

# Grenzen van de low-code tool Mendix.

## Optionele ondertitel.

---

**Senne Timbreur.**

Scriptie voorgedragen tot het bekomen van de graad van  
Professionele bachelor in de toegepaste informatica

**Promotor:** Mevr. F. Spriet

**Co-promotor:** Mevr. J. Alexander

**Academiejaar:** 2024–2025

**Tweede examenperiode**

**Departement IT en Digitale Innovatie .**

**HO  
GENT**



# Woord vooraf

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

# Samenvatting

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

# Inhoudsopgave

<b>Lijst van figuren</b>	<b>viii</b>
<b>Lijst van tabellen</b>	<b>ix</b>
<b>Lijst van codefragmenten</b>	<b>x</b>
<b>Lijst van afkortingen</b>	<b>xi</b>
<b>1 Inleiding</b>	<b>1</b>
1.1 Kadering begrippen	1
1.1.1 High-code	1
1.1.2 Low-code	2
1.2 Probleemstelling	2
1.3 Onderzoeksvraag	3
1.3.1 Probleemdomein	3
1.3.2 Oplossingsdomein	3
1.4 Onderzoeksdoelstelling	3
1.5 Opzet van deze bachelorproef	4
<b>2 Stand van zaken</b>	<b>6</b>
2.1 Model-Driven Development	7
2.1.1 Kernconcepten van Model-Driven Development	7
2.1.2 Historische context en evolutie	8
2.1.3 Voordelen van Model-Driven Development	8
2.1.4 Belangrijkste functionaliteitsgebieden voor MDD-tools	9
2.1.5 De evolutie naar low-code platformen	10
2.2 Low-code platformen	10
2.2.1 OutSystems	11
2.2.2 Joget DX	12
2.2.3 Mendix	13
2.2.4 Keuze voor Mendix als low-code platform	14
2.3 Mendix en high-code	15
2.3.1 Verschillende benaderingen	15
2.3.2 Enterprise Flexibiliteit door Model-Based Engineering	15
2.3.3 Low-Code als Brug tussen Business en IT	16
2.3.4 Kunnen beiden elkaar aanvullen?	16
2.4 Keuze tussen high-code en low-code?	17

<b>3</b>	<b>Methodologie</b>	<b>18</b>
3.1	Methodologie	18
3.1.1	Fase 1: Analyse van het probleemdomein	18
3.1.2	Fase 2: Onderzoek naar het oplossingsdomein	19
<b>4</b>	<b>Huidige gang van zaken</b>	<b>22</b>
4.1	Analyse van historische projectdata	22
4.2	Interviews met experts	23
4.3	Documentatieonderzoek	25
<b>5</b>	<b>Hoe kan het beter?</b>	<b>26</b>
5.1	Literatuuroverzicht	26
5.1.1	Praktijkvoorbeelden en Casestudies	26
5.1.2	Ervaringen en Knelpunten uit Ontwikkelaarsgemeenschappen	27
5.1.3	Academische Inzichten in Low-Code Ontwikkeling	27
5.2	Praktisch onderzoek	27
5.2.1	Schaalbaarheid en prestaties	28
5.2.2	Ontwikkelingsnelheid en onderhoud	38
5.2.3	Integratiemogelijkheden	40
5.2.4	Aanpasbaarheid en uitbreidbaarheid	42
5.2.5	Beveiliging en compliance	43
5.2.6	Samenvattende conclusie praktisch onderzoek	44
5.3	Reflectie op eigen ervaringen	45
5.3.1	Reflectie op ervaringen van experts	45
5.4	Evaluatie van Mendix-uitbreidingen	46
5.5	Ontwikkeling beslissingskader	47
5.5.1	1. Cijfermatige evaluatiematrix	47
5.5.2	2. Beoordelingscriteria per dimensie	49
5.5.3	3. Richtlijnen voor hybride projecten	49
5.5.4	4. Risicobeheersing en leerstrategieën	49
<b>6</b>	<b>Conclusie</b>	<b>51</b>
<b>A</b>	<b>Onderzoeksvoorstel</b>	<b>53</b>
A.1	Inleiding	54
A.2	Literatuurstudie	56
A.2.1	Low-code en Mendix	56
A.2.2	Low-code use cases	56
A.2.3	Beperkingen van low-code	57
A.2.4	Low-code versus high-code	58
A.3	Methodologie	58
A.3.1	Fase 1: Analyse van het probleemdomein	59
A.3.2	Fase 2: Onderzoek naar het oplossingsdomein	60

A.4 Verwacht resultaat, conclusie . . . . .	60
---	----

<b>Bibliografie</b>	<b>62</b>
---------------------	-----------

# Lijst van figuren

2.1	Evolution of programming	7
2.2	Visuele ontwikkelomgeving Outsystems	11
2.3	Visuele ontwikkelomgeving Joget DX	12
2.4	Visuele ontwikkelomgeving Mendix	13
5.1	Homepage Mendix applicatie	28
5.2	Homepage Mendix applicatie	28
5.3	Lighthouse testresultaten JavaScript/React - Links: Test 1, Midden: Test 2, Rechts: Test 3	30
5.4	Lighthouse testresultaten Mendix - Links: Test 1, Midden: Test 2, Rechts: Test 3	31
5.5	Testresultaten toevoegen object in JavaScript/React	33
5.6	Testresultaten toevoegen object in Mendix	35
5.7	Testresultaten load test: ophalen objecten in JavaScript/React	36
5.8	Testresultaten load test: ophalen objecten in Mendix	37
5.9	Checkbox aanzetten filters	39
5.10	Homepage with search bar	39
5.11	Homepage with search bar JavaScript/React	40



# Lijst van tabellen

2.1	Low-Code Platforms Comparison . . . . .	15
5.1	Testresultaten laadtijd JavaScript/React . . . . .	30
5.2	Testresultaten laadtijd Mendix . . . . .	31
5.3	Vergelijkingstabel laadtijd . . . . .	32
5.4	Testresultaten reactietijd JavaScript/React . . . . .	34
5.5	Breakdown reactietijd JavaScript/React . . . . .	34
5.6	Testresultaten reactietijd Mendix . . . . .	35
5.7	Breakdown reactietijd Mendix . . . . .	35
5.8	Loadtestresultaten JavaScript/React: Kernmetriecken . . . . .	37
5.9	Loadtestresultaten Mendix: Kernmetriecken . . . . .	37
5.10	Integratiemogelijkheden . . . . .	42
5.11	Beoordelingscriteria per dimensie . . . . .	48

# Lijst van codefragmenten

1	Trainingstabel met zoekfunctie op naam . . . . .	40
---	--	----

# Lijst van afkortingen

- AI** Artificiële intelligentie. 13
- API** Application Programming Interface. 14, 15, 38, 39
- CASE** Computer Aided Software Engineering. 8
- CLS** Cumulative Layout Shift. 29–32
- DEMO** Design and Engineering Methodology for Organizations. 16
- DOM** Document Object Model. 34
- FCP** First Contentful Paint. 29–32
- FOD MOB** Federale Overheidsdienst Mobiliteit. 22, 24, 25
- GC** Grabage Collection. 35
- LCP** Largest Contentful Paint. 29–32
- MBE** Model-Based Engineering. 16
- MDA** Model Driven Architecture. 8
- MDD** Model Driven Development. 6–10
- ML** Machine Learning. 13
- PWA** Progressieve Webapp. 12, 15
- TBT** Total Blocking Time. 29–32
- UI** User Interface. 34, 35
- UML** Unified Modeling Language. 9
- UX** User Experience (gebruikerservaring). 12
- XHR** XMLHttpRequest. 34, 36
- xUML** executable Unified Modeling Language. 8

# 1

## Inleiding

Apvine, een toonaangevend IT-consultancybedrijf, maakt intensief gebruik van low-code platformen zoals Mendix om zakelijke applicaties snel en efficiënt te ontwikkelen. Deze aanpak verlaagt de ontwikkeltijd en biedt een flexibele oplossing voor veel projecten. Echter, wanneer applicaties complexe bedrijfslogica, real-time gegevensverwerking of integraties met legacy-systemen vereisen, kunnen de beperkingen van low-code ontwikkeling zichtbaar worden.

In zulke gevallen kan een hybride of high-code aanpak nodig zijn, maar zonder een duidelijk beslissingskader is het lastig om te bepalen wanneer deze overstap gemaakt moet worden. Dit gebrek aan richtlijnen kan leiden tot projectvertragingen, hogere kosten en onverwachte aanpassingen, wat een negatieve impact heeft op zowel Apvine als haar klanten.

Dit onderzoek richt zich op het vaststellen van de grenzen van Mendix en het ontwikkelen van een gestructureerd beslissingskader, zodat projectmanagers en softwarearchitecten beter onderbouwde keuzes kunnen maken tussen low-code en high-code ontwikkeling.

### 1.1. Kadering begrippen

Voorafgaand aan de uitwerking van deze bachelorproef is het belangrijk om enkele kernbegrippen te kaderen. Door vooraf duidelijk te omschrijven wat we precies verstaan onder termen zoals low-code en high-code, zorgen we ervoor dat alle lezers met dezelfde basiskennis starten. Dit voorkomt verwarring en maakt de verdere analyse en besluitvorming beter te volgen en te interpreteren.

#### 1.1.1. High-code

High-code, ook wel traditionele softwareontwikkeling genoemd, houdt in dat applicaties volledig worden geprogrammeerd met behulp van programmeertalen zoals

Java, C#, of Python. Deze aanpak biedt maximale controle, technische diepgang en flexibiliteit, en maakt het mogelijk om robuuste, schaalbare en volledig op maat gemaakte oplossingen te bouwen. High-code is bij uitstek geschikt voor toepassingen met complexe bedrijfslogica, uitgebreide integraties, hoge prestaties of specifieke beveiligingseisen. Hoewel het meer tijd en gespecialiseerde kennis vereist dan low-code, levert het doorgaans beter onderhoudbare en diep geïntegreerde software op.

### 1.1.2. Low-code

Low-code is een methode van softwareontwikkeling waarbij applicaties voornamelijk worden gebouwd via visuele interfaces, met gebruik van drag-and-drop componenten en vooraf geconfigureerde modules. Dit stelt gebruikers in staat om snel functionerende applicaties te creëren met minimale handmatige codering. Low-code platforms zijn ontworpen om de ontwikkelsnelheid te verhogen en softwareontwikkeling toegankelijk te maken voor een breder publiek, inclusief business users zonder diepgaande programmeerkennis. Belangrijk om te vermelden is dat deze platforms doorgaans de mogelijkheid bieden om waar nodig aanvullende code (high-code) toe te voegen. Hierdoor kunnen ontwikkelaars de standaardcomponenten uitbreiden of aanpassen aan complexe vereisten, wat de flexibiliteit van low-code aanzienlijk vergroot.

## 1.2. Probleemstelling

De groeiende populariteit van low-code platformen zoals Mendix biedt bedrijven de mogelijkheid om sneller en efficiënter zakelijke applicaties te ontwikkelen. Deze aanpak verlaagt de technische drempel en versnelt het ontwikkelproces. Toch stuiten bedrijven zoals Apvine op beperkingen van deze technologie wanneer ze complexe bedrijfslogica, real-time gegevensverwerking en integraties met legacy-systemen moeten implementeren.

Er ontbreekt momenteel een duidelijk beslissingskader binnen Apvine om te bepalen wanneer low-code een geschikte oplossing is en wanneer high-code ontwikkeling noodzakelijk wordt. Dit leidt tot projectvertragingen, hogere kosten en mogelijke klantontevredenheid door onverwachte aanpassingen in de ontwikkelingsstrategie.

Dit onderzoek richt zich op het identificeren van de grenzen van Mendix en het formuleren van een gestructureerd beslissingskader. Hiermee kunnen projectmanagers en softwarearchitecten beter onderbouwde keuzes maken over de inzet van low-code en high-code technologieën in verschillende scenario's.

## **1.3. Onderzoeksvraag**

Dit onderzoek richt zich op de onderzoeksvraag: "Wat zijn de beperkingen van de low-code-tool Mendix bij het ontwikkelen van zakelijke applicaties en welke criteria bepalen wanneer een overstap naar high-code ontwikkeling noodzakelijk wordt?" Om deze centrale vraag te beantwoorden, worden de volgende deelvragen onderzocht:

### **1.3.1. Probleemdomein**

- Welke gevolgen heeft het ontbreken van een beslissingskader voor de keuze tussen low-code en high-code binnen Apvine?
- Welke problemen ontstaan er in projecten door het gebrek aan richtlijnen voor de keuze tussen ontwikkelmethoden?
- Wat zijn de huidige criteria die Apvine gebruikt bij het kiezen tussen low-code en high-code ontwikkeling?
- Welke knelpunten ervaren projectmanagers en architecten bij het maken van deze keuze?

### **1.3.2. Oplossingsdomein**

- Wat zijn de technische beperkingen van Mendix bij het omgaan met complexe bedrijfslogica, real-time gegevensverwerking en integraties met legacy-systemen?
- In welke specifieke scenario's binnen Mendix-projecten kan een hybride of high-code oplossing nodig zijn, en welke factoren bepalen deze overstap?
- Hoe kunnen projectomvang, tijdslijnen en klantvereisten de beslissing beïnvloeden om wel of niet Mendix (low-code) te gebruiken of over te schakelen naar high-code?
- Welke lessen kunnen worden getrokken uit eerdere projecten bij Apvine waarin Mendix werd ingezet, en hoe kunnen deze inzichten bijdragen aan een effectief beslissingskader?

Door deze vragen te beantwoorden, beoogt dit onderzoek een onderbouwd beslissingskader te ontwikkelen dat projectmanagers en softwarearchitecten ondersteunt bij het maken van een weloverwogen keuze tussen low-code en high-code ontwikkelmethoden.

## **1.4. Onderzoeksdoelstelling**

Het primaire doel van dit onderzoek is het ontwikkelen van een uitgebreid beslissingskader dat projectmanagers en softwarearchitecten van Apvine in staat stelt

om gefundeerde keuzes te maken tussen low-code (Mendix) en high-code ontwikkelingsmethoden. Dit kader zal worden ontworpen om de technische beperkingen van Mendix te identificeren en te bepalen wanneer de overstap naar high-code oplossingen noodzakelijk wordt.

De concrete resultaten van dit onderzoek zullen omvatten:

- Een gedetailleerde analyse van de beperkingen van Mendix bij het verwerken van complexe bedrijfslogica, real-time gegevensverwerking en integraties met legacy-systemen
- Een gestructureerd beslissingskader met duidelijke criteria voor het bepalen van de optimale ontwikkelingsaanpak
- Een reeks praktische richtlijnen en beoordelingsinstrumenten die projectmanagers kunnen toepassen in de beginfase van projectplanning
- Casestudies van eerdere Apvine-projecten, die succesvolle en uitdagende implementaties van low-code oplossingen illustreren
- Aanbevelingen voor het implementeren van hybride benaderingen wanneer dit passend is

Het succes van dit onderzoek zal worden gemeten aan de hand van:

- De toepasbaarheid van het kader op verschillende projecttypen en klantver-eisten
- Validatie door technische experts en projectmanagers van Apvine met betrekking tot de praktische bruikbaarheid van het kader
- Het vermogen om potentiële technische knelpunten te voorspellen voordat ze impact hebben op projectplanningen of budgetten
- Duidelijke documentatie van het besluitvormingsproces die met klanten kan worden gedeeld om de transparantie te verbeteren

Het eindresultaat zal een uitgebreid rapport zijn met het beslissingskader, vergezeld van praktische hulpmiddelen (zoals beoordelingschecklists en beslisbomen) die direct kunnen worden geïntegreerd in de projectmethodologie van Apvine.

## **1.5. Opzet van deze bachelorproef**

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 wordt een overzicht van de huidige situatie, waarbij de bestaande processen en werkwijzen worden geanalyseerd. Er wordt gekeken naar de effectiviteit van deze gang van zaken en eventuele knelpunten die zich voordoen.

In Hoofdstuk 5 worden mogelijke verbeteringen besproken, op basis van de bevindingen uit het vorige hoofdstuk. Er worden alternatieve benaderingen en oplossingen gepresenteerd die kunnen bijdragen aan een efficiëntere en effectievere werkwijze

In Hoofdstuk 6, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.



# 2

## Stand van zaken

Mendix vertegenwoordigt een van de toonaangevende low-code ontwikkelplatformen in het huidige softwareontwikkelingslandschap (Hermans & Mileff, [2023](#)). Als implementatie van **Model Driven Development (MDD)** principes stelt Mendix zowel professionele ontwikkelaars als zakelijke gebruikers in staat om applicaties te creëren via visuele modellering in plaats van traditionele codering. Het platform heeft aanzienlijke tractie gewonnen onder ondernemingen die hun digitale transformatie-initiatieven willen versnellen door ontwikkeltijd en technische complexiteit te verminderen (van Oosten, [2020](#)).

Hoewel Mendix via zijn modelgestuurde aanpak tal van voordelen biedt, waaronder verhoogde ontwikkelingssnelheid en bredere participatie van niet-technische belanghebbenden, presenteert het ook bepaalde beperkingen die de effectiviteit in verschillende contexten beïnvloeden (Yerukala, [2022](#)). Deze bachelorproef onderzoekt Mendix als een hedendaagse manifestatie van **MDD** en analyseert kritisch de beperkingen ervan op het gebied van technische mogelijkheden, uitdagingen bij organisatorische adoptie en comparatieve nadelen ten opzichte van vergelijkbare platformen.

Door zowel de theoretische onderbouwing van **MDD** als de praktische implementatie ervan in Mendix te begrijpen, beoogt dit onderzoek een uitgebreide beoordeling te geven van waar en hoe Mendix mogelijk tekortschiet in het aanpakken van complexe enterprise software behoeften, ondanks de innovatieve benadering van applicatieontwikkeling.

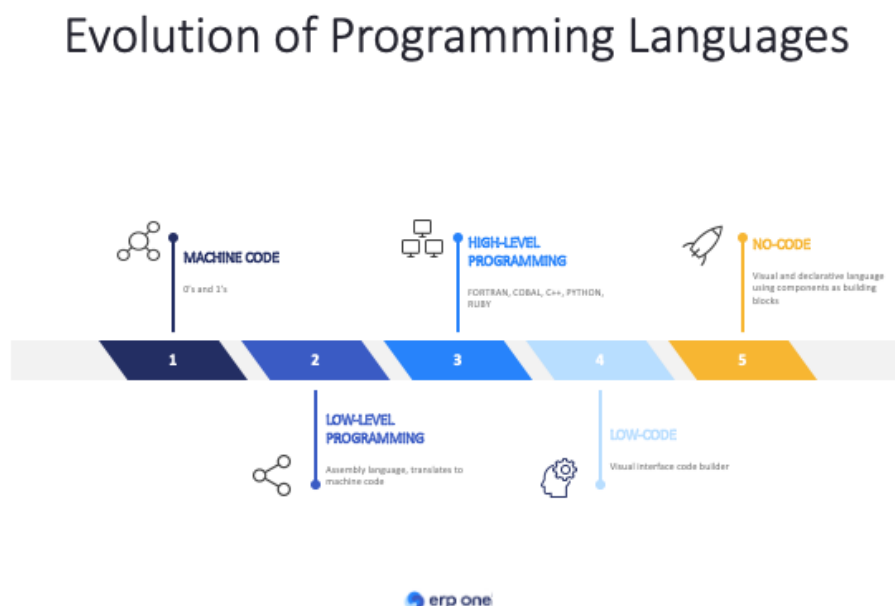
## 2.1. Model-Driven Development

In dit hoofdstuk wordt de theoretische achtergrond van MDD en low-code ontwikkeling uiteengezet, met een specifieke focus op Mendix. Dit sluit aan bij de introductie van de stand van zaken, waarin het belang van MDD en low-code ontwikkeling werd geïntroduceerd. Dit hoofdstuk biedt een diepgaande analyse van MDD, inclusief de voordelen en beperkingen, en gaat na hoe Mendix binnen dit kader past.

### 2.1.1. Kernconcepten van Model-Driven Development

Volgens het onderzoek Hailpern en Tarr (2006) is MDD “een software engineering aanpak die bestaat uit de toepassing van modellen om het abstractieniveau te verhogen” in softwareontwikkeling. Deze aanpak ontstond als een natuurlijke evolutie in de overgang van low-level programmeertalen zoals assembly naar higher-level talen zoals Java en C#, waarbij modellen de volgende stap in deze abstractie-evolutie vertegenwoordigen.

Het fundamentele principe achter MDD is dat het werken op hogere abstractieniveaus ontwikkelaars in staat stelt om complexe systemen effectiever en met minder inspanning te beheren. Dit komt overeen met de historische evolutie van programmeertalen zoals te zien in onderstaande afbeelding. Hierbij wordt bij elke nieuwe generatie de abstractie vergroot om de productiviteit te verbeteren en de cognitieve belasting van ontwikkelaars te verminderen.



**Figuur 2.1:** Evolutie van programmeertalen naar steeds hogere abstractieniveau (Vanderkooy, 2021).

### 2.1.2. Historische context en evolutie

MDD heeft een rijke geschiedenis die meer dan 40 jaar teruggaat (Henkel & Stirna, 2010). De evolutie verliep in verschillende fasen:

In de jaren '80 ontstonden de eerste geavanceerde modelleringsmethoden om de vereisten van informatiesystemen vast te leggen. Dit vormde de conceptuele basis voor MDD (Henkel & Stirna, 2010).

De jaren '90 brachten propriëtaire Computer Aided Software Engineering (CASE)-tools, die informatiesystemen gedeeltelijk konden genereren op basis van modellen. Deze tools hadden echter beperkingen - vaak konden ze alleen codetemplates genereren waarna programmeurs de rest handmatig moesten aanvullen (Case, 1985).

Modernere benaderingen omvatten Model Driven Architecture (MDA) en executable Unified Modeling Language (xUML). MDA werkt met verschillende modelleringsniveaus en transformaties tussen deze niveaus naar code, waarbij de gegenereerde code verder kan worden uitgewerkt (Soley & the OMG Staff Strategy Group, 2000). xUML daarentegen streeft naar volledige codegeneratie zonder handmatige aanvullingen.

Er bestaan grote verschillen tussen oudere CASE-tools zoals Oracle Forms en Microsoft Access Forms en nieuwere tools gebaseerd op open standaarden zoals OptimalJ. Hoewel beide soorten tools MDD ondersteunen door abstractieniveaus te verhogen, bieden tools zoals OptimalJ meer controle over modeltransformaties en codegeneratie, terwijl formulier-gebaseerde tools zoals Oracle Forms minder controle bieden (Henkel & Stirna, 2010).

Ondanks deze verschillen delen alle MDD-benaderingen één fundamenteel doel: het verhogen van het abstractieniveau in softwareontwikkeling, waardoor ontwikkelaars complexere systemen kunnen bouwen met minder inspanning - vergelijkbaar met de evolutie van assembly-taal naar hogere programmeertalen zoals C# en Java.

### 2.1.3. Voordelen van Model-Driven Development

De literatuur identificeert verschillende belangrijke voordelen van het gebruik van MDD-benaderingen (Soley & the OMG Staff Strategy Group, 2000). Door te werken met goed gedefinieerde modellen kunnen ontwikkelaars het aantal implementatiefouten verminderen, waardoor de softwarekwaliteit verbetert. Hogere abstractieniveaus stellen ontwikkelaars in staat om zich te richten op bedrijfslogica in plaats van technische implementatiedetails, wat de productiviteit van ontwikkelaars verhoogt. Modellen geven een duidelijker inzicht in de systeemstructuur en het gedrag, waardoor het ontwikkelproces beter onder controle is.

Bovendien vermindert het automatisch genereren van code de handmatige coördineringsinspanning en de bijbehorende fouten, wat leidt tot lagere ontwikkel- en onderhoudskosten. Snellere ontwikkelcycli helpen organisaties efficiënter om te gaan

met hun applicatieontwikkelingsbehoeften, waardoor de ontwikkelingsachterstand afneemt. De mogelijkheid om snel iteraties uit te voeren en werkende functionaliteit te demonstreren verbetert de betrokkenheid van belanghebbenden, waardoor de klanttevredenheid toeneemt.

#### **2.1.4. Belangrijkste functionaliteitsgebieden voor MDD-tools**

Om een MDD-tool effectief te laten zijn, moet het verschillende kritieke functionaliteiten ondersteunen, die kunnen worden gecategoriseerd in ondersteuning van zowel het modelleren als het ontwikkelproces.

Effectieve ondersteuning bij het modelleren vereist dat tools de juiste abstractieniveaus hanteren, waarbij irrelevante details worden verborgen terwijl essentiële concepten worden blootgelegd.

Dit is vooral belangrijk omdat belanghebbenden modellen voor verschillende doeleinden gebruiken. Modellen moeten begrijpelijk zijn voor zowel technische als niet-technische belanghebbenden, idealiter met behulp van intuïtieve, voorspelbare notatie - veel tools maken om deze reden gebruik van **Unified Modeling Language (UML)**, omdat dit wordt beschouwd als de de-facto standaard in de industrie (Marín e.a., 2015) . Modellen moeten uitvoerbaar zijn, zelfs als ze incompleet zijn, zodat incrementele ontwikkeling mogelijk is en ontwikkelaars het gedrag van het systeem kunnen voorspellen door middel van experimenten of formele analyse.

Volwassen MDD-tools ondersteunen de verfijning van modellen en transformaties tussen verschillende abstractieniveaus, waardoor aanpasbare model-naar-model en model-naar-code transformaties mogelijk zijn. Volgens Marín e.a. (2015) “moeten MDD-tools de uitvoering van modellen mogelijk maken, ook al zijn ze onvolledig, maar wel geldig”, wat modelcorrectie en -validatie in een vroeg stadium vergemakkelijkt.

Ondersteuning van het ontwikkelproces omvat een reeks functies. Tools moeten duidelijke feedback geven over fouten, idealiter door direct te wijzen naar de modelcomponenten die problemen veroorzaken, vergelijkbaar met hoe compilers problematische code markeren. Omdat software meestal door teams wordt ontwikkeld, moeten MDD-tools modelvergelijking, samenvoeging en versiebeheer ondersteunen.

Marín e.a. (2015) benadrukken dat “versiebeheer van modellen absoluut noodzakelijk is om industriële samenwerkingsprojecten te beheren, waarbij verschillende leden van een ontwikkelteam aan hetzelfde model kunnen werken”. Effectieve tools moeten snel compileren en implementeren, met een bijzonder efficiënte afhandeling van incrementele wijzigingen.

MDD-tools moeten integreren met bestaande systemen en ontwikkelomgevingen, waardoor verbindingen met ERP-systemen, legacy applicaties en andere bedrijfsinfrastructuur mogelijk worden. Een van de belangrijkste voordelen van MDD is dat een breder scala aan mensen kan deelnemen aan de ontwikkeling, waaronder

bedrijfskundigen met beperkte programmeervaardigheden. Tools moeten de definitie en het gebruik van herbruikbare componenten, patronen en best practices in projecten vergemakkelijken.

Door deze functies op te nemen kunnen MDD-tools beter aansluiten op de behoeften van de industrie en de succesvolle toepassing van het MDD-paradigma in softwareontwikkelingsprojecten ondersteunen.

### 2.1.5. De evolutie naar low-code platformen

MDD is de laatste jaren sterk geëvolueerd, vooral met de opkomst van low-code platformen zoals Mendix. Deze platformen vertegenwoordigen een verfijning van MDD-principes, waardoor ze toegankelijker worden voor ontwikkelaars met verschillende vaardigheidsniveaus, terwijl de kernvoordelen van modelgedreven benaderingen behouden blijven.

Zoals Henkel en Stirna (2010) aantoonde in hun analyse van Mendix, proberen moderne MDD-tools een balans te vinden tussen abstractie en controle, waardoor ontwikkelaars op bedrijfsniveau kunnen redeneren terwijl ze toch voldoende controle houden over het systeemgedrag. Deze evolutie heeft MDD-principes toegankelijk gemaakt voor een breder publiek, waardoor softwareontwikkeling democratischer wordt dan alleen voor traditionele programmeerexperts. Henkel en Stirna (2010) benadrukken dat “low-code platforms zoals Mendix met succes de kloof hebben overbrugd tussen abstractie op hoog niveau en technische implementatie, waardoor zowel zakelijke belanghebbenden als ontwikkelaars effectief kunnen samenwerken.”

Concluderend kan gesteld worden dat MDD een significante vooruitgang betekent in software engineering methodologie, door modellen te verheffen van secundaire documentatie tot primaire ontwikkelartefacten. Door het abstractieniveau te verhogen stelt MDD ontwikkelaars in staat om complexiteit effectiever te managen, waardoor de productiviteit kan toenemen, de kwaliteit kan verbeteren en softwareontwikkeling toegankelijker wordt voor een breder scala aan belanghebbenden. De integratie van MDD-principes in low-code platforms heeft het bereik verder vergroot, waardoor het een krachtig hulpmiddel is geworden voor moderne softwareontwikkeling.

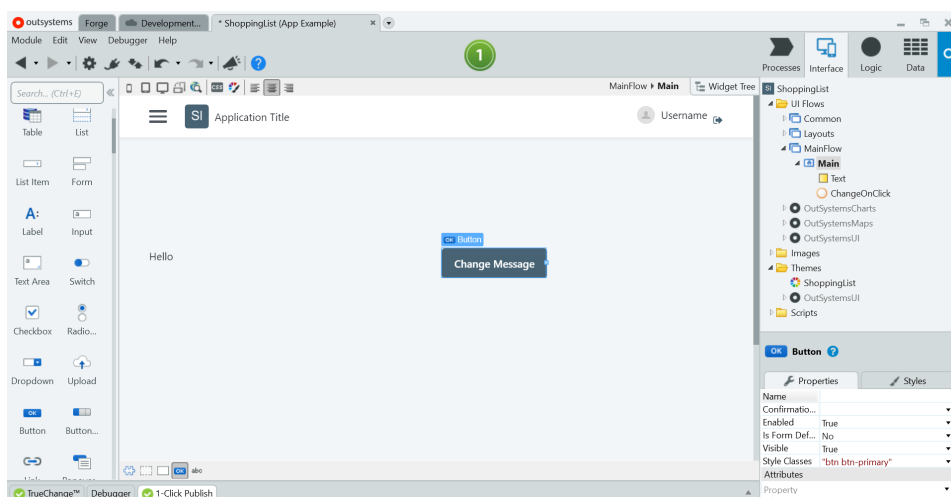
## 2.2. Low-code platformen

In de volgende paragrafen zullen we de belangrijkste kenmerken, sterke punten en beperkingen van verschillende toonaangevende low-code ontwikkelplatformen onderzoeken en vergelijken, waaronder OutSystems, Joget DX en Mendix. Elk platform biedt unieke mogelijkheden en komt tegemoet aan verschillende organisatorische behoeften en use cases. Door hun benaderingen van visuele ontwikkeling, schaalbaarheid, inzetmogelijkheden, maatwerk en integratiemogelijkheden te onderzoeken, willen we een duidelijk inzicht geven in hoe deze platforms zich tot el-

kaar verhouden. Uiteindelijk zal deze vergelijking duidelijk maken waarom Mendix de meest uitgebreide en veelzijdige oplossing is, met de beste balans tussen flexibiliteit, schaalbaarheid en geavanceerde functies voor bedrijven die hun digitale transformatie willen versnellen.

### 2.2.1. OutSystems

OutSystems is een robuust low-code platform dat bekend staat om zijn uitgebreide functies en enterprise-gerichte capaciteiten. Het platform onderscheidt zich door een intuïtieve visuele ontwikkelomgeving waarmee ontwikkelaars via drag-and-drop interfaces en voorgedefinieerde componenten snel applicaties kunnen bouwen. Deze gebruiksvriendelijke interface stelt zowel ervaren ontwikkelaars als gebruikers met minder technische kennis in staat om efficiënt te werken (Sido & Emon, 2024). Op het gebied van schaalbaarheid biedt OutSystems sterke prestaties voor enterprise-toepassingen, waarbij de architectuur is ontworpen om mee te groeien met toenemende gebruikersaantallen en datavolumes. Het platform ondersteunt zowel cloud-native als on-premises implementatieopties, waardoor organisaties flexibel kunnen kiezen voor de deploymentstrategie die het beste past bij hun infrastructuur en compliance-vereisten.



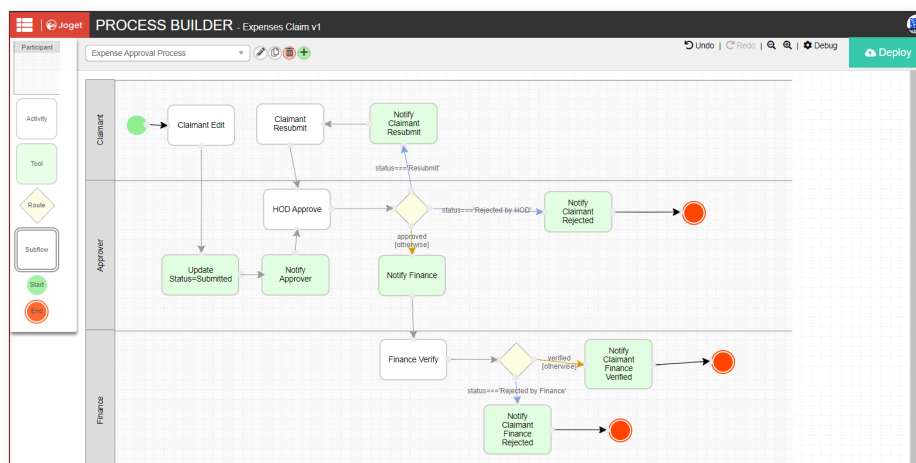
**Figuur 2.2:** Visuele ontwikkelomgeving van OutSystems (Figueira-Putresza, 2021).

Wat betreft maatwerk biedt OutSystems uitgebreide aanpassingsmogelijkheden via eigen extensies en de integratie van custom code wanneer de standaard visuele ontwikkelingstools niet toereikend zijn. Dit stelt ontwikkelaars in staat om complexe bedrijfslogica te implementeren, hoewel dit soms ten koste kan gaan van de ontwikkelsnelheid (Sido & Emon, 2024). Voor integraties beschikt het platform over diverse mogelijkheden om te koppelen met externe systemen en API's, maar er zijn beperkingen bij de integratie met sommige databases en uitdagingen bij migraties, vooral bij complexe legacy-systemen. OutSystems kan verder bogen op een levendig ecosysteem en community, samen met sterke beveiligingsfuncties en na-

leving van industriestandaarden. Het licentiemodel en de kostenbeperkingen kunnen echter onbetaalbaar zijn voor sommige organisaties, terwijl de kwaliteit van de documentatie en de geïsoleerde ondersteuning van de community barrières kunnen opwerpen voor ontwikkelaars die het volledige potentieel van het platform willen benutten (Sido & Emon, 2024).

### 2.2.2. Joget DX

Joget DX is een open-source low-code platform dat zich onderscheidt door zijn toegankelijkheid en focus op snelle applicatieontwikkeling. Op het gebied van visuele ontwikkeling biedt het platform een intuïtieve interface die ontworpen is voor eenvoud, waardoor ook niet-technische gebruikers snel aan de slag kunnen met het bouwen van applicaties. Het is vooral sterk in de ontwikkeling van **Progressieve Webapp (PWA)**'s en legt veel nadruk op **User Experience (gebruikerservaring) (UX)**, waardoor eindgebruikers kunnen profiteren van moderne, responsieve interfaces (Sido & Emon, 2024). Qua schaalbaarheid kent Joget DX enkele beperkingen door het abonnementsmodel en app-beperkingen, wat uitdagingen kan opleveren voor grootschalige enterprise-toepassingen die hoge prestatie-eisen stellen.



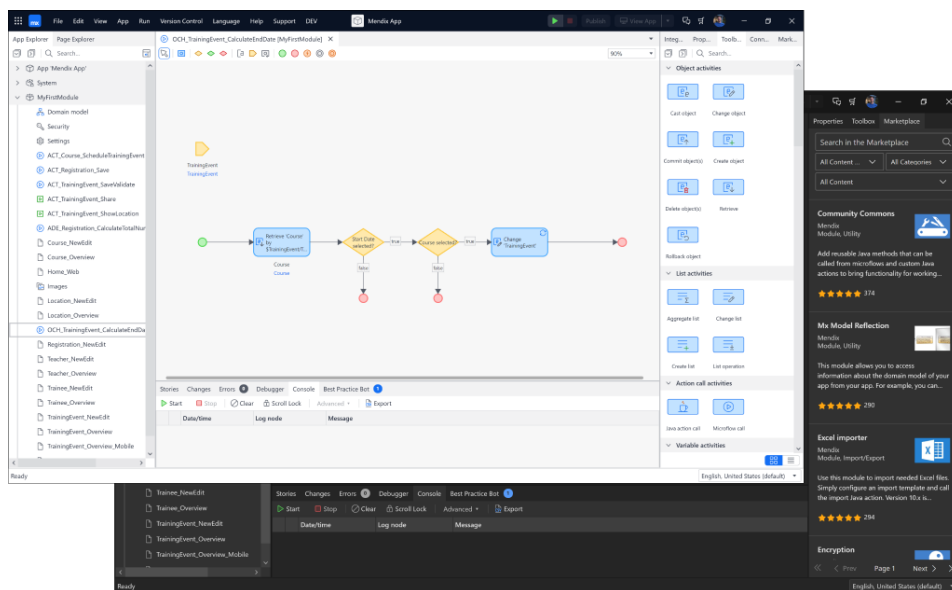
**Figuur 2.3:** Visuele ontwikkelomgeving van Joget DX (Hugo, 2024).

Voor inzetmogelijkheden biedt het platform goede integratie met DevOps-praktijken, waardoor het goed past binnen moderne ontwikkelworkflows en continuous delivery-processen. Het open-source karakter geeft organisaties flexibiliteit in de implementatie, hoewel de beperkte databasetoegang sommige deployment-scenario's kan bemoeilijken. Op het gebied van maatwerk beschikt Joget DX over uitbreidbaarheid via add-on builders en verbeterde workflowmogelijkheden, waardoor het een flexibele keuze is voor bedrijven die specifieke bedrijfsprocessen willen automatiseren en aanpassen aan hun behoeften (Sido & Emon, 2024). De integratiemogelijkheden zijn redelijk robuust voor standaard use-cases, maar kunnen beperkingen vertonen bij complexere scenario's of legacy-systemen, wat de algehele aan-

pasbaarheid kan verminderen. Bovendien kunnen de leercurve en aanpassingscomplexiteit van het platform uitdagingen vormen voor gebruikers die overstappen van traditionele ontwikkelmethoden, wat extra training en gewenningstijd kan vereisen.

### 2.2.3. Mendix

Mendix wordt gepositioneerd als een veelzijdig low-code platform dat een breed scala aan functies biedt voor zowel technische als niet-technische gebruikers. Het platform maakt gebruik van een modelgedreven ontwikkelaanpak, gecombineerd met microservices en containerisatie, wat bijdraagt aan de schaalbaarheid, flexibiliteit en draagbaarheid. Mendix ondersteunt zowel cloud-native als on-premise infrastructuren, wat zorgt voor flexibiliteit in de inzetmogelijkheden. Het platform biedt volledige levenscyclusbeheer, evenals functionaliteiten voor **Artificiële intelligentie (AI)**, **Machine Learning (ML)** en procesautomatisering, waardoor het geschikt is voor een breed scala aan zakelijke toepassingen (Sido & Emon, 2024). Daarnaast biedt Mendix openheid en uitbreidbaarheid, wat integratie met externe systemen mogelijk maakt en aanpassingen voor specifieke bedrijfsbehoeften ondersteunt. Hoewel er enkele beperkingen zijn, zoals de beperkte aanpasbaarheid van thema's en het risico van vendor lock-in, bieden de functionaliteiten en het gebruiksgemak aanzienlijke voordelen (Sido & Emon, 2024).



**Figuur 2.4:** Visuele ontwikkelomgeving van Mendix (Mendix, 2025b).



#### 2.2.4. Keuze voor Mendix als low-code platform

Hoewel OutSystems en Joget DX waardevolle functies en mogelijkheden bieden, komt Mendix naar voren als het beste platform vanwege zijn uitgebreide en flexibele benadering van low-code ontwikkeling. Zoals onderstaande vergelijkingstabel laat zien, blinkt Mendix uit op alle belangrijke criteria: visuele ontwikkeling, schaalbaarheid, inzetmogelijkheden, maatwerk en integratiemogelijkheden. Het vermogen van Mendix om complexe bedrijfsapplicaties te ondersteunen met zijn geavanceerde visuele interface is superieur aan de concurrentie. De uitstekende schaalbaarheid voor enterprise-toepassingen overtreft de beperkingen die bij Joget DX worden ervaren. Daarnaast biedt Mendix meer veelzijdige deployment-opties dan beide concurrenten, terwijl de uitgebreide aanpassingsmogelijkheden en krachtige Microflow-editor meer flexibiliteit bieden dan de alternatieven. Waar OutSystems worstelt met databaseintegratie en Joget DX beperkt wordt in complexere integratiescenario's, biedt Mendix uitstekende **Application Programming Interface (API)**-integratie met uitgebreide connectors voor externe systemen. Deze combinatie van sterke punten maakt Mendix de ideale keuze voor organisaties die op efficiënte en effectieve wijze digitale transformatie willen stimuleren, ongeacht de complexiteit van hun bedrijfsprocessen of technische vereisten.

Criteria	OutSystems	Joget DX	Mendix
<b>Visuele ontwikkeling</b>	Intuïtieve drag-and-drop interface	Eenvoudige interface gericht op PWA's	Zeer gebruiksvriendelijk; geschikt voor alle niveau's
<b>Schaalbaarheid</b>	Sterk; ontworpen voor groei	Beperkt door abonnementsmodel	Uitstekend voor enterprise; hoge belasting
<b>Inzetmogelijkheden</b>	Cloud en on-premises opties	DevOps-integratie; beperkte database-opties	Veelzijdig (cloud/on-premises/hybrid); CI/CD-integratie
<b>Maatwerk</b>	Extensies en custom code mogelijk	Add-on builders; goede workflows	Uitgebreid; Java-extensies; krachtige Microflow
<b>Integratiemogelijkheden</b>	Diverse opties; enkele database-beperkingen	Adequaat voor standaard gebruik; beperkingen bij complexiteit	Uitstekende API's; veel connectors; legacy-ondersteuning

Tabel 2.1: Vergelijking Low-code platformen.

## 2.3. Mendix en high-code

### 2.3.1. Verschillende benaderingen

Low-code en high-code representeren twee verschillende benaderingen van softwareontwikkeling, elk met unieke voordelen en uitdagingen. Low-code platforms, zoals Mendix, bieden een visuele ontwikkelomgeving waarin applicaties grotendeels zonder handmatig programmeren kunnen worden gebouwd (Krouwel e.a., 2022). Dit versnelt de ontwikkeltijd en maakt softwareontwikkeling toegankelijker voor niet-technische gebruikers. Daarentegen biedt high-code, waarbij programmeertalen zoals Java of .NET worden gebruikt, maximale flexibiliteit en controle, wat essentieel is voor complexe of sterk aangepaste oplossingen (Krouwel e.a., 2022). Hoewel low-code efficiënter is en snellere iteraties mogelijk maakt, kan het beperkingen hebben in maatwerk en prestaties vergeleken met high-code. De keuze tussen beide hangt af van de behoeften van de organisatie: low-code is ideaal voor snelle, aanpasbare applicaties, terwijl high-code de voorkeur geniet voor diepgaande technische en schaalbare oplossingen.

### 2.3.2. Enterprise Flexibiliteit door Model-Based Engineering

Krouwel e.a. (2022) benadrukt dat enterprise agility een cruciale succesfactor is in een steeds dynamischere markt, waarin bedrijven te maken hebben met hyper-

concurrentie, veranderende regelgeving en technologische innovaties. Traditionele informatiesystemen vormen vaak een beperkende factor voor deze flexibiliteit, omdat ze hardcoded bedrijfsregels en structuren bevatten die moeilijk aanpasbaar zijn. Door **Model-Based Engineering (MBE)** te gebruiken, kunnen organisaties hun informatiesystemen ontwikkelen op basis van ontologische modellen, zoals die binnen de **Design and Engineering Methodology for Organizations (DEMO)**-methodologie worden gehanteerd. Deze modellen beschrijven de essentiële processen en interacties binnen een onderneming, los van specifieke implementaties. Dit maakt het mogelijk om snel en systematisch wijzigingen door te voeren, zonder dat ontwikkelaars handmatig code hoeven aan te passen. Het resultaat is een grotere wendbaarheid van zowel de organisatie als haar IT-systemen, waardoor bedrijven sneller kunnen inspelen op veranderingen zonder dat technologie een beperkende factor vormt.

### 2.3.3. Low-Code als Brug tussen Business en IT

Een belangrijke conclusie uit Krouwel e.a. (2022) is dat low-code technologie niet alleen zorgt voor een snellere ontwikkeling van software, maar ook de samenwerking tussen business en IT aanzienlijk verbetert. Traditionele softwareontwikkeling vereist vaak uitgebreide specificaties en langdurige communicatie tussen ontwikkelaars en business stakeholders, wat leidt tot vertragingen en misinterpretaties. Low-code platforms, zoals Mendix, bieden een visuele ontwikkelomgeving waarin bedrijfsgebruikers direct kunnen meedenken en bijdragen aan het ontwerp van applicaties (Mendix, 2023). Dit verlaagt de drempel om wijzigingen door te voeren en zorgt ervoor dat IT-systemen beter aansluiten op de behoeften van de organisatie. Doordat bedrijfsprocessen en bijbehorende applicaties sneller en effectiever kunnen worden aangepast, wordt de time-to-market verkort en kunnen organisaties flexibeler inspelen op nieuwe kansen en uitdagingen.

### 2.3.4. Kunnen beiden elkaar aanvullen?

Volgens Krouwel e.a. (2022) kunnen low-code en high-code elkaar versterken door de flexibiliteit van **MBE** te combineren met de aanpasbaarheid van traditionele codering. Low-code platforms zoals Mendix maken gebruik van abstracties en modelgestuurde technieken, waardoor applicaties sneller kunnen worden ontwikkeld zonder dat complexe code nodig is. Tegelijkertijd biedt high-code de mogelijkheid om de gegenereerde applicaties verder te verfijnen, bijvoorbeeld door aangepaste logica, integraties of prestatie-optimalisaties toe te voegen. In het onderzoek wordt benadrukt dat low-code vooral geschikt is om snel te reageren op veranderingen in bedrijfsprocessen, terwijl high-code nodig blijft voor diepgaande aanpassingen en geavanceerde automatiseringen. Door beide methoden te combineren, ontstaat een hybride aanpak waarbij organisaties profiteren van de snelheid en wendbaarheid van low-code zonder in te boeten op de kracht en maatwerkopties van high-

code.

## **2.4. Keuze tussen high-code en low-code?**

Bij het kiezen tussen high-code en low-code ontwikkelmethoden is het essentieel om vooraf duidelijk te bepalen welke functionaliteiten en mate van maatwerk je voor je applicatie nodig hebt (Ballejos, [2024](#)). High-code ontwikkeling biedt maximale flexibiliteit en is geschikt voor complexe, op maat gemaakte oplossingen, maar vereist aanzienlijke tijd en technische expertise. Low-code platforms versnellen het ontwikkelproces door gebruik te maken van visuele tools en vooraf gebouwde componenten, wat ideaal is voor applicaties die snel moeten worden ontwikkeld met een zekere mate van aanpasbaarheid (Ballejos, [2024](#)). Door vooraf je specifieke eisen en doelen te definiëren, kun je de ontwikkelmethode kiezen die het beste aansluit bij de behoeften van je project en organisatie.

# 3

## Methodologie

### 3.1. Methodologie

Dit onderzoek is opgedeeld in twee fasen: eerst het in kaart brengen van het probleemdomain (het ontbreken van een beslissingskader), gevolgd door onderzoek naar het oplossingsdomain (de ontwikkeling van het kader).

#### 3.1.1. Fase 1: Analyse van het probleemdomain

Om een grondig inzicht te krijgen in het probleemdomain worden de volgende onderzoeksmethoden gebruikt:

##### Analyse van historische projectdata

Om deelvragen 1 en 2 te beantwoorden ("Wat zijn de gevolgen van het ontbreken van een beslissingskader?" en "Welke problemen ontstaan er in projecten?"), wordt een gestructureerde analyse uitgevoerd van afgesloten projecten. Bij deze analyse worden initiële projectschattingen vergeleken met werkelijke uitkomsten om de impact op kosten en doorlooptijd nauwkeurig te kwantificeren. Daarnaast wordt de projectdocumentatie systematisch doorgenomen met als doel het identificeren van specifieke momenten waarop keuzes tussen low-code en high-code ontwikkelmethoden tot problemen hebben geleid. Het onderzoek inventariseert ook situaties waarin projectteams gedwongen waren om tijdens het project over te schakelen naar alternatieve ontwikkelmethoden vanwege onvoorziene beperkingen. Ten slotte wordt de financiële en tijdsimpact van deze late aanpassingen grondig geëvalueerd om de werkelijke kosten van suboptimale initiële technologiekeuzes inzichtelijk te maken.

##### Interviews met experts

Voor het beantwoorden van deelvragen 3 en 4 ("Wat zijn de huidige criteria?" en "Welke knelpunten ervaren projectmanagers?") worden diepte-interviews gehou-

den met verschillende belanghebbenden binnen het ontwikkelproces. Projectmanagers worden geïnterviewd over hun huidige besluitvormingsproces bij technologiekeuzes, waarbij specifiek wordt gefocust op de impliciete en expliciete criteria die zij hanteren. Daarnaast delen architecten hun ervaringen met technologiekeuzes, inclusief de technische overwegingen die doorslaggevend zijn bij het maken van platformkeuzes. Pre-sales consultants worden bevraagd over hun methodiek bij het inschatten van projectgeschiktheid voor verschillende ontwikkelplatformen in de offertefase. Tot slot worden ontwikkelaars geïnterviewd over de praktische uitdagingen die zij ervaren wanneer technologiekeuzes zijn gemaakt en zij deze moeten implementeren, met bijzondere aandacht voor de discrepantie tussen verwachtingen en werkelijkheid.

### **Documentatieonderzoek**

Om deelvraag 3 verder te onderbouwen wordt bestaande interne documentatie geanalyseerd. Dit omvat een grondige bestudering van de huidige richtlijnen en procedures voor projectaanpak, waarin impliciet of expliciet keuzes worden gemaakt over ontwikkelmethoden. Ook worden pre-sales documentatie en offertes onder de loep genomen om inzicht te krijgen in de initiële afwegingen en beloftes die worden gedaan voordat een project daadwerkelijk start. Project kick-off documenten worden onderzocht om de uitgangspunten en verwachtingen aan het begin van projecten te identificeren, met specifieke aandacht voor de technologiekeuzes die in deze vroege fase worden vastgelegd. Tenslotte worden architectuurbeslissingen en design documents geanalyseerd om te begrijpen hoe technische overwegingen worden gedocumenteerd en gecommuniceerd binnen projectteams, en hoe deze documenten bijdragen aan het besluitvormingsproces rondom low-code versus high-code ontwikkeling.

#### **3.1.2. Fase 2: Onderzoek naar het oplossingsdomein**

Na het volledig in kaart brengen van het probleem, richt het onderzoek zich op het ontwikkelen van een oplossing middels:

### **Literatuuroverzicht**

Een diepgaande analyse van bestaande documentatie over Mendix en andere low-code platforms vormt een essentieel onderdeel voor het beantwoorden van deelvraag 5 en 7. Deze analyse omvat diverse bronnen, waaronder officiële Mendix productgidsen die gedetailleerde technische specificaties en functionaliteiten beschrijven, casestudies van derden en rapporten uit de industrie die praktijkvoorbeelden en onafhankelijke evaluaties bieden, online forums en ontwikkelaarsgemeenschappen waar praktijkervaringen en knelpunten worden gedeeld, en academische publicaties over low-code ontwikkeling die theoretische onderbouwing en wetenschappelijke inzichten verschaffen over de bredere context van deze technologie.

### Praktisch onderzoek

Voor de beantwoording van deelvraag 6 worden praktische tests uitgevoerd om de grenzen van het Mendix-platform te onderzoeken. Dit onderzoek richt zich op verschillende aspecten, waaronder schaalbaarheid en prestaties, waarbij de reactietijd, belastingstestresultaten en het vermogen om te schalen bij toenemende gebruikersaantallen worden geëvalueerd. Daarnaast worden de integratiemogelijkheden getest om de compatibiliteit met externe systemen, API's en databases te beoordelen. De aanpasbaarheid en uitbreidbaarheid van het platform worden geanalyseerd door te onderzoeken in hoeverre maatwerkfunctionaliteiten en uitbreidingen kunnen worden geïmplementeerd. Ook de ontwikkelingssnelheid wordt onder de loep genomen, waarbij wordt gekeken naar de efficiëntie van de ontwikkelomgeving en de snelheid waarmee applicaties kunnen worden gebouwd en aangepast. Tot slot wordt de beveiliging en compliance beoordeeld om vast te stellen in hoeverre het platform voldoet aan relevante regelgeving en beveiligingsstandaarden. Deze tests bieden een diepgaand inzicht in de sterke en zwakke punten van Mendix en helpen bij het bepalen van de geschiktheid van het platform voor specifieke toepassingen.

### Reflectie op eigen ervaringen

In dit onderzoek wordt een gestructureerde reflectie uitgevoerd op de transitie van high-code naar low-code ontwikkeling binnen een lopend Mendix-project. Hierbij worden persoonlijke ervaringen gedocumenteerd, waarbij zowel de uitdagingen als successen in de praktijk worden belicht. Er wordt een vergelijkende analyse gemaakt van de ontwikkelingsefficiëntie tussen traditionele high-code methoden en de Mendix-aanpak, met aandacht voor de verschillen in snelheid, flexibiliteit en onderhoudbaarheid. Daarnaast worden specifieke situaties geïdentificeerd waarin high-code kennis een toegevoegde waarde biedt binnen een Mendix-omgeving, bijvoorbeeld bij complexe logica of integraties met externe systemen. Tot slot wordt de leercurve geëvalueerd en worden de benodigde aanpassingen in denkwijze besproken die gepaard gaan met de overstap naar een low-code platform. Deze reflectie biedt waardevolle inzichten in de impact van low-code ontwikkeling op bestaande programmeerervaringen en werkwijzen.

### Evaluatie van Mendix-uitbreidingen

In dit onderzoek wordt de effectiviteit van verschillende uitbreidingsstrategieën voor Mendix geanalyseerd. Een belangrijk aspect daarbij is de vergelijking tussen standaard Marketplace-modules en custom ontwikkeling, waarbij wordt gekeken naar factoren zoals flexibiliteit, implementatietijd en onderhoudsgemak. Daarnaast wordt de impact van Java-uitbreidingen onderzocht, met specifieke aandacht voor de invloed op onderhoudbaarheid en toekomstbestendigheid van het platform. Ook wordt een analyse uitgevoerd van de kosten-batenverhouding van verschillende uitbreidingsmethoden, waarbij zowel ontwikkel- als beheerkosten wor-

den meegenomen. Tot slot worden de integratie-uitdagingen beoordeeld die zich voordoen bij het combineren van Mendix met externe systemen en services, om zo inzicht te krijgen in de haalbaarheid en complexiteit van verschillende uitbreidingsopties. Deze analyse draagt bij aan een beter begrip van de meest effectieve strategieën voor het uitbreiden van Mendix-toepassingen binnen verschillende bedrijfscontexten.

### **Ontwikkeling beslissingskader**

Voor het beantwoorden van deelvraag 8, die zich richt op het destilleren van best practices en richtlijnen voor de keuze tussen Mendix en traditionele ontwikkeling, wordt een gestructureerd beslissingskader ontwikkeld. Dit kader ondersteunt projectmanagers en architecten bij het maken van weloverwogen keuzes over de inzet van Mendix in vergelijking met conventionele ontwikkelmethoden. Het biedt beslissingspunten voor de initiële keuze tussen Mendix en traditionele ontwikkeling en stelt criteria vast om te bepalen welke projectonderdelen beter geschikt zijn voor high-code ontwikkeling. Daarnaast worden richtlijnen geformuleerd voor het effectief combineren van low-code en high-code in hybride projecten, zodat beide benaderingen optimaal kunnen worden ingezet. Verder bevat het kader aanbevelingen voor de vroege identificatie van potentiële beperkingen en risico's, waardoor projectrisico's proactief kunnen worden gemitigeerd. Tot slot worden strategieën ontwikkeld om op basis van eerdere ervaringen en literatuur weloverwogen keuzes te maken bij toekomstige projecten. Dit beslissingskader draagt bij aan een systematische en onderbouwde aanpak voor het selecteren van de meest geschikte ontwikkelmethode binnen uiteenlopende projectcontexten.



# 4

## Huidige gang van zaken

### 4.1. Analyse van historische projectdata

In de samenwerking tussen Apvine en **Federale Overheidsdienst Mobiliteit (FOD MOB)**, een klant met een intern Java-developmentteam, komt de keuze tussen high-code en low-code regelmatig terug als een belangrijk discussiepunt. Het ontbreken van een duidelijk en gestructureerd beslissingskader bemoeilijkt het maken van weloverwogen technologisch onderbouwde keuzes, wat vaak leidt tot suboptimale beslissingen. Dit maakt **FOD MOB** tot een uitstekend casevoorbeeld om de gevolgen van besluiteloosheid en het ontbreken van heldere richtlijnen in het technologiekeuzeproces te analyseren. In dit onderzoek worden initiële projectschattingen vergeleken met de werkelijke uitkomsten, met als doel de impact op kosten en doorlooptijd nauwkeurig te kwantificeren. Daarnaast wordt de projectdocumentatie systematisch doorgenomen om concrete momenten te identificeren waarop de keuze tussen low-code en high-code tot frictie heeft geleid. Het onderzoek inventariseert ook in welke gevallen projectteams, bijvoorbeeld binnen **FOD MOB**, gedwongen waren om tijdens het project over te schakelen naar alternatieve ontwikkelmethoden, als gevolg van onvoorziene beperkingen van het gekozen platform of ontwikkelparadigma. De financiële en tijdsimpact van deze late aanpassingen wordt vervolgens grondig geëvalueerd, om de werkelijke kosten van een gebrekkig technologisch beslissingskader inzichtelijk te maken.

Het ontbreken van een duidelijk beslissingskader heeft aanzienlijke gevolgen voor de technologische keuzes binnen de samenwerking tussen Apvine en **FOD MOB**. In plaats van op objectieve criteria en een gestructureerd evaluatieproces te vertrouwen, worden de keuzes voor high-code versus low-code vaak beïnvloed door informele voorkeuren en machtsverhoudingen binnen het team. Dit gebrek aan een formele en transparante methodologie leidt tot verschillende problemen die

de effectiviteit van het project kunnen ondermijnen:

### **Onzekerheid in de technologiekeuze**

Zonder een duidelijk beslissingskader is het voor betrokkenen moeilijk om met zekerheid te bepalen welke technologie het beste past bij de specifieke projectvereisten. Dit leidt vaak tot keuzes die niet goed afgestemd zijn op de werkelijke behoeften van het project, wat in de praktijk kan resulteren in inefficiëntie en suboptimale resultaten.

### **Frictie binnen het projectteam**

Wanneer er geen duidelijke richtlijnen zijn voor het maken van technologische keuzes, ontstaat er vaak frictie binnen het projectteam. Teamleden hebben mogelijk verschillende opvattingen over de voor- en nadelen van low-code versus high-code, wat leidt tot onduidelijkheid en conflicten. Dit vergroot de kans op vertragingen, aangezien beslissingen niet snel genomen kunnen worden door gebrek aan consensus.

### **Vertraagde besluitvorming**

Het proces van het maken van een keuze wordt uitgesteld of herhaald, omdat er geen objectief beslissingskader is om de discussie te structureren. Deze vertragingen kunnen het project belemmeren, vooral in de vroege fasen waarin cruciale technologische keuzes gemaakt moeten worden. Dit vertraagt de voortgang van het project en kan leiden tot hogere kosten en een langere doorlooptijd.

### **Noodzaak tot herziening van keuzes**

Wanneer er geen formele afstemming en evaluatie plaatsvindt, worden technologische keuzes vaak pas later in het project beoordeeld. Dit kan betekenen dat gekozen technologieën niet voldoen aan de functionele eisen, schaalbaarheid of onderhoudsvereisten. Het projectteam moet dan dure en tijdrovende herzieningen doorvoeren, zoals het aanpassen van architectuur of het implementeren van alternatieve technologieën, wat het project verder vertraagt en de kosten verhoogt.

Kortom, het ontbreken van een helder beslissingskader maakt het lastig om weloverwogen keuzes te maken, verhoogt de kans op interne conflicten en vertragingen, en leidt uiteindelijk tot een verhoogd risico op het maken van suboptimale technologische keuzes die de voortgang en het succes van het project belemmeren.

## **4.2. Interviews met experts**

TODO

Binnen deze interviews worden projectmanagers van Apvine bevraagd over hun

huidige besluitvormingsproces bij het selecteren van technologieën. Apvine is een bedrijf dat zich hoofdzakelijk richt op low-code ontwikkeling, waardoor de keuze voor een low-code platform vaak als uitgangspunt wordt genomen in plaats van als open vraag. Deze commerciële en strategische voorkeur kan een significante invloed uitoefenen op de manier waarop projectmanagers technologische keuzes benaderen, en op hoe zij hun beslissingen motiveren tegenover klanten zoals FOD MOB.

Er wordt in de interviews expliciet gefocust op zowel de expliciete als impliciete criteria die zij hanteren bij het maken van een keuze tussen low-code en high-code trajecten. In de praktijk blijkt dat deze afweging zelden volledig neutraal wordt gemaakt: projectmanagers proberen vaak de voordelen van low-code te benadrukken en mogelijke beperkingen ervan te relativeren om het platform te positioneren als een geschikte oplossing. Hierbij is het interessant om te onderzoeken in welke mate risico-inschatting, technische haalbaarheid en klantverwachtingen werkelijk een rol spelen in hun besluitvorming, of eerder achteraf worden gebruikt om een reeds gemaakte keuze te onderbouwen.

In de interviews wordt nagegaan hoe projectmanagers omgaan met de inherente voorkeur voor low-code binnen Apvine, en hoe dit hun technologische keuzes beïnvloedt. Aangezien low-code ontwikkeling een fundamenteel onderdeel vormt van de bedrijfsstrategie, is het interessant om te verkennen in hoeverre projectmanagers ruimte ervaren om alternatieven te overwegen wanneer de klantnoden of projectvereisten daar mogelijk om vragen. Daarnaast wordt onderzocht hoe zij omgaan met situaties waarin low-code minder geschikt blijkt, en of er intern ruimte bestaat om de beperkingen van het platform te erkennen of dat men voornamelijk inzet op het verdedigen van de gekozen oplossing.

Vragen voor deze interviews:

- Welke stappen doorloopt u typisch bij het maken van een technologiekeuze voor een nieuw project?
- Welke criteria weegt u het zwaarst bij de keuze tussen low-code en high-code?
- In hoeverre is uw beslissing gebaseerd op objectieve data versus ervaring of intuïtie?
- Worden deze criteria vastgelegd of zijn ze vooral impliciet aanwezig?
- Hoe gaat u om met klanten die een voorkeur hebben voor high-code of sceptisch staan tegenover low-code?
- Kunt u een voorbeeld geven van een project waarbij low-code achteraf gezien niet de beste keuze bleek?

Daarnaast delen solution- en softwarearchitecten hun ervaringen met technologiekeuzes. Zij geven inzicht in de doorslaggevende technische overwegingen, zoals

schaalbaarheid, integratiemogelijkheden en lange termijn onderhoud, die zij meenemen bij het adviseren over platformkeuzes binnen de context van FOD MOB-projecten.

Ook pre-sales consultants van Apvine worden geïnterviewd, met de focus op hun aanpak tijdens de offerte- en analysefase. Zij lichten toe hoe zij per project inschatten welk ontwikkelplatform het meest geschikt is, welke informatie ze hiervoor nodig hebben van de klant (zoals FOD MOB), en welke aannames hierin een rol spelen. Tot slot worden ontwikkelaars van Apvine zelf geïnterviewd over de praktische implicaties van gemaakte technologiekeuzes. Hierbij komt vooral de kloof tussen verwachtingen en realiteit aan bod, zoals technische beperkingen die pas tijdens de implementatie duidelijk worden, of de noodzaak om onderdelen alsnog in high-code te herschrijven vanwege beperkingen van het gekozen low-code platform.

### **4.3. Documentatieonderzoek**

Uit de analyse van de interne documentatie blijkt dat de keuze tussen low-code en high-code ontwikkeling momenteel niet is verankerd in duidelijke en formeel vastgelegde richtlijnen. In plaats daarvan lijkt deze beslissing grotendeels te worden genomen op basis van intuïtie, eerdere ervaringen en de directe beschikbaarheid van zowel interne als externe ontwikkelteams. Hoewel er in documenten zoals pre-sales offertes, kick-off verslagen en architectuurbesluiten wel verwijzingen zijn naar technologiekeuzes, ontbreekt een gestructureerd afwegingskader dat systematisch wordt toegepast. Hierdoor ontstaat het risico dat keuzes niet altijd consistent of onderbouwd zijn, en dat belangrijke contextuele factoren zoals schaalbaarheid, onderhoudbaarheid of klantbehoeften, onvoldoende worden meegewogen in het besluitvormingsproces.

Een bijkomende observatie is dat de mate van personalisatie van de applicatie eveneens een doorslaggevende rol speelt in de keuze voor de ontwikkelmethode. Wanneer een diep gepersonaliseerde oplossing vereist is, wordt vaker gekozen voor het interne Java-team, vanwege hun mogelijkheid om maatwerk te leveren. Daarentegen wordt bij projecten met een strakke deadline of een minder complexe functionaliteitsbehoefte vaak gekozen voor een low-code platform, omdat hiermee sneller resultaat kan worden geboekt. Deze pragmatische aanpak is begrijpelijk, maar onderstreept tegelijkertijd het ontbreken van expliciete richtlijnen die dergelijke afwegingen structureren.

# 5

## Hoe kan het beter?

### 5.1. Literatuuroverzicht

In aanvulling op de eerder besproken stand van zaken biedt dit literatuuroverzicht een verdiepend inzicht in het gebruik van Mendix als low-code platform op basis van bestaande literatuur, praktijkcases en academisch onderzoek. Het platform, dat bekend staat om zijn visuele ontwikkelomgeving en ondersteuning voor zowel professionele als niet-technische gebruikers, wordt steeds vaker ingezet voor snelle en iteratieve applicatieontwikkeling (Alamin e.a., [2021](#)). Daarnaast zijn er diverse praktijkvoorbeelden en academische analyses die de voordelen en beperkingen van low-code ontwikkeling, en Mendix in het bijzonder, verder belichten (Gadia, [2025](#)).

#### 5.1.1. Praktijkvoorbeelden en Casestudies

Diverse organisaties hebben Mendix succesvol ingezet voor uiteenlopende toepassingen:

- **Gemeente Rotterdam**

Door het gebruik van Mendix heeft de gemeente meer dan 100 low-code applicaties ontwikkeld, waaronder een COVID-responsportaal en een parkeervergunning-app. Deze applicaties ondersteunen meer dan 100.000 gebruikers en zijn gemiddeld binnen 8-12 weken ontwikkeld (Mendix, [2025a](#)).

- **PostNL**

Het nationale postbedrijf van Nederland heeft zijn ordermanagementsysteem herbouwd met Mendix, gebruikmakend van een microservices-architectuur. Hierdoor kunnen ze meer dan 1 miljoen pakketten per dag verwerken en hebben ze een achterstand van twee jaar in zes maanden weggewerkt (Zaman, [2024](#)).

- **AZL**

Deze pensioenfondsbeheerder heeft met Mendix een klantportaal ontwikkeld dat het pensioenkeuzeproces voor deelnemers vereenvoudigt. Door de overstap naar een agile werkwijze in combinatie met low-code ontwikkeling, konden ze sneller inspelen op klantbehoeften en de gebruikerservaring verbeteren (Mendix, [z.d.](#)).

### 5.1.2. Ervaringen en Knelpunten uit Ontwikkelaarsgemeenschappen

Onderzoek naar discussies op platforms zoals Stack Overflow en Reddit onthult dat ontwikkelaars verschillende uitdagingen ervaren bij het gebruik van low-code platforms:

- **Aanpassingsmogelijkheden**

Veel ontwikkelaars ondervinden moeilijkheden bij het aanpassen van gebruikersinterfaces en het implementeren van dynamische event-handling in low-code omgevingen. Een studie toonde aan dat meer dan 40% van de vragen op Stack Overflow betrekking had op dergelijke aanpassingen, waarbij een significant deel onbeantwoord bleef (Alamin e.a., [2021](#)).

- **Integratie met externe systemen**

Hoewel platforms als Mendix en OutSystems integratiemogelijkheden bieden, ervaren ontwikkelaars soms beperkingen bij het koppelen van specifieke externe systemen of bij het beheren van complexe dataflows (Gadia, [2025](#)).

### 5.1.3. Academische Inzichten in Low-Code Ontwikkeling

Academisch onderzoek benadrukt zowel de voordelen als de uitdagingen van low-code ontwikkeling:

- **Versnelling van ontwikkelprocessen**

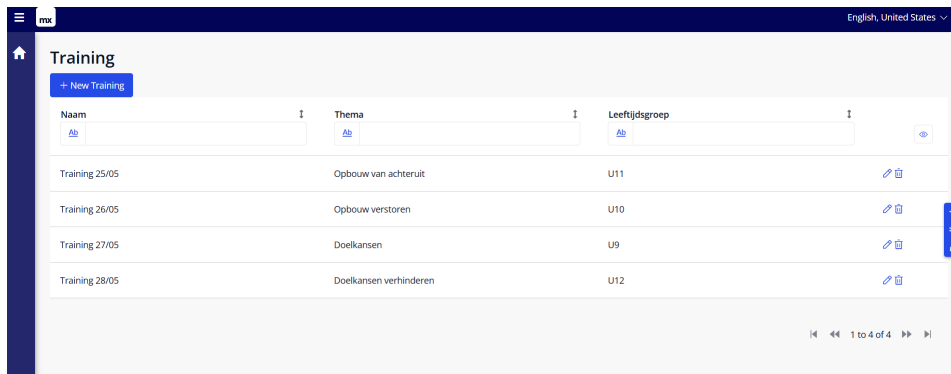
Low-code platforms stellen organisaties in staat om sneller applicaties te ontwikkelen, wat met name voordelig is in domeinen die behoefte hebben aan geautomatiseerde processen en workflows (Luo e.a., [2021](#)).

- **Beperkingen in complexiteit**

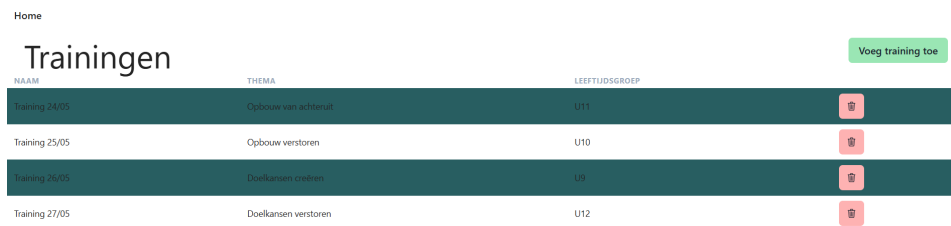
Hoewel low-code geschikt is voor veel toepassingen, kunnen er beperkingen zijn bij het ontwikkelen van zeer complexe of gespecialiseerde applicaties, waarbij traditionele programmeertalen meer flexibiliteit bieden.

## 5.2. Praktisch onderzoek

In dit praktisch onderzoek worden twee applicaties met elkaar vergeleken: één ontwikkeld met het low-code platform Mendix en één ontwikkeld met JavaScript en React.



**Figuur 5.1:** Homepage Mendix applicatie



**Figuur 5.2:** Homepage JavaScript/React applicatie

Het doel van deze vergelijking is om inzicht te krijgen in de sterktes en beperkingen van beide technologieën op basis van praktijkgerichte criteria. De evaluatie richt zich op zes hoofdgebieden: prestatie en schaalbaarheid, integratiemogelijkheden, aanpasbaarheid en uitbreidbaarheid, ontwikkelsnelheid, beveiliging en compliance, en de algehele gebruikerservaring. Voor elk van deze categorieën worden gerichte testen uitgevoerd, variërend van technische benchmarks (zoals laadtijd en load testing) tot ontwikkelervaring en gebruiksvriendelijkheid. Door beide applicaties op systematische wijze te testen, wordt duidelijk in welke scenario's Mendix of juist een traditionele JavaScript/React-stack de voorkeur verdient. De resultaten uit deze vergelijking vormen de basis voor de uiteindelijke conclusie over de toepasbaarheid van beide benaderingen binnen diverse zakelijke contexten.

### 5.2.1. Schaalbaarheid en prestaties

Om een objectieve vergelijking te kunnen maken tussen de in Mendix en in JavaScript/React ontwikkelde applicaties, zijn verschillende prestatie- en schaalbaarheidstesten uitgevoerd. Deze testen richten zich op drie kernaspecten: de laadtijd van de applicatie, de reactietijd op gebruikersacties en de prestaties onder belas-

ting. Voor het meten van de laadtijd zijn tools zoals Google Lighthouse en Chrome DevTools gebruikt, waarmee onder andere de **First Contentful Paint (FCP)** en de **Largest Contentful Paint (LCP)** zijn geanalyseerd. Daarnaast is de reactietijd op interacties met de gebruikersinterface gemeten om de responsiviteit van beide applicaties te beoordelen. Tot slot wordt een load test uitgevoerd met tools als Apache JMeter en k6 om inzicht te krijgen in de schaalbaarheid en het gedrag van de applicaties bij een hoge gebruikersbelasting. Deze metingen geven een duidelijk beeld van de praktische prestaties van beide technologieën in een realistische gebruiksomgeving.

### Laadtijd (Performance)

De laadtijd van een webapplicatie is een cruciale factor voor de gebruikerservaring, aangezien gebruikers doorgaans verwachten dat een pagina binnen 2 à 3 seconden volledig geladen is. Om de prestaties van beide applicaties objectief te meten, werd gebruikgemaakt van Google Lighthouse. Deze tool analyseert verschillende web performance-indicatoren, waaronder:

- **FCP**
- **LCP**
- **Total Blocking Time (TBT)**
- **Speed Index**
- **Cumulative Layout Shift (CLS)**
- **Performance Score** (algemene score op 100)

### Resultaten

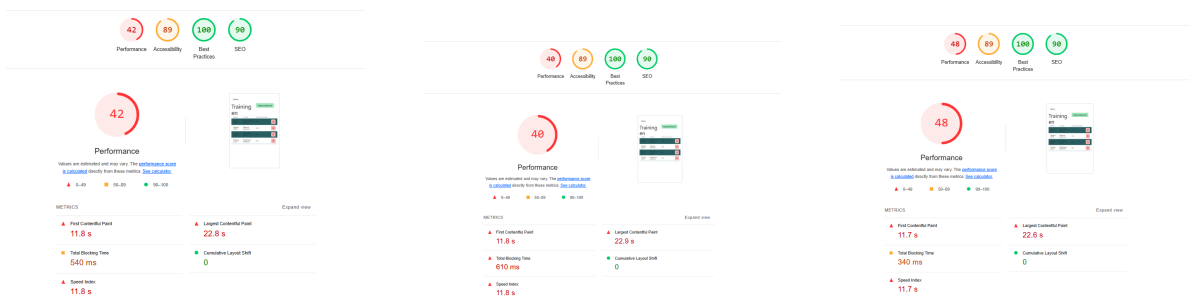
Voor zowel de Mendix-applicatie als de JavaScript/React-applicatie werden telkens drie afzonderlijke metingen uitgevoerd. De resultaten van deze testen zijn gemiddeld om zo een eerlijke vergelijking te maken. In onderstaande tabel worden de gemiddelde waarden van de prestaties per platform weergegeven, inclusief een kolom met het rekenkundig gemiddelde tussen de twee platformen ter referentie.



## JavaScript/React resultaten laadtijd

Metriek	Test 1	Test 2	Test 3	Gemiddelde
<b>FCP</b>	11.8s	11.8s	11.7s	11.8s
<b>LCP</b>	22.8s	22.9s	22.6s	22.8s
<b>TBT</b>	540ms	610ms	340ms	497ms
<b>Speed Index</b>	11.8s	11.8s	11.7s	11.8s
<b>CLS</b>	0	0	0	0
<b>Performance Score</b>	42	40	48	43

Tabel 5.1: Testresultaten JavaScript/React.

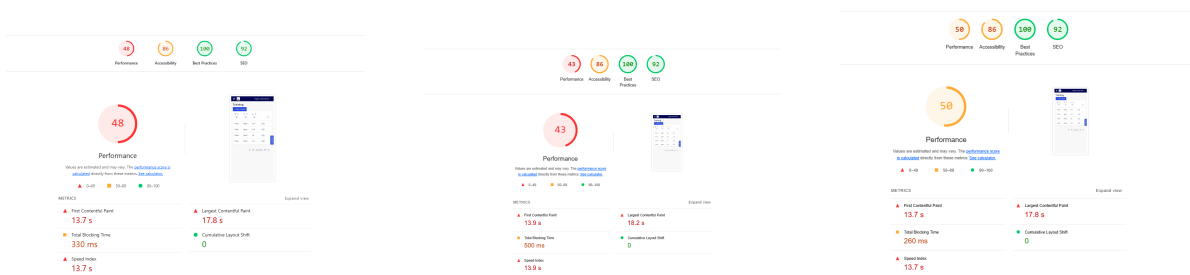


Figuur 5.3: Lighthouse testresultaten JavaScript/React - Links: Test 1, Midden: Test 2, Rechts: Test 3

## Mendix resultaten laadtijd

Metriek	Test 1	Test 2	Test 3	Gemiddelde
<b>FCP</b>	13.7s	13.9s	13.7s	13.8s
<b>LCP</b>	17.8s	18.2s	17.8s	17.9s
<b>TBT</b>	330ms	500ms	260ms	363ms
<b>Speed Index</b>	13.7s	13.9s	13.7s	13.8s
<b>CLS</b>	0	0	0	0
<b>Performance Score</b>	48	43	50	47

Tabel 5.2: Testresultaten Mendix



Figuur 5.4: Lighthouse testresultaten Mendix - Links: Test 1, Midden: Test 2, Rechts: Test 3

### Vergelijkingstabel

Metriek	JavaScript/ React	Mendix	Vershil	Beste Prestatie
<b>FCP</b>	11.8s	13.8s	-2.0s	JavaScript/React
<b>LCP</b>	22.8s	17.8s	+4.9s	Mendix
<b>TBT</b>	497msms	363ms	+134ms	Mendix
<b>Speed Index</b>	11.8s	13.8s	-2.0s	JavaScript/React
<b>CLS</b>	0	0	Gelijk	Gelijk
<b>Performance Score</b>	43	47	-4	Mendix

**Tabel 5.3:** Vergelijkingstabel laadtijd

### Samenvatting

- JavaScript/React is sneller bij **FCP** en Speed Index (2s sneller)
- Mendix presteert beter bij **LCP** (4,9s sneller) en **TBT** (134ms minder)
- Mendix heeft een hogere Performance Score (47 vs 43)
- Beide platforms hebben perfecte **CLS** scores
- Beide platforms scoren onder acceptabele performance niveaus (<50)

De prestatieverschillen tussen Mendix en JavaScript/React kunnen verklaard worden door fundamentele verschillen in architectuur en uitvoering. JavaScript/React scoort beter op **FCP** doordat het doorgaans gebruikmaakt van een lichte initiële bundel en client-side rendering, waardoor visuele elementen snel zichtbaar zijn. Mendix presteert daarentegen beter op **LCP** en **TBT** dankzij de geïntegreerde backend, geoptimaliseerde dataverwerking en beperkte client-side JavaScript. Door logica server-side af te handelen en automatisch platformoptimalisaties toe te passen, vermindert Mendix de belasting op de browser, wat resulteert in betere prestaties bij het laden van grote elementen en minder blokkeringen tijdens interacties.

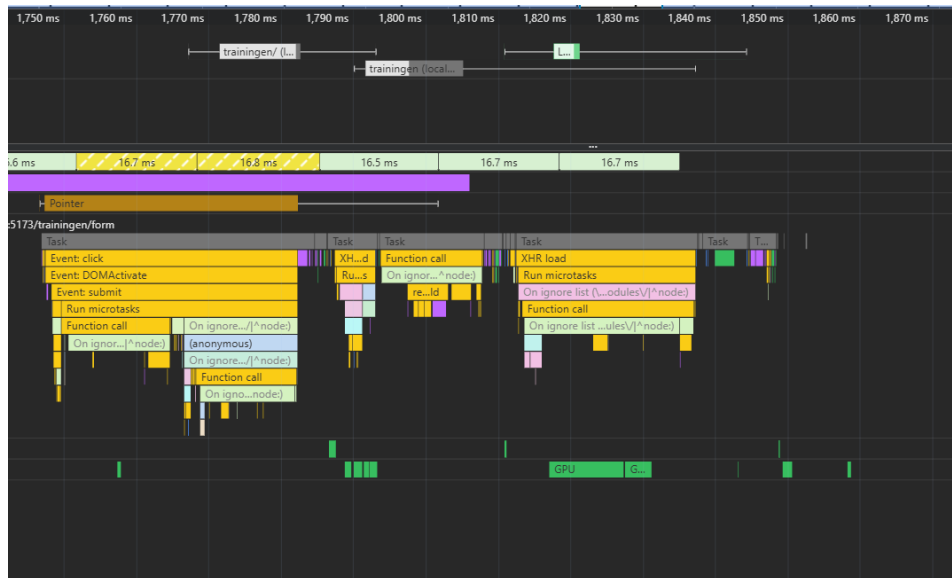
### Reactietijd op gebruikersacties

De reactietijd van een webapplicatie, oftewel de tijd tussen een gebruikersactie (zoals een klik) en het zichtbare resultaat, is essentieel voor een vloeiende gebruikerservaring. Gebruikers verwachten dat interacties vrijwel onmiddellijk reageren, idealiter binnen 100 tot 200 milliseconden. Om de reactietijd objectief te evalueren, werd een handmatige performance-analyse uitgevoerd met behulp van de ontwikkelaarstools in Google Chrome (Performance tab).

## Resultaten

Bij de testen werd specifiek gekeken naar het toevoegen van een nieuw item binnen zowel de Mendix- als de JavaScript/React-applicatie. Door de belangrijkste fasen van dit proces in kaart te brengen, van het moment waarop een gebruiker een actie uitvoert tot aan de visuele terugkoppeling op het scherm, kon een nauwkeurige vergelijking worden gemaakt van de snelheid en efficiëntie van beide implementaties.

### JavaScript/React resultaten reactietijd



**Figuur 5.5:** Testresultaten toevoegen object in JavaScript/React

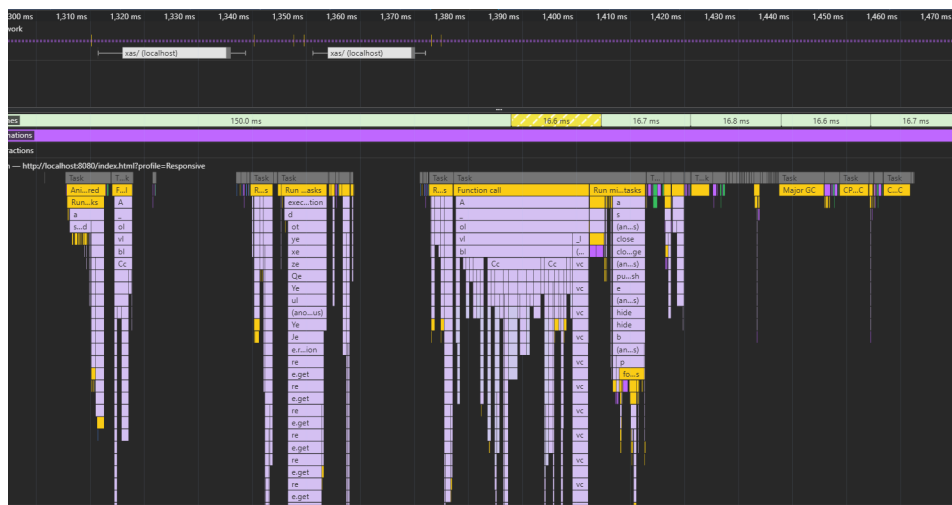
Metriek	Tijd/Duur	Beschrijving
<b>Start operatie/ Event Click</b>	~1.750 ms	Start van gebruikersinteractie
<b>Document Object Model (DOM) Activate</b>	~1.752 ms	Verwerking van DOM-event
<b>Event Submit</b>	~1.754 ms	Formulier wordt verstuurd
<b>Microtasks</b>	1.754 – 1.756 ms	Uitvoering van JavaScript-taken
<b>XMLHttpRequest (XHR) Load</b>	1.790 – 1.820 ms	Communicatie met de server
<b>Function Calls</b>	1.820 – 1.840 ms	Verwerking van de serverrespons
<b>GPU Rendering</b>	~1.850 ms	Visuele weergave (rendering)
<b>Reactietijd</b>	≈ 100 ms	Tijdsduur van klik tot weergave

Tabel 5.4: Testresultaten reactietijd JavaScript/React

Fase	Duur	Activiteit
<b>Eventverwerking</b>	~4 ms	Van klik tot submit
<b>XHR-verzoek</b>	~30 ms	Verzoek naar en antwoord van de server
<b>Responsverwerking</b>	~20 ms	Verwerking van ontvangen data
<b>Visuele rendering</b>	~46 ms	User Interface (UI) wordt bijgewerkt op het scherm

Tabel 5.5: Breakdown reactietijd JavaScript/React

## Mendix resultaten reactietijd



Figuur 5.6: Testresultaten toevoegen object in Mendix

Metriek	Tijd/Duur	Beschrijving
<b>Start operatie</b>	~1.300 ms	Start van gebruikersinteractie
<b>Animation/Render</b>	1.300 - 1.320 ms	UI animaties en rendering
<b>Run Tasks</b>	1.320 - 1.340 ms	Asynchrone task verwerking
<b>Function Calls</b>	1.340 - 1.380 ms	Core applicatie logica
<b>Major Garbage Collection (GC)</b>	1.790 – 1.820 ms	Communicatie met de server
<b>Function Calls</b>	~1.440 ms	Garbage collection
<b>Operatie Compleet</b>	~1.470 ms	Item volledig toegevoegd
<b>Totale reactietijd</b>	≈ 170 ms	Tijdsduur van klik tot weergave

Tabel 5.6: Testresultaten reactietijd Mendix

Fase	Duur	Activiteit
<b>Initial Animation</b>	~20 ms	UI feedback start
<b>Task Processing</b>	~20 ms	Asynchrone verwerking
<b>Function Execution</b>	~40 ms	Business logica
<b>Rendering and GC</b>	~90 ms	UI updates + cleanup

Tabel 5.7: Breakdown reactietijd Mendix

## Conclusie

Hoewel JavaScript met 100 ms iets sneller is dan Mendix met 170 ms van start tot completion, is het verschil van 70 ms relatief klein en valt het binnen een categorie van zeer responsieve prestaties. JavaScript profiteert van snellere verwerking, maar omvat ook een server roundtrip van ongeveer 30 ms via **XHR** calls, terwijl Mendix juist meer lokale verwerking en garbage collection uitvoert, wat zorgt voor extra framework overhead. Beide platformen tonen vergelijkbare tijden voor de kernlogica (50-60 ms), wat aangeeft dat het verschil vooral zit in de UI rendering en framework processing. Voor een kleine applicatie is deze lichte vertraging in Mendix een acceptabele trade-off, gezien de rijkere functionaliteit en gebruikerservaring die het biedt. Bij grotere en complexere applicaties kan de Mendix overhead echter toenemen, wat de prestaties meer beïnvloedt. Desondanks leveren beide technologieën een vrijwel directe respons die gebruikers als “instant” ervaren, waardoor deze realistische vergelijking een eerlijk beeld geeft van de praktische prestaties in echte toepassingen.

## Load test

Om de schaalbaarheid en robuustheid van de webapplicaties te evalueren onder verschillende belastingniveaus, werden ook loadtesten uitgevoerd. Deze testen simuleren een toenemend aantal gelijktijdige gebruikers om te bepalen hoe de applicaties presteren onder stress en om eventuele knelpunten te identificeren.

## Resultaten

De loadtesten zijn uitgevoerd door het simuleren van HTTP-verzoeken voor een oplopend aantal gebruikers (10, 100, 250, 500, 1000, 2500 gebruikers) voor zowel de Mendix- als de JavaScript/React-applicatie. Hierbij is gebruikgemaakt van Apache JMeter, een veelgebruikte tool voor het uitvoeren van prestatietests, waarmee realistische gebruikersinteracties kunnen worden nagebootst. De belangrijkste metingen die zijn geanalyseerd, omvatten de gemiddelde reactietijd, de doorvoer (throughput) en het foutpercentage (Error %).

## JavaScript/React resultaten load testen

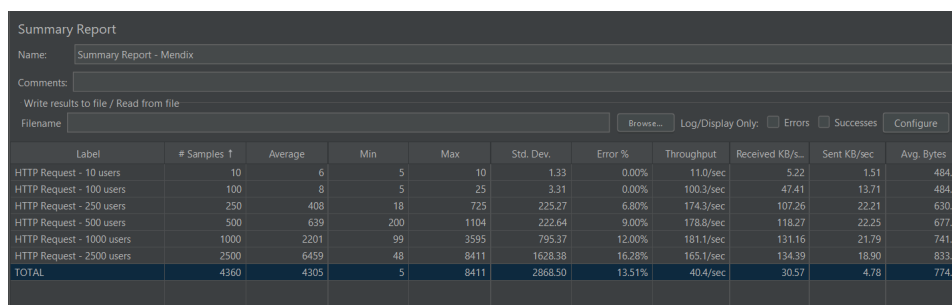
Summary Report										
Name:	Summary Report - JavaScript/React									
Comments:										
Write results to file / Read from file										
Filename										
	Browse... Log/Display Only: <input type="checkbox"/> Errors <input type="checkbox"/> Successes <input type="checkbox"/> Configure									
Label	# Samples ↑	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
HTTP Request - 10 users	10	8	7	9	0.63	0.00%	11.0/sec	11.85	1.40	1098.0
HTTP Request - 100 users	100	8	7	14	1.37	0.00%	100.3/sec	107.55	12.73	1098.0
HTTP Request - 250 users	250	536	44	760	206.76	0.00%	151.2/sec	162.17	19.20	1098.0
HTTP Request - 500 users	500	1537	226	2273	555.96	28.00%	153.5/sec	228.96	14.03	1527.2
HTTP Request - 1000 users	1000	2985	69	4536	1125.63	39.40%	161.3/sec	268.13	12.41	1702.0
HTTP Request - 2500 users	2500	7768	133	11736	2048.74	53.20%	143.9/sec	268.91	8.55	1913.6
TOTAL	4360	5346	7	11736	3329.76	42.75%	18.3/sec	31.33	1.33	1753.4

**Figuur 5.7:** Testresultaten load test: ophalen objecten in JavaScript/React

Aantal gelijktijdige gebruikers	Gemiddelde reactietijd (ms)	Error percentage (%)	Throughput (requests/sec)
10	8	0.00	11.0
100	8	0.00	100.3
250	536	0.00	151.2
500	1537	28.00	153.5
1000	2985	39.40	161.3
2500	7768	53.20	143.9

Tabel 5.8: Loadtestresultaten JavaScript/React: Kernmetriekeken

## Mendix resultaten load testen



The screenshot shows a 'Summary Report' window in Mendix. It includes fields for Name (Summary Report - Mendix), Comments, and a section to write results to a file. Below these is a table with the following data:

Label	# Samples ↑	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
HTTP Request - 10 users	10	6	5	10	1.33	0.00%	11.0/sec	5.22	1.51	484.0
HTTP Request - 100 users	100	8	5	25	3.31	0.00%	100.3/sec	47.41	13.71	484.0
HTTP Request - 250 users	250	408	18	725	225.27	6.80%	174.3/sec	107.26	22.21	630.0
HTTP Request - 500 users	500	639	200	1104	222.64	9.00%	178.8/sec	118.27	22.25	677.2
HTTP Request - 1000 users	1000	2201	99	3595	795.37	12.00%	181.1/sec	131.16	21.79	741.6
HTTP Request - 2500 users	2500	6459	48	8411	1628.38	16.28%	165.1/sec	134.39	18.90	833.5
TOTAL	4360	4305	5	8411	2868.50	13.51%	40.4/sec	30.57	4.78	774.0

Figuur 5.8: Testresultaten load test: ophalen objecten in Mendix

Aantal gelijktijdige gebruikers	Gemiddelde reactietijd (ms)	Error percentage (%)	Throughput (requests/sec)
10	6	0.00	11.0
100	8	0.00	100.3
250	408	6.80	174.3
500	639	9.00	178.8
1000	2201	12.00	181.1
2500	6459	16.28	165.1

Tabel 5.9: Loadtestresultaten Mendix: Kernmetriekeken

## Conclusie

Hoewel een JavaScript/React-applicatie theoretisch het potentieel heeft om Mendix te overtreffen in absolute prestaties, blijkt dit in de huidige load-testscenario's



niet het geval onder verhoogde belasting. Waar React flexibiliteit en volledige controle biedt voor optimalisatie, vereist dit ook aanzienlijke expertise en afstemming om daadwerkelijk robuust te blijven bij hoge gelijktijdige gebruikersaantallen. In de praktijk toont Mendix zich in deze tests stabiel, met name op het gebied van foutafhandeling en consistentie bij toenemende load. Dit is te danken aan de platformgebonden optimalisaties, standaardisatie en ingebouwde schaalbaarheidsmechanismen.

Beide technologieën hebben dus hun sterktes: Mendix excelleert in betrouwbaarheid en voorspelbaarheid bij grotere gebruikersvolumes, terwijl JavaScript/React – mits zorgvuldig geoptimaliseerd – de mogelijkheid biedt tot betere piekprestaties. De keuze hangt dan ook sterk af van de context en prioriteiten: snelle ontwikkeltrajecten met robuuste baseline-prestaties (Mendix) tegenover maximale flexibiliteit en performancepotentieel met meer technische investering (JavaScript/React). In de geteste scenario's levert Mendix momenteel het meest consistente gedrag onder druk, wat voor veel toepassingen een doorslaggevend voordeel kan zijn.

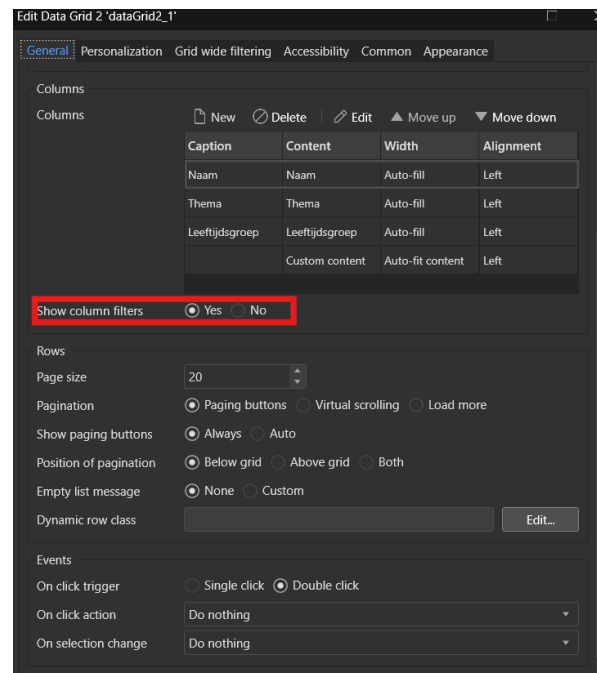
### 5.2.2. Ontwikkelingsnelheid en onderhoud

#### Ontwikkeltijd

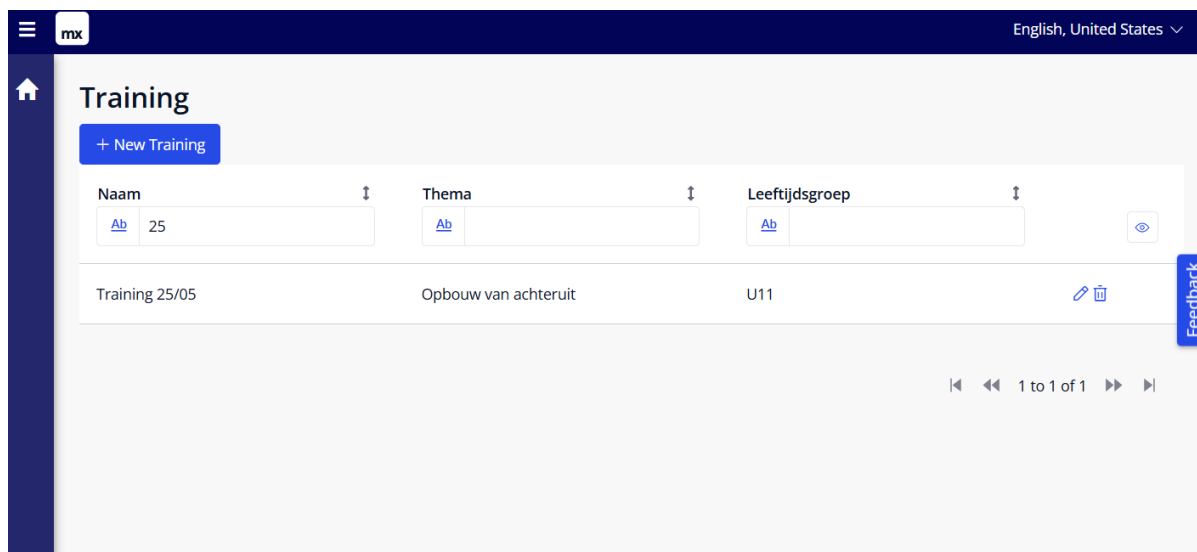
De ontwikkeltijd is handmatig bijgehouden voor beide applicaties om een realistische inschatting te geven van de benodigde inspanning. Voor de JavaScript/React-applicatie bedroeg de totale ontwikkeltijd ongeveer 5 uur, waarbij tijd werd besteed aan het opzetten van de ontwikkelomgeving, het schrijven van de logica, het afhandelen van API-calls, en het bouwen van de gebruikersinterface. Voor de Mendix-applicatie daarentegen was slechts 30 minuten nodig om een functioneel equivalent te realiseren. Deze aanzienlijke tijdswinst is te verklaren door het gebruik van vooraf gedefinieerde componenten, visuele modellering en standaardfunctionaliteit binnen het Mendix-platform. Hoewel JavaScript/React meer flexibiliteit en controle biedt, vereist het ook meer technische expertise en ontwikkelinspanning. Mendix blijkt in dit opzicht bijzonder efficiënt voor het snel realiseren van functionele prototypes of standaardapplicaties.

#### Herbruikbaarheid / uitbreidbaarheid

Om de aanpasbaarheid en herbruikbaarheid van componenten te evalueren, is bij beide applicaties een zoekfunctie toegevoegd. Dit biedt inzicht in hoe snel en flexibel nieuwe functionaliteit kan worden geïntegreerd. In Mendix kon de zoekfunctionaliteit in slechts 1 minuut worden geactiveerd door een bestaande zoekoptie in het component eenvoudig aan te zetten via de modeler-interface, een handeling die letterlijk neerkomt op het selecteren van een checkbox.



**Figuur 5.9:** Checkbox aanzetten filters



**Figuur 5.10:** Homepage met zoekbalk Mendix

In JavaScript/React duurde het toevoegen van een vergelijkbare zoekfunctie ongeveer 15 minuten, waarbij logica handmatig moest worden geschreven, de zoekinput geïntegreerd diende te worden in de UI, en filtering op de dataset geïmplementeerd werd.

```
export default function TrainingsTable( trainingen, onDelete ) {
  const [searchTerm, setSearchTerm] = useState(""); // Filter trainingen op basis
  van zoekterm
  const filteredTrainingen = trainingen.filter((training) => training.naam.toLowerCase().includes(searchTerm.toLowerCase()));
  if (trainingen.length === 0) return <div className="alert alert-info">Nog geen trainingen.</div>;
  return (
    <Box> /* Zoekbalk */
    <Input placeholder="Zoek op naam..." value=searchTerm onChange=(e) => setSearchTerm(e.target.value) mb=4 maxWidth="300px" />
    /* Tabel */
    <Table variant="striped" colorScheme="teal" size="sm">
      <Thead>
        <Tr>
          <Th>Naam</Th>
          <Th>Thema</Th>
          <Th>Leeftijdsgroep</Th>
        </Tr>
      </Thead>
      <Tbody>
        {filteredTrainingen.length > 0 ? (
          filteredTrainingen.map((training) => (
            <Training key=training.id onDelete=onDelete ...training />
          ))
        ) : null}
      </Tbody>
    </Table>
    </Box>
  );
}
```

**Codefragment 1:** Trainingstabel met zoekfunctie op naam



**Figuur 5.11:** Homepage met zoekbalk JavaScript/React

Hoewel de implementatie in beide gevallen vlot verliep, toont dit voorbeeld aan dat Mendix dankzij herbruikbare en vooraf geconfigureerde componenten bijzonder efficiënt is bij het uitbreiden van functionaliteit, terwijl JavaScript/React meer flexibiliteit biedt maar ook meer ontwikkelwerk vraagt.

### 5.2.3. Integratiemogelijkheden

Zowel Mendix als JavaScript/React bieden uitgebreide mogelijkheden om te integreren met externe systemen, maar de manier waarop deze integraties tot stand komen verschilt aanzienlijk vanwege hun aard: Mendix als low-code platform en JavaScript/React als code-first benadering.

#### Mendix

Mendix is ontworpen om eenvoudig te koppelen met zowel moderne als legacy systemen, met standaard ondersteuning voor verschillende integratietechnieken:

- **REST API's:** volledige ondersteuning voor het bouwen en consumeren van RESTful APIs. Integraties zoals met Salesforce of SAP zijn vlot op te zetten via visuele flows.
- **SOAP Webservices:** ondersteuning voor oudere protocollen blijft behouden, wat belangrijk is voor bedrijven met legacy infrastructuren.
- **GraphQL:** niet native ondersteund, maar via extensies of community-modules kunnen GraphQL-endpoints alsnog worden aangesproken.
- **OData Services:** Mendix biedt OData-integraties voor onder meer BI-tools zoals Power BI.
- **JDBC-databaseconnecties:** directe verbinding met relationele databases zoals SQL Server, MySQL, PostgreSQL en Oracle.

Daarnaast is er toegang tot Mendix Connect en de Mendix Marketplace, waar herbruikbare connectors beschikbaar zijn voor populaire systemen zoals SAP, Salesforce, Kafka en meer. Dit maakt het voor ontwikkelaars mogelijk om snel integraties op te zetten zonder complexe code.

### JavaScript/React

JavaScript/React biedt in essentie volledige vrijheid voor integratie, maar dit betekent ook dat ontwikkelaars zelf verantwoordelijk zijn voor de implementatie van alle benodigde logica en security rond die integraties:

- **REST en GraphQL:** beide worden uitstekend ondersteund via moderne bibliotheken zoals Axios, Fetch API, Apollo Client of Relay.
- **WebSocket en realtime communicatie:** makkelijk op te zetten met bibliotheken zoals Socket.IO.
- **SOAP:** mogelijk, maar vereist aparte libraries (zoals soap in Node.js), aangezien SOAP minder gangbaar is in moderne JavaScript-ecosystemen.
- **Directe databaseverbindingen:** in een typische React frontend niet van toepassing, maar in combinatie met een Node.js backend kunnen via ORM's (zoals Sequelize, Prisma) databases als MySQL, PostgreSQL of MongoDB worden aangesproken.

JavaScript/React biedt maximale controle, flexibiliteit en toegang tot een breed scala aan open-source libraries en SDK's. Wel vraagt het opzetten van integraties meer technische kennis en aandacht voor zaken als foutafhandeling, authenticatie (OAuth, JWT) en datastructurering.

## Vergelijking

Aspect	Mendix	JavaScript/React
<b>Complexiteit</b>	Laag, via visuele modelering en kant-en-klare connectors	Hoog, handmatig implementeren via code
<b>Ondersteunde protocollen</b>	REST, SOAP, OData, JDBC, GraphQL (indirect)	REST, GraphQL, WebSockets, SOAP (met lib)
<b>Herbruikbare integraties</b>	Beschikbaar via Mendix Marketplace	Zelf te beheren of via externe NPM packages
<b>Flexibiliteit</b>	Beperkt tot wat het platform ondersteunt	Zeer hoog, volledige controle over gedrag
<b>Snelheid van integratie</b>	Zeer snel (minuten) met visuele ondersteuning	Langzamer (uren), afhankelijk van de setup

**Tabel 5.10:** Integratiemogelijkheden

In essentie is Mendix zeer geschikt wanneer snelheid, standaardisatie en visuele eenvoud voorop staan. JavaScript/React is de juiste keuze wanneer maximale flexibiliteit, diepgaande aanpassing en custom integraties vereist zijn — mits de nodige technische expertise aanwezig is.

### 5.2.4. Aanpasbaarheid en uitbreidbaarheid

Het Mendix-platform combineert de snelheid van low-code ontwikkeling met voldoende flexibiliteit voor geavanceerde uitbreidingen. Standaard biedt Mendix een brede waaier aan voorgeprogrammeerde blokken en componenten, zoals widgets, datakoppelingen, validatieacties en gebruikersinterface-elementen, die eenvoudig via drag-and-drop in de applicatie kunnen worden geïntegreerd. Deze componenten versnellen het ontwikkelproces aanzienlijk en vereisen minimale configuratie. Toch blijft het platform niet beperkt tot wat standaard beschikbaar is. Voor situaties waarin maatwerk vereist is, biedt Mendix meerdere uitbreidingsmogelijkheden:

- **Java-acties:** voor het uitvoeren van complexe server-side logica kunnen aangepaste Java-methoden worden ingebouwd.
- **Pluggable widgets:** ontwikkelaars kunnen hun eigen UI-componenten maken in JavaScript of React, die vervolgens naadloos integreren met het Mendix-model.
- **Mendix Model SDK:** hiermee kunnen ontwikkelaars programmatic wijzigingen aanbrengen in het domeinmodel of app-logica, bijvoorbeeld voor code-generatie of CI/CD-integratie.

- **Extensibility API:** maakt het mogelijk om Studio Pro zelf uit te breiden met nieuwe functionaliteit, wat handig is voor grotere ontwikkelteams of platformontwikkeling.

In vergelijking met JavaScript/React biedt Mendix een meer gestandaardiseerde aanpak, waarbij veelgebruikte patronen al zijn voorgedefinieerd. Dit maakt het zeer efficiënt voor snelle ontwikkeling en eenvoudige aanpassingen. JavaScript/React daarentegen biedt volledige vrijheid en maatwerk vanaf nul, maar vereist meer handmatige inspanning en technische kennis voor uitbreidbaarheid.

Mendix positioneert zich daarmee als een platform dat zowel toegankelijk is voor snelle ontwikkeling via visuele bouwblokken, als krachtig genoeg om uit te breiden wanneer standaardcomponenten niet volstaan, zonder de ontwikkelaar volledig vast te zetten binnen het framework.

### 5.2.5. Beveiliging en compliance

Beveiliging en compliance zijn cruciale aspecten bij de ontwikkeling van moderne toepassingen, en zowel Mendix als JavaScript/React bieden hier oplossingen voor wel op verschillende manieren.

#### Mendix

Mendix hecht veel waarde aan beveiliging en biedt een breed scala aan ingebouwde beveiligingsmechanismen. Standaard voorziet het platform in:

- **Rollen- en toegangsbeheer op applicatie- en entiteitsniveau**, configureerbaar via de modeler.
- **Gegevensversleuteling** zowel in rust als tijdens transport.
- **Authenticatie-integratie** met SSO, OAuth 2.0, SAML en Active Directory.
- **Automatische beveiligingsupdates** en naleving van standaarden zoals ISO 27001, SOC 2 en GDPR.

Doordat veel beveiligingsmaatregelen standaard in het platform zitten, hoeven ontwikkelaars zich minder zorgen te maken over implementatiedetails. Daarnaast ondersteunt Mendix opslag in beheerde of eigen SQL-databases (zoals PostgreSQL, MS SQL of Oracle), wat organisaties flexibiliteit biedt in gegevensbeheer en compliance.

#### JavaScript/React

In een JavaScript/React-omgeving ligt de verantwoordelijkheid voor beveiliging volledig bij het ontwikkelingsteam. React zelf biedt geen beveiligingsfunctionaliteit; beveiliging moet worden geïmplementeerd via aanvullende tools en best practices:

- **Toegangscontrole en authenticatie** worden vaak geregeld via third-party services (zoals Auth0, Firebase Auth) of via backend-implementaties (JWT, OAuth).
- **Encryptie en veilige opslag** moeten handmatig worden geconfigureerd op server- en transportniveau (bv. HTTPS, TLS, encryptie van gevoelige data in databases).
- **Veiligheidsrisico's zoals XSS en CSRF** moeten proactief worden gemitigeerd via frameworks of headers (CSP, SameSite cookies, sanitization).
- **Compliance** met regelgeving (zoals GDPR) vereist eigen implementatie van dataverwerking, audit logging en toestemmingsbeheer.

Hoewel React dus maximale controle biedt, vraagt het ook een aanzienlijke investering in kennis en tooling om een vergelijkbaar beveiligingsniveau als bij Mendix te bereiken.

### Vergelijking

Mendix biedt een zeer veilig ecosysteem met veel standaardmaatregelen die de ontwikkelaar ontlasten. Voor organisaties met strikte compliance-eisen of beperkte beveiligingsexpertise is dit een groot voordeel. JavaScript/React daarentegen biedt volledige vrijheid en controle, maar vereist een zorgvuldige en goed onderbouwde aanpak om dezelfde mate van beveiliging en compliance te realiseren.

#### 5.2.6. Samenvattende conclusie praktisch onderzoek

Uit het praktisch onderzoek blijkt dat zowel Mendix als JavaScript/React sterke troeven bieden voor de ontwikkeling van moderne webapplicaties, maar dat hun inzetbaarheid sterk afhankelijk is van de context en vereisten van het project.

Qua integratiemogelijkheden toont Mendix zich als een krachtig low-code platform met standaard ondersteuning voor veelgebruikte protocollen zoals REST, SOAP en OData. De beschikbaarheid van kant-en-klare connectors via de Mendix Marketplace zorgt voor snelle, visuele integratie met externe systemen. JavaScript/React daarentegen biedt maximale flexibiliteit en controle bij het integreren met externe services, maar vereist meer technische expertise en handmatige implementatie.

Op vlak van aanpasbaarheid en uitbreidbaarheid scoort Mendix hoog met zijn plug-gable widgets, Java-acties en Model SDK, wat toelaat om standaardcomponenten uit te breiden wanneer nodig. Toch blijft JavaScript/React onovertroffen in maatwerk en volledige controle over zowel front- als backendlogica, zij het met een hogere ontwikkelkost.

Wat beveiliging en compliance betreft, biedt Mendix een veilig ecosysteem met veel ingebouwde maatregelen en certificeringen, wat het platform bijzonder geschikt maakt voor organisaties met hoge compliance-eisen of beperkte beveiligingsexpertise. JavaScript/React vereist daarentegen een meer proactieve aanpak van het ontwikkelingsteam om dezelfde beveiligingsstandaarden te behalen.

Samengevat is Mendix ideaal voor projecten waarbij snelheid, standaardisatie en visuele ontwikkeling centraal staan, terwijl JavaScript/React de voorkeur geniet in situaties waar volledige controle, flexibiliteit en diepgaande technische aanpassing vereist zijn. De keuze tussen beide technologieën is dus geen kwestie van beter of slechter, maar eerder van passend bij het juiste type project en ontwikkelteam.

### **5.3. Reflectie op eigen ervaringen**

Op basis van mijn ervaring kan ik bevestigen dat low-code, zoals Mendix, bijzonder krachtig is voor het snel opzetten van generieke applicaties met standaardfunctionaliteiten. Het stelt je in staat om in korte tijd werkende oplossingen te bouwen, wat vooral in iteratieve of proof-of-concept contexten een grote meerwaarde biedt. Tegelijk merk ik dat zodra een project meer 'custom' noden heeft – zoals complexe bedrijfslogica of verfijnde integraties – de grenzen van het platform sneller voelbaar worden. In die gevallen is het een duidelijke troef om een high-code achtergrond te hebben: je begrijpt beter wat er onder de motorkap gebeurt, kunt gerichter zoeken naar workarounds en neemt bewuster beslissingen over de architectuur van je oplossing. Bovendien zie ik dat een traditionele programmeerachtergrond ook de leesbaarheid en structuur van je low-code logica ten goede komt. Je denkt in patronen, zorgt voor herbruikbaarheid en hanteert best practices die niet vanzelfsprekend zijn in een puur visuele ontwikkelomgeving.

#### **5.3.1. Reflectie op ervaringen van experts**

Er zijn ook enkele ontwikkelaars met een klassieke high-code achtergrond die inmiddels volledig actief zijn binnen low-code projecten, met name op het Mendix-platform. Ik sprak met hen over hun ervaringen en vatte hun bevindingen samen. Ze benadrukken dat low-code bijzonder krachtig is voor het snel ontwikkelen van generieke applicaties met standaardfunctionaliteiten. Dit maakt het ideaal voor situaties waarin snelheid en iteratieve ontwikkeling belangrijk zijn. Daarnaast wordt low-code vaak gepositioneerd als een brug tussen IT en business, doordat ook gebruikers zonder programmeerervaring relatief snel aan de slag kunnen. In de praktijk leidde dit er echter soms toe dat businessgebruikers eigen applicaties opstartten die later door ervaren ontwikkelaars moesten worden overgenomen. Deze overdracht bleek niet altijd evident: de onderliggende logica bleek vaak moeilijk leesbaar en voldeed zelden aan gangbare ontwikkelstandaarden of best practices, wat extra werk met zich meebracht om de applicatie te stabiliseren en verder te ontwikkelen.

Wat betreft de ontwikkelervaring binnen Mendix, werd er gemengd gereageerd op de version control-functionaliteit. Hoewel het systeem in principe krachtig is en goed integreert met teamwerk, kunnen foutmeldingen en merge-conflicten soms moeilijk te doorgronden zijn. Wanneer alles echter correct functioneert, biedt het versiebeheer een betrouwbare en efficiënte manier van samenwerken. De in-



tegratie van agile werkmethodeken binnen het Mendix-platform werd unaniem positief beoordeeld: user stories, sprints en voortgang kunnen rechtstreeks via de projectpagina opgevolgd en beheerd worden, wat de samenwerking tussen ontwikkelaars en stakeholders vergemakkelijkt.

Ook het gebruik van herbruikbare modules uit de Mendix Marketplace werd als een groot voordeel genoemd. Het toevoegen van bestaande componenten versnelt de ontwikkeling aanzienlijk en voorkomt dat het wiel telkens opnieuw moet worden uitgevonden. Tegelijk wordt opgemerkt dat een groot aantal van deze modules weinig tot geen documentatie bevat, waardoor het tijd kost om hun werking te doorgronden of aan te passen aan specifieke projectbehoeften.

Over het geheel genomen beschouwen deze ontwikkelaars Mendix als een toegankelijke en efficiënte ontwikkelomgeving, die eenvoudig aanvoelt in de basis, maar bij complexere noden ook de nodige technische diepgang vereist. Hun ervaring met high-code vormt daarbij een duidelijke meerwaarde: het helpt hen om concepten sneller te begrijpen, kwalitatieve oplossingen te bouwen en de leesbaarheid en onderhoudbaarheid van hun low-code toepassingen te verbeteren.

#### **5.4. Evaluatie van Mendix-uitbreidingen**

Op basis van gesprekken met ervaren ontwikkelaars die dagelijks met Mendix werken, blijkt dat het gebruik van standaard Marketplace-modules doorgaans als efficiënt en tijdbesparend wordt ervaren, vooral bij generieke functionaliteiten. Deze modules kunnen snel geïntegreerd worden, wat de implementatietijd aanzienlijk verkort. Toch werd ook opgemerkt dat veel van deze modules onvoldoende of zelfs geheel geen documentatie bevatten. Dit gebrek aan transparantie leidt tot vertragingen tijdens implementatie en beperkt de flexibiliteit wanneer aanpassingen nodig zijn. In zulke gevallen biedt custom ontwikkeling vaak meer controle en beter afgestemde oplossingen, hoewel dit uiteraard gepaard gaat met een langere ontwikkeltijd en hogere initiële kosten.

Java-uitbreidingen binnen Mendix worden door ontwikkelaars met een high-code achtergrond beschouwd als waardevolle tools om de beperkingen van het platform te omzeilen, bijvoorbeeld bij complexe logica of integraties met externe systemen. Deze uitbreidingen verhogen echter ook de technische complexiteit van het project en kunnen de onderhoudbaarheid op lange termijn negatief beïnvloeden, zeker wanneer ze niet goed gedocumenteerd zijn of slechts door een beperkte groep binnen het team begrepen worden.

Wat betreft de kosten-batenverhouding, geven ontwikkelaars aan dat low-code in combinatie met herbruikbare modules initieel voordeliger is in zowel ontwikkeling als beheer. Naarmate de applicatie complexer wordt en meer 'maatwerk' vereist, verschuift dit evenwicht echter, en kunnen custom uitbreidingen of Java-integraties duurder uitvallen, zowel in termen van ontwikkeluren als bij latere aanpassingen of onderhoud.

Tot slot werd ook het integreren van externe systemen als een uitdaging benoemd. Hoewel Mendix hier voldoende ondersteuning voor biedt, vergt het combineren van low-code met externe services een goed begrip van beide kanten. Een technische achtergrond blijkt in die context bijzonder waardevol, zowel voor het begrijpen van de externe API's als voor het bouwen van robuuste, onderhoudbare koppelingen binnen het platform. Al met al tonen deze bevindingen aan dat een doordachte strategie, met oog voor schaalbaarheid en onderhoud, essentieel is bij het uitbreiden van Mendix-toepassingen.

## **5.5. Ontwikkeling beslissingskader**

### **5.5.1. 1. Cijfermatige evaluatiematrix**

Voor een snelle vergelijking tussen Mendix en traditionele ontwikkeling is in Tabel ?? een scorematrix opgenomen. Per dimensie is een score van 1 tot 5 toegekend, waarbij 5 duidt op een sterke geschiktheid van de technologie in die context. Deze matrix ondersteunt de eerste oriëntatie in het besluitvormingsproces.

Dimensie	Vraag	Score (1-5)	Toelichting
<b>Tijd &amp; Scope</b>	Moet er binnen enkele weken een MVP live zijn?		1 = Ja, 5 = Nee
	Is het project iteratief/veranderlijk (agile, MVP-gedreven)?		1 = Sterk veranderlijk, 5 = Vast ontwerp
<b>Technische Vereisten</b>	Is de businesslogica complex of sterk veranderlijk?		1 = Simpel, 5 = Complex/custom algoritmes
	Zijn real-time functionaliteiten vereist (sockets, streaming)?		1 = Nee, 5 = Ja
<b>Teamcapaciteit</b>	Zijn er voldoende ervaren developers beschikbaar (JS/Node/etc.)?		1 = Nee, 5 = Ja
	Zijn er citizen developers of functioneel beheerders betrokken?		1 = Ja, 5 = Nee
<b>Integratiebehoefte</b>	Betreft het veel standaard systemen (SAP, Salesforce)?		1 = Ja, 5 = Nee
	Gaat het om non-standaard of complexe API-integraties?		1 = Nee, 5 = Ja
<b>UX/Design</b>	Moet de UI sterk afwijken van standaardcomponenten?		1 = Nee, 5 = Ja
	Is er nauwe samenwerking nodig met UX-designers of branding-teams?		1 = Beperkt, 5 = Intensief
<b>Beveiliging &amp; Compliance</b>	Zijn er strikte eisen (GDPR, ISO, interne audits)?		1 = Ja, 5 = Nee
	Wordt gevoelige of gereguleerde data verwerkt?		1 = Ja, 5 = Nee

Tabel 5.11: Beoordelingscriteria per dimensie

### Interpretatie van de score

De totaalscore op basis van de beoordelingsmatrix uit Sectie 2 biedt richting bij de keuze van ontwikkelmethode:

- **10–20 punten: Mendix aanbevolen**

Geschikt voor projecten die vragen om snelheid, visueel modelleren, standaardisatie en minder technische complexiteit.

- **21–34 punten: Hybride oplossing overwegen**

Combineer Mendix met traditionele ontwikkeling, bijvoorbeeld Mendix voor workflows en beheerfuncties, en traditionele technologieën voor UI of complexe businesslogica.

- **35–50 punten: Traditionele ontwikkeling aanbevolen**

Vereist maximale flexibiliteit, diepgaande technische implementaties of specifieke designvereisten die moeilijk in low-code te realiseren zijn.

#### Let op:

Bij een hybride of twijfelgeval speelt het teamvertrouwen een belangrijke rol. Als een team meer ervaring of vertrouwen heeft in één van beide technologieën, verdient het de voorkeur om die richting te volgen, mits dit past binnen de technische kaders van het project.

De matrix fungeert als sneltoets, maar dient aangevuld te worden met projectcontext en nuance. Hiervoor zijn verdiepende beoordelvragen opgenomen in de volgende paragraaf.

### 5.5.2. 2. Beoordelingscriteria per dimensie

Voor elk project kunnen onderstaande vragen per dimensie als leidraad dienen bij het maken van een weloverwogen keuze...

### 5.5.3. 3. Richtlijnen voor hybride projecten

### 5.5.4. 4. Risicobeheersing en leerstrategieën

Voor het beantwoorden van deelvraag 8, die zich richt op het destilleren van best practices en richtlijnen voor de keuze tussen Mendix en traditionele ontwikkeling, wordt een gestructureerd beslissingskader ontwikkeld. Dit kader ondersteunt projectmanagers en architecten bij het maken van weloverwogen keuzes over de inzet van Mendix in vergelijking met conventionele ontwikkelmethoden. Het biedt beslissingspunten voor de initiële keuze tussen Mendix en traditionele ontwikkeling en stelt criteria vast om te bepalen welke projectonderdelen beter geschikt zijn voor high-code ontwikkeling. Daarnaast worden richtlijnen geformuleerd voor het effectief combineren van low-code en high-code in hybride projecten, zodat beide benaderingen optimaal kunnen worden ingezet. Verder bevat het kader aanbevelingen voor de vroege identificatie van potentiële beperkingen en risico's, waardoor

projectrisico's proactief kunnen worden gemitigeerd. Tot slot worden strategieën ontwikkeld om op basis van eerdere ervaringen en literatuur weloverwogen keuzes te maken bij toekomstige projecten. Dit beslissingskader draagt bij aan een systematische en onderbouwde aanpak voor het selecteren van de meest geschikte ontwikkelmethode binnen uiteenlopende projectcontexten.

# 6

## Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetur libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem. Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.



## Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

### Samenvatting

In het digitale landschap van vandaag zoeken bedrijven steeds naar manieren om zakelijke applicaties efficiënter te ontwikkelen. Low-code ontwikkelplatformen, zoals Mendix, zijn de laatste jaren sterk gegroeid als een vervanging voor traditionele high-code ontwikkeling. Deze platformen beloven snellere ontwikkelingstijden en toegankelijkheid door visuele interfaces en herbruikbare componenten. Er is echter nog onduidelijkheid over wat de grenzen van de low-code-tool Mendix zijn bij het ontwikkelen van zakelijke applicaties, en wanneer high-code-ontwikkeling meer geschikt is.

Dit onderzoek duikt diep in de technische mogelijkheden en beperkingen van Mendix om deze centrale vraag te beantwoorden. Door de complexiteit van deze vraag te ontrafelen, zal de studie een genuanceerd beeld schetsen van de huidige stand van low-code technologie.

Een gemengde onderzoeksmethodologie zal worden gehanteerd, bestaande uit een diepgaande literatuurstudie, praktische experimenten en interviews met ervaren ontwikkelaars. Deze veelzijdige onderzoeksaanpak stelt ons in staat om de technische grenzen van Mendix grondig te analyseren, de prestaties in verschillende scenario's te evalueren en concrete richtlijnen te formuleren voor organisaties die overwegen low-code te implementeren.

Het onderzoek zal een gedetailleerde analyse geven van de technische beperkingen van Mendix en use cases identificeren waarin de ontwikkeling van high-code meer geschikt is. Het zal ook een beslissingskader bieden, zodat bedrijven en an-



dere instellingen kunnen kiezen tussen low-code en high-code methoden. Uiteindelijk zal het richtlijnen bevatten voor het optimaal combineren van de twee methoden om de beste resultaten te behalen.

Gezien de groeiende adoptie van low-code platforms is er dringend behoefte aan objectief onderzoek naar hun mogelijkheden en beperkingen. Deze bachelorproef helpt om deze behoefte aan kennis te vervullen en biedt handige adviezen voor organisaties die overwegen om low-code of traditionele ontwikkelmethoden toe te passen.

## **A.1. Inleiding**

Apvine, een toonaangevend IT-consultancybedrijf, richt zich op het creëren van applicaties met low-code platforms, voornamelijk Mendix. Deze aanpak is zeer effectief gebleken voor de meeste van hun projecten en maakt snelle ontwikkeling en implementatie mogelijk. Toch zijn er situaties waarin projecten uitdagingen bieden die de grenzen van low-code platforms opzoeken. Denk hierbij aan ingewikkelde bedrijfslogica, intensieve real-time verwerking of complexe integraties met legacy-systemen, die de mogelijkheden van een puur low-code methode kunnen overstijgen.

In dergelijke situaties kan het noodzakelijk zijn om over te stappen op een hybride of high-code methode. Het is echter niet eenvoudig om te beslissen wanneer deze overgang moet gebeuren. Zonder duidelijke richtlijnen loopt Apvine het risico op vertragingen, hogere uitgaven en ontevreden klanten door late of onverwachte aanpassingen in de ontwikkelingsstrategie. Om deze valkuilen te vermijden, richt dit onderzoek zich op de centrale vraag: “Wat zijn de grenzen van de low-code-tool Mendix bij het ontwikkelen van zakelijke applicaties, en wanneer is high-code-ontwikkeling meer geschikt?”

Dit onderzoek heeft als doel een raamwerk voor besluitvorming te ontwikkelen dat projectmanagers en architecten van Apvine ondersteunt bij het beoordelen of ze moeten blijven met low-code of overstappen naar high-code voor een specifiek project. Het raamwerk zal worden gebaseerd op:

- Systematische analyse van eerdere projecten om gemeenschappelijke succesfactoren en uitdagingen te identificeren.
- Richtlijnen voor het implementeren van hybride methoden om zowel flexibiliteit als complexiteit te combineren.
- Een hulpmiddel dat beslissingen ondersteunt tijdens de fases van pre-sales en planning.

Om de centrale vraag te beantwoorden, worden de volgende deelvragen onder-

zocht:

**Probleemdomein:**

1. Wat zijn de gevolgen van het ontbreken van een beslissingskader voor de keuze tussen low-code en high-code bij Apvine?
2. Welke problemen ontstaan er in projecten door het gebrek aan richtlijnen voor de keuze tussen ontwikkelmethoden?
3. Wat zijn de huidige criteria die Apvine gebruikt bij het kiezen tussen low-code en high-code ontwikkeling?
4. Welke knelpunten ervaren projectmanagers en architecten bij het maken van de keuze tussen low-code en high-code?

**Oplossingsdomein:**

5. Wat zijn de technische beperkingen van Mendix bij het omgaan met complexe bedrijfslogica, intensieve real-time verwerking en integraties met legacy-systemen?
6. In welke specifieke scenario's binnen Mendix-projecten kan een hybride of high-code oplossing nodig zijn, en wat zijn de triggers voor het maken van deze overstap?
7. Hoe kunnen projectomvang, tijdslijnen en klantvereisten de beslissing beïnvloeden om wel of niet met Mendix (low-code) te werken, of over te schakelen naar high-code?
8. Welke lessen kunnen worden getrokken uit de ervaring van eerdere projecten bij Apvine waarin Mendix werd ingezet, en hoe kunnen deze inzichten helpen bij het bepalen wanneer een hybride of high-code oplossing nodig is?

Het beslissingskader dat uit dit onderzoek voortkomt, biedt een gestructureerde aanpak om organisaties te helpen bij het maken van een weloverwogen keuze tussen low-code en high-code ontwikkelmethoden. Met dit kader zijn bedrijven zoals Apvine in staat om niet alleen effectiever te plannen, maar ook de risico's van late veranderingen in projecten te verlagen, de ontwikkelingskosten effectiever te beheersen en hoogwaardige oplossingen te bieden die voldoen aan de unieke wensen van hun klanten.

## A.2. Literatuurstudie

### A.2.1. Low-code en Mendix

Het landschap van digitale klantervaringen evolueert snel, aangewakkerd door de groeiende verwachtingen van klanten en de behoefte voor bedrijven om steeds consistente en hoogwaardige ervaringen te bieden. Zoals beschreven in de tekst “Deliver Standout Digital Customer Experiences with Low-Code” door het bedrijf Mendix (2023), worden bedrijven geconfronteerd met aanzienlijke uitdagingen om aan deze eisen te voldoen vanwege de complexiteit van traditionele ontwikkelpraktijken en beperkte middelen.

Ongelijksoortige processen en afzonderlijke oplossingen leiden vaak tot onsamenvhangende ervaringen die niet aansluiten bij de wensen van de klant (**Mendix2023**). Ondernemingen moeten worstelen met het beheer van meerdere complexe technologieën, afzonderlijke applicaties en een vertraagde time-to-market. Dit alles hindert hen erin om zich snel aan te passen en de ervaringen te bieden die klanten verwachten.

Low-code ontwikkelplatforms zijn aangetoond als een effectieve oplossing voor deze problemen, waardoor bedrijven de controle kunnen nemen over de klantervaring. Low-code platformen maken zowel technische als niet-technische teams mogelijk om samen te werken aan de snelle ontwikkeling van multi-ervaringstoepassingen door snelle ontwikkelingskansen en een klantgerichte benadering te bieden.

Met name het low-code Mendix-platform is ontworpen om bedrijven te ondersteunen bij digitalisering en om consistente, toekomstbestendige contactpunten voor klanten te creëren voor diverse kanalen en apparaten. Dankzij functies zoals automatisering, AI-ondersteuning en cloud-native schaalbaarheid stelt Mendix bedrijven in staat om snel applicaties te bouwen en te implementeren, terwijl ze tegelijkertijd flexibel en consistent blijven. (Mendix, 2023)

### A.2.2. Low-code use cases

Low-code ontwikkelplatforms maken het voor organisaties mogelijk om een breed scala aan bedrijfsapplicaties te creëren, zodat ze hun bedrijfsprocessen kunnen faciliteren en de gebruikerservaringen kunnen optimaliseren. Zo zijn er veelvoorkomende use cases:

- Legacy modernisering

Met low-code hebben bedrijven de mogelijkheid om legacy-systemen te combineren met nieuwere technologieën, bestaande kansen te vergroten of verouderde systemen te vervangen om in te spelen op veranderende bedrijfsbehoeften. Banco de Occidente maakte gebruik van een low-code platform om hun verouderde systemen te integreren en hun processen te verbeteren, wat leidde tot een betere ervaring voor zowel klanten als medewerkers (Bunce, 2024b).

- Portalen  
Low-code stelt gebruikers in staat om op maat gemaakte, webgebaseerde portals te creëren die hen van belangrijke informatie en acties voorzien, wat de efficiëntie en gebruikerservaring ten goede komt. DHL Group heeft een platform ontwikkeld voor het beheer van leveranciersstamgegevens, gebaseerd op een low-code platform, waarmee het werk wordt geautomatiseerd. (Bunce, 2024b).
- Mobiele apps  
Low-code applicaties functioneren op diverse apparaten zonder dat er hercodering vereist is, waardoor gebruikers toegang krijgen tot het apparaat dat ze verkiezen. Super Bock Group maakte gebruik van low-code voor het ontwikkelen van een mobiel goedkeuringsproces voor aankoopaanvragen, wat resulteerde in kortere responstijden (Bunce, 2024a).
- Integratie  
Low-code platforms maken het mogelijk om verschillende systemen en gegevensbronnen te integreren, zodat gebruikers in één interface toegang krijgen tot informatie. Zo maakte bijvoorbeeld Unilever gebruik van een low-code platform om hun SAP-organisaties en prijsstrategieën voor klanten te integreren, wat resulteerde in een hogere efficiëntie en precisie (Bunce, 2024b).

Dit zijn slechts een paar voorbeelden van de veelzijdigheid en brede toepassing van low-code ontwikkelplatformen in diverse industrieën en gebruikssituaties. Bij een volledige literatuurstudie kan/zal dit verder uitgewerkt worden.

### **A.2.3. Beperkingen van low-code**

Hoewel low-code ontwikkelplatformen veel voordelen opleveren bij het versnellen van de applicatielevering, hebben ze ook enkele opvallende nadelen waar organisaties rekening mee moeten houden. Het artikel “9 Low Code Limitations in 2024 to Know About” (Malak, 2024) beschrijft verschillende belangrijke beperkingen die verband houden met low-code methoden. Deze beperkingen omvatten:

- Een tekort aan controle over de automatisch gegenereerde code kan een obstakel vormen voor ingewikkelde applicaties die codeoptimalisatie vereisen.
- Potentiële vendor lock-in door eigen frameworks of programmeertalen maakt het lastig om van platform te veranderen of met andere systemen te integreren.
- Er zijn beperkte mogelijkheden voor aanpassing, omdat low-code platforms beter geschikt zijn voor algemene functies dan voor ingewikkelde, specifieke vereisten binnen een bepaalde sector.

- Beveiligingsproblemen ontstaan doordat low-code ontwikkeling sommige beveiligingsaspecten kan over het hoofd zien, vooral in streng gereguleerde sectoren.
- Complexiteit van integratie komt voort uit het feit dat low-code platforms mogelijk niet perfect kunnen integreren met de huidige infrastructuur van een organisatie en externe diensten.
- Beperkingen in de schaalbaarheid, omdat low-code applicaties problemen kunnen ondervinden bij het aanpassen aan de groeiende vraag van gebruikers en het volume aan gegevens.

De geschiktheid van low-code voor een specifiek project zal uiteindelijk afhankelijk zijn van elementen zoals de ingewikkeldheid van de applicatie, de technische expertise van de organisatie, de behoefte aan beveiliging en naleving, evenals de langetermijnschaalbaarheidseisen. Door de in dit artikel genoemde beperkingen grondig te beoordelen, kunnen bedrijven beter geïnformeerde keuzes maken over het gebruik van low-code ontwikkeling en wanneer traditionele codering beter aansluit.

#### **A.2.4. Low-code versus high-code**

Hoewel ze fundamenteel van elkaar verschillen, kunnen low-code en high-code ontwikkelmethoden elkaar aanvullen om aan diverse projecteisen te voldoen. High-code ontwikkeling levert ongeëvenaarde controle, flexibiliteit en de optie om complexe, aangepaste toepassingen vanaf nul te ontwikkelen, wat het perfect maakt voor grote of ingewikkelde projecten die afhankelijk zijn van geavanceerde functies en integraties. Low-code-platforms vergemakkelijken het ontwikkelproces met visuele hulpmiddelen en kant-en-klare componenten, wat ervoor zorgt dat prototypes en de ontwikkeling van middelmatig complexe toepassingen sneller verlopen.

Door deze methoden in evenwicht te brengen, kunnen bedrijven hun productiviteit maximaliseren, de time-to-market voor bepaalde projecten verkorten en ervaren ontwikkelaars inzetten waar ze het meest nodig zijn, zoals Ballejos (2024) benadrukt in uitgebreide overzicht van de verschillen en mogelijke wisselwerking tussen deze methodologieën.

#### **A.3. Methodologie**

Dit onderzoek is opgedeeld in twee fasen: eerst het in kaart brengen van het probleemdomain (het ontbreken van een beslissingskader), gevolgd door onderzoek naar het oplossingsdomain (de ontwikkeling van het kader).

**A.3.1. Fase 1: Analyse van het probleemdomain**

Om een grondig inzicht te krijgen in het probleemdomain worden de volgende onderzoeksmethoden gebruikt:

**Analyse van historische projectdata**

Om deelvragen 1 en 2 te beantwoorden ("Wat zijn de gevolgen van het ontbreken van een beslissingskader?" en "Welke problemen ontstaan er in projecten?"), wordt een gestructureerde analyse uitgevoerd van afgesloten projecten. Deze analyse omvat:

- Vergelijking van initiële projectschattingen versus werkelijke uitkomsten om impact op kosten en doorlooptijd te kwantificeren
- Analyse van projectdocumentatie om momenten te identificeren waar keuzes tussen low-code en high-code tot problemen leidden
- Inventarisatie van situaties waarin late overschakeling naar alternatieve ontwikkelmethoden nodig was
- Evaluatie van de financiële en tijdsimpact van deze late aanpassingen

**Interviews met experts**

Voor het beantwoorden van deelvragen 3 en 4 ("Wat zijn de huidige criteria?" en "Welke knelpunten ervaren projectmanagers?") worden diepte-interviews gehouden met:

- Projectmanagers over hun huidige besluitvormingsproces
- Architecten over hun ervaringen met technologie-keuzes
- Pre-sales consultants over hun aanpak bij het inschatten van projectgeschiktheid
- Ontwikkelaars over de uitdagingen die zij ervaren bij technologie-keuzes

**Documentatieonderzoek**

Om deelvraag 3 verder te onderbouwen wordt bestaande interne documentatie geanalyseerd:

- Huidige richtlijnen en procedures voor projectaanpak
- Pre-sales documentatie en offertes
- Project kick-off documenten
- Architectuurbeslissingen en design documents

**A.3.2. Fase 2: Onderzoek naar het oplossingsdomein**

Na het volledig in kaart brengen van het probleem, richt het onderzoek zich op het ontwikkelen van een oplossing middels:

**Literatuuroverzicht**

Een diepgaande analyse van bestaande documentatie over Mendix en andere low-code platforms voor het beantwoorden van deelvraag 5 en 7:

- Officiële Mendix productgidsen
- Casestudies van derden en rapporten uit de industrie
- Online forums en ontwikkelaarsgemeenschappen
- Academische publicaties over low-code ontwikkeling

**Praktisch onderzoek**

Uitvoering van praktische tests voor deelvraag 6 om de grenzen van het Mendix-platform te onderzoeken:

- Schaalbaarheid en prestaties
- Integratiemogelijkheden
- Aanpasbaarheid en uitbreidbaarheid
- Ontwikkelingssnelheid
- Beveiliging en compliance

**Ontwikkeling beslissingskader**

Voor het beantwoorden van deelvraag 8 ("Welke best practices en richtlijnen voor de keuze tussen Mendix en traditionele ontwikkeling kunnen worden gedestilleerd uit de projectervaringen en literatuur?") wordt op basis van alle verzamelde inzichten een gestructureerd beslissingskader ontwikkeld dat projectmanagers en architecten ondersteunt bij het maken van weloverwogen keuzes over de inzet van Mendix versus traditionele ontwikkelmethoden.

**A.4. Verwacht resultaat, conclusie**

Door het combineren van de kennis uit het literatuuronderzoek, de praktische bevindingen uit de experimenten en de inzichten uit de interviews met experts, heeft het onderzoek als doel een uitgebreid begrip te ontwikkelen van de mogelijkheden en beperkingen van Mendix in de context van de ontwikkeling van bedrijfsapplicaties.

Als afsluiting van dit onderzoek wordt ook een beslissingskader opgesteld dat organisaties een gestructureerde aanpak biedt om een best passende keuze te maken tussen low-code en high-code ontwikkelmethoden. Dit kader stelt bedrijven zoals Apvine in staat om niet alleen effectiever te plannen, maar ook de risico's van late veranderingen in projecten te beperken. Hierdoor worden de ontwikkelingskosten beter beheerd en hoogwaardige oplossingen opgeleverd die aansluiten bij de unieke wensen van hun klanten.



# Bibliografie

- Alamin, M. A. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., & Iqbal, A. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. <https://doi.org/10.48550/ARXIV.2103.11429>
- Ballejos, L. (2024, juni 6). *High Code vs. Low Code vs. No Code: Navigating the Best Coding Solutions for Your Needs*. Verkregen november 15, 2024, van <https://www.ninjaone.com/blog/high-code-vs-low-code-vs-no-code/>
- Bunce, C. (2024a, april 2). *Low-Code Use Cases: What Can You Build With a Low-Code Platform?* Verkregen november 15, 2024, van <https://www.bizagi.com/en/blog/low-code-use-cases>
- Bunce, C. (2024b, mei 21). *Low-Code vs High-Code: Decoding the Two Development Approaches*. Verkregen november 15, 2024, van <https://www.bizagi.com/en/blog/low-code-vs-high-code>
- Case, A. F. (1985). Computer-aided software engineering (CASE): technology for improving software development productivity. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 17(1), 35–43. <https://doi.org/https://doi.org/10.1145/1040694.1040698>
- Figueira-Putresza, Y. (2021). OutSystems tutorial: Learn low-code development and get your first certificate. Verkregen maart 7, 2025, van <https://pretius.com/blog/outsystems-tutorial/>
- Gadia, S. (2025, april 25). *Mendix vs OutSystems: Faceoff Between Two Best Low Code Development Platforms*. Verkregen mei 8, 2025, van <https://www.softude.com/blog/mendix-vs-outsystems-faceoff-between-two-best-low-code-development-platforms>
- Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. In *IBM Systems Journal* (pp. 451–461, Deel 45). IBM. <https://doi.org/https://doi.org/10.1147/sj.453.0451>
- Henkel, M., & Stirna, J. (2010). Pondering on the Key Functionality of Model Driven Development Tools: The Case of Mendix. In *Perspectives in Business Informatics Research* (pp. 146–160). Springer Berlin Heidelberg. [https://doi.org/https://doi.org/10.1007/978-3-642-16101-8\\_12](https://doi.org/https://doi.org/10.1007/978-3-642-16101-8_12)
- Hermans, M. P., & Mileff, P. (2023). Short introduction to Mendix. <https://doi.org/https://doi.org/10.32968/psaie.2023.3.5>
- Hugo. (2024, februari 9). *Joget DX 7 Knowledge Base - Process Builder*. <https://dev.joget.org/community/display/DX7/Process+Builder>

- Krouwel, M. R., Op 't Land, M., & Proper, H. A. (2022). Generating Low-Code Applications from Enterprise Ontology. In *The Practice of Enterprise Modeling* (pp. 18–32). Springer International Publishing. [https://doi.org/https://doi.org/10.1007/978-3-031-21488-2\\_2](https://doi.org/https://doi.org/10.1007/978-3-031-21488-2_2)
- Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021). Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective. <https://doi.org/10.48550/ARXIV.2107.07482>
- Malak, H. A. (2024, oktober 13). *9 Low Code Limitations in 2024 to Know About*. Verkregen november 15, 2024, van <https://theecmconsultant.com/low-code-limitations/>
- Marín, B., Salinas, A., Morandé, J., Giachetti, G., & de la Vara, J. L. (2015). Main Features for MDD Tools: An Exploratory Study. In *Model-Driven Engineering and Software Development* (pp. 183–196). Springer International Publishing. [https://doi.org/10.1007/978-3-319-25156-1\\_12](https://doi.org/10.1007/978-3-319-25156-1_12)
- Mendix. (z.d.). *A journey through Low-Code and Agile Development with AZL*. [mendix.com/customer-stories/a-journey-through-low-code-and-agile-development-with-azl/](https://mendix.com/customer-stories/a-journey-through-low-code-and-agile-development-with-azl/)
- Mendix. (2023, maart 14). *Deliver Standout Digital Customer Experiences with Low-Code*. <https://www.mendix.com/strategies/digital-customer-experiences/>
- Mendix. (2025a). *Low-Code Examples and Use Cases*. <https://www.mendix.com/low-code-guide/low-code-use-cases/>
- Mendix. (2025b, februari 26). *Download Studio Pro*. <https://marketplace.mendix.com/link/studiopro>
- MxTechies. (z.d.). *Open and Extensible Solutions*. Verkregen april 26, 2025, van <https://www.mxtechies.com/mendix-extensibility>
- Sido, N., & Emon, E. A. (2024, mei 31). *Low/No Code Development and Generative AI*.
- Soley, R., & the OMG Staff Strategy Group. (2000). *Model Driven Architecture*. Verkregen maart 7, 2025, van [https://www.vico.org/aRecursosMDA/MDA\\_paper3\\_2.pdf](https://www.vico.org/aRecursosMDA/MDA_paper3_2.pdf)
- van Oosten, A. (2020, juni 11). *Achieve New Levels of Business Value with Low Code*. Mendix. <https://www.mendix.com/blog/achieve-new-levels-of-business-value-with-low-code/>
- Vanderkooy, M. (2021). *The Rise of No-code*. <https://www.erp-one.com/blog/the-rise-of-no-code>
- Yerukala, M. (2022, oktober 10). *What is Mendix*. MindMajix. <https://mindmajix.com/mendix-tutorial#benefits>
- Zaman, S. (2024, september 1). *15 Low-Code Use Cases and Examples (Featuring Mendix)*. <https://impalaintech.com/blog/low-code-use-cases>