THOMAS MORE

Elegant

Realisatie Document

Serneels Kenzo

2025 - 2026

Inhoud

Lijst met figuren

1. Inleiding

Dit document beschrijft de realisatie van het case-systeem van Elegant. Dit systeem biedt medewerkers een gecentraliseerd platform om klantproblemen te registreren, op te volgen, op te lossen en efficiënter samen te werken. De implementatie omvat zowel technische als functionele componenten, met een sterke focus op gebruiksvriendelijkheid, schaalbaarheid en betrouwbaarheid.

Het case-systeem vormt één van de eerste bouwstenen binnen een groter CRM-systeem dat momenteel binnen Elegant wordt ontwikkeld. Dit document richt zich op de concrete uitvoering en implementatie van dit onderdeel en bouwt verder op het initiële projectplan. De verschillende fasen van het ontwikkelingsproces worden hierbij gedetailleerd toegelicht.

De structuur van dit document is als volgt:

- Analyse: een bespreking van de vereisten voor het case-systeem.
- Elegant: achtergrondinformatie over het bedrijf en de diensten die het aanbiedt.
- Design fase: een overzicht van de ontwerpprincipes, architecturale keuzes en gebruikte bibliotheken ter ondersteuning van de ontwikkelingsfase.
- Ontwikkeling structuur: toelichting bij de manier waarop het ontwikkelingsteam van Elegant hun systemen opzet en beheert.
- Authenticatie: beschrijving van de gebruikte authenticatiemethode, veiligheidsprincipes en toegangscontrole binnen het systeem.
- Database structuur: een overzicht van het datamodel, de belangrijkste tabellen en de relaties die de kern vormen van het case-systeem.
- Gebruikte tools en technologieën: een opsomming en motivatie van de software, frameworks en ontwikkeltools die tijdens het project zijn gebruikt.
- Eindresultaat: een samenvatting van de gerealiseerde functionaliteiten en hoe deze bijdragen aan de toekomstige uitbouw van het CRM-platform.

Dit document dient als referentie voor zowel de technische realisatie als de functionele opbouw van het case-systeem en vormt een basis voor verdere uitbreiding binnen het bredere CRM-project van Elegant.

2. Analyse

De eerste stap in het ontwikkelproces was het analyseren van het oude systeem (ES3). Hierbij werd onderzocht hoe het systeem functioneerde, welke functies essentieel waren en welke onderdelen niet langer voldeden aan de behoeften. Daarnaast werden uitgebreide gesprekken gevoerd met het customer support-team om inzicht te krijgen in welke functionaliteiten zij wilden behouden, verbeteren of verwijderen. Deze analyses waren noodzakelijk om de functionele vereisten voor het nieuwe case-systeem correct te definiëren.

2.1. Belangrijke analyses van het case-systeem

De ontwikkeling van het case-systeem omvatte meerdere analyses die van invloed zijn op de werking, efficiëntie en gebruiksvriendelijkheid van het uiteindelijke product. In dit document worden de volgende kernonderdelen besproken:

- Audit log.
- Detail case.
- Attachments.
- Marktberichten.

Deze onderdelen zijn ontworpen om het systeem zo gebruiksvriendelijk mogelijk te maken binnen Elegant en om het werkproces voor medewerkers te stroomlijnen. Hieronder volgt een beschrijving van elk onderdeel, inclusief de gemaakte ontwerpkeuzes en de motivatie daarvoor.

Audit log

De audit log is een essentieel onderdeel van de applicatie, omdat het alle belangrijke acties en wijzigingen binnen een case registreert. De log vormt een chronologische tijdlijn van gebeurtenissen, zoals het aanmaken van een case, aangepaste details en welke medewerkers eraan hebben gewerkt.

Waarom we de audit log implementeerden

De keuze om een audit log te voorzien was vanzelfsprekend, aangezien deze functionaliteit ook aanwezig was in ES3. In het oude systeem was de auditinformatie echter te uitgebreid en bevatte ze veel irrelevante gegevens. In het nieuwe case-systeem werd daarom gekozen voor een vereenvoudigde en gerichte audit log die enkel de noodzakelijke informatie bijhoudt. Dit verhoogt niet alleen de veiligheid en betrouwbaarheid van het systeem, maar zorgt ook voor transparantie en maakt foutopsporing efficiënter.

Functionaliteiten van de audit log

- Tijdstempel: elke actie binnen een case wordt vastgelegd met datum en tijd, zodat duidelijk traceerbaar is wanneer een wijziging heeft plaatsgevonden.
- Gebruiker: de identificatie van de medewerker die de actie heeft uitgevoerd, waardoor altijd zichtbaar is wie welke handeling heeft verricht.
- Actieomschrijving: een korte beschrijving van de uitgevoerde actie, bijvoorbeeld: gebruiker heeft case aangemaakt op datum/tijd.
- Detail: specifieke waarden die zijn aangepast, bijvoorbeeld: naam van veld gewijzigd van ‘oude waarde’ naar ‘nieuwe waarde’.

Detail case

De detailpagina van een case vormt de kernfunctionaliteit van het case-systeem. Dit onderdeel biedt medewerkers een volledig overzicht van alle relevante informatie over een case, waaronder: klantgegevens, attachments, case-specifieke details, de audit log en marktberichten. Door al deze informatie op één plek te bundelen, wordt het beheer van cases overzichtelijker en efficiënter.

Waarom we detailpagina hebben toegevoegd

De detailpagina is het centrale component van het systeem en vormt het hart van de stageopdracht. Het stelt medewerkers in staat om snel inzicht te krijgen in alle aspecten van een case, zonder tussen verschillende schermen of systemen te hoeven schakelen. Hierdoor kan het werkproces binnen Elegant gestroomlijnd worden en kunnen fouten sneller worden opgespoord en opgelost.

Functionaliteiten van de detailpagina

- Klantgegevens: bevat alle relevante informatie over de klant, zoals klantnummer, e-mailadres, adres, EAN-code, enzovoort.
- Case details: toont alle informatie over de case, bijvoorbeeld type case, inhoud en relevante opmerkingen.
- Attachments: hier kunnen documenten of foto’s aan de case worden toegevoegd. Beschikbare acties zijn uploaden, bekijken en verwijderen.
- Marktberichten: wordt getoond wanneer een case is aangemaakt op basis van een marktbericht. De functionaliteit wordt uitgebreider toegelicht in het onderdeel marktberichten.

Marktberichten

De markberichten-functionaliteit verwerkt berichten die afkomstig zijn van externe partijen, zoals Engie of Fluvius. Wanneer een marktbericht binnenkomt, wordt er automatisch een case aangemaakt in het systeem. Dit zorgt ervoor dat relevante meldingen van externe bronnen direct worden opgevolgd, zonder dat medewerkers handmatig cases hoeven aan te maken.

Waarom we marktberichten hebben toegevoegd

De functionaliteit voor marktberichten bestond al in het oude systeem (ES3). Marktberichten leveren waardevolle informatie over klantvragen, storingen of andere markt gerelateerde gebeurtenissen. In het nieuwe case-systeem zijn deze berichten beter geïntegreerd: ze worden automatisch omgezet in cases, waardoor medewerkers sneller kunnen reageren en alle relevante informatie overzichtelijk en gestructureerd binnen het systeem beschikbaar blijft.

Functionaliteiten van marktberichten

- Automatische casecreatie: elk binnenkomend bericht van Engie of Fluvius genereert automatisch een nieuwe case.
- Weergave binnen de detailpagina: medewerkers zien het gekoppelde marktbericht direct bij de case, inclusief alle relevante gegevens.
- Context en opvolging: medewerkers kunnen het bericht lezen, notities toevoegen en verdere acties plannen, waardoor het werkproces gestroomlijnd blijft.

Door marktberichten te integreren in het case-systeem wordt het opvolgen van externe meldingen efficiënter en blijft alle informatie gestructureerd en overzichtelijk.

3. Elegant

Elegant is een Belgische energieleverancier die actief is in heel Vlaanderen. Het bedrijf levert elektriciteit en aardgas aan particulieren en wil energie op een eerlijke, eenvoudige en transparante manier aanbieden. Elegant wil tonen dat energievoorziening niet ingewikkeld hoeft te zijn. Daarom zetten ze sterk in op duidelijke communicatie en eenvoudige processen.

De missie van Elegant is om klanten bewust te maken van hun energieverbruik en hen te ondersteunen richting een duurzamere toekomst. Door helder te communiceren en de klant centraal te zetten, probeert Elegant zich te onderscheiden van andere leveranciers.

3.1. Bedrijfsachtergrond en filosofie

Elegant ontstond vanuit het idee dat de energiemarkt eerlijker en transparanter kan. De organisatie wil contracten en facturatie begrijpelijk houden, zonder onnodige complexiteit. Ze geloven dat energie een basisdienst is die duidelijk en betrouwbaar moet zijn.

Hun belangrijkste waarden zijn:

- Eerlijkheid en transparantie: klanten moeten weten waar ze aan toe zijn.
- Eenvoud: geen moeilijk gedoe en verborgen kosten.
- Duurzaamheid: klanten helpen bewuster om te gaan met energie.
- Klantgerichtheid: een persoonlijke aanpak en snelle ondersteuning.

Elegant probeert in alles de drempel zo laag mogelijk te houden, zowel voor klanten als medewerkers.

3.2. Diensten en expertise

Elegant levert vooral aan particuliere klanten en biedt:

- **Energiecontracten voor elektriciteit en aardgas.**
  - Deze contracten zijn eenvoudig en duidelijk opgesteld, met transparante tarieven.
- **Klantendienst en ondersteuning.**
  - Het supportteam behandelt vragen rond facturen, meterstanden, verhuis, marktberichten en algemene problemen. Hierbij wordt veel intern gewerkt met cases en opvolgingstools.
- **Digitale processen.**
  - Elegant werkt voortdurend aan interne digitalisatie, zoals:
    - Dashboards.
    - CRM-modules.
    - Automatische verwerking van interne processen.

Deze digitale verbetering zorgt dat medewerkers efficiënter kunnen werken en klanten sneller geholpen worden.

4. Design mockups

Het ontwerp van het nieuwe case-systeem werd stap voor stap opgebouwd aan de hand van zowel low-fidelity als high-fidelity mockups. Deze mockups vormden een cruciale schakel tussen de analysefase en de uiteindelijke ontwikkeling. Ze hielpen bij het bepalen van de gebruikersstructuur, het visualiseren van functionaliteiten en het verzamelen van feedback van interne stakeholders.

4.1. Low-fidelity mockups (schetsen)

De ontwerpfase begon met low-fidelity mockups, handgetekende schetsen die in een vroeg stadium inzicht gaven in de structuur van de applicatie. Deze eenvoudige tekeningen focusten volledig op:

- Lay-out van de schermen.
- Positie van belangrijke elementen.
- Gebruikersflow tussen pagina's.
- Belangrijke functionaliteitsblokken zoals filters, tabellen en detailpanelen.

Door deze schetsen vroeg te bespreken met het customer support-team konden belangrijke aanpassingen snel worden doorgevoerd, nog voordat er tijd in visuele details of prototypes werd geïnvesteerd. De schetsen vormden dan ook de basis voor het verdere ontwerp.

4.2. Figma als ontwerptool

Voor het uitwerken van gedetailleerde mockups werd gebruikgemaakt van Figma, een web gebaseerde design – en prototyping tool die zeer geschikt is voor UI/UX-ontwerp. Figma biedt de mogelijkheid om in real-time samen te werken met andere teamleden. Dit maakte het ideaal voor ontwerpen van het case-systeem.

Figma werd gekozen omdat het meerdere voordelen bood:

- Toegankelijkheid: Figma werkt volledig in de browser, waardoor alle betrokkenen gemakkelijke toegang hadden tot de mockups zonder extra software te installeren.
- Samenwerking: zowel het ontwikkelingsteam als het customer support-team konden naar de mockups kijken, opmerkingen plaatsen en feedback geven.
- Consistentie: dankzij componenten, stijlen en grids kon een uniform design worden opgebouwd dat in elke pagina terugkomt.
- Snel itereren: ontwerpen konden eenvoudig aangepast en opnieuw gedeeld worden wanneer functionaliteiten of inzichten veranderden.
- Figma fungeerde hierdoor als een belangrijke schakel tussen de analysefase, het visuele ontwerp en de daadwerkelijke implementatie.

4.3. Lovable als aanvullende prototyping tool

Naast Figma werd ook gebruikgemaakt van Lovable, een AI-gedreven prototyping tool die het mogelijk maakt om snel interactieve en realistisch aanvoelende mockups te genereren. Waar Figma voornamelijk werd ingezet voor het visuele UI-ontwerp en consistente componentstructuren, werd Lovable gebruikt om de werking en interacties van het case-systeem concreet te demonstreren.

Lovable bood hierbij enkele duidelijke voordelen

- Snelle visualisatie van functionaliteit en gebruikersflows.
- Mogelijkheid om interacties en schermovergangen te tonen.
- Betere begrijpbaarheid voor niet-technische werknemers.
- Snelle validatie van ideeën vóór implementatie.

Lovable fungeerde hiermee als een aanvullende brug tussen statische mockups en de uiteindelijke applicatie.

4.4. High-fidelity mockups in Figma

Na de low-fidelity schetsen werden de ontwerpen vertaald naar high-fidelity mockups in Figma. Deze mockups benaderen de uiteindelijke look-and-feel van de applicatie en werden gebruikt om:

- Visuele stijl en kleuren te bepalen.
- Componenten (tabbellen, knoppen, modals) consistent te maken.
- Duidelijke feedbackmomenten te organiseren met het ontwikkelingsteam en customer support.

De high-fidelity mockups fungeerden als ontwerpkader en communicatiemiddel, maar vormden geen definitief ontwerp. Het uiteindelijke product verschilt op meerdere punten van de mockups, doordat de look-and-feel en interacties gedurende het ontwikkelingsproces iteratief zijn aangepast in overleg met developers en customer support, zodat het ontwerp zowel technisch uitvoerbaar als gebruiksvriendelijk bleef.

4.5. Case overzichtspagina

De case overzichtspagina is ontworpen als het startpunt voor medewerkers. Ze biedt een gestructureerd en overzichtelijk beeld van alle lopende en afgewerkte cases. Dit scherm speelt een centrale rol in het werkproces, omdat medewerkers van hieruit snel kunnen navigeren naar detailpagina's of acties kunnen ondernemen.

Belangrijkste elementen

Zoek- en filtermogelijkheden: medewerkers kunnen cases filteren op klantnummer, status, prioriteit, due date en andere relevante parameters.

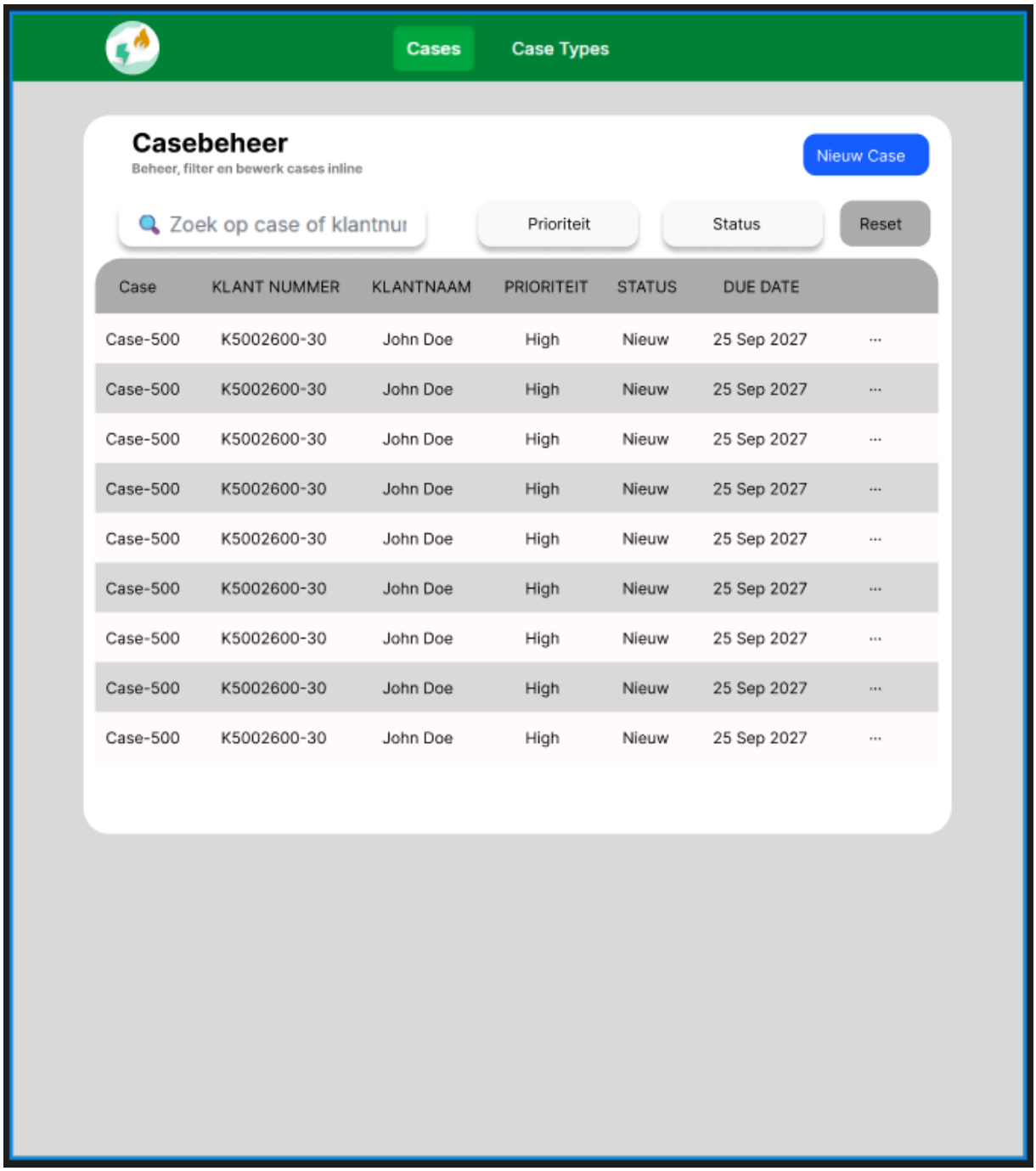
Datatabel met cases: elk record toont de belangrijkste basisinformatie, zoals case-ID, klantnaam, status, prioriteit en uiterste datum ('due date').

Actiemenu's: vanuit elke caserecord kan snel doorgeklikt worden naar de detailpagina of kunnen bepaalde acties inline uitgevoerd worden.

Doel van dit ontwerp

Het hoofddoel van deze pagina is een duidelijk, snel doorzoekbaar en gebruiksvriendelijk overzicht aanbieden. Medewerkers kunnen hierdoor efficiënter navigeren, prioriteiten bepalen en sneller reageren op urgente cases, terwijl het ontwerp een moderne en frisse uitstraling behoudt.

Figuur 1 – Case overzicht pagina mockup



4.6. Detailpagina van een case

De detailpagina vormt het centrale onderdeel van het case-systeem. Hier vinden medewerkers alle relevante informatie over een specifieke case, gegroepeerd in overzichtelijke secties. Het ontwerp is bewust eenvoudig en logisch opgebouwd zodat medewerkers snel inzicht krijgen in de status en de inhoud van de case.

Voor de case-detailpagina werd een interactieve mockup uitgewerkt met Lovable. Deze mockup werd gebruikt om de werking van de detailpagina te demonstreren, zoals het navigeren tussen tabs, het raadplegen van casegegevens en het bekijken van gerelateerde informatie.

Dankzij Lovable kon op een snelle manier een gedetailleerde en functionele mockup worden opgebouwd, waardoor werknemers niet alleen het ontwerp, maar ook de gebruikservaring en interacties konden beoordelen. Dit maakte het eenvoudiger om feedback te verzamelen en verbeteringen door te voeren vóór de effectieve ontwikkeling.

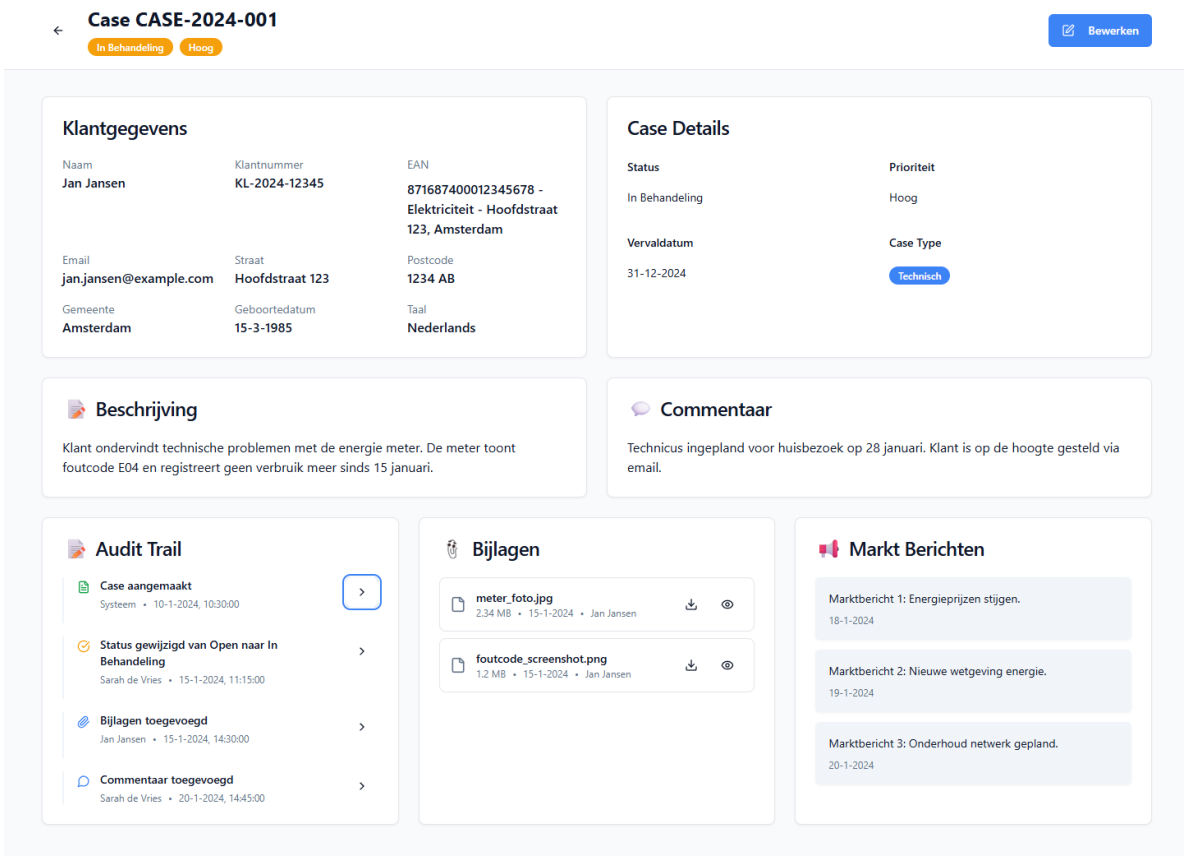
Belangrijkste elementen

- Klantgegevens: klantnummer, contactinformatie, adres, EAN-code en andere essentiële klantinformatie.
- Case-informatie: type case, beschrijving, status, prioriteit, verantwoordelijke medewerker en datum van aanmaak.
- Tabs voor aanvullende informatie:
    - Attachments.
    - Audit log.
    - Marktberichten.

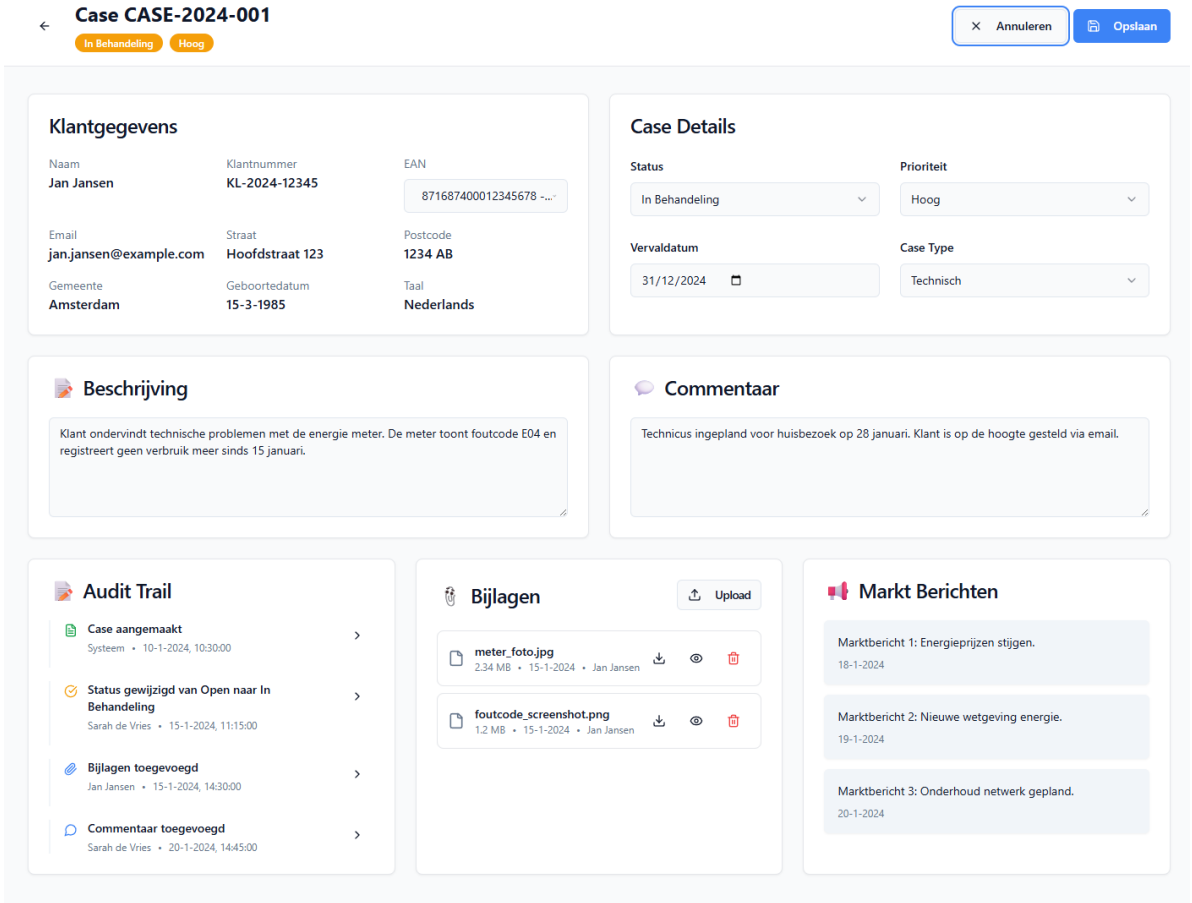
Doel van dit ontwerp

Alle informatie die relevant is voor het behandelen van een case wordt op één pagina samengebracht. Hierdoor moeten medewerkers niet tussen verschillende schermen navigeren en kunnen zij efficiënter en foutloos werken.

Figuur 3 - Case details mockup



Figuur 2 - Case details edit mockup



4.7. Case types overzichtspagina

Naast het case overzicht werd ook een aparte pagina ontworpen voor het beheren van case types. Deze pagina gebruikt een tabelstructuur, vergelijkbaar met de weergave van cases, waardoor alle case types overzichtelijk en gestructureerd worden weergegeven.

Belangrijkste elementen

Zoek- en filtermogelijkheden: medewerkers kunnen eenvoudig het overzicht doorzoeken of filteren titel, actief.

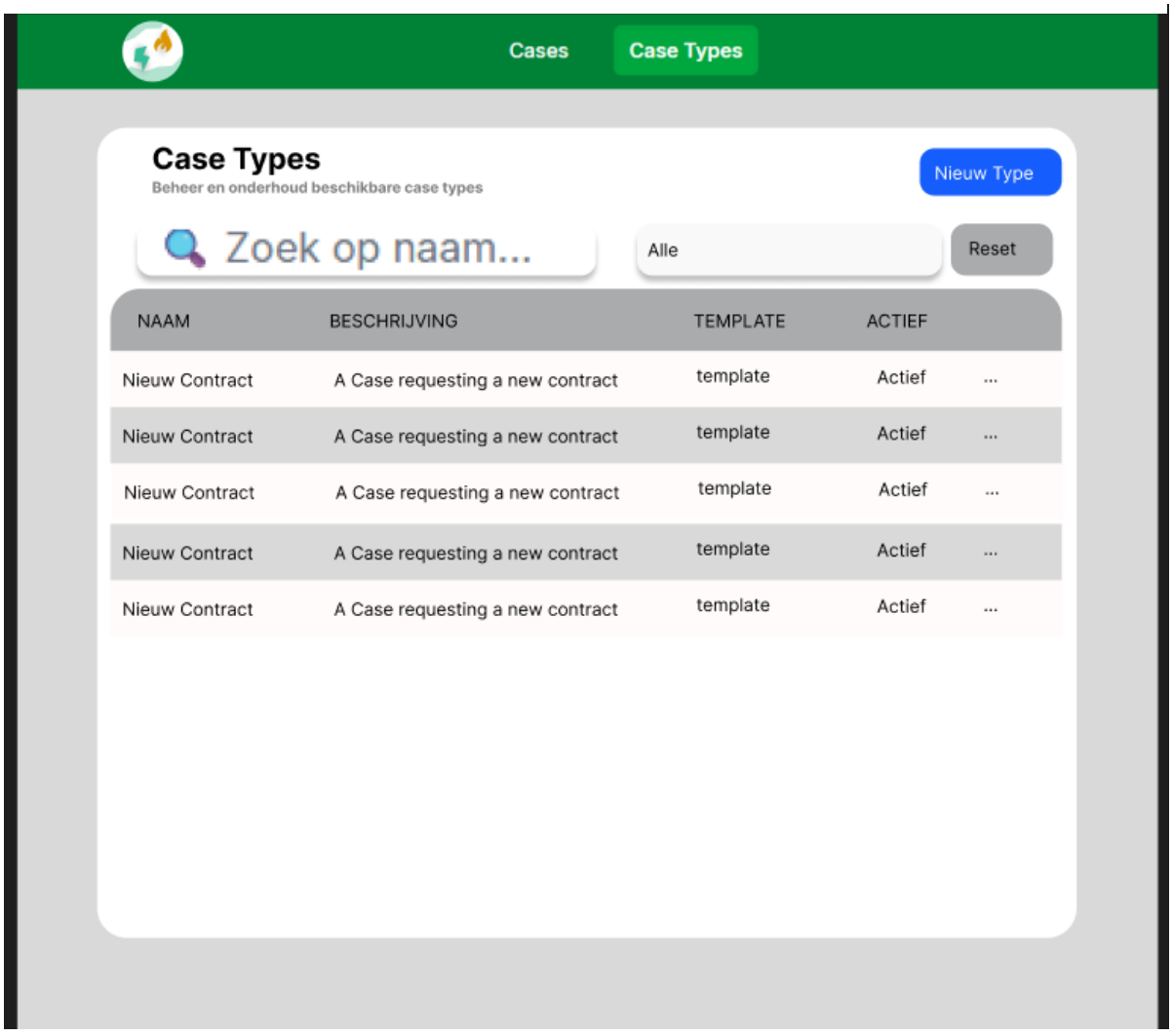
Datatabel met case types: elk record toont de informatie van een case type, naam, beschrijving, template en actief.

Actiemenu: elk case type kan je inline bewerken. Er is geen mogelijkheid om ze te verwijderen via de UI.

Doel van dit ontwerp

De tabelstructuur biedt een duidelijk overzicht van alle case types en stelt medewerkers in staat om efficiënt te werken. Door de uniforme opzet en overzichtelijke kolommen kan men snel het juiste case type selecteren, wijzigen of beheren, waardoor de kans op fouten wordt verminderd en het beheer van case types sneller verloopt.

Figuur 4 - Case types mockup



5. Microsoft Authenticatie

Binnen dit project speelt authenticatie een cruciale rol in het waarborgen van veiligheid, toegangscontrole en het correct toewijzen van rechten aan gebruikers. Elegant maakt gebruik van een VPN waarbij toegang tot applicaties alleen mogelijk is via een Microsoft-account. Deze infrastructuur vormt de basis voor het toegangsbeheer en wordt in dit project als uitgangspunt genomen.

De frontend van de applicatie haalt bij het gaan naar de applicatie automatisch op wie er is ingelogd op de computer. Deze gebruikersinformatie wordt vervolgens meegestuurd bij API-aanroepen, zodat het systeem kan registreren welke gebruiker welke acties uitvoert. Op deze manier kan nauwkeurig worden gecontroleerd wie toegang heeft tot welke functionaliteiten en welke rechten zij hebben.

Authenticatieproces

- Inloggen via Microsoft-account: toegang tot het systeem verloopt via het Microsoft-authenticatie mechanisme, dat veilig en betrouwbaar is.
- Gebruikersidentificatie: de frontend bepaalt welke gebruiker actief is en stuurt deze informatie mee met API-aanvragen.
- Beveiliging en controle: door deze integratie kan de backend bij elke aanvraag verifiëren of de gebruiker de juiste rechten heeft, waardoor ongeautoriseerde toegang wordt voorkomen.

Groepen en rollen

Het project maakt gebruik van een fijnmazig toegangsmodel op basis van groepen en rollen. Dit model zorgt ervoor dat functionaliteiten en gevoelige acties alleen beschikbaar zijn voor bevoegde gebruikers.

Gecontroleerde groepsindeling

Bij het lokaal testen van de implementatie wordt gecontroleerd tot welke groep een gebruiker behoort. Dit betekent dat gebruikers niet automatisch, maar bewust en gecontroleerd aan specifieke groepen worden toegevoegd. Zo kan nauwkeurig worden getest of rol gebaseerde rechten correct functioneren voordat het systeem wordt geïntegreerd in de volledige Elegant-omgeving.

Roltoewijzing

Afhankelijk van de toegewezen groep(en) krijgt een gebruiker specifieke rollen, zoals:

- Casebeheerder: mag cases aanmaken, bewerken en verwijderen.
- Teamlid: mag cases bekijken en bewerken, maar niet verwijderen.

Deze aanpak maakt het mogelijk om functies en acties op gebruikersniveau af te schermen en te zorgen voor een veilig en gecontroleerd systeem.

Backend autorisatie

De backend valideert bij elke API-aanvraag of de gebruiker beschikt over de juiste rol voor de gevraagde actie. Hierdoor wordt ongeautoriseerde toegang actief geblokkeerd en blijft het systeem veilig, zelfs bij pogingen om handmatig API-aanvragen te manipuleren.

Toepassing in het project

Het case systeem is een relatief complexe module met veel logica en methodes. Door het rollen- en groepsmodel kan de applicatie precies bepalen wie cases mag verwijderen, bewerken of enkel bekijken.

Conclusie

Het authenticatie- en autorisatie model vormt een essentieel onderdeel van de beveiligingsarchitectuur binnen het project. Door gebruik te maken van Microsoft-authenticatie, het gecontroleerd toewijzen van gebruikers aan groepen en het inzetten van een rol gebaseerd rechtenmodel, wordt een veilig, betrouwbaar en gestructureerd toegangsbeheer gegarandeerd.

Dankzij de koppeling tussen frontend-gebruikersidentificatie en backend-autorisatie kan het systeem nauwkeurig bepalen wie welke acties mag uitvoeren. Dit voorkomt ongeautoriseerde toegang en zorgt ervoor dat gevoelige functionaliteiten enkel beschikbaar zijn voor bevoegde gebruikers.

De lokale implementatie van het groepen- en rollenmodel maakt het mogelijk om de volledige functionaliteiten uitgebreid te testen voordat deze wordt geïntegreerd in het Elegant-systeem. Hierdoor is de basis gelegd voor een toekomstbestendige, schaalbare en veilige authenticatie-oplossing die probleemloos kan meegroeien met de verdere ontwikkeling van het project.

6. Database structuur en werkwijze Elegant

6.1. Inleiding

Bij Elegant wordt de database-architectuur zorgvuldig ontworpen om zowel veiligheid als efficiëntie te waarborgen. Een kerncomponent in deze architectuur is het Backend-for-Frontend (BFF) patroon, dat dient als een tussenlaag tussen de frontend en de backend-services. Dit patroon stelt Elegant in staat om de database te beschermen tegen ongewenste of foutieve query's, terwijl de frontend eenvoudig en efficiënt toegang krijgt tot de benodigde data.

6.2. Wat is Backend-for-Frontend (BFF)?

Backend-for-Frontend is een architecturaal patroon waarbij een specifieke backend laag wordt ontwikkeld voor een specifieke frontend. In plaats van dat een frontend direct communiceert met meerdere backend-services, verzorgt de BFF deze communicatie en biedt het een geoptimaliseerde interface voor de cliënt.

De belangrijkste kenmerken van een BFF zijn:

1. Frontend-specifiek: elk frontend (web, mobiel, tablet) kan een eigen BFF hebben die inspeelt op de unieke behoeften van de cliënt.

2. Data-aggregatie en mapping: de BFF kan data uit verschillende backend-services combineren en transformeren naar een formaat dat direct bruikbaar is voor de frontend.

3. Beveiliging en validatie: ongeautoriseerde of foutieve requests worden geblokkeerd voordat ze de kernservices of database bereiken.

4. Vermindering van frontend logica: de frontend hoeft minimale modellen of transformaties te implementeren, omdat de BFF deze taken al uitvoert.

6.3. Werkwijze bij Elegant

Bij Elegant fungeert de BFF als gateway tussen de frontend en de backend, waardoor het systeem een duidelijke scheiding van verantwoordelijkheden heeft:

1. Frontend: stuurt requests naar de BFF in plaats van rechtstreeks naar de backend
2. Bff/gateway:
    a. Valideert en filtert binnenkomende requests.
    b. Haalt data op uit één of meerde backend/gateway services.
    c. Transformeert de data zodat deze direct bruikbaar is voor de frontend.
3. Frontend ontvangt: de data is kant-en-klaar, waardoor extra modellering of transformatie in de frontend overbodig is.

6.4. Bescherming van de database

De implementatie van BFF biedt meerdere voordelen op het gebied van veiligheid en integriteit:

- Preventie van foutieve calls: alleen goedgekeurde requests bereiken de backend.
- Rate limiting: vermindert het risico op overbelasting van de database.
- Centrale datatransformatie: alle mapping en formating van data gebeurt binnen de BFF, waardoor foutgevoelige client-logica wordt vermeden.

Door deze aanpak wordt het risico op database-corruptie, onnodige belasting of veiligheidsproblemen aanzienlijk beperkt.

6.5. Data Mapping en modellen

Een van de grootste voordelen van BFF is dat frontend vrijwel geen eigen datamodellen hoeft te onderhouden. De BFF kan:

- Data uit meerdere services combineren tot één coherent object.
- Complexe data structuren en transformeren naar een eenvoudig, consistent viewmodel voor de frontend.
- Verandering in de onderliggende backend centraal afhandelen, zonder dat iedere frontend hierop moet worden aangepast.

Voorbeeldconcept
Het case systeem heeft een detailscherm met heel veel informatie.

- Details van de case.
- Klantinformatie.
- Marktberichten

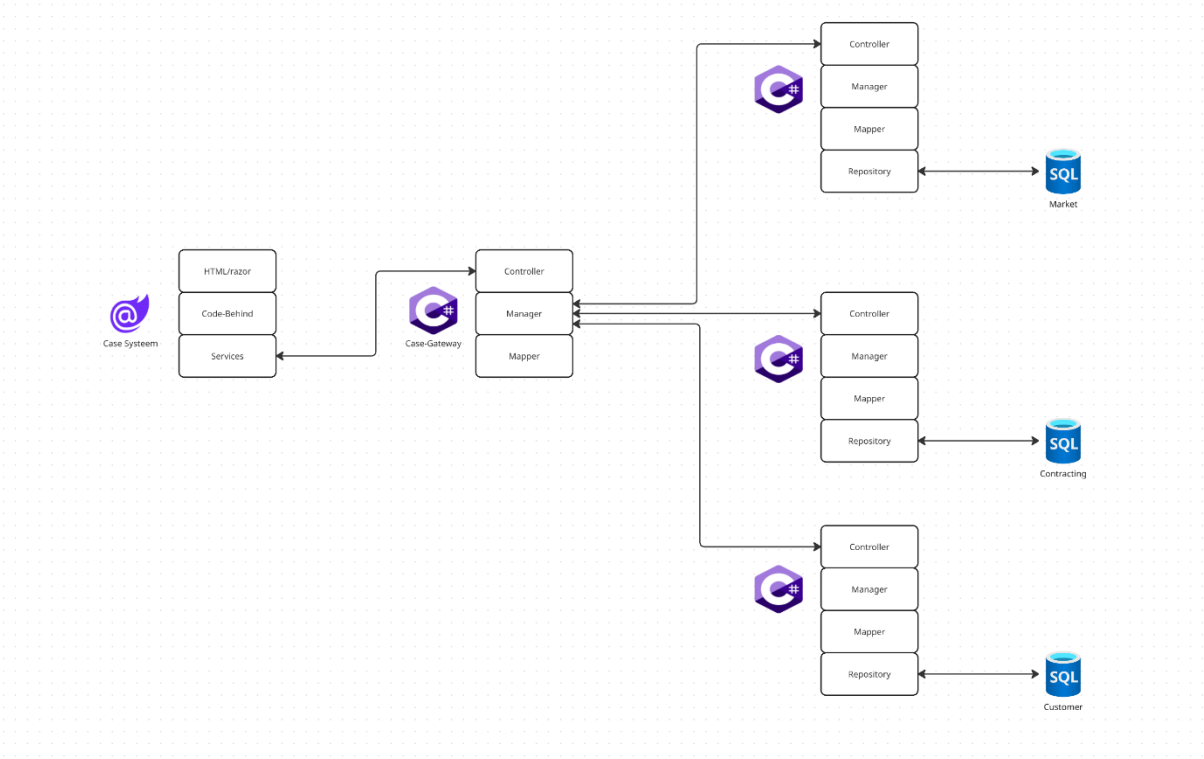
Als we dit niet via BFF zouden doen dan moet de frontend beide services aanspreken en de data zelf combineren.

Dit is niet wat we willen natuurlijk. Het maakt het niet herbruikbaar en geeft te veel logica in de frontend. Met de BFF stuurt de frontend één request, en de BFF haalt beide datasets op, combineert ze en stuurt één gestructureerde response terug.

6.6. Visualisatie van de werkwijze

Dit om de werking van het BFF-patroon bij Elegant beter te begrijpen en om te tonen hoe ik het heb gebruikt bij mijn stage. In de figuur is een schematische weergave opgenomen. Het diagram toont de interactie tussen de frontend, de BFF (gateway) en de backend-services/database.

Figuur 5 - BFF visualisatie



7. Gebruikte tools

Bij het ontwikkelen van softwareprojecten is het inzetten van de juiste tools één van de meest bepalende factoren voor een efficiënt en kwalitatief ontwikkelingsproces. Tools ondersteunen zowel de technische implementatie als de samenwerking, communicatie en documentatie binnen het projectteam. Binnen dit project werd een breed en zorgvuldig geselecteerd scala aan tools gebruikt, afgestemd op de functionele behoeften van het project en op de best practices binnen de industrie.

In deze sectie wordt een overzicht gegeven van alle gebruikte tools, logisch onderverdeeld per toepassingsgebied. Voor projectorganisatie en communicatie werden Trello en Microsoft 365 ingezet. Deze tools zorgden voor een gestructureerde workflow, duidelijke taakverdeling en efficiënte communicatie tussen mezelf en mijn stagementor.

Op het vlak van versiebeheer en samenwerking rond broncode werd gebruik gemaakt van Bitbucket in combinatie met Sourcetree. Bitbucket fungeerde als centrale Git-repository voor het opslaan en beheren van branches aanzienlijk vereenvoudigd. Deze combinatie zorgde voor overzichtelijke versiecontrole en een gestroomlijnde workflow tijdens de ontwikkelfases.

Voor het testen en documenteren van API-endpoints werd Swagger gebruikt, een tool die het mogelijk maakt om REST-API's interactief te valideren en te visualiseren. De primaire ontwikkelomgeving was Visual Studio, waarbinnen de backendlogica, api-functionaliteiten en andere softwarecomponenten werden ontwikkeld, getest en gedebugd. Verder werd Github Copilot, een AI-ondersteunende programmeer assistent, ingezet om codevoorstellen te genereren en repetitieve programmeertaken te versnellen, wat de productiviteit merkbaar verhoogde.

Tot slot werd voor databasebeheer en -optimalisatie gebruikgemaakt van SQL Server Management Studio (SSMS). Deze tool speelde een cruciale rol bij het analyseren van query's, het beheren van tabellen en het onderhouden van de volledige database-architectuur.

De keuze voor deze tools is gebaseerd op beproefde industriepraktijken en op de specifieke technische vereisten van het project. In de volgende paragrafen wordt elke tool afzonderlijk toegelicht, inclusief de motivatie voor het gebruik ervan en de rol die het vervulde binnen het ontwikkelingsproces.

7.1. Projectmanagement

Tijdens dit project werd gebruikgemaakt van verschillende tools om de opdrachten on track te houden, de communicatie te ondersteunen en overzicht te bewaren over de belangrijkste taken en aandachtspunten. Deze tools werden niet ingezet als strikt planningsinstrument, maar eerder als ondersteuning om structuur, duidelijkheid en continuïteit binnen het project te garanderen.

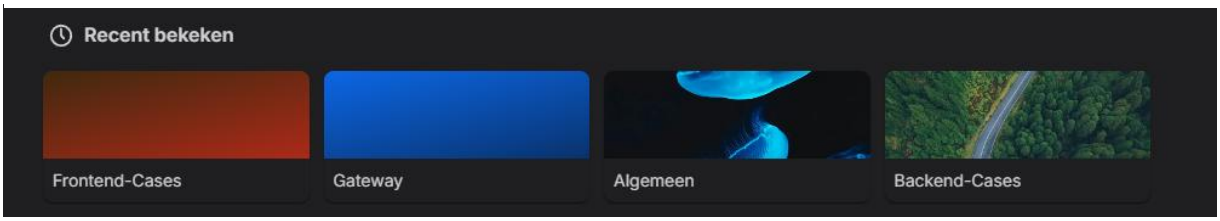
Door het gebruik van deze projectmanagement- en communicatietools konden taken tijdig worden opgevolgd, afspraken worden vastgelegd en feedback overzichtelijk worden verwerkt. Dit droeg bij aan een efficiënte samenwerking en een vlotte voortgang van het project.

7.2. Trello

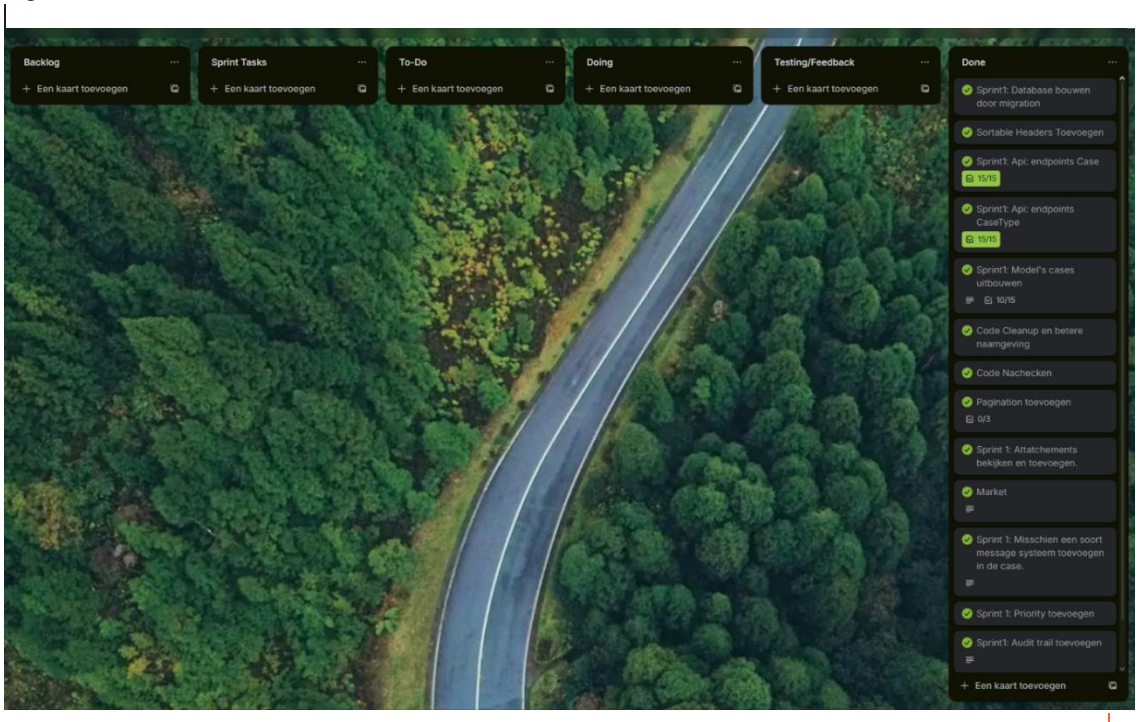
Trello werd gebruikt als ondersteunende tool om specifieke stappen en aandachtspunten binnen het project bij te houden en niet te vergeten. In plaats van het volledige ontwikkelingsproces te plannen, fungeerde Trello voornamelijk als overzicht om losse taken, reminders en opvolgpunten te registreren. Op die manier bleef belangrijke informatie bewaard en konden acties op het juiste moment worden opgevolgd.

Er werd niet gewerkt met één centraal Trello-bord, maar met meerdere borden, onderverdeeld in **Algemeen**, **Backend**, **Gateway** en **Frontend**. Deze opdeling zorgde voor extra overzicht en droeg bij aan structuur en duidelijkheid in de samenwerking tussen mijn stagementor en mij, zonder dat Trello als strikt planningsinstrument werd ingezet.

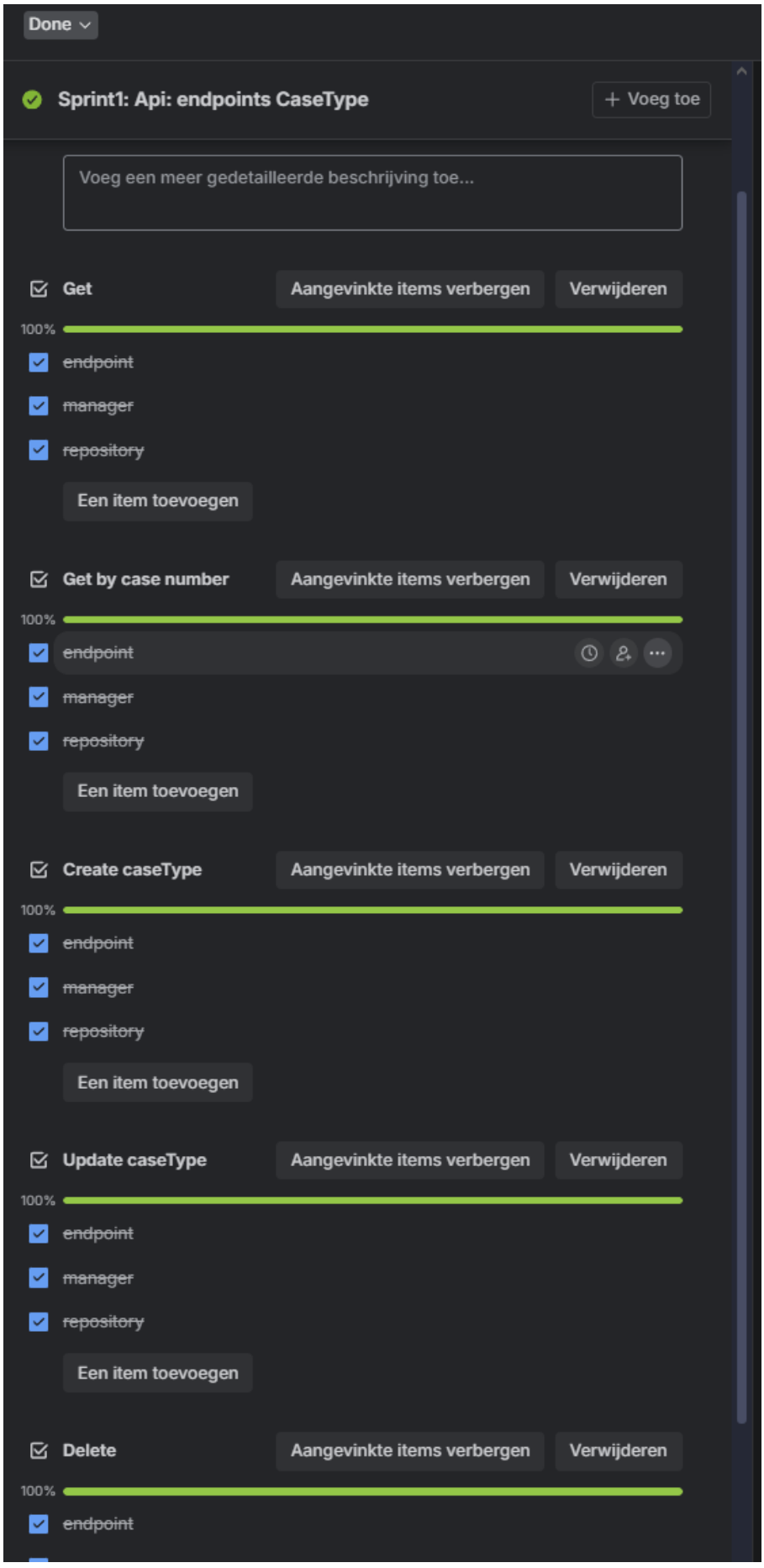
Figuur 6 - Trello overzicht



Figuur 7 - Trello backend



Figuur 8 - Trello API point details



7.3. Microsoft 365

Microsoft 365 ondersteunde de communicatie en documentatie gedurende het project. Tools zoals Microsoft Teams, Outlook, Onedrive en Word werden frequent gebruikt voor overlegmomenten, bestanden delen en het opstellen van documentatie. De integratie tussen deze tools zorgde ervoor dat informatie consistent en overzichtelijk opgeslagen werd.

Conclusie

Het gebruik van Trello en Microsoft 365 zorgde voor een gestructureerde, transparante en goed georganiseerde workflow. Taken konden duidelijk worden opgevolgd, communicatie bleef overzichtelijk en documentatie werd centraal beheerd. Deze combinatie verbeterde de samenwerking met mijn stagementor en droeg bij aan een efficiënte planning en vlotte projectvoortgang.

7.4. Versiebeheer

7.5. Bitbucket

Bitbucket fungeerde als de Centrale Git-repository. Alle broncode van het project werd hierin beheerd via verschillende branches voor onder andere nieuwe functionaliteiten, bugfixes en experimenten.

Door het gebruik van Bitbucket bleef de code

- Voorzien van versiecontrole.
- Veilig opgeslagen en toegankelijk.
- Gemakkelijk te beheren.
- Volledig traceerbaar.

Deze tool speelde een sleutelrol in het waarborgen van samenwerking en stabiliteit binnen het project.

7.6. Sourcetree

Sourcetree werd gebruikt als grafische Git-client. In plaats van Git-commando's manueel in de terminal uit te voeren, bood Sourcetree een visuele interface voor:

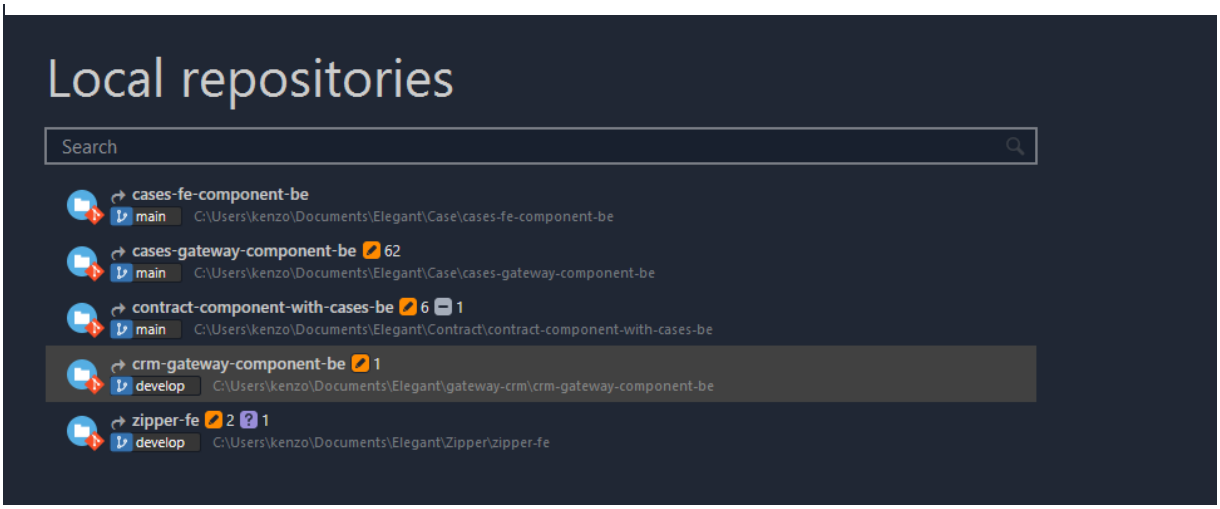
- Commits.
- Merges.
- Resolving conflicts.
- Branchbeheer.

Dit verminderde de kans op fouten en maakte het eenvoudiger om complext Git-operaties correct uit te voeren.

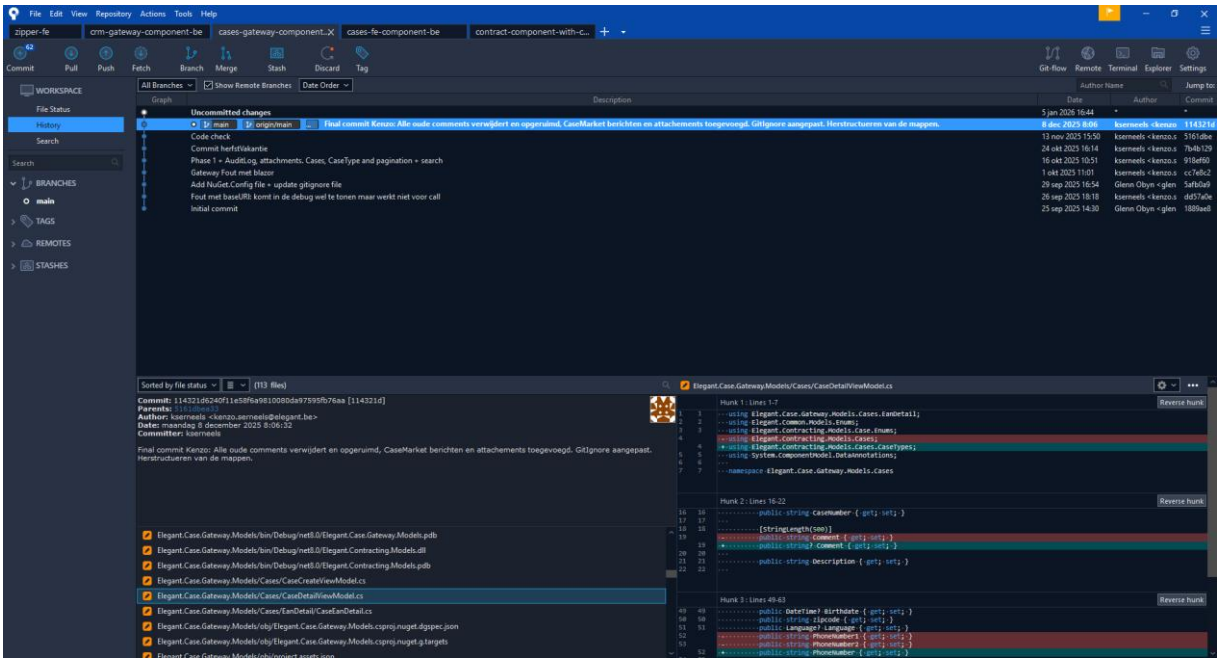
Conclusie

Bitbucket en Sourcetree vormden een stabiele en betrouwbare basis voor versiebeheer binnen het project. Door de duidelijke branchstructuren en de visuele Git-interface konden wijzigingen veilig worden doorgevoerd en conflictsituaties eenvoudig worden opgelost. Dit resulteerde in een consistente codebase, minder technische fouten en een efficiëntere ontwikkelingsworkflow.

Figuur 9 - Overzicht repositories



Figuur 10 - Dieper inzicht repository



7.7. API

Swagger

Swagger werd ingezet voor het testen en documenteren van API's. Het bood een interactieve interface waarin endpoints konden worden bekeken, getest en gevalideerd. Dit zorgde ervoor dat:

- API's eenvoudig gecontroleerd konden worden.
- Foutieve request-structuren snel werden opgespoord.
- Documentatie automatisch gegenereerd werd.
- Swagger vergrootte de kwaliteit en betrouwbaarheid van de API's binnen dit project.

Conclusie

Swagger bleek een onmisbare tool voor het testen, documenteren en valideren van de API. Dankzij de interactieve interface konden endpoints snel gecontroleerd worden en werden foutieve request- of response-structuren vroegtijdig gedetecteerd. Dit verbeterde zowel de kwaliteit als de betrouwbaarheid van de API en versnelde het ontwikkel- en testproces aanzienlijk.

7.8. IDE

Visual studio

Visual Studio fungeerde als de primaire ontwikkelomgeving binnen dit project. Deze IDE werd gebruikt voor zowel de backend-ontwikkeling als de ontwikkeling van de Blazor-frontend. Visual Studio bood een stabiele en uitgebreide omgeving waarin alle onderdelen van de applicatie konden worden geschreven, getest en gedebugd.

Binnen de backend werd Visual Studio gebruikt voor het ontwikkelen van de API-functionaliteiten, het opzetten van de architectuur en het integreren van Businesslogica. Dankzij de ingebouwde ondersteuning van .NET en C# konden ontwikkeltaken efficiënt worden uitgevoerd, waarbij IntelliSense en debuggingtools hielpen om fouten snel op te sporen en op te lossen.

Daarnaast werd Visual Studio ook ingezet voor de Blazor-frontend. Omdat Blazor een .Net-gebaseerd framework is, sloot Visual Studio naadloos aan bij de technologieën die in dit project gebruikt werden. Dit maakte het eenvoudig om componenten te ontwikkelen, databinding toe te passen en UI-functies snel te testen, ondanks het feit dat de frontend in een afzonderlijk solution zat.

De belangrijkste voordelen van Visual Studio binnen dit project waren:

- Volledige ondersteuning voor .NET, C# en Blazor, wat essentieel was voor zowel backend-als frontendontwikkeling.
- Krachtige debuggingtools, zoals breakpoints, variabeleninspectie en live debugging.
- Duidelijke projectstructuren per solution, wat hielp om backend en frontend logisch van elkaar gescheiden te houden.
- NuGet Package Manager voor het beheren van externe libraries en dependencies.

Dankzij de veelzijdigheid en stabiliteit van Visual Studio kon zowel de backend- als de frontendontwikkeling op een gestructureerde, efficiënte en betrouwbare manier worden uitgevoerd, zelfs zonder gedeelde solution.

Conclusie

Visual Studio bood een krachtige, stabiele en geïntegreerde omgeving voor zowel backend- als frontend ontwikkeling. De uitgebreide debugging mogelijkheden, IntelliSense-ondersteuning en naadloze integratie met .NET maakten het ontwikkelen efficiënter en foutbestendiger. Hierdoor kon de software op een gestructureerde en professionele manier worden opgebouwd, ondanks de scheiding tussen backend- en frontendsolutions.

7.9. AI tools

Binnen het project zijn verschillende AI-gebaseerde tools ingezet om het ontwikkelingsproces te versnellen, codekwaliteit te verbeteren en repetitieve taken te automatiseren. De belangrijkste AI-tools die gebruikt werden, zijn Github Copilot, ChatGPT en Claude Ai. Deze tools ondersteunden zowel de backend- als frontend ontwikkeling en hielpen bij de documentatie en probleemoplossing.

Github Copilot

GitHub Copilot fungeerde als AI-gebaseerde programmeer assistent binnen Visual Studio. De tool bood suggesties en volledige codefragmenten op basis van context en geschreven code. Copilot hielp bij:

- Het genereren van codevoorstellen, waardoor ontwikkeltijd werd verkort.
- Het versnellen van repetitieve programmeertaken, zoals het schrijven van loops, Crud-operaties of standaardfuncties.
- Verhogen van productiviteit, doordat standaardpatronen automatisch werden aangeleverd.
- Verminderen van implementatiefouten, doordat de Ai vaak syntaxis- en structurele correcties suggereerde.
- Integratie met de IDE, aangezien Github Copliot naadloos binnen Visual Studio functioneert en Elegant beschikte over een eigen licentie, wat deze keuze eenvoudig en efficiënt maakte.

Copilot ondersteunde het schrijfproces en zorgde ervoor dat veelvoorkomende programmeerstappen sneller,consistenter en betrouwbaarder werden uitgevoerd.

ChatGPT

ChatGPT werd voornamelijk gebruikt als ondersteunend hulpmiddel bij documentatie, analyse en probleemoplossing. De toepassingen binnen dit project omvatten:

- Genereren en structureren van technische documentatie, zoals API-handleidingen, workflow beschrijving en interne projectnotities.
- Ondersteunen bij analyse en kennisoverdracht, door uitleg te geven over complext concepten of codefragmenten.
- Assisteren bij code review en best practices, bijvoorbeeld suggesties voor verbetering van leesbaarheid, structuur of conventies.

ChatGPT werd vooral ingezet voor documentatie en kennismanagement, waardoor het team sneller en consistenter kon werken en belangrijke informatie centraal en toegankelijk bleef.

Claude AI

Claude AI werd gebruikt als gespecialiseerde ondersteuning voor de frontend ontwikkeling, met focus op design en visuele componenten. Belangrijke toepassingen waren:

- Ondersteuning bij UI-design en Tailwind CSS, door suggesties te geven voor componentopbouw, styling en responsivelayouts.
- Analyseren van frontend-structuren en workflows, waardoor de implementatie van herbruikbare componenten efficiënter verliep.
- Optimaliseren van visuele consistentie en UX, door feedback te genereren in designkeuzes en lay-out structuren.

Door Claude AI te gebruiken, kon ik sneller en beter gefundeerde keuzes maken bij frontend ontwikkeling vooral bij het ontwerpen van componenten en het toepassen van consistente stylingpatronen.

Conclusie

De inzet van AI-tools leverde een directe meerwaarde op binnen het ontwikkelingsproces. Github Copilot versnelde codering en verminderde repetitieve taken, ChatGPT verbeterde documentatie en technische toelichting, en Claude AI ondersteunde vooral bij frontenddesign en Tailwind-componenten. Deze combinatie zorgde voor een hogere productiviteit, betere codekwaliteit en snellere probleemoplossing.

7.10. Database

Databases vormen de kern van vrijwel elke moderne softwaretoepassing. Ze zijn verantwoordelijk voor het efficiënt opslaan, beheren en ophalen van gegevens en ondersteunen zo de stabiliteit en betrouwbaarheid van het volledige systeem. Binnen dit project wordt gewerkt met een rationele database, waarbij SQL (Structured Query Language) wordt gebruikt voor het uitvoeren van query's en het manipuleren van data. Een zorgvuldig en gestructureerd databasebeheer is essentieel voor de prestaties, beveiliging en consistentie van de applicatie.

Voor dit project werd voornamelijk gebruikgemaakt van SQL Server Management Studio (SSMS), een professionele en veelzijdige beheeromgeving voor SQL Server-databases. SSMS biedt een overzichtelijke interface voor het onderhouden van datamodellen, het uitvoeren van query's en het monitoren en optimaliseren van databaseprestaties.

SSMS / SQL Server Management Studio

SQL Server Management Studio (SSMS) is de officiële beheertool van Microsoft voor SQL Server en vormde een cruciaal onderdeel van het databasebeheer binnen dit project. De tool biedt een uitgebreide en gebruiksvriendelijke omgeving voor het creëren, wijzigen en beheren van databaseobjecten, zoals tabellen, views, indexen, relaties en gebruikersrechten.

Binnen dit project werd SSMS gebruikt voor onder meer:

- Het ontwerpen en beheren van databasestructuren.
- Het schrijven, testen en optimaliseren van SQL-query's.
- Het analyseren en verbeteren van databaseprestaties.
- Het beheren van beveiligingsinstellingen, zoals gebruikersrechten en permissies.
- Het opzetten en onderhouden van testdata.
- Het werken met stored procedures, triggers en transacties.

Een van de grootste sterktes van SSMS is de naadloze integratie met Microsoft SQL Server, waardoor complexe datamodellen en grote datasets efficiënt en betrouwbaar beheerd kunnen worden. De tool beschikt bovendien over krachtige analysetools en debuggingfunctionaliteiten die helpen bij het identificeren en oplossen van prestatieproblemen.

Door het gebruik van SSMS kon de database gedurende het hele project op een gestructureerde, veilige en professionele manier beheerd worden. Dit droeg rechtstreeks bij aan de stabiliteit, betrouwbaarheid en presentaties van de backend-API en alle andere componenten die afhankelijk zijn van de onderliggende databank.

Conclusie

SSMS bood een professionele en overzichtelijke omgeving om de SQL Server-database ontwerpen, te beheren en te optimaliseren. Door de sterke analystetools, debuggingfunctionaliteiten en duidelijke structuur konden datamodellen efficiënt onderhouden worden en prestaties gericht worden verbeterd. Dit verhoogde de stabiliteit en betrouwbaarheid van de backend en alle afhankelijke applicatiecomponenten.

8. Technologieën

In dit hoofdstuk worden alle technologieën besproken die zijn gebruikt bij de ontwikkeling van het project. Elke technologie vervult een specifieke rol binnen de architectuur en werking van de applicatie en ondersteunt zowel de frontend- als backend ontwikkeling, databasebeheer en styling. De keuzes zijn gemaakt op basis van betrouwbaarheid, efficiëntie en best practices in de industrie.

8.1. .NET 8

.NET 8 is het open-source, cross-platform framework van Microsoft dat wordt gebruikt voor het ontwikkelen van moderne applicaties. Het stelt ontwikkelaars in staat om zowel backend- als frontend logica te implementeren met C#, met een consistente programmeerervaring.

Belangrijke kenmerken van .NET 8

- Cross-platform: applicaties kunnen draaien op Windows, Linux en macOS.
- Sterke integratie met Blazor: ondersteunt zowel server-side als client-side webontwikkeling.
- Ondersteuning voor moderne architecturen: API's, microservices en cloudapplicaties.
- Performance en veiligheid: geoptimaliseerd voor snelle verwerking en veilige datacommunicatie.

Toepassing in het project

.Net 8 werd gebruikt als fundament voor zowel de backend-API's als de backend- en frontendlogica. Het bood een consistente programmeerervaring en maakte integratie met Blazor, de database en AI-tools eenvoudig en efficiënt.

Conclusie

.Net 8 zorgde voor een solide en toekomstbestendige basis voor de volledige applicatie, waarbij zowel prestaties als veiligheid gewaarborgd werden en integratie tussen backend, frontend en externe tools soepel verliep.

8.2. Blazor

Blazor is een modern webframework binnen het .NET-ecosysteem waarmee interactieve webapplicaties kunnen worden ontwikkeld met C# in plaats van JavaScript. Het framework stelt ontwikkelaars in staat om zowel server-side als client-side webapplicaties te bouwen, waarbij de frontend volledig in C# kan worden geschreven en naadloos kan communiceren met backendlogica.

Belangrijke aspecten van Blazor

- Server-side Blazor: de applicatielogica draait op de server en de UI wordt via SignalR realtime naar de client gestuurd. Dit is ideaal voor scenario's waarin beveiliging, complexe logica en intensieve dataverwerking centraal staan.
- Blazor WebAssembly (Client-side): de applicatie draait volledig in de browser van de gebruiker, waardoor interacties lokaal worden verwerkt. Dit ontlast de server en is geschikt voor minder complexe onderdelen of interactieve UI-componenten die weinig serverinteractie vereisen.
- Component-gebaseerd: Blazor werkt modulair met herbruikbare componenten, waardoor de frontend onderhoudbaar, schaalbaar en overzichtelijk blijft.
- Databinding en event-handling: Blazor biedt eenvoudige integratie tussen frontend Ui en backendlogica via data- en event-binding. Dit maakt real-time updates en interactieve interfaces eenvoudiger te implementeren.

Toepassing in het project

Het case-systeem is een relatief complexe frontend-module binnen het project, met veel logica en methodes. Hiervoor is gekozen voor server-side Blazor, zodat verwerking op de server gebeurt en de prestaties en efficiëntie gewaarborgd blijven. De component gebaseerde structuur van Blazor maakt het mogelijk dat de frontend van het volledige project in de toekomst uit meerdere frontend-solutions kan bestaan. Zo kunnen nieuwe modules of mindere complexe onderdelen modulair worden toegevoegd, met behoud van overzichtelijkheid en schaalbaarheid. Lichtere frontend-modules zouden eventueel client-side Blazor kunnen gebruiken om serverbelasting te verminderen en resources efficiënter in te zetten.

Conclusie

Blazor biedt binnen dit project een stabiele en flexibele basis voor de frontendontwikkeling. De component-gebaseerde aanpak, databinding en integratie met .NET zorgden voor een overzichtelijke, modulair opgebouwde frontend die schaalbaar en goed beheersbaar is. Zo kan de frontend in de toekomst eenvoudig worden uitgebreid met nieuwe modules of frontend-solutions, terwijl performance en onder houdbaarheid behouden blijven.

8.3. SQL / Database

SQL (Structured Query Language) is de standaardtaal voor het beheer van relationele databases. Binnen het project ondersteunt SQL het efficiënt opslaan, beheren en ophalen van gegevens en vormt het de ruggengraat van de applicatie.

Belangrijke functies van SQL

- Dataopslag en -beheer via tabellen, relaties en constraints.
- Dataquery's voor ophalen, invoegen, bijwerken en verwijderen van gegevens.
- Transacties voor veilige, consistente bewerkingen.
- Performance-optimalisatie via indexen en geoptimaliseerde query's.
- Beveiliging en rechtenbeheer om toegang tot data te controleren.

Toepassing in het project

De backend communiceert via SQL-query's met de database om data te manipuleren en op te halen voor de API en de frontend. Complexe query's en stored procedures werden gebruikt om data efficiënt en veilig te verwerken.

Conclusie

SQL is een fundamentele technologie die zorgt voor een stabiele, veilige en betrouwbare dataverwerking wat direct bijdraagt aan de prestaties van de backend en alle afhankelijke componenten.

8.4. Tailwind CSS

Tailwind CSS is een utility-first CSS-framework dat snelle en consistente styling van webapplicaties mogelijk maakt. In tegenstelling tot traditionele CSS-frameworks biedt Tailwind kleine, herbruikbare classes waarmee lay-outs en componenten direct gestyled kunnen worden.

Belangrijke kenmerken

- Utility-first: volledige controle over styling met kleine, herbruikbare CSS-classes die directe controle over styling bieden. Dit maakt het eenvoudig om specifieke elementen te stijlen zonder uitgebreide custom CSS te schrijven.
- Responsiveness: Tailwind biedt uitgebreide ondersteuning voor mobiele en desktopweergave via responsieve utilities, waardoor lay-outs automatisch aangepast kunnen worden aan verschillende schermformaten.
- Consistentie en hergebruik: de utility-classes bevorderen consistente styling in de hele applicatie en verminderen herhaling van code. Componenten kunnen snel hergebruikt worden, wat onderhoud en uitbreidingen vereenvoudigt.
- Snelle ontwikkeling: dankzij kant-en-klare utilities kunnen ontwikkelaars snel UI-componenten opzetten zonder veel tijd kwijt te zijn aan CSS-structuur of class-namen.
- Aanpasbaarheid: Tailwind is volledig configureerbaar via het configuratiebestand (tailwind.config.js) waardoor kleuren, spacing, breakpoints en andere designregels op projectniveau kunnen worden gestandaardiseerd.

Toepassing in het project

Tailwind werd gebruikt voor het ontwerpen van Blazor-componenten en het creëren van consistente layouts en visuele elementen. Het versnelde de frontendontwikkeling door snelle styling zonder complexe CSS-bestanden.

Conclusie

Tailwind maakte de frontendontwikkeling sneller en consistenter, waardoor visueel aantrekkelijke en uniforme interfaces konden worden gebouwd.

8.5. MudBlazor

Mudblazor is een component library voor Blazor dat kant-en-klare UI-componenten biedt, zoals knoppen, formulieren, tabellen en modals.

Belangrijke kenmerken

- Component-gebaseerd: eenvoudig te integreren in Blazor-projecten.
- Stijlconsistentie: kant-en-klare thema's en componenten zorgen voor uniforme UI.
- Functionaliteit: interactieve componenten zoals datatables, dialogs en snackbars (toasts).

Toepassing in het project

Mudblazor werkt gebruikt als een alternatieve manier voor CSS en het bouwen van DataGrid. Dit is gedaan zodat het development team het verschil van Tailwind vs Mudblazor goed kan zien en kan aangeven welke ze verkiezen.

Conclusie

Mudblazor bespaart ontwikkeltijd en zorgt voor een consistente, interactieve en professionele gebruikersinterface.

8.6. JavaScript

Hoewel Blazor de primaire frontend-logica beheert, werd JavaScript gebruikt voor specifieke kleine functionaliteiten die niet meteen met Blazor konden worden uitgevoerd, zoals het aansturen van toasts en andere interactieve elementen.

Toepassing in het project

- Aanroepen van toast-meldingen met SwalToast.
- Copy methoden maken bijv. voor klantnummer.

Conclusie

JavaScript vulde de functionaliteit aan waar Blazor beperkt was en ondersteunde zo een vloeiende en interactieve gebruikerservaring.

9. Eindresultaat

In dit hoofdstuk wordt het eindresultaat van de ontwikkeling van het nieuwe case systeem gepresenteerd. Dit systeem vormt één van de eerste moderne bouwblokken binnen de bredere vernieuwingstrajecten van Elegant en markeert een belangrijke stap in het geleidelijk afbouwen en uiteindelijk vervangen van het verouderde ES3-platform. Hoewel het project in deze fase nog niet volledig is afgerond, is een functioneel, toekomstbestendig en uitbreidbaar fundament gerealiseerd dat de basis legt voor verdere digitalisering en modernisering van de interne processen.

Het nieuw ontwikkelde case-systeem combineert een moderne frontendarchitectuur, een sterk beveiligde backendstructuur op basis van het Backend-for-Frontend-patroon, geïntegreerde Microsoft-authenticatie en een gebruiksvriendelijk interfaceontwerp. De toepassing centraliseert klantcases, verhoogt de transparantie van interne processen en minimaliseert fouten die in het verleden door ES3 werden veroorzaakt door verouderde technologie, gebrek aan flexibiliteit en beperkte uitbreidbaarheid.

Het resultaat is niet enkel een replica van de bestaande ES3-casefunctionaliteit, maar een herontwerp dat uitgaat van hedendaagse softwareprincipes, consistente datamodellen en moderne user experience-richtlijnen. Daarbij werd rekening gehouden met de toekomstige groei van het CRM-systeem, de integratie van andere modules en een mogelijke volledige migratie van legacy-functies naar nieuw ontwikkelde services.

9.1 Doelstellingen van het eindresultaat

Bij de start van het project werden drie centrale doelstellingen vastgelegd, die als maatstaf dienen voor de evaluatie van het eindresultaat van het case systeem. Deze doelstellingen zijn zorgvuldig gekozen op basis van een analyse van het oude systeem (ES3), de wensen van het customer-support team en de toekomstige ambitie van Elegant om een schaalbaar en modulair CRM-platform te ontwikkelen. De drie doelstellingen zijn:

Moderniseren van de Case functionaliteit uit ES3.

Verbeteren van de gebruikerservaring en workflow van medewerkers.

Garanderen van een schaalbare, veilige en onderhoudbare architectuur.

Elke doelstelling wordt hieronder uitgebreid besproken, inclusief de achtergrond, motivatie, implementatieaanpak en de verwachte impact op de organisatie.

Moderniseren van de casefunctionaliteit uit ES3

Het oude case systeem ES3 was technisch en functioneel sterk verouderd. De technologie bood weinig flexibiliteit en onderhoudsbaarheid, de datastromen waren onduidelijk gedocumenteerd. Hierdoor ontstonden regelmatig fouten en vertragingen bij het verwerken van cases en was het systeem moeilijk aan te passen aan veranderde behoeften.

De modernisering van de casefunctionaliteiten had daarom verschillende doelen:

Efficiëntie verbeteren: door de backend te migreren naar .NET 8 en de frontend naar Blazor Server, konden verwerkingstijden worden verkort en complexe logica op de server worden afgehandeld. Hierdoor zijn gegevens sneller beschikbaar voor medewerkers en wordt de kans op fouten verminderd.

Onderhoudsbaarheid vergroten: de applicatie is modulair opgebouwd met herbruikbare componenten en duidelijke scheiding tussen frontend, BFF-laag en backend-services. Dit maakt toekomstige aanpassingen eenvoudiger en veiliger, zonder risico op regressies.

Foutreductie: door centrale validatie van data via de BFF en geautomatiseerde workflow, worden inconsistenties en menselijke fouten beperkt. De audit log biedt een transparante en traceerbare geschiedenis van acties, waardoor problemen sneller opgespoord kunnen worden.

Modulariteit: door het systeem op te delen in afzonderlijke modules (casebeheer, case types; attachments, marktberichten), kunnen nieuwe functionaliteiten of integraties in de toekomst eenvoudig worden toegevoegd zonder het bestaande systeem te verstoren.

Deze modernisering stelt Elegant in staat om bestaande processen niet alleen te digitaliseren, maar ook structureel te verbeteren. Zo kunnen medewerkers cases sneller en accurater verwerken, terwijl het systeem klaar is voor toekomstige uitbreidingen.

Verbeteren van de gebruikerservaring en workflow van medewerkers

Naast technische modernisering was het verbeteren van de gebruiksvriendelijkheid een centrale doelstelling. Uit gesprekken met het customer support-team bleek dat het oude systeem te complex en tijdrovend was, met meerdere schermen om basisinformatie in te zien.

Het nieuwe case-systeem is daarom ontworpen met gebruikersgerichte aanpak:

Overzichtelijke interface: de case overzicht pagina biedt een direct inzicht in lopende en afgeronde cases, met filter-en zoekfunctionaliteit, zodat medewerkers sneller prioriteiten kunnen bepalen.

Centraal informatiepunt: de detailpagina combineert klantgegevens, case details, attachments, audit log en marktberichten op één scherm. Medewerkers hoeven niet tussen schermen te navigeren, wat de kans op fouten vermindert en de verwerkingstijd verkort.

Automatisering van repetitieve taken: marktberichten worden automatisch omgezet in cases, waardoor handmatige acties overbodig worden en meldingen sneller worden opgevolgd.

Feedback en interactieve verbetering: low- en highfidelity mockups in Figma maakten het mogelijk om continu feedback van medewerkers te integreren, waardoor het ontwerp zowel functioneel als intuïtief werd.

Door de workflow van medewerkers centraal te stellen wordt niet alleen de efficiëntie verhoogd, maar ook de tevredenheid en adoptie van het systeem binnen Elegant.

Garanderen van een schaalbare, veilige en onderhoudbare architectuur.

De derde doelstelling richt zich op de architectuur en toekomstbestendigheid van het case systeem.

Elegant heeft de ambitie om het case systeem onderdeel te maken van een groter CRM-platform. Dit vereist een architectuur die zowel schaalbaar als veilig is en eenvoudig kan meegroeien met nieuwe modules en gebruikersaantallen.

Schaalbaarheid: door gebruik van .NET 8 en een BFF-laag kan de backend meerdere frontend-clients bedienen, terwijl de dataverwerking centraal en efficiënt blijft. Nieuwe modules kunnen eenvoudig worden toegevoegd zonder impact op bestaande functionaliteiten.

Beveiliging: authenticatie via Microsoft Accounts, gecombineerd met een fijnmazig rollen – en groepsmodel, zorgt dat gevoelige data alleen toegankelijk is voor bevoegde medewerkers. Alle API-aanvragen worden gecontroleerd door de backend, wat ongeautoriseerde toegang voorkomt.

Onder houdbaarheid en documentatie: door modulaire opbouw, gebruik van component-bases Blazor-Frontend en gestandaardiseerde datamodelen via BFF, kan toekomstige ontwikkeling gestructureerd plaatsvinden. Documentatie en code worden ondersteund door tools zoals GitHub Copilot, ChatGPT en SSMS, wat het onderhoud en de uitbreiding vereenvoudigt.

Monitoring en audit: de audit log en logging van backend-processen bieden continu controle en inzicht in systeemgebruik, waardoor snel kan worden ingegrepen bij fouten of verdachte activiteiten.

Deze doelstelling waarborgt dat het case systeem niet alleen voldoet aan de huidige eisen, maar ook robuust en toekomstbestendig is, klaar om op termijn een kerncomponent van het bredere Elegant CRM-platform te vormen.

Demo

De demo van het case systeem biedt een uitgebreide en praktische illustratie van de belangrijkste functionaliteiten en gebruikersstromen binnen het systeem. Deze demonstratie is zorgvuldig samengesteld om zowel technische als niet-technische stakeholders een duidelijk begrip te geven van de mogelijkheden en voordelen van het systeem, met bijzondere aandacht voor de gebruikerservaring

Een volledig functionele demonstratie van het case-systeem is beschikbaar via de volgende link:

Link naar filmpje demo

Conclusie

De drie doelstellingen - modernisering van functionaliteiten, verbetering van gebruiksvriendelijkheid en waarborgen van schaalbare en veilige architectuur – zijn nauw met elkaar verbonden. Het nieuwe case-systeem van Elegant:

- Verbetert de efficiëntie en betrouwbaarheid van dagelijkse processen.
- Verhoogt de tevredenheid en productiviteit van medewerkers.
- Legt een solide basis voor toekomstige integraties binnen het bredere CRM-Platform.

Door deze doelstellingen systematisch te vertalen naar technische en functionele keuzes, voldoet het project niet alleen aan de huidige behoeften, maar creëert het een toekomstbestendig systeem dat kan meegroeien met de organisatie.

CONTACT

Kenzo Serneels | Student
R0889622 @Student. thomasmore.be
Tel. + 32 493 14 52 35

VOLG ONS

www.thomasmore.be
fb.com/ThomasMoreBE
#WeAreMore

THOMAS MORE