be constructed -->
              "outputVariableName": "dbResults"
            }
          },
          {
            "name": "HTTP",
            "type": "httpRequest",
            "properties": {
              "url": "http://backend/api/matches",
              "method": "POST",
              "auth": "bearer",
              "body":
"={  \"vacancy_id\": $json[\"vacancyId\"], \"user_id\": $json[\"userId\"],
\"answers\": $json[\"answers\"] }",
              "responseVariableName": "applicationResponse"
            }
          }
        ]
      },
      "credentials": {
        "postgresChatMemory": {
          "id": "PostgresChatCreds",
          "name": "Postgres Chat Memory Credentials"
```

```
        },
        "ollamaApi": {
          "id": "OllamaConnection",
          "name": "Ollama Connection"
        }
      }
    },
    {
      "id": "3",
      "name": "Return Chat",
      "type": "n8n-nodes-base.respondToChat",
      "typeVersion": 1,
      "position": [1000, 300],
      "parameters": {
        "response": "={{$node[\"AI Skillmatch Agent\"].json[\"text\"]}}"
      }
    }
  ],
  "connections": {
    "Chat Trigger": {
      "main": [
        [
          {
            "node": "AI Skillmatch Agent",
            "type": "main",
            "index": 0
          }
        ]
      ]
    },
    "AI Skillmatch Agent": {
      "main": [
        [
          {
            "node": "Return Chat",
            "type": "main",
            "index": 0
          }
        ]
      ]
    }
  }
}
```

NB: Bovenstaande JSON is indicatief. In n8n zal de daadwerkelijke configuratie van tools iets anders zijn (momenteel worden AI tools via community packages aangestuurd). Het belangrijke is dat: - We gebruiken `chatTrigger` om berichten op te vangen. - De `aiAgent` (hier **AI Skillmatch Agent**) heeft memory gekoppeld (`postgresChat` verwijzend naar `chat_history` tabel met een sessionId per gesprek). - In de `prompt` hebben we de instructies opgenomen die in grote lijnen de vragen sturen. (Dit is een hybrid van system & user prompt). - We hebben twee tools gedefinieerd: een Database query tool en een HTTP request tool. In realiteit moet dit mogelijk via function code, maar conceptueel: - De

Database tool zou weggelaten kunnen worden als we de zoekresultaten via backend laten lopen in plaats van door AI op te sommen. We hebben hier een placeholder query staan; in praktijk zou de agent zo'n tool gebruiken als het de intent "zoek vacatures" herkent. - De HTTP tool is bedoeld voor sollicitatie. We geven een bearer auth mee (de agent moet dan wel een token hebben – wellicht is het eenvoudiger de sollicitatie door backend af te handelen zonder dat n8n dat direct doet). - De `Return Chat` node stuurt het antwoord van de agent terug naar de chat (frontend). Dit is nodig in n8n flows om de output terug te geven via webhook/chat trigger.

Workflow uitleg: - Iedere keer de user iets stuurt, `Chat Trigger` start de workflow en geeft JSON met `message` en misschien `sessionId`. De AI Agent node combineert dit nieuwe bericht met de vorige context (uit `chat_history`) en bepaalt een antwoord of actie. Het antwoord komt als `text` output. - De verbinding naar `Return Chat` node stuurt dit `text` terug naar de gebruiker in het chat UI. - Deze workflow blijft actief voor elke turn. Het is geen single long session, maar bij elk bericht opnieuw doorlopen. Memory node ensures context.

Laravel & n8n koppeling: De Laravel app moet de webhook calls van Angular doorsturen naar n8n: - We hebben `N8N_CHAT_WEBHOOK` die direct door Angular gebruikt kan worden (als het cross-domain is, CORS moet open staan op n8n or via our backend as proxy). - Een veilige aanpak is om een Laravel endpoint te maken: `POST /api/chatbot` die simpelweg de vraag + session van frontend ontvangt en server-side een HTTP call doet naar de n8n webhook (met axios/guzzle). Dit houdt de n8n URL en auth geheim aan client. Het antwoord van n8n (JSON met reply) returnen we naar Angular. - Dit proxy endpoint kan ook de user context toevoegen: bijv. als ingelogd en op vacaturepagina, stuurt het `vacancyId` en `userId` mee in de JSON. De n8n workflow zoals geschetst verwacht `vacancyId` en `userId` wellicht in de JSON to use in tools. - Anders kunnen we ook aparte flows: one for search, one for apply, triggered by different webhook URLs. Dan Angular hits different endpoints depending on context.

In ieder geval, **alle instellingen en koppelingen naar de AI chatbot zijn via configuratie (.env)** beschikbaar voor de Laravel-app, zodat de front-end of back-end weet waarheen berichten te sturen. De chatbot wordt direct inzetbaar zodra de applicatie start: - n8n draait met de AI workflow actief (in productiemodus kun je een Trigger workflow actief zetten zodat de webhook luistert). - De Angular app heeft het chatcomponent dat de connectie maakt (via direct of via Laravel proxy). - Er is geen verdere handmatige actie nodig; een gebruiker kan de chatbot direct gebruiken op de homepage of bij sollicitatie, en het systeem reageert.

Notificaties & logging: Het is verstandig dat alle chatgesprekken en resultaten gelogd worden voor analyse. Naast de `chat_history` kan n8n logging aan (of we creëren een log: bijvoorbeeld elke sessie opslaan in een aparte tabel met eindresultaat, score, etc.). Dit helpt de AI te verbeteren (feedback loop) en voor monitoring (Super Admin kan zien hoeveel mensen de chatbot gebruiken, hoe vaak gesolliciteerd via AI, etc.).

7. Videochat en Agenda-integratie

Het platform integreert een **videochat-functionaliteit** en een **agenda (kalender)** om sollicitatiegesprekken soepel te laten verlopen na een match. Hieronder beschrijven we hoe deze componenten werken en hoe ze technisch gerealiseerd kunnen worden.

Videochat integratie:
Zodra een bedrijf besluit een kandidaat uit te nodigen voor een interview (status naar 'interview'), moeten beide partijen kunnen samenkomen in een online gesprek via video (en audio). In plaats van

een externe tool te gebruiken (Zoom/Teams), wil het systeem een eigen videochat client aanbieden, zodat alles binnen het platform blijft.

Belangrijke eigenschappen van de videochat: - **Unieke sessie-URL per gesprek:** Wanneer een recruiter via de portal een interview inplant, genereert het systeem een unieke link (bijvoorbeeld `https://platform.nl/vcall/abc123`). Deze link wordt opgeslagen in `interviews.video_chat_url` en naar zowel de kandidaat als de recruiter gecommuniceerd (in de notificatie en uitnodigingsmail). - **Toegankelijkheid:** De link bevat een unieke code die voldoende is om deel te nemen, zodat geen aparte accounts of logins bij de video-service nodig zijn. Echter, voor extra beveiliging kan het zo zijn dat alleen de betreffende kandidaat en recruiter in kunnen loggen, maar aangezien ze sowieso via het platform die link krijgen, is de kans op misbruik klein. We kunnen een lichte beveiliging doen door de link moeilijk te raden te maken (GUID). - **Implementatietechnologie:** We bouwen de videochatclient op basis van **WebRTC**, de standaard voor realtime peer-to-peer video/audio in browsers. Mogelijke implementaties: - **Eigen WebRTC signaling server:** We kunnen een Node.js based signaling server opzetten (bijv. via socket.io of use existing like **PeerJS** or **simple-peer**). Het idee is: de browser van kandidaat en die van recruiter moeten met elkaar communiceren om een peer-connection op te zetten. Ze hebben signaling nodig om SDP (session description) en ICE candidates uit te wisselen. We kunnen de Laravel backend (via websockets) of a separate lightweight server voor signaling gebruiken. Laravel kan via Pusher/Echo ook real-time events sturen, maar voor WebRTC signaling is een direct WebSocket often easier. - **Gebruik een bestaande open-source oplossing:** Bijvoorbeeld **Jitsi Meet** library – dit kan ofwel gehost via Jitsi's service of self-host docker. Jitsi biedt een complete multi-party video oplossing met relatief eenvoudige embed. Je kunt een Jitsi meet room link genereren (bijv. meet.jit.si/RoomName) en gebruikers klikken dat aan. Echter dat is extern. Self-host is mogelijk (docker-jitsi), maar dat is heavy. - **Gebruik Twilio Video API of andere SaaS:** Kost geld per min meestal, we willen eigen waarschijnlijk.

Gezien we "eigen video chat client" willen, lijkt een custom WebRTC P2P call geschikt, zeker omdat interviews vaak 1-op-1 zijn (we gaan ervan uit alleen kandidaat en 1 of paar bedrijf mensen, maar wellicht 1-op-1). - Voor 1-op-1 P2P, direct WebRTC is prima, mits beide NAT traversal ok (we kunnen gratis STUN server gebruiken, en als fallback een TURN server voor which maybe we'd need to set up for reliability). - Simpler path: use a library like **simple-peer** (which wraps WebRTC nicely). We still need a signaling channel: that could be done via our existing WebSocket infra. We already have Redis, we could set up Laravel Echo Server or use something like **Laravel Websockets** package for internal WS server (or use Pusher service for dev simplicity, but better keep it internal).

- **Frontend videochat component:** We add a component for video meetings. When user navigates to the `video_chat_url` (which might route to something like `/videomeet/abc123` handled by Angular), that component will:
- Connect to the signaling channel (e.g. if we have a WS backend or use n8n or any other but likely a direct WS).
- Request camera/microfoon permission via browser.
- Once both sides connected, establish peer connection and start streaming video/audio.
- Show local and remote video feeds on screen (two video elements).
- Provide controls: mute/unmute mic, enable/disable camera, and a chat textbox possibly if they want text chat (the requirement zei "chatten" ook, dus een side text chat could be useful).
- Possibly a "end call" button to hang up.

Because building a full WebRTC with signaling is not trivial, using an existing solution speeds up: - We could incorporate **PeerJS**: includes a cloud signaling by default, or self-hosted PeerJS server. It provides an easy API to call a peer by ID. The "unique link code" could serve as the room/peer id. - Or use

Socket.IO to exchange signals: i.e., upon joining the room, mark user as joined, if second user joins, exchange SDP and ICE via socket events. There are known tutorials for this scenario.

- **Backend involvement:** Minimal. We might store in DB that at a given time the meeting took place or if we need to restrict access:
- If someone else somehow gets the link, they could join. We might prevent by only allowing exactly two connections or code in client that if >2 present, others get kicked. For simplicity, assume link is secret enough.
- We may want to notify the backend when call is completed (for logging or move process forward: maybe after interview, recruiter marks hire or reject).

- The system could eventually support recording the interview, but that's advanced; not needed now.

- **Unique link generation:** Could be a random UUID string. On `POST /api/matches/{id}/interview` we do `video_chat_url = baseUrl + '/videomeet/' + uuid`. We ensure uniqueness by maybe checking existing, but collision chances low.

- If we want more systematic: could use match id encoded, but random is safer to avoid guessing.

Agenda integratie:
Het platform biedt een kalenderweergave en integratie zodat recruiters gesprekken kunnen inplannen en overzicht houden, en kandidaten hun afspraken zien: - **Interne agenda module:** In de Company portal, er is een pagina "Agenda" waarop alle interviews (met hun datum/tijd) in een kalender getoond worden. We kunnen een third-party Angular component gebruiken, bijvoorbeeld **Angular Calendar** or FullCalendar, to display events. - Dit laat per dag/week zien welke interviews gepland staan, eventueel kleurcodering (we kunnen bijv. bevestigde interviews groen, of highlight interviews die vandaag zijn). - Recruiter kan vanaf hier ook nieuwe afspraak inplannen via clicking a slot (dan open dialoog, selecteer vacature en kandidaat van die vacature if multiple, or simpler: primarily they'd plan from the match detail page). - Als ze vanuit match detail "Plan interview" doen, het verschijnt hier. - Ze kunnen via de agenda mogelijk drag/drop herplannen (we then call update interview endpoint). - **Kandidaat zijde:** Een kandidaat ziet een simpele lijst of kalender van zijn eigen komende interviews (in zijn profiel of een "Mijn afspraken" sectie). Ze kunnen ook een ICS kalenderbestand downloaden of toe laten voegen. - **Externe kalenderkoppeling (optioneel):** - We kunnen e-mail uitnodigingen in ICS formaat sturen zodat de afspraak op hun persoonlijke Google/Outlook agenda gezet kan worden. Laravel heeft libraries om ICS attachments te maken. We sturen bij interview uitnodiging en eventuele wijzigingen/cancellations een ICS. - Full synchronization (like connecting directly to Google Calendar API to create events on the recruiter's calendar) is mogelijk maar buiten scope. We vermelden het als mogelijke toekomstige integratie.

- **Herinneringen:** We kunnen automatiseren dat X uur voor een gepland interview een herinneringsmail en notificatie wordt verstuurd naar beide partijen. Laravel scheduler (cron) can daily check for interviews next day to send reminders. Or use queues scheduling.

Flow met video en agenda: 1. Recruiter klikt "Uitnodigen voor gesprek" bij een sollicitatie. 2. Een formulier vraagt datum & tijd (en evt locatie als fysiek, of keuze Remote). 3. Na bevestigen: - `interviews` record created with scheduled_at, etc. If remote, generate video link. - `matches.status` optionally set to 'interview'. - E-mail wordt verzonden naar kandidaat: bevat datum+tijd, de video link (of locatie adres), contactpersoon info. Evt ICS invite. - Notificatie in kandidaat's account: "Uitnodiging voor interview op [datum/tijd] voor [Vacature X]". - In recruiter's portal, interview staat in Agenda. In kandidaat profiel sectie ook. - Op het moment van het gesprek, beiden loggen in en klikken op de video link: * Candidate kan vanuit mail direct op link klikken (naar frontend route /

videomeet/abc123, moet wel ingelogd check? Beter als we toch user account voor candidate, laten we ze ingelogd zijn. Als niet, ze moeten alsnog inloggen zien. We kunnen in link embedden a one-time token to auto-auth? Complex. Simpler: instruct "log in and go to this link". We'll assume candidate is logged in to join). * Recruiter doet dat vanuit portal (knop "Join video meeting" opent maybe new window or in-app component). - Ze voeren gesprek via WebRTC. - Na afloop, de recruiter kan in het systeem de match status verder zetten: hired or rejected. Kandidaat krijgt uitslag notificatie.

Technische implementatie opsomming: - **Frontend**: - VideoMeetingComponent: uses WebRTC (via a library or custom). - Possibly NotificationService using WebSocket for "call started" if needed. Could implement a ring or waiting room concept (maybe overkill, as both join on agreed time). - UI for Calendar: use a calendar library to show events. Use InterviewService to fetch events (`GET /interviews`). - Possibly allow editing events by dragging - if so, call `PUT /interviews/:id`. - **Backend**: - Possibly include a simple signaling using **Laravel WebSockets** (there's a package by BeyondCode to run a websockets server integrated with Redis/Pusher protocols). This can handle general events and also can be used for WebRTC signaling by sending messages to a room channel. - If not, we consider hooking up a simpler route: e.g. use the chat (text chat integrated in video page) as a fallback to exchange info. But better dedicate a small Node server or use existing (maybe n8n could even do, but let's stick to Laravel). - Considering time, an easier approach: use **PeerJS** cloud which eliminates need for own signaling. Each user gets a PeerJS id (the unique code) and they connect via PeerJS network. It's free for some usage, or self-hostable. - We can mention that we either self-host signaling or use third-party if needed. The key is the link is the common identifier.

Conclusie: De videochat-functionaliteit voegt een belangrijk element toe: het stroomlijnt het interviewproces binnen het platform zelf. De technische invulling gebruikt WebRTC peer-to-peer technologie zodat we geen grote serverbelasting hebben voor de video (behalve eventueel een TURN server voor NAT traversal als back-up). Voor de eerste versie volstaat P2P; als scaling nodig is (grotere gesprekken of slechte verbindingen) kan later een SFU (Selective Forwarding Unit, zoals mediasoup, Jitsi) overwogen worden.

De agenda-integratie zorgt ervoor dat gebruikers overzicht behouden en professioneel afspraken kunnen beheren, met opties tot herinneringen en externe kalender koppeling voor gebruiksgemak.

8. Notificatiesysteem

Het platform bevat een uitgebreid **notificatiesysteem** om gebruikers op de hoogte te houden van relevante gebeurtenissen, zowel via in-app meldingen als via e-mail. Hier beschrijven we hoe notificaties worden gegenereerd, beheerd en weergegeven.

In-app notificaties (bell icon):
- In de rechtsboven hoek van zowel de frontend (voor kandidaten) als de backend portal (voor bedrijven) bevindt zich een **melding-icoon** (meestal een bell-symbool). Wanneer er ongelezen notificaties zijn voor de ingelogde gebruiker, wordt een klein rood badge-cijfertje getoond op dit icoon met het aantal ongelezen meldingen. - Wanneer de gebruiker op het icoon klikt, opent een dropdown of zijpaneel met een lijst van recente notificaties: - Elke notificatie vermelden we kort wat er gebeurd is, bijvoorbeeld: - *"Nieuwe sollicitatie ontvangen voor Backend Developer van kandidaat Jan Jansen."* (voor een recruiter) - *"Je bent uitgenodigd voor een interview voor Frontend Developer bij ACME BV op 12 okt 2025 om 15:00."* (voor een kandidaat) - *"Kandidaat Sara de Vries is aangenomen voor Marketing Manager. Bevestig betaling van €15.000 via Mollie."* (voor een recruiter of financiën contactpersoon) - *"Betaling voor match 1234 is nog niet voldaan. Klik hier om te betalen."* (voor bedrijf, als herinnering) - Notificaties kunnen voorzien zijn van een type of icoon (bv. een brief-icoon voor nieuwe sollicitatie, kalender-icoon voor

interview, waarschuwingsicoon voor betaling). Deze details maken het visueel snel scanbaar. - Notificaties zijn gesorteerd op datum (nieuwste boven). - Ongelezen notificaties kunnen vet gemarkeerd worden. - **Klikken op notificatie:** Als de gebruiker een notificatie aanklikt, gebeurt er doorgaans twee dingen: 1. Markeer de notificatie in de database als gelezen (`is_read = true`), zodat deze niet meer als ongelezen telt. Dit gebeurt via de API (bijv. `PUT /notifications/{id}` op de achtergrond). 2. Voer een relevante actie: vaak navigatie naar een bepaalde pagina: - Nieuwe sollicitatie -> open de sollicitatie-detailpagina (match detail) of de vacature-inzendingenlijst. - Interview uitnodiging -> open de betreffende vacature sollicitatie of aparte pagina met details en de link. - Betalingsherinnering -> open een betalingspagina of popup. - Etc. 3. Eventueel kan extra info getoond worden bij klikken. In het projectplan stond: “Door hierop te klikken kan de contactpersoon de melding inzien door erop te klikken die dan in het groot weergegeven wordt.” Dit suggereert dat we misschien een modal openen met volledige detail van de melding (bijvoorbeeld de hele sollicitatiebrief of alle info). In de meeste gevallen is dat niet nodig omdat we direct naar de detailpagina navigeren waar de info toch staat. Maar we kunnen het als UI nuance interpreteren: als notificatie melding heel lang zou zijn of een conversatie, dan in modal tonen. Simpel gezegd: we zorgen dat click user naar de context brengt waar hij moet zijn om die gebeurtenis af te handelen of te bekijken.

- **Realtime updates:** We willen dat notificaties vrijwel direct verschijnen nadat de gebeurtenis zich voordoet (bijv. zodra een kandidaat solliciteert, moet de recruiter binnen een paar seconden de melding krijgen zonder pagina refresh). Hiervoor kunnen we websockets gebruiken:
- Laravel kan bij aanmaken van een Notification record een event broadcasten via (bijv.) **Laravel Echo**. We hebben al Redis geconfigureerd; we kunnen een websockets server opzetten (via **laravel-websockets** package, die met Redis sub/pub werk) of een service als Pusher. Gezien we Docker gebruiken en open-source, is laravel-websockets (by BeyondCode) een goede optie die op laravel horizon/echo draait.
- De Angular frontend zou een WebSocket client (via @aspnet/signalr or via pusher-js or laravel-echo npm) kunnen openen zodra ingelogd. De client luistert naar een channel, bijvoorbeeld `notifications-user-{id}`. Wanneer een event binnenkomt (`NewNotification` event with payload of notification), voegt Angular dat toe aan de lijst en verhoogt badge count.
- Als websockets is teveel voor nu, alternatief is pollen: de frontend kan elke X seconden / notifications checken op nieuwe. Maar realtime is mooier.

- Hier in de spec benoemen we dat we realtime meldingen ondersteunen (met websockets), wat een modern touch geeft.

- **Aanmaken van notificaties (backend):** In Laravel maken we gebruik van het **Notifications systeem** (Laravels Notification classes) of simpelweg direct onze `notifications` tabel. Voor elk relevant event in de applicatie roept de backend een function aan om notificaties te creëren:

- Bij nieuwe match (sollicitatie) -> creëer Notification for:
  - de eigenaar van de vacature (company contact of admin).
  - eventueel ook voor de Super Admin, maar waarschijnlijk niet nodig voor elke match.
- Bij interview gepland -> Notification for:
  - kandidaat (met datum en link).
  - eventueel voor de recruiter zelf als bevestiging (maar die zit er zelf achter, dus niet per se nodig).
- Bij kandidaat aangenomen (status = hired) -> Notification for:
  - bedrijf (bijv. financiële afdeling of gewoon admin) dat betaling vereist is. Dit kan wat specifieker: we kunnen kijken of company contactpersoon of adminrol krijgt dit.

    - Super Admin een melding zodat hij overzicht houdt? In plan staat dat Super Admin een overzicht heeft, maar ook dat hij een melding kan sturen handmatig. Misschien geen automatische noti naar SA nodig, hij kan zelf kijken op overzicht.

- Bij betalingsherinnering verstuurd -> Notification for:

    - bedrijfsgebruiker (contact) dat er een herinnering is verstuurd (maar dat is dubbel op eigenlijk met de herinneringsmail zelf).
    - Dit punt in plan: "Super Admin with one click on unpaid icon sends a reminder and also a mail to company, and the company sees a notification top right." – Dus die melding ontstaat door actie van SA: in dat geval, als SA klikt 'stuur herinnering' op een onbetaalde match, backend:
    - stuurt mail naar company contact,
    - creëert Notification record for that company contact like "Herinnering: betaling voor match X is verzonden. Gelieve z.s.m. te betalen."
    - Of misschien direct "Openstaand bedrag voor hire van [naam] op [vacature]. Klik om te betalen." – dat is wel heel direct. Misschien beter: Een melding die linkt naar betalingen sectie.
    - Voor implementatie: wanneer SA op icoontje klikt, call `POST /payments/{matchId}/reminder` -> backend sends mail & creates noti.

- Eventueel bij algemene dingen: if we implement admin actions like "Tenant created" -> noti to SA, maar dat is intern.

- Password reset etc. laten we als email only.

- **Markeren als gelezen:**

- Zoals gezegd, via API endpoints of via websockets ack. We update in DB and then possibly broadcast an update event (so other devices or the user see it gone).

- In Angular, we might mark UI instantly and send an API call in background.

- **Notificatie opschoning:** Over tijd kunnen veel notificaties ontstaan. We kunnen:

- Automatisch alles ouder dan X maanden archiveren of verwijderen to keep table small.
- Or store an `is_read` and only fetch unread by default.

- Possibly have an "All notifications" page to scroll through history if needed, but not mandatory.

- **E-mail vs in-app:** Sommige events gaan zowel via in-app notification als via e-mail:

- Nieuwe sollicitatie: ja, stuur direct e-mail naar recruiter (belangrijk, want die zit niet continu ingelogd). Tegelijk, in-app noti voor wanneer hij online is.
- Interview uitnodiging: e-mail naar kandidaat plus in-app noti.
- Betalingsherinnering: e-mail plus noti.
- Sommige events alleen in-app: bv. een algemene mededeling als we een nieuwe feature aankondigen (future).

- Het email template systeem (zie sectie 9) dekt de inhoud van de mails; het notificatiesysteem de korte in-app melding.

- **Technologies:**

- In Laravel, we can use the built-in Notification channels: Database channel (which writes to notifications table) and Mail channel (which sends mail) and Broadcast channel (for websockets). We can define Notification classes for events like `NewApplicationNotification`, `InterviewInviteNotification`, etc., containing logic for toDatabase(), toMail(), toBroadcast.
- Using those ensures consistency and easy maintenance.
- On Angular side, for websockets, we include libraries needed and connect to the channels, which requires configuration (the echo instance with auth token possibly).
- Alternatively, could use a simpler approach: when user is on page, rely on periodic refresh which is easier but less dynamic.

User experience: - De notificaties zorgen dat gebruikers niks belangrijks missen. Door de badge weten ze dat er iets vereist hun aandacht. - Dit draagt bij aan snelle opvolging: bv. kandidaat ziet meteen als er een interview is gepland en kan bevestigen of voorbereiden; bedrijf ziet meteen nieuwe kandidaten en kan actie ondernemen.

In conclusie biedt het notificatiesysteem een centrale plek voor communicatie van gebeurtenissen. Door combinatie van real-time meldingen en e-mailalerts blijven gebruikers altijd op de hoogte, wat de efficiëntie en gebruikerstevredenheid verhoogt.

9. E-mailtemplate beheer (CMS met variabelen)

Het platform verstuurt diverse geautomatiseerde e-mails naar gebruikers (kandidaten en recruiters). Om de inhoud van deze mails flexibel te houden en aan te passen aan tone-of-voice of branding, is er een **e-mailtemplatebeheersysteem** ingebouwd. Dit stelt beheerders in staat om via een CMS-interface de tekst van e-mails te beheren, met gebruik van variabelen voor dynamische gegevens.

Te versturen e-mails (overzicht): Enkele belangrijke e-mails die het systeem zal sturen zijn: - **Registratie bevestiging:** Naar nieuwe gebruikers (kandidaten of bedrijf) met wellicht verificatielink of gewoon welkomstboodschap. - **Wachtwoord reset mail:** Bevat link om wachtwoord te wijzigen. - **2FA code mail:** Indien gebruiker kiest voor verificatie via e-mail, het bericht met de 6-cijfer code. - **Nieuwe sollicitatie notificatie (naar bedrijf):** Inhoud: er is een sollicitatie binnen op vacature X van kandidaat Y. Bevat beknopte info en wellicht een link naar het platform om details te zien. - **Sollicitatie bevestiging (naar kandidaat):** Bedankt voor je sollicitatie op X bij bedrijf Y, we nemen contact op etc. - **Interview uitnodiging (naar kandidaat):** Je bent uitgenodigd voor gesprek op [datum/tijd] voor vacature X bij Y. Met eventueel instructies (bv. "Klik op deze link op het tijdstip om deel te nemen aan de video call" als remote). - **Interview bevestiging (naar bedrijf):** (Eventueel) ter info dat men kandidaat X heeft uitgenodigd op datum Y, ter administratie. - **Herinnering interview (naar kandidaat en bedrijf):** 24 uur voor gepland gesprek een herinnering. - **Kandidaat afwijzing (naar kandidaat):** Spijtig bericht dat niet geselecteerd etc. (dit kan template zijn dat bedrijf handmatig triggert vanuit portal als ze op "Afwijzen & verstuur mail" klikken). - **Kandidaat aangenomen (naar kandidaat):** Felicitatie mail (als bedrijf dat via platform wil sturen) of dit laten we wellicht bedrijven buitenom doen. Maar zou mooi zijn als platform faciliteert. - **Betalingsverzoek (naar bedrijf):** Wanneer ze iemand hebben aangenomen, een mail met factuur of betaallink van Mollie. ("Gefeliciteerd met uw nieuwe medewerker. Klik hier om de plaatsingsfee te voldoen...") - **Betalingsherinnering (naar bedrijf):** Als betaling uitblijft X dagen. - **Tenant aanmaak (naar nieuwe company admin):** Welkomstmail met instructies hoe in te loggen etc. (als SuperAdmin een bedrijf toevoegt). - ... en eventuele notificaties voor systeemgebruik.

Beheer van templates: - Via de tabel `email_templates` (zie DB schema) beheren we templates. Super Admin kan in de Super Admin panel naar een sectie "E-mail Templates". - Daar ziet hij een lijst van template names (eventueel gegroepeerd per categorie). - Hij kan een template aanklikken om te bewerken: er is een formulier met velden: - Subject (met variabelen mogelijk). - Body (rich text editor). - Taal selecteren (NL/EN). - Mogelijk een dropdown "Tenant: Global / [TenantName]" als we tenant-specifieke overrides toe staan. - Een hint-weergave van beschikbare placeholders (variabelen) voor deze template. Deze definities liggen in code of config. - Bijvoorbeeld voor "Interview invite candidate" template, placeholders: `{CANDIDATE_NAME}`, `{JOB_TITLE}`, `{COMPANY_NAME}`, `{DATE}`, `{TIME}`, `{VIDEO_LINK}`. De beheerder moet deze precies zo gebruiken in de tekst. - Na bewerken drukt hij opslaan, de data gaat naar DB. Vanaf dan zal het systeem bij het sturen van die mail de nieuwe inhoud gebruiken.

- **Opmaak:** We ondersteunen HTML opmaak in de email body (Laravel mail kan markdown of pure HTML templating). Een WYSIWYG editor (bv. TinyMCE or similar) in de admin panel zou handig zijn zodat men wat basis styling en links kan doen. We moeten wel opletten dat variabelen intact blijven (niet door de editor geescape).
- **Branding:** Als meerdere tenants eigen mails willen, zou je per tenant eigen logo/ondertekening willen. Eenvoudige route: voeg in templates placeholders voor logo (maar dat logo pad kennen we per tenant via companies table). We kunnen in global template iets als `<img src="{{company.logo_url}}">` als die info door backend meegegeven wordt bij render.

- Simpler: globale templates hebben generiek branding (platform name). Tenant-specific templates if needed: e.g. een corporate hiring uses their own style. Dat zou wel geavanceerd, maar de table structure with company_id allows customizing content for that company.

- **Template retrieval bij verzenden:**

- Wanneer een bepaald event zich voordoet en backend moet mail sturen, we schrijven code:

```
$template = EmailTemplate::where('name', 'new_application_company')
              ->where('language', $userPrefLang)
              ->where(function($q) use ($tenantId) {
                  $q->where('company_id', null)->orWhere('company_id',
$tenantId);
              })
              ->orderBy('company_id', 'desc') // ensures tenant-specific
overrides preferred
              ->first();
$subject = TemplateParser::parse($template->subject, $variables);
$bodyHtml = TemplateParser::parse($template->body, $variables);
Mail::to($recipientEmail)->send(new GenericMail($subject, $bodyHtml));
```

  - TemplateParser is een helper die door de tekst gaat en `{VAR}` vervangt met echt waardes uit een array $variables.
  - Deze $variables vullen we per geval. Voor `new_application_company` bijv:

```
$variables = [
   'COMPANY_NAME' => $vacancy->company->name,
   'JOB_TITLE' => $vacancy->title,
```

```
    'CANDIDATE_NAME' => $candidate->fullName(),
    'CANDIDATE_EMAIL' => $candidate->email,
    'APPLICATION_DATE' => date('d-m-Y H:i'),
    'PLATFORM_URL' => config('app.url'),
    // etc.
];
```

    - Als variabele niet in array staat maar wel in template, we kunnen ze leeg laten of fallback. Beter dat template beheerder exact weet welke beschikbaar zijn.

- **Placeholders notatie:** Hier gebruiken we {ALL_CAPS} stijl, of we kunnen %%VAR%%. Belangrijk is dat ze niet conflict met whatever templating (we likely don't use Blade for these, as they come from DB).

- We documenteren deze placeholders voor de admin in de UI (bv een legend onder de editor).

- **Preview & Test:** Het CMS kan een "Stuur testmail" knop hebben zodat admin zichzelf een test kan sturen om lay-out te checken. Dit zou de eerste in de mailbox laten zien hoe het eruit ziet.

- **Multilingual:**

- We hebben duplicate templates per language. Als de platform supports EN and NL, admin can toggle editing each language. If a language version is missing, fallback to default language maybe.
- The system chooses which language to send in. We can base it on the recipient's preference (for candidates likely they pick a language in profile; for companies possibly default to NL or a per-tenant language setting).

- Example: If candidate has language=EN, we choose the EN template of that name. If not found, use NL.

- **Tenant context:**

- Possibly a tenant admin can customize templates used for their own company (like customizing the invite email that candidates get from them). The table allows company_id for that. We would allow Tenant Admin role to edit templates where company_id = their company or create such record (the system by default uses global, but if company override exists, that one is used).
- We must ensure a tenant admin cannot see or edit global templates or other tenant's templates.

- This gives flexibility for bigger companies to tailor the messaging or include their branding in the content.

- **Email sending mechanism:**

- Laravel's Mailables for each scenario can be fairly generic because content comes from DB. We might have a single Mailable class `SystemTemplateMail` which takes subject and body (already processed with variables) and just builds that (with maybe a generic layout).

- Alternatively, use Notification channel: toMail($notifiable) could fetch template and fill variables. That way same Notification class logic can be reused and content easily updated.

- **CMS UI security:** Only Super Admin (and possibly tenant Admin for their own templates) has access. We implement form validation to ensure required placeholders remain present if needed (for e.g. a template should probably include certain critical placeholders, but we might trust the admin to not remove them).

- **Storage of media (images) for emails:** If an email template needs an image (like a logo or banner), we can handle that in a few ways:

- Hardcode platform logo in a base layout (not part of template body).
- Or allow template to have `<img src="...">` to e.g. a public URL. Possibly the platform has a setting for logo URL (like companies table has logo field).

- Keep emails mostly simple text/HTML, maybe no heavy images except logo.

- **Variables injection:**

- Danger: If an admin writes something like "Dear {USERNAME}" but we provide only {FIRST_NAME}, they'll get an empty or unresolved var. We could have the parser leave the placeholder if not found or remove it.

- So we should clearly communicate which placeholders are available for each template and ideally enforce those keys exist in the $variables array when sending (log if missing).

- **Examples:** The admin can for instance edit the "payment_reminder" template to adjust tone, or the "new_application_company" to add a line "Log in to the portal to view CV and details."

Door dit e-mailtemplate systeem kunnen wijzigingen in mailcommunicatie doorgevoerd worden zonder code-aanpassingen. Dit is belangrijk voor het gebruik in de praktijk, omdat de gewenste inhoud of taal zich kan evolueren (bijv. formeel vs informeel aansprekend, toevoegen van een helpdesk contact in de mails, etc.). Het verhoogt ook de meertalige ondersteuning: templates kunnen naar behoeven vertaald worden.

10. Beheerfunctionaliteit voor Super Admin

De **Super Admin** gebruiker heeft een speciale rol met uitgebreide rechten om het gehele platform te beheren. Hieronder een overzicht van de specifieke beheerfunctionaliteiten en schermen die voor de Super Admin beschikbaar zijn, naast de al genoemde zaken:

- **Dashboard/Overzicht:** Bij inloggen als Super Admin ziet men een dashboard met key metrics:
- Aantal actieve tenants (bedrijven).
- Aantal geregistreerde kandidaten.
- Totaal aantal vacatures geplaatst.
- Aantal matches (sollicitaties) tot nu toe, en eventueel aantal hires.
- Eventueel grafieken: bijvoorbeeld nieuwe vacatures per week, sollicitaties per week, etc., mogelijk filterbaar per tenant.
- Deze statistiekenpagina kan filters hebben (Super Admin kan filteren per tenant, of bepaalde periode).

- Dit geeft Cursor App inzicht in gebruik en succes van het platform.

- **Tenant beheer:** Een sectie waar alle tenants (bedrijven) worden beheerd:

- **Lijst van tenants:** Tabel met kolommen: naam, aantal users, aantal open vacatures, status (actief/inactief), datum aangemaakt, etc.
- Mogelijkheid om te zoeken op tenant naam.
- **Toevoegen nieuwe tenant:** via formulier, invullen bedrijfsgegevens en meteen een Admin-gebruiker aanmaken (met e-mailadres waar uitnodiging naartoe gaat).
- **Bewerken tenant:** aanpasbare gegevens: naam, adres, contactpersoon, eventueel licentie-instellingen als dat zou bestaan (nu niet, maar bijvoorbeeld max aantal vacatures etc. in SaaS context).
- **Deactiveren/Verwijderen tenant:** Deactiveren zodat men niet meer kan inloggen en vacatures offline gaan. Verwijderen zou alle data van dat bedrijf weghalen (gevaarlijk, dus wellicht alleen op verzoek of na export).

- **Impersonatie/switch:** Op tenant detailpagina een knop "Inloggen als deze tenant" waarmee de Super Admin de interface ziet alsof hij Admin van dat bedrijf is. Technisch: log uit eigen user en log in als dat bedrijf's admin (of een impersonation route). Dit is handig voor support/debug. Wel moet gelogd worden wie impersonated voor veiligheid.

- **Gebruikersbeheer (platform breed):**

- Lijst van alle gebruikers in het systeem (optioneel, want dat kan er veel zijn; wellicht vooral nuttig voor support).
- Filter per tenant of rol.
- Mogelijkheid om een gebruiker te bewerken, rol aan te passen, of te blokkeren/wissen.
- Super Admin kan dus ingrijpen als er misbruik is of accounts samenvoegen etc.

- Ook hier impersonation per user zou kunnen (login as user) voor support, maar dat is gevaarlijker privacygewijs (niet gevraagd, wellicht niet nodig).

- **Rollen & Rechten beheer:**

- Super Admin kan globale rollen toevoegen (al zijn de main roles al genoemd).
- Beheren welke permissies een rol heeft: een UI met checkboxes per permission per rol (matrix of per rol listing).
- Permissies kunnen generiek "read/write/delete" of meer specifiek per module (vacancy_read, vacancy_create etc.). Als we Spatie gebruiken, we definieren bij seed wat default is.
- UI scenario: Super Admin gaat naar "Roles & Permissions" menu:
    - Ziet lijst rollen (Super Admin, Admin, Manager, Editor, eventueel Candidate al is die geen company role).
    - Klik op rol -> lijst permissies met toggles.
    - Voor globale rollen (Super Admin): waarschijnlijk alles aan en dit scherm read-only "Super Admin heeft alle rechten".
    - Voor Admin (tenant): standaard alles behalve global management.
    - Voor Manager/Editor: wat wel/niet, Super Admin kan dit fine-tunen.
- Tenant Admin zou mogelijk ook deze sectie zien maar dan alleen zijn tenant-rol en custom rollen.

- Dit komt overeen met "geavanceerde rechtenstructuur via Laravel Spatie" eis.

- **Configuraties (vacaturesettings):**

- Pagina waar de Super Admin de globale waarden beheert zoals de lijst van categorieën, lijst van dienstverbandtypes, etc.

• Dit kan verschillende tabs:
    ◦ Categorieën: tabel, voeg toe, bewerk naam/slug, verwijder (als niet in gebruik).
    ◦ Dienstverband types: lijst (Fulltime, Parttime, etc.) - beheer, vertalingen misschien (Fulltime in NL is Full-time in EN, we kunnen ofwel slug en do translation in front-end, of we treat these as label per language).
    ◦ Werktijden: lijst ('09:00-17:00' etc.).
    ◦ Vacature status opties: standaard 3 zijn genoemd, wellicht laat maar wel uitbreidbaar (maar in workflow wellicht beperkt).
    ◦ Eventueel opleidingsniveau lijst als we dat als distinct entity willen (nu niet in DB).
    ◦ Sectoren (die overlapt categorie wellicht).

• Interface: simpel crud, en presumably these tie into job_configurations table.

• **Mollie/payment instellingen:**

• Super Admin moet de Mollie API key kunnen instellen via de UI (tenzij hij .env direct doet, maar UI is gebruiksvriendelijker).
• Een Settings page waar Payment provider (Mollie) keys, perhaps test/production mode toggle, and what the fee amount is (15000).

• Of Payment fee per tenant verschillend? Niet gevraagd maar in de toekomst misschien. Voor nu uniform.

• **Betaaloverzicht:**

• Super Admin wil een overzicht van alle matches die op 'hired' staan en de status van betaling daarvan. (Zodat hij kan monitoren wie heeft wel betaald en wie niet).

• Dit kan onder een menu "Payments" of onderdeel van matches overzicht:

    ◦ Lijst van succesvol gematchte kandidaten (hired) met kolommen: Bedrijf, Vacature, Kandidaat, hire date, betaald ja/nee, betaaldatum, eventueel betaal-ID.
    ◦ Sorteren/filter op betaald/niet betaald.
    ◦ Als niet betaald: een actieknop "Stuur herinnering" (zoals eerder vermeld). Dit stuurt direct vanuit dit scherm de reminder e-mail+notificatie naar dat bedrijf.
    ◦ UI wise: een envelop-icoon of alarm-icoon naast de entry, rood gemarkeerd als unpaid. Klik -> confirm modal "Herinnering sturen?". Ja -> call API voor reminder.
    ◦ Na success: mark maybe a field "reminder_sent_at" or so, and update UI (maybe icon turns grey or so).
    ◦ Als de betaling voldaan is (on webhook van Mollie), we zouden matches.payment_status = paid etc updaten en hier tonen betaald op datum X.
    ◦ Betaald entries kunnen groen check icon hebben, no action needed.
    ◦ Super Admin kan ook betaling opnieuw controleren (als mis bij webhook, or manually mark if offline pay? Possibly not needed if Mollie).

• **Statistieken/Reporting:**

• Als uitbreiding misschien grafieken, maar basis is genoemd op dashboard.

• Kan ook export functies bevatten (download CSV van alle hires etc.).

• **Systeeminstellingen:**

- Misschien een sectie voor algemene instellingen: zoals platform naam, logo, default email templates (some of that we already cover), toggling features, etc.
- Niet expliciet gevraagd, maar kan deel uitmaken:
    - Bijv. multi-language enabling new languages (if extended).
    - Changing default theme or email sender addresses.
    - Manage API credentials for external integration (like if we integrate LinkedIn Jobs API or Google in future, we might store keys).
    - For now, minimal: Payment key (we covered), maybe SMS gateway key if 2FA via SMS is used and we want it changeable in UI.
- **Logboeken/Audit:**
- Optioneel: Overzicht van alle activiteiten (who did what) if needed for security. Spatie has Activity Log package if needed.

Super Admin UI: - Dit zou grotendeels een uitbreiding van de Angular admin module zijn. The Material Admin Template can cover a lot of it. - We should ensure only Super Admin sees those menus. - Possibly incorporate an icon to switch impersonation context (like a dropdown listing tenants or an input to search tenant and switch context). - Per plan: "Switch tussen tenants via dropdown". Concreet, in de header, SuperAdmin has a dropdown listing all companies by name. Selecting one refreshes the view to act as that tenant (essentially sets a context so that when he navigates to vacancies or users, it filters by that tenant as if he is admin there). - Implementation: we can simply store a selectedTenantId in a global admin service or local storage and have backend read a header as earlier. Or on each request from admin, attach X-Impersonate-Company header or so. - This is alternative to actual impersonation login, easier to implement in multi-tenant approach: * E.g. SuperAdmin queries `GET /vacancies?tenant_id=X` to see that company's jobs. We might not allow that by default due to scope, but we can override if role is super. * Or we temporarily set an internal session variable like "impersonated_tenant = X" for that user. - UI: The dropdown might list only active tenants, maybe with search if many.

- **Zie alle matches**:
- In plan: "Overzicht alle matches en of betalingen zijn uitgevoerd" hebben we onder betalingen eigenlijk.
- Possibly also a way to see all applications across platform (less useful except maybe for analysis of usage).
- But could be a page listing all matches with filter by tenant or vacancy, mainly for admin curiosity or debugging.

In essentie heeft de Super Admin een control center om het platform draaiende te houden: bedrijven en gebruikers managen, instellingen tunen en zorgen dat financiële afspraken (betalingen) nagekomen worden. Alles in een veilige afgeschermde omgeving (Super Admin rol is zeldzaam, meestal maar 1-2 accounts). Deze functies maken het mogelijk om het systeem te configureren en te modereren zonder direct in de database te hoeven duiken.

11. Flow van AI-matching, sollicitatie, beoordeling en betaling

In dit deel beschrijven we de **end-to-end flow** van het systeem: van het moment dat een kandidaat een potentiële vacature zoekt, via de AI-match en sollicitatie, door het selectieproces (interviews, beoordeling) tot en met de betaling bij een succesvolle match. We verbinden hiermee de eerder gespecificeerde onderdelen tot één geheel, stap voor stap:

Stap 0: Platform setup en vacaturepublicatie
- Bedrijf (tenant) ABC maakt via de backend een nieuwe vacature aan: *"Senior Java Developer"* met alle details (categorie IT, Fulltime, locatie Amsterdam, salarisrange etc.). Status staat op *Open*, dus na publicatie is deze direct zichtbaar in het openbare vacatureoverzicht. De SEO meta-tags zijn gegenereerd of ingevuld, zodat bijvoorbeeld Google de pagina kan indexeren met trefwoorden "Senior Java Developer Amsterdam..." enz. - Kandidaten kunnen deze vacature zien tussen alle openstaande. De platform homepage toont wellicht prominente vacatures of heeft zoek/filter.

Stap 1: AI-geassisteerd zoeken naar vacatures
- Kandidaat X bezoekt de site op de homepage of vacatureoverzicht. In plaats van handmatig filters te klikken, besluit hij de **chatbot** te gebruiken om te zoeken. - Hij klikt op het chat-icoon "Vind een passende baan via chat". De **chatbot** begroet met *“Waar ben je naar op zoek in je volgende baan?”*. Kandidaat antwoordt in natuurlijke taal: *"Ik ben net afgestudeerd in informatica en zoek een baan als Java developer in de Randstad."* - De chatbot stelt vervolgvragen: locatie (kandidaat: *"Amsterdam of Utrecht"*), type dienstverband (kandidaat: *"Fulltime"*), etc., doorloopt de lijst (sommige antwoorden had hij impliciet al gegeven, maar de AI kan toch vragen ter bevestiging of details). - Nadat alle relevante voorkeuren bekend zijn, zegt de chatbot bijvoorbeeld: *"Bedankt! Op basis van je input ga ik passende vacatures zoeken..."*. De n8n workflow gebruikt de verzamelde criteria en voert een zoekactie op de database uit. - Vrijwel direct toont de frontend nu gefilterde zoekresultaten: bijvoorbeeld 3 vacatures, waaronder "Senior Java Developer bij ABC in Amsterdam" (de eerder genoemde vacature) en nog 2 anderen. De chatbot meldt: *"Ik heb 3 vacatures gevonden die aansluiten bij je voorkeur. Zie de lijst hieronder voor details."* - (Als de kandidaat niet was ingelogd tijdens chatzoeken, is dat niet erg voor zoeken. Voor solliciteren straks wel nodig.)

Stap 2: Sollicitatie met AI-chatbot assistentie
- Kandidaat X bekijkt de resultaten en klikt "Senior Java Developer" vacature om de details te lezen. Het spreekt hem aan, dus hij klikt op **"Solliciteer"**. - Omdat hij nog niet is ingelogd, wordt hij nu gevraagd in te loggen of een account te maken. Hij registreert zich als kandidaat (met 2FA verificatie via e-mail, voert code in, etc.) en komt vervolgens automatisch terug op de vacaturepagina. - Nu ingelogd, klikt hij opnieuw "Solliciteer". Dit activeert de **sollicitatie-chatbot**: een chatvenster verschijnt met een begroeting specifiek voor sollicitatie, bijvoorbeeld: *"Leuk dat je wilt solliciteren voor Senior Java Developer bij ABC! Ik zal je een paar vragen stellen om je sollicitatie compleet te maken."* - De chatbot (via n8n AI agent) heeft op de achtergrond de vacature-info opgehaald (vereisten, etc.) en begint vragen te stellen: 1. *"Kun je kort vertellen wat je aantrekt in deze functie en waarom je denkt dat je een goede kandidaat bent?"* – (Deze vraag staat niet expliciet in de lijst eerder, maar een AI kan beslissen een open motivatievraag toe te voegen voor extra input.) 2. *"(Ervaring) Welke relevante ervaring of projecten met Java heb je?"* – Kandidaat vertelt over stages/projecten. 3. *"(Opleiding) Welk opleidingsniveau heb je voltooid?"* – Kandidaat: "Ik heb een MSc in Computer Science behaald aan TU Delft." 4. Chatbot merkt dat opleiding en starter-level duidelijk zijn, vraagt dan: *"Ben je per direct beschikbaar of heb je een opzegtermijn?"* – Kandidaat: "Per direct beschikbaar." 5. *"Wat is je gewenste salaris voor deze functie?"* – Kandidaat noemt bijv. "Rond €3000 per maand." 6. (De chatbot slaat eventueel de vraag over sector, want is al duidelijk - IT.) 7. *"Heb je nog vragen voor het bedrijf of iets dat je wilt toevoegen aan je sollicitatie?"* – Kandidaat: "Nee, alles is me duidelijk. Ik hoop van jullie te horen!" - De chatbot concludeert: *"Dankjewel voor je antwoorden. Ik ga je gegevens verwerken en je sollicitatie indienen..."* - Op dit moment stuurt de n8n workflow de verzamelde Q&A naar de Laravel backend (via API). Laravel creëert een **Match** record: - user_id = X, vacancy_id = ABC's Java Dev vacature. - status = 'matched'. - De AI genereert op basis van de antwoorden een `score` bijvoorbeeld 0.90 (90%) match, aangezien zijn profiel goed aansluit op de eisen. Dit kan op simpele wijze: de chatbot of backend vergelijkt opleidingsniveau en het feit dat hij exact Java ervaring heeft, etc. - `ai_feedback`: hierin plaatst de backend bijvoorbeeld een samenvatting: "Kandidaat heeft MSc Computer Science, enige projectervaring in Java, en zoekt een fulltime rol in Amsterdam. Beschikbaar per direct, salarisindicatie ~€3000." (De chatbot zou zo'n

samenvatting kunnen aanleveren, of we stellen 'motivation' = zijn antwoord op waarom hij goede kandidaat is). - Zodra match is aangemaakt: - De kandidaat krijgt op het chatvenster te zien: *"Je sollicitatie is verstuurd! Je ontvangt een bevestiging per e-mail."* (en eventueel: "Bedankt voor je tijd, je hoort binnenkort van ons."). - Laravel verstuurt een **e-mail** naar kandidaat: "Bevestiging sollicitatie Senior Java Developer bij ABC - Bedankt voor je sollicitatie, we nemen spoedig contact op...". - Laravel verstuurt ook een **e-mail** naar de bedrijf contactpersoon/admin: "Nieuwe sollicitatie ontvangen voor Senior Java Developer – Kandidaat: [Naam], [Korte samenvatting/AI feedback]. Log in om de details te bekijken." - In de portal voor ABC verschijnt een **notificatie** voor de verantwoordelijke: melding (1) bij het bell-icoon. "Nieuwe sollicitatie: [Naam kandidaat] voor Senior Java Developer." - De bedrijf user logt mogelijk in en ziet meteen op dashboard het aantal open sollicitaties verhoogd, en notificatie. - De chatbot voor sollicitatie sluit af (session ended). De kandidaat kan in zijn profiel zien dat sollicitatie in behandeling is.

Stap 3: Beoordeling door bedrijf (matching en interview)
- Recruiter (of hiring manager) bij ABC logt in de portal, gaat naar "Sollicitaties" voor de betreffende vacature. Hij ziet kandidaat X met status *matched*, en een AI-matchscore van 90%. Misschien ziet hij een highlight dat AI deze kandidaat sterk passend acht (boven 95% was doel, maar 90% is ook goed). - Hij klikt op de kandidaat om details te zien: de antwoorden die de kandidaat gaf kunnen hier overzichtelijk staan (de bot heeft die opgeslagen of de recruiter kan een "motivatie" lezen uit ai_feedback). Ook CV als die was geupload (in dit scenario niet gedaan, want we kozen chat in plaats van CV). - Gezien de kandidaat net afgestudeerd is maar wel goede achtergrond, de recruiter besluit hem uit te nodigen voor een gesprek. In de portal klikt hij **"Uitnodigen voor interview"** voor kandidaat X: - Hij kiest een datum en tijd, zegt 3 dagen later om 10:00, en selecteert "video gesprek" (remote) of dit is standaard. - Het systeem maakt een Interview record: scheduled_at = die datum, video_chat_url = bv. `https://platform.nl/vcall/abcdef123`. - Automatisch gaat er een **e-mail** naar kandidaat: "Uitnodiging interview – Beste [Naam], u bent uitgenodigd voor een gesprek bij ABC op [datum] om [tijd]. [Als remote:] Klik op deze link om deel te nemen: [link]. We kijken ernaar uit." - De kandidaat krijgt ook een **in-app notificatie**: "Uitnodiging: gesprek gepland op [datum] voor Senior Java Developer bij ABC." - In de portal wordt de match status voor deze kandidaat op 'interview' gezet (oranje markering). - De portal toont dit interview in de **Agenda** sectie voor ABC (en bij de kandidaat in zijn account als hij daar kijkt). - De geplande dag en tijd komt. Kandidaat en interviewer klikken op de link (kandidaat logt in, navigatie naar /vcall/abcdef123, recruiter wellicht direct vanuit portal). - De **videochat** start in hun browsers. Ze voeren het gesprek, audio/video werkt prima. Misschien is er ook een tekstchat zijbalk als ze links willen delen (optioneel). - Gesprek afgelopen; beide sluiten de call.

Stap 4: Resultaat van sollicitatie
- Recruiter besluit de kandidaat aan te nemen. In de portal selecteert hij bij de match van kandidaat X: **"Aannemen"** (hired). - Het systeem vraagt eventueel bevestiging "Kandidaat aannemen? Hiermee wordt een fee in rekening gebracht." Recruiter bevestigt. - Match status wordt op 'hired' gezet (groene markering bij de kandidaat in de lijst). - Vacature kan automatisch op 'Gesloten' gezet worden nu er een hire is (als we aannemen dat vacature vervuld is). Of men laat open als er meerdere posities, maar meestal dicht. - Het systeem genereert een **betaaltransactie** via Mollie: * Een unieke betalingsaanvraag van €15.000 voor tenant ABC wordt aangemaakt via Mollie API. We krijgen een payment link or checkout URL terug. * We slaan bij de match of in aparte payment record het Mollie payment ID, status 'open', amount, due date, etc. - Er gaat een **e-mail** naar de contactpersoon van bedrijf ABC: "Gefeliciteerd met de succesvolle match! Klik hier om de plaatsingsfee van €15.000 te voldoen." Deze mail bevat de Mollie-betaallink. - In de bedrijf portal komt een prominente melding (of notificatie): "Betaling vereist voor aangenomen kandidaat [Naam] - Bedrag €15.000. Status: Openstaand." * Wellicht tonen we een banner of in het vacature/matches overzicht een rood icoon bij die match. * Een betaalknop is aanwezig die ook de Mollie link opent. - De kandidaat krijgt ook een **e-mail** (en eventueel notificatie) dat hij is aangenomen: "Goed nieuws! [Bedrijf] wil je aannemen voor

[Functie]. Ze zullen contact met je opnemen voor het vervolg. Gefeliciteerd!" * Deze mail is meer voor de kandidaat persoonlijk, de platform faciliteert dat voor het bedrijf. (Eventueel in overleg of bedrijf dat zelf doet, maar we kunnen het automatiseren). - Nu is de match succesvol tot hire gebracht. Voor platform rest alleen nog dat betaling gedaan wordt.

Stap 5: Betaling en opvolging
- De contactpersoon bij ABC klikt op de betaallink en voltooit de betaling via Mollie (iDeal of creditcard etc.). Mollie stuurt een webhook naar onze Laravel backend upon payment success: - We vangen `/api/payment/webhook` op, identificeren de betaling bij match X, markeren `payment_status = paid` (en `paid_at = now`). - Vacature en match zijn daarmee financieel afgehandeld. - We kunnen direct een **notificatie** sturen naar Super Admin: "Betaling ontvangen van ABC voor match [vacature] - €15.000". (Zodat hij weet dat die binnen is). Dit is optioneel, maar handig voor administratief. - Misschien ook e-mail factuur naar bedrijf (Mollie kan dat zelf ook genereren, maar we kunnen een factuur PDF genereren). - In de bedrijf portal verdwijnt de open-betaling melding of wordt een groen check getoond bij die match. - **Indien betaling uitblijft:** Stel ABC heeft na 1 week niet betaald: - Super Admin ziet in zijn Payments overview dat ABC match X staat als 'openstaand' sinds >7 dagen. - Hij klikt op "stuur herinnering". Het systeem: * Stuurt opnieuw een e-mail naar ABC: "Herinnering: betaling van €15.000 voor [vacature hire] staat nog open. Gelieve z.s.m. te betalen via [betaallink]." * Creëert een in-app notificatie voor ABC: "Betalingsherinnering: bedrag €15.000 nog open voor [vacature]. Klik om te betalen." - De Super Admin melding dat hij verstuurd heeft, en kan eventueel escaleren als nog later. - Uiteindelijk betaalt ABC. Als niet, SuperAdmin kan contact opnemen buiten systeem of account blokkeren voor nieuwe vacatures.

Stap 6: Afsluiting
- De vacature "Senior Java Developer" wordt als gesloten gemarkeerd, kandidaten die niet aangenomen waren kunnen ook via platform een afwijzingsmail krijgen (als recruiter dat doet via bijv "Markeer rest als afgewezen en stuur bulkmail"). Dat is een nice-to-have feature: - Recruiter klikt "Afwijs alle overige sollicitanten" en het systeem stuurt naar die matches status=rejected en mails "Bedankt voor je moeite, helaas niet gekozen". - Kandidaat X staat nu als hired bij ABC. Het systeem zou dit kunnen noteren voor statistieken (success rate). - Super Admin ziet op dashboard: matches +1, hires +1, revenue +15k, etc.

AI feedback loop (toekomst): - Over de tijd leert het AI model van succesvolle matches: bv. het ziet dat in deze case, de kandidaat score 90% en daadwerkelijk hired -> model kan dat als bevestiging gebruiken. - Als een kandidaat hoog scoorde maar is afgewezen, kan dat feedback zijn om model te fine-tunen (maar dit vereist data science approach later). - Voor nu, AI did initial scoring.

Deze complete flow toont hoe alle componenten samenwerken: - De **AI-chatbot** versnelt zoeken en solliciteren, en verhoogt matching kwaliteit. - **Multi-tenant structuur** zorgt dat alles gescheiden blijft (ABC bedrijf ziet alleen zijn vacatures/sollicitaties, enz.). - **Notificaties en e-mails** zorgen voor vlotte communicatie op elke stap. - **Videochat en agenda** integratie maken het mogelijk binnen het platform sollicitaties af te handelen. - **Betalingskoppeling Mollie** maakt de commerciële afronding na een geslaagde hire efficiënt. - **Super Admin tools** waarborgen controle en mogelijkheid tot ingrijpen.

Met dit technisch ontwerp is Cursor App in staat de volledige applicatie te ontwikkelen. Alle nodige details, van database tot API en integraties, zijn uitgewerkt. Het resultaat is een modern, AI-ondersteund vacatureplatform dat gebruiksgemak biedt aan kandidaten en recruiters, en de efficiëntie van het matchingsproces maximaliseert.
