AI-agents bouwen op Power Platform

Praktijkgids voor low-code agents waar klanten en medewerkers echt iets aan hebben

Door Albert-Jan Schot, CTO Blis Digital

Waar we het over gaan hebben:

Waarom ik dit voor jou geschreven heb:

Ik ben Albert-Jan Schot, CTO bij Blis Digital. Mijn collega's noemen me Appie. Ik houd me bezig met drie dingen: R&D om nieuwe technologie te verkennen, presales om klanten te helpen met complexe vraagstukken en gewoon lekker met mijn voeten in de klei software maken. AI heeft alle drie die rollen fundamenteel veranderd.

De afgelopen tijd hebben we bij Blis Digital veel geschreven over AI-agents: wat ze kunnen, waarom ze waardevol zijn, welke impact ze hebben op organisaties en hoe je ermee werkt in de praktijk. Maar één vraag bleef telkens terugkomen:

Oké, mooi verhaal. Maar hóé bouw je ze dan echt?

Dit boekje is mijn antwoord daarop. Niet de waarom of de wat, maar de hoe. Hoe je AI-agents bouwt die niet alleen slim zijn, maar ook doen wat ze moeten doen. Agents die morgen al in productie kunnen draaien.

Ik schrijf dit voor jou als je een maker bent. Als je houdt van techniek die echt werkt. Als je niet denkt in abstracties, maar in werkende oplossingen. Het maakt me niet uit of er 'CTO', 'dev lead', 'product owner', 'developer' of iets totaal anders op je LinkedIn staat. Het gaat erom dat je iemand bent die wil dat het werkt en dat gebruikers er snel mee kunnen werken.

In dit boek laat ik je zien hoe je AI-agents bouwt op Microsoft Power Platform. Waarom Power Platform? Omdat het snel naar productie gaat, omdat het naadloos integreert met Microsoft AI en omdat je direct kunt werken met de data die al in je organisatie zit. Geen maanden ontwikkeltijd, maar morgen al een werkende agent.

Je leest over architectuurprincipes, over het ontwerpen van goede conversaties, over integratie met je bedrijfssystemen en over testen op wat er fout kan gaan. En vooral: je krijgt praktische voorbeelden die je direct kunt toepassen.

Geen theorie om de theorie. Gewoon: dit werkt, zo doe je het, begin vandaag.

Dus laten we gauw beginnen.

AI-agents bouwen in low-code: dit is hoe je het doet

Laten we bij het begin beginnen. In dit hoofdstuk hebben we het over keuzes maken. En hoe die keuzes bepalen of je straks met een werkende agent eindigt of met een demo die nooit productie haalt.

Van ontwerp naar realisatie

In het whitepaper "Ontwerpen voor het AI-tijdperk" beschreven mijn collega's Kim en Akhil hoe je klassieke ontwerpprincipes vertaalt naar een AI-gedreven wereld. Dat paper laat zien hoe interacties veranderen, wat een agent nodig heeft om goed te functioneren en hoe je ontwerpt voor een systeem dat niet alleen antwoorden geeft, maar ook acties uitvoert. De kern daarvan draait om drie principes:

1. **Begin altijd met waarom:** welk probleem los je op?

2. **Beschouw AI-agents als stakeholders:** wat hebben ze nodig om goed te functioneren?

3. **Denk verder dan de chatbot:** AI kan ook onzichtbaar op de achtergrond werken

Die principes zijn het fundament. Maar na ontwerp komt realisatie. En daar wordt het concreet. Want je kunt het mooiste ontwerp hebben, maar als je het niet snel genoeg naar productie krijgt, heeft niemand er wat aan.

Waarom Power Platform?

Er zijn veel manieren om AI-toepassingen te bouwen. Je kunt full-code gaan met Python en frameworks, je kunt experimenteren met tools die je helpen snel prototypes te maken, maar bij Blis kiezen we vaak voor Microsoft Power Platform. Niet omdat het de enige optie is, maar omdat het een aantal dingen buitengewoon goed doet:

- ↗ **Snelheid naar productie:** Met Power Platform bouw je niet alleen snel een prototype, je bouwt meteen iets dat productiewaardig is. Geen maanden ontwikkeltijd, geen eindeloze discussies over infrastructuur. Je begint vandaag, test morgen en draait volgende week in productie. Die snelheid is geen luxe, het is een noodzaak als je wilt leren van echte gebruikers, niet van theoretische scenario's.

- **Integratie met Microsoft AI-functies:** Power Platform zit vol met ingebouwde connecties naar Azure OpenAI, AI Builder en andere Microsoft AI-services. Je hoeft niet zelf de integratie te bouwen, die zit er al in. Dat betekent minder technische complexiteit en meer focus op wat de agent moet kunnen in plaats van hoe je hem technisch aan de praat krijgt.

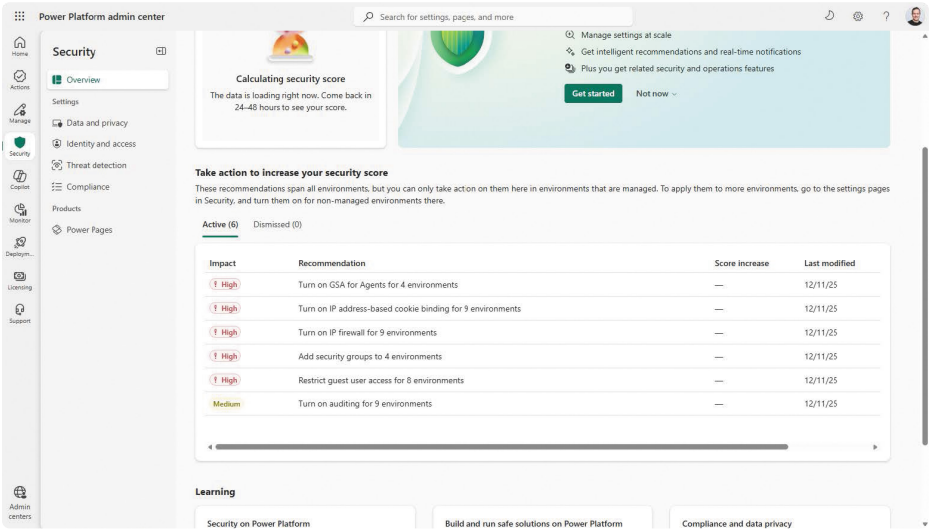
- **Werken met je bedrijfsdata:** Dit is misschien wel het allerbelangrijkste: Power Platform is gebouwd om te werken met de data die al in je organisatie zit. Dataverse, SharePoint, Dynamics, Teams - alles is al verbonden. Een AI-agent is zo slim als de data die hij krijgt en als die data al beschikbaar en gestructureerd is, win je enorm veel tijd.

Governance is geen zorg voor later

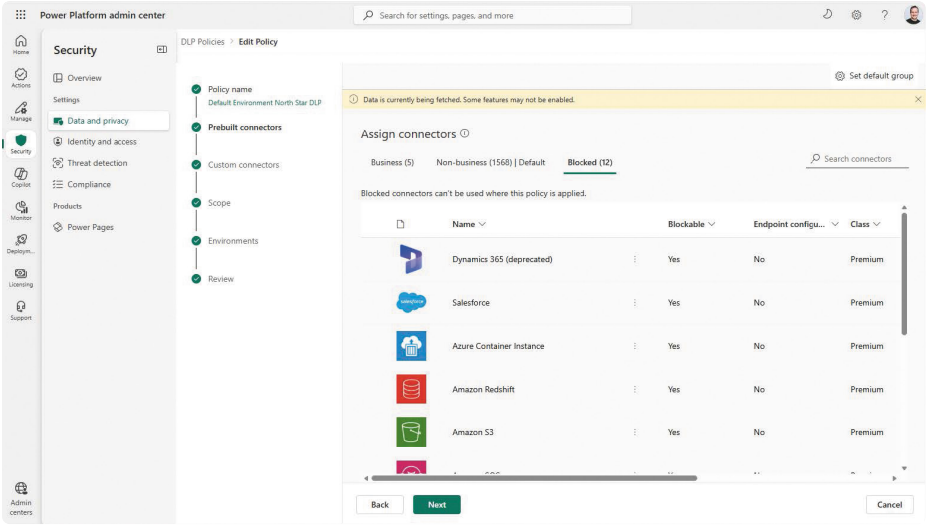
Hier komt het punt waar veel organisaties de fout ingaan: ze beginnen met experimenteren zonder na te denken over governance. Governance klinkt saai, maar het is essentieel vanaf dag één. Governance betekent: wie mag wat, hoe hou je controle en hoe voorkom je dat je agent straks dingen doet die niet mogen?

Denk dus vanaf het begin na over dataveiligheid: welke data mag je agent inzien? Mag hij persoonlijke informatie verwerken? Wie controleert wat er gebeurt met gevoelige gegevens? En denk aan toegangsrechten: welke gebruikers mogen de agent gebruiken? Welke acties mag de agent uitvoeren namens een gebruiker?

In Power Platform regel je dit via Data Loss Prevention (DLP) policies, security roles en environment-instellingen. Dat klinkt technisch, maar het principe is simpel: bepaal vooraf wat wel en niet mag en tot welke data en systemen je agent toegang heeft. Bouw je agent op basis van deze afbakening. Anders krijg je straks een agent die fantastisch werkt, maar die je niet live mag zetten omdat hij privacyregels schendt of toegang heeft tot systemen waar hij niet hoort te komen.



↳ Het Power Platform Admin center biedt over verschillende assen inzicht in Data & Privacy en andere security controls.



↳ Overzicht van configuratie van een van de DLP policies waarin je duidelijk ziet dat een aantal (12) stuks zijn geblokeerd, zodat ze niet gebruikt kunnen worden.

Begin klein, schaal onder governance

Start je AI-avontuur met één concreet scenario. Geen mega-agent die alles kan, maar een oplossing voor één specifiek probleem. Een HR-agent voor verlofaanvragen. Een IT-agent voor wachtwoord resets. Een finance-agent die rapporten ophaalt. Klein, overzichtelijk, meetbaar.

Bouw die agent binnen een environment met duidelijke regels. Test hem grondig. Meet of hij doet wat hij moet doen. En als het werkt? Dan schaal je op. Voeg een tweede scenario toe. Breid uit naar andere afdelingen. Maar altijd binnen de kaders van je governance-model. Zo bouw je niet alleen snel, maar ook verantwoord. En dat is precies waar het om gaat: technologie die werkt én veilig is.

In de volgende hoofdstukken duiken we dieper in de architectuur, het ontwerp van gesprekken, integraties en praktische voorbeelden. Maar onthoud dit: voordat je begint met bouwen, zorg dat je weet waarom je bouwt, hoe je het veilig houdt en dat je klein begint.

“Start je AI-avontuur met één concreet scenario”

Architectuur principes voor AI-agents op Power Platform

Een chatbot is niet meteen een AI-agent. Dat is het eerste wat je moet begrijpen. Een chatbot reageert op wat je vraagt met informatie. Een AI-agent voert autonoom taken uit om een doel te bereiken. Het verschil? Jij stelt het doel, de agent kiest zelf de aanpak. Vertel een chatbot dat je verlof aan wilt vragen en hij zegt: "Ga naar het HR-portaal." Vraag het een agent en hij regelt het meteen, inclusief goedkeuringsflow en notificatie naar je manager.

Laten we even wat uitgebreider stilstaan bij de verschillende soorten AI-oplossingen, want ze hebben allemaal hun eigen sterke en zwakke punten. Dus is het belangrijk dat je de juiste keuze maakt. Bij Blis Digital onderscheiden we drie hoofdcategorieën van AI-oplossingen:

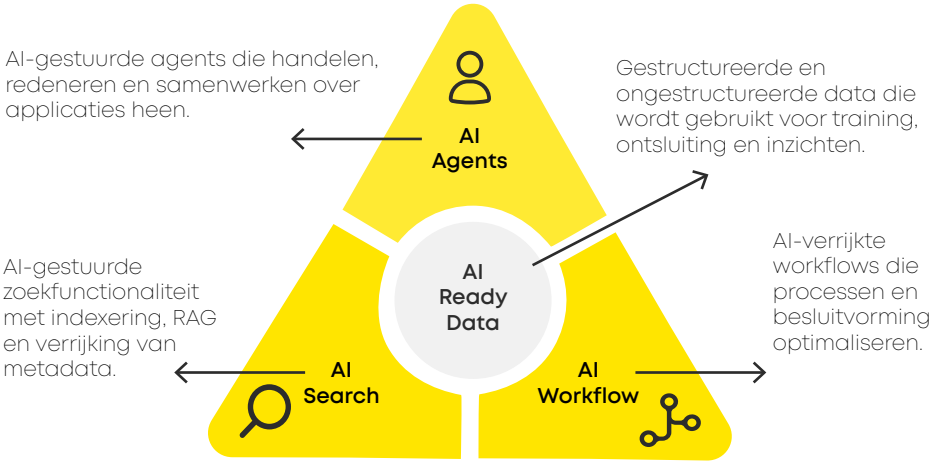
- **AI agents:** AI agents zijn autonome softwareprogramma's. Ze kunnen redeneren, handelingen uitvoeren en interacteren met data en systemen. Jij geeft de agent een doel ("regel dit voor me") en hij gaat zelf bedenken hoe hij daar komt. Hij probeert verschillende wegen uit tot het gelukt is. Dat maakt AI agents krachtig, maar soms ook onvoorspelbaar.

- **Agentic workflows:** Zoek je meer controle? Dan heb je een agentic workflow nodig. Hierbij bepaal jij exact welke stappen er gezet worden, in welke volgorde en wat er moet gebeuren als iets misgaat. AI speelt een rol als één van de radertjes, maar altijd binnen de grenzen die jij stelt. Sommige stappen zijn normale automatiseringen (code die altijd hetzelfde doet), andere stappen besteed je uit aan een AI agent.

- **AI search en dataverrijking:** De derde categorie krijgt vaak minder aandacht, maar is minstens zo belangrijk: AI search en dataverrijking. Zowel je agents als je workflows zijn nergens zonder goede data. AI search zorgt ervoor dat die data vindbaar, begrijpelijk en bruikbaar is. Het is de onmisbare laag tussen je ruwe bedrijfsdata en de AI-oplossingen die je erop bouwt. Met AI kun je semantisch zoeken. Je zoekt dan niet meer op zoekwoorden, maar op betekenis. AI kan data ook verrijken door tags en metadata toe te voegen aan documenten, zodat je dwarsverbanden kunt leggen die je eerder niet zag.

Bij het maken van een keuze is het principe altijd: begin bij het probleem, niet bij de oplossing. Als je je businessprobleem begrijpt, kun je daarna een (AI-) aanpak kiezen die daarbij past.

Maar welke technologie je ook kiest, het succes van je oplossing staat of valt altijd met de kwaliteit van de data. Want als de input of de context niet kloppen, maakt zelfs de best opgezette agent, workflow of zoekmachine cruciale fouten.



↳ Drie typen AI-oplossingen, met AI ready data als basis.

Architectuur op orde: 7 kernprincipes

Het autonome karakter van agents maakt ze krachtig, maar ook complex. Want als een agent zelf beslissingen neemt, moet je architectuur op orde zijn. Geen shortcuts, geen “dat regelen we later wel”. Bij agents is techniek geen bijzaak, het is de basis van betrouwbaarheid en vertrouwen.

De keuzes die je maakt rond omgeving, security en onderhoud hebben directe impact op bruikbaarheid, schaalbaarheid en vertrouwen. Hieronder de kernprincipes die je helpen bij het maken van de juiste keuzes.

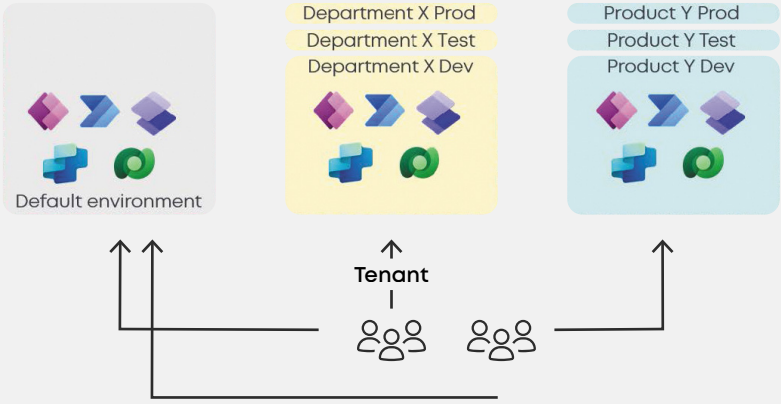
1. Begin met een Solution

Dit klinkt als een open deur, maar ik zie het toch te vaak fout gaan: mensen bouwen agents los van hun flows, connectoren en data. Dat werkt niet. Gebruik altijd een Power Platform Solution om alles bij elkaar te houden. Een Solution bundelt je agent, flows, connectoren, tabellen en configuratie in één pakket.

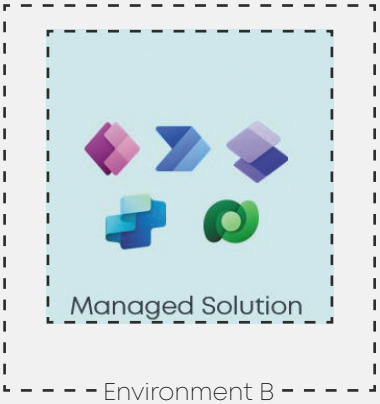
Waarom is dat belangrijk?

- **Overdraagbaarheid:** Je bouwt in development, test in acceptatie, rolt uit naar productie. Met een Solution exporteer je alles in één keer. Geen losse onderdelen die je handmatig moet nabouwen.
- **Versiebeheer:** Je wilt weten welke versie van je agent waar draait. Een Solution maakt dat inzichtelijk. Je publiceert een nieuwe versie, test die en als het misgaat rol je terug naar de vorige.
- **CI/CD-pipelines:** Als je serieus agents wilt bouwen, automatiseer je deployment. Met Solutions kun je via pipelines automatisch uitrollen naar verschillende environments. Dat scheelt tijd en voorkomt fouten.

Als je al met Power Platform werkt, doe je dit waarschijnlijk al. Maar zo niet: dit is stap één.



↳ Overzicht van mogelijke omgevingsindelingen. Naast de standaardomgeving kunnen per afdeling of per product meerdere omgevingen worden ingericht, zoals development, test en productie. Zo kun je agents veilig ontwikkelen en testen zonder het risico dat een wijziging direct impact heeft op de productieomgeving.



→ In Power Platform werk je met Solutions. Dit is een soort 'map' waarin je alles rondom een agent bundelt, zoals de agent zelf, gekoppelde tools en data, en verbindingen met andere systemen. Je rolt een Solution uit naar een omgeving om test en productie te scheiden, zodat je aanpassingen kunt doorvoeren zonder dat gebruikers deze direct zien.

2. Eén scenario, één agent

Een agent kan prima een chat-interface hebben. Sterker nog, dat is vaak de meest natuurlijke manier om met een agent te communiceren. Maar het verschil met een chatbot zit hem in wat er achter die chat gebeurt. Een chatbot geeft informatie, een agent voert acties uit. Hij roept flows aan, past data aan, stuurt notificaties, maakt tickets aan. Dat is waarom je per scenario een aparte agent bouwt.

Ik zie vaak de neiging om meerdere functies in te bouwen in een soort 'super-agent'. HR, IT, finance, facilities - allemaal in één gesprek. Dat lijkt efficiënt, maar is het niet. Zo'n mega-agent wordt onbeheersbaar. De prompts worden te complex, de flows te vertakt en de gebruiker raakt verdwaald in een gesprek dat alle kanten op schiet.

Maak in plaats daarvan per scenario een aparte agent. Een HR-agent voor verlof en verzuim. Een IT-agent voor wachtwoorden en toegang. Een finance-agent voor rapporten en declaraties. Klein, overzichtelijk, met een duidelijk doel. Als scenario's elkaar overlappen, laat je agents samenwerken via orkestratie. Daar komen we later op terug.

3. Gebruik aparte Environments

Ontwikkeling, test en productie horen gescheiden te zijn. Altijd. Je wilt niet dat een test-agent per ongeluk echte data aanpast of emails verstuurt. In Power Platform regel je dat met Environments: aparte omgevingen met eigen data, eigen security en eigen configuratie.

> **Mijn advies:** gebruik minimaal drie environments. Development voor bouwen en experimenteren. Test voor validatie en user acceptance testing. Productie voor live gebruik.

Beperk per environment de toegang op basis van rollen. Developers mogen alles in development, maar niet in productie. Eindgebruikers zien alleen productie.

En stel per environment de juiste Data Loss Prevention (DLP) policies in. Die policies bepalen welke connectoren samen mogen werken. Bijvoorbeeld: je wilt wel dat je agent data uit Dataverse haalt, maar niet dat hij die zomaar naar een externe API stuurt. DLP voorkomt dat.

4. Snap de impact van DLP policies

DLP policies zijn geen formaliteit. Ze bepalen letterlijk wat je agent kan en niet kan. Een agent die data uit Dataverse ophaalt en ook Outlook gebruikt, kan geblokkeerd worden als die connectoren niet in dezelfde policy-groep zitten. En een agent die via een Custom Connector met een externe API praat, vereist expliciete toestemming.

Inventariseer dus vooraf welke connectoren je nodig hebt. Overleg met je Power Platform admin en je CISO (of zet zelf die pet op). En test je agent in alle environments, want DLP policies kunnen per environment verschillen.

5. Kies je connectoren bewust

Copilot Studio werkt met standaardconnectoren. Die zitten ingebouwd en vragen geen extra licentie. Vaak doen deze connectoren wat je wilt, maar soms heb je meer nodig, bijvoorbeeld connectors voor legacy-systemen of externe API's. Er zijn ook premium connectoren die geavanceerde functionaliteit bieden. Kies je ervoor om connectoren te bouwen of te kopen, dan kost dat geld en vraagt beheer.

> **Mijn advies:** gebruik alleen wat je echt nodig hebt. Less is more. Elke connector is een afhankelijkheid, een potentieel risicopunt en iets dat onderhouden moet worden. Houd het simpel.

6. Documenteer alles

Dit klinkt saai, maar het is cruciaal. Zorg dat je bij elke agent weet welke connectoren en acties worden gebruikt, welke data wordt gelezen of geschreven en welke fallback- of escalatiepaden er zijn. Waarom? Omdat je over drie maanden vergeten bent hoe het werkt. En omdat iemand anders het misschien moet overnemen.

Documentatie hoeft niet fancy te zijn. Een README in je Solution met architectuurschets, dependencies en contactpersonen is al genoeg. Dat kan overigens alleen als je source code integration hebt ingeschakeld, dus dat doen wij standaard. Het gaat erom dat iemand die de agent niet gebouwd heeft, toch begrijpt wat hij doet en hoe. Natuurlijk gebruik je hier ook AI voor.

7. Beheer agents als software

Agents zijn software. Behandel ze ook zo:

- **Gebruik Application Lifecycle Management (ALM):** sla wijzigingen op in een source control, gebruik pipelines voor het uitrollen, plan updates en rollbackscenario's. Dit is geen overkill, dit is professioneel werken.
- **Security by design:** Authenticatie en autorisatie zijn geen extraatje. Wie mag de agent starten? Wat mag de agent doen namens een gebruiker? Gebruik Azure Key Vault voor secrets, beperk toegang via rollen en log alles wat de agent doet.

Een goed ontworpen agent is niet alleen slim aan de voorkant, maar robuust, veilig en beheersbaar onder de motorkap. Zie het bouwen van agents niet als een losstaand experiment, maar als integraal onderdeel van je platformstrategie. Want dat is het.

“Gebruik
alleen
wat je echt
nodig hebt.

Less
is more”

Stappenplan: Van idee naar agent

Je hebt een probleem geïdentificeerd. Je weet waarom je een agent wilt bouwen. Je hebt je architectuur op orde. Nu wordt het concreet: hoe bouw je dat agent daadwerkelijk? Niet als theoretisch concept, maar als werkende oplossing die morgen al draait.

De waarheid is: er is geen universeel recept. Elke usecase is anders, elke organisatie heeft eigen systemen en elke gebruikersgroep heeft eigen verwachtingen. Maar er is wel een structuur die werkt. Zeven stappen die je van idee naar productie brengen. En het mooie? Bij stap drie heb je al iets dat je kunt laten zien.

Stap 1: Kies een specifieke usecase

Hoe specifieker het probleem dat je oplost met je agent, hoe beter. Begin dus niet met “een HR-assistent” of iets dergelijks, begin met “een agent die verlofaanvragen afhandelt”. Specifiek, meetbaar, met duidelijke in- en output. Een medewerker wil verlof aanvragen. De agent checkt het tegoed, vraagt de juiste gegevens op, stuurt een goedkeuringsverzoek naar de manager en bevestigt zodra het geregeld is.

Waarom zo specifiek? Omdat je wilt leren. Je wilt zien of je agent werkt, of gebruikers het snappen en of het waarde toevoegt. Dat lukt het beste met een simpel scenario dat je in een week live kunt zetten. En ja, straks bouw je meer agents. Voor ziekmeldingen, voor onboarding, voor salarisadministratie. Maar niet allemaal tegelijk. Eén scenario, bewijzen dat het werkt, dan pas verder

Stap 2: Ontwerp de conversatieflow

Hier komt het verschil tussen developer en designer naar voren. Als developer denk ik in data en logica: welke velden heb ik nodig, welke validaties moet ik doen, welke API’s moet ik aanroepen? Maar een agent begint niet met code, het begint met taal.

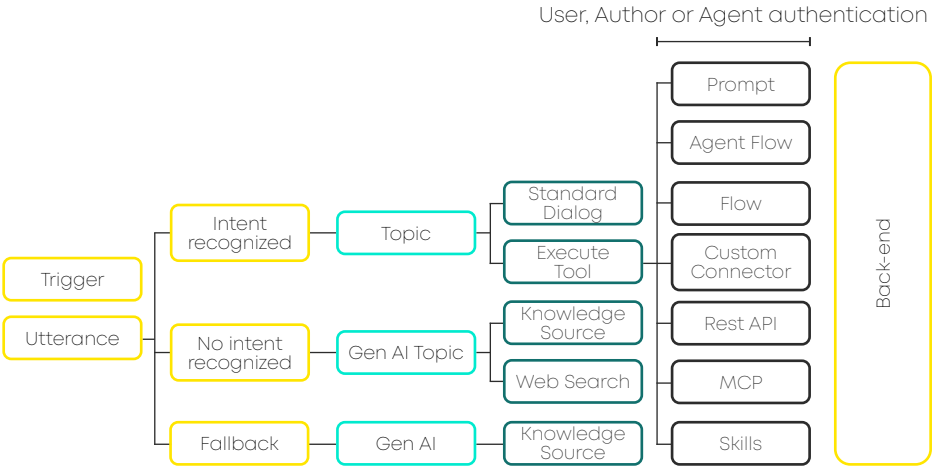
Hoe start het gesprek? “Ik wil verlof aanvragen” is een goede conversation starter. Maar wat komt daarna? Vraag je meteen om data? Of geef je eerst context? “Prima, ik help je daarmee. Hoeveel dagen wil je opnemen en wanneer?” Klinkt natuurlijker dan een formulier met verplichte velden.

En dan de **edge cases**. Wat als iemand zegt: “Ik wil vrij”? Of “Volgende week maandag tot woensdag”? Of gewoon “Vakantie”? Je prompt moet robuust genoeg zijn om variaties te herkennen, maar ook slim genoeg om door te vragen als iets onduidelijk is.

Dit is waar je als developer moet leren denken als een schrijver. Niet: “Welke parameter moet ik meegeven?”, maar: “Hoe zou ik dit tegen een collega zeggen?” Het helpt om dit op papier te zetten voordat je Copilot Studio opent. Teken de flow uit, bedenk varianten, test het in je hoofd. Het scheelt je later uren debuggen.

Meestal zet je hiervoor een combinatie in van generatieve AI en ‘ouderwetse’ chatbot-techniek die op basis van de invoer de bedoeling van de gebruiker herkent en dan een standaardflow (‘topic’ in Power Platform) in gang zet.

Herkent je agent de topic niet, dan kan hij geen actie ondernemen maar wel informatie verstrekken op basis van de beschikbare kennis.



In de afbeelding zie je hoe Copilot Studio een vraag van een gebruiker afhandelt. Als er een vraag binnenkomt, kijkt de agent of hij er topic voor heeft. Zo ja (‘Intent recognized’), dan handelt hij de vraag af met een standaardgesprek of met het uitvoeren van een actie in achterliggende systemen of het aanroepen van een andere agent.

Begrijpt de agent niet wat de gebruiker wil, dan probeert hij het met generatieve AI op te lossen. Tijdens die conversatie kan de agent altijd weer teruggaan naar topics die wel bekend zijn. Bijvoorbeeld als de gebruiker het gesprek beëindigt: dan weten veel agents dat ze feedback moeten vragen met de bekende ‘5 smileys’.

Stap 3: Voeg kennis en data toe

Een agent zonder data is een dom script. Kennis en context maken hem slim. Bij verlofaanvragen heb je minimaal drie dingen nodig: hoeveel tegoed heeft deze medewerker, wie is de manager en welke afdelingsregels gelden.

Die data zitten ergens. Misschien in Dataverse, misschien in een HR-systeem, misschien in SharePoint. Het maakt niet uit waar, als je het maar kunt ophalen. In Power Platform gebruik je connectoren om data te ontsluiten. Dataverse is vaak de makkelijkste: gestructureerd, veilig en native geïntegreerd. Maar externe systemen kunnen ook, via REST API's, custom connectors of Model Context Protocol (MCP) server..

Hier komt ook Retrieval Augmented Generation (RAG) in beeld. Stel: je wilt dat het agent verlofregels uitlegt. Die regels staan in een beleidsdocument op SharePoint. Je kunt dat document handmatig in je prompt stoppen, maar dat schaalt niet. RAG haalt automatisch relevante stukken uit documenten op basis van de vraag. "Hoeveel ouderschapsverlof mag ik opnemen?" triggert een zoekopdracht, vindt het juiste hoofdstuk en de agent geeft antwoord op basis van die context.

Stap 4: Koppel acties

Een agent die alleen informatie geeft, is een chatbot. Een agent die acties uitvoert, is veel waardevoller. Bij verlofaanvragen betekent dat: data wegschrijven in Dataverse, een goedkeuringsflow triggeren in Power Automate en notificaties sturen naar Teams.

Hier komen drie soorten acties bij kijken:

- **Agent flows:** De standaard voor procesorkestratie. Je agent triggert een flow, die flow doet het zware werk: goedkeuringen, notificaties, data updates. Clean, beheersbaar, herbruikbaar.

- **Custom connectors:** Voor systemen zonder standaardconnector. Legacy HR-systemen, ticketing tools, ERP-pakketten. Je bouwt een connector, definieert de API-calls en je agent kan ermee praten alsof het native integratie is.

- **RPA via Power Automate Desktop:** Voor systemen zonder API. Ja, dat bestaat nog steeds. Een oud personeelssysteem waar je moet inloggen, door schermen moet klikken en formulieren moet invullen. RPA automatiseert die handelingen. Niet ideaal, maar soms de enige optie.

> Kies altijd de simpelste aanpak die werkt. API's boven RPA, standaardconnectoren boven custom. **Less is more.**

Stap 5: Ontwerp de interactie

Tekst is prima, maar niet altijd het meest efficiënt. Een Adaptive Card met dropdowns en knoppen is sneller en duidelijker. "Welk type verlof?" met keuzes: Vakantie, Ziekte, Ouderschapsverlof, Onbetaald. Eén klik in plaats van typen en hopen dat het agent het snapt.

Adaptive Cards zijn ook visueel sterker. Je kunt samenvatten wat de gebruiker heeft ingevuld, laten bevestigen en pas dan de actie uitvoeren. "Je vraagt 5 dagen vakantie aan van 10 tot 14 juni. Manager: Jan Jansen. Bevestigen?" Met knoppen Ja en Nee. Geen verwarring, geen interpretatiefouten.

We gaan hier later dieper op in, maar onthoud: goede agents combineren conversatie met interface-elementen. Het een sluit het ander niet uit.

Stap 6: Test grondig

De **happy path** van agent is makkelijk te testen. Een agent werkt prima als alles klopt: juiste input, volledige data, geen fouten. Maar de wereld is niet perfect. Gebruikers typen onduidelijk, informatie ontbreekt of systemen zijn traag. Daar moet je op ook testen.

We behandelen dit uitgebreid in een later hoofdstuk, maar hier alvast de basis: test op onverwachte input, test op incomplete data, test op systeemfouten. En bouw **fallbacks** in. Als een agent iets niet snapt, moet hij dat zeggen en een alternatief bieden. "Ik begrijp je vraag niet helemaal. Wil je verlof aanvragen, je tegoed checken of heb je een andere vraag?"

Stap 7: Deployment en monitoring

Je hebt getest, het werkt, dus nu kan het live. Maar deployment is meer dan een knop indrukken: ten eerste rol je uit naar productie, maar moet je

development en test beschikbaar houden voor doorontwikkeling. En dan is er monitoring: je wilt weten welke gesprekken goed lopen, waar je agent afhaakt en welke vragen voor verwarring zorgen.

Ook dit komt later uitgebreid aan bod, maar het principe is simpel: live gaan is niet het eindpunt, het is het begin van leren wat werkt en steeds weer verbeteren.

De stappen zijn eigenlijk geen stappen

Deze zeven 'stappen' zijn geen projectplan en zeker geen waterval. Je doorloopt ze niet lineair en bent dan klaar. Je itereert. Je bouwt een basisversie, test die, leert ervan, verbouwt. Bij stap drie heb je al een werkend prototype. Bij stap vijf kun je gebruikers laten testen. Bij stap zeven draait het live, maar dan ben je pas echt begonnen met leren.

Het mooie van Power Platform is dat itereren snel gaat. Geen weken wachten op deployment, geen complexe release cycles. Je past aan, je test, je rolt de nieuwe versie uit. Die snelheid maakt het verschil tussen een agent die werkt en een agent die echt gebruikt wordt.

Low-code AI-agents en prompt-ontwerp: zo bouw je een goede conversatie

Het bouwen van een goede AI-agent begint niet bij techniek. Het begint bij taal. Letterlijk.

Dat is voor mij als developer soms lastig. Of in ieder geval een andere manier van over software nadenken. Ik ben gewend aan code die precies doet wat ik opschrijf. If-then-else, loops, functies. Voorspelbaar. Maar een agent werkt anders. Je schrijft geen instructies, je voert een gesprek. En dat gesprek moet natuurlijk aanvoelen, ook al is het met een machine.

Dit hoofdstuk gaat over hoe je dat gesprek ontwerpt. Hoe je een agent leert wat hij moet doen, hoe hij zich moet gedragen en hoe hij reageert op wat gebruikers zeggen. En ja, dat vraagt om een andere manier van denken dan je misschien gewend bent.

Systeemprompts: hoe je rol en gedrag van je agent bepaalt

Een systeemprompt is de basis van je agent. Het is de instructie die bepaalt wie of wat de agent is, hoe hij zich gedraagt en wat hij wel en niet mag doen. Denk aan het als de functieomschrijving van een nieuwe collega. Je legt uit wat de verantwoordelijkheden zijn, welke tone of voice je verwacht en waar de grenzen liggen.

In Copilot Studio bouw je deze systeemprompt op in het systeemprompt-veld. Je kunt dat op verschillende manieren doen: platte tekst, markdown of YAML. Welke je kiest hangt af van complexiteit en voorkeur. Laten we kijken naar een concreet voorbeeld voor ons verlof-agent.

Platte tekst is het simpelst. Je schrijft gewoon op wat de agent moet doen:

```
Je bent een HR-assistent die medewerkers helpt met
verlofaanvragen. Je taak is om het verlofaanvraagproces
te begeleiden. Vraag altijd naar het type verlof, de
gewenste periode en controleer of er voldoende tegoed
is. Wees vriendelijk en professioneel. Bij onbekende
situaties verwijs je door naar de HR-afdeling.
```

Dit werkt prima voor eenvoudige agents. Het is direct, leesbaar en je kunt snel aanpassen. Maar bij complexere agents wordt het snel onoverzichtelijk.

Markdown voegt structuur toe. Je gebruikt headers, lijsten en formatting om de prompt georganiseerd te houden:

```
# Rol
Je bent een HR-assistent gespecialiseerd in
verlofaanvragen.

# Verantwoordelijkheden
  - Begeleid medewerkers door het
    verlofaanvraagproces
  - Controleer verloftegoed voordat je een aanvraag
    accepteert
  - Leg verlofregels uit wanneer daarom gevraagd
    wordt
  - Verwijs door naar HR bij complexe vragen

# Gedragsregels
  - Wees vriendelijk en professioneel
  - Gebruik duidelijke, begrijpelijke taal
  - Vraag altijd om bevestiging voordat je een
    aanvraag indient
  - Bescherm de privacy van andere medewerkers

# Beperkingen
Beperk je toegang tot verlofgegevens van de medewerker
zelf. Vraag manager-approval voor uitzonderingen op
standaardregels.
```

Markdown is mijn voorkeur voor de meeste agents. Het is leesbaar, je kunt secties duidelijk scheiden en het blijft overzichtelijk als de prompt groeit.

YAML is het meest gestructureerd. Je definieert de prompt als data:

```
agent:
  role: "HR-assistent voor verlofaanvragen"
  tone: "vriendelijk en professioneel"

responsibilities:
  -    "Begeleid verlofaanvragen van start tot finish"
  -    "Controleer verloftegoed"
  -    "Leg verlofregels uit"
  -    "Verwijs door bij complexe vragen"
```

```
behavior:
  language: "duidelijk en begrijpelijk"
  confirmation: "altijd vragen voor actie"
  privacy: "bescherm informatie over andere
  medewerkers"

constraints:
  - "Beperk toegang tot verlofgegevens van de
    gebruiker zelf"
  - "Vraag manager-approval voor uitzonderingen"
  - "Verwijs door naar HR bij twijfel"
```

YAML is krachtig als je veel variabelen hebt of als je de prompt programmatisch wilt genereren. Maar voor de meeste usecases is het overkill. Gebruik het alleen als je echt de structuur nodig hebt.

Wat zijn conversation starters en waarom zijn ze belangrijk?

Nog voordat een gebruiker een vraag stelt, wil je duidelijk maken dat het agent klaarstaat om te helpen. Dat is waar **conversation starters** in beeld komen: vooraf gedefinieerde prompts die gesprekken op gang helpen.

Vergelijk het met een baliemedewerker die vriendelijk vraagt: "Waarmee kan ik u helpen vandaag?" Zonder zo'n uitnodiging blijven gebruikers vaak hangen in twijfel of vaagheid. Een goede conversation starter biedt herkenning ("ah, dat bedoel ik!"), richting ("zo kan ik beginnen") en vertrouwen ("deze agent snapt mijn wereld").

In Copilot Studio zijn conversation starters voorgedefinieerde tekstprompts die verschijnen zodra een gebruiker het gesprek met het agent start. Ze zijn bedoeld om het ijs te breken en geven aan hoe een gebruiker veel voorkomende taken of vragen kan aanpakken.

Conversation starters er in de praktijk

Voorbeelden voor onze verlof-agent:

- "Ik wil verlof aanvragen"
- "Hoeveel verlofdagen heb ik nog?"
- "Hoe vraag ik ouderschapsverlof aan?"
- "Wanneer wordt mijn verlofaanvraag goedgekeurd?"

Het zijn geen abstracte categorieën als ‘Personeelszaken’ of ‘HR-informatie’, maar actiegerichte zinnen die je ook tegen een collega zou zeggen. Ze geven duidelijk de intentie en ondersteunde scenario’s van je agent aan.

Ontwerpprincipes voor sterke conversation starters

Volg deze vier principes als je conversation starters bedenkt voor je agent:

- **Denk in intenties, niet in onderwerpen:** Gebruikers zoeken geen informatie, ze willen iets gedaan krijgen. “Ik wil verlof aanvragen” werkt beter dan “Informatie over verlof”.

- **Gebruik alledaagse taal:** Vermijd jargon of formeel bedrijfsproza. Denk: “Hoe vraag ik ouderschapsverlof aan?” in plaats van “Aanvraagprocedure ouderschapsverlof initiëren”. Als jij het zelf niet zo zou zeggen, doe het dan niet.

- **Focus op acties:** Start je prompt met “Ik wil…” of “Hoe doe ik…”. Dat helpt de AI ook om beter te matchen op intentie. Het verschil tussen “Verlof” en “Ik wil verlof aanvragen” is voor de reactie van een agent enorm.

- **Houd het herkenbaar:** Baseer starters op veelvoorkomende vragen in je service- of HR-portaal, of analyseer eerder chatverkeer. Als 80% van je gebruikers vraagt naar vakantiedagen, zorg dat dat een conversation starter is.

Van starter naar gesprek

Een goede conversation starter is het begin, maar daarna komt het echte gesprek. Wat gebeurt er als iemand klikt op “Ik wil verlof aanvragen”? Dan moet je systeemprompt het overnemen. Je agent moet weten wat de volgende vraag is, welke data hij nodig heeft en hoe hij omgaat met onduidelijkheid.

Bijvoorbeeld: iemand zegt “Ik wil verlof aanvragen”. De agent weet uit de systeemprompt dat hij vriendelijk moet zijn en altijd moet bevestigen. Dus hij antwoordt: “Prima, ik help je daarmee. Hoeveel dagen wil je opnemen en wanneer?” En als de gebruiker zegt “Twee weken in juli”, dan snapt de agent dat en vraagt door: “Welke twee weken precies? Ik zie dat je nog 15 dagen tegoed hebt.”

Dit is waar **prompt engineering** kunst wordt. Je moet de balans vinden tussen sturen (zorg dat het gesprek op koers blijft) en flexibiliteit (gebruikers

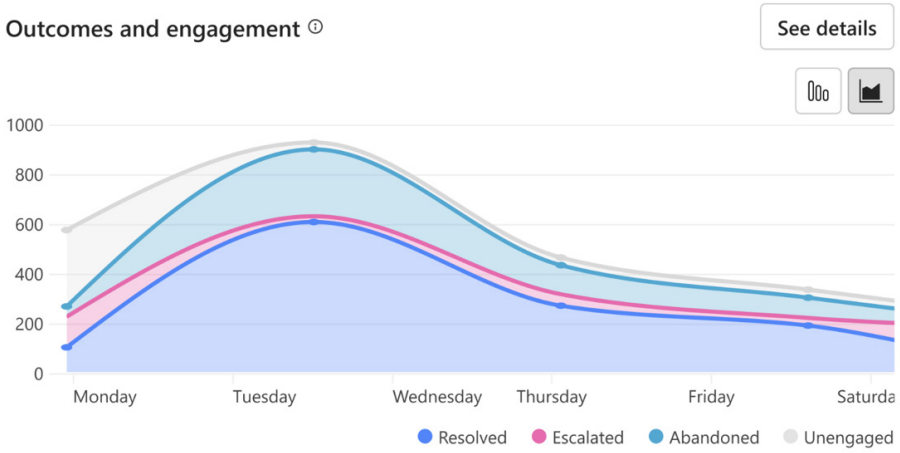
zeggen dingen op hun eigen manier). Te streng en het voelt als een formulier, te los en de agent raakt de weg kwijt.

Testen en verfijnen

De eerste versie van je prompts werkt nooit perfect. Gebruikers zeggen dingen die je niet verwacht had. Ze typen onduidelijk, ze geven incomplete informatie, ze springen tussen onderwerpen. Dat is normaal. De kunst is om daarop te anticiperen.

Test je conversation starters met echte gebruikers. Niet met collega's die weten hoe het werkt, maar met mensen die de agent voor het eerst zien. Kijk waar ze blijven haken. Welke starter kiezen ze? Waar raakt de agent de weg kwijt? Pas aan op basis van wat je ziet, niet op basis van wat je dénkt dat gebeurt.

En log alles. Copilot Studio heeft ingebouwde analytics. Gebruik die. Welke vragen worden vaak gesteld maar hebben geen conversation starter? Waar escaleert de agent naar een mens? Die informatie is goud waard voor het verbeteren van je prompts.



↳ Dashboard met AI-agent gespreksresultaten: resolved (succesvol opgelost), escalated (doorverwezen naar medewerker), abandoned (voortijdig afgebroken) en unengaged (geen interactie).

Toegankelijkheid en Inclusiviteit: AI-agents voor iedereen

Niet iedereen typt vlot. Niet iedereen ziet scherp. En niet iedereen voelt zich thuis in digitale interfaces vol tekstvelden en dropdowns. Toegankelijkheid is dus geen extraatje dat je er later nog wel bij plakt. Het is een fundamenteel ontwerpprincipe. En zoals mijn collega's al schreven in hun design-paper: als je AI-agents bouwt die bedoeld zijn voor een breed publiek, moet je verder kijken dan tekst alleen.

Het goede nieuws? Copilot Studio biedt volop mogelijkheden om je agent multimodaal te maken. Via spraak, knoppen, velden, kaarten. Door verschillende vormen van input te combineren, maak je het niet alleen makkelijker, maar ook menselijker en inclusiever. En dat is precies wat je wilt: een agent die iedereen kan gebruiken, ongeacht hun digitale vaardigheden of fysieke mogelijkheden.

Waarom toegankelijkheid bij agents zo belangrijk is

Je bouwt een verlofagent. Geweldig. Maar bedenk: niet iedere medewerker werkt op kantoor achter een toetsenbord. Sommigen zijn onderweg, anderen werken in productie of op de werkvloer. Sommige gebruikers hebben visuele beperkingen, anderen motorische uitdagingen. En dan heb je nog de groep die gewoon niet zo digitaal vaardig is, of voor wie Nederlands niet de eerste taal is.

Als je toegankelijkheid serieus neemt, bouw je een agent die al die groepen kan bedienen. Dat is geen mooi gebaar, dat is gewoon goed ontwerp. Want een agent die maar door de helft van je organisatie gebruikt kan worden, heeft weinig nut.

Multimodale input: meer dan alleen typen

De kracht van een goed ontworpen agent zit hem in de flexibiliteit. Geef gebruikers keuze in hoe ze met de agent communiceren:

- **Tekstinvoer** blijft de basis. Voor gebruikers die precies weten wat ze willen, is typen "Ik wil verlof aanvragen van 15 tot 19 juli" nog altijd het snelst.

- **Spraak** is het tweede kanaal. Vooral mobiel, of voor gebruikers die visueel beperkt zijn of gewoon liever praten dan typen. Copilot Studio integreert met Microsoft 365-kanalen die spraak ondersteunen, zoals Teams Mobile, en je kunt via Azure Speech Services spraakherkenning toevoegen. Je hoeft daar niet diep in de techniek te duiken, je kunt het gewoon aanzetten.

- **Knoppen** zijn goud waard voor snelle keuzes. “Ja”, “Nee”, “Start aanvraag”, “Annuleren”. Een gebruiker hoeft niet na te denken over formulering, gewoon klikken en klaar.

- **Dropdowns en keuzelijsten** helpen bij langere opties. Denk aan het kiezen van verloftype: vakantie, zorgverlof, ouderschapsverlof. In plaats van vrij typen (met kans op typefouten), kies je uit een lijst.

- **Adaptive Cards** maken content overzichtelijk en aanklikbaar. Daar gaan we later dieper op in, maar het punt is: je kunt informatie structureren op een manier die scanbaar en toegankelijk is.

Hoe ontwerp je spraakinteractie?

Als je spraak toevoegt, denk dan aan een paar dingen. Spraakherkenning is nooit 100% betrouwbaar. Mensen hebben verschillende accenten, spreken in verschillende tempo’s en soms is er achtergrondgeluid. Ontwerp daarom op fouttolerantie.

Geef duidelijke feedback. Als de agent een gesproken vraag niet begrijpt, zeg dat dan vriendelijk. “Sorry, ik verstond je niet helemaal. Kun je het herhalen?” Of: “Bedoelde je: verlof aanvragen?” Laat de gebruiker de herkenning bevestigen voordat je actie onderneemt.

Gebruik korte, duidelijke zinnen in je prompts. Geen vakjargon, geen complexe constructies. Denk aan hoe je tegen een collega zou praten, niet hoe je een formeel memo schrijft. En test met echte gebruikers die spraak gebruiken. Dat leert je meer dan welke technische specificatie dan ook.

Best practices voor een inclusieve agent

Laat me nog een keertje concreet maken wat toegankelijkheid in de praktijk betekent:

- **Combineer modaliteiten:** Laat gebruikers kiezen hoe ze communiceren. Tekst, spraak, knoppen, dropdowns - het hoeft niet óf-óf te zijn, het kan allemaal tegelijk.
- **Denk aan de context:** Een gebruiker op de werkvloer heeft andere behoeften dan iemand achter een bureau. Mobiel is spraak logischer, op desktop is typen vaak sneller.
- **Test met diverse groepen:** Niet alleen met je collega's die elke dag met software werken, maar ook met mensen die minder digitaal vaardig zijn, of die beperkingen hebben.
- **Zorg voor heldere taal:** Vermijd jargon. "Aanvraag indienen" in plaats van "Verlofmutatie initiëren". Dit helpt niet alleen mensen met taalachterstand, maar iedereen.
- **Bied altijd een uitweg:** Soms lukt het gewoon niet. Zorg dat er een optie is om door te schakelen naar een mens, of om het proces later opnieuw te proberen.

Tot slot: het is geen checklist

Toegankelijkheid is geen checklist. Het is een houding. Door na te denken over spraak, knoppen en alternatieve input geef je meer mensen toegang tot jouw oplossing. En vaak zijn het niet alleen mensen mét een beperking die daarvan profiteren. Een goed ontworpen voice-agent is niet alleen sneller in gebruik, maar ook meer benaderbaar voor iedereen.

De vraag is dus niet of je multimodaliteit moet overwegen. De vraag is: waarom zou je het níet doen?

Adaptive Cards voor rijke interacties

In het vorige hoofdstuk hadden we het over multimodaliteit: spraak, knoppen, dropdowns. Allemaal manieren om gebruikers meer keuze te geven in hoe ze met je agent communiceren. Maar er is nog een aspect dat vaak over het hoofd wordt gezien: visuele structuur. Want je hebt het vast meegemaakt: je stelt een vraag aan een chatbot en krijgt een eindeloze lap tekst terug. Niet overzichtelijk, niet scanbaar en zeker niet uitnodigend.

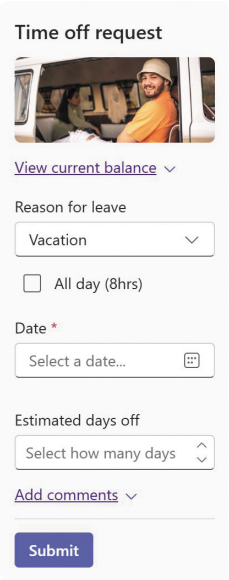
Dat is precies waar Adaptive Cards het verschil maken. In plaats van eenrichtingsverkeer in tekstvorm, geef je je AI-agent visuele kracht: kaarten met knoppen, keuzelijsten, tekstvelden en duidelijke layout. Het resultaat? Meer grip, minder verwarring en een betere gebruikerservaring.

Wat zijn Adaptive Cards?

Adaptive Cards zijn flexibele, platformonafhankelijke UI-componenten. Dat klinkt abstract, dus laat me het simpeler maken: het zijn visuele 'kaarten' die je agent kan tonen in plaats van platte tekst. Ze werken in Teams, Outlook, SharePoint en Power Apps. Ze zijn interactief, contextueel en lichtgewicht. En ze werken op mobiel, desktop en web zonder dat je voor elk platform iets anders hoeft te bouwen.

Denk aan een Adaptive Card als een formulier dat je agent voor je invult. In plaats van dat de gebruiker moet typen "Ik wil vakantieverlof van 15 tot 19 juli", toont de agent een kaart met een dropdown voor verloftype, datumvelden voor begin en eind en een knop "Verstuur aanvraag". De gebruiker klikt, selecteert, bevestigt en klaar.

↳ Voorbeeld van een adaptive card voor het opgeven van lunch dagen



→ Voorbeeld van een adaptive card voor verlof aanvragen

Waarom je Adaptive Cards moet gebruiken

Er zijn een paar goede redenen waarom Adaptive Cards geen extraatje zijn, maar een essentieel onderdeel van een goed ontworpen agent:

- **Beter scanbaar dan platte tekst:** Mensen lezen niet, ze scannen. Een kaart met headers, bullets en knoppen is in één oogopslag duidelijk. Een tekstblok van tien regels niet.
- **Rijkere interactie:** Via knoppen, formulieren en dropdowns kan een gebruiker direct actie ondernemen. Geen heen-en-weer gepraat over wat bedoel je precies, gewoon kiezen en doorgaan.
- **Consistentie en hergebruik:** Als je een goede Card hebt ontworpen, gebruik je die overal. In Teams, in Outlook, in je Power App. Eén ontwerp, alle kanalen.
- **Toegankelijkheid door gestructureerde informatie:** Dit sluit aan bij het vorige hoofdstuk: niet iedereen kan goed omgaan met vrije tekst. Een gestructureerde kaart met duidelijke velden helpt gebruikers die moeite hebben met open vragen.

Hoe voeg je Adaptive Cards toe in Copilot Studio?

Het toevoegen van een Adaptive Card in Copilot Studio is geen rocket science. Je gaat naar de 'Send message' stap in je conversatieflow en selecteert 'Adaptive Card' in plaats van gewone tekst. Daarna heb je twee opties: je gebruikt een standaardtemplate of je ontwerpt er zelf een via de Adaptive Card Designer.

Die Designer is een visuele tool waar je elementen naar je kaart sleept: tekstvelden, knoppen, afbeeldingen, dropdowns. Je ziet meteen hoe het eruitziet. En als je tevreden bent, genereer je de code en plak je die in Copilot Studio. Geen handmatig JSON typen, gewoon klikken en slepen.
Zodra je Card er staat, kun je data eraan koppelen. Bijvoorbeeld: het verloftegoed van de gebruiker, de naam van hun manager, de datum van vandaag. Dat doe je via variabelen die je hebt opgehaald uit Dataverse of een andere databron. De Card vult zichzelf met actuele informatie, elke keer weer.

En het mooie? Je kunt Adaptive Cards ook triggeren vanuit Power Automate flows. Stel: een manager krijgt een verlofaanvraag. In plaats van een saaie email, krijgt hij een Card in Teams met alle details en twee knoppen: Goedkeuren of Afwijzen.

Tips voor sterke Cards

Een Adaptive Card maken is makkelijk. Een goeie maken vraagt wat meer aandacht. Hier zijn een paar dingen die het verschil maken:

- **Ontwerp visueel, denk interactief:** Een Card is geen statische afbeelding. Het is een interface. Zorg dat knoppen duidelijk zijn, labels kloppen en de flow logisch is. Test hoe het voelt om ermee te werken.
- **Herbruik cards via Power Automate of component libraries:** Als je een goede Card hebt, sla hem op. Gebruik hem in andere flows, in andere agents. Geen dubbel werk, wel consistentie.
- **Test op mobiel en desktop:** Cards passen zich aan, maar niet altijd perfect. Kijk hoe je Card eruitziet op een klein scherm. Zijn knoppen groot genoeg? Is tekst leesbaar? Zo niet, pas aan.
- **Houd het simpel:** Een Card met tien velden en vijftien knoppen is overweldigend. Focus op wat de gebruiker nu nodig heeft. De rest kan later

Integreer je AI-agent met de rest van het Power Platform. En de rest van de wereld

Een agent die alleen met zichzelf kan praten, is nutteloos. De echte kracht van een AI-agent zit hem in de verbindingen die hij maakt. Met je bedrijfsdata, met andere systemen, met workflows die al draaien. Want jouw organisatie draait niet op één tool. Je hebt Dataverse, SharePoint, misschien ServiceNow, SAP, Salesforce. En die systemen moeten met elkaar kunnen praten, wil je agent écht waardevol zijn.

Maar voordat we in de techniek duiken, eerst het belangrijkste punt:

> **Niet alles is of hoeft een agent te zijn.**

Sommige zaken handel je handiger af met 'ouderwetse' workflows of AI-search. De kunst is om te weten waar je agent past in het proces en hoe de andere delen geautomatiseerd zijn.

Het begint met data

Maar, zoals gezegd: integratie begint altijd bij data. Welke informatie heeft je agent nodig om zijn werk te doen? Voor een verlofagent: wie is de gebruiker, hoeveel verloftegoed heeft hij, wie is zijn manager, welke aanvragen heeft hij al lopen? Voor een IT-agent: welke systemen heeft de gebruiker, welke tickets staan er open, wat zijn de standaard-oplossingen?

Die data zitten ergens. In Dataverse, in SharePoint, in een extern systeem. En je agent moet erbij kunnen.

Power Automate: de lijm tussen systemen

Power Automate is het gereedschap waarmee je acties verbindt. Je agent stelt een vraag, Power Automate haalt data op, verwerkt die en stuurt het resultaat terug. Of andersom: je agent krijgt input van een gebruiker, Power Automate triggert een approval-workflow, stuurt een notificatie naar Teams en update Dataverse.

Denk aan het verlof-voorbeeld. De gebruiker vraagt verlof aan via de agent. De agent roept een Power Automate flow aan die checkt of er genoeg tegoed is, de aanvraag opslaat in Dataverse, een goedkeuringsverzoek stuurt naar de manager via Teams en de gebruiker een bevestiging geeft. Allemaal geautomatiseerd, allemaal via Power Automate.

Het mooie van Power Automate is dat het honderden standaardconnectoren heeft. SharePoint, Outlook, Teams, Dynamics, Azure services - het zit er allemaal al in. Geen custom code nodig, kwestie van klikken en configureren.

Custom connectors: als standaard niet genoeg is

Soms zitten je data of functionaliteit in een systeem dat Power Platform niet standaard kent. Een legacy ERP, een externe API, een specifiek ticketsysteem zoals ServiceNow. Dan bouw je een **custom connector**.

Een custom connector is eigenlijk een vertaallaag. Het vertaalt wat je agent vraagt naar wat het externe systeem begrijpt en andersom. Je definieert welke acties mogelijk zijn (data ophalen, aanmaken, updaten), welke authenticatie nodig is en hoe de responses eruitzien. Eenmaal gebouwd, gebruik je die connector alsof het een standaardconnector is.

En met de laatste ontwikkelingen kun je ook een Model Context Protocol (MCP) server koppelen. Hiermee geef je de Agent een 'verzameling' tools in plaats van losse connectoren, de agent kan op basis van eigen kennis kiezen welke tool het beste past bij de taak die uitgevoerd moet worden, en zelfs verschillende tools combineren.

Voorbeeld: je IT-agent moet een ticket aanmaken in ServiceNow. Je bouwt en koppelt custom connector die met de ServiceNow API praat. In je agent roep je de juiste tool vanuit de MCP server aan met de ticket details, ServiceNow maakt het ticket aan en de tool geeft het ticketnummer terug aan de agent. Voor de gebruiker voelt het naadloos.

Dataverse en RAG: kennis ophalen op het juiste moment

Dataverse is de database van Power Platform. Daar sla je gestructureerde data op: gebruikers, aanvragen, transacties, configuratie. Je agent kan rechtstreeks met Dataverse praten om data op te halen of bij te werken.

Maar niet alle kennis is gestructureerd. Soms zit informatie in documenten, in SharePoint-pagina's, in PDF's. Daar komt RAG om de hoek kijken. RAG staat voor Retrieval-Augmented Generation, maar laten we het simpel houden: het is een manier om je agent snelle toegang te geven tot ongestructureerde kennis.

Je geeft je agent toegang tot een set documenten. Als een gebruiker een vraag stelt, zoekt de agent in die documenten naar relevante informatie,

haalt die op en gebruikt die om een antwoord te genereren. Niet door alles te kopiëren, maar door context te begrijpen en samen te vatten.

Voorbeeld: je finance-agent krijgt de vraag “Wat waren de Q3-resultaten?”. De agent zoekt in SharePoint naar het Q3-rapport, vindt de relevante sectie en geeft een samenvatting: “Omzet steeg met 12%, operationele marge verbeterde naar 18%.” De gebruiker hoeft niet zelf te zoeken, de agent doet het werk.

Als er geen API is…

Soms is er geen API of connector. Soms moet je agent werken met een systeem dat alleen een scherm heeft en geen integratiemogelijkheden. Dan gebruik je een agent die een eigen virtuele ‘computer’ heeft en dus een ‘scherm’ kan zien. Deze techniek, de we ‘computer use’ noemen en die in feite de slimme opvolger is van RPA (Robotic Process Automation), laat jouw bot letterlijk lezen, klikken en typen zoals een mens dat zou doen.

Power Platform heeft hier de techniek voor, maar in de praktijk blijkt het voor een agent niet altijd makkelijk om de weg te vinden door de schermen en formulieren van een legacy-app. We gebruiken deze techniek dus eigenlijk alleen als er geen andere optie is.

De juiste tool voor de juiste klus

En dan komen we terug bij het punt van het begin: niet alles is een agent. Als je alleen data hoeft op te zoeken, is AI-search vaak efficiënter. Als je een vast proces hebt zonder variatie, is een workflow beter. Een agent gebruik je als er beslissingen genomen moeten worden, als context belangrijk is, of als het proces dynamisch is.

Kijk naar je use case. Vraag je: heeft dit conversatie nodig, of kan het met een flow? Moet een mens erbij, of kan het volledig geautomatiseerd? Door die vragen te stellen, kies je de juiste tool. En vaak is het antwoord: een combinatie. Een agent die workflows triggert, die AI-search gebruikt voor kennis en die naadloos integreert met je bestaande systemen.

Dat is waar de kracht zit. Niet in de agent alleen, maar in hoe hij samenwerkt met de rest van je digitale ecosysteem.

Slim testen van je AI-agent

De happy flow krijgt altijd aandacht. Je bouwt een verlofagent, test of hij verlof kan aanvragen en als dat werkt ben je tevreden. Maar wie échte kwaliteit wil leveren, test vooral op de dingen die fout gaan. Want daar bewijst je agent zijn waarde - of faalt hij genadeloos.

Wat gebeurt er als een gebruiker “Ik wil vrij” typt, zonder data? Of “kaasbroodje in de cloud” omdat hij de agent even wil pesten? Of “8e maart” in plaats van “08-03-2024”? Juist in die momenten zie je of je agent robuust is of een hoopje ellende wordt.

Waarom testen op edge cases essentieel is

Testen op de happy flow is makkelijk. Je weet wat je verwacht, je weet wat er moet gebeuren en je test of dat klopt. Maar de echte wereld is rommelig. Gebruikers typen onduidelijk, stellen incomplete vragen, of doen dingen die je nooit had verwacht.

Als je daar niet op test, krijg je agents die frustreren in plaats van helpen en die je reputatie beschadigen in plaats van versterken. Bovendien: elke mislukte interactie is een kans om te leren: welke intentie miste er? Welke vraag had je moeten voorzien? Welke **fallback** had beter gekund?

Daarom is fallback-testing geen luxe, maar noodzaak. Het beschermt je reputatie, geeft inzicht in wat je mist en zorgt voor een betere gebruikerservaring door slimme “weet ik niet”-antwoorden.

Veelvoorkomende testscenario’s

Laat me concreet maken welke scenario’s je moet testen. Deze kom je overal tegen, in elke agent, voor elke usecase.

- **Onbegrijpelijke input:** De gebruiker typt “Bla bla hallo?” of “Kaasbroodje in de cloud?”. Wat doet je agent? Zegt hij “Ik begrijp je niet” en laat de gebruiker hangen? Of zegt hij “Ik snap het even niet helemaal. Ik kan je helpen met verlof aanvragen, je verloftegoed opvragen, of je lopende aanvragen bekijken. Wat wil je doen?”

- **Gedeeltelijke vragen:** “Ik wil aanvragen” zonder te zeggen wat. Of “Wanneer is…” zonder onderwerp. De gebruiker weet wat hij bedoelt, maar de agent niet. Test hoe je agent omgaat met ontbrekende context. Vraagt hij door? Biedt hij opties? Of crasht hij?

- **Conversieproblemen:** Gebruikers typen data in allerlei formaten. “8e maart”, “twee-en-twintig”, “next Friday”. Je agent moet daar mee omgaan. Test of je conversielogica robuust genoeg is, of dat rare invoer tot rare resultaten leidt.

- **Edge cases en misbruik:** Beledigende input. Vragen die niks met je agent te maken hebben. Pogingen om je agent dingen te laten doen die niet mogen. Test hoe je agent grenzen bewaakt. Blijft hij beleefd maar duidelijk? Of laat hij zich verleiden tot ongewenst gedrag?

- **Intentie-verwarring:** “Wachtwoord wissen” versus “wachtwoord resetten”. Voor een gebruiker klinkt dat hetzelfde, maar technisch kan het iets anders betekenen. Test of je agent die nuance oppakt, of vraagt om verduidelijking.

Hoe ontwerp je slimme fallbacks?

Een fallback is niet “Sorry, dat begrijp ik niet” en klaar. Dat is lui. Een goede fallback geeft erkenning, richting en alternatieven:

- **Erkenning:** Laat weten dat je het hoort, ook al snap je het niet helemaal. “Ik snap het even niet helemaal…” voelt vriendelijker dan “Onbegrijpelijke invoer”.

- **Alternatieven:** Bied knoppen met mogelijke opties. “Bedoelde je: verlof aanvragen, verloftegoed opvragen, of iets anders?” De gebruiker hoeft niet opnieuw te typen, gewoon klikken.

- **Doorverwijzen:** Als je agent echt niet kan helpen, verwijs dan slim door. Met een link naar de kennisbank, of een optie om door te schakelen naar een mens.

- **Leren:** Log onbekende vragen. Analyseer wat gebruikers vragen waar je niet op voorbereid was. Voeg die intenties toe aan je agent, of pas je prompts aan. Elke gemiste vraag is een kans om beter te worden.

Tools die helpen bij testen

Je hoeft het wiel niet opnieuw uit te vinden. Er zijn tools die je helpen bij testen:

- **Testcanvas in Copilot Studio:** Hier speel je gesprekken na, simuleer je variaties en bekijk je welke triggers matchen. Je ziet direct of je agent doet wat je verwacht.
- **Conversation transcript logging via monitoring:** Analyseer niet-gematchte vragen. Kijk welke input je agent niet begreep en pas je ontwerp aan.
- **LUIS of Azure Language Studio:** Gebruik NLU-training om te testen hoe intenties afwijken. Zie welke varianten van een vraag je agent wel en niet oppikt.
- **Prompt flow testing met variabelen:** Simuleer ontbrekende data. Test wat er gebeurt als een variabele leeg is, of een onverwachte waarde heeft.

Deployment en governance

We hadden het in hoofdstuk 2 al over Application Lifecycle Management en governance. Hier de korte herhaling: gebruik ALM voor deployment. Commit changes naar source control, gebruik DevOps of GitHub-pipelines, plan updates en rollback-scenario's.

Inmiddels zijn er ook technieken waarmee je agents geautomatiseerd kunt testen, net zoals je dat met code doet. Deze 'evaluations' zijn gebruikersprompts met daarbij de kwaliteitscriteria waar de antwoorden aan moeten voldoen. Zo kun je na iedere aanpassing zien of je agent nog de juiste antwoorden geeft en zich houdt aan de regels die je hebt gesteld.

En vergeet governance niet. DLP policies bepalen wat je agent mag. Logging en auditing houden bij wat hij doet. En beheersing van reputatierisico's voorkomt dat je agent verkeerde output geeft die gebruikers frustreert of schade aanricht.

Dit hoort bij testen. Want je test niet alleen of je agent werkt, maar ook of hij veilig werkt, binnen de regels, met controle en met de mogelijkheid om snel terug te rollen als het toch misgaat.

Van theorie naar praktijk: werkende AI-agents in actie

We hebben het gehad over architectuur, prompts, testing en toegankelijkheid. Nu wordt het concreet. Hoe ziet een werkende AI-agent er in de praktijk uit? Niet als theoretisch model, maar als daadwerkelijke oplossing die morgen al zou kunnen draaien in jouw organisatie.

Hieronder vind je een paar praktische scenario's die laten zien hoe AI-agents in Power Platform échte problemen oplossen. Van een eenvoudige HR-verlofaanvraag tot een multi-agent samenwerking tussen afdelingen. Het doel? Je een blauwdruk geven die je kunt aanpassen aan jouw eigen situatie.

Scenario 1: HR-agent voor verlofaanvragen via Teams

Medewerkers willen snel verlof aanvragen zonder in portalen te hoeven zoeken. De agent biedt conversation starters zoals "Ik wil verlof aanvragen". Vervolgens toont hij een Adaptive Card met dropdown voor verloftype, datumvelden voor begin en eind en een bevestigingsknop.

De agent checkt het verloftegoed in Dataverse, triggert een Power Automate approval workflow naar de manager en stuurt een Teams-notificatie bij goedkeuring of afwijzing. Edge cases: wat als tegoed onvoldoende is? De agent waarschuwt vooraf en biedt alternatieven. Wat bij overlappende aanvragen? De agent checkt bestaande aanvragen en meldt conflicten.

Copilot Studio → Dataverse → Agent Flows → Teams

Scenario 2: Finance-agent voor rapportage en samenvatting

Finance teams willen snel rapporten ophalen en laten samenvatten. Een gebruiker vraagt "Geef me het Q3

rapport". De agent zoekt in SharePoint naar het document, gebruikt Azure AI Search om de inhoud te verwerken en genereert via prompt engineering een samenvatting met key takeaways. Die samenvatting toont hij in een Adaptive Card met bullets voor de belangrijkste punten. Via RAG haalt de agent context uit eerdere rapporten om trends te herkennen. Security is cruciaal: de agent checkt via Microsoft Entra ID of de gebruiker toegang heeft tot het gevraagde rapport.

Copilot Studio → SharePoint → Azure AI → gebruiker

Scenario 3: Multi-agent orchestratie – samenwerking tussen HR en IT

Nieuwe medewerker onboarding vraagt om samenwerking tussen HR en IT. Een manager meldt "Nieuwe collega start volgende week". De HR-agent regelt contract, verloftegoed en introductie in Dataverse. Tegelijkertijd triggert hij de IT-agent via een handoff in Copilot Studio. Die IT-agent bestelt een laptop, creëert accounts in Azure AD en regelt toegang tot systemen.

Beide agents delen data via Dataverse zodat ze elkaar niet in de weg zitten. Power Automate orchestreert de volgorde: eerst HR-taken, dan IT-taken. De gebruiker ziet een naadloze overdracht tussen agents zonder dat hij merkt dat er twee systemen werken.

Multi-agent flow met gedeelde context via Dataverse

Bonus scenario's voor inspiratie

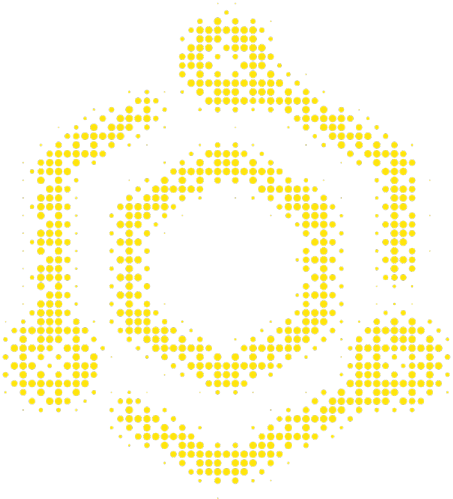
- **Facilities-agent:** Medewerker wil vergaderruimte boeken voor morgen 14:00 met 8 personen. Agent checkt beschikbaarheid in Outlook, boekt de ruimte, vraagt of catering nodig is en regelt parkeerplaatsen voor externe bezoekers.

- **Sales-agent:** Lead komt binnen via webformulier. Agent kwalificeert de lead op basis van bedrijfsgrootte en budget, update CRM met leadstatus en stuurt een follow-up herinnering naar de accountmanager na drie dagen.

- **Support-agent:** Gebruiker meldt "Mijn applicatie crasht". Agent zoekt in de kennisbank naar bekende problemen, vindt een oplossing en deelt die. Werkt het niet? Agent maakt automatisch een ticket aan in het supportsysteem met alle details.

- **Legal-agent:** Medewerker zoekt een NDA-template voor een nieuwe klant. Agent zoekt in SharePoint naar goedgekeurde contracten, checkt of er specifieke compliance-eisen zijn voor dit land en genereert een template met de juiste variabelen ingevuld.

- **Marketing-agent:** Campagnemanager vraagt "Wat is de status van onze Q4-campagne?". Agent haalt data uit marketing automation tool, toont open rate, click rate en conversies en suggereert op basis van AI-analyse welke content het best presteert.



Hoe meet je of de agent succesvol is?

Om het succes van een agent te meten kun je de volgende metrics hanteren:

- Aantal afgehandelde aanvragen per dag/week/maand
- Gebruikerstevredenheid scores (CSAT, NPS)
- Tijdwinst vs. handmatig proces
- Percentage succesvol afgeronde gesprekken (completion rate)
- Aantal escalaties naar mens
- Adoptiegraad: hoeveel gebruikers maken er daadwerkelijk gebruik van?

Jij bepaalt welke metric(s) passen bij jouw specifieke situatie. Bij een verlofaanvraag-agent is bijvoorbeeld het aantal verwerkte aanvragen wel relevant, en een CSAT-score niet. Kies pragmatisch: welke cijfers geven écht inzicht in de toegevoegde waarde?

Lessons learned uit de praktijk

- Start simpel, schaal daarna
- Gebruikers willen geen AI zien, maar resultaat
- Fallbacks zijn net zo belangrijk als happy flows
- Governance vanaf dag 1, niet achteraf
- Meet succes met concrete metrics

“Start simpel, schaal daarna”

Begin vandaag, leer onderweg

We zijn begonnen met de vraag: hoe bouw je AI-agents die écht werken? Niet als theoretisch model, maar als oplossing die morgen al draait in je organisatie. De antwoorden staan in de voorgaande hoofdstukken. Maar laat me het samenvatten.

1. **Begin met waarom:** Welk probleem los je op? Voor wie? En waarom is een AI-agent de juiste oplossing? Zonder duidelijk antwoord op die vragen, bouw je iets dat niemand gebruikt.

2. **Zorg dat je architectuur op orde is:** Solutions environments, DLP policies, security - dat zijn geen details, dat is de basis. Zonder stevige fundering stort je agent in zodra je probeert te schalen.

3. **Ontwerp je conversatie met zorg:** Goede prompts, slimme conversation starters en vooral: goede fallbacks. Want je agent bewijst zijn waarde niet in de happy flow, maar als het even niet lukt.

4. **Integreer met de rest van je ecosysteem:** Data is koning. Zonder data is je agent een lege huls. Maak verbinding met Dataverse, SharePoint, externe systemen. Gebruik Power Automate als lijm.

5. **Test grondig:** Niet alleen de happy flow, maar vooral de edge cases. Wat gebeurt er als iets fout gaat? Hoe reageert je agent? Dat bepaalt of gebruikers hem vertrouwen.

6. **En begin klein:** Eén scenario, één agent, goed gebouwd. Schaal daarna. Binnen governance, met metrics, met leren van wat werkt en wat niet.

De toekomst van werk is niet mens óf AI. Het is mens én AI, samen. Agents die taken overnemen zodat mensen kunnen focussen op wat echt belangrijk is. Dat bouw je niet in één keer. Dat bouw je stap voor stap, met elke agent die je lanceert.

Dus: kies een scenario. Bouw het. Test het. Leer ervan. De rest komt vanzelf.

AI-agents bouwen op Power Platform

Praktijkgids voor makers die low-code AI-agents willen bouwen die echt gebruikt worden.

Met heldere principes, een praktisch stappenplan en herkenbare voorbeelden staat je agent morgen in productie.

Over Albert-Jan Schot

Albert-Jan is senior cloudarchitect en CTO bij Blis Digital, verantwoordelijk voor de technische visie en strategie van het low-code domein.

Hij is Microsoft MVP voor M365 Development en M365 Apps & Services en deelt actief zijn praktijkervaring en kennis met de community en Microsoft.

Blis Digital

Unleash the power of technology

www.blisdigital.com