Web API’s bij Zilveren Kruis Achmea

Afstudeerscriptie

Mark Staarink – 1618662
Software Engineering
Hogeschool Utrecht
Jan Willem Pauw
Zilveren Kruis Achmea
19-12-2016

Managementsamenvatting

Bij Zilveren Kruis Achmea is een onderzoek uitgevoerd om een architecturale manier te vinden om gebruikerservaring van de Zilveren Kruis websites te verbeteren.

De businesswensen van de websites zijn:

- Verbeterde performance, zodat gebruikers binnen twee seconden opgevraagde gegevens te zien krijgen.
- Een lagere Time To Market, zodat verbeteringen binnen twee weken bij de gebruikers terecht komen.
- De mogelijkheid tot hot deployment, zodat de websites tijdens het doorvoeren van wijzigingen beschikbaar blijven.
- Kanaalonafhankelijke bediening, zodat gebruikers hun gegevens in meerdere websites kunnen beheren en wijzigingen. Wijzigingen zijn direct in andere websites, apps en systemen zichtbaar.
- Customer journeys kunnen volgen om gebruikspatronen in beeld te krijgen. Dit ter verbetering van procesvoering en de gebruikerservaring van de websites.

Het uitgevoerde onderzoek draait om de hoofdvraag:

Hoe kunnen de Zilveren Kruis portalen overstappen naar een architectuur welke gebruik maakt van web API's, om complexiteit en koppeling te verminderen?

Het onderzoek resulteert in een advies aan Zilveren Kruis Achmea voor een doelarchitectuur en migratieplan.

Binnen de huidige architectuur zijn verschillende lagen te onderscheiden:

- **Frontend**: Bevat web applicaties en hun content. Veelal geïmplementeerd in SharePoint en dynamisch gemaakt middels Knockout JS.
- **Middleware**: Faciliteert de communicatie tussen de frontend applicaties en de backend systemen en hun services.
    - **Transactie-laag**: Biedt business logica voor de frontend. Bevat een Transactie Component, welke de domein API vormt. Geïntegreerd in de SharePoint frontend.
    - **Integratie-laag**: Biedt interfaces voor het aanroepen van de services van de backend systemen. Geïmplementeerd middels verschillende software componenten, waaronder IBM Message Queues.
- **Backend**: Bevat de verschillende backend systemen met hun services en achterliggende databases.

Er is samengewerkt met de portalen afdeling en andere stakeholders op basis van de agile werkwijze van Zilveren Kruis Achmea. Er is in meetings samengewerkt om te komen tot kennisdeling, een gezamenlijk beeld en een gedragen architectuur. De architectuurproblemen, eisen en randvoorwaarden zijn geïnventariseerd. Samen zijn er mogelijke oplosrichtingen boven water getild en beoordeeld. Op basis van dit onderzoek is een doelarchitectuur en migratieplan opgesteld.

De problemen van de huidige architectuur zijn:

- Sterke koppeling, veel onderlinge afhankelijkheden in de transactie-laag.
- Versionering van de transactie-laag is beschikbaar maar wordt niet gebruikt.
- Performance is laag. Caching kan beter en service aanroepen zijn langzaam.
- Verouderde code blijft in gebruik, wat onderhoud vermoeilijkt.
- Verschillende soorten logica worden door elkaar heen gebruikt, wat onderhoud en vernieuwing vermoeilijkt.
- Service interfaces sluiten slecht aan op de gebruikers use cases.
- De implementatie van de transactie-laag maakt kanaalonafhankelijke bediening niet mogelijk.

De onderzoeksvraag gaat primair over de transactie-laag. De voorgestelde architectuur uit het onderzoek omvat alle lagen. Over de front- en backend worden aanbevelingen gedaan die per portaal verder onderzocht moeten worden.

De voornaamste wijzigingen aan de transactie-laag zijn:

- De transactie-laag stapt van SharePoint af en wordt als generieke ASP.NET applicatie geïmplementeerd op een IIS server. Dit verhoogt flexibiliteit.
- De domein API wordt opgesplitst om de ongewenste koppeling weg te werken en versionering te realiseren.
- Caching wordt verbeterd om performance te verhogen.
- Domeinfunctionaliteit wordt ontsloten door data-gerichte, portaal-generieke web API's welke de domein API's ontsluiten om kanaalonafhankelijke bediening te realiseren.

Voor de frontend zijn de volgende wijzigingen aanbevolen:

- Website content wordt gerealiseerd middels het JavaScript framework Angular 2, ter behoeve van testbaarheid en voldoen aan de eisen.
- De frontend stapt van SharePoint af en wordt middels Sitecore geïmplementeerd. Dit biedt nuttige functionaliteit ter behoeve van performance en personalisatie.

Het volgende wordt aanbevolen voor alle backend services:

- Gefilterde service data wordt in een geoptimaliseerde cache database voor de frontend opgeslagen. Hierdoor wordt de performance verbeterd. Per service is onderzoek nodig of de kosten opwegen tegen de baten.

In het migratieplan naar de doelarchitectuur worden web API's als eerste gerealiseerd, waardoor de migratie van de frontend ontkoppeld wordt. Daarna zijn er meerdere migratiedelen welke onafhankelijk van elkaar uitgevoerd kunnen worden, waardoor het ontwikkelteam de flexibiliteit heeft te prioriteren op basis van huidige en toekomstige werkzaamheden.

De architecten en ontwikkelaars zijn akkoord met de doelarchitectuur en migratieplan. Alle problemen worden verholpen en de businesswensen kunnen worden gerealiseerd. Het vervolg na dit onderzoek is het maken van een proof of concept en het formaliseren van de architectuur in de portalen roadmap.

Inhoudsopgave

Begrippenlijst

Begrip	Definitie
Agile	Een set van principes voor softwareontwikkeling.
Angular 2	JavaScript framework voor het realiseren van Single Page Applications (web applicaties welke binnen een enkele webpagina werken).
ASP.NET	Server-side web applicatie framework voor het realiseren van dynamische web pagina's.
Big Bang oplevering	In één stap grote veranderingen opleveren.
Customer journey	De manier waarop een klant een product gebruikt, van ontdekking tot en met interactie.
DEMO	Design & Engineering Methodology for Organizations, een methodiek voor het modelleren van organisaties en hun processen.
DLL	Dynamic-link library, een manier om software te laden in Microsoft-omgevingen.
Hot deployment	Het vrijgeven van een nieuwe versie van software, zonder het functioneren van het betrokken systeem(onderdeel) te belemmeren.
HTTPS	Protocol voor veilige communicatie over een computer netwerk. Uitbreiding op HTTP, het Hypertext Transfer Protocol.
IIS	Een web server voor het Windows platform.
Kanaalonafhankelijke bediening	Bediening waarbij de gebruikerservaring over meerdere kanalen (website, mobiele app) heen plaats kan vinden, terwijl de functionaliteit consistent blijft.
Knockout JS	JavaScript framework op basis van het Model-View-ViewModel patroon met templates.
REST	Representational state transfer, een manier om web service interfaces te ontwerpen voor het verkrijgen en manipuleren van data volgens operaties zonder state.
SharePoint	Een web applicatie platform voor collaboratiesites.
Sitecore	Een web content management systeem.
SOAP	Simple Object Access Protocol, een protocol voor het uitwisselen van data tussen web services welke gebruik maakt van XML voor het beschrijven en realiseren van berichtformaten.
Time To Market	Marktintroductietijd, de benodigde tijd om een nieuwe versie van de software te ontwikkelen en vrij te geven.

1. Inleiding

Ter afstuderen bij de studie Software Engineering aan de Hogeschool Utrecht heeft Mark Staarink onderzoek uitgevoerd bij Zilveren Kruis Achmea. Dit document dient als verslaglegging van dit onderzoek.

Allereerst worden Achmea en de aanleiding van het onderzoek besproken, evenals een overzicht van het onderzoek zelf. Vervolgens worden de onderzoeksresultaten besproken, welke een gedetailleerdere context en probleemanalyse bevatten en op basis van die kennis toewerken naar een onderbouwd advies.

2. De Achmea organisatie

Om de context waarin het onderzoek zich afspeelt beter te begrijpen is het belangrijk om een beeld te hebben van de organisatie waarin dit zich afspeelt.

Gedurende het uitvoeren van het onderzoek heeft een reorganisatie plaatsgevonden. Dit heeft geen significante impact gehad op het onderzoek.

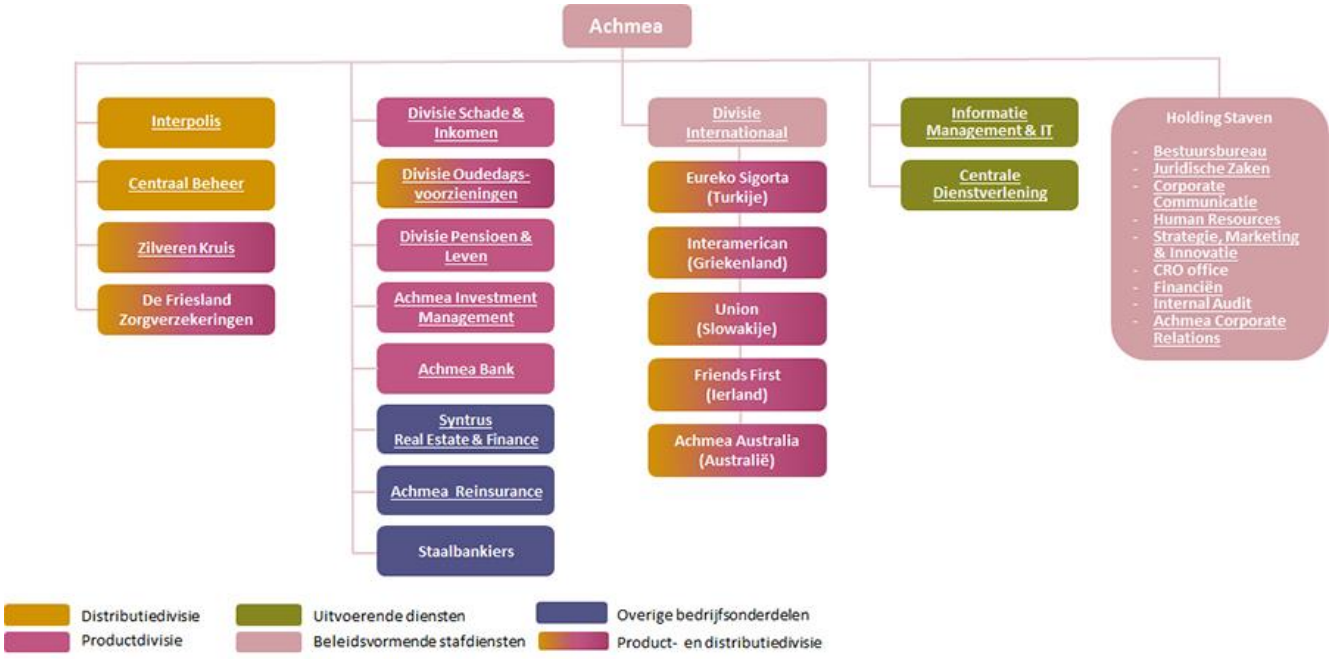
2.1. Achmea & Zilveren Kruis

Achmea is een verzekeringsconcern, opgericht in 1811. Het omvat verschillende bekende verzekeringsmerken, van zorg- tot schadeverzekeringen. De grootste daarvan zijn Zilveren Kruis, Interpolis, FBTO, Centraal Beheer en Avéro Achmea. (Achmea, 2016, p. 2)

De doelstelling van Achmea is om de meest vertrouwde digitale verzekeraar te zijn. Hiertoe is tussen 2009 en 2011 gewerkt aan het op orde krijgen van het interne IT-landschap. Waar dit eerst gedecentraliseerd en geografisch verspreid was, is dit geconsolideerd en onder controle gebracht. Sinds 2012 wordt gefocust op het verminderen van de complexiteit in business en IT om beter ondersteuning te kunnen bieden aan toekomstige innovatie. (Achmea, 2015, pp. 9-10)

In Figuur 1 (Achmea, z.j.) wordt het organogram van Achmea gegeven. Hierin is aangegeven hoe de verschillende merken onder Achmea vallen, en welke ondersteunende divisies en diensten bij Achmea aanwezig zijn.

Figuur 1 – Organogram Achmea. Herdrukt van *AchmeaNet* website, door Achmea, z.j., opgehaald van https://organisatie.achmeanet.nl/onzeorganisatie/Paginas/organogram.aspx.

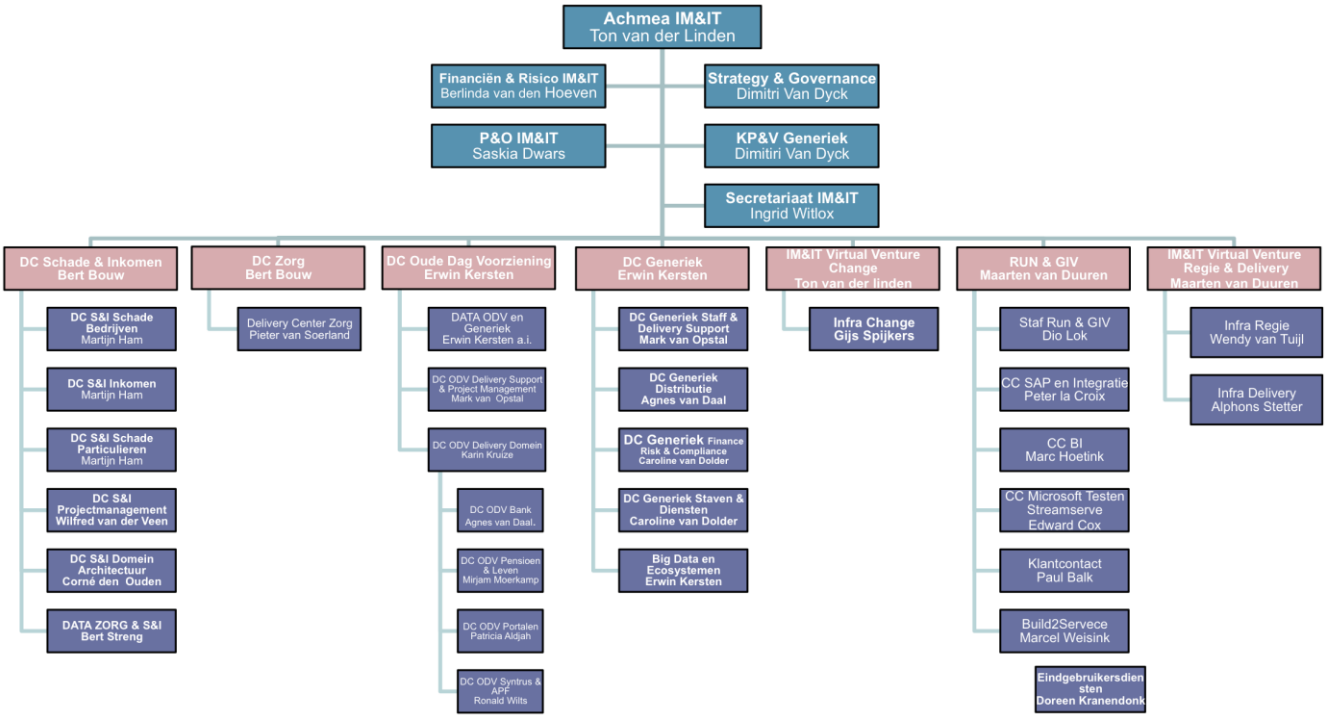


Zilveren Kruis is het grootste zorg-merk van Achmea. Het koopt jaarlijks voor 4,1 miljoen klanten zorg in (met een totaalwaarde van 11,6 miljard Euro), en verwerkt 2,5 miljoen wijzigingen in hun polissen. 2.800 medewerkers staan Zilveren Kruis hierin bij. Hun callcenter handelt in een jaar 3,7 miljoen telefoongesprekken af, en beantwoord 402.000 mails en 180.000 chat-gesprekken. (Achmea, 2016, p. 3)

Sinds 2016 gebeuren de zorginkopen van Zilveren Kruis als zorgmodules. Deze worden opgesteld op basis van de manier waarop klanten zorg gebruiken. Hierbij wordt gekeken naar drie waarden die een klant kan ervaren: kwaliteiten van zorg, kosten van zorg, en kwaliteit van leven. (Achmea, 2016, p. 4)

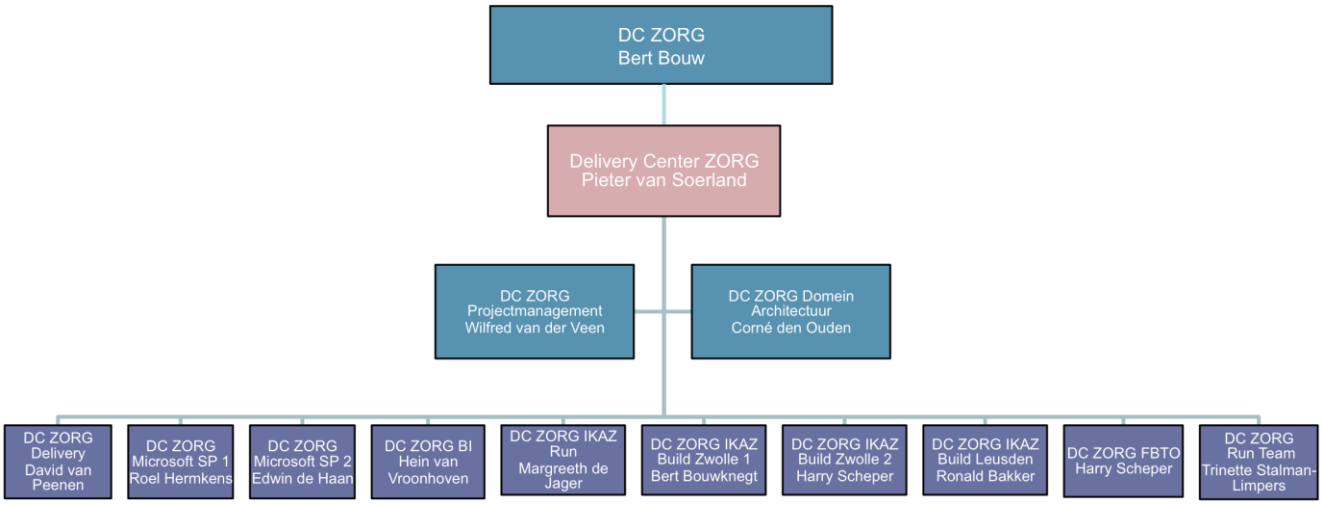
De uitvoerende dienst Informatie Management & Informatie Technologie (afgekort IM&IT) houdt zich bezig met de informatiesystemen welke de werking van de andere afdelingen ondersteunen. Het organogram van IM&IT is in Figuur 2 (Achmea, 2016) afgebeeld.

Figuur 2 – Organogram IM&IT. Herdrukt van *AchmeaNet* website, door Achmea, 2016, opgehaald van https://organisatie.achmeanet.nl/010/Documents/Organogram%20IMenIT%2020161117.pptx.



Binnen IM&IT vallen meerdere partijen, waaronder de Delivery Centers. Zo ook Delivery Center Zorg, weergegeven in Figuur 3 (Achmea, 2016), welke de verschillende teams omvat die zich bezighouden met alle zorg-gerelateerde systemen. Denk hierbij zowel aan de klantwebsites als de achterliggende gegevensverwerking.

Figuur 3 – Organogram Delivery Center Zorg. Herdrukt van *AchmeaNet* website, door Achmea, 2016, opgehaald van https://organisatie.achmeanet.nl/010/Documents/Organogram%20IMenIT%20201611117.pptx.



2.2. Positie en verantwoordelijkheden

Tijdens het uitvoeren van de opdracht wordt als stagiair meegelopen met Scrum Team 1 van Microsoft SP 2, welke binnen Delivery Center Zorg valt en zich bezig houdt met het ontwikkelen en onderhouden van de verschillende zorgportalen (web interfaces voor klanten en medewerkers). De taken hierbij blijven beperkt tot het werk wat voor de opdracht moet gebeuren, met daarnaast alle Scrum-gerelateerde taken zoals het bijwonen van de daily standups, sprint reviews, en dergelijke.

3. De opdracht

Het onderzoek draait om het vinden van een architecturale oplossing voor problemen welke de gebruikerservaring beïnvloeden. In dit hoofdstuk worden zowel deze problemen als de opbouw van het onderzoek beschreven.

3.1. Beperkende architectuur

Achmea biedt een breed scala aan zogeheten portalen, (web) interfaces welke zowel klanten als medewerkers in staat stellen om gegevens in te zien en aan te passen. Denk hierbij bijvoorbeeld aan de klantwebsite voor het beheren van verzekeringsgegevens, of de interface welke de afdeling Klant Contact gebruikt om klantgegevens in te zien.

De backend, waar al deze gegevens opgeslagen en verwerkt worden, bestaat uit ongeveer tachtig verschillende services. Deze kunnen en mogen, onder andere dankzij hun oude technologieën, niet direct met de moderne frontend praten. Om deze communicatie te faciliteren is de transactie-laag als middleware tussen de front- en backend gekomen, geïmplementeerd als DLL's welke door de frontend worden aangeroepen. In de loop der tijd is er ook business logica in de transactie-laag terecht gekomen, waardoor die nu ook de verantwoordelijkheid draagt van zaken als data-validatie.

De huidige architectuur, zoals hierboven globaal beschreven, brengt een aantal problemen met zich mee. Hieronder vallen zaken als gelimiteerde performance en flexibiliteit, en een ongewenst grote opleveringstijd bij veranderingen. Daarnaast is het momenteel lastig voor derden om gegevens van Achmea af te nemen. Dit alles leidt ertoe dat doelen welke op basis van gebruikerswensen (waarbij de gebruikers zowel klanten als medewerkers zijn) gesteld zijn langzaam, deels of niet gehaald worden, wat uiteindelijk resulteert in een slechtere gebruikerservaring.

3.2. Architectuur met web API's

Gezien de ondervonden problemen hun oorzaak hebben liggen in de huidige softwarearchitectuur, is de wens om de problemen op te lossen door middel van een architectuurwijziging. Op deze wens is de opdracht gebaseerd en daaruit volgen twee gerelateerde doelstellingen.

In de eerste plaats moet er een nieuwe softwarearchitectuur komen welke de problemen met de huidige architectuur oplost en voorkomt dat deze in de toekomst terugkeren. In de nieuwe architectuur nemen web API's een van de rollen van de transactie-laag uit de huidige architectuur over en breiden deze uit: een interface bieden aan de frontend voor het ophalen en aanpassen van informatie. Waar de transactie-laag momenteel alleen beschikbaar is voor de SharePoint frontend, moeten de web API's platform-onafhankelijk te gebruiken zijn. Zo kunnen ook derde

partijen informatie van Achmea afnemen. Ook moet er voldaan worden aan andere eisen op basis van richtlijnen, randvoorwaarden en criteria welke tijdens het onderzoek verhelderd zijn.

Ten tweede moet het onderzoek een migratieplan opleveren om van de huidige naar de nieuwe architectuur te komen. Hierbij zijn meerdere tijdens het onderzoek bepaalde criteria belangrijk, waaronder de tijd die de migratie zal kosten en hoeveel van de bestaande code aangepast moet worden.

Tijdens het realiseren van deze doelstellingen moet er, naast gestelde eisen, ook rekening gehouden worden met de mate waarin de beslissende partijen te overtuigen zijn het voorstel aan te nemen en te gebruiken. De architectuur moet gedragen worden. Er zit meer waarde in een suboptimaal voorstel waar “ja” tegen gezegd wordt, dan een volledig optimaal voorstel wat niemand toe zal passen.
Op basis van de onderzoeksresultaten kan een advies opgesteld worden. Wanneer dit advies opgeleverd is en door betrokkenen geaccepteerd wordt zal de opdracht geslaagd zijn.

Na afronding van het onderzoek wordt het advies eventueel getest door middel van een proof of concept.

3.3. Doelstellingen

Aan Zilveren Kruis Achmea worden de volgende producten opgeleverd:

- **Een advies**:
  - **Doelarchitectuur**: Een architectuur welke de problemen van de huidige architectuur oplost en voldoet aan de gestelde eisen en randvoorwaarden.
  - **Migratieplan**: Een uitlijning van de stappen welke nodig zijn om van de huidige architectuur naar de doelarchitectuur te komen.
- **Proof of concept**: Te maken na voltooiing van het onderzoek, om aan te tonen dat het gegeven advies in de praktijk kan functioneren.

3.4. Het onderzoek

Het onderzoek omvat de hoofdvraag:

> **Hoe kunnen de Zilveren Kruis portalen overstappen naar een architectuur welke gebruik maakt van web API’s, om complexiteit en koppeling te verminderen?**

Om tot een antwoord op deze hoofdvraag te komen zijn vijf deelvragen beantwoord. Deze zijn, samen met hun bijbehorende onderzoeksmethoden, weergegeven in Tabel 1. Deze deelvragen hebben elk hun eigen, kleinere deelvragen welke onderzocht zijn. De volledige lijst is terug te vinden in Bijlage B. De onderzoeksmethoden zijn hieronder verder toegelicht.

Tabel 1 – Deelvragen en onderzoeksmethoden

Deelvraag	Methode(n)
Wat is de context van de opdracht	Literatuuronderzoek, interviews
Hoe ziet de huidige situatie eruit?	Literatuuronderzoek, interviews & meetings
Hoe kan de nieuwe architectuur eruit gaan zien?	Literatuuronderzoek, interviews & meetings, vergelijkingsonderzoek
Hoe kan van de oude naar de nieuwe architectuur gemigreerd worden?	Interviews & meetings, vergelijkingsonderzoek
Welke architectuur met bijbehorend migratiepad kan aanbevolen worden?	Vergelijkingsonderzoek

Literatuuronderzoek
Veel informatie over Achmea en de manieren waarop software ontwikkeld wordt is terug te vinden in interne documentatie. Binnen het onderzoek is hier dan ook gebruik van gemaakt voor het achterhalen van eisen, richtlijnen, ontwerpen en ideeën. Daarnaast zijn zowel digitale als fysieke externe bronnen geraadpleegd voor adviezen over het ontwerpen van software architecturen en andere aan het onderzoek verwante onderwerpen.

Interviews & meetings
Hoewel er veel interne documentatie beschikbaar is, omvat dit niet alle voor het onderzoek nodige kennis. Aan software wordt constant gewerkt en afwijkingen van documentatie worden niet altijd vastgelegd. Wat er vastgelegd is, is niet altijd compleet, of beschrijft zaken vanuit een voor het onderzoek irrelevante invalshoek (bijvoorbeeld, technische implementatie). De nodige kennis is echter wel beschikbaar binnen Achmea: haar werknemers hebben veel ervaring met de bestaande systemen en hun werking. Om deze kennis te vergaren is met medewerkers uit verschillende teams en afdelingen gesproken. Dit gebeurde in een-op-een interviews, maar ook in meetings met grotere groepen waarin agile werkvormen werden gebruikt.

Vergelijkingsonderzoek
Om tot de juiste keuzes te komen bij het ontwerpen van de doelarchitectuur en het migratieplan daarnaartoe zijn verschillende mogelijkheden met elkaar vergeleken op basis van hun voor- en nadelen, en hun voldoening aan eisen, richtlijnen en best practices.

4. Onderzoeksresultaten

De zaken welke in dit hoofdstuk aan bod komen vormen de kern van de onderzoeksresultaten. De volledige, uitgebreide resultaten zijn terug te vinden in Bijlage C.

4.1. Context en businesswensen

Voor de uitvoering van het onderzoek moeten de context en het probleem zelf bekend zijn. Allereerst is hier vanuit bedrijfsperspectief naar gekeken, om de aanleiding van het onderzoek en het landschap waarin het speelt beter te begrijpen.

4.1.1. De zorgverzekeraar

Zilveren Kruis Achmea is een zorgverzekeraar. Om de functies van het bedrijf te begrijpen, moeten de transacties die tussen Zilveren Kruis en de betrokken externe partijen plaatsvinden in kaart worden gebracht.

Sterk versimpeld kan gezegd worden dat verzekerden zorg afnemen bij zorgverleners en de zorgverzekeraar hierbij bemiddeld. Het verder uitwerken van deze relaties middels de DEMO methodiek resulteert in transacties welke alle interactie tussen de drie partijen samenvat. Bij deze transacties kan een van de partijen op elk moment de transactie afwijzen.

Tussen de verzekerde en de zorgverzekeraar vinden de volgende transacties plaats:

- Verzekerde schrijft zich in,
  Zorgverzekeraar levert een polisbescheiden aan.
- Verzekerde muteert de gegevens van zijn inschrijving,
  Zorgverzekeraar levert een polisbescheiden aan.
- Verzekerde schrijft zich uit,
  Zorgverzekeraar bevestigt de uitschrijving.
- Verzekerde vraagt toestemming voor een behandeling aan,
  Zorgverzekeraar verleent toestemming.
- Verzekerde vraagt een persoonsgebonden budget aan,
  Zorgverzekeraar verstrekt een persoonsgebonden budget.
- Verzekerde dient restitutie declaratie in,
  Zorgverzekeraar levert vergoeding.
- Zorgverzekeraar vraagt premiebetaling aan,
  Verzekerde betaalt premie.
- Zorgverzekeraar vraagt eigen risicobetaling aan,
  Verzekerde betaalt eigen risico.

Tussen de zorgverlener en zorgverzekeraar vinden de volgende transacties plaats:

- Zorgverlener biedt een contract aan,
  Zorgverzekeraar bevestigt het contract.
- Zorgverlener beëindigt contract,
  Zorgverzekeraar bevestigt de beëindiging.
- Zorgverlener dient een natura declaratie in,
  Zorgverzekeraar levert vergoeding.

4.1.2. Het Achmea applicatielandschap

De applicaties welke Achmea aanbiedt zijn web interfaces, portalen genoemd. De voornaamste portalen zijn de Klant Domein (KD) waarmee klanten van Achmea's verschillende labels hun eigen gegevens kunnen managen, en het Medewerkers Domein (MD), ook wel Mikado genoemd, waarmee medewerkers van de afdeling Klant Contact klantgegevens kunnen managen. Een groot deel van de overige portalen wordt ook intern gebruikt.

De portalen doen voor het implementeren van hun functionaliteiten aanroep op de service applicaties. Deze services omvatten alle gegevens waar Achmea mee in aanraking komt. Verreweg de grootste groep services is de Integrale Kantoorautomatisering Zorgverzekeringen (IKAZ). Het wordt voornamelijk gebruikt voor polis management en het verwerken van declaraties, waarbij gegevens en business-regels van minimaal vijf jaar terug (maar in de praktijk veel langer) onthouden en ondersteund blijven.

4.1.3. Businesswensen

De afdeling Klant Contact van Achmea heeft wensen waarmee de gebruikerservaring van zowel medewerkers als klanten verbeterd kan worden. Een aantal van deze wensen kunnen door problemen met de huidige architectuur niet gerealiseerd worden. Deze zijn als volgt op te sommen:

- Verbeterde performance, zodat gebruikers binnen twee seconden opgevraagde gegevens te zien krijgen.
- Een lagere Time To Market, zodat verbeteringen sneller bij de gebruikers terecht komen.
- De mogelijkheid tot hot deployment, zodat de systemen tijdens het doorvoeren van wijzigingen beschikbaar blijven.
- Kanaalonafhankelijke bediening, zodat gebruikers hun gegevens middels meerdere applicaties kunnen beheren en wijzigingen direct overal zichtbaar zijn.
- Customer journeys kunnen volgen, om gebruikspatronen in beeld te krijgen, onder andere ter verbetering van procesvoering en de gebruikerservaring.

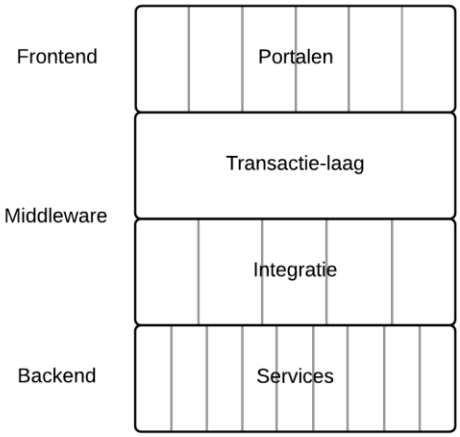
Daarnaast is er de doelstelling van de ontwikkelteams om agile te werken. De absentie van hot deployment zorgt voor noodgedwongen Big Bang opleveringen, wat resulteert in een hoge Time To Market. Dit gaat in tegen de agile werkwijze waar naar gestreefd wordt.

Zolang de huidige architectuur onveranderd blijft zullen de bovengenoemde wensen niet te realiseren zijn.

4.2. Huidige architectuur

Binnen de huidige softwarearchitectuur zijn vier lagen te onderscheiden, zoals weergegeven in Figuur 4.

Figuur 4 – Versimpelde architectuur stack



De verschillende lagen zijn als volgt te beschrijven:

- **Frontend**: De verschillende portalen en hun content.
- **Middleware**: Faciliteert de communicatie tussen de front- en backend.
    - **Transactie-laag**: Biedt business logica voor de frontend.
    - **Integratie-laag**: Biedt versimpelde interfaces voor het aanroepen van de services.
- **Backend**: Bevat de verschillende services en hun databases.

Om de architectuur beter te begrijpen worden de Microsoft stack (frontend en transactie-laag) en de integratie van de backend apart in meer detail bekeken.

4.2.1. De Microsoft stack van Achmea

De portalen en transactie-laag zijn allen geïmplementeerd middels Microsoft technologieën, en worden daarom samen de Microsoft stack genoemd. De meeste portalen werken op SharePoint. De transactie-laag is geprogrammeerd in C# .NET code en wordt door de portalen als DLL aangeroepen. Zowel de bedoelde functionaliteiten als de implementatie zijn terug te vinden in het originele ontwerpdocument (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013) en het klassendiagram van de huidige situatie (van 't Klooster, cd TotalClass, 2014).

4.2.1.1. Functionaliteiten

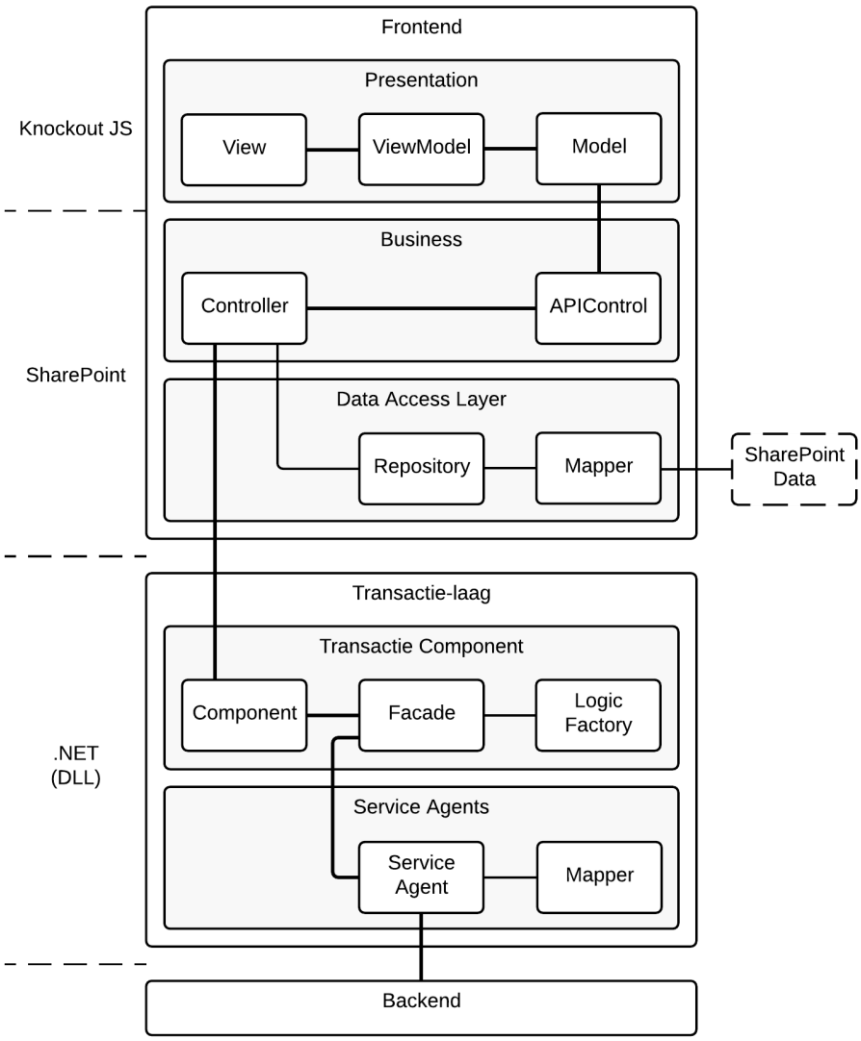
De verschillende portalen hebben allen hun eigen functie, maar de transactie-laag gedraagt zich voor elk portaal grotendeels hetzelfde. Het bemiddeld bij service aanroepen en biedt hierbij de volgende functionaliteiten:

- **Data-validatie**: Gegevens worden op correctheid gecontroleerd voordat deze naar de services gestuurd worden, om “overbodige” aanroepen te voorkomen.
- **Data-aanvulling**: Wanneer nodig wordt incomplete data aangevuld.
- **Caching**: De uit de services ontvangen resultaten worden gecachet om redundante aanroepen te voorkomen.
- **Compensatie**: Business logica kan tekortkomingen van de services opvangen.

4.2.1.2. Implementatie

De frontend en transactie-laag zijn grotendeels opgebouwd uit software ontwerppatronen. Hun architectuur is in Figuur 5 weergegeven en daaronder verder toegelicht.

Figuur 5 – Versimpelde architectuur frontend en transactie-laag



- **Web pagina**, Knockout JS.
  - **View-ViewModel-View**: Ter communicatie met de onderliggende logica (via de APIControl).
- **Frontend logica**, SharePoint.
  - **Controller**: Ophalen van data uit de transactie-laag. Bevat applicatielogica voor de verschillende merken.
  - **Repository & Mapper**: Eenduidige manier om domein entiteiten consistent te houden met hun achterliggende data.
- **Transactie-laag**, .NET, gebruikt als DLL.
  - **Component**: Interface van de transactie-laag.
  - **Facade**: Versimpelde interface voor het aanroepen van de services. Voert eventuele business logica uit.
  - **Service Agent**: Communiceert met de services. Wordt door middel van een Service Locator opgehaald (Inversion of Control).
  - **Mapper**: Vertaalt de uit de services verkregen data naar werkbare entiteiten.

De implementatie van de verschillende portalen verschilt: niet alle portalen werken op (dezelfde versie van) SharePoint, en Knockout JS wordt nog niet overal toegepast.

De portalen welke gebruik maken van SharePoint kunnen geen parallelle aanroepen doen op het onderliggende Transactie Component, wat betekend dat wanneer er meerdere gegevensaanvragen nodig zijn, deze één voor één uitgevoerd moeten worden. Dit belemmert de performance bij grotere en complexere aanvragen.

Het Transactie Component was ontworpen om per domein en merk opgesplitst te worden. (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013, p. 35) Momenteel is er echter een enkel Component in gebruik door alle portalen. Domein- en merk-specifieke business logica wordt in de Facade (met behulp van een Logic Factory) verwerkt.

De logica achter de portalen doet aanroep op het Transactie Component als zijnde een DLL. Door de manier waarop de Componenten en hun afhankelijkheden geïmplementeerd zijn, is versionering niet mogelijk: er wordt steeds maar één versie van de DLL gebruikt. Ook is het niet mogelijk om parallelle aanroepen op de DLL te doen, wat performance verhindert.

4.2.2. Integratie van de backend services

De verschillende services welke de backend vormen zijn veelal oude stukken software welke op oude mainframes draaien. Ter beveiliging en vergemakkelijking van communicatie worden deze niet direct naar de buitenwereld aangeboden, maar ontsloten door een integratie-laag.

4.2.2.1. Functionaliteiten

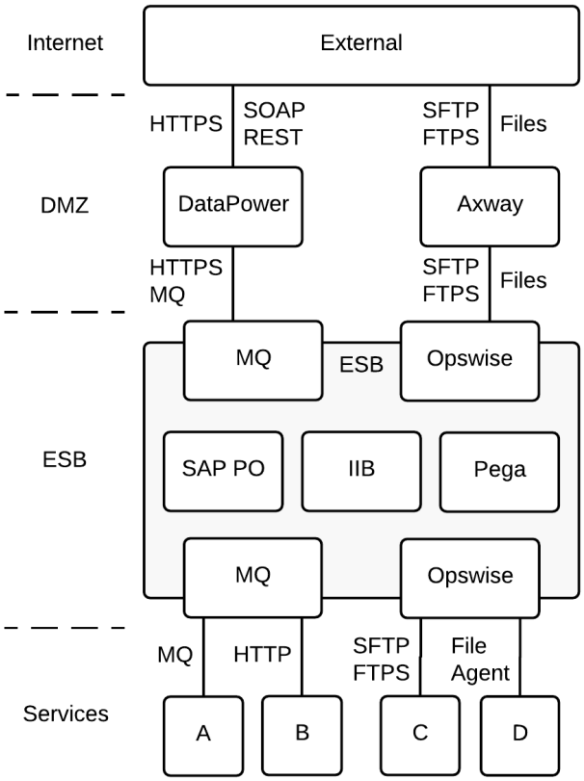
De integratie-laag faciliteert dus bij de communicatie met de services. Hierbij biedt het de volgende functionaliteiten: (Bril, 2012, pp. 11-13)

- **Transformeren**: Om communicatie met de (oude) services te vergemakkelijken vertaald de integratie-laag tussen verschillende formaten en protocollen.
- **Routeren**: De integratie-laag garandeert dat berichten bij de juiste (instantie van een) service terecht komen, en dat eventuele responsen bij de plek van aanroepen terug komen. Ook worden versimpelde interfaces aangeboden voor het aanroepen van services.
  Bij het routeren worden verschillende soorten berichten onderscheden:
  - **Fire and Forget (FF)**: Na versturen wordt geen antwoord verwacht.
  - **Request/Reply (RR)**: Na versturen wordt synchroon een antwoord verwacht.
- **Beveiliging**: Ter bescherming van de services vereist de integratie-laag een beveiligd communicatiekanaal en controleert het berichtinhoud.
- **Monitoring**: Gegevens worden bijgehouden over binnenkomende aanvragen, om rapportage te doen of direct op mogelijk kritieke situaties te reageren.

4.2.2.2. Implementatie

De integratie-laag bestaat uit verschillende sub-lagen, alleen opgebouwd uit softwarepakketten. Hun samenhang is in Figuur 6 weergegeven en daaronder verder toegelicht. (G. Bril, persoonlijke communicatie, 4 oktober 2016)

Figuur 6 – Versimpelde architectuur integratie-laag



- **Demilitarized Zone (DMZ)**: Losstaande infrastructuur, dient als proxy/firewall ter bescherming tegen aanvallen. Dwingt veilige communicatie-protocollen af.
    - **IBM DataPower**: Gateway voor berichtenverkeer.
    - **Axway**: Gateway voor bestandsverkeer.
- **Enterprise Service Bus (ESB)**: Berichtenbroker, transformeert en routeert berichten.
    - **IBM MQ**: Biedt Message Queues aan. Garandeert exactly-once aflevering van berichten. Kan prioriteren. Ziet alle berichten als platte data.
    - **Stonebranch's Opswise**: Framework voor het plannen en uitvoeren van taken, waaronder het beheren van bestandsverkeer.
    - **SAP Process Orchestration (PO)**: Broker voor de SAP services, transformeert berichten en routeert de resultaten.
    - **IBM Integration Bus (IIB)**: Generieke broker, transformeert berichten en routeert de resultaten. Stateless.
    - **Pega**: Business Process Manager tool. Logica is gedefinieerd als regels in Pega's database. Stateful.

- **Services**: Communiceren in hun eigen protocol en formaat, alle vertaling wordt door de ESB geregeld. Kunnen met elkaar communiceren via de ESB. Hebben eigen databases voor gegevensopslag.

De integratie van sommige services is verouderd en gebeurt niet conform de huidige standaarden die binnen Achmea worden gehanteerd.

4.2.3. Architectuurproblemen

De architectuur zoals deze momenteel in gebruik is bevat een aantal problematische aspecten, waardoor de in 4.1.3. genoemde wensen geblokkeerd worden. Deze aspecten en hun oorzaken zijn hieronder weergegeven. (Persoonlijke communicatie, 18 oktober 2016)

- **Sterke koppeling**
  - Veel onderlinge afhankelijkheden in de transactie-laag zorgen ervoor dat er te veel, te vaak getest moet worden, waardoor de Time To Market hoog is.
  - De transactie-laag is testbaar, maar door de sterke koppeling is het onduidelijk wat er precies getest wordt.
- **Versionering**
  - De Transactie Componenten zijn als DLL in de Global Assembly Cache (GAC) van SharePoint geïmplementeerd, waardoor versionering van de componenten niet mogelijk is. Hierdoor is hot deployment niet mogelijk.
- **Performance**
  - Caching van de portalen gebeurt in de session state en biedt ruimte tot verbetering. Een betere cache zou performance bevorderen.
  - Er kunnen momenteel geen parallelle calls gemaakt worden naar de transactie-laag, wat de performance beperkt.
  - Het aanroepen van services is langzaam. Dit vertraagt alle aanroepende applicaties.
- **Frontend code**
  - De programmering van sommige portalen is verouderd en verschilt van de huidige standaard, waardoor aanpassingen moeizaam gaan.
  - De frontend code bevat business logica, terwijl deze alleen portaal-specifieke code horen te bevatten.
- **Transactie Component code**
  - Bevat display logica, wat niet in het Transactie Component thuis hoort.
- **Services & service interfaces**
  - De service interfaces vereisen en geven te veel informatie. Ze zouden simpeler in gebruik moeten zijn.
  - De beschikbaarheid van meerdere versies van services levert extra werk en verwarring.
  - De naamgeving van de service interfaces is onduidelijk.

    - Mutaties gebeuren meestal via Fire and Forget berichten, waardoor er geen bevestiging van verwerking teruggestuurd wordt.
- **Kanaalonafhankelijke bediening**
    - Door de manier waarop de transactie-laag geïmplementeerd is en ontsloten wordt (als DLL) is kanaalonafhankelijke bediening niet mogelijk.

Deze problemen zijn uiteraard niet bewust in de architectuur terechtgekomen. Bij het herleiden van hun oorzaken komen drie punten sterk naar voren. (Persoonlijke communicatie, 25 oktober 2016)

- De **communicatie en afstemming** tussen ontwikkelaars van de services en ontwikkelaars van de applicaties die ze afnemen is onvoldoende. Hierdoor worden problemen niet snel genoeg opgelost en sluiten de service interfaces en de scenario's waarin zij gebruikt worden niet goed op elkaar aan.
- Wanneer **nieuwe eisen of architecturen** worden toegepast wordt (wegens de nodige ontwikkeltijd) niet alle code herschreven om hieraan te voldoen. Hierdoor is er nog code in gebruik welke verouderd en inflexibel is.
- **SharePoint**. De in het platform ingebouwde functionaliteiten worden binnen de transactie-laag amper gebruikt, maar de restricties die het met zich mee brengt blijven wel gelden. Dit verhindert ontwikkeling en het deployment proces.

4.3. Eisen en randvoorwaarden

Bij het ontwerpen van een doelarchitectuur speelt de gehele context van de architectuur een belangrijke rol. Dit betekent dat er, naast de eisen die aan softwarearchitectuur worden gesteld, ook rekening gehouden wordt met randvoorwaarden en de criteria die voor een architectuur en migratiepad gelden. Hieronder zijn deze drie belangrijke aspecten van de context uitgewerkt.

4.3.1. Geest van de architectuureisen

Er wordt een breed scala aan eisen gesteld aan alle aspecten van softwarearchitectuur. Om verzanding te voorkomen wordt er primair naar de geest van de eisen gekeken, welke met onderstaande punten op te sommen is. (Persoonlijke communicatie, 25 oktober 2016)

- Zo min mogelijk code schrijven, om de kans op bugs te verkleiner en de onderhoud te vergemakkelijken.
    - Hiertoe moet zo veel mogelijk herbruikbare, generieke code geschreven worden.
- Ontkoppeling is belangrijk, verschillende componenten moeten zo min mogelijk afhankelijkheden hebben.
- Er moeten makkelijk nieuwe componenten toegevoegd kunnen worden,
- De software moet simpel en robuust zijn.

- Er moet één entiteitenmodel gehanteerd worden zodat er een eenduidige manier van data-interpretatie is.
- De software moet veilig zijn.
- De frontend moet op de gebruiker toegespitst kunnen worden.

Gedetailleerdere eisen welke uit verschillende afdelingen voortkomen zijn terug te vinden in hoofdstukken 4.1.1 tot en met 4.1.4 van Bijlage C.

4.3.2. Randvoorwaarden

Hoewel de huidige architectuur problemen met zich mee brengt, bevat het ook aspecten welke wel naar behoren functioneren en het gewenste effect heb. Indien mogelijk moeten de onderstaande aspecten behouden worden. (Persoonlijke communicatie, 25 oktober 2016)

- Er worden generieke componenten en andere software-stukken gebruikt.
- Stubbing is makkelijk.
- De frontend is ontkoppeld.
    - De frontend kan onafhankelijk gedeployed worden.
- De architectuur is gelaagd.
- De transactie-laag is onafhankelijk van SharePoint: geschreven als generieke ASP.NET code, kan op andere systemen neergezet worden.
- Inversion of Control voor onder andere configuratie (hoewel dit soms onnodig wordt toegepast).
- De Facade levert meerdere voordelen:
    - Maakt het makkelijk meerdere services aan te roepen.
    - Verrijkt verkregen data.
    - Voert business logica uit.
- SharePoint's sandboxing functionaliteit wordt gebruikt voor het deployen van content.

Daarnaast moet de nieuwe architectuur ondersteuning bieden aan de onderstaande wijzigingen. (Persoonlijke communicatie, 25 oktober 2016)

- Service wijzigingen:
    - Nieuwe service (interface).
    - Gewijzigde service (interface).
    - Service (interface) vervalt.
- Web API wijzigingen:
    - Nieuwe operatie.
    - Gewijzigde operatie.
    - Verwijderde operatie.
- Gedrag van de (frontend) schermen verandert.
- Wijzigingen van pagina-inhoud en de presentatie (styling) daarvan.

Gezien de wens van de ontwikkelteams om agile te werken is het belangrijk dat de voorgestelde oplossing dit niet in de weg staat.

4.3.3. Migratiecriteria

De migratie naar de doelarchitectuur wordt beoordeeld op kosten en risico. Beiden moeten zo laag mogelijk zijn. De zaken die bij deze twee aspecten komen kijken zijn hieronder uitgewerkt.

Kosten zijn de middelen die besteed worden om de migratie plaats te laten vinden:

- **Tijd:** De lengte van de periode waarin de migratie uitgevoerd wordt, zowel voorbereidend werk als het migreren zelf.
- **Manuren:** De hoeveelheid tijd die aan de migratie besteed wordt door werknemers.
- **Opleiding:** Het trainen van ontwikkelaars om met de nieuwe architectuur te werken.
- **Software:** Softwarepakketten die aangeschaft worden ter behoeve van de nieuwe architectuur.
- **Infrastructuur:** Hardware infrastructuur die aangepast of uitgebreid moet worden ter behoeve van de nieuwe architectuur.

Risico is de kans op verhinderingen en nadelige effecten van de migratie:

- **Vertraging:** De migratie duurt langer dan verwacht.
- **Falen:** De migratie wordt niet voltooid, bijvoorbeeld door onvoorziene problemen of veranderde prioritering.
- **Downtime:** Het systeem is tijdens (delen van) de migratie niet beschikbaar.
- **Bugs:** De migratie introduceert ongewenste functionaliteit.

4.4. Doelarchitectuur en migratie

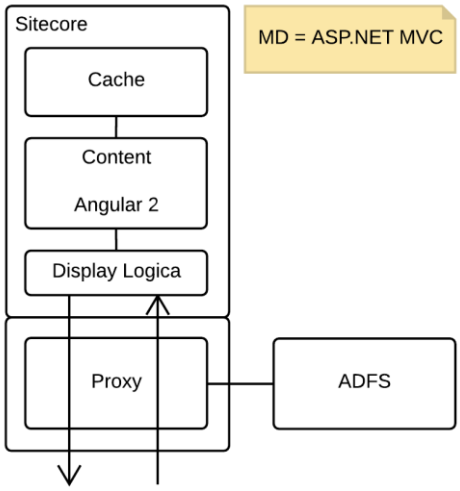
Om tot een doelarchitectuur en migratieplan te komen is, naast de inventarisatie van eisen en wensen, gekeken naar best practices welke binnen enterprise softwarearchitectuur van toepassing zijn. Ook is er met ontwikkelaars overlegd om hun wensen en ideeën concreet in beeld te krijgen. Met dit alles is rekening gehouden bij het ontwerpen van de doelarchitectuur. Een uitgebreidere toelichting en verantwoording is terug te vinden in Bijlage C.

4.4.1. Doelarchitectuur

Samen met medewerkers zijn verschillende mogelijke doelarchitecturen geïnventariseerd. Op basis van deze voorstellen is de architectuur uit Figuur 7 en Figuur 8 opgebouwd. Deze “combinatiearchitectuur” gebruikt verschillende voordelige aspecten uit de voorgestelde doelarchitecturen om een geheel te maken wat meer problemen oplost en sterker aan de eisen voldoet.

De wijzigingen welke door de doelarchitectuur aan de front- en backend worden gemaakt zijn generiek, en niet toegespitst op specifieke portalen of services. Dit zijn dan ook aanbevelingen. Per portaal en per service zal verder onderzocht moeten worden in hoeverre de aanbevelingen passend zijn.

Figuur 7 – Doelarchitectuur frontend



- **Portalen**, Sitecore (MD: ASP.NET MVC):
  - **Caching**: Geïmplementeerd middels Sitecore's caching functionaliteit, voor het context-bewust cachen van web pagina's en hun content.
  - **Angular 2**: Ter vervanging van Knockout JS. Realiseert de logica welke pagina's ondersteund.
  - **Display logica**: Compatibel met Sitecore. Bevat geen business logica meer. Zorgt dat pagina's bij het initiële laden al domein data bevatten, in plaats van dit pas na het laden via JavaScript (middels de web API) op te halen.
  - **Proxy**: Beveiligt de web API en is verantwoordelijk voor authenticatie.

4.4.1.1. JavaScript framework

Voor het ondersteunen van de web pagina's is door Achmea reeds onderzoek gedaan naar het overstappen op Angular 2. Hoewel hier al voor gekozen is, is deze keuze tijdens het onderzoek geverifieerd. In Tabel 2 zijn beide frameworks bekeken op basis van aan de eisen gerelateerde criteria. (R. ter Velde, persoonlijke communicatie, 17 november 2016)

Tabel 2 – Knockout JS tegenover Angular 2

Criteria	**Knockout JS**	**Angular 2**
Applicatiestructuur richtlijnen	Geen.	Aanwezig.
Programmeren op…	Lager niveau.	Hoger niveau.
Testbaarheid	Geen verbetering ten opzichte van standaard JavaScript.	Dependency injection vergemakkelijkt testen.
Toekomstvastheid	Geen bijzonderheden ten opzichte van standaard JavaScript.	Werkt goed samen met Web Components.
Populariteit (Google, z.j.)	Lager.	Hoger.

Angular 2 is de betere optie, op basis van bovenstaande criteria. De goede testbaarheid en richtlijnen die het stelt aan applicatiestructuur maken het beter geschikt voor enterprise toepassingen.

4.4.1.2. (Web)applicatie platform

Binnen Achmea is reeds gekozen om de portalen naar Sitecore te verhuizen en van SharePoint af te stappen. Deze keuze is tijdens het onderzoek geverifieerd. In Tabel 3 zijn de belangrijkste aspecten van de twee platforms tegen elkaar uitgezet.

Tabel 3 – SharePoint tegenover Sitecore

Criteria	**SharePoint**	**Sitecore**
Gemaakt voor	Collaboratie sites.	Web content management sites.
Context-bewuste caching	Nee.	Ja.
Ondersteuning voor mobiele websites	Nee.	Ja.
Marketing toepassingen	Geen.	Lokalisatie, personalisatie, analytics, A/B testing.
Ondersteuning	Grotere community, betere officiële ondersteuning.	Kleine community.
Populariteit (Google, z.j.)	Hoger.	Lager.

Sitecore biedt vele functionaliteiten die beter aansluiten op de huidige problemen en use cases (performance, testen, personalisatie) dan wat SharePoint biedt. Dit maakt het de betere keuze.

Sitecore's caching functionaliteit maakt het mogelijk om naast statische content ook klant-specifieke (context-afhankelijke) pagina's te cachen. Dit wordt gebruikt om een frontend cache te implementeren welke performance bij het laden van reeds opgebouwde pagina's verbeterd.

Het medewerkerdomein (MD) is een portaal wat alleen voor medewerkers toegankelijk is. Hierdoor zijn veel functionaliteiten welke Sitecore faciliteert, zoals personalisatie, analytics en (tot zekere hoogte) performance niet van belang. Dit maakt het mogelijk om MD als ASP.NET MVC applicatie te implementeren, in plaats van op Sitecore. In Tabel 4 zijn de voor- en nadelen gegeven van het implementeren van MD middels ASP.NET MVC. (B. Neijssen, persoonlijke communicatie, 22 november 2016)

Tabel 4 – Voor- en nadelen ASP.NET MVC voor MD

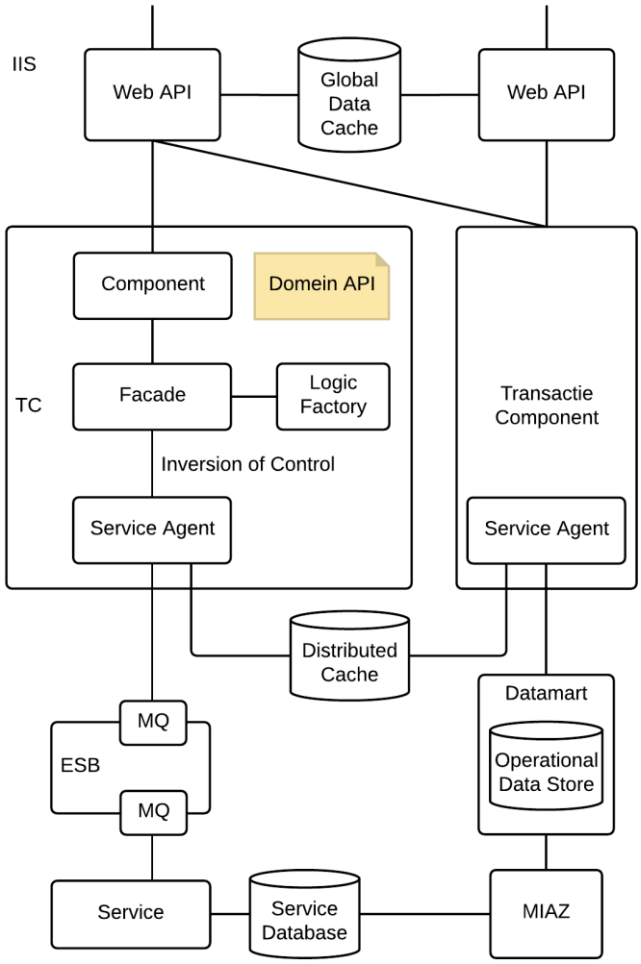
Voordelen ASP.NET MVC MD	Nadelen ASP.NET MVC MD
• Minder afhankelijkheden van en structuur opgedwongen door het platform.	• Maakt het moeilijker voor niet-technische gebruikers om aanpassingen te maken.

ASP.NET MVC heeft de voorkeur voor MD omdat gebruik van Sitecore geen significante voordelen met zich mee brengt. Door ASP.NET MVC te gebruiken blijft de implementatie flexibel, en het implementeren van functionaliteit die gemist wordt levert niet veel extra werk op. (R. ter Velde, persoonlijke communicatie, 23 november 2016)

4.4.1.3. Web API ontsluiting

Een web API proxy is gebruikt om de onderliggende web API's te ontsluiten, wat overeenkomt met de eisen welke vanuit Domein Architectuur gesteld worden. (Quakernaat & Thissen, 2014, pp. 21-22) (Zie ook hoofdstuk 4.1.2 in Bijlage C) Deze proxy is verantwoordelijk voor authenticatie, wat gebeurt middels Active Directory Federation Services (ADFS). Hier kunnen in de toekomst extra authenticatiemethoden aan toegevoegd worden. Voor het implementeren van deze proxy zijn reeds generieke componenten beschikbaar, voor zowel SharePoint als Sitecore. Het kost dus geen extra werk. (R. ter Velde, persoonlijke communicatie, 28 november 2016)

Figuur 8 – Doelarchitectuur middleware + backend



- **Transactie-laag**, IIS:
  - **Web API's**: Dunne laag bovenop de domein API (Transactie Componenten). Bevatten geen display of business logica, en doen alleen data uit de domein API aggregeren en aanbieden ter ondersteuning van frontend schermen. Portaal-generieke, data-gebaseerde interfaces. Worden geversioneerd.
  - **Transactie Componenten**: Structuur behouden. Opgesplitst per domein, bevatten domein-specifieke business logica. Vormen de domein API's, onafhankelijk van elkaar. Worden geversioneerd.
  - **Caching**: Ter behoeven van performance. Verschillende caches voor verschillende soorten data:
    - **Global Data Cache**: Voor generieke gegevens.
    - **Distributed Cache**: Voor service data, waaronder klant specifieke gegevens.
  - Generieke componenten (cross-cutting, domeinmodel, etc.) zijn behouden.

- **Integratie-laag en services**:
    - **ESB & MQ**: Blijven behouden. Er wordt verder onderzoek gedaan naar de mogelijkheid om meer Request/Reply berichten te gebruiken en daarmee MQ's te ontwijken.
    - **Services**: Blijven onveranderd.
    - **MIAZ**: Reeds bestaande software voor het aanbieden van versimpelde service data.
    - **Datamart**: Bevat onveranderlijke data uit de services en maakt deze makkelijk toegankelijk. Dit vermindert de werklading op de services. Performance voor zowel datamart- als service-aanvragen zal verbeteren.

4.4.1.4. Transactie-laag platform

De transactie-laag is van SharePoint naar een IIS server verplaatst, en zodoende als generieke ASP.NET applicatie geïmplementeerd. Door het afstappen van SharePoint worden de restricties die het oplegt verholpen, wat flexibiliteit ten goede komt. Het wegnemen van de huidige SharePoint afhankelijkheden zal niet veel kosten met zich mee brengen.

4.4.1.5. Web API ontwerp

De web API's zijn opgesplitst en gegroepeerd op basis van data. Dit is gedaan om RESTful API's aan te kunnen bieden, waardoor zij generiek te gebruiken zijn en niet op een enkele (interne) use case gericht zijn. Dit ligt in lijn met de Web API Solution Architectuur. (Achmea, 2014) (Zie ook hoofdstuk 4.3.1 in Bijlage C.)
Het spiegelen van de domein API is een overwogen optie, maar hier is niet voor gekozen omdat dit aansluiting met het uiteindelijke gebruik van de web API's niet waarborgt, domeinkennis vereist, en afhankelijkheid van het domein introduceert.

In overeenstemming met de eisen aan softwarearchitectuur (Quakernaat & Thissen, 2014, p. 14) bevat de web API alleen logica voor het aggregeren en presenteren van data. Hiertoe doet het aanroep op de domein API's om de nodige data te verzamelen.

4.4.1.6. Domein API ontwerp

Het Transactie Component is opgesplitst in verschillende domein API's, gegroepeerd per domein. Deze splitsing ligt in lijn met het originele ontwerp van de transactie-laag (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013, p. 35) en realiseert ontkoppeling.

Het is mogelijk om deze domein API's door een overkoepelende domein API te laten ontsluiten. Deze domein API zou de data aggregatie verantwoordelijkheid van de web API's overnemen. In Tabel 5 zijn de voor- en nadelen van het gebruik van een overkoepelende domein API gegeven.

Tabel 5 – Voor- en nadelen toepassen overkoepelende domein API

Voordelen overkoepelende domein API	Nadelen overkoepelende domein API
• Centrale plek voor data aggregatie. • Conform referentiearchitectuur.	• Minder controle in de web API's met betrekking tot de domein API versie die ze aanroepen. • Extra werk.

Een overkoepelende domein API is niet nodig, in ieder geval in eerste instantie. Er zijn geen grote, directe winsten te halen ten opzichte van data aggregatie in de web API's te laten plaatsvinden. Mocht het nodig blijken, dan kan een overkoepelende domein API altijd nog geïmplementeerd worden.

4.4.1.7. Domein caching

Caching wordt op meerdere plekken toegepast zodat er toegespitst kan worden op de verschillende soorten data. Hierdoor verbeterd performance, omdat data waar mogelijk uit de cache gehaald wordt in plaats van uit onderliggende logica.
Globale data, waarbij geen afhankelijkheid is van de identiteit welke bij de aanvraag hoort, wordt gecachet binnen de web API's zodat simpele aanvragen zonder verdere verwerking direct beantwoord kunnen worden.
Service data wordt door Transactie Componenten opgeslagen in een gedistribueerde cache. Meerdere TC's kunnen aanroep doen op dezelfde services. Door een gedistribueerde cache in te zetten kunnen ze opgehaalde gegevens (waaronder ook persoonsgebonden gegevens) onderling cachen. Dit verbeterd performance van service aanroepen.

4.4.1.8. Service data

MIAZ is een systeem wat binnen Achmea reeds in gebruik is, en als "data versimpelingssysteem" acteert. Het haalt data uit service databases en andere bronnen en zet dit om naar een logisch datamodel, waarbij alle technische attributen uitgefilterd worden om zo alleen functioneel relevante data aan te bieden.
Met bestaande infrastructuur kan aan MIAZ een zogeheten datamart gekoppeld worden. Dit is een database waarin een subset van de MIAZ data wordt geplaatst zodat applicaties hier aanroep op kunnen doen, inclusief handelingen als sortering en filtering. Dit is echter alleen nuttig voor onveranderlijke data, omdat een datamart niet continu up-to-date gehouden wordt.

In Tabel 6 zijn de voor- en nadelen van het gebruik van een datamart gegeven, ten opzichte van direct aanroep doen op de services. (C. Piso, persoonlijke communicatie, 21 november 2016)

Tabel 6 – Voor- en nadelen toepassen datamart

Voordelen datamart	Nadelen datamart
• Verbeterde performance: data kan direct worden opgehaald in plaats van aan de service opgevraagd en geaggregeerd. • Hogere beschikbaarheid van de data, minder afhankelijkheid van de bronsystemen. • Ontkoppeling: Bij veranderde datawensen (van applicaties) of veranderd datamodel (van services) alleen aanpassing in datamart nodig. • Verantwoordelijkheden voor data ontsluiten en data naar informatie omzetten worden expliciet gemaakt. • Meer technische flexibiliteit (bijvoorbeeld eenvoudiger sorteren, aggregeren en filteren van data). • Geen afhankelijkheid met de afdeling Integratie.	• Data zou niet real-time in de datamart terecht komen, waardoor voor actuele gegevens alsnog service aanroepen gedaan moeten worden. Dit dwingt een hybride-oplossing af, waarin de datamart niet voor alle data aanvragen toegepast wordt. • De verantwoordelijken voor de datamart zullen ook domeinkennis van de services moeten hebben om correct met de data om te kunnen gaan. Dit zorgt voor kennisverspreiding, wat interpretatie-risico met zich mee brengt. • Business regels worden verspreid. • Extra kosten voor het ontwikkelen van iets nieuws. (Software, opleiding.)

Een datamart wordt ingezet, omdat de voordelen opwegen tegen de nadelen. Vermindering van afhankelijkheden en verhoging van performance, beschikbaarheid en flexibiliteit liggen allen sterk in lijn met de huidige eisen en wensen, wat een datamart tot een goede oplossing maakt.

Er is verder onderzoek nodig om te kunnen zeggen welke services van een datamart kunnen profiteren. De hoeveelheid data welke middels een datamart beschikbaar gemaakt wordt moet opwegen tegen de kosten van implementatie en onderhoud. Hoe meer onveranderlijke data en hoe vaker deze opgevraagd wordt, hoe groter het voordeel dat een datamart kan leveren.

Een alternatief voor de datamart is het direct aanroep doen op de databases van services. In Tabel 7 zijn de voor- en nadelen van deze methode ten opzichte van een datamart weergegeven.

Tabel 7 – Voor- en nadelen directe service database aanroep

Voordelen service database aanroep	Nadelen service database aanroep
• Geen compleet nieuwe ontwikkeling nodig. • Veranderlijke data ook direct toegankelijk.	• Service logica moet naar transactie-laag gehaald worden. (Ter behoeve van data-interpretatie.) • Sterkere koppeling tussen service afnemers en de service data, waardoor aanpassingen nodig zijn bij verandering service datamodel.

Hoewel er kosten bespaard kunnen worden door service databases direct aan te roepen, schaadt dit de voldoening aan de eisen. Een verhoogde koppeling en complexiteit (door verplaatsing van logica, wat afbreuk doet aan de duidelijke gelaagdheid van de architectuur) zijn resultaten welke vermeden moeten worden. De voordelen wegen daar in dit geval niet tegenop.

4.4.1.9. Overige

Er moet verder onderzoek gedaan worden naar de mogelijkheid om service interfaces te verbeteren. Hierbij moeten waar mogelijk de interfaces versimpeld worden en middels Request/Reply berichten verstuurd worden. Uiteraard geld dit alleen voor de interfaces welke na toepassen van een datamart nog in gebruik zijn.

Overige zaken zijn behouden uit de huidige architectuur omdat deze voldoende blijven functioneren en het behalen van de doelstellingen niet belemmeren.

4.4.2. Migratieplan

Op het gebied van migreren van software zijn er twee uitersten: Big Bang en gefaseerde migratie.
Bij een Big Bang migratie wordt de nieuwe implementatie los van de huidige implementatie gerealiseerd, en wordt er op één moment (“de Big Bang”) overgestapt.
Tijdens een gefaseerde migratie, daarentegen, wordt de oude implementatie geleidelijk omgezet naar de nieuwe implementatie. Tijdens deze overstap blijft het systeem in gebruik.

Gefaseerde migraties hebben, wanneer het om enterprise systemen gaat, nagenoeg altijd de voorkeur. Gedurende de migratie kan aan nieuwe functionaliteiten en probleemoplossingen gewerkt blijven worden, dankzij de beschikbaarheid van het systeem kan de migratie indien nodig tijdelijk stil gelegd worden, en door het geleidelijk maken van aanpassingen is het risico veel lager.

Bij het bepalen van de juiste volgorde van de migratiestappen spelen meerdere factoren een rol:

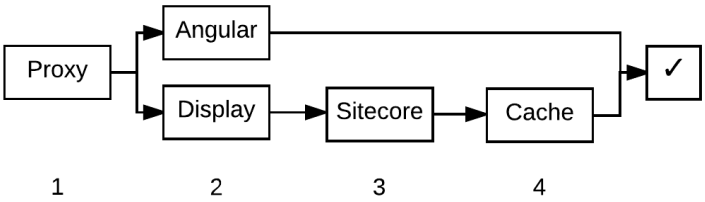
- **De Scrum backlog**: De taken welke op korte termijn gedaan zullen worden beïnvloeden de migratie. Het toevoegen van nieuwe functionaliteiten kan de complexiteit van de architectuur en code verhogen, waardoor de migratie meer tijd zal kosten. Hierdoor is de voorkeur om migratie uit te voeren voordat gerelateerde functionaliteiten worden geïmplementeerd.
- **Prioritering van problemen**: Afhankelijk van de urgentie van het oplossen van de verschillende problemen kunnen bepaalde delen van de migratie voorrang vereisen.
- **Opleiding**: Daar waar de doelarchitectuur sterk afwijkt van de huidige architectuur moeten de ontwikkelaars leren hoe hiermee gewerkt moet worden. Dit kan geleerd worden tijdens het implementeren van de veranderingen.
- **Agile**: Om aan de Agile werkwijze te voldoen moet er altijd een werkend systeem zijn, en zo min mogelijk branching in de codebase.

Hoe deze factoren de migratie precies beïnvloeden is binnen de context van het onderzoek slecht in te schatten, en zal door de medewerkers bepaald moeten worden. Er zijn migratiepaden opgesteld welke aanbevelingen maken over de volgorde en groepering van de verschillende wijzigingen, met als doel een flexibel plan aan te bevelen.

In Figuur 9 en Figuur 10 hieronder zijn de migratiepaden van respectievelijk de frontend en de middleware met backend weergegeven. Waar het migratiepad opsplitst kunnen verschillende stappen onafhankelijk van elkaar worden uitgevoerd. De volgorde waarin die stappen worden uitgevoerd hangt in dat geval af van de hierboven beschreven factoren.

4.4.2.1. Frontend

Figuur 9 – Migratiepad frontend



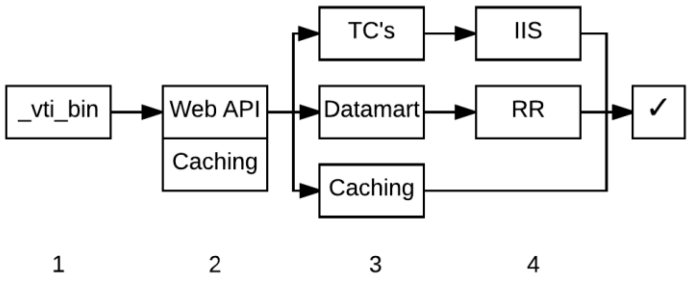
Tabel 8 – Migratiestappen frontend

Korte naam	Beschrijving
Proxy	Proxy voor toegang tot onderliggende web API's. Verantwoordelijk voor authenticatie. Kan zonder extra kosten geïmplementeerd worden omdat er reeds generieke componenten voor bestaan (voor zowel SharePoint als Sitecore). (R. ter Velde, persoonlijke communicatie, 28 november 2016)
Angular	Angular 2 volledig adopteren en alle JavaScript hiernaar overzetten.
Display	Display logica Sitecore compatibel maken (voor MD: ASP.NET MVC) en wegwerken van eventuele business logica. Dit zal geen langdurig of ingewikkeld werk opleveren. (B. Lijten, persoonlijke communicatie, 20 oktober 2016)
Sitecore	Verhuizen van de frontend naar Sitecore (voor MD: ASP.NET MVC).
Cache	Implementeren van de frontend cache op basis van Sitecore functionaliteit.

Allereerst wordt een proxy geïmplementeerd. Dit is een vereiste voor de realisatie van web API's, welke de frontend migratie van de middleware en backend migratie loskoppelen.
Gezien content en logica ontkoppeld blijven kunnen hun migratiestappen vervolgens onafhankelijk van elkaar worden uitgevoerd.

4.4.2.2. Backend

Figuur 10 – Migratiepad middleware en backend



Tabel 9 – Migratiestappen middleware en backend

Korte naam	Beschrijving
_vti_bin	Implementeren van web API's op basis van SharePoint's _vti_bin. Tijdelijke tussenstap tot het implementeren van "echte" web API's, wordt reeds aan gewerkt.
Web API	Implementeren van web API's middels het ASP.NET Web API 2 framework.
Caching	Implementeren van de Global Data Cache en Distributed Cache.
TC's	Opsplitsen van de Transactie Componenten en wegwerken van eventuele display logica.
IIS	Verhuizen van de transactie-laag van SharePoint naar een ASP.NET applicatie op een IIS server. Dit zal geen lastig of duur werk opleveren. (R. ter Velde, persoonlijke communicatie, 17 november 2016)
Datamart	Realiseren en gebruiken van een Datamart voor het ophalen van onveranderlijke service data.
RR	Service interfaces versimpelen en waar mogelijk Request/Reply berichten gebruiken die direct via de ESB gaan.

Allereerst wordt de interface van de web API's opgezet. Dit ontkoppeld de frontend migratie van de backend en middleware migratie.
De migratiestappen voor de Transactie Componenten en de service toegang kunnen onafhankelijk van elkaar uitgevoerd worden, omdat de Service Agents de interfaces voor service aanroepen consistent houden. Caching heeft geen invloed op functionaliteit en kan dus ook onafhankelijk uitgevoerd worden.

4.4.3. Probleemoplossing en voldoening aan de eisen

Om de waarde en toepasbaarheid van de ontworpen oplossing te beoordelen is gekeken naar de mate waarin het de problemen van de huidige situatie oplost en voldoet aan de eisen en richtlijnen welke vanuit Achmea gesteld worden.

4.4.3.1. Architectuur

De voorgestelde doelarchitectuur biedt oplossingen voor alle architectuurproblemen, en vervult daarmee de businesswensen die erdoor geblokkeerd werden. In Tabel 10 en Tabel 11 zijn deze oplossingen toegelicht.

Tabel 10 – Oplossingen architectuurproblemen

Architectuurprobleem	**Oplossing**
Koppeling	Bij het opsplitsen van de Transactie Componenten is koppeling weggenomen.
Versionering	Versionering van de Transactie Componenten is mogelijk.
Performance	Parallelle aanroepen op de Transactie Componenten zijn mogelijk, er is nieuwe en verbeterde caching toegepast en (delen van) de service data is beschikbaar via een datamart.
Frontend code	Is vernieuwd (Angular 2) en business logica is weggewerkt.
Transactie Component code	Display logica is weggewerkt.
Services & service interfaces	Waar mogelijk worden service interfaces vermeden door gebruik van een datamart. Er wordt verder onderzoek gedaan naar verbeteringsmogelijkheden van de interfaces die in gebruik blijven.
Kanaalonafhankelijke bediening	De web API's zijn portaal-generiek en platform-onafhankelijk aan te roepen.

Tabel 11 – Vervulling businesswensen

Businesswens	**Vervulling**
Performance	Zie Tabel 10.
Hot Deployment	Versionering van de domein API is mogelijk. Inversion of Control blijft toegepast.
Time To Market	Lagere koppeling en de mogelijkheid tot Hot Deployment resulteren in een kortere Time To Market.
Kanaalonafhankelijke bediening	Zie Tabel 10.
Customer journey	De web API's bieden een eenduidig punt voor het bijhouden van aanroepen.

Met deze oplossingen wordt ook nagenoeg volledig voldaan aan de eisen, te ondersteunen wijzigingen en te behouden aspecten, zoals toegelicht in Tabel 12, Tabel 13 en Tabel 14 respectievelijk.

Tabel 12 – Voldoening aan eisen

Eis	Toelichting
Herbruikbare, generieke code	Transactie Componenten bevatten alleen domein-specifieke logica, generieke componenten blijven bestaan voor overkoepelende logica.
Ontkoppeling	Transactie Componenten zijn opgesplitst en ontkoppeld.
Makkelijk nieuwe componenten toevoegen	Web API's zijn makkelijk te implementeren. Toevoegen van nieuwe (web of domein) API componenten heeft geen externe invloeden.
Simpel en robuust	Deels voldaan. Web API's vormen een simpele laag en ontkoppeling van de Transactie Componenten vermindert complexiteit. Het in gebruik nemen van een datamart introduceert echter nieuwe elementen welke eigen business logica implementeren, en geeft de Service Agents nieuwe verantwoordelijkheid.
Eén entiteitenmodel	Ongewijzigd.
Veilig	Proxy beveiligt de web API's. Andere veiligheidsstructuren blijven behouden.
Personalisatie	Sitecore bevat functionaliteit om personalisatie te vergemakkelijken.

Tabel 13 – Ondersteuning wijzigingen

Wijziging	Toelichting
Service wijziging	Datamart dient als ontkoppelingspunt van de services. Hoeft alleen aangepast te worden wanneer service wijzigingen plaatsvinden met betrekking tot "interessante" data.
Web API wijziging	Web API's worden geversioneerd.
Gedrag van (frontend) schermen	Wijzigingen in de frontend logica hebben geen impact op de rest van de architectuur en zijn dus onafhankelijk te maken.
Pagina inhoud en presentatie	Content kan (nog steeds) onafhankelijk gedeployed worden.

Tabel 14 – Behouden van aspecten

Aspect	Toelichting
Gebruik van generieke componenten	Ongewijzigd.
Stubbing is makkelijk	Web API's zijn makkelijk te stubben, de rest is ongewijzigd.
Ontkoppeling en onafhankelijke deployment frontend	Ontkoppeld door de web API's.
Gelaagde architectuur	Ongewijzigd.
Transactie-laag onafhankelijk van SharePoint	Niet langer op SharePoint geïmplementeerd.
Inversion of Control	Ongewijzigd.
Facade voor makkelijke service aanroep, data-verrijking, business logica	Ongewijzigd.
Sandbox voor content deployment	Niet voldaan. Gezien er geen SharePoint frontend meer gebruikt wordt kan er ook geen gebruik gemaakt worden van de SharePoint sandboxing functionaliteit.

Naast probleemoplossing en voldoening aan de eisen is het ook belangrijk dat de architectuur gedragen wordt. Betrokkenen zijn reeds bezig met kritisch nadenken over het voorstel en voegen hier eigen inzichten aan toe. (R. ter Velde, persoonlijke communicatie, 14 december 2016) De onderzoeksresultaten zijn meegenomen in gerelateerde discussies en planningen welke na afloop van de stage een rol blijven spelen. (E. de Haan, persoonlijke communicatie, 15 december 2016)

4.4.3.2. Migratie

Voor het migratieplan zijn kosten en risico zo laag mogelijk gehouden. Hoewel het accuraat inschatten van de kosten buiten de scope van het onderzoek valt is hieronder wel aangegeven welke kostenbronnen er zijn. Voor de risico's is genoemd hoe deze beperkt zijn.

Kosten

- **Tijd**: De duur van de migratie is afhankelijk van het aantal migratiestappen welke (waar mogelijk) tegelijkertijd worden uitgevoerd, en de hoeveelheid werk welke naast de migratie plaats vindt.
- **Manuren**: Het aantal manuren wat besteed moet worden om de migratie te voltooien is afhankelijk van het voordoen van onverwachte problemen. Zie ook de risico's hieronder.
- **Opleiding**: Er moeten verantwoordelijken komen voor de datamart. Zij moeten kennis hebben van de service data om deze op de juiste manier te

ontsluiten. (C. Piso, persoonlijke communicatie, 21 november 2016) Daarnaast moeten ontwikkelaars met de nieuwe architectuur leren werken, maar deze opleiding kan (deels) tijdens de implementatie plaatsvinden.
- **Software**: Voor Sitecore is reeds een licentie aangeschaft. IIS is reeds beschikbaar. De kosten voor SharePoint komen te vervallen wanneer hier volledig van afgestapt is. Software levert dus geen kosten op.
- **Infrastructuur**: Alle benodigde infrastructuur voor het realiseren van een datamart is reeds beschikbaar. (C. Piso, persoonlijke communicatie, 30 november 2016) Infrastructuur levert dus geen kosten op.

Risico's

- **Vertraging**: Verschillende delen van de migratie zijn van elkaar ontkoppeld. Hierdoor worden niet alle delen beïnvloed wanneer een enkele stap vertraging oploopt. Hierdoor blijft de impact van eventuele vertraging beperkt.
- **Falen**: Niet alle delen van de migratie zijn van elkaar afhankelijk, waardoor de kans klein is dat het geheel zal falen. Praktische haalbaarheid wordt na afloop van het onderzoek getest middels een proof of concept.
- **Downtime**: Downtime zal waarschijnlijk plaatsvinden bij overstappen tussen verschillende platform. Van SharePoint naar Sitecore bij de frontend, en naar IIS bij de middleware. Verdere downtime kan met behulp van versionering voorkomen worden.
- **Bugs**: Tijdens de migratie zal middels de huidige (en waar nodig vernieuwde) tests gelet worden op onbedoelde functionaliteit.

4.4.4. Procesverbetering

Naast architecturale verbeteringen zijn er in de procesvoering rondom de software ook verbetering mogelijk. Tijdens het onderzoek zijn best practices onderzocht welke hierbij kunnen helpen, om verdere ontwikkeling te ondersteunen.

4.4.4.1. Continuous Integration

Continuous integration (CI) is het frequent bouwen en testen en integreren van software, vaak meerdere malen per dag, en meestal geautomatiseerd. Het voorkomt dat oplevering veel tijd en moeite kost. M. Fowler (2006) beschrijft de volgende onderdelen als belangrijk voor CI:

- **Continuous integration server:** Een server die verantwoordelijk is voor het automatiseren van het CI proces.
- **Versiebeheer:** De software moet zich in een versiebeheersysteem bevinden, en hierbij alles bevatten wat nodig is om de software te bouwen. Er moet een CI branch zijn waar de CI server de software vandaan haalt. Dit kan ook de main branch zijn.
- **Automatische builds:** De software moet automatisch gebouwd worden. Als de software niet gebouwd kan worden moet dit automatisch gemeld worden.

- **Automatische testen:** Nadat de software gebouwd is moet deze automatisch getest worden. Ook hier moet falen automatisch gemeld worden.
- **Dagelijkse commits:** Door vaak naar de CI branch van het versiebeheer te committen worden conflicten tussen code van twee ontwikkelaars snel gevonden, en zijn deze makkelijker op te lossen.
- **Build trigger:** Elk commit naar de CI branch zou een automatische build moeten triggeren.
- **Kapotte builds fixen:** Wanneer een build niet succesvol voltooid kan worden, moet dit direct opgelost worden. Dit kan eventueel ook op de tests toegepast worden.
- **Snelle build:** Het bouwen en eventueel automatisch testen van de software moet binnen een redelijke hoeveelheid tijd gebeuren.
- **Test op een kloon van de productieomgeving:** De testen moeten zo goed mogelijk de werkelijkheid nabootsen.
- **Maak de meest recente versie toegankelijk:** Ter ondersteuning van het Agile proces.
- **Toegankelijke status:** Het moet voor iedereen makkelijk te zien zijn wat de status van het CI proces is.
- **Automatisch deployment:** De software automatisch op de nodige omgevingen uitbrengen is sneller en veiliger dan dit handmatig doen.

Op het punt “kapotte builds fixen” wordt CI bekritiseerd. Er wordt gesteld dat niemand een directe oplossing nodig heeft, en dat teams na enkele tijd de CI server foutmeldingen negeren. Om de oorzaak (kapotte builds) te verhelpen, wordt voorgesteld om nieuwe of aangepaste code automatisch te testen voordat deze in de CI branch van het versiebeheer opgenomen wordt. Wanneer een aanpassing er voor zou zorgen dat het bouwen of tests falen, wordt de code niet in de CI branch geaccepteerd. (Bugayenko, 2014)

4.4.4.2. Web API documentatie

Voor het documenteren van web API’s bestaat de OpenAPI Specification. (Open API Initiative, z.j.) Het gebruik van deze specificatie zorgt ervoor dat API documentatie door zowel mensen als machines uit te lezen is, zodat zij de geboden functionaliteiten kunnen ontdekken en begrijpen zonder naar de broncode te kijken. Dit maakt het makkelijker om de web API’s te gebruiken.
Een alternatief is RAML, de RESTful API Modeling Language (RAML Workgroup, z.j.), welke dezelfde functie als de OpenAPI Specification dient.

4.4.4.3. Migratie als architectuurwerk

Vanuit het oogpunt van de business levert een migratie op zichzelf geen directe waarde. Taken met betrekking tot nieuwe functionaliteiten zullen dan ook prioriteit krijgen over migratie-taken, wat kan leiden tot het nooit volledig afmaken van de migratie. Om de business tevreden te houden tijdens een migratie is het belangrijk om nieuwe functionaliteit te blijven leveren. Een gefaseerde migratie leent zich hier goed tot.

D. Leffingwell (2011, pp. 408-409, 423-424) geeft de volgende adviezen over het uitvoeren van architectuurmigraties:

- Maak architectuurwerk in uitvoering zichtbaar, om te voorkomen dat degenen die eraan werken te veel werk krijgen.
- Stel limieten aan de hoeveelheid architectuurwerk in uitvoering.
- Stuur op effectieve samenwerking tussen architecten en ontwikkelaars.
- Faciliteer een kwantitatieve basis om beslissingen te maken, zodat de juiste dingen in de juiste volgorde gedaan worden.
- Laat de implementatie van de migratie over aan bestaande interne ontwikkelteams.

4.4.4.4. Dogfooding

“Dogfooding”, ook wel bekend als “eating your own dog food”, omvat het interne gebruik van producten en services die ook naar de buitenwereld aangeboden worden.

Door medewerkers hun eigen producten en services te laten gebruiken, komen tijdens de ontwikkeling sneller problemen naar boven die anders pas door de eindgebruikers ontdekt zouden worden. Het dwingt testen en kwaliteitscontrole af, en zorgt dat het product aansluit bij echte gebruiksscenario’s. (Microsoft, z.j.) Daarnaast wordt hergebruik gestimuleerd: medewerkers binnen het ene domein willen toegang hebben tot dezelfde functionaliteiten als medewerkers uit andere domeinen. Dit is het makkelijkst te bereiken wanneer functionaliteit generiek wordt geïmplementeerd.

5. Conclusies & aanbevelingen

Er is onderzoek gedaan naar de context met een focus op de spelende problemen. Met de afbakening van de eisen en richtlijnen, en werkende met ideeën en wensen van ontwikkelaars is uitgekomen op een doelarchitectuur welke alle geïnventariseerde problemen oplost en aan nagenoeg alle eisen voldoet.

5.1. Doelarchitectuur & migratie

De doelarchitectuur implementeert de frontend in Sitecore en de middleware als ASP.NET applicatie op IIS. Hiermee zijn alle restricties van SharePoint weggenomen en komen er voor de frontend nieuwe functionaliteiten beschikbaar. Dit lost flexibiliteitsproblemen op en maakt personalisatie en betere frontend caching mogelijk.
De Transactie Componenten worden opgesplitst om koppeling weg te werken en versionering te realiseren.
Service data wordt toegankelijk via een datamart, en er wordt slimmere caching toegepast. Dit verhoogt performance.
Domeinfunctionaliteit wordt ontsloten door portaal-generieke, data-gerichte web API's om de frontend los te koppelen en kanaalonafhankelijke bediening te realiseren.

De migratie naar de doelarchitectuur wordt gefaseerd uitgevoerd om risico laag te houden.
Web API's worden als eerste migratiestap genomen om de migratie van de frontend los te koppelen.
Niet alle stappen zijn afhankelijk van elkaar, waardoor het migratiepad flexibel blijft. Hierdoor kunnen ontwikkelaars naar eigen inzicht prioriteiten stellen aan de verschillende stappen.

Met de architectuur en het migratieplan is ingestemd door de architecten en ontwikkelaars. Het vervolg na dit onderzoek is het maken van een proof of concept om aan te tonen dat de voorgestelde architectuur in de praktijk kan gaan werken en het formaliseren van de architectuur in de portalen roadmap.

5.2. Hoofdvraag

Dit alles samenvattend kan de hoofdvraag beantwoord worden:

> **Hoe kunnen de Zilveren Kruis portalen overstappen naar een architectuur welke gebruik maakt van web API's, om complexiteit en koppeling te verminderen?**

Door het opsplitsen van huidige componenten per domein en web API's als dunne laag daarop te implementeren. Deze overstap wordt gefaseerd gemaakt, zodat de migratie flexibel blijft en het risico laag.

5.3. Aanbevelingen

Voor de migratiestap omtrent de communicatie met de services is verder onderzoek nodig naar de service interfaces. Er moet onderzocht worden welke interfaces versimpeld kunnen worden om ze makkelijker in gebruik te maken en beter aan te sluiten op de werkelijke gebruiksscenario's.
Ook moet er naar het gebruik van Fire and Forget berichten gekeken worden. Een klacht die naar voren kwam was dat deze niet genoeg duidelijkheid boden, maar bij veel transacties is deze vorm van communicatie onvermijdelijk. Er moet onderzocht worden waar op dit gebied verbetering mogelijk is.

Een datamart brengt kosten met zich mee. Wanneer een service weinig onveranderlijke data aanbiedt zijn de voordelen van een datamart voor die service minimaal. Er moet verder onderzoek gedaan worden om te bepalen voor welke services een datamart voldoende winst levert om de kosten van implementatie en onderhoud te verantwoorden.

Ook wordt er procesverbetering aanbevolen om verdere ontwikkeling soepeler te laten verlopen.

- Continuous integration kan Time To Market verder verlagen door integratie te automatiseren.
- Om web API's te documenteren kan een open standaard gehanteerd worden, zodat de API's door zowel mensen als machines gelezen kunnen worden, wat de bruikbaarheid verhoogd.
- Bij het uitvoeren van een migratie is het belangrijk dit architectuurwerk goed in beeld te houden, en stel hier limieten aan. Dit voorkomt dat er te veel aan architectuur gewerkt wordt, voor de business blijven functionaliteiten immers belangrijker.
- "Eat your own dog food" door producten en services binnen hun ontwikkelafdelingen ook te gebruiken, om gebruiksproblemen sneller te ontdekken.

Om de scope van het onderzoek te beperken is grotendeels portaalgeneriek gewerkt en is er weinig naar implementatie gekeken. Dit betekend dat de voorgestelde oplossing niet binnen elke context optimaal zal zijn. Hierdoor moet bij het gebruik van de resultaten rekening gehouden worden met de verschillen tussen de portalen en andere implementatiedetails. De juiste keuzes, bij zowel architectuur als migratie, zullen aan de hand van de betreffende implementatie gemaakt moeten worden.

6. Evaluatie van procesgang

Om van gemaakte misstappen te leren en advies te kunnen leveren voor vergelijkbare projecten is het belangrijk om terug te blikken op de uitvoering van het onderzoek.

Tijdens de eerste maand van de opdracht werd voornamelijk aan de hand van documentatie gewerkt. Hoewel dit veel relevante informatie bood, veroorzaakte dit ook verzanding. Uit de documentatie alleen was het lastig de belangrijke en onbelangrijke details te onderscheiden, en er bleek veel domeinkennis nodig te zijn voor het correct interpreteren van en omgaan met de gedocumenteerde informatie.

Na overleg met de primaire bedrijfsbegeleider werd geconcludeerd dat het betrekken van collega's bij het onderzoeksproces cruciaal zou zijn voor het tijdig en succesvol uitvoeren van het onderzoek. Er werden meetings georganiseerd om vanuit het perspectief van de ontwikkelaars te kijken naar eisen en wensen, en om gezamenlijk ideeën op te doen over de mogelijkheden van de doelarchitectuur. Waar na de eerste anderhalve maand een achterstand op de planning was opgelopen, kon aan het begin van de derde maand een voorsprong geconstateerd worden.

Er is voldoende samengewerkt met ontwikkelaars en andere betrokkenen om het onderzoek uit te kunnen voeren en een gedragen architectuur op te leveren. Toch wordt er voor vergelijkbare opdrachten aangeraden om belanghebbenden nog nauwer bij het onderzoek te betrekken. Een probleemoorzaak welke uitgesproken werd was de slechte afstemming tussen verschillende partijen, iets wat bij een adviesopdracht niet mag gebeuren. Uiteraard brengen betrokkenen veel kennis met zich mee wat erg nuttig is, maar dit is het beste te gebruiken door de betrokkenen zelf wanneer er gezamenlijk gewerkt wordt richting een gedeeld einddoel.

Daarnaast was er in eerste instantie beoogd om meerdere doelarchitecturen én migratieplannen op te stellen en daaruit de meest geschikte combinatie als advies aan te bieden. Hoewel er wel meerdere mogelijke doelarchitecturen boven water zijn gehaald, is er een enkel concreet migratieplan opgesteld op basis van de gekozen doelarchitectuur. Hoewel het advies voldoet aan de eisen, en het migratiepad flexibel is, kan dit ervoor gezorgd hebben dat meer passende mogelijkheden misgelopen zijn.

7. Bibliografie

Achmea. (2014, maart 5). *Web API Solution Architecture.* Opgeroepen op oktober 27, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/Documents/Achmea%20Web%20API%20Solution%20Architecture_v1.docx

Achmea. (2015, januari 6). *Achmea Group BIP 2015-2017.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/010/strategieimit/Documents/GroupBIP%20-%206%20jan%202015%20-%20interne%20versie.pdf

Achmea. (2016, november 17). *Organogram Achmea IM&IT.* Opgeroepen op december 9, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/010/Documents/Organogram%20IMenIT%2020161117.pptx

Achmea. (2016, januari). *Speakernotes corporate presentatie Zilveren Kruis.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/007/Documents/Zo%20communiceren%20wij/corporate%20presentatie/DEF%20speakernotes%20corporate%20presentatie%20Zilveren%20Kruis.docx

Achmea. (z.j.). *Onze organisatie: Organogram.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/onzeorganisatie/Paginas/organogram.aspx

Bril, G. (2012, december 1). *Technische Domein Architectuur Integratie Services.* Opgeroepen op oktober 5, 2016, van AchmeaNet: https://achmeanet-sites.hosting.corp/com/10753/Architectuur/TDA - Integratie Services v1.0.doc

Bugayenko, Y. (2014, juli 21). *Master Branch Must Be Read-Only*. Opgeroepen op oktober 17, 2016, van Yegor256: http://www.yegor256.com/2014/07/21/read-only-master-branch.html

Fowler, M. (2006, mei 1). *Continuous Integration*. Opgeroepen op oktober 17, 2016, van MartinFowler.com: http://www.martinfowler.com/articles/continuousIntegration.html

Google. (z.j.). *Google Trends - angular, knockout*. Opgeroepen op november 22, 2016, van Google Trends: https://www.google.nl/trends/explore?cat=1227&q=angular,knockout

Google. (z.j.). *Google Trends - sharepoint, sitecore*. Opgeroepen op december 16, 2016, van Google Trends: https://www.google.nl/trends/explore?cat=1227&q=sharepoint,sitecore

Leffingwell, D. (2011). *Agile Software Requirements.* Westford, Massachusetts, United States: Pearson Education, Inc.

Microsoft. (z.j.). *About Early Technology Adoption (Dogfooding)*. Opgeroepen op november 11, 2016, van Microsoft TechNet: https://technet.microsoft.com/en-us/library/cc627315.aspx?f=255&MSPPError=-2147217396

Open API Initiative. (z.j.). *OpenAPI Specification*. Opgeroepen op oktober 19, 2016, van Open API Initiative: https://openapis.org/specification

Quakernaat, J., & Thissen, A. (2014, november 06). *Microsoft CC Referentie Architectuur.* Opgeroepen op oktober 3, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/Referentie%20Architectuur%20v2.0%20alpha%20-%20ter%20review.docx

RAML Workgroup. (z.j.). *RAML*. Opgeroepen op oktober 19, 2016, van RAML: http://raml.org/

van 't Klooster, R. (2013, februari 27). *Definitief Ontwerp Zorg Portalen.* Opgeroepen op oktober 3, 2016, van Hoofd team site voor het Zorg Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Generiek/Shared%20Documents/[02]%20Architectuur/Definitief_ontwerp_zorg_portalen.docx

van 't Klooster, R. (2014, februari 5). *cd TotalClass.* Opgeroepen op oktober 3, 2016, van TFS hosting: \tfs\MS1\Zorg.Klantdomein\DEV\Diagrams\Diagrams\TotalClass.classdiagram

8. Bijlagen

Bijlage A: Plan van Aanpak

Plan van Aanpak

Web API's bij Zilveren Kruis Achmea

Mark Staarink – 1618662
Software Engineering
Hogeschool Utrecht
Zilveren Kruis Achmea
12-10-2016

Samenvatting

De student Mark Staarink zal afstudeerstage lopen bij Zilveren Kruis Achmea, een zorgverzekeraar. De softwarearchitectuur die door het bedrijf wordt gebruikt om hun frontend (waaronder de klantwebsites en medewerker interfaces) te laten communiceren met de backend services (onder andere polis-management en klantbeheer) brengt enkele problemen met zich mee. Hierdoor kunnen vooruitstrevende doelstellingen als performance en flexibiliteit niet gehaald worden, en is het lastig voor derde partijen om van buitenaf met de services van het bedrijf te communiceren.

De wens is om de huidige architectuur aan te passen zodat de doelstellingen wel gehaald kunnen worden en de services via web API's af te nemen zijn, door zowel interne als externe partijen. Hiertoe zal de student onderzoek verrichten om tot een voorstel te komen wat zowel een nieuwe architectuur als een migratieplan bevat.

De hoofdvraag van het onderzoek luidt:

Hoe kunnen de Zilveren Kruis portalen overstappen naar een architectuur welke gebruik maakt van web API's, om complexiteit en koppeling te verminderen?

Om tot een antwoord op de hoofdvraag te komen zullen de volgende deelvragen beantwoord worden:

- Wat is de context van de opdracht?
- Hoe ziet de huidige situatie eruit?
- Hoe kan de nieuwe architectuur er uit gaan zien?
- Hoe kan van de oude naar de nieuwe architectuur gemigreerd worden?
- Welke architectuur met bijbehorend migratiepad kan aanbevolen worden?

Het onderzoek zal grotendeels plaatsvinden binnen het bedrijf. De literatuur en kennis die daar beschikbaar is zal een belangrijke rol spelen, maar externe bronnen zijn onmisbaar op het gebied van best practices en andere breed toepasbare adviezen.

Om de kwaliteit van het eindresultaat te bewaken zal de student regelmatig contact houden met de stagebegeleiders over de tussenresultaten van het onderzoek. Ook zal indien de tijd dit toelaat een proof of concept van de aanbevolen architectuur gemaakt worden.

Gezien het korte tijdsbestek van de opdracht zal op een hoog abstractieniveau gewerkt worden om langdurig detailwerk te voorkomen. Ook zal de focus gelegd worden op onderdelen van het onderzoek die de meeste waarde hebben voor het bedrijf, om de beperkte tijd zo effectief mogelijk te besteden.

Inhoudsopgave

A.1. Inleiding

Ter afstuderen bij de studie Software Engineering aan de Hogeschool Utrecht zal Mark Staarink (hierna genoemd “de student”) een afstudeerstage lopen bij Zilveren Kruis Achmea (hierna genoemd “het bedrijf”).

In dit document zal allereerst het bedrijf beschreven worden, evenals de situatie daarbinnen waaruit de opdracht is voortgekomen. De gewenste situatie zal besproken worden, en het benodigde onderzoek om daar te komen uitgelijnd. Hierbij wordt, naast de onderzoeksvragen, ook gekeken naar de planning welke bij het uitvoeren van het onderzoek aangehouden zal worden. Tot slot zal de aanpak besproken worden: de kaders en grenzen waarbinnen het project valt, de te gebruiken methoden en middelen, en het risicomanagement.

A.2. Context

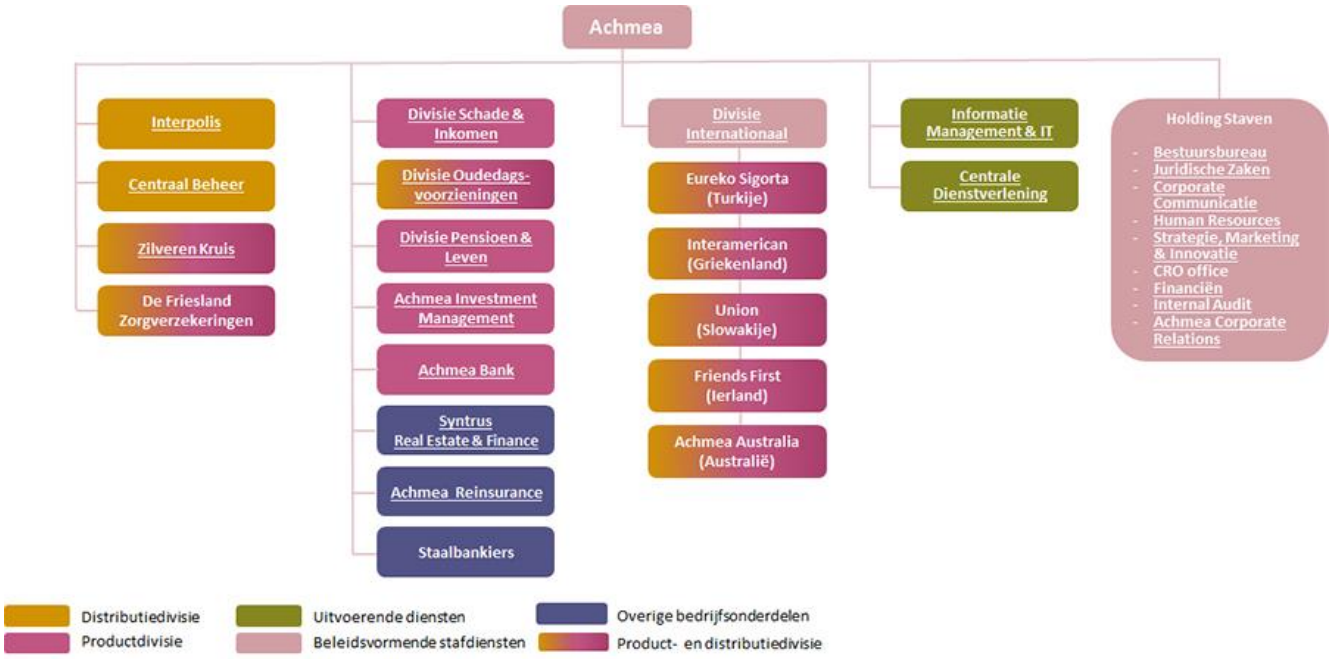
Voordat de opdracht en gerelateerde zaken besproken kunnen worden moet eerst een beeld van het bedrijf geschetst worden. In dit hoofdstuk zal naar de functies en structuur van het bedrijf gekeken worden, en zal beschreven worden hoe de student en de opdracht daarbinnen liggen.

A.2.1. Achmea & Zilveren Kruis

Achmea is een verzekeringsconcern, opgericht in 1811. Het omvat verschillende bekende verzekeringsmerken, van zorg- tot schadeverzekeringen. De grootste daarvan zijn Zilveren Kruis, Interpolis, FBTO, Centraal Beheer en Avéro Achmea. (Achmea, 2016, p. 2)

De doelstelling van Achmea is om de meest vertrouwde digitale verzekeraar te zijn. Hiertoe is tussen 2009 en 2011 gewerkt aan het op orde krijgen van het interne IT-landschap. Waar dit eerst gedecentraliseerd en geografisch verspreid was, is dit geconsolideerd en onder controle gebracht. Sinds 2012 wordt gefocust op het verminderen van de complexiteit in business en IT om beter ondersteuning te kunnen bieden aan toekomstige innovatie. (Achmea, 2015, pp. 9-10)

In Figuur 1 (Achmea, z.j.) wordt het organogram van Achmea gegeven. Hierin zijn aangegeven hoe de verschillende merken onder Achmea vallen, en welke ondersteunende divisies en diensten bij Achmea aanwezig zijn.

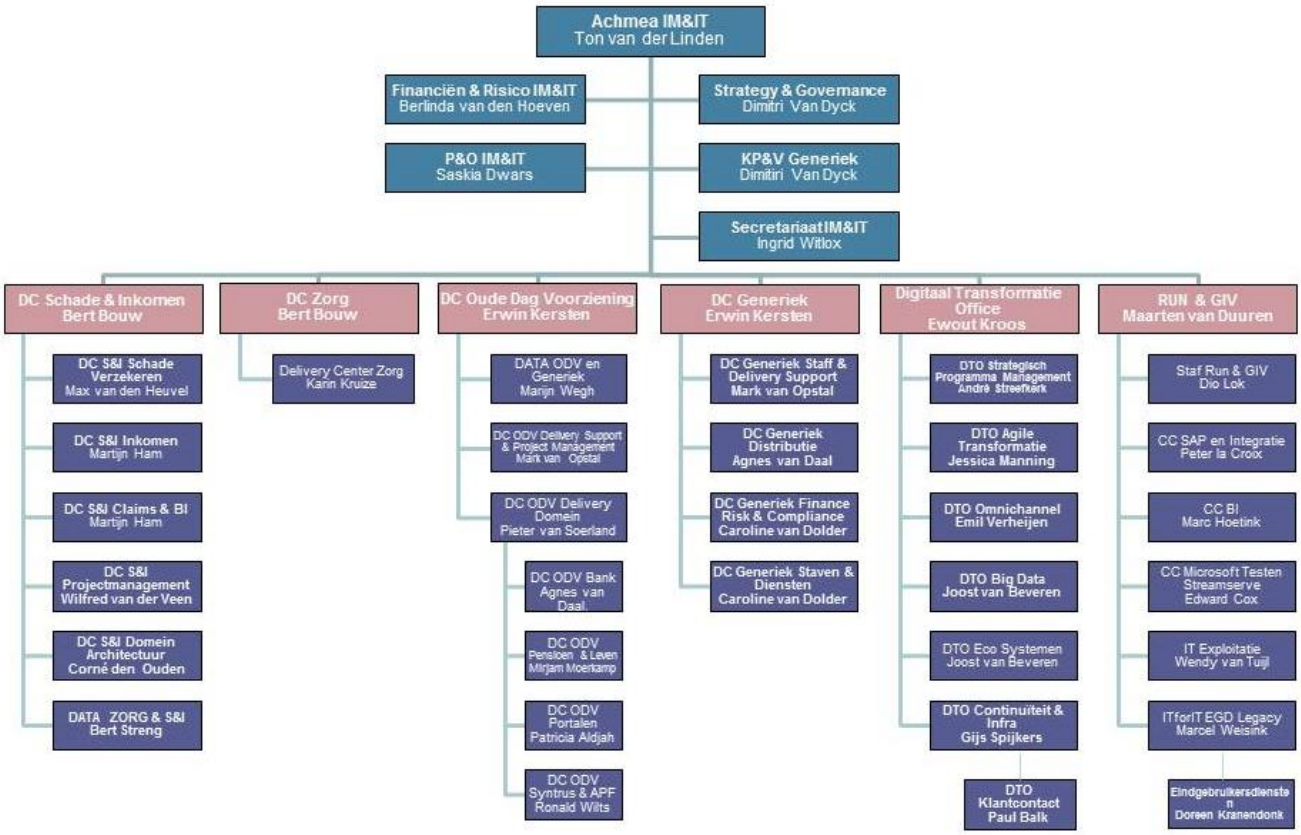


Figuur 1 – Organogram Achmea. Herdrukt van *AchmeaNet* website, door Achmea, z.j., opgehaald van https://organisatie.achmeanet.nl/onzeorganisatie/Paginas/organogram.aspx.

Zilveren Kruis is het grootste zorg-merk van Achmea. Het koopt jaarlijks voor 4,1 miljoen klanten zorg in (met een totaalwaarde van 11,6 miljard Euro), en verwerkt 2,5 miljoen wijzigingen in hun polissen. 2.800 medewerkers staan het bedrijf hierin bij. Hun callcenter handelt in een jaar 3,7 miljoen telefoongesprekken af, en beantwoord 402.000 mails en 180.000 chat-gesprekken. (Achmea, 2016, p. 3)

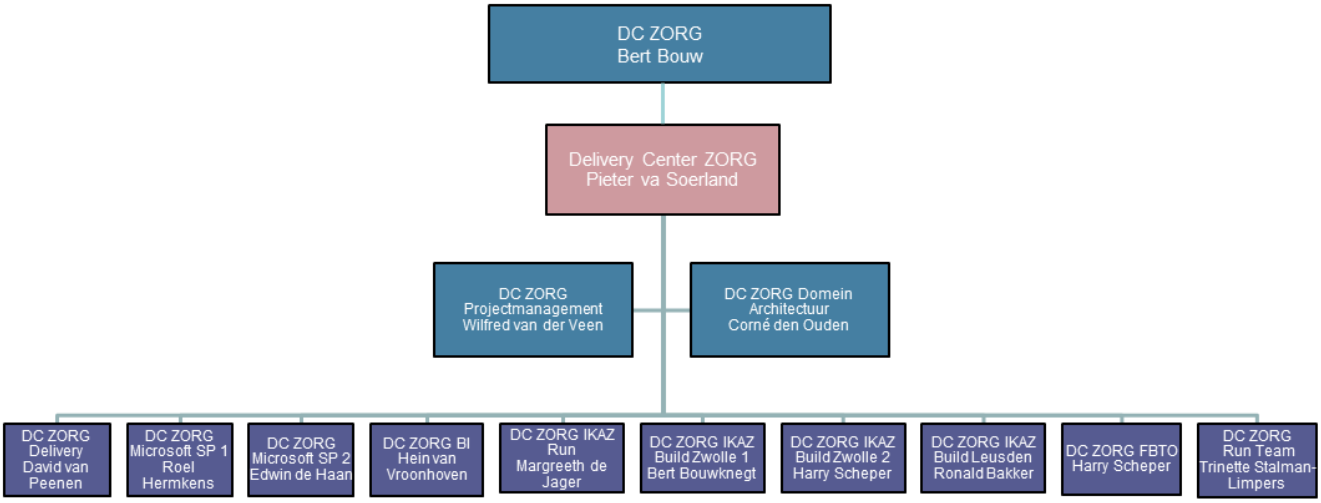
Sinds 2016 gebeuren de zorginkopen van Zilveren Kruis als zorgmodules. Deze worden opgesteld op basis van de manier waarop klanten zorg gebruiken. Hierbij wordt gekeken naar drie waarden die een klant kan ervaren: kwaliteiten van zorg, kosten van zorg, en kwaliteit van leven. (Achmea, 2016, p. 4)

De uitvoerende dienst Informatie Management & IT (afgekort IM&IT) houdt zich bezig met de informatiesystemen welke de werking van de andere afdelingen ondersteunen. Het organogram van IM&IT is in Figuur 2 (Achmea, 2016) afgebeeld.



Figuur 2 – Organogram IM&IT. Herdrukt van *AchmeaNet* website, door Achmea, 2016, opgehaald van https://organisatie.achmeanet.nl/010/Organogram/20160908%20Organogram%20Achmea%20IM%20IT.pptx.

Binnen IM&IT vallen meerdere partijen, waaronder de Delivery Centers. Zo ook Delivery Center Zorg, weergegeven in Figuur 3 (Achmea, 2016), welke de verschillende teams omvat die zich bezighouden met alle zorg-gerelateerde systemen. Denk hierbij zowel aan de klantwebsites als de achterliggende gegevensverwerking.



Figuur 3 – Organogram Delivery Center Zorg. Herdrukt van *AchmeaNet* website, door Achmea, 2016, opgehaald van https://organisatie.achmeanet.nl/010/Organogram/20160908%20Organogram%20Achmea%20IM%20IT.pptx.

A.2.2. Positie en verantwoordelijkheden

De student zal tijdens het uitvoeren van de opdracht als stagiair meelopen met Zorg Realisatie Team 1, welke binnen Delivery Center Zorg valt en zich bezig houdt met het ontwikkelen en onderhouden van de verschillende zorgportalen (web interfaces voor klanten en medewerkers). De taken van de student zullen beperkt blijven tot het werk wat voor de opdracht moet gebeuren, met daarnaast alle Scrum-gerelateerde taken zoals het bijwonen van de daily standups, sprint reviews, en dergelijke.

A.2.3. Relatie tot andere projecten

De infrastructuur waarbinnen de opdracht uitgevoerd wordt is volop in gebruik door andere projecten. Deze mag dan ook niet aangepast worden. Hetzelfde geldt voor de doelarchitectuur. Verder heeft de opdracht geen afhankelijkheden.

A.3. De opdracht

In dit hoofdstuk worden de verschillende aspecten van de opdracht besproken. Er zal gekeken worden naar de huidige situatie en de kwestie die daarin speelt, de oplossing die hiervoor gewenst is, en welke producten daarbij opgeleverd zullen worden.

A.3.1. Beperkende architectuur

Het bedrijf biedt een breed scala aan zogeheten portalen, frontend (web) interfaces welke zowel klanten als medewerkers in staat stellen om gegevens in te zien en aan te passen. Denk hierbij bijvoorbeeld aan de klantwebsite voor het beheren van verzekeringsgegevens, of de interface welke de afdeling Klant Contact gebruikt om klantgegevens in te zien.

De backend, waar al deze gegevens opgeslagen en verwerkt worden, bestaat uit ongeveer tachtig verschillende services. Deze kunnen, onder andere dankzij hun oude technologieën, niet direct met de moderne frontend praten. Om deze communicatie te faciliteren is de transactie-laag als middleware tussen de front- en backend gekomen, geïmplementeerd als DLL's welke de frontend aanroept. In de loop der tijd is er ook business logica in de transactie-laag terecht gekomen, waardoor die nu ook de verantwoordelijkheid draagt van zaken als data-validatie.

De huidige architectuur, zoals hierboven globaal beschreven, brengt een aantal problemen met zich mee. Hieronder vallen zaken als gelimiteerde performance en flexibiliteit, en een ongewenst grote opleveringstijd bij veranderingen. Daarnaast is het momenteel lastig voor derden om gegevens van Achmea af te nemen, gezien de transactie-laag die de gegevensuitwisseling met de backend omsluit niet ontworpen is om externe toegang te ondersteunen. Dit alles leidt ertoe dat doelen welke op basis van gebruikerswensen (waarbij de gebruikers zowel klanten als medewerkers zijn) gesteld zijn langzaam, deels of niet gehaald worden, wat uiteindelijk resulteert in een slechtere gebruikerservaring.

A.3.2. Architectuur met web API's

Gezien de problemen die ondervonden worden hun oorzaak hebben liggen in de huidige softwarearchitectuur, is de wens om de problemen op te lossen door middel van een architectuur-wijziging. Op deze wens is de opdracht gebaseerd, en daaruit volgen twee gerelateerde doelstellingen die hieronder beschreven zullen worden.

Er moet een nieuwe softwarearchitectuur ontworpen worden welke de problemen met de huidige architectuur oplost en voorkomt dat deze in de toekomst terugkeren. In de nieuwe architectuur zullen web API's een van de rollen van de transactie-laag uit de huidige architectuur moeten overnemen en uitbreiden: een interface bieden aan de frontend voor het ophalen en aanpassen van informatie. Waar de transactie-

laag momenteel alleen beschikbaar is voor de SharePoint frontend, zullen de web API's platform-onafhankelijk te gebruiken zijn. Zo zouden ook derde partijen informatie van het bedrijf af kunnen nemen. Daarnaast moet er voldaan worden aan andere eisen op basis van richtlijnen, randvoorwaarden en criteria welke tijdens het onderzoek verhelderd moeten worden.

Daarnaast moet een migratieplan opgesteld worden om van de huidige naar de nieuwe architectuur te komen. Hierbij zullen meerdere vooralsnog te onderzoeken criteria belangrijk zijn, waaronder de tijd die de migratie zal kosten en hoeveel van de bestaande code aangepast moet worden.

Voor beide doelstellingen zal gekeken worden naar meerdere mogelijke oplossingen. Vervolgens moet er gekeken worden welke combinatie van architectuur en migratieplan aan de meeste gestelde eisen voldoet om tot een uiteindelijk advies te komen. Hierbij zullen niet alle eisen even zwaar wegen.
Ook zal rekening gehouden moeten worden met de mate waarin de beslissende partijen te overtuigen zijn het voorstel aan te nemen en te gebruiken. Er zit meer waarde in een suboptimaal voorstel waar "ja" tegen gezegd wordt dan een volledig optimaal voorstel wat niemand toe zal passen.
Uiteindelijk zal een advies opgesteld kunnen worden. Wanneer dit advies opgeleverd is en door het bedrijf opgenomen wordt zal de opdracht geslaagd zijn.

Na afronding van het onderzoek zal het advies getest worden door middel van een proof of concept.

A.3.3. Producten

Aan het eind van de opdracht worden de volgende producten opgeleverd aan het bedrijf.

- **Een aantal mogelijke architecturen.** Bij elke architectuur zullen aan de hand van de eisen voor- en nadelen besproken worden.
- **Een aantal mogelijke migratieplannen.** Ook bij de migratieplannen zullen voor- en nadelen op basis van de eisen vermeld worden.
- **Een advies.** Kijkende naar de verschillende combinaties van architectuur en migratieplan, en wederom de eisen, zal een uitspraak worden gedaan over de optimale oplossing.
- **Proof of concept.** Het gegeven advies zal aangetoond worden te werken door middel van een proof of concept, waarbij minimaal één web API zal worden ontworpen en geïmplementeerd. (Uit te voeren ná de voltooiing van het onderzoek.)

De eisen en criteria die aan deze producten worden gesteld zullen door het onderzoek naar boven moeten komen.

Daarnaast zijn er enkele producten welke ter behoeve van het afstuderen aan de Hogeschool Utrecht opgeleverd moeten worden.

- **Het plan van aanpak.** Naar dit document zal de afstudeerdocent kijken om te besluiten of de opdracht definitief goedgekeurd wordt.
- **Het afstudeercontract.** Dit zal ondertekend moeten worden door de bedrijfsbegeleider, de docentbegeleider en de student.
- **De scriptie.** Legt het gedane onderzoek en de daarbij bevonden resultaten vast. De aan het bedrijf op te leveren producten zullen dus in de scriptie terug te vinden zijn.

A.4. Onderzoeksplan

In dit hoofdstuk zal de hoofdvraag met bijbehorende deelvragen gegeven worden. Deze vragen vormen de basis van het onderzoek dat uitgevoerd moet worden om de opdracht tot een succes te leiden. Tevens wordt de planning gegeven waarbinnen de deelvragen beantwoord zullen worden.

A.4.1. Onderzoeksvragen

De huidige softwarearchitectuur die binnen het bedrijf gebruikt wordt om de frontend met de backend te verbinden brengt een aantal problemen met zich mee. Om deze op te lossen zal overgestapt worden op een verbeterde, te ontwerpen architectuur. Hieruit volgt de hoofdvraag:

> **Hoe kunnen de Zilveren Kruis portalen overstappen naar een architectuur welke gebruik maakt van web API's, om complexiteit en koppeling te verminderen?**

Om tot een antwoord op deze hoofdvraag te komen zullen vijf "hoofddeelvragen" onderzocht moeten worden, elk met hun eigen deelvragen. Deze worden hieronder genoemd, met waar nodig een korte uitleg.
Het is mogelijk dat tijdens het onderzoeken van een deelvraag nieuwe vragen naar boven komen. Deze zullen in de uiteindelijke scriptie terug te vinden zijn, maar kunnen in dit plan van aanpak nog niet gedocumenteerd worden.
Om te integreren met de Scrum werkwijze van het team waarin de student meeloopt krijgt elke deelvraag een eigen taak. Tussen haken wordt achter elke deelvraag de initiële grove urenschatting van de bijbehorende taak gegeven.

A.4.1.1. Wat is de context van de opdracht?

- Hoe werkt een zorgverzekeraar? (6)
  Inzicht in de verschillende transacties die het bedrijf maakt zal helpen bij het ontwerpen van het front-facing deel van de architectuur.
- Wat doet Portalen Zilveren Kruis? (3)
  Om web API's te maken en groeperen zal bekend moeten zijn wat de functionaliteiten van de frontend zijn die ze aan zullen gaan roepen.
- Wat doen IKAZ, SAP, BI en Output? (9)
  Het is belangrijk om de grootste backend services te begrijpen, gezien zij (al dan niet indirect) door de web API's worden aangeroepen.
- Wat is het probleem dat opgelost moet worden? (7)
- Waar speelt het probleem? (3)
- Waarom is het een probleem? (3)
- Hoe groot is het probleem? (3)

A.4.1.2. Hoe ziet de huidige situatie er uit?

- Hoe ziet de huidige architectuur er uit? (12)

    - Hoe werkt de Microsoft stack bij Achmea?
    - Hoe werkt de integratie van de backend systemen?
- Hoe is de huidige architectuur ontstaan? (5)
- Welke achterliggende regels, eisen en ideeën vormen de huidige architectuur? (14)
  Om te voorkomen dat teveel de diepte in wordt gedoken zal het meeste ontwerp-werk op dit abstractieniveau plaatsvinden.
- Waar wordt er afgeweken van die regels, eisen en ideeën? (7)
- Waarom wordt er afgeweken van die regels, eisen en ideeën? (7)
  De nieuwe architectuur zal rekening moeten houden met de behoeften en uitdagingen die in de oude architectuur afwijkingen hebben veroorzaakt.
- Op welke punten voldoet de huidige architectuur niet aan de eisen en wensen? (7)
- Welke oplossingen heeft Achmea tot nu toe geïdentificeerd? (14)
- Wat zijn de best practices voor het oplossen van de problemen? (7)
- In welke mate worden deze best practices binnen Achmea toegepast? (7)

A.4.1.3. Hoe kan de nieuwe architectuur er uit gaan zien?

- Welke regels en eisen worden gesteld? (28)
    - Welke regels en eisen worden gesteld vanuit Enterprise Architectuur?
    - Welke regels en eisen worden gesteld vanuit Domein Architectuur?
    - Welke regels en eisen worden gesteld vanuit Informatie Management?
    - Welke regels en eisen worden gesteld vanuit Solution Architectuur?
- Hoe kan gezorgd worden dat de nieuwe architectuur gedragen wordt? (7)
  Het is belangrijk dat de nieuwe architectuur op zijn minst serieus overwogen wordt. Hierbij moet met sociale en intern-politieke factoren rekening gehouden worden.
- Welke randvoorwaarden aan de architectuur zijn reeds vastgelegd? (14)
    - Hoe moeten web API's bij Achmea werken?
    - Wat moet uit de huidige architectuur behouden worden?
- Welke wijzigingen moet de architectuur binnen welk tijdsbestek ondersteunen? (7)
  Niet alle wijzigingen hoeven even snel gemaakt te kunnen worden.
- Welke transacties zullen in eerste instantie via web API's plaatsvinden? (5)
- Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur design best practices? (4)
- Wat zijn de best practices op het gebied van enterprise architectuur design? (14)
  Om op praktische mogelijkheden voor de nieuwe architectuur te kunnen komen zullen ter kennisvorming relevante betrouwbare bronnen geraadpleegd moeten worden.
- Welke mogelijke architecturen zijn er? (35)
    - Waar moet de grens voor de web API's komen?
    - Waar moet de grens voor de domein API's komen?
    - Hoe kunnen de web API's gegroepeerd worden?

    - Hoe kan het afwijken van de bedoelde architectuur voorkomen worden?
- In welke mate voldoen de mogelijke architecturen aan de regels, eisen en randvoorwaarden? (14)
  Er wordt naar de voor- en nadelen van elke architectuur gekeken.

A.4.1.4. Hoe kan van de oude naar de nieuwe architectuur gemigreerd worden?

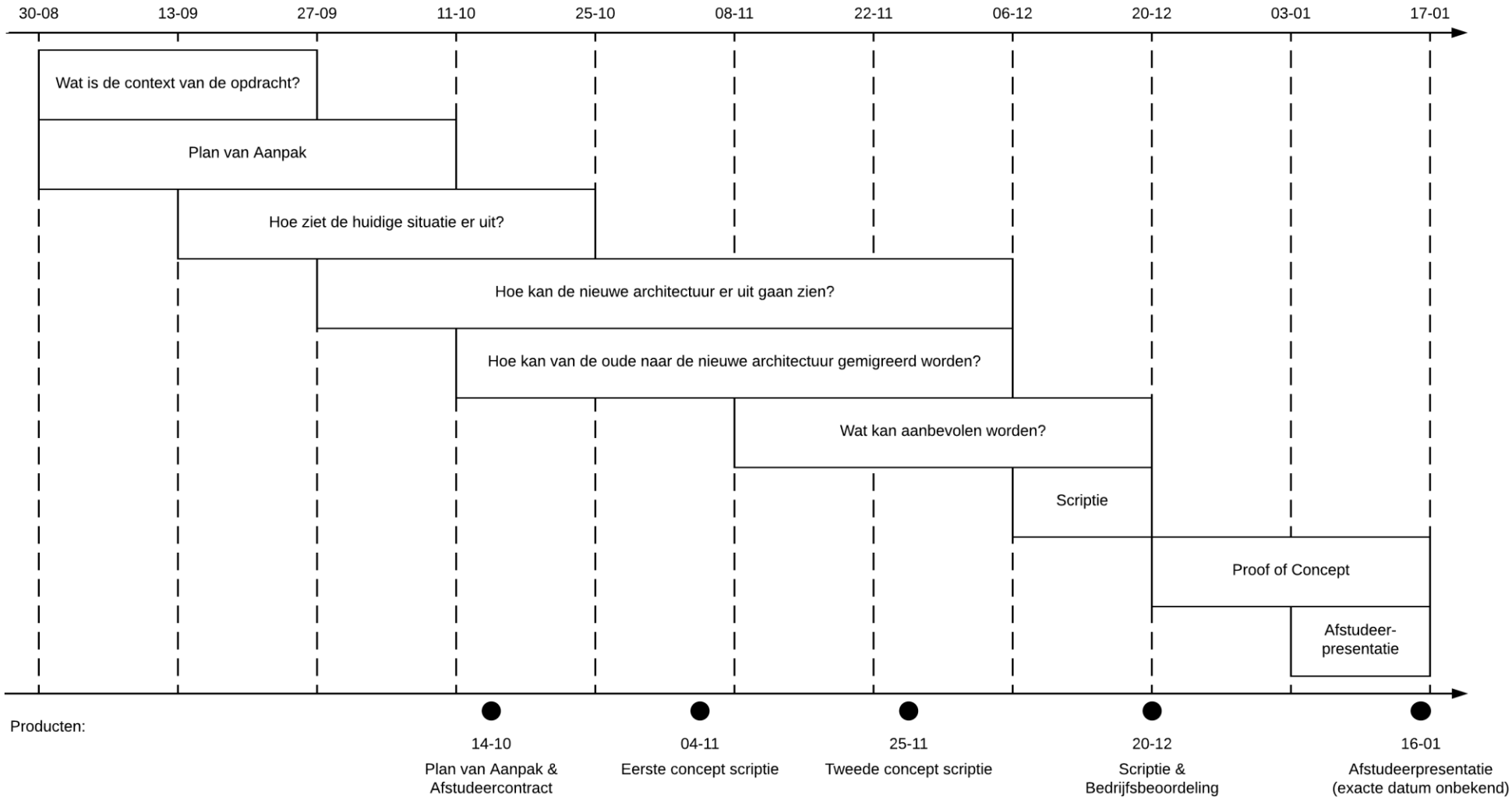
- Welke criteria worden gesteld aan de architectuur-migratie? (7)
- Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur migratie best practices? (4)
- Wat zijn de best practices op het gebied van enterprise architectuur migratie? (14)
  Wederom zullen ter kennisvorming relevante betrouwbare bronnen geraadpleegd moeten worden.
- Welke mogelijke migratiepaden zijn er? (28)
- In welke mate voldoen de mogelijke migratiepaden aan de criteria? (14)
  Er wordt naar de voor- en nadelen van elk migratiepad gekeken.

A.4.1.5. Welke architectuur met bijbehorend migratiepad kan aanbevolen worden?

- Welke migratiepaden zijn mogelijk voor welke architecturen? (14)
- Welke architectuur-migratiepad combinatie voldoet het beste aan de eisen? (14)
  Hierbij worden en voor- en nadelen van elke architectuur en elk migratiepad met elkaar samen gewogen.

A.4.2. Planning

Binnen het team waarin de student zal werken worden Scrum sprints van twee weken (tien werkdagen) gehanteerd. In lijn met de Scrum werkwijze zullen de “hoofddeelvragen” worden opgenomen als user stories, met hun bijbehorende deelvragen als taken. In de schematische planning in *Figuur 4* worden diezelfde tijdperiodes gebruikt voor het verdelen van taken over de beschikbare tijd. Deadlines voor de oplevering van producten worden als exacte data genoteerd. (De exacte datum van de afstudeerzitting is op het moment van schrijven onbekend.)



Figuur 4 – Planning

De opdracht zal uitgevoerd worden door de deelvragen grotendeels sequentieel te beantwoorden. Tijdens de eerste paar sprints zal de focus voornamelijk liggen op het bekend raken met de context van de opdracht. Op basis van die kennis kan het plan van aanpak opgesteld worden. Ook zal tijdig begonnen worden met het inventariseren van de huidige situatie. Zodra al die benodigde kennis vergaard is kan nagedacht gaan worden over de doelarchitectuur en mogelijke migratiepaden die genomen kunnen worden. Wanneer enkele antwoorden gevonden zijn kan al begonnen worden aan een uiteindelijke aanbeveling.
Er wordt beoogt na het voltooien van het onderzoek een proof of concept te maken voor de gemaakte aanbeveling. De resultaten hiervan zullen opgenomen worden in de presentatie die bij de afstudeerzitting gegeven wordt.

Het is de bedoeling dat aan de planning wordt gehouden, ook wanneer nog niet alle non-essentiële deelvragen volledig beantwoord zijn. Zo zou het bijvoorbeeld acceptabel zijn om deelvragen uit "*Wat is de context van de opdracht?*" deels onbeantwoord te laten ter behoeven van het onderzoek richting de nieuwe architectuur. Dit wordt gedaan om vertraging te voorkomen.
Daardoor wordt het ook belangrijk om gaandeweg te documenteren, en dit niet pas te doen wanneer een compleet antwoord gegeven kan worden. Hierdoor zal in de tijdsplanning het schrijven van de onderzoekscriptie samenvallen met het onderzoek zelf. De tijd die voor een deelvraag gepland staat zal zowel het onderzoek als de documentatie van het onderzoeksresultaat beslaan.

A.5. Aanpak

Om het project en bijbehorend onderzoek doeltreffend uit te kunnen voeren zal met enkele zaken rekening gehouden moeten worden. De grenzen van het project waarbinnen het werk moet blijven moeten duidelijk zijn. Er moet bekend zijn welke werkwijzen en middelen gebruikt zullen worden, evenals de kennisgebieden waar aan de opdracht relevante informatie te vergaren is. Daarnaast is het belangrijk om vast te bepalen hoe de kwaliteit van het gedane werk gewaarborgd wordt, en zullen risico's verminderd moeten worden.

A.5.1. Projectgrenzen

In het ontwerpen van de nieuwe architectuur wordt veel vrijheid gegeven, maar niet alles is toegestaan of mogelijk. Componenten uit de backend worden gegeven "as-is" en kunnen niet veranderd worden. De frontend staat, op de communicatie met de middleware na, ook vast. Daarbuiten mag vrij gedacht worden over architecturen, maar de eisen zullen daarbij een belangrijke rol blijven spelen.

De op te leveren architectuurontwerpen hoeven niet tot op implementatieniveau uitgewerkt te zijn. Wel moet voldoende detail aanwezig zijn zodat de architecturen gerealiseerd kunnen worden door de verantwoordelijke ontwikkelafdeling, zonder dat er onzekerheid ontstaat over hoe iets precies geïmplementeerd moet worden.

A.5.2. Onderzoeksmethoden en middelen

De antwoorden op een groot deel van de deelvragen zijn binnen het bedrijf te vinden. Literatuuronderzoek van de beschikbare documentatie op het gebied van onder andere richtlijnen en standaarden zal nuttige informatie opleveren. Ook zullen meetings met collega's een belangrijke rol spelen. Niet alleen voor het vergaren van impliciete kennis, maar ook het bediscussiëren van aandachtspunten en ideeën. Een nauwere samenwerking met collega's zal sneller leiden tot een gedragen voorstel.

Voor context-generieke informatie, zoals best practices op een bepaald gebied, ook buiten het bedrijf gekeken worden. Er zal gezocht worden naar betrouwbare bronnen welke een overzicht kunnen geven van veelvoorkomende problemen en de oplossingen daarvoor, om vervolgens deze generieke kennis toe te passen tijdens het ontwerpen. Ook kan gekeken worden naar de oplossingen van andere bedrijven, indien deze openlijk gepubliceerd zijn.

A.5.4. Communicatie

Binnen de opdracht speelt communicatie met de betrokkenen een belangrijke rol in zowel onderzoekuitvoering als risicovermindering. Hieronder worden de voornaamste betrokkenen, hun rollen binnen de opdracht, en hun persoonsgegevens gegeven.

Edwin de Haan
Email: *edwin.de.haan@achmea.nl*
Tel. Nr.: +31 6 30806216
Primaire bedrijfsbegeleider. Zal de student assisteren in het opzetten en uitvoeren van de opdracht.

Rowan ter Velde
Email: *rowan.ter.velde@achmea.nl*
Tel. Nr.: +31 6 18329530
Secundaire bedrijfsbegeleider. Werkt binnen het team waar de student in meeloopt.

Rob van ’t Klooster
Email: *rob.van.'t.klooster@achmea.nl*
Tel. Nr.: +31 6 51763301
Secundaire bedrijfsbegeleider.

Communicatie met de bedrijfsbegeleiders zal op zijn minst voortgangsbesprekingen en controle van producten omvatten. Dit zal voornamelijk via meetings gebeuren. Indien dit niet mogelijk is kan via email of instant messaging contact gevonden worden. Uiterlijk één week voor de oplevering van de producten (maar liever eerder en vaker) worden deze bij de bedrijfsbegeleiders neergelegd ter controle. Eventuele feedback kan dan nog voor de oplevering verwerkt worden.

Frank Verbruggen
Email: *feamar@icloud.com*
Tel. Nr.: +31 6 12202274
Docentbegeleider. Beschikbaar voor vragen over de gang van zaken rondom het afstuderen, hulp bij het uitvoeren van de opdracht en tussentijdse controle van documenten.

Communicatie met de docentbegeleider zal grotendeels beperkt blijven tot email en instant messaging. Documenten worden via het OnStage systeem ingeleverd, zodat deze zowel bij de afstudeeradministratie van de Hogeschool Utrecht als de stagedocent terecht komt. Hieronder vallen het plan van aanpak, de concept-scripties en de uiteindelijke scriptie.

Het afstudeerleerteam
Gedurende het afstuderen zal student deelnemen aan een afstudeerleerteam. Dit team bestaat uit een zestal studenten welke feedback kunnen geven op elkaars

afstudeerdocumenten. Hiertoe worden meerdere meetings op de Hogeschool Utrecht georganiseerd, waarbij ook een begeleidende docent aanwezig zal zijn.

Mark Staarink
Email: *mark.staarink@student.hu.nl*
Tel. Nr.: +31 6 43225570
De student.

A.5.5. Theoretisch kader

Betreft generieke kennis over methodieken en concepten bieden de hieronder gegeven weblinks relevante informatie. Waar mogelijk zijn de officiële partijen en instituten van het onderwerp als bron gebruikt.

- *http://www.demo.nl/*
  Theorie over DEMO, "Design & Engineering Methodology for Organizations", te gebruiken bij het onderzoeken van de processen binnen het bedrijf.
- *http://www.iaria.org/conferences2012/filesICSNC12/Normalized%20Systems%20Towards%20Designing%20Evolvable%20Modular%20Structures.pdf*
  Introductie tot Normalized Systems, een theorie voor het ontwerpen en ontwikkelen van evolueerbare informatiesystemen.
- *http://www.martinfowler.com/articles/continuousIntegration.html*
  Beginpunt voor best practices op het gebied van continuous integration.
- *https://en.wikipedia.org/wiki/Adoption_(software_implementation)*
  Overzicht van verschillende bekende methodieken voor het overstappen tussen informatiesystemen, te gebruiken als introductie.
- *http://www.scrumguides.org/*
  Definitie en doelstellingen van de Scrum methodiek.

A.5.6. Kwaliteitscontrole

Naast het toetsen van de bedachte oplossingen tegen de vastgestelde eisen zal er nauw contact gehouden worden met stagebegeleiders en andere belanghebbenden. regelmatig bespreken van de voortgang van het project en de onderzoeksresultaten zal er voor zorgen dat wanneer een onderdeel van het project de verkeerde kant op gaat dit tijdig opgevangen en bijgestuurd kan worden.

Daarnaast zal indien de tijd het toelaat een proof of concept gemaakt worden om aan te tonen of de gemaakte aanbeveling in de praktijk kan werken.

A.5.8. Risico's

Om de kans op slagen van de opdracht te maximaliseren zullen de risico's geïnventariseerd en voorkomen moeten worden. In *Tabel 1* wordt een samenvatting gemaakt van de voorziene risico's, hun oorzaak, en de maatregelen die getroffen zullen worden om hun kans van voorkomen en/of hun impact te verkleinen.

Oorzaak	**Risico**	**Maatregel**
De architectuur waarmee in de opdracht gewerkt wordt is erg omvangrijk.	Het overzicht en de tijd raken snel kwijt wanneer naar de details gekeken wordt.	Waar mogelijk wordt op een hoog abstractieniveau gewerkt. Zo wordt bijvoorbeeld gekeken nar de regels en principes achter de architectuur, in plaats van de implementatie-details die daaruit voort zijn gevloeid.
De opdracht moet binnen een korte tijdsperiode uitgevoerd worden.	De opdracht kan niet binnen de beschikbare tijd volledig worden afgerond.	De voortgang wordt regelmatig besproken met de bedrijfsbegeleiders, om hun op de hoogte te houden van de voortgang, zodat bijgestuurd kan worden wanneer de tijd niet optimaal benut wordt of een verkeerde richting ingeslagen wordt.
		De focus wordt gelegd op deelvragen wiens antwoord voor het bedrijf de meeste waarde heeft. Hierdoor wordt de kans vergroot dat een bruikbaar resultaat geleverd wordt, ook al is dat resultaat (volgens de initiële planning) niet compleet.
	Er wordt veel tijd verspilt met het wachten op communicatie.	Wanneer een contactpersoon (zonder hier van tevoren voor te waarschuwen) niet binnen enkele werkdagen reageert dan zal dit bij de contactpersoon aangekaart worden. Mocht dit vaker voorkomen dan wordt geëscaleerd en/of een alternatief contactpersoon gezocht.
Gedurende de opdracht zal veel en langdurig op de computer gewerkt worden.	Symptomen van RSI treden op en verhinderen de student bij het uitvoeren van de opdracht.	Bij het innemen van een werkplek zal de student bureau- en stoelhoogte aanpassen om een goede werkhouding te accommoderen. Daarnaast zal er geprobeerd worden om regelmatig korte pauzes te nemen. Mochten klachten zich voordoen en/of verergeren, dan zal dit gecommuniceerd worden met de bedrijfsbegeleiders.
De student heeft eerder in zijn carrière last gehad van RSI.		

Opslagmedia kunnen corrupt raken of op andere manier falen.	Bij dataverlies raken de reeds gemaakte documenten verloren.	De student zal meerdere (digitale) kopieën van de documenten van de opdracht bewaren, op verschillende locaties. • Persoonlijke laptop • Persoonlijke laptop backup • Bedrijfslaptop • Persoonlijke server • Persoonlijke flash drive Deze kopieën zullen dagelijks up-to-date gehouden worden. Deze redundantie zal dataverlies voorkomen.

Tabel 1 – Risico's met oorzaak en maatregelen

A.6. Persoonlijke ontwikkeling

Binnen en rondom het project zijn er meerdere uitdagingen en blokkades te overkomen, op zowel persoonlijk als professioneel gebied.

De aanpak van de opdracht vereist veel communicatie met collega's uit het team en andere betrokkenen. Dit soort werkvormen zijn natuurlijk veel geoefend gedurende de studie, maar studieopdrachten zullen nooit de werkelijkheid na kunnen bootsen. Hierbij spelen mentale blokkades ook een rol. Onzekerheid op dit gebied mag het voordelig gebruiken van deze werkvorm niet in de weg staan.

De opdracht richt zijn bijna exclusief op theoretisch onderzoek en architectuurwerk, in tegenstelling tot uitvoerend werk (programmeren). Hier heeft de student zowel minder passie als minder ervaring liggen. Dit, in combinatie met de andere uitdagingen, kan de opdracht stressvol maken. De motivatie voor en het vertrouwen in de opdracht mag hierdoor niet beïnvloed worden.

A.7. Bibliografie

Achmea. (2015, januari 6). *Achmea Group BIP 2015-2017.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/010/strategieimit/Documents/GroupBIP%20-%206%20jan%202015%20-%20interne%20versie.pdf

Achmea. (2016, september 8). *Organogram Achmea IM IT.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/010/Organogram/20160908%20Organogram%20Achmea%20IM%20IT.pptx

Achmea. (2016, januari). *Speakernotes corporate presentatie Zilveren Kruis.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/007/Documents/Zo%20communiceren%20wij/corporate%20presentatie/DEF%20speakernotes%20corporate%20presentatie%20Zilveren%20Kruis.docx

Achmea. (z.j.). *Onze organisatie: Organogram.* Opgeroepen op oktober 4, 2016, van AchmeaNet: https://organisatie.achmeanet.nl/onzeorganisatie/Paginas/organogram.aspx

Bijlage B: Onderzoeksvragen

Wat is de context van de opdracht?

- Hoe werkt een zorgverzekeraar?
- Wat doet Portalen Zilveren Kruis?
- Wat doen IKAZ, SAP, BI en Output?
- Wat is het probleem dat opgelost moet worden?
- Waar speelt het probleem?
- Waarom is het een probleem?
- Hoe groot is het probleem?

Hoe ziet de huidige situatie eruit?

- Hoe ziet de huidige architectuur eruit?
    - *Hoe werkt de Microsoft stack bij Achmea?*
    - *Hoe werkt de integratie van de backend systemen?*
- Hoe is de huidige architectuur ontstaan?
- Welke achterliggende regels, eisen en ideeën vormen de huidige architectuur?
- Waar wordt er afgeweken van die regels, eisen en ideeën?
- Waarom wordt er afgeweken van die regels, eisen en ideeën?
- Welke oplossingen heeft Achmea tot nu toe geïdentificeerd?
- Wat zijn de best practices voor het oplossen van de problemen?
- In welke mate worden deze best practices binnen Achmea toegepast?

Hoe kan de nieuwe architectuur eruit gaan zien?

- Welke regels en eisen worden gesteld?
    - *Welke regels en eisen worden gesteld vanuit Enterprise Architectuur?*
    - *Welke regels en eisen worden gesteld vanuit Domein Architectuur?*
    - *Welke regels en eisen worden gesteld vanuit Informatie Management?*
    - *Welke regels en eisen worden gesteld vanuit Solution Architectuur?*
- Hoe kan gezorgd worden dat de nieuwe architectuur gedragen wordt?
- Welke randvoorwaarden aan de nieuwe architectuur zijn reeds vastgelegd?
    - *Hoe moeten web API's bij Achmea werken?*
    - *Wat moet uit de huidige architectuur behouden worden?*
- Welke wijzigingen moet de architectuur binnen welk tijdsbestek ondersteunen?
- Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur design best practices?
- Wat zijn de best practices op het gebied van enterprise architectuur design?
- Welke mogelijke architecturen zijn er?
    - *Waar moet de grens voor de domein API's komen?*
    - *Waar moet de grens voor de web API's komen?*
    - *Hoe kunnen de web API's gegroepeerd worden?*

    - *Hoe kan het afwijken van de bedoelde architectuur voorkomen worden?*
- In welke mate voldoen de mogelijke architecturen aan de regels, eisen en randvoorwaarden?

Hoe kan van de oude naar de nieuwe architectuur gemigreerd worden?

- Welke criteria worden gesteld aan de architectuur migratie?
- Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur migratie best practices?
- Wat zijn de best practices op het gebied van enterprise architectuur migratie?
- Welke mogelijke migratie-paden zijn er?
- In welke mate voldoen de mogelijke migratie-paden aan de criteria?

Welke architectuur met bijbehorend migratiepad kan aanbevolen worden?

- Welke migratie-paden zijn mogelijk voor welke architecturen?
- Welke architectuur-migratiepad combinatie voldoet het beste aan de eisen?

Onderzoeksresultaten

Web API's bij Zilveren Kruis Achmea

Mark Staarink – 1618662
Software Engineering
Hogeschool Utrecht
Zilveren Kruis Achmea
16-12-2016

Inhoudsopgave

C.1. Inleiding

In dit document zijn de antwoorden op alle deelvragen uit het onderzoek vastgelegd. Waar de scriptie enkel de kern van de onderzoeksresultaten overdraagt, zijn in dit document alle details terug te vinden.

C.2. Wat is de context van de opdracht?

Om de opdracht uit te voeren moeten de context van het probleem en het probleem zelf bekend zijn. Dit omvat de werking van zorgverzekeraars in het algemeen, de functionaliteiten die door Zilveren Kruis Achmea worden aangeboden en hoe deze functionaliteiten ondersteund worden. Ook de verschillende aspecten van het probleem zijn belangrijk: wat het precies is, waar het zich voordoet, en wat de impact ervan is op het bedrijf.

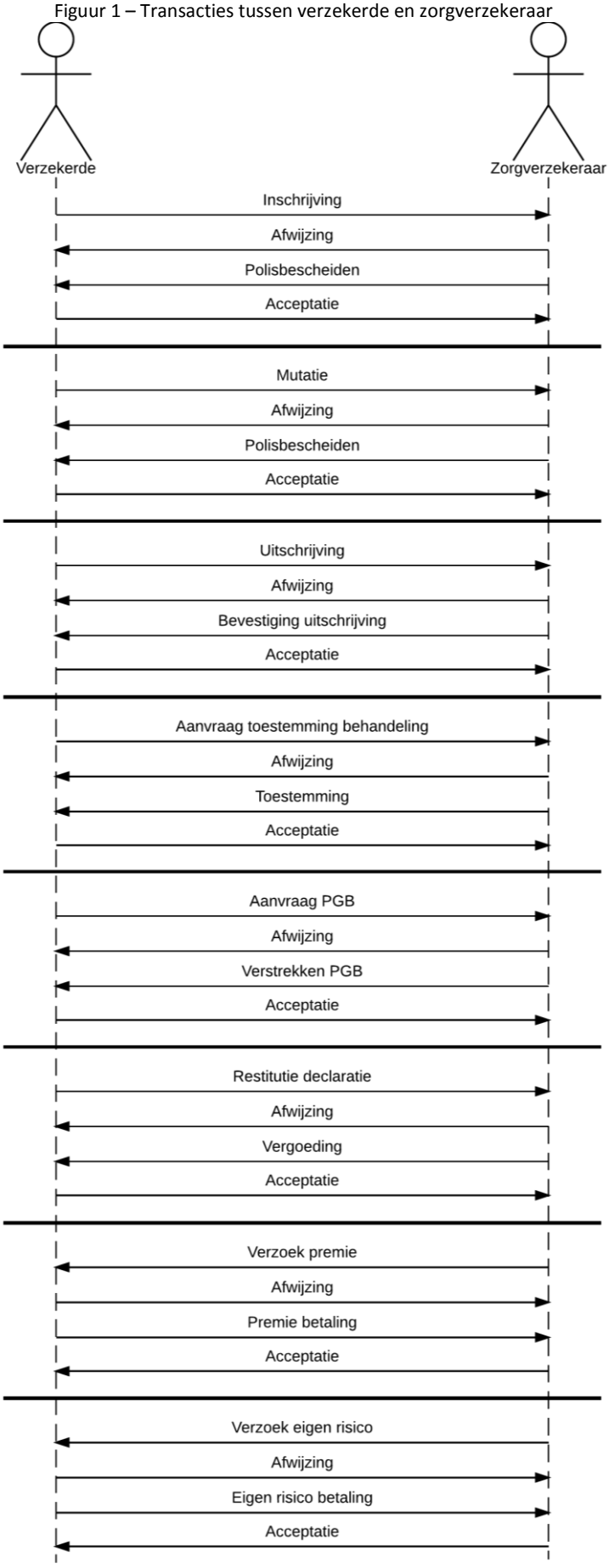
C.2.1. Hoe werkt een zorgverzekeraar?

Bij de processen van een zorgverzekeraar zijn drie partijen betrokken: de verzekerde, de zorgverlener, en natuurlijk de zorgverzekeraar zelf. Tussen deze drie partijen vinden transacties plaats welke de basis vormen voor het functioneren van een zorgverzekeraar. Sterk versimpelt kan gezegd worden dat verzekerden zorg afnemen bij zorgverleners, en de zorgverzekeraar hierbij op verschillende manieren bemiddeld.

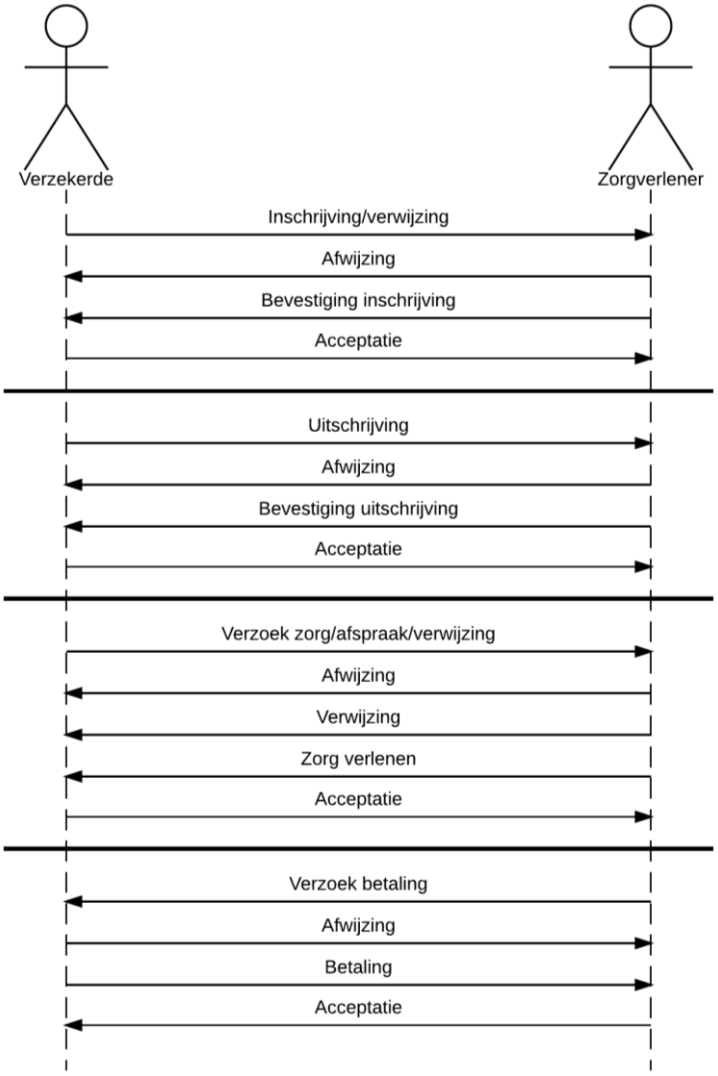
Om de transacties in beeld te krijgen zijn deze uitgewerkt volgens de principes van DEMO. Deze stellen dat organisaties opereren op een basis van personen die transacties met elkaar aangaan. Deze transacties volgen een vast patroon, waarbij twee actoren meerdere coördinerende handelingen uitvoeren omtrent één productie handeling. (Dietz, 2011, p. 5)

In Figuur 1 tot en met Figuur 5 zijn alle uitvoerende transacties uitgewerkt. Bij de visualisatie van de transacties is niet gebruik gemaakt van de DEMO-modellering om deze kennis niet van de lezer te vereisen. Desondanks wordt het transactie-patroon snel duidelijk: de initiërende actor vraagt een feit aan, de uitvoerende actor kan dit verzoek afwijzen, of accepteren om vervolgens een feit te produceren en te leveren, waarna de initiërende actor het geproduceerde feit kan afwijzen of accepteren. (Dietz, 2011, pp. 6-9) (Wanneer er meerdere pijlen op rij één richting op gaan betreft het een keuze tussen die pijlen.)

Figuur 1 – Transacties tussen verzekerde en zorgverzekeraar

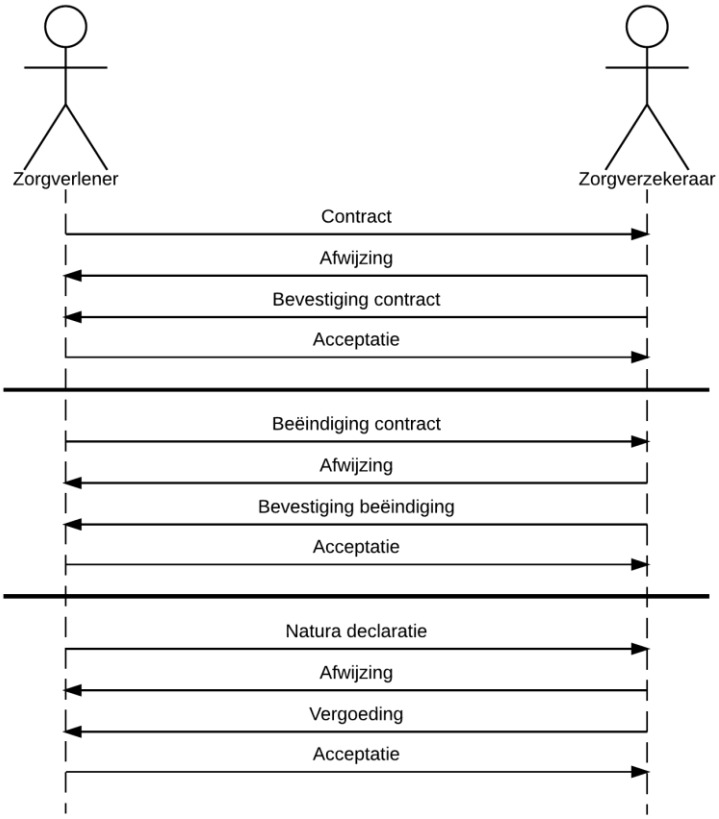


Figuur 2 – Transacties tussen verzekerde en zorgverlener

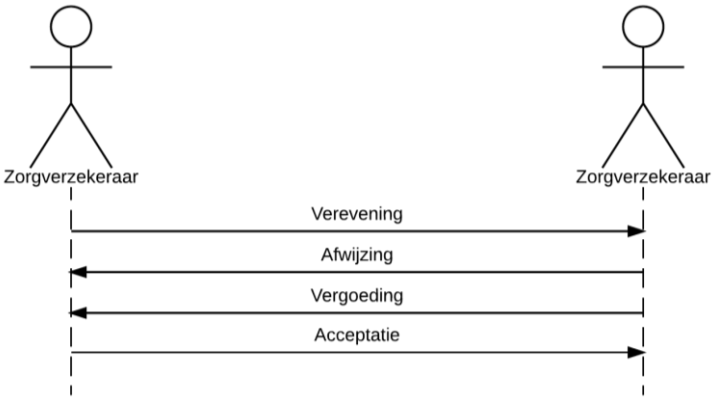


Figuur 3 – Transacties tussen zorgverleners

Figuur 4 – Transacties tussen zorgverlener en zorgverzekeraar



Figuur 5 – Transacties tussen zorgverzekeraars



C.2.2. Wat doet Portalen Zilveren Kruis?

Portalen Zilveren Kruis, ook bekend als de zorgportalen, zijn de (web) interfaces welke door zowel klanten als medewerkers van het bedrijf gebruikt worden om alle klant- en zorg-gerelateerde gegevens te beheren. Hiertoe maken de portalen gebruik van de verschillende services die het bedrijf beheert voor het omgaan met deze gegevens. Voor verschillende handelingen en functionaliteiten bestaan verschillende portalen.

De meeste portalen hebben betrekking op klantgegevens en de gerelateerde transacties, door de klant zelf of door medewerkers. Klant Contact is de eerste lijn

van contact voor klanten die informatie of hulp nodig hebben. Mocht een klant niet door Klant Contact geholpen kunnen worden, bijvoorbeeld bij een zeer complexe zorg transactie, of een probleem van technische aard (denk aan uitval of andere problemen met de systemen), dan wordt er doorverwezen naar Operations.

De portalen in Tabel 1 worden binnen Delivery Center Zorg, de afdeling waar de student stage loopt, ontwikkeld en onderhouden. (R. ter Velde, 20 september 2016)

Tabel 1 – De Zorg portalen

Portaal	**Functionaliteit**	**Platform**	**Eigenaar**
Publiek Domein Zilveren Kruis / Prolife	Statische informatie-site, overlapt met het Klant Domein.	SharePoint 2010	Klant Contact
XHI	Portaal voor expats, voornamelijk buitenlandse werknemers in Nederland.	SharePoint 2010	Operations
Regie op Controle	Workflow voor controle op onder andere declaraties.	SharePoint 2007	Operations
Klant Domein (alle labels)	Management portaal voor klanten.	SharePoint 2010	Klant Contact
LightLogin	Onderdeel van Klant Domein waarbij ingelogd worden zonder DigiD (en dus minder handelingen gedaan kunnen worden).	SharePoint 2010	Klant Contact
Medewerkers Domein (Mikado)	Gebruikt door Klant Contact voor management van klanten.	SharePoint 2010	Klant Contact
One Desk	Wordt in Mikado opgenomen.	SharePoint 2007	Klant Contact
Vergoedingen Tool	Inzien van vergoedingen.	SharePoint 2010	Klant Contact
Werkgevers Portaal	Managen van collectieve verzekeringen, afnemen van producten.	SharePoint 2010	Klant Contact
ZorgBeheer (Nota Sniffer)	Nota beheer tool.	MVC	Klant Contact
Communicatie Manager (Output)	Regelt de communicatie tussen IKAZ en Output, voor het afdrukken van brieven en documenten.	WCF / .NET	Operations
IDNS	Nota service. Converteert foto's naar digitaal leesbare nota's.	WCF	Klant Contact
OZF	Publiek domein van OZF.	SharePoint 2007	Klant Contact
Zorg Kantoren	Uitvoering wet langdurige zorg.	SharePoint 2007	Klant Contact
Declaratie Forms	Onderdeel van Publiek Domein.	SharePoint 2007	Klant Contact

Binnen Klant Contact zijn daarnaast nog andere portalen in gebruik. In Tabel 2 zijn de portalen gegeven die met Delivery Center Zorg koppelen.

Tabel 2 – De Klant Contact portalen

Portaal	Functionaliteit	Externe partij	Eigenaar
ZorgZoeker	Zoekmachine om zorgverleners te vinden.	ETTU	Klant Contact
CollectiviteitZoeker	Zoekmachine om informatie over collectiviteiten te vinden.	ETTU	Klant Contact
VergoedingentoolWAPI kloon	Kopie van web API voor gebruik buiten de webAPIproxy om.	ETTU	Klant Contact
SalesFunnel	Verkoop van nieuwe polissen.	ETTU	Klant Contact

C.2.3. Wat doen IKAZ, SAP, BI en Output?

IKAZ, SAP, BI en Output zijn de vier van de vele componenten die de portalen waar klanten en medewerkers mee in aanraking komen ondersteunen. Zij bieden de functionaliteiten waar de portalen aanroep op doen. Hieronder zullen deze componenten beschreven worden.

C.2.3.1. IKAZ

Integrale Kantoorautomatisering Zorgverzekeringen, bekend onder de afkorting IKAZ (van Alphen, 2015), is met meer dan dertigduizend functiepunten verreweg het grootste component. Het dient ter ondersteuning van de uitvoering van zorgverzekeringen, en wordt daarbij voornamelijk gebruikt voor polis management en het verwerken van declaraties. Hierbij moeten gegevens en business-regels van minimaal vijf jaar terug onthouden en ondersteund blijven.

IKAZ is opgebouwd uit de volgende subsystemen: (Achmea, z.j.)

- IKAZ Voorportaal
- IKAZ Propositie
- IKAZ Polis
- IKAZ Vergoeding
- IKAZ Tarief
- IKAZ Product
- IKAZ DSS
- IKAZ TF
- IKAZ Zorgaanbieder
- IKAZ Tabeldistributie
- IKAZ Contract
- IKAZ Naportaal

C.2.3.2. SAP

“Systemanalyse und Programmentwicklung”, SAP, is een maker van enterprise software voor het managen van bedrijfsoperaties en klantrelaties. Binnen de context van Achmea speelt SAP-CRM (Customer Relationship Management), het SAP software-pakket voor klantrelaties, een belangrijke rol. SAP BP (Business Partner) is een groot component van het CRM pakket, waarbinnen de gegevens van een relatie (persoon of organisatie) worden opgeslagen.
Daarnaast wordt ook SAP-CD (Collections and Disbursements) gebruikt binnen het bedrijf. Hiermee worden zowel binnenkomende als uitgaande betalingen afgehandeld. (van Alphen, 2015, p. 6)

Momenteel wordt er nog een tweede component gebruikt, Click, welke de relatiegegevens van klanten binnen de domeinen van Zilveren Kruis beheert. Deze gegevens worden ook door IKAZ, de SAP pakketten, en andere componenten gebruikt, welke allen een eigen kopie bewaren. Click heeft binnen deze situatie de leidende rol: het ontsluit de gegevens naar buiten, en synchroniseert mutaties met de andere applicaties die de gegeven nodig hebben. Bij gegevens-conflicten wordt Click als leidend gehanteerd. In 2017 wordt Click uit gefaseerd en neemt SAP-CRM de leidende rol over. (Piso, 2016, p. 16)

C.2.3.3. BI

De software systemen houden hun activiteiten bij door middel van logging. Om hier goed gebruik van te maken wordt Business Intelligence toegepast op de logbestanden om analyses uit te voeren welke kunnen helpen bij het verbeteren van de bedrijfsprocessen en systemen.

C.2.3.4. Output

Output omvat alle functionaliteit die gebruikt wordt voor het managen van email en brieven. Het Communicatie Manager portaal gebruikt Output voor het afdrukken van (voornamelijk individu-gerichte) brieven. Binnen Mikado (het medewerker domein, gebruikt door Klant Contact) heeft Document Wizard een eigen portaal, welke het gebruiken van template documenten vergemakkelijkt. Daarnaast is er nog het Document Archief, waarin WOF (Werk Opdracht Formulier) documenten worden opgeslagen die afgehandeld moeten worden door de afdeling Operations.

C.2.4. Wat is het probleem dat opgelost moet worden?

Vanuit de huidige software architectuur richtlijnen is het niet mogelijk om alle doelstellingen van Zilveren Kruis Klant Contact te halen. Dit komt omdat de architectuur de volgende problemen en beperkingen met zich mee brengt:

- De (nieuwe) performance eisen worden niet gerealiseerd.
- De gewenste “Time To Market” wordt niet gehaald. Dagelijks een nieuwe versie uitbrengen op de productie-omgeving is gewenst, maar het feit dat kleine wijzigingen grote storingen kunnen veroorzaken speelt dit tegen. Daarom wordt noodgedwongen in een “Big Bang” opgeleverd, wat niet aansluit bij de wens om agile te werken.
- “Hot deployment” is niet mogelijk, bij elke oplevering moet het betrokken systeem offline gaan. Gezien de “Big Bang” oplevering is de downtime hierbij verre van minimaal.
- Kanaalonafhankelijke bediening, waarbij de backend vanaf verschillende platforms aan te roepen is, is moeilijk te realiseren. Dit beperkt de flexibiliteit van de architectuur.
- Het is niet mogelijk om verzekerden door hun “customer journey” heen te volgen, wat gegevens op zou kunnen leveren waarmee de gebruikerservaring gerichter verbeterd kan worden.

C.2.5. Waar speelt het probleem?

Het probleem heeft de grootste impact op de portalen, welke afhankelijk zijn van de rest van de architectuur voor hun communicatie met de services. De ontwikkelteams achter de portalen ervaren dan ook de meeste storende effecten, welke allen neerkomen op het kwijt raken van tijd. Denk hierbij aan tijdrovende opleveringprocessen en code waar, door de grote hoeveelheid koppeling, moeizaam aanpassingen in gemaakt worden.

De wensen voor de portalen worden gecommuniceerd met IKAZ, zodat deze twee delen op elkaar afgestemd kunnen worden. Indirect worden de IKAZ ontwikkelteams dus beïnvloed door het probleem, maar hun ontwikkelproces wordt er niet door gestoord.

C.2.6. Waarom is het een probleem?

De doelstellingen van Klant Contact worden opgesteld met de gebruikerservaring van zowel de medewerkers als de klanten in gedachten. Wanneer deze doelstellingen niet gehaald kunnen worden, worden de gebruikerswensen niet vervuld. Voor de medewerkers kan dit betekenen dat er minder efficiënt gewerkt wordt. Voor klanten kan dit betekenen dat de gebruikerservaring achteruit gaat, waardoor klanttevredenheid daalt.

C.2.7. Hoe groot is het probleem?

Zolang het probleem niet opgelost wordt zullen een aantal nadelen elkaar blijven versterken:

- De afhankelijkheden van de portalen worden steeds groter. De transactie-laag blijft groeien, en één wijziging daarin kan meerdere portalen raken.
- De complexiteit van het systeem blijft oplopen. Hierdoor wordt ontwikkeling niet alleen risicovoller (de kans bestaat dat een verandering onvoorziene effecten heeft), het kan ook meer tijd gaan kosten, bij zowel het ontwikkelen als het testen.
- Door de toenemende complexiteit wordt het steeds lastiger om de kwaliteit van de code te waarborgen.
- De “Big Bang” manier van opleveren brengt extra risico met zich mee, gezien een grote hoeveelheid veranderingen in één slag op productie terechtkomt, wat onvoorziene effecten met zich mee kan brengen.

Dit alles leidt er toe dat niet snel gereageerd kan worden op klantwensen waardoor, vanuit het oogpunt van de klant, er altijd achter de feiten aangelopen wordt. Dit effect blijft zichzelf opbouwen tenzij de problemen verholpen worden.

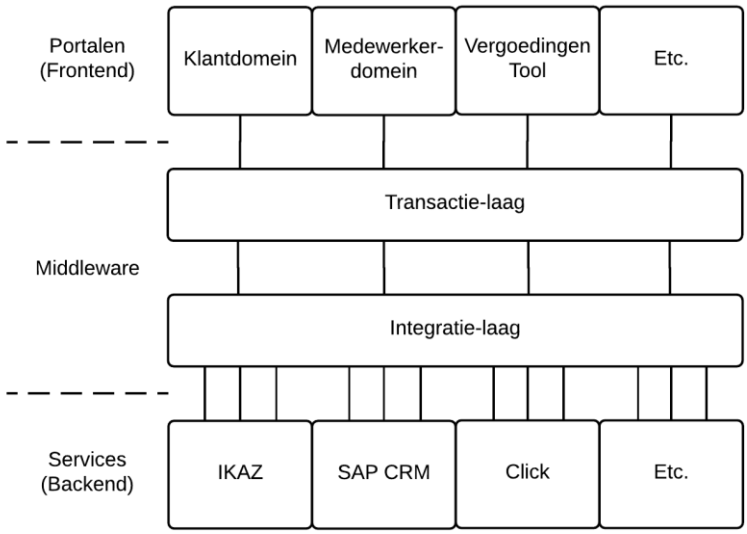
C.3. Hoe ziet de huidige situatie eruit?

Voor zowel het onderzoeken van de probleemoorzaken en het inventariseren van de mogelijkheden is het belangrijk om de huidige architectuur met alle achterliggende invloeden in beeld te krijgen. In dit hoofdstuk zijn deze uitgelicht en is er gekeken naar best practices welke voor de op te lossen problemen gelden.

C.3.1. Hoe ziet de huidige architectuur eruit?

In de huidige architectuur zijn drie lagen te onderscheiden: de frontend, de backend, en de middleware daartussen. In Figuur 6 wordt een sterk versimpeld overzicht van deze drie lagen gegeven.

Figuur 6 – Versimpeld architectuuroverzicht



- **Frontend**: Bevat de verschillende portalen, (web-)interfaces waar gebruikers mee in aanraking komen. De meeste zijn gerealiseerd in Microsoft SharePoint (zie ook het hoofdstuk *Wat doet Portalen Zilveren Kruis?*).
- **Middleware**: Faciliteert de communicatie tussen de front- en backend.
    - **Transactie-laag**: Biedt business logica voor de frontend, communiceert hiertoe met de integratie-laag. Door de frontend aangeroepen als DLL.
    - **Integratie-laag**: Biedt versimpelde interfaces voor het aanroepen van de services.
- **Backend**: Bevat de verschillende services.

Om de werking van de architectuur beter te begrijpen wordt hieronder dieper ingegaan op zowel de Microsoft stack (welke bestaat uit de portalen en de transactie-laag) als de integratie van de services

C.3.1.1. Hoe werkt de Microsoft stack bij Achmea?

De Microsoft stack bestaat uit de portalen en de transactie-laag welke zij gebruiken om met de services te communiceren. De meeste portalen zijn geïmplementeerd in Microsoft SharePoint. De transactie-laag is geprogrammeerd in C# .NET, en wordt als DLL's door de portalen aangeroepen. De bij deze deelvraag besproken situatie is terug te vinden in het originele ontwerpdocument (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013) en het klassendiagram van de huidige situatie (van 't Klooster, cd TotalClass, 2014).

C.3.1.1.1. Functionaliteit

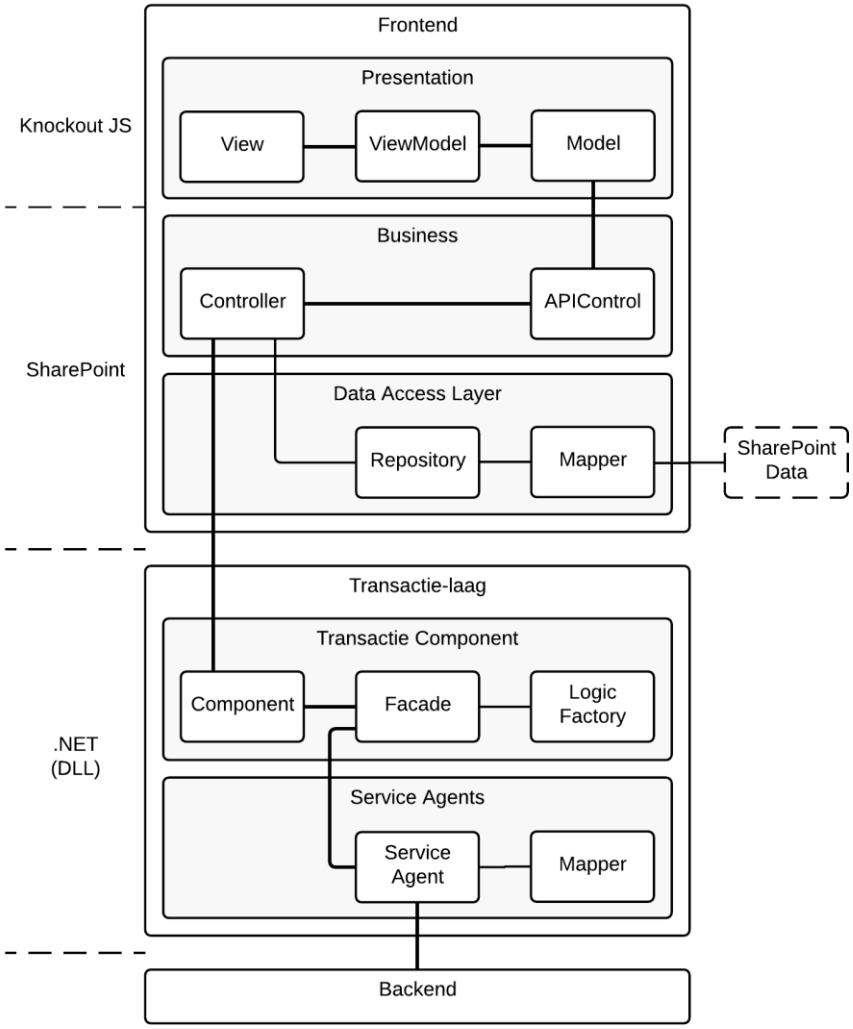
Naast het bieden van een simpele, generieke interface voor het aanroepen van services door de portalen, brengt de bemiddeling van de transactie-laag nog enkele andere functionaliteiten.

- **Data-validatie:** Gegevens kunnen op correctheid gecontroleerd worden nog voordat deze naar de services gestuurd worden. Bij duidelijk ongeldige data kan zo een "overbodige" aanroep voorkomen worden.
- **Data-aanvulling:** Wanneer nodig kan incomplete data aangevuld worden.
- **Caching:** Om redundante aanroepen van de services te voorkomen kunnen de ontvangen resultaten gecachet worden.
- **Compensatie:** Business logica kan tekortkomingen van de services opvangen.

C.3.1.1.2. Implementatie

In Figuur 7 wordt een versimpelde weergave van de architectuur van de frontend (portalen) en de transactie-laag gegeven welke de "lijn" van de frontend naar de backend weergeeft en inzicht biedt in de gebruikte architectuurpatronen. Deze worden hieronder nader toegelicht.

Figuur 7 – Versimpelde architectuur frontend en transactie-laag



Patronen

- **Web pagina**, Knockout JS.
  - **View**: Gebruikersinterface.
  - **ViewModel**: Vertaling van gegevens tussen Model en View.
  - **Model**: Communicatie met de Controller (via APIControl).
- **Frontend logica**, SharePoint.
  - **Controller**: Ophalen data uit transactie-laag, applicatielogica voor verschillende merken. Wordt door middel van een Service Locator per pagina opgehaald (Inversion of Control).
  - **Repository**: Eenduidige manier van data-toegang voor Controller.
  - **Mapper**: Houdt data uit domein entiteiten en buitenwereld consistent.
- **Transactie-laag**, .NET, gebruikt als DLL.
  - **Component**: Interface van de transactie-laag.
  - **Facade**: Versimpelde interface voor het aanroepen van services. Voert eventuele business logica uit.
  - **Service Agent**: Communiceert met de services. Wordt door middel van een Service Locator door de Facade opgehaald (Inversion of Control).

- **Mapper**: Vormt de uit de services verkregen data om naar een werkbare structuur.

Overige

- Er wordt gebruikt gemaakt van een domeinmodel wat alle business entiteiten voor de zorg applicaties bevat.
- Alle fouten welke zich binnen de transactie-laag voordoen worden als transactie-exception opgegooid en door de afnemer verder verwerkt.
- Caching wordt in de front- en backend toegepast. Pagina-inhoud voor 15 minuten, service-gegevens maximaal 24 uur (’s nachts gereset).

C.3.1.2. Hoe werkt de integratie van de backend systemen?

De backend, ook wel de backoffice genoemd, bestaat uit meerdere componenten die allen een service aanbieden. Deze services worden niet direct afgenomen door de buitenwereld (waaronder de transactie-laag), maar worden door een integratie-laag ontsloten. In deze laag staat een ESB (Enterprise Service Bus) centraal. Deze faciliteert een transparante infrastructuur voor het realiseren van een SOA (Service Oriented Architecture), wat ingezet wordt om ontkoppeling en hergebruik te realiseren. (Bril, Technische Domein Architectuur Integratie Services, 2012, p. 12)

C.3.1.2.1. Functionaliteit

Onder andere de onderstaande functionaliteit wordt door de integratie-laag ondersteund of aangeboden. (Bril, Technische Domein Architectuur Integratie Services, 2012, pp. 11-13)

- **Transformeren:** Om communicatie te faciliteren worden berichten vertaald tussen verschillende formaten en protocollen.
- **Routeren:** Er wordt garandeert dat berichten op de juiste plaats afgeleverd worden, bij de correcte (instantie van een) service, en dat eventuele responsen weer bij de plek van aanroepen terecht komen. Dit geldt voor verschillende services, maar ook verschillende interfaces van de services. Services die veel kleine interfaces aanbieden worden door de ESB versimpeld aangeboden. Er kan prioriteit worden gegeven aan bepaalde aanvragen.
- **Beveiliging:** Om de services tegen misbruik te beschermen gebeurt de beveiliging van de aanvragen en responsen in de integratie-laag. Denk hierbij aan het vereisen van een beveiligd communicatiekanaal, of het controleren van de berichtinhoud.
- **Monitoring:** Er worden gegevens bijgehouden over de binnenkomende aanvragen, zoals het aantal aanroepen en hun duur. Op basis daarvan kan gerapporteerd worden, maar er kan ook direct gereageerd worden, bijvoorbeeld wanneer een SLA (Service Level Agreement) doelstelling niet gehaald wordt.

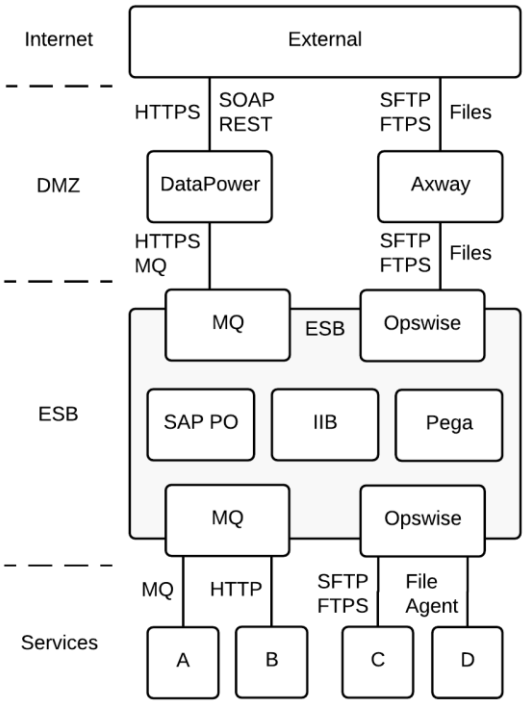
Bij het routeren van berichten worden verschillende soorten aanvragen onderscheden. (Bril & Kollau, Integratie Services – Visie op interactie tussen Enterprise services en Enterprise Service Bus (ESB), 2008, p. 19)

- **Fire and Forget:** Op basis van een aanvraag wordt een verzoek uitgevoerd. De aanvrager vertrouwt er op dat dit correct afgehandeld wordt.
- **Request/Reply:** Op basis van een aanvraag wordt een verzoek uitgevoerd. De aanvrager wacht op een antwoord en verwerkt deze synchroon.
- **Fire and Confirm:** Op basis van een aanvraag wordt een verzoek uitgevoerd. De aanvrager start een separaat proces, om daarmee het antwoord asynchroon te verwerken.
- **Publish/Subscribe:** Een leverancier publiceert berichten waarop geabonneerd kan worden. Deze berichten worden verwerkt door de abonnees wanneer deze ontvangen worden.

C.3.1.2.2. Implementatie

De integratie-laag bestaat uit verschillende componenten. In Figuur 8 is een versimpelde weergave van deze componenten en hun relaties te zien. Deze zijn hieronder nader toegelicht. (G. Bril, persoonlijke communicatie, 4 oktober 2016)

Figuur 8 – Versimpelde integratie-laag



- **Demilitarized Zone (DMZ)**: Dient als proxy/firewall om de integratie-laag te beschermen tegen zaken als denial-of-service attacks en SQL injectie. Dwingt veilige communicatie-protocollen af. Staat fysiek los van de rest van de integratie-laag.
    - **IBM DataPower**: Gateway voor berichtenverkeer.
    - **Axway**: Gateway voor bestandsverkeer.

- **Enterprise Service Bus (ESB)**: Dient als berichtenbroker, transformeert en routeert berichten. Gebruikt hiertoe verschillende stukken software.
    - **IBM MQ**: Gebruikt om Message Queues te bieden. Garandeert exactly-once aflevering van berichten. Kan prioriteiten toekennen. Ziet alle berichten als platte data.
    - **Stonebranch's Opswise**: Framework met een centrale Controller om taken te plannen, en op elk platform een Agent voor het uitvoeren van taken, waaronder het maken, versturen en verwerken van bestanden en het uitvoeren van scripts.
    - **SAP Process Orchestration (PO)**: Broker voor de SAP services, transformeert berichten en routeert de resultaten.
    - **IBM Integration Bus (IIB)**: Generieke broker, transformeert berichten en routeert de resultaten. Stateless.
    - **Pega**: Business Process Manager (BPM) tool. Logica is gedefinieerd als regels in Pega's database. Stateful.
- **Services**: Communiceren via hun eigen protocol en formaat, de ESB regelt alle vertaling. Services kunnen elkaar aanroepen via de ESB. Maken gebruik van agents om bestands-monitoring te doen. Hebben een eigen database voor gegevensopslag.

C.3.2. Hoe is de huidige architectuur ontstaan?

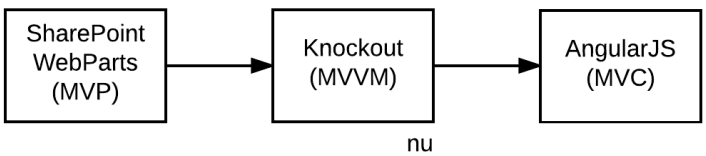
De huidige architectuur is voortgebouwd op voorgaande iteraties. Hieronder staan de ontwikkelingen die tot de huidige architectuur hebben geleid beschreven.

C.3.2.1. Microsoft stack

Onder het project "Voorkantvernieuwing" (waarbij de systemen die gebruikt werden door medewerkers vervangen werden door één enkel systeem) (van den Broek, 2015, p. 1) is de architectuur van de portalen op de schop gegaan. Een nieuwe architectuur, waarbij de portalen gebruik maakten van een transactie-laag ter communicatie met de services, werd in 2013 ontworpen door Rob van 't Klooster. Dit nieuwe ontwerp werd geaccepteerd en uitgewerkt, en is grotendeels onveranderd gebleven.

Een punt waarop het ontwerp wel doorontwikkeld is, is de architectuur van de portalen zelf. In Figuur 9 is weergegeven welke technologieën voor de portalen zijn gebruikt. Hieronder wordt dit verder toegelicht.

Figuur 9 – Ontwikkeling portaal technologieën

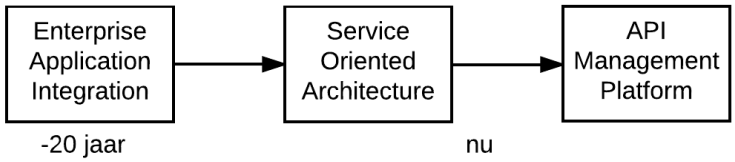


1. In eerste instantie werd gebruik gemaakt van een Model-View-Presenter patroon wat SharePoint pagina componenten gebruikte om sterk te koppelen met de onderliggende SharePoint omgeving. Hierbij werd er state opgeslagen in de business laag.
2. Om de hoeveelheid code te verminderen is van SharePoint WebParts overgestapt naar een JavaScript oplossing. Knockout JS is een framework gebaseerd op het Model-View-ViewModel patroon. Nog niet alle portalen maken hier gebruik van.
3. In de toekomst zal AngularJS gebruikt worden voor de functionaliteit van de web pagina's.

C.3.2.2. Integratie

In Figuur 10 is de ontwikkeling van de integratie architectuur gegeven. Hieronder wordt dit verder toegelicht.

Figuur 10 – Ontwikkeling integratie architectuur



1. Ongeveer twintig jaar geleden is begonnen met Enterprise Applicatie Integratie, wat ervoor moest zorgen dat makkelijk met de verschillende services gepraat kon worden. Er stond al wel een berichtenbroker centraal, maar er werden ook onveilige protocollen en verouderde berichtformaten gebruikt.
2. Vervolgens is een Service Oriented Architecture gerealiseerd door middel van de ESB. Deze ging dienen als bemiddelaar tussen service aanbieder en afnemer. Hiermee werden een hogere beveiligingsgraad en standaardisatie op één berichtformaat binnen de integratie-laag bereikt. (Bril & Kollau, Integratie Services – Visie op interactie tussen Enterprise services en Enterprise Service Bus (ESB), 2008)
3. Momenteel wordt al gewerkt aan API Management. Hierbij is de volgende stap het selecteren van een API Management Platform, wat de focus sterker zou leggen op het toegankelijk maken van de services, en performance kan verbeteren door middel van caching en ondersteuning voor parallelle service aanroepen.

C.3.3. Welke achterliggende regels, eisen en ideeën vormen de huidige architectuur?

De huidige architectuur is sterk gevormd door de verschillende eisen die gesteld worden vanuit het bedrijf, en de doelstellingen welke bij het ontwerpen van de architectuur beoogd werden. Hieronder zijn deze invloeden beschreven.

C.3.3.1. Microsoft stack

Bij het ontwikkelen van de Microsoft stack is beoogd de volgende richtlijnen en doelstellingen te volgen.

- Zo min mogelijk code schrijven. Dit verkleint de kans op bugs en vergemakkelijkt het onderhoud.
    - De frontend gebruikt JavaScript in plaats van SharePoint componenten.
- De Microsoft richtlijnen en best practices op het gebied van data-toegang. (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013, p. 24)
    - Repository patroon: Zorgt voor een gecentraliseerde, geïsoleerde data-toegang laag.
    - Mapper patroon: Zorgt voor isolatie tussen domein entiteiten en de buitenwereld.
    - Service Locator patroon: Koppelt Service Agents los van de data-toegang implementatie, resulteert in Inversion of Control.
- Dubbele code tussen portalen moet voorkomen worden. (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013, p. 9)
    - De transactie-laag biedt een generieke manier voor het aanroepen van de (zorg) services.
    - De transactie componenten bevatten business logica.
- KISS (Keep It Simple, Stupid) en YAGNI (You Ain't Gonna Need It), om hergebruik van code te bevorderen. (Quakernaat & Thissen, 2014, p. 52)
    - Productiviteit omhoog en kosten omlaag.
- Domein- en merk-specifieke functionaliteit moet gescheiden blijven, ter behoeve van onderhoud.
    - Er werd beoogt meerdere transactie-componenten in te zetten.

Bij het ontwerpen van softwarearchitectuur binnen Achmea moet rekening houden met de Referentie Softwarearchitectuur, welke opgesteld is om te zorgen dat oplossingen aansluiten op de business technologie strategie en op een eenduidige manier worden geïmplementeerd. (Quakernaat & Thissen, 2014) Dit document is gebaseerd op het initiële ontwerp van de huidige Microsoft stack architectuur. Implementatiedetails wijken hierin soms af, maar de visie komt overeen.

C.3.3.2. Integratie

- De Enterprise Integratie Patronen worden toegepast. "Out of the box" werkende oplossingen worden verkozen boven maatwerk. (Vorgers, 2015)
- De standaarden, richtlijnen en principes die vastgelegd zijn in de Technische Domein Architectuur Integratie Services. (Bril, Technische Domein Architectuur Integratie Services, 2012, pp. 23-29)
- Er moet voldaan worden aan het hoogste niveau van de BIV-classificatie. (Beschikbaarheid, Integriteit, Vertrouwelijkheid.)

C.3.4. Waar wordt er afgeweken van die regels, eisen en ideeën?

De huidige implementatie van de architectuur is niet volledig conform het beoogde ontwerp en de daarbij gestelde eisen. Hieronder staan deze afwijkingen beschreven.

C.3.4.1. Microsoft stack

- Het Service Locator patroon zou gebruikt worden om de juiste Service Agent bij een aanvraag te zoeken.
    - Dit patroon is zelden toegepast, en de benodigde functionaliteit is opgenomen in een Factory patroon.
- Er zouden meerdere transactie componenten gebruikt worden voor de verschillende domeinen en merken.
    - Er is maar een enkel component gemaakt wat door alle portalen gebruikt wordt. De Facade gebruikt een Logic Factory om domein- en merk-specifieke business logica te verkrijgen.
- Sommige content wordt in SharePoint lijsten opgeslagen, wat de grens tussen content en logica vervaagt.

C.3.4.2. Integratie

De integratie van sommige services gebeurt niet conform de standaard patronen en eisen. Zo maakt de IKAZ service nog gebruik van het onveilige FTP protocol om bestandsuitwisseling met de ESB te doen.

C.3.5. Waarom wordt er afgeweken van die regels, eisen en ideeën?

De voornaamste oorzaak van het afwijken van richtlijnen is dat deze samen met de prioriteiten veel verandering doormaken, waarbij die software zelden direct volgt.

C.3.5.1. Microsoft stack

Toen het ontwerp waar de huidige Microsoft stack architectuur op gebaseerd is werd ontworpen lag de prioriteit voornamelijk op snel kunnen schrijven van code, wat leidde tot generieke componenten en hergebruik. In de loop van de levensduur van die architectuur is de prioriteit verschoven naar het omlaag halen van de kosten van de services. Hierbij was performance een groot doelwit, de services zouden minder kosten als zij efficiënter konden draaien. Dit resulteerde in een sterkere focus op performance, waarvoor onder andere caching geïmplementeerd werd in de transactie-laag.

Op sommige punten verschillen de Microsoft stack en de Software Referentiearchitectuur. Hier is het de Referentiearchitectuur die afwijkt, gezien deze is gebaseerd op de architectuur van de Microsoft stack.

C.3.5.2. Integratie

Waar service integratie niet conform de richtlijnen geïmplementeerd is, komt dit doordat de service geïntegreerd is voordat de huidige richtlijnen opgesteld waren, en de implementatie sinds die tijd niet herschreven is.

C.3.6. Op welke punten voldoet de huidige architectuur niet aan de eisen en wensen?

Gezien de huidige architectuur al enkele jaren stand houdt zijn er punten waarop het niet meer voldoet aan de constant groeiende en veranderende eisen en wensen vanuit de verschillende belanghebbenden. Er is een meeting georganiseerd om deze problemen naar boven te halen. Hieruit is het volgende naar voren gekomen: (persoonlijke communicatie, 18 oktober 2016)

- **Sterke koppeling, veel afhankelijkheden.**
    - Te vaak/te veel testen.
        - Time To Market is te hoog.
- **Versionering.**
    - Voor transactie componenten niet mogelijk, want geïmplementeerd als DLL in GAC in plaats van bin folder (hot deployment ook niet mogelijk).
- **Performance.**
    - Caching aan voorkant moet agressiever, ipv pas onderaan transactie-laag (maar web API cache > frontend cache).
    - Caching gebeurt in session state.
    - JavaScript performance in IE laag.
    - Transactie-laag als DLL, SharePoint ondersteunt geen parallelle calls.
    - Aanroepen op services zijn niet snel genoeg.
- **Services & service interfaces.**
    - Vereisen en geven te veel informatie, interfaces moeten simpeler kunnen (alleen relatienummer en merk vereisen, alleen nodige data teruggeven).
    - Verschillende versies levert extra werk en verwarring, idealiter geen nieuwe versies van interfaces, op zijn hoogst één oudere versie ondersteund.
    - Onduidelijke naamgeving.
    - Request/Reply berichten moeten door MQ heen (trager), MQ zou alleen voor batches moeten zijn.
    - Mutaties moeten soms via Fire and Forget: geen bevestiging.
        - Mikado cache clear knop, om refresh te forceren, om te kijken of mutatie doorgevoerd is.

        - Gewenst: ontvangstbevestiging en/of “ja dit kan” bevestiging.
    - Algeheel slechte aansluiting.
        - Te weinig communicatie over use cases/wensen.
- **Testbaarheid.**
    - Transactie-laag is testbaar (auto-alex), maar niet duidelijk wat er getest wordt.
- **Frontend code.**
    - Verouderd. MVP, WebForms aanpassen gaat moeizaam.
    - Bevat business logica. Moet zo dun mogelijk zijn, zou geen gedeelde code nodig moeten zijn, alleen specifieke code.
- **Transactie component code.**
    - Bevat scherm logica.
- **Kanaalonafhankelijke bediening niet mogelijk.**

Bij het bediscussiëren van de problemen komen drie oorzaken sterk naar voren: (persoonlijke communicatie, 25 oktober 2016)

- De communicatie en afstemming tussen service providers en service afnemers (de medewerkers) is onvoldoende: duurt lang, is onduidelijk etc. Hierdoor worden problemen niet snel genoeg opgelost en sluiten de service interfaces en hun gebruik niet goed op elkaar aan.
- Wanneer nieuwe eisen of architecturen worden toegepast wordt niet alle code herschreven om hieraan te voldoen. Hierdoor is er nog code in gebruik welke sterk verouderd en inflexibel is.
- SharePoint. De functionaliteit die het biedt wordt amper gebruikt, maar de restricties blijven nog wel gelden. Dit verhindert het deployment proces en houdt ontwikkeling tegen.

C.3.7. Welke oplossingen heeft Achmea tot nu toe geïdentificeerd?

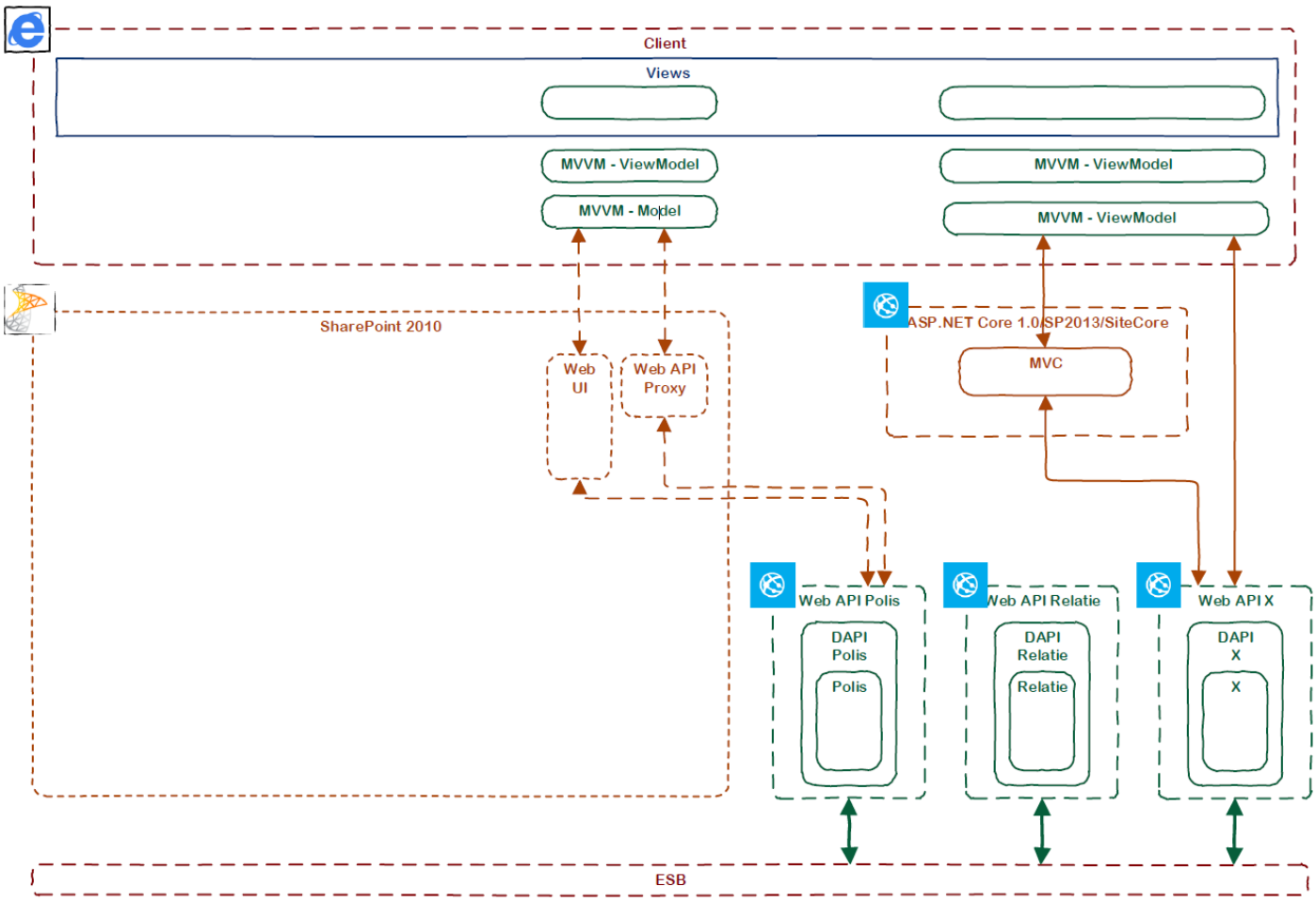
Mikado concept migratieplan

Voor het medewerkersdomein (Mikado) is een concept migratieplan opgesteld. (ter Velde, 2016) Hieronder is het plan kort opgesomd. In Figuur 11 is het eindresultaat gegeven.

1. Voor behandelde User Stories worden operaties uit het Transactie Component omgezet naar domein API’s, met als uitgangspunt hergebruik van de bestaande Service Adapters en domein logica.
2. De domein API’s worden ontsloten door web API’s, maar blijven ook een instantie binnen het Transactie Component houden. Deze twee instanties gebruiken dezelfde codebase, maar een afzonderlijke cache. Web API’s worden naar buiten ontsluiten door een proxy.
3. Er worden op basis van behandelde User Stories meer domein API’s met bijbehorende web API’s ontwikkeld. Operaties die niet door Mikado gebruikt worden blijven achter in het Transactie Component.
4. Overgebleven functionaliteit wordt naar het domein en web API model omgezet. Mikado maakt geen gebruik meer van het Transactie Component.

5. De overgebleven dunne toepassing-laag uit de SharePoint omgeving wordt als Model-View-Controller op het doel platform geïmplementeerd.

Figuur 11 – Mikado migratieplan eindfase. Herdrukt van *Hoofd team sitevoor het Zorg Domein* website, door R. ter Velde, 31 maart 2016, opgehaald van http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Medewerkerdomein/Shared%20Documents/[02]%20Architectuur/SAD/Concept/Migratieplan_concept_v0.2.pdf.



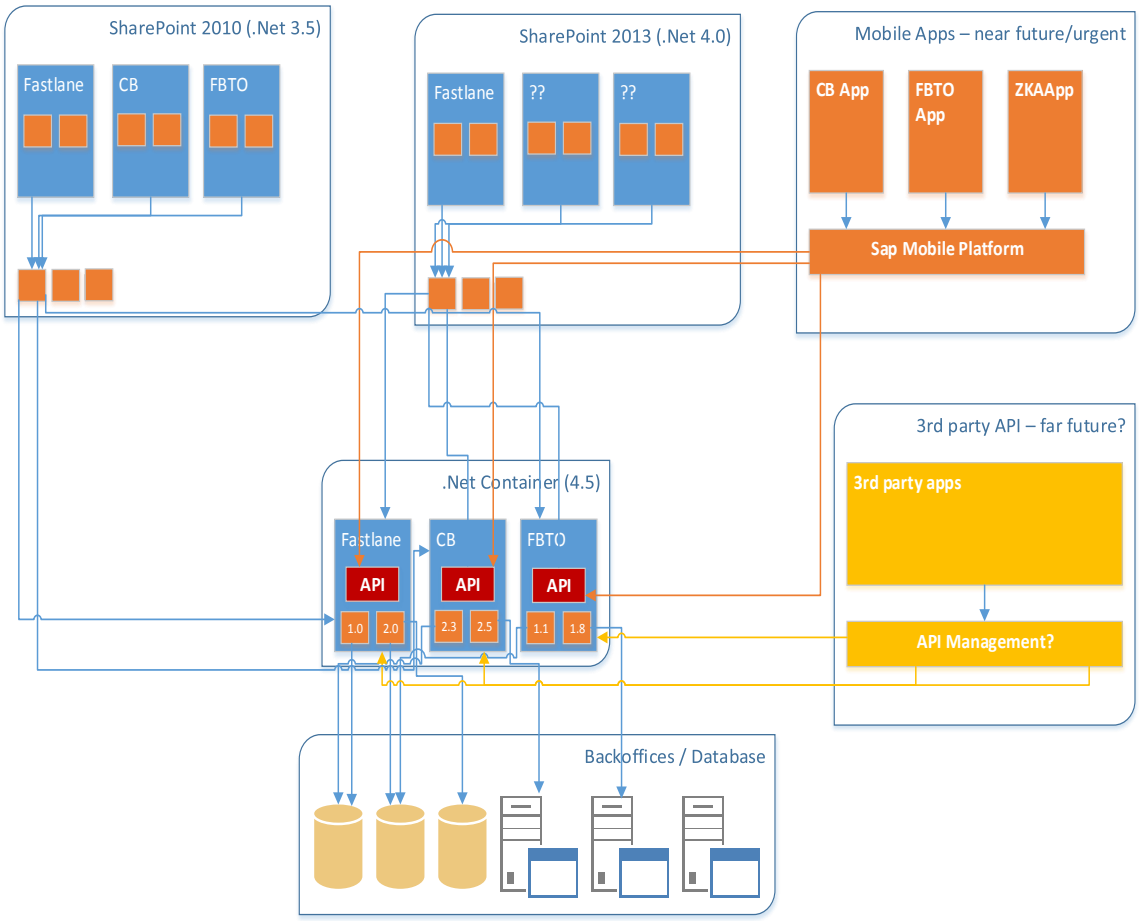
Referentie softwarearchitectuur

In de referentie softwarearchitectuur (Lijten, Referentie Architectuur, 2014, pp. 9-16) wordt een doelarchitectuur met web API's gegeven. Deze wordt gevormd door de volgende doelstellingen:

- Transactie Component gesplitst op basis van domein en merk (in plaats van een enkel Transactie Component voor alles).
- De componenten bieden een web API aan, en ondersteunen hierbij meerdere versies.
- De componenten worden generiek (platform-onafhankelijk) geïmplementeerd.
- Voor authenticatie en autorisatie wordt een OAuth server met JWT's (JSON Web Tokens) gebruikt.
- Mobiele applicaties krijgen via een API Proxy (SAP Mobile Platform) toegang tot de web API's.

In Figuur 12 wordt de doelarchitectuur weergegeven die hieruit voortkomt.

Figuur 12 – Referentie softwarearchitectuur met web API's. Herdrukt van *AchmeaNet: Design Authority* website, door B. Lijten, 12 december 2014, opgehaald van https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/Documenten%20SA/archief/Referentie%20architectuur%20-%20concretisering%20deel%201.pptx.



C.3.8. Wat zijn de best practices voor het oplossen van de problemen?

Om een beeld te krijgen van gangbare oplosrichtingen voor de problemen wordt gekeken naar relevante best practices welke binnen het vakgebied gelden, in de hoop dat zij ondersteuning kunnen bieden bij het oplossen van de problemen.

C.3.8.1. DEMO & Normalized Systems

DEMO is een methodologie voor het ontwerpen, ontwikkelen en implementeren van organisaties. Het toepassen van DEMO op een organisatie levert een enterprise ontologie op, de essentie van een organisatie, welke beschrijft wat een organisatie doet maar onafhankelijk is van hoe dit gerealiseerd is. Een complete ontologie bestaat uit de volgende vier modellen: (Dietz, 2011, p. 24)

- Construction Model
- Process Model
- Fact Model
- Action Model

Normalized Systems (Mannaert, 2012) is een theorie voor het ontwerpen van informatiesystemen. Het stelt dat modulariteit en verandering samen zorgen voor Combinatorial Effects (CE). Dit zijn (al dan niet verborgen) koppelingen en afhankelijkheden, welke verergeren naarmate een systeem groter wordt. Deze effecten voorkomen evolueerbare modulariteit. Normalized Systems zijn bestand tegen CE's voor een set van verwachte veranderingen.
Bij het ontwerpen van Normalized Systems worden vier principes vastgehouden:

- Separation of concerns
- Data version transparency
- Action version transparency
- Separation of states

Normalized Systems stelt voor om bestaande software constructies (functies en classes) te encapsuleren in vijf generiekere software elementen:

- Data Elements
- Action Elements
- Workflow Elements
- Connector Elements
- Trigger Elements

Het gebruik van deze granulaire structuren (waarbij hun complexiteit beperkt moet blijven) moet leiden tot robuuste systemen welke makkelijk aangepast kunnen worden om verwachtte veranderingen door te voeren, waarbij geen CE's voorkomen.

Het is mogelijk om uit een DEMO ontologie een Normalized System af te leiden, gezien de twee methodologieën overeenkomende principes gebruiken. M. R. Krouwel en M. Op 't Land (2012) beschrijven de volgende stappen, waarbij uit verschillende DEMO modellen elementen van een Normalized System worden gehaald:

1. **State Model:** Een data element voor elke object class, scale en category.
2. **Construction Model:** Voor elke transactie twee data elementen: aanvrager en uitvoerder. Beiden met hun eigen workflow element met action elementen.
3. **Action Model:** Beslissingsregels en de volgorde van transacties (workflow). Extra action elementen wanneer een andere transactie gestart moet worden (bridge action).

Na het uitvoeren van de stappen is het resultaat een Normalized System wat de operaties van een organisatie ondersteund.

C.3.8.2. Continuous Integration

Continuous Integration (CI) is het frequent integreren, bouwen en testen van software, vaak meerdere malen per dag, en meestal geautomatiseerd. M. Fowler (2006) beschrijft de volgende onderdelen als belangrijk voor CI:

- **Continuous Integration server:** Een server die verantwoordelijk is voor het automatiseren van het CI proces.
- **Versiebeheer:** De software moet zich in een versiebeheersysteem bevinden, en hierbij alles bevatten wat nodig is om de software te bouwen. Er moet een CI branch zijn waar de CI server de software vandaan haalt. Dit kan ook de main branch zijn.
- **Automatische builds:** De software moet automatisch gebouwd worden. Als de software niet gebouwd kan worden moet dit automatisch gemeld worden.
- **Automatische testen:** Nadat de software gebouwd is moet deze automatisch getest worden. Ook hier moet falen automatisch gemeld worden.
- **Dagelijkse commits:** Door vaak naar de CI branch van het versiebeheer te committen worden conflicten tussen code van twee ontwikkelaars snel gevonden, en zijn deze makkelijker op te lossen.
- **Build trigger:** Elk commit naar de CI branch zou een automatische build moeten triggeren.
- **Kapotte builds fixen:** Wanneer een build niet succesvol voltooid kan worden, moet dit direct opgelost worden. Dit kan eventueel ook op de tests toegepast worden.
- **Snelle build:** Het bouwen en eventueel automatisch testen van de software moet binnen een redelijke hoeveelheid tijd gebeuren.
- **Test op een kloon van de productieomgeving:** De testen moeten zo goed mogelijk de werkelijkheid nabootsen.
- **Maak de meest recente versie toegankelijk:** Ter ondersteuning van het Agile proces.
- **Toegankelijke status:** Het moet voor iedereen makkelijk te zien zijn wat de status van het CI proces is.
- **Automatisch deployment:** De software automatisch op de nodige omgevingen uitbrengen is sneller en veiliger dan dit handmatig doen.

Op het punt “kapotte builds fixen” wordt CI bekritiseerd. Er wordt gesteld dat niemand een directe oplossing nodig heeft, en dat teams na enkele tijd de CI server foutmeldingen negeren. Om de oorzaak (kapotte builds) te verhelpen, wordt voorgesteld om nieuwe of aangepaste code automatisch te testen voordat deze in de CI branch van het versiebeheer opgenomen wordt. Wanneer een aanpassing er voor zou zorgen dat het bouwen of tests falen, wordt de code niet in de CI branch geaccepteerd. (Bugayenko, 2014)

C.3.8.3. Web API's

Voor het ontwerpen en toepassen van web API's worden door meerdere partijen de volgende aanbevelingen gemaakt: (Fenniak, 2013; Mulloy, 2012; Narumoto, 2016)

- Ontwerp RESTful API's,
- Hou naamgeving en URI's simpel,
- Pas versionering toe in de URI's, hou dit simpel,
- Spits API ontwerp toe op de use cases, niet op de business objects,
- Baseer de web API's op zelfstandig naamwoorden, niet op werkwoorden,
- Goede, vindbare documentatie is belangrijk,
- Gebruik HTTP status codes om resultaat aan te geven,
- Vereis SSL (HTTPS) voor alle aanroepen,
- Gebruik OAuth 2.0 voor authenticatie.

Op architectuurniveau stelt K. Clark (2015) meerdere mogelijkheden voor. Deze hebben allen één ding gemeen: er wordt een Demilitarized Zone (DMZ) ingezet om aanvragen op te vangen voordat deze verwerkt worden. Of hier een simpele HTTP gateway in zit of alle API's gedefinieerd worden verschilt per implementatie. Ook andere zaken, zoals de manier waarop web API's intern bereikbaar zijn, zijn architectuur-afhankelijk.

Voor het documenteren van web API's bestaat de OpenAPI Specification. (Open API Initiative, z.j.) Het gebruik van deze specificatie zorgt ervoor dat API documentatie door zowel mensen als machines uit te lezen is, zodat zij de geboden functionaliteiten kunnen ontdekken en begrijpen zonder naar de broncode te kijken. Dit maakt het makkelijker om de web API's te gebruiken.
Een alternatief is RAML, de RESTful API Modeling Language (RAML Workgroup, z.j.), welke dezelfde functie als de OpenAPI Specification dient.

C.3.9. In welke mate worden deze best practices binnen Achmea toegepast?

DEMO en Normalized Systems zijn binnen Achmea vrij nieuw, en worden nog niet op brede schaal toegepast.

Web API's worden op sommige plekken al ingezet voor intern gebruik. Hierbij wordt gehouden aan de richtlijnen die Microsoft biedt voor het werken met ASP.NET web API's, en eisen die het bedrijf zelf heeft opgesteld. Hieronder valt het gebruik van generieke error codes, waarbij HTTP status codes worden toegepast. (Legtenberg & van Sluijsveld, 2016) Ook zijn reeds beslissingen gemaakt over de URI's van web API's. De naamgeving is specifiek, en gebruikt simpele versionering. (Achmea Design Authority, 2016) Authenticatie gebeurt middels ADFS (Active Directory Federation Services) in plaats van OAuth 2.0, hoewel daar wel in de toekomst naar overgestapt gaat worden.

C.4. Hoe kan de nieuwe architectuur eruit gaan zien?

Bij het ontwerpen van een doelarchitectuur moet rekening gehouden worden met alle eisen, richtlijnen en randvoorwaarden. Uiteraard is het bij uitzonderlijke gevallen mogelijk om, met goede onderbouwing, buiten de gestelde kaders te treden, maar bij het ontwerp van de doelarchitectuur wordt gestreefd aan de eisen te voldoen.

C.4.1. Welke regels en eisen worden gesteld?

Binnen het bedrijf bestaan vele documenten waarin eisen en richtlijnen zijn vastgelegd. Om de kern hiervan te vinden is met ontwikkelaars en andere betrokkenen in gesprek gegaan. Hieruit zijn de volgende kern-eisen gekomen: (persoonlijke communicatie, 25 oktober 2016)

- Zo min mogelijk code schrijven, om de kans op bugs te verkleiner en de onderhoudbaarheid te vergroten,
    - Hiertoe moet zo veel mogelijk herbruikbare, generieke code geschreven worden,
- Ontkoppeling is belangrijk, verschillende componenten moeten zo min mogelijk afhankelijkheden hebben,
- Er moeten makkelijk nieuwe componenten toegevoegd kunnen worden,
- De software moet simpel en robuust zijn,
- Er moet één entiteitenmodel gehanteerd worden zodat er een eenduidige manier van data-interpretatie is,
- De software moet veilig zijn,
- De frontend moet op de gebruiker toegespitst kunnen worden.

Verdere eisen en richtlijnen zijn gedetailleerder vastgelegd in meerdere documenten. Het grootste deel is terug te vinden in de referentie softwarearchitectuur (afgekort SRA), welke andere requirements-documenten als bronnen heeft. Deze eisen en richtlijnen worden hieronder per domein behandeld.

C.4.1.1. Welke regels en eisen worden gesteld vanuit Enterprise Architectuur?

Op bedrijfsniveau worden eisen en principes vastgelegd waaraan gehouden moet worden bij het ontwerpen en ontwikkelen van software. Voor Enterprise Architectuur gelden de volgende principes: (Quakernaat & Thissen, 2014, pp. 41-44)

- De Enterprise Architectuur maakt het mogelijk om bij het ontwikkelen van nieuwe Achmea diensten en producten gebruik te maken van bestaande IT diensten en producten.
- De Enterprise Architectuur richt zich op het zo efficiënt en effectief mogelijk inrichten van diensten, producten, organisatie, processen en informatiesystemen. Redundantie dient daarbij te worden voorkomen.

- De Enterprise Architectuur moet het mogelijk maken om te voldoen aan de voorschriften van interne en externe toezichthouders op het gebied van compliance.
- De Enterprise Architectuur houdt vooraf rekening met de mogelijke koop of verkoop van bedrijfsonderdelen en/of bedrijfsdiensten.
- De rode draad van de Enterprise Architectuur is het zorgen voor en in stand houden van een betrouwbare informatievoorziening.
- Klanten en medewerkers van Achmea dienen onafhankelijk van tijd en plaats toegang te hebben tot de informatie die ze mogen verwachten.
- De Enterprise Architectuur maakt het mogelijk dat klantinformatie gedeeld wordt tussen de verschillende Business Domeinen van Achmea.
- De architectuur richt zich op samenwerking. Zowel onderling tussen Achmea medewerkers, als met klanten en met externe service providers. Deze samenwerking vindt plaats op basis van services. Er worden daarbij bedrijfsservices, processervices, applicatieservices en technologieservices onderkend.

Voor de ontwikkeling van software binnen het bedrijf moet aan de volgende doelstellingen gehouden worden: (Quakernaat & Thissen, 2014, p. 47)

- Eigenaarschap verdeeld over distributie en productie,
- Applicaties conformeren zich aan een service architectuur,
- Applicaties zijn gestructureerd in beheerbare componenten,
- Applicaties conformeren zich aan geselecteerde standaarden,
- Applicaties ondersteunen 24*7 beschikbaarheid,
- Applicatie-ontwikkeling is sterk gestandaardiseerd,
- Technologiekeuzen zijn strategisch,
- Technologie is gebonden aan een geselecteerd inzetgebied,
- De IT omgeving beheert zichzelf autonoom.

C.4.1.2. Welke regels en eisen worden gesteld vanuit Domein Architectuur?

Aan de architectuur van software op domeinniveau worden vanuit de software referentiearchitectuur de volgende eisen gesteld: (Quakernaat & Thissen, 2014, pp. 21-22)

- Een Web API mag alleen een onderliggende laag aanroepen waarbij het eigenaarschap hetzelfde is,
- Een Web API is onafhankelijk van de cliënt applicatie,
- Een Web API is eenvoudig in gebruik en ontworpen voor gebruik door derden,
- Een Web API wordt zo specifiek mogelijk ontwikkeld qua operaties en parameters. Een voorbeeld is het aanvragen van een propositie waarbij alleen de benodigde parameters ingevuld dienen te worden, en niet alle attributen van het gerelateerde product,
- Een Web API moet ontwikkeld worden met het ASP.NET Web API 2 framework,

- Een Web API mag geen business logica bevatten. Validatie logica is wel toegestaan,
- Een cliënt applicatie mag meerdere Web API's aanroepen zolang geen business transactie wordt samengesteld,
- Een domein API component mag alleen een afhankelijkheid hebben met een meer generiek domein API component (niveau van eigenaarschap),
- In het geval van een samengestelde business transactie moet de Web API alle eindgebruikers invoerwaarden ontvangen die nodig zijn om operaties uit te voeren met afhankelijke domein API's,
- De Web API mag niet aangeroepen worden met parameters welke door een backoffice gereproduceerd kan worden,
- Een domein API mag een domein API uit een ander domein aanroepen waarbij het niveau van eigenaarschap gelijk of meer generiek is,
- Abstracte interfaces worden gebruikt om te koppelen per laag en binnen de business laag indien relevant.

Daarnaast zijn er beslissingen gemaakt over caching welke invloed hebben op de architectuur: (Quakernaat & Thissen, 2014, pp. 35-36)

- Stateless tenzij er een valide reden is om caching te introduceren,
- API operaties mogen alleen gebruik maken van cache binnen dezelfde API,
- Caching moet geïsoleerd worden per API (verschillende cache containers),
- Er moet onderscheid gemaakt worden in het type data en levensduur in het concrete ontwerp van een component,
- Niet reproduceerbare of kostbaar op te halen data mag opgeslagen worden.
- Veel geraadpleegde systeem data mag op domein API niveau worden gecached,
- Alleen de domein API mag gebruikt worden voor de opslag en het managen van cache,
- Een cache mag alleen gemuteerd worden binnen dezelfde succesvolle transactie waarmee een back-office systeem mutatie wordt uitgevoerd,
- Een Web API mag geen expliciete invalidatie operatie bevatten.

C.4.1.3. Welke regels en eisen worden gesteld vanuit Informatie Management?

Informatie Management stelt enkele eisen vanuit het business perspectief: (J.J.P. Overtoom, persoonlijke communicatie, 2 november 2016)

- Betreft de snelheid waarmee software geleverd kan worden:
    - Bij lage snelheid van leveren in herbruikbaarheid van software belangrijk, om tijd te besparen.
    - Bij hoge snelheid van leveren kan wegwerp-code gepermitteerd worden.
- Software ontwikkeling moet conform marktstandaarden gebeuren.
- Technologieën en technieken moeten gebruikt worden waar zij voor bedoeld zijn.

- Kosten moeten deel zijn van het keuzeproces. Tenzij anders verantwoordbaar moet de goedkopere keuze gemaakt worden.
- Er moet conform de wetgeving met persoonsgegevens omgegaan worden.

C.4.1.4. Welke regels en eisen worden gesteld vanuit Solution Architectuur?

De solution architecten van het bedrijf moeten zich houden aan het Design Manifesto (van Sluijsveld & van der Plas, 2016, pp. 4-5). Hierin zijn de volgende architectuurprincipes vastgelegd:

- Autonomie funnel: Iedere funnel moet onafhankelijk kunnen worden gedeployed.
- Versioning: Meerdere versies van dezelfde component moeten naast elkaar kunnen draaien.
- We vertrouwen de klant niet: Informatie die uit bedrijfssystemen kan worden achterhaald wordt gebruikt boven door de klant aangedragen informatie.
- Alleen klanthandelingen ontsluiten naar buiten: Op de publieke webapi's mogen alleen operaties bestaan voor handelingen die een klant zelf mag uitvoeren.

Voor SharePoint 2010 oplossingen worden de richtlijnen van Microsoft aangehouden: (Lijten, van Loon, & van Hooijdonk, Achmea SharePoint 2010 Solution Architecture Guidelines, 2012, p. 5)

- SharePoint Guidance: *https://msdn.microsoft.com/en-us/library/ff650022.aspx*
- Logical architecture components (SharePoint Server 2010): *https://technet.microsoft.com/en-us/library/cc263121(office.14).aspx*
- SharePoint Foundation 2010 Building Blocks: *https://msdn.microsoft.com/en-us/library/ee534971.aspx*

C.4.2. Hoe kan gezorgd worden dat de nieuwe architectuur gedragen wordt?

Het Agile Manifesto (Beedle, et al., 2001) stelt dat "The best architectures, requirements, and designs emerge from self-organizing teams." De beste architecturen worden dus niet van bovenaf opgelegd, maar worden door de teams zelf ontwikkeld.

Met dit in gedachten worden ontwikkelaars, architecten en andere belanghebbenden bij het ontwerpproces van de architectuur betrokken. Zij krijgen inzicht in het ontstaan van de doelarchitectuur en kunnen hier inspraak op leveren. Zo kan de doelarchitectuur beter aansluiten op hun wensen en ideeën, en kunnen zij zorgen dat de nodige kennis beschikbaar is tijdens het ontwerpen. Door deze samenwerking zal de architectuur sneller gedragen worden.

C.4.3. Welke randvoorwaarden aan de architectuur zijn reeds vastgelegd?

Naast de eisen en restricties zijn er ook andere randvoorwaarden waar rekening mee gehouden moet worden. Zo is het belangrijk te weten dat de gebruikersinterface modern gehouden wordt en dus regelmatig verandering doormaakt.
Ook zijn er reeds beslissingen gemaakt over de werking van web API's, en zijn er zaken uit de huidige architectuur die betrokkenen graag willen behouden. Deze worden hieronder in meer detail beschreven.

C.4.3.1. Hoe moeten web API's bij Achmea werken?

Naast dat web API's onderhevig zijn aan de globale eisen en richtlijnen, gelden er ook requirements die specifiek op het ontwerpen en ontwikkelen van web API's zijn gericht.

De referentie softwarearchitectuur stelt: (Quakernaat & Thissen, 2014, p. 14)

> "*De Web API kan data presenteren en bevat geen complexe logica of orchestratie. Het component is eenvoudig in gebruik omdat Achmea specifieke kennis niet is vereist.*"

De Web API Solution Architectuur (Achmea, 2014) legt de volgende beslissingen vast:

- Bouw een afnemer-onafhankelijke web API.
- Gebruik OAuth 2 voor authenticatie, de OAuth client credential flow voor anonieme toegang.
- Gebruik het JWT token formaat.
- Gebruik de CORS functionaliteit van het .NET Web API 2 framework.
- Voor websites, gebruik een generiek proxy component.
- Host web API's op een toegewijde IIS tier, buiten SharePoint.

Ook worden aanbevelingen gemaakt:

- Gebruik het ASP.NET Web API framework.
- Schrijf RESTful of REST-like services.

Hierbij moet identiteit- en toegangsmanagement claims-based zijn, en gehouden aan de Achmea Standaard claim-set. (Roes & Thissen, 2012) Een claim is een statement wat een entiteit (bijvoorbeeld persoon of organisatie) over zichzelf maakt, zoals naam of groep. Deze claims worden door een "issuer" verpakt in security token en uitgegeven. Claims-based applicaties gebruiken de gegevens uit deze claims voor authenticatie en autorisatie. Dit heeft als doel om authenticatie en autorisatie los te koppelen van de applicaties die het nodig hebben. (Microsoft, z.j.) De exacte werking van claims binnen het bedrijf zijn vastgelegd in het document over de Achmea Standaard claim-set.

Over het hosten van web API's zijn reeds de volgende beslissingen gemaakt: (Achmea Design Authority, 2016)

- Per merk een IIS site met virtuele folder voor de API.
- Daarbinnen een virtuele folder per component.
- Daarbinnen een IIS applicatie per versie van de API.
- Elke applicatie heeft een eigen applicatie pool met een eigen gebruikersaccount.
- Generieke API's (zonder merk-specifieke configuratie) komen in de achmea-api container.
- Componenten met merk-specifieke configuratie gaan naar de merk container.

Dit bepaalt ook hoe de URI's van web API's er uit komen te zien, en dat versionering via de URI's gebeurt.

Daarnaast zijn er generieke errorcodes vastgelegd welke gehanteerd moeten worden binnen zowel web als domein API's. (Legtenberg & van Sluijsveld, 2016)

C.4.3.2. Wat moet uit de huidige architectuur behouden worden?

Hoewel de huidige architectuur problemen bevat, worden er ook voordelen gezien die graag behouden blijven: (persoonlijke communicatie, 25 oktober 2016)

- Er worden generieke componenten en andere software-stukken gebruikt.
- Stubbing is makkelijk.
- De frontend is ontkoppeld.
    - De frontend kan onafhankelijk gedeployed worden.
- De architectuur is gelaagd.
- De transactie-laag is onafhankelijk van SharePoint: geschreven als generieke ASP.NET code, kan op andere systemen neergezet worden.
- Inversion of Control voor onder andere configuratie (hoewel dit soms onnodig wordt toegepast).
- De Facade levert meerdere voordelen:
    - Maakt het makkelijk meerdere services aan te roepen.
    - Verrijkt verkregen data.
    - Voert business logica uit.
- Sandbox wordt gebruikt voor het deployen van content.

Vanuit het oogpunt van haalbaarheid mogen de services (welke zich onderaan de architectuur bevinden) niet aangepast worden.

C.4.4. Welke wijzigingen moet de architectuur binnen welk tijdsbestek ondersteunen?

De volgende wijzigingen moeten ondersteund worden: (persoonlijke communicatie, 25 oktober 2016)

- Service wijzigingen:
    - Nieuwe service (interface).
    - Gewijzigde service (interface).

    - Service (interface) vervalt.
- Web API wijzigingen:
    - Nieuwe operatie.
    - Gewijzigde operatie.
    - Verwijderde operatie.
- Gedrag van de (frontend) schermen verandert.
- Wijzigingen van pagina-inhoud en de presentatie (styling) daarvan.

Het tijdsbestek waarbinnen deze wijzigingen doorgevoerd moeten worden is deels afhankelijk van versionering. Wanneer dit wordt toegepast kan een service of web API nieuwe of aangepaste functionaliteit aan bieden zonder de oude functionaliteit direct te laten vallen. Afnemers hebben dan tijd om de nodige wijzigingen te maken totdat de oude functionaliteit niet langer aangeboden wordt.
De gewijzigde functionaliteit zelf mag maximaal evenveel tijd kosten als binnen de huidige architectuur nodig is.

C.4.5. Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur design best practices?

TOGAF (The Open Group Architecture Framework) is een framework wat principes biedt voor het ontwerpen, ontwikkelen en beheren van enterprise architectuur. DYA (Dynamic Enterprise Architecture) is ook een enterprise architectuur framework, maar focust zich meer op software design in het algemeen. Beiden zijn ontwikkeld door grote, vertrouwde groepen of bedrijven.
De frameworks zijn sterker gericht op de business architectuur dan de software architectuur, maar bieden voor dit tweede wel bruikbare patronen.

Voor het oplossen van veelvoorkomende problemen in de softwarearchitectuur kan naar bekende patronen gekeken worden. In de enterprise context is *Patterns of Enterprise Application Architecture* van Martin Fowler een goede bron. *Design Patterns* van de “Gang of Four” blijft op dit gebied ook relevant. (F. Verbruggen, persoonlijke communicatie, 21 oktober 2016)

C.4.6. Wat zijn de best practices op het gebied van architectuur design?

Voor de meeste problemen in software architectuurontwerp zijn reeds oplossingen bedacht. In plaats van het wiel opnieuw uit te vinden kan voor oplossingen naar bestaande software patronen gekeken worden. Hierbij is het wel van belang om patronen alleen toe te passen waar zij nodig zijn, niet overal waar mogelijk.

M. Fowler (Patterns of Enterprise Application Architecture, 2002) raadt het opsplitsen van de architectuur in verschillende, ontkoppelde lagen aan. Hierbij wordt genoemd dat de presentatie-laag veelal onafhankelijk kan zijn van de onderliggende lagen.

Ook concludeert hij dat wanneer de domein logica van een applicatie erg complex is het Domein Model patroon makkelijker te implementeren en onderhouden is dan andere patronen voor het organiseren van domein logica.

Betreft de procesvoering rondom architectuur worden er vanuit de Agile methodiek acht principes gesteld: (Leffingwell, 2011, p. 390)

- De teams die het systeem programmeren, ontwerpen het systeem ook.
- Bouw de simpelste architectuur die kan werken.
- Bij twijfel, programmeer of modelleer het.
- Zij bouwen het, zij testen het.
- Hoe groter het systeem, hoe groter de runway (de mogelijkheid om nieuwe functionaliteit te implementeren zonder veel refactoring).
- Systeemarchitectuur is een samenwerking.
- Er bestaat geen monopolie op innovatie.
- Implementeer architecturale flow (continuele verbetering van het proces).

C.4.7. Welke mogelijke architecturen zijn er?

Voordat een complete doelarchitectuur ontworpen wordt moeten er beslissingen gemaakt worden over de opzet hiervan, en de algehele werking van een centraal onderdeel, de web API's.

C.4.7.1. Waar moet de grens voor de domein API's komen?

Binnen de huidige architectuur vormt het Transactie Component de domein API. Het wordt door de portalen gebruikt voor alle aanroepen naar de backend services. Dit komt overeen met het originele ontwerp, op splitsing per domein na.

Wanneer de domein API grens lager gelegd wordt zullen de verantwoordelijkheden van de domein API (business logica, caching, etc.) dichter bij of in de services komen. Dit vergroot koppeling en moet dus vermeden worden.

C.4.7.2. Waar moet de grens voor de web API's komen?

De web API moet boven de domein API komen te liggen om aan de eisen te kunnen voldoen. De web API mag geen business logica bevatten (dit moet in de domein API blijven), maar dit is wel nodig om service data te aggregeren en presenteren.

De simpelste oplossing is om de web API's aanroep te laten doen op domein API's om zo de opgevraagde data te verzamelen en operaties aan te sturen. Zo blijft de web API laag zo dun en simpel mogelijk, wat onderhoud vergemakkelijkt.

C.4.7.3. Hoe kunnen de web API's gegroepeerd worden?

Volgens de eisen die aan web API hosting gesteld worden moeten de API's per merk, domein en component gegroepeerd worden. Hierbij is de splitsing van componenten afhankelijk van hoe dit binnen de architectuur gebeurt.

Idealiter worden de web API's gegroepeerd op basis van losstaande entiteiten. De manier waarop dit momenteel gebeurt in de domein API is gegeven in Tabel 3, waarbij ook de ondersteunende services genoemd worden. De entiteiten zelf staan los van elkaar, maar de onderliggende aanroepen overlappen soms (deels). (R. van 't Klooster, persoonlijke communicatie, 15 november 2016)

Tabel 3 – Domein entiteiten en hun bronservices

Entiteit	**Service(s)**
Polis/Schade	IKAZ Polis / IKAZ Schade
Financieel	SAP CD / SAP CRM
Documenten	FileNet / OpenText
Relatie	Click / Click ODS / SAP CRM
Collectiviteit	Scope / IKAZ
Voorkeuren	Communication Manager / IKAZ
Declaraties	IDNS
Suggesties	Database
Logging	Database

In eerste instantie is het voornamelijk belangrijk dat de web API's aansluiten op de functionaliteit die de portalen nodig hebben. Hiertoe zou het voordelig zijn als de web API's zodanig simpel te implementeren zijn (bijvoorbeeld, alleen aanroepen op de domein API doen en die data aggregeren) dat de ontwikkelteams van de verschillende portalen zelf de web API's die zij nodig hebben kunnen implementeren. Zo kunnen zij zelf de functionaliteiten definiëren welke de door hun gebruikte web API nodig heeft.

Uiteraard is het spiegelen van de domein API ook een optie, maar dit zal niet automatisch aansluiten bij het uiteindelijke gebruik van de web API's en introduceert te veel vereiste kennis en afhankelijkheid van het domein.

Afhankelijk van de wensen voor web API's die door derden gebruikt kunnen worden, kunnen de ontstane web API's doorontwikkeld worden om meer generiek te zijn en beter aan te sluiten bij de klant events. Hierbij kan gekozen worden om een volledig RESTful (in plaats van REST-like) API te ontwerpen, waarin aanroeppunten gebaseerd zijn op data in plaats van functionaliteiten. Op basis van de transacties welke in hoofdstuk C.2.1 uitgewerkt zijn komen de volgende groepen naar voren:

- klanten
- polissen
- behandelingen
- pgbs
- declaraties
- premies
- eigenrisicos

C.4.7.4. Hoe kan het afwijken van de bedoelde architectuur voorkomen worden?

Het overgrote deel van de huidige afwijking van de architectuurstandaard vindt plaats in verouderde code welke nooit naar de nieuwe standaard is gehaald. Om dit te voorkomen wordt, als onderdeel van de architectuur en het migratieplan daarnaar, het updaten van bestaande code naar nieuwe standaarden meegenomen.

Nieuwe afwijkingen kunnen ontstaan door verschuivende prioriteiten, of wanneer niet-ondersteunde functionaliteit geïmplementeerd moet worden. Om dit te voorkomen moet de architectuur flexibel genoeg zijn om alle verwachte aanpassingen ondersteunen.

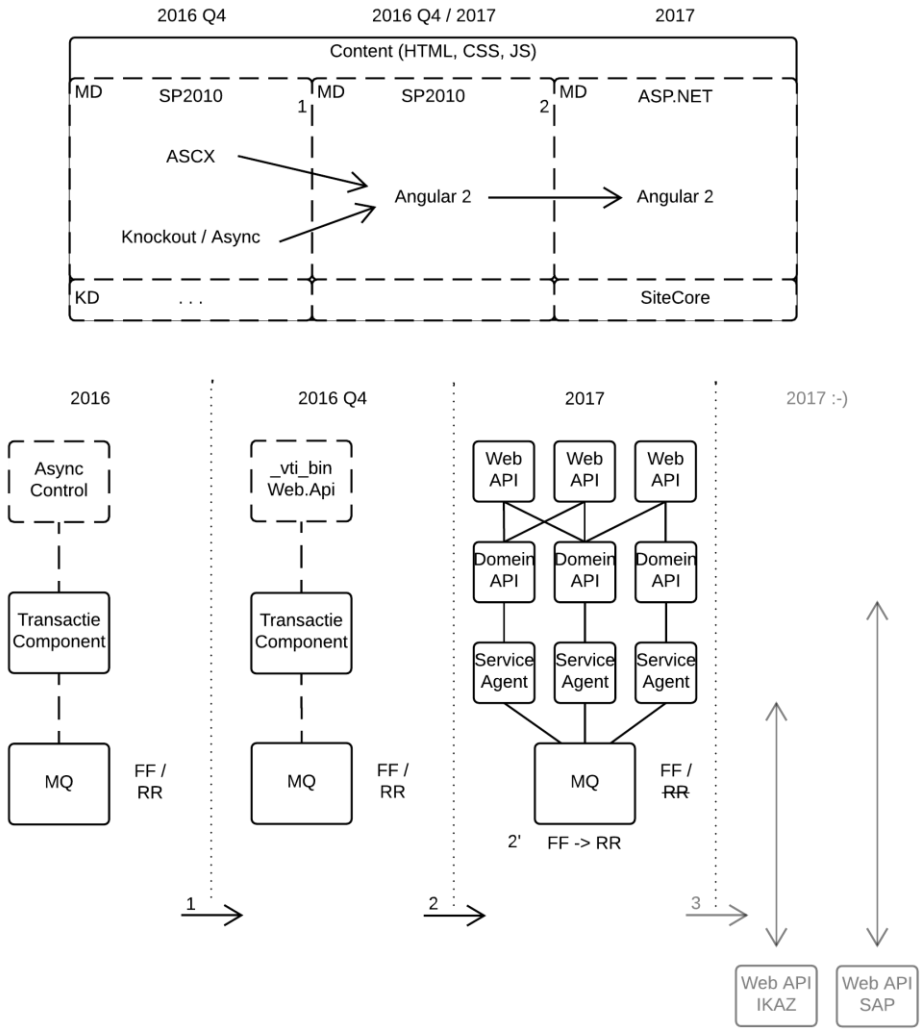
Naast architecturale oplossingen kan procesverbetering ook uitkomst bieden. Door communicatie tussen de verschillende verantwoordelijken deel te maken van het ontwerp- en ontwikkelproces kan voorkomen worden dat de aansluiting tussen delen van de architectuur verslechterd.

C.4.7.5. Mogelijke architecturen

De volgende doelarchitecturen zijn door betrokkenen als mogelijkheden voorgesteld. (Persoonlijke communicatie, 1 november 2016) De voor- en nadelen die elke architectuur met zich mee brengt zijn ook vermeld. Hierbij zijn de opgeloste problemen en voldoening aan de eisen buiten beschouwing gelaten. Deze komen in de beoordeling van de architecturen terug.

C.4.7.5.1. Doelarchitectuur 1

Figuur 13 – Doelarchitectuur 1



Doordat de frontend en middleware ontkoppeld blijven kunnen deze onafhankelijk van elkaar gemigreerd worden. Er wordt een gefaseerde migratie voorgesteld.

Frontend:

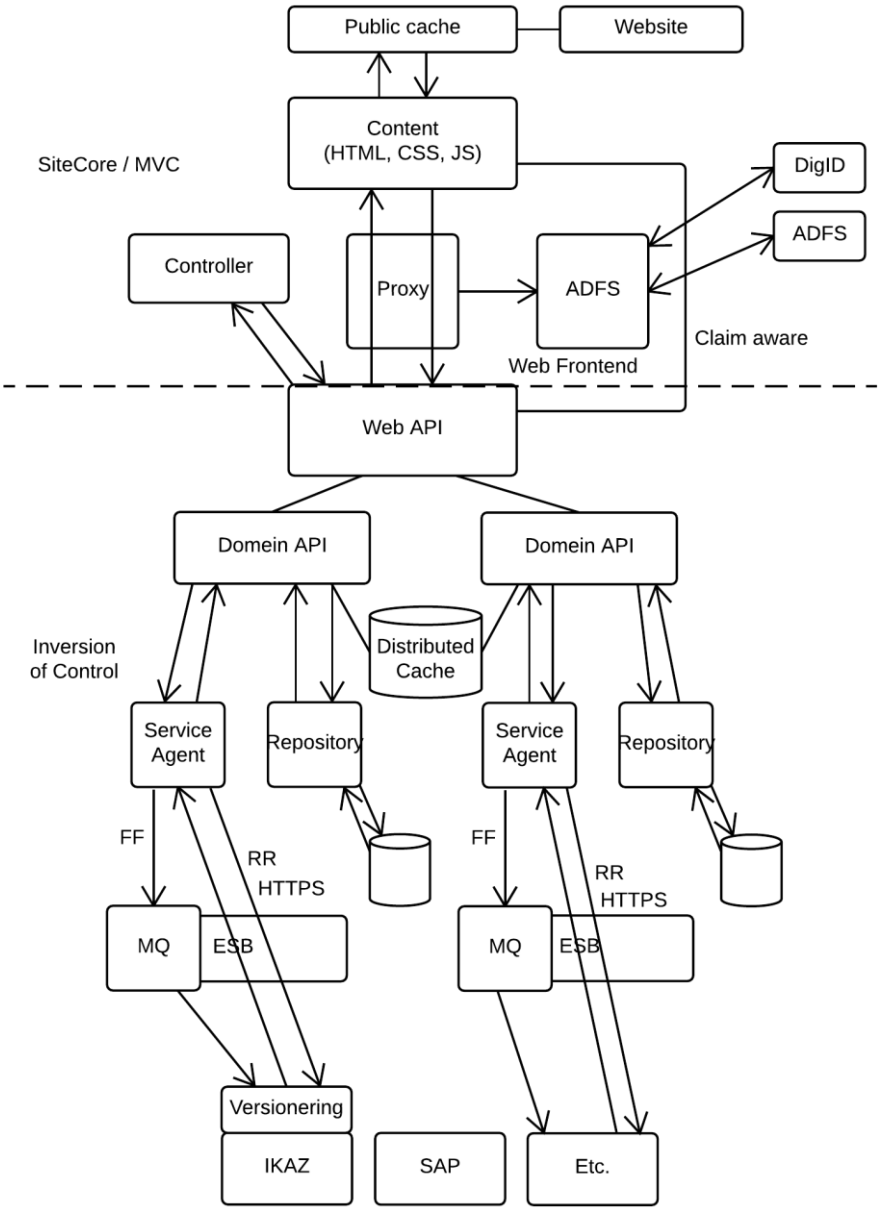
1. De web interfaces stappen over naar Angular 2.
2. Het medewerkersdomein stapt over op ASP.NET. Het klantdomein stapt over op Sitecore.

Middleware:

1. Het Transactie Component wordt aangeboden als web service door middel van SharePoint's _vti_bin folder.
2. Het Transactie Component wordt opgesplitst in verschillende domein API's. Deze worden door web API's aangeroepen om functionaliteit naar buiten toe aan te bieden.
   Hierbij wordt de MQ beperkt tot Fire-and-Forget berichten. Waar mogelijk/nodig worden aanroepen omgezet naar Request/Reply.
3. De services bieden hun eigen web API's aan, welke direct naar buiten ontsloten kunnen worden.

C.4.7.5.2. Doelarchitectuur 2

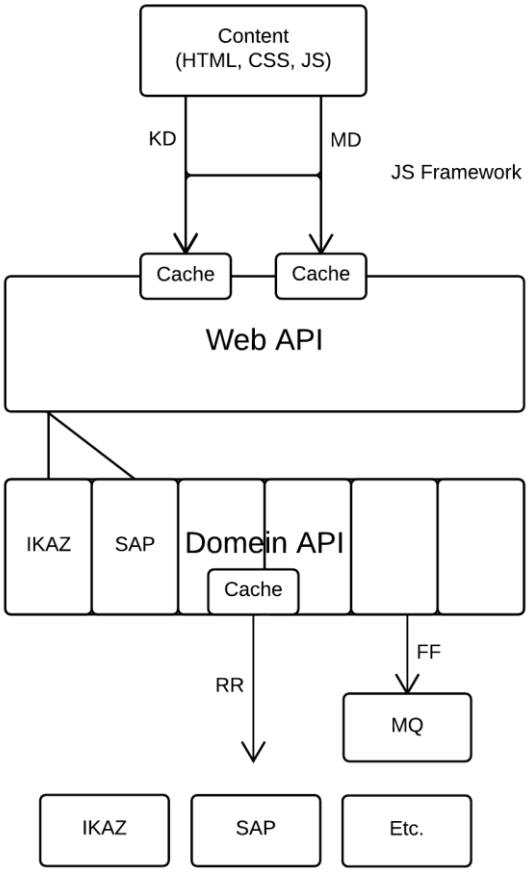
Figuur 14 – Doelarchitectuur 2



De frontend berust op Sitecore. De website maakt gebruik van een cache voor het leveren van pagina's. Er wordt een proxy gebruik voor het ontsluiten van de web API's. Deze proxy maakt gebruik van ADFS om authenticatie te bieden.
Voor de domein API's wordt de bestaande Transactie Component structuur aangehouden, maar wel fijner opgesplitst. De MQ wordt enkel nog gebruikt voor Fire-and-Forget berichten, Request/Reply berichten gaan middels HTTPS direct naar de services toe.

C.4.7.5.3. Doelarchitectuur 3

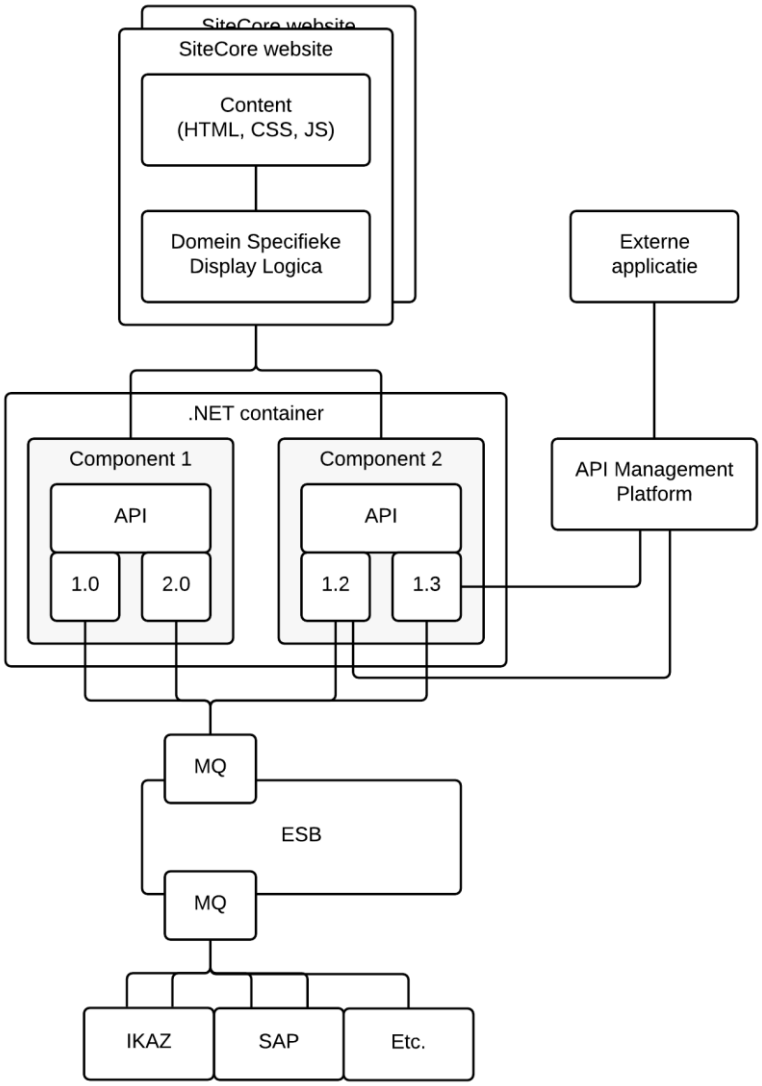
Figuur 15 – Doelarchitectuur 3



De frontend doet aanroep op web API's, welke antwoorden cachet. De web API's roepen domein API's aan, opgesplitst per service, welke de resultaten uit de services cachen. Fire and Forget berichten blijven via de MQ gaan, Request/Reply berichten gaan direct naar de services.

C.4.7.5.4. Doelarchitectuur 4

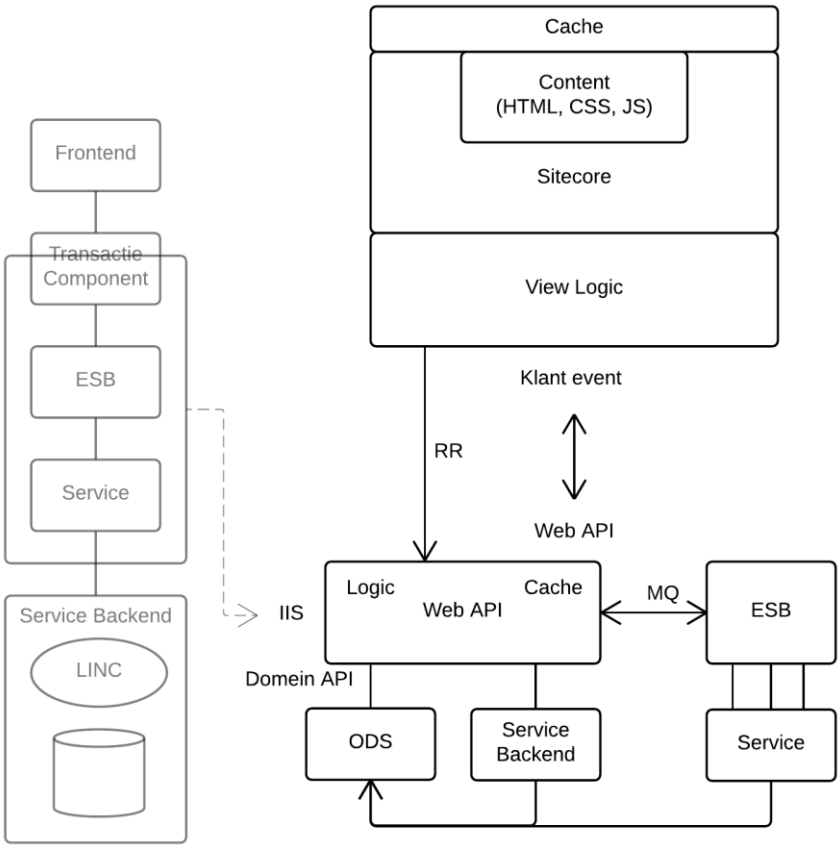
Figuur 16 – Doelarchitectuur 4



De referentie doelarchitectuur welke bij C.3.7 reeds besproken is.

C.4.7.5.5. Doelarchitectuur 5

Figuur 17 – Doelarchitectuur 5



Web API's worden aangeboden op basis van klant events en zijn geïmplementeerd op IIS. Deze web API's doen waar mogelijk (en alleen voor onveranderlijke data) aanroep op een ODS (operational data store), welke een spiegeling van de service databases bevat. Hierdoor kunnen de ESB en service overgeslagen worden, wat performance verhoogt en de werklading van de service verlaagt. Wel moet er nieuwe code geschreven worden om de synchronisatie van service data naar de ODS te voorzien. Waar dit niet mogelijk is blijft de huidige communicatie via de ESB in gebruik. Voor IKAZ wordt alle service logica en business logica naar de web API verhuist, om daarmee direct aanroep te doen op de service database.
Sitecore's caching functionaliteit wordt gebruikt. De web API's zelf bevatten caching van hun resultaten.

C.4.7.5.6. Beoordelingen

Deze doelarchitecturen zijn door betrokkenen beoordeeld op verschillende categorieën. (Persoonlijke communicatie, 1 november 2016) In Tabel 4 worden de gemiddelde beoordelingen gegeven (afgerond op twee decimalen). Hierin zijn cijfers met een significante afwijking (minimaal 0,5) van het gemiddelde van die categorie met groen en rood aangegeven voor een negatieve en positieve afwijking respectievelijk.

De score voor “Duidelijkheid & volledigheid” is voor de rest van het onderzoek onbelangrijk. De beoordeling bij “Vernieuwing” is niet per definitie een indicatie van verbetering.

Tabel 4 – Beoordeling mogelijke doelarchitecturen

Categorie	**1**	**2**	**3**	**4**	**5**
Duidelijkheid & volledigheid	7,6	8	6,5	6,9	7,6
Lage koppeling/hoge modulariteit	7,3	7,7	7,5	7,2	7,6
Testbaarheid	7	6,5	7	6,8	6,9
Probleemoplossing	6,4	7,8	7,3	6,6	8,1
Vernieuwing	7,6	7,1	7,2	7,1	8,5
Migratieplan	7,9	4,8	4,2	4,7	6,8
Gemiddeld	**7,3**	**7**	**6,6**	**6,5**	**7,6**

C.4.7.5.7. Een combinatiearchitectuur

Op basis van de voorgestelde doelarchitecturen wordt een combinatiearchitectuur opgesteld welke verschillende onderdelen van de doelarchitecturen gebruikt om een geheel te vormen wat beter scoort dan de individuele onderdelen.
Er wordt gekeken naar veelvoorkomende en unieke architectuurelementen, en er worden afwegingen gemaakt tussen verschillende technologieën.

Sommige veranderingen komen in meerdere doelarchitecturen terug. Dit kan gezien worden als een lijst van gewenste aanpassingen.

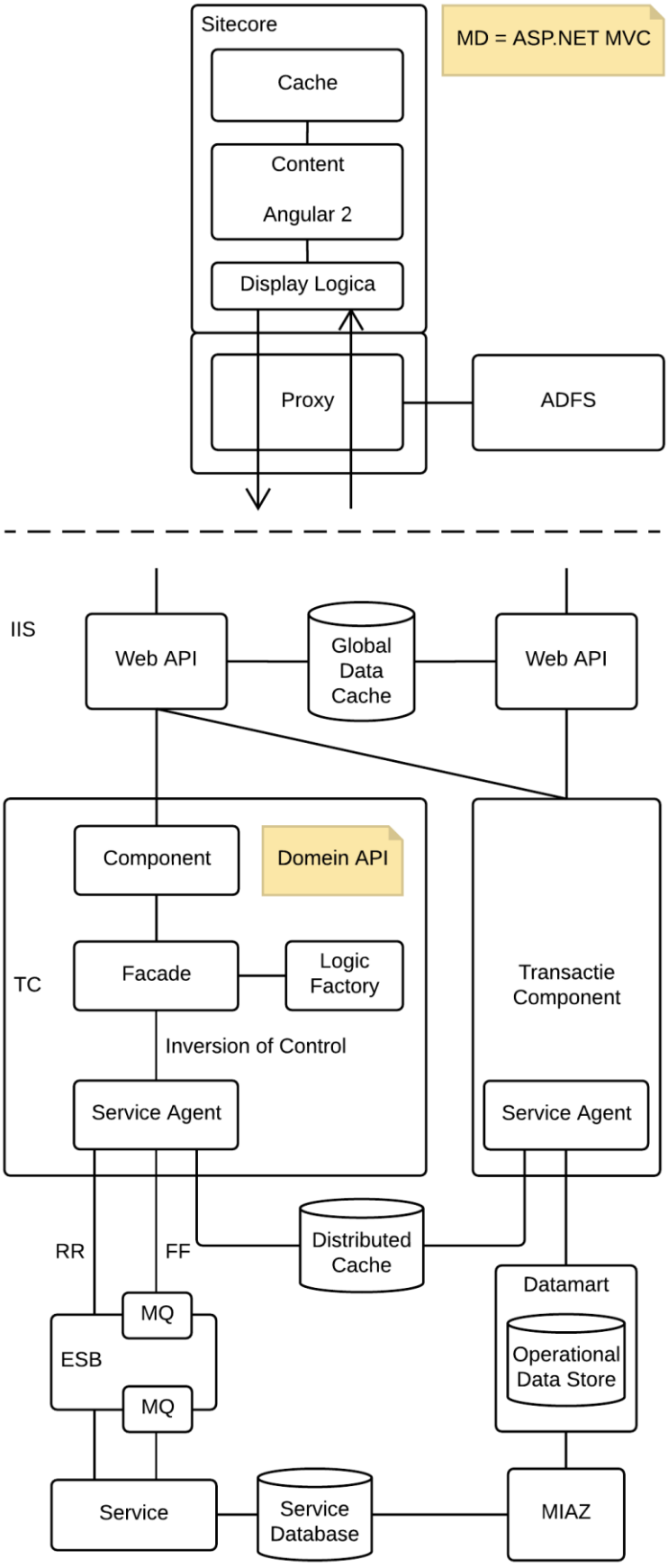
- Frontend cache van Sitecore. (2, 5)
- Gesplitste domein API. (1, 2, 3, 4, 5)
- MQ alleen voor Fire and Forget berichten. (1, 2, 3, 5)

Daarnaast zijn er enkele eigenschappen welke in een enkel ontwerp terug te vinden zijn, maar significant genoeg in hun impact om te overwegen:

- Gefaseerde migratie. (1)
- ASP.NET MVC voor het medewerkerdomein. (1)
- Web API proxy en authenticatie. (2)
- Object Data Store. (5)
- Ontwijken ESB, directe service DB aanroep. (5)

In Figuur 18 wordt de bedachte combinatiearchitectuur weergegeven. De keuzes welke gemaakt zijn om tot deze architectuur te komen zijn in C.4.7.6 verder toegelicht.

Figuur 18 – Combinatiearchitectuur



- **Frontend**, Sitecore (MD: ASP.NET MVC):
    - **Caching**: Geïmplementeerd middels Sitecore's caching functionaliteit, voor het context-bewust cachen van web pagina's en hun content.
    - **Angular 2**: Ter vervanging van Knockout JS. Realiseert de logica welke pagina's ondersteund.
    - **Display logica**: Compatibel met Sitecore. Bevat geen business logica meer. Zorgt dat pagina's bij het initiële laden al domein data bevatten, in plaats van dit pas na het laden via JavaScript (middels de web API) op te halen.
    - **Proxy**: Beveiligt de web API en is verantwoordelijk voor authenticatie.
- **Transactie-laag**, IIS:
    - **Web API's**: Dunne laag bovenop de domein API (Transactie Componenten). Bevatten geen display of business logica, en doen alleen data uit de domein API aggregeren en aanbieden. Portaal-generieke, data-gebaseerde interfaces. Worden geversioneerd.
    - **Transactie Componenten**: Structuur behouden. Opgesplitst per domein. Vormen de domein API's, onafhankelijk van elkaar. Worden geversioneerd.
        - **Service Agents**: Ondersteunen waar nodig het halen van data uit datamarts.
    - **Caching**: Ter behoeven van performance. Verschillende caches voor verschillende soorten data:
        - **Global Data Cache**: Voor generieke gegevens.
        - **Distributed Cache**: Voor service data, waaronder klant specifieke gegevens.
    - Generieke componenten (cross-cutting, domeinmodel, etc.) zijn behouden.
- **Integratie-laag en services**:
    - **ESB & MQ**: Blijven behouden. Waar mogelijk worden Request/Reply berichten gebruikt en de MQ's ontweken.
    - **MIAZ**: Reeds bestaande software voor het aanbieden van versimpelde service data.
    - **Datamart**: Bevat onveranderlijke data uit de services en maakt deze makkelijk toegankelijk. Dit vermindert de werklading op de services. Performance voor zowel datamart- als service-aanvragen zal verbeteren.

C.4.7.6. Keuzeverantwoording

Bij het ontwerpen van de doelarchitectuur zijn afwegingen gemaakt over de te gebruiken platformen en frameworks. Om te bepalen of oplossing wel of niet gebruikt moet worden zijn voor- en nadelen tegenover elkaar gezet. Hieronder is dit alles uitgewerkt.

C.4.7.6.1. JavaScript framework

Momenteel worden de web pagina's van (de meeste) portalen ondersteund door JavaScript middels het Knockout JS framework. Er is door het bedrijf reeds onderzoek gedaan naar Angular 2 als volgende stap in het modern houden van de frontend. In Tabel 5 zijn de belangrijkste aspecten van de twee frameworks tegenover elkaar gezet. (R. ter Velde, persoonlijke communicatie, 17 november 2016)

Tabel 5 – Knockout JS tegenover Angular 2

Criteria	**Knockout JS**	**Angular 2**
Applicatiestructuur richtlijnen	Geen.	Aanwezig.
Programmeren op…	Lager niveau.	Hoger niveau.
Testbaarheid	Geen verbetering ten opzichte van standaard JavaScript.	Dependency injection vergemakkelijkt testen.
Toekomstvastheid	Geen bijzonderheden ten opzichte van standaard JavaScript.	Werkt goed samen met Web Components.
Populariteit (Google, z.j.)	Lager.	Hoger.

Op basis van bovenstaande punten kan gezegd worden dat **Angular 2** de betere optie is. De goede testbaarheid en banden die het legt aan applicatiestructuur maken het beter geschikt voor enterprise toepassingen.

C.4.7.6.2. (Web)applicatie platformen

Voor het beheren van de portalen wordt momenteel SharePoint gebruikt. Binnen het bedrijf is reeds besloten dat er naar Sitecore overgestapt gaat worden. Toch zal hier kort naar gekeken worden om te verifiëren dat dit inderdaad een juiste keuze is. In Tabel 6 zijn de belangrijkste aspecten van de twee platforms tegen elkaar uitgezet. (B. Neijssen, persoonlijke communicatie, 22 november 2016)

Tabel 6 – SharePoint tegenover Sitecore

Criteria	SharePoint	Sitecore
Gemaakt voor	Collaboratie sites.	Web content management sites.
Context-bewuste caching	Nee.	Ja.
Ondersteuning voor mobiele websites	Nee.	Ja.
Marketing toepassingen	Geen.	Lokalisatie, personalisatie, analytics, A/B testing.
Ondersteuning	Grotere community, betere officiële ondersteuning.	Kleine community.
Populariteit (Google, z.j.)	Hoger.	Lager.

Sitecore biedt vele functionaliteiten die beter aansluiten op de huidige problemen en use cases (performance, testen, personalisatie) dan wat SharePoint biedt. Dit maakt het de betere keuze.

Wanneer Angular 2 binnen een Sitecore context gebruikt wordt is het niet nodig om rekening te houden met de manier waarop hun verschillende werkwijzen samengaan. Wanneer een enkele grote, single-page Angular applicatie gerealiseerd worden gaan veel van de voordelen die Sitecore levert verloren. B. Lijten (persoonlijke communicatie, 13 december 2016) raadt aan meerdere kleine Angular applicaties te realiseren binnen Sitecore's rendering stap, om zo nog Sitecore's functionaliteiten toe te kunnen passen. Dit betekend wel dat de Angular applicaties niet volgens hun officiële werkwijze geïmplementeerd worden.

Het medewerkerdomein (MD) is een portaal wat alleen voor medewerkers toegankelijk is. Hierdoor zijn veel functionaliteiten welke Sitecore faciliteert, zoals personalisatie, analytics en (tot zekere hoogte) performance niet van belang. Dit maakt het mogelijk om MD als ASP.NET MVC applicatie te implementeren. In Tabel 7 worden van beide mogelijkheden de voordelen gegeven welke van toepassing zijn op MD. (B. Neijssen, persoonlijke communicatie, 22 november 2016)

Tabel 7 – Sitecore voor MD tegenover ASP.NET MVC voor MD

Sitecore MD	ASP.NET MVC MD
• Vergemakkelijkt het maken van aanpassingen door niet-technische gebruikers.	• Geen afhankelijkheden van of structuur opgedwongen door het platform.

Naast het hierboven genoemde punt biedt Sitecore geen functionaliteit welke voor het medewerkerdomein nuttig zou zijn. Het niet hanteren van Sitecore voor MD houdt de implementatie flexibel, en het implementeren van functionaliteit die gemist wordt zou niet veel extra werk opleveren. (R. ter Velde, persoonlijke

communicatie, 23 november 2016) Gezien het gebruik van Sitecore geen significante voordelen met zich mee brengt heeft **ASP.NET MVC** de voorkeur voor MD.

Over het gebruik van een IIS server voor de transactie-laag wordt een vergelijkbare overweging gemaakt. Hierbij is de keuze tussen de transactie-laag als SharePoint applicatie blijven gebruiken, of als ASP.NET applicatie op een IIS server zetten. Hier zijn enkele SharePoint functionaliteiten reeds in gebruik (Property Bags, Secure Store en het logging mechanisme), maar het verhelpen van deze afhankelijkheden zal niet veel kosten. (R. ter Velde, persoonlijke communicatie, 23 november 2016) Omdat het wegstappen van SharePoint ook voor de transactie-laag restricties verhelpt, is het beter om naar een platform-onafhankelijke ASP.NET applicatie op een **IIS server** te gaan.

C.4.7.6.3. Overkoepelende domein API

Het is mogelijk om de Transactie Componenten (domein API's) door een overkoepelende domein API te laten ontsluiten, in plaats van de web API's direct op de TC's aanroep te laten doen. Deze domein API zou de data aggregatie verantwoordelijkheid van de web API's overnemen. In Tabel 8 zijn de voor- en nadelen van het gebruik van een overkoepelende domein API gegeven.

Tabel 8 – Voor- en nadelen toepassen overkoepelende domein API

Voordelen overkoepelende domein API	Nadelen overkoepelende domein API
• Centrale plek voor data aggregatie. • Conform referentiearchitectuur.	• Minder controle in de web API's met betrekking tot de domein API versie die ze aanroepen. • Meer rompslomp.

Het gebruik van een overkoepelende domein API is in eerste instantie niet nodig. Gezien er geen grote, directe winsten te halen zijn, zal de data aggregatie in eerste instantie in de web API's plaatsvinden. Mocht het nodig blijken, dan kan een overkoepelende domein API altijd nog geïmplementeerd worden.

C.4.7.6.4. Datamart

MIAZ is een systeem wat binnen het bedrijf reeds in gebruik is, en als "data versimpelingssysteem" acteert. Het haalt data uit service databases en andere bronnen en zet dit om naar een logisch datamodel, waarbij alle technische attributen uitgefilterd worden om zo alleen functioneel relevante data aan te bieden.
Met bestaande infrastructuur kan aan MIAZ systeem een zogeheten datamart gekoppeld worden. Dit is een database waarin een subset van de MIAZ data wordt geplaatst zodat applicaties hier aanroep op kunnen doen om de data die zij nodig hebben te verkrijgen. Hier kunnen zij dan ook sortering, filtering, aggregatie en andere operaties op uit laten voeren. (Stored Procedure functionaliteit kan gebruikt worden om hiertoe "interfaces" aan de applicaties aan te bieden.) (C. Piso, persoonlijke communicatie, 30 november 2016)
Het komt met enige regelmaat voor dat het datamodel van services veranderd. Deze veranderingen worden reeds opgevangen door MIAZ. Wanneer datamarts aangeboden gaan worden zal er een goede afstemming tussen de MIAZ beheerders en applicatie ontwikkelaars moeten plaatsvinden om te zorgen dat MIAZ nuttige data blijft aanbieden. Alleen wanneer hier verandering in komt zal de logica welke de datamart ondersteund aangepast moeten worden. (C. Piso, persoonlijke communicatie, 1 december 2016)

In Tabel 9 zijn de voor- en nadelen van het gebruik van een datamart gegeven, ten opzichte van direct aanroep doen op de services. (C. Piso, persoonlijke communicatie, 21 november 2016)

Tabel 9 – Voor- en nadelen toepassen datamart

Voordelen datamart	Nadelen datamart
• Verbeterde performance: data kan direct worden opgehaald in plaats van aan de service opgevraagd en geaggregeerd. • Hogere beschikbaarheid van de data, minder afhankelijkheid van de bronsystemen. • Ontkoppeling: Bij veranderde datawensen (van applicaties) of veranderd datamodel (van services) alleen aanpassing in datamart nodig. • Verantwoordelijkheden voor data ontsluiten en data naar informatie omzetten worden expliciet gemaakt. • Meer technische flexibiliteit (bijvoorbeeld eenvoudiger sorteren, aggregeren en filteren van data). • Geen afhankelijkheid met de afdeling Integratie.	• Data zou niet real-time in de datamart terecht komen, waardoor voor actuele gegevens alsnog service aanroepen gedaan moeten worden. Dit dwingt een hybride-oplossing af, waarin de datamart niet voor alle data aanvragen toegepast wordt. • De verantwoordelijken voor de datamart zullen ook domeinkennis van de services moeten hebben om correct met de data om te kunnen gaan. Dit zorgt voor kennisverspreiding, wat interpretatie-risico met zich mee brengt. • Business regels worden verspreid. • Extra kosten voor het ontwikkelen van iets nieuws. (Software, opleiding.)

De voordelen van het toepassen van een **datamart** wegen op tegen de nadelen omdat vermindering van afhankelijkheden en verhoging van performance, beschikbaarheid en flexibiliteit allen sterk in lijn liggen met de huidige eisen en wensen.
Er is verder onderzoek nodig om te kunnen zeggen welke services een zodanig groot percentage aan onveranderlijke data hebben dat een datamart een positieve baten/kosten verhouding oplevert. Hoe meer onveranderlijke data en hoe vaker deze opgevraagd wordt, hoe groter het voordeel dat een datamart kan leveren.

Het zou ook mogelijk zijn om informatie niet via een datamart op te halen, maar direct aanroep te doen op de databases van de service. In Tabel 10 zijn de voor- en nadelen van deze methode ten opzichte van een datamart geïnventariseerd.

Tabel 10 – Voor- en nadelen directe service database aanroep

Voordelen service database aanroep	Nadelen service database aanroep
• Geen compleet nieuwe ontwikkeling nodig. • Veranderlijke data ook direct toegankelijk.	• Service logica moet naar transactie-laag gehaald worden. (Ter behoeve van data-interpretatie.) • Sterkere koppeling tussen service afnemers en de service data, waardoor aanpassingen nodig zijn bij verandering service datamodel.

Hoewel er kosten bespaard kunnen worden door service databases direct aan te roepen, schaadt dit de voldoening aan de eisen. Een verhoogde koppeling en complexiteit (door verplaatsing van logica, wat afbreuk doet aan de scheiding tussen de lagen van de architectuur) zijn resultaten welke vermeden moeten worden, en de voordelen wegen daar in dit geval niet tegenop.

C.4.7.6.5. Kleinere beslissingen

Caching is op meerdere plekken toegepast zodat er toegespitst wordt op de verschillende soorten data.
De Sitecore cache wordt gebruikt om volledige webpagina's en andere content te cachen.
Voor globale data, waarbij geen afhankelijkheid is van de identiteit welke bij de aanvraag hoort, wordt caching gedaan binnen de web API's, zodat simpele aanvragen direct beantwoord kunnen worden.
Omdat verschillende Transactie Componenten aanroep kunnen doen op dezelfde services maken zij gebruik van een gedistribueerde cache om service-data tijdelijk op te slaan. Wanneer het ene TC gegevens uit een service heeft opgehaald, kunnen andere TC's deze gegevens vervolgens uit de cache halen. Hieronder vallen ook persoonsgebonden gegevens.

De web API's zijn ontsloten door een proxy, wat in lijn ligt met de eisen van Domein Architectuur (zie ook hoofdstuk C.4.1.2). Deze proxy is verantwoordelijk voor authenticatie, wat gebeurt middels Active Directory Federation Services (ADFS). Hier kunnen in de toekomst extra authenticatiemethoden aan toegevoegd worden.
Om deze proxy te implementeren zijn reeds generieke componenten beschikbaar, voor zowel SharePoint als Sitecore. De proxy kost dus geen extra werk. (R. ter Velde, persoonlijke communicatie, 28 november 2016)

De web API's zijn ontworpen en gegroepeerd op basis van data, zoals beschreven in C.4.7.3, ter behoeve van ontkoppeling. De web API's kunnen hierdoor apart geversioneerd worden. Ze worden geïmplementeerd middels het ASP.NET Web API 2 framework, zoals gesteld door de Domein Architectuur eisen.

Het Transactie Component is in meerdere componenten opgesplitst, voor elk domein een TC. Dit realiseert ontkoppeling (ze mogen niet van elkaar afhankelijk zijn) en staat in lijn met het originele ontwerp van de transactie-laag. (van 't Klooster, Definitief Ontwerp Zorg Portalen, 2013, p. 35)

Voor de aanroepen die op services gemaakt moeten worden blijft de ESB bestaan. Het blijft verantwoordelijk voor de transformatie en routering van berichten zodat de Service Agents zich daar niet mee hoeven te moeien.
Wel wordt er gekeken in hoeverre het mogelijk is om Request/Reply berichten direct door de ESB heen te laten gaan, zonder gebruik van de Message Queues. Dit zou performance winst kunnen leveren.

Waar mogelijk zijn Request/Reply berichten gebruikt. Idealiter worden alle mutaties via RR-berichten gedaan, maar dit is niet overal mogelijk door de vertraagde verwerking van aanvragen, vooral wanneer de services zwaarder belast worden. Verder onderzoek is nodig om te bepalen waar Fire and Forget berichten omgezet kunnen worden naar RR-berichten.

Overige zaken welke behouden zijn uit de huidige architectuur zijn onveranderd omdat deze voldoende blijven functioneren en het behalen van de doelstellingen niet belemmeren.

C.4.8. In welke mate voldoen de mogelijke architecturen aan de regels, eisen en randvoorwaarden?

Om de waarden van de verschillende doelarchitecturen te beoordelen is geïnventariseerd welke problemen zij oplossen, en aan welke eisen en randvoorwaarden zij voldoen. De resultaten zijn hieronder in Tabel 11 samengevat. De percentages geven per categorie aan hoeveel doelstellingen een architectuur haalt. Naast het gemiddelde wordt ook een gewogen gemiddelde weergegeven wat de nadruk legt op probleemoplossing.
De volledige beoordelingen van de voorgestelde architecturen zijn terug te vinden in Bijlage A. De inventarisatie van de combinatiearchitectuur is hieronder, in Tabel 12 tot en met Tabel 16, volledig weergegeven.

Tabel 11 – Samenvatting inventarisatie doelarchitecturen

Categorie (totaal)	**1**	**2**	**3**	**4**	**5**	**C**
Problemen (4)	100%	100%	100%	75%	100%	100%
Architectuur-problemen (6)	83%	67%	67%	50%	100%	100%
Eisen (7)	100%	100%	86%	100%	86%	93%
Wijzigingen (4)	100%	100%	100%	100%	75%	100%
Behouden (8)	88%	88%	75%	88%	88%	88%
Gemiddelde	**94%**	**91%**	**85%**	**83%**	**90%**	**96%**
Gemiddelde (problemen x2)	**93%**	**89%**	**85%**	**77%**	**93%**	**97%**

Uit deze beoordelingen blijkt dat de combinatiearchitectuur hogere waarde levert dan de voorgestelde doelarchitecturen. Het lost alle geïnventariseerde problemen op en voldoet aan nagenoeg alle eisen en randvoorwaarden, zoals hieronder toegelicht.

Tabel 12 – Probleemoplossing combinatiearchitectuur

Probleem	Opgelost?	Toelichting
Performance	Ja	Parallelle calls, caching, datamart.
Hot deployment & Time To Market	Ja	Versionering, minder koppeling en Inversion of Control.
Kanaalonafhankelijke bediening	Ja	Web API's zijn platform-onafhankelijk aan te roepen.
Customer journey	Ja	Web API calls tracken.

Tabel 13 – Architectuurprobleemoplossing combinatiearchitectuur

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	Opsplitsing domein API's.
Versionering	Ja	Transactie Componenten niet langer als DLL.
Services & service interfaces	Ja	Waar mogelijk datamart gebruiken. Verder onderzoek naar verbeteringsmogelijkheden van de interfaces.
Testbaarheid	Ja	Opsplitsing domein API's, domein en web API's als testpunten.
Verouderde frontend code	Ja	De frontend wordt vernieuwd.
Gemixte logica	Ja	Opschoning.

Tabel 14 – Voldoening aan de geest van de eisen combinatiearchitectuur

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Ja	Componenten generieke basis voor componenten.
Ontkoppeling	Ja	Splitsing van Transactie Componenten.
Makkelijk nieuwe componenten toevoegen	Ja	Componenten (web API's, Transactie Componenten) bevatten alleen specifieke logica.
Simpel en robuust	Deels	Simpele web API's, splitsing van Transactie Componenten vermindert complexiteit. Maar datamart introduceert nieuwe elementen welke eigen business logica implementeren.
Eén entiteitenmodel	Ja	Ongewijzigd.
Veilig	Ja	Web API proxy.
Personalisatie	Ja	Sitecore bevat personalisatie-functionaliteit.

Tabel 15 – Wijzigingen combinatiearchitectuur

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Ja	Datamart alleen aanpassen bij wijzigingen met betrekking tot “interessante” data.
Web API wijzigingen	Ja	Versionering.
Gedrag van (frontend) schermen	Ja	Wijzigingen in de frontend logica hebben geen impact op de rest van de architectuur.
Pagina inhoud en presentatie	Ja	Content kan (nog steeds) onafhankelijk gedeployed worden.

Tabel 16 – Behouden van aspecten combinatiearchitectuur

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	Cross-cutting, generieke functionaliteit.
Stubbing is makkelijk	Ja	Web API’s makkelijk te stubben, de rest ongewijzigd.
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	Ontkoppeld door web API’s.
De architectuur is gelaagd	Ja	Ongewijzigd.
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	Niet meer op SharePoint geïmplementeerd.
Inversion of Control	Ja	Ongewijzigd.
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	Ongewijzigd.
Sandbox voor het deployen van content	Nee	Geen SharePoint frontend meer.

C.5. Hoe kan van de oude naar de nieuwe architectuur gemigreerd worden?

Omdat de doelarchitectuur sterk afwijkt van de huidige architectuur is een migratieplan nodig aan de hand waarvan naar de doelarchitectuur toe gewerkt kan worden. Om tot een acceptabel plan te komen moeten niet alleen de criteria bekend zijn, maar ook de algeheel gangbare oplossingen voor architectuur migratie.

C.5.1. Welke criteria worden gesteld aan de architectuur migratie?

Bij de architectuur migratie zijn de factoren op te splitsen in twee categorieën: kosten en risico.
Kosten zijn de middelen die besteed worden om de migratie plaats te laten vinden:

- **Tijd:** De lengte van de periode waarin de migratie uitgevoerd wordt, zowel voorbereidend werk als het migreren zelf.
- **Manuren:** De hoeveelheid tijd die aan de migratie besteed wordt door werknemers.
- **Opleiding:** Het trainen van ontwikkelaars om met de nieuwe architectuur te werken.
- **Software:** Softwarepakketten die aangeschaft worden ter behoeve van de nieuwe architectuur.
- **Infrastructuur:** Hardware infrastructuur die aangepast of uitgebreid moet worden ter behoeve van de nieuwe architectuur.

Risico is de kans op verhinderingen en nadelige effecten van de migratie:

- **Vertraging:** De migratie duurt langer dan verwacht.
- **Falen:** De migratie wordt niet voltooid, bijvoorbeeld door onvoorziene problemen of veranderde prioritering.
- **Downtime:** Het systeem is tijdens (delen van) de migratie niet beschikbaar.
- **Bugs:** De migratie introduceert ongewenste functionaliteit.

De verschillende kosten en risico's moeten geminimaliseerd worden, maar liggen daar ook in tweestrijd. Manieren om risico's te verminderen gaan vaak gepaard met een verhoging van de kosten. Er moet een goede balans gevonden worden.

C.5.2. Wat zijn betrouwbare bronnen op het gebied van enterprise architectuur migratie best practices?

Gezien de migratie binnen een Agile software team wordt uitgevoerd is het belangrijk om de migratie hierop aan te sluiten. Hierbij biedt SAFe, het Scaled Agile Framework (Scaled Agile Inc., z.j.), richtlijnen en adviezen met betrekking tot architectuur migratie. Ook het boek Agile Software Requirements (Leffingwell, 2011) bevat hier informatie over.

C.5.3. Wat zijn de best practices op het gebied van enterprise architectuur migratie?

Generiek genomen zijn er twee uiterste soorten migratie te onderscheiden, met beide hun voor- en nadelen. Deze zijn in Tabel 17 uitgewerkt. In principe moet een afweging worden gemaakt tussen de duur van de migratie en het risico ervan.

Tabel 17 – Generieke migratie-methoden

Methode	Beschrijving	Voordelen	Nadelen
Big Bang	De nieuwe implementatie wordt los van de huidige gerealiseerd, en er wordt op één moment (de "big bang") overgestapt.	• Enkel overstapmoment, korte duur. • "Fall back" naar oude systeem simpel.	• Logge overstap, vergroot risico. • Mogelijk lange downtime.
Gefaseerd	De oude implementatie wordt geleidelijk omgezet naar de nieuwe implementatie. Tijdens deze overgang blijft het systeem in gebruik.	• Problemen van nieuwe implementatie worden snel duidelijk. • Tijd en kans om problemen op te lossen. • Modulair, mogelijkheid om migratie tijdelijk stil te leggen.	• Potentieel lange duur. • Onduidelijke mijlpalen, kans op nooit voltooide migratie. • "Fall back" naar het oude systeem wordt elke iteratie moeilijker. • Implementatie moet vaak getest worden.

Over het algemeen zijn Big Bang migraties af te raden voor enterprise systemen. (F. Verbruggen, persoonlijke communicatie, 4 november 2016) De overstap komt op een enkel moment te berusten waardoor het migratiewerk als geheel gedaan worden, wat veel tijd in beslag neemt en risico met zich mee brengt. Een gefaseerde migratie, daarentegen, verspreid tijd en risico over meerdere stappen, welke alleen een werkend geheel vormen. (Leffingwell, 2011, p. 400) Voor enterprise systemen heeft dit de voorkeur.

C.5.3.1. Gefaseerde migratie

Vanuit het oogpunt van de business levert een migratie op zichzelf geen directe waarde. Taken met betrekking tot nieuwe functionaliteiten zullen dan ook prioriteit krijgen over migratie-taken, wat kan leiden tot het nooit volledig afmaken van de migratie. Om de business tevreden te houden tijdens een migratie is het belangrijk om nieuwe functionaliteit te blijven leveren. De migratie-aanpak moet dit faciliteren. Een gefaseerde migratie kan zich hier goed tot lenen. Door vernieuwing om de bestaande architectuur heen te bouwen en deze daarbij geleidelijk uit te faseren, kan tijdens elke iteratie nieuwe functionaliteit ontwikkeld en vrijgegeven worden. (Fowler, Strangler Application, 2004)

D. Leffingwell (2011, pp. 408-409, 423-424) geeft over gefaseerde migratie de volgende adviezen:

- Maak architectuurwerk in uitvoering zichtbaar, om te voorkomen dat degenen die eraan werken te veel werk krijgen.
- Stel limieten aan de hoeveelheid architectuurwerk in uitvoering.
- Stuur op effectieve samenwerking tussen architecten en ontwikkelaars.
- Faciliteer een kwantitatieve basis om beslissingen te maken, zodat de juiste dingen in de juiste volgorde gedaan worden.
- Laat de implementatie van de migratie over aan bestaande interne ontwikkelteams.

C.5.3.2. Realisatie

De realisatie van de nieuwe software-implementatie kan onder één van drie categorieën vallen: (F. Verbruggen, persoonlijke communicatie, 4 november 2016)

- **Herbouw:** Het huidige systeem wordt vanaf de grond opnieuw opgebouwd.
- **Nieuwbouw:** Een nieuw systeem wordt vanaf de grond opgebouwd.
- **Aanpassing:** Het huidige systeem wordt aangepast om tot de doelsituatie te komen.

In de praktijk kosten zowel herbouw als nieuwbouw te veel tijd en brengen te veel risico met zich mee om bruikbare optie te zijn. Het aanpassen van een bestaand systeem sluit beter aan bij gefaseerde migratie.

Tijdens de realisatie kan het mogelijk zijn om code generators toe te passen. Indien stukken code voldoen aan een standaard zou deze automatisch getransformeerd kunnen worden tot de nieuwe implementatie. Afhankelijk van de hoeveelheid code en de eenvoud van de code generator kan veel handmatig werk bespaard worden.

C.5.3.3. Dogfooding

Hoewel het meer deel uit maakt van het algehele ontwikkelproces dan van de migratie zelf, is “dogfooding” aan te raden. Ook wel bekend als “eating your own dog food”, het omvat het interne gebruik van producten en services die ook naar de buitenwereld aangeboden worden.
Door medewerkers hun eigen product te laten gebruiken, komen tijdens de ontwikkeling sneller problemen naar boven die anders pas door de eindgebruikers ontdekt zouden worden. Het dwingt testen en kwaliteitscontrole af, en zorgt dat het product aansluit bij echte gebruiksscenario’s. (Microsoft, z.j.) Daarnaast wordt hergebruik gestimuleerd: medewerkers binnen het ene domein willen toegang hebben tot dezelfde functionaliteiten als medewerkers uit andere domeinen. Dit is het makkelijkst te bereiken wanneer functionaliteit generiek wordt geïmplementeerd.

C.5.4. Welke mogelijke migratie-paden zijn er?

Gezien gefaseerde migraties ontwikkeling niet stil leggen en een laag risico hebben zijn deze de beste optie voor het bedrijf. De mogelijk lange duur van een gefaseerde migratie is een nadeel, maar weegt niet tegen de genoemde voordelen op.

Bij het uitvoeren van de migratie spelen meerdere factoren een rol:

- **De Scrum backlog**: De taken welke op korte termijn gedaan zullen worden beïnvloeden de migratie. Het toevoegen van nieuwe functionaliteiten kan de complexiteit van de architectuur en code verhogen, waardoor de migratie meer tijd zal kosten. Hierdoor is de voorkeur om migratie uit te voeren voordat gerelateerde functionaliteiten worden geïmplementeerd.
- **Prioritering van problemen**: Afhankelijk van de urgentie van het oplossen van de verschillende problemen kunnen bepaalde delen van de migratie voorrang vereisen.
- **Opleiding**: Daar waar de doelarchitectuur sterk afwijkt van de huidige architectuur moeten de ontwikkelaars leren hoe hiermee gewerkt moet worden. Dit kan geleerd worden tijdens het implementeren van de veranderingen.
- **Agile**: Om aan de Agile werkwijze te voldoen moet er altijd een werkend systeem zijn, en zo min mogelijk branching in de codebase.

Hoe deze factoren de migratie precies beïnvloeden wordt door de medewerkers bepaald, en is binnen de context van het onderzoek slecht in te schatten.

Er zijn migratiepaden opgesteld welke aanbevelingen maken over de volgorde en groepering van verschillende wijzigingen, met als doel een flexibel plan aan te bevelen.

C.5.4.1. Migratie van de frontend

De aanpassingen die aan de frontend gemaakt moeten worden zijn in Tabel 18 gegeven, inclusief korte naam ter referentie.

Tabel 18 – Migratiestappen frontend

Korte naam	Beschrijving
Angular	Angular 2 volledig adopteren en alle JavaScript hiernaar overzetten.
Cache	Implementeren van de frontend cache op basis van Sitecore functionaliteit.
Display	Display logica Sitecore compatibel maken (voor MD: ASP.NET MVC). Hierbij wordt business logica weggewerkt. Dit zal geen langdurig of ingewikkeld werk opleveren. (B. Lijten, persoonlijke communicatie, 20 oktober 2016)
Proxy	Proxy voor toegang tot onderliggende web API's. Verantwoordelijk voor authenticatie. Kan zonder extra kosten geïmplementeerd worden omdat er reeds generieke componenten voor bestaan (voor zowel SharePoint als Sitecore). (R. ter Velde, persoonlijke communicatie, 28 november 2016)
Sitecore	Verhuizen van de frontend naar Sitecore (voor MD: ASP.NET MVC).

Proxy moet als eerste, om web API's (en dus ontkoppeling tussen de frontend en middleware) te realiseren. De proxy kan zonder extra kosten geïmplementeerd worden omdat er reeds generieke componenten voor bestaan (voor zowel SharePoint als Sitecore). (R. ter Velde, persoonlijke communicatie, 28 november 2016)
Deze proxy is niet nodig zolang de frontend nog op SharePoint draait én er nog gebruik wordt gemaakt van SharePoint's _vti_bin web API's, omdat aanroepen dan binnen de SharePoint context kunnen blijven. (R. van 't Klooster, persoonlijke communicatie, 29 november 2016)

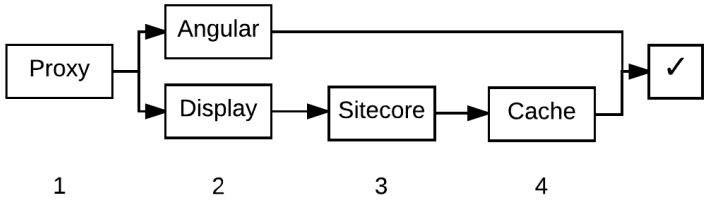
Angular is onafhankelijk, omdat de content onafhankelijk te deployen blijft.

Display en Sitecore hangen samen, omdat er niet naar Sitecore verhuist kan worden zonder display logica die erop kan draaien.

Cache volgt op Sitecore, omdat voor caching de bij Sitecore ingebouwde functionaliteit gebruikt wordt.

Hieruit volgt het migratiepad uit Figuur 19.

Figuur 19 – Migratiepad frontend



C.5.4.2. Migratie van de middleware en backend

De aanpassingen die binnen de middleware en backend gemaakt moeten worden zijn in Tabel 19 gegeven, inclusief een korte naam ter referentie.

Tabel 19 – Migratiestappen middleware en backend

Korte naam	Beschrijving
_vti_bin	Implementeren van web API's op basis van SharePoint's _vti_bin. Tijdelijke tussenstap tot het implementeren van "echte" web API's, wordt reeds aan gewerkt.
Caching	Implementeren van de Global Data Cache en Distributed Cache.
Datamart	Realiseren en gebruiken van een Datamart voor het ophalen van onveranderlijke service data.
IIS	Verhuizen van de transactie-laag van SharePoint naar een ASP.NET applicatie op een IIS server. Dit zal geen lastig of duur werk opleveren. (R. ter Velde, persoonlijke communicatie, 17 november 2016)
RR	Service interfaces versimpelen en waar mogelijk Request/Reply berichten gebruiken die direct via de ESB gaan.
TC's	Opsplitsen van de Transactie Componenten.
Web API	Implementeren van web API's middels het ASP.NET Web API 2 framework.

Caching wordt tijdens andere stappen uitgevoerd, omdat het meerdere onderdelen van de architectuur raakt. Idealiter wordt het direct meegenomen tijdens de realisatie van nieuwe onderdelen (web API's en het Global Data Cache). Wanneer het reeds bestaande onderdelen betreft (Service Agents en hun Distributed Cache) is de plaatsing in de migratie afhankelijk van de eerder genoemde factoren.

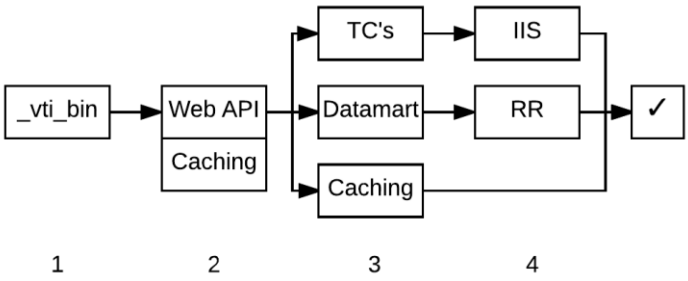
_vti_bin moet als eerste, omdat het aanbieden van een web API de migratie van de frontend ontkoppeld. Het is mogelijk de _vti_bin oplossing over te slaan en direct ASP.NET web API's te implementeren. Mits de interface en het gebruik daarvan consistent kan blijven is het ook mogelijk om web API's onafhankelijk van de rest van de migratie te implementeren.
Tijdens het realiseren van web API's kan ook direct de Global Data Cache toegevoegd worden, gezien er nieuwe onderdelen gerealiseerd worden.

IIS volgt op TC's, omdat eerst de SharePoint-afhankelijkheden uit de transactie-laag weggewerkt moeten worden. Dit wordt idealiter gedaan wanneer de TC's opgesplitst worden, zodat er maar een enkele keer aan gewerkt moet worden.

RR volgt op Datamart, omdat na het in gebruik nemen van de datamart sommige service aanroepen zullen vervallen in ruil voor datamart aanroepen. Om te voorkomen dat werk wordt verricht aan service interfaces welke niet meer in gebruik zullen zijn, wordt de datamart eerst gerealiseerd.

Gezien de Service Agents ontkoppeling vormen tussen de transactie- en integratie-lagen kunnen de aanpassingen aan die lagen onafhankelijk van elkaar worden uitgevoerd.
Hieruit volgt het migratiepad uit Figuur 20.

Figuur 20 – Migratiepad middleware en backend



C.5.5. In welke mate voldoen de mogelijke migratie-paden aan de criteria?

Het accuraat inschatten van kosten valt buiten de scope van het onderzoek, maar de kostenbronnen zijn hieronder wel aangegeven. Voor de risico's is genoemd hoe deze beperkt zijn.

Kosten

- **Tijd**: De duur van de migratie is afhankelijk van het aantal migratiestappen welke (waar mogelijk) tegelijkertijd worden uitgevoerd, en de hoeveelheid werk welke naast de migratie plaats vindt.
- **Manuren**: Het aantal manuren wat besteed moet worden om de migratie te voltooien is afhankelijk van het voordoen van onverwachte problemen. Zie ook de risico's hieronder.
- **Opleiding**: Er moeten verantwoordelijken komen voor de datamart. Zij moeten kennis hebben van de service data om deze juist te kunnen ontsluiten. (C. Piso, persoonlijke communicatie, 21 november 2016) Daarnaast moeten ontwikkelaars met de nieuwe architectuur leren werken, maar deze opleiding kan (deels) tijdens de implementatie plaatsvinden.
- **Software**: Sitecore is reeds aangeschaft. IIS is reeds beschikbaar. De kosten voor SharePoint komen te vervallen wanneer hier volledig van afgestapt is. Software levert dus geen kosten op.
- **Infrastructuur**: Alle benodigde infrastructuur voor het realiseren van een datamart is reeds beschikbaar. (C. Piso, persoonlijke communicatie, 30 november 2016)

Risico's

- **Vertraging**: Verschillende delen van de migratie zijn van elkaar ontkoppeld. Hierdoor worden niet alle delen beïnvloed wanneer een enkele stap vertraging oploopt. Hierdoor blijft de impact van eventuele vertraging beperkt.
- **Falen**: Niet alle delen van de migratie zijn van elkaar afhankelijk, waardoor de kans klein is dat het geheel zal falen. Praktische haalbaarheid wordt na afloop van het onderzoek getest middels een proof of concept.
- **Downtime**: Downtime zal waarschijnlijk plaatsvinden bij overstappen tussen verschillende platform. Van SharePoint naar Sitecore bij de frontend, en naar IIS bij de middleware. Verdere downtime kan met behulp van versionering voorkomen worden.
- **Bugs**: Tijdens de migratie zal middels de huidige (en waar nodig vernieuwde) tests gelet worden op onbedoelde functionaliteit.

C.6. Welke architectuur met bijbehorend migratiepad kan aanbevolen worden?

Bij de beoordeling van de verschillende doelarchitecturen in C.4.8 is gebleken dat de combinatiearchitectuur, kijkende naar probleemoplossing en eisenvoldoening, het hoogste scoort.

Het migratiepad beschreven in C.5.4 is op de combinatiearchitectuur toegespitst, en niet relevant voor (of toepasbaar op) andere architecturen.

In eerste instantie zouden er meerdere architectuur en migratiepad combinaties onderzocht worden, wat antwoord zou geven op de onderstaande deelvragen:

- “Welke migratie-paden zijn mogelijk voor welke architecturen?”
- “Welke architectuur-migratiepad combinatie voldoet het beste aan de eisen?”

Door de hierboven beschreven omstandigheden is dit niet langer nodig.

De in C.4.7.5.7 gegeven architectuur en het in C.5.4 beschreven migratiepad vormen samen het advies wat aan het bedrijf wordt gegeven.

C.7. Bibliografie

Achmea. (2014, maart 5). *Web API Solution Architecture.* Opgeroepen op oktober 27, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/Documents/Achmea%20Web%20API%20Solution%20Architecture_v1.docx

Achmea Design Authority. (2016, februari 24). *Architecture decision - WebAPI Hosting.* Opgeroepen op oktober 24, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/MScc-DA-018%20-%20Hosting%20WebAPI.docx

Achmea. (z.j.). *Bedrijfsarchitectuur.* Opgeroepen op september 21, 2016, van AchmeaNet: P&VM Proces en Data management: https://samenwerken.achmeanet.nl/sites/14079/Insite/Architectuur/aa_reportstart.html

Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Highsmith, J., . . . Thomas, D. (2001). *Principles behind the Agile Manifesto*. Opgehaald van Manifesto for Agile Software Development: http://agilemanifesto.org/principles.html

Bril, G. (2012, december 1). *Technische Domein Architectuur Integratie Services.* Opgeroepen op oktober 5, 2016, van AchmeaNet: https://achmeanet-sites.hosting.corp/com/10753/Architectuur/TDA - Integratie Services v1.0.doc

Bril, G., & Kollau, P. (2008, juli 02). *Integratie Services – Visie op interactie tussen Enterprise services en Enterprise Service Bus (ESB).* Opgeroepen op september 27, 2016, van AchmeaNet: Kenniscentrum Integratie: https://achmeanet-sites.hosting.corp/afd/11402/Openbare%20documenten/Integratie%20Services%20-%20Visie%20ESB%20versie%201.1.doc

Bugayenko, Y. (2014, juli 21). *Master Branch Must Be Read-Only*. Opgeroepen op oktober 17, 2016, van Yegor256: http://www.yegor256.com/2014/07/21/read-only-master-branch.html

Clark, K. (2015, maart 18). *Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration*. Opgeroepen op oktober 17, 2016, van IBM developerWorks: https://www.ibm.com/developerworks/websphere/library/techarticles/1503_clark/1305_clark.html

Dietz, J. L. (2011). *Concise Summary of DEMO-3.* Opgeroepen op september 14, 2016, van Enterprise Engineering Institute: http://www.ee-institute.org/download.php?id=98&type=doc

Fenniak, M. (2013, april 15). *The Web API Checklist*. Opgeroepen op oktober 17, 2016, van Mathieu Fenniak: https://mathieu.fenniak.net/the-api-checklist/

Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional.

Fowler, M. (2004, juni 29). *Strangler Application*. Opgeroepen op november 14, 2016, van MartinFowler.com: http://www.martinfowler.com/bliki/StranglerApplication.html

Fowler, M. (2006, mei 1). *Continuous Integration*. Opgeroepen op oktober 17, 2016, van MartinFowler.com: http://www.martinfowler.com/articles/continuousIntegration.html

Google. (z.j.). *Google Trends - angular, knockout*. Opgeroepen op november 22, 2016, van Google Trends: https://www.google.nl/trends/explore?cat=1227&q=angular,knockout

Google. (z.j.). *Google Trends - sharepoint, sitecore*. Opgeroepen op december 16, 2016, van Google Trends: https://www.google.nl/trends/explore?cat=1227&q=sharepoint,sitecore

Krouwel, M., & Op 't Land, M. (2012, februari 15). *Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems.* Opgeroepen op oktober 14, 2016, van Capgemini Nederland: https://www.nl.capgemini.com/resource-file-access/resource/pdf/Combining_DEMO_and_Normalized_Systems_for_Developing_Agile_Enterprise_Information_Systems_0.pdf

Leffingwell, D. (2011). *Agile Software Requirements.* Westford, Massachusetts, United States: Pearson Education, Inc.

Legtenberg, R., & van Sluijsveld, C. (2016, maart 24). *Generic errorcodes WebAPI's and DomainAPI's.* Opgeroepen op oktober 24, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/Generic%20errorcodes%20WebAPI%27s%20and%20DomainAPI%27s%20-%20R1.1.docx

Lijten, B. (2014, december 12). *Referentie Architectuur.* Opgeroepen op oktober 31, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/Documenten%20SA/archief/Referentie%20architectuur%20-%20concretisering%20deel%201.pptx

Lijten, B., van Loon, G., & van Hooijdonk, S. (2012, april 10). *Achmea SharePoint 2010 Solution Architecture Guidelines.* Opgeroepen op oktober 27, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/SharePoint%202010%20Architecture%20guidelines%20for%20Achmea.docx

Mannaert, H. (2012, november 21). *Normalized Systems: Towards Designing Evolvable Modular Structures.* Opgeroepen op oktober 14, 2016, van International Academy, Research, and Industry Association: http://www.iaria.org/conferences2012/filesICSNC12/Normalized%20Systems%20Towards%20Designing%20Evolvable%20Modular%20Structures.pdf

Microsoft. (z.j.). *About Early Technology Adoption (Dogfooding)*. Opgeroepen op november 11, 2016, van Microsoft TechNet: https://technet.microsoft.com/en-us/library/cc627315.aspx?f=255&MSPPError=-2147217396

Microsoft. (z.j.). *An Introduction to Claims*. Opgeroepen op oktober 27, 2016, van Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ff359101.aspx

Mulloy, B. (2012, maart). *Web API Design: Crafting Interfaces that Developers Love.* Opgeroepen op oktober 17, 2016, van Pearson Developers Network: http://developer.pearson.com/sites/default/files/api-design-ebook-2012-03%20(1)_0.pdf

Narumoto, M. (2016, juli 13). *API design guidance*. Opgeroepen op oktober 17, 2016, van Microsoft Azure: https://azure.microsoft.com/en-us/documentation/articles/best-practices-api-design/

Open API Initiative. (z.j.). *OpenAPI Specification*. Opgeroepen op oktober 19, 2016, van Open API Initiative: https://openapis.org/specification

Piso, C. (2016, mei 19). *Mikado 1.1 (vervanging 1Desk).* Opgeroepen op september 21, 2016, van Hoofd team site voor het Zorg Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Medewerkerdomein/Shared%20Documents/[02]%20Architectuur/PSA/PSA%20Mikado%201.1%20(Vervanging%201Desk).docx

Quakernaat, J., & Thissen, A. (2014, november 06). *Microsoft CC Referentie Architectuur.* Opgeroepen op oktober 3, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/Referentie%20Architectuur%20v2.0%20alpha%20-%20ter%20review.docx

RAML Workgroup. (z.j.). *RAML*. Opgeroepen op oktober 19, 2016, van RAML: http://raml.org/

Roes, E., & Thissen, A. (2012, mei 14). *Achmea Standaard claim-set.* Opgeroepen op oktober 27, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/MSCC%20Standaarden/Achmea_Standaard_Claim-set_1.5.doc

Scaled Agile Inc. (z.j.). *SAFe for Lean Software and System Engineering*. Opgeroepen op november 14, 2016, van Scaled Agile Framework: http://scaledagileframework.com/

ter Velde, R. (2016, maart 31). *Mikado - Solution Architecture Migration Plan - IST.* Opgeroepen op oktober 13, 2016, van Hoofd team site voor het Zorg Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Medewerkerdomein/Shared%20Documents/[02]%20Architectuur/SAD/Concept/Migratieplan_concept_v0.2.pdf

van Alphen, P. (2015, maart 20). *Microsoft Competence Center Glossary voor Zorg.* Opgeroepen op september 22, 2016, van Hoofd team site voor het Zorg

Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Shared%20Documents/Achmea_MSCC_Zorg_Glossary.doc

van den Broek, N. (2015, april 7). *Delta SAD Voorkantvernieuwing.* Opgeroepen op oktober 7, 2016, van Hoofd team site voor het Zorg Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Klantdomein/Shared%20Documents/[02]%20Architectuur/[OVERGEZET]/Ontvlechten/Delta%20SAD%20Voorkantvernieuwing.docx

van Sluijsveld, C., & van der Plas, E. (2016, juli 6). *Design Manifesto.* Opgeroepen op oktober 27, 2016, van AchmeaNet: Design Authority: https://achmeanet-sites.hosting.corp/afd/10687_mscc/DesignAuthority/Documenten%20SA/Design%20Manifesto.docx

van 't Klooster, R. (2013, februari 27). *Definitief Ontwerp Zorg Portalen.* Opgeroepen op oktober 3, 2016, van Hoofd team site voor het Zorg Domein: http://tfs.hosting.corp/sites/Microsoft1/Achmea.CodePlex/Zorg/Generiek/Shared%20Documents/[02]%20Architectuur/Definitief_ontwerp_zorg_portalen.docx

van 't Klooster, R. (2014, februari 5). *cd TotalClass.* Opgeroepen op oktober 3, 2016, van TFS hosting: \tfs\MS1\Zorg.Klantdomein\DEV\Diagrams\Diagrams\TotalClass.classdiagram

Vorgers, P. (2015, december). *Enterprise Integratie Patronen.* Opgeroepen op oktober 5, 2015, van AchmeaNet: https://achmeanet-sites.hosting.corp/com/10949/EA documenten/Enterprise Integratie Patronen v1.60.ppt

C.8. Bijlagen

C.8.1. Bijlage A: Beoordelingen doelarchitecturen

C.8.1.1. Samenvatting

Probleem	**1**	**2**	**3**	**4**	**5**
Performance	Ja	Ja	Ja	Nee	Ja
Hot deployment & Time To Market	Ja	Ja	Ja	Ja	Ja
Kanaalonafhankelijke bediening	Ja	Ja	Ja	Ja	Ja
Customer journey	Ja	Ja	Ja	Ja	Ja

Architectuurprobleem	**1**	**2**	**3**	**4**	**5**
Sterke koppeling	Ja	Ja	Ja	Ja	Ja
Versionering	Ja	Ja	Ja	Ja	Ja
Services & service interfaces	Nee	Nee	Nee	Nee	Ja
Testbaarheid	Ja	Ja	Ja	Ja	Ja
Verouderde frontend code	Ja	Nee	Ja	Nee	Ja
Gemixte logica	Ja	Ja	Nee	Nee	Ja

Eisen	**1**	**2**	**3**	**4**	**5**
Herbruikbare, generieke code	Ja	Ja	Ja	Ja	Ja
Ontkoppeling	Ja	Ja	Ja	Ja	Ja
Makkelijk nieuwe componenten toevoegen	Ja	Ja	Ja	Ja	Ja
Simpel en robuust	Ja	Ja	Ja	Ja	Nee
Eén entiteitenmodel	Ja	Ja	Ja	Ja	Ja
Veilig	Ja	Ja	Ja	Ja	Ja
Personalisatie	Ja	Ja	Nee	Ja	Ja

Wijzigingen	1	2	3	4	5
Service wijzigingen	Ja	Ja	Ja	Ja	Nee
Web API wijzigingen	Ja	Ja	Ja	Ja	Ja
Gedrag van (frontend) schermen	Ja	Ja	Ja	Ja	Ja
Pagina inhoud en presentatie	Ja	Ja	Ja	Ja	Ja

Behouden aspect	1	2	3	4	5
Gebruik van generieke componenten	Ja	Ja	Ja	Ja	Ja
Stubbing is makkelijk	Ja	Ja	Ja	Ja	Ja
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	Ja	Ja	Ja	Ja
De architectuur is gelaagd	Ja	Ja	Ja	Ja	Ja
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	Ja	Ja	Ja	Ja
Inversion of Control	Ja	Ja	Ja	Ja	Ja
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	Ja	Nee	Ja	Ja
Sandbox voor het deployen van content	Nee	Nee	Nee	Nee	Nee

C.8.1.2. Toelichtingen

C.8.1.2.1. Architectuur 1

Probleem	Opgelost?	Toelichting
Performance	Ja	Caching, parallel calls.
Time To Market	Ja	Hot deployment.
Hot deployment	Ja	Gesplitste domein API's, testbaarheid.
Kanaalonafhankelijke bediening	Ja	Web API's.
Customer journey	Ja	Web API calls tracken.

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	Gesplitste domein API’s.
Versionering	Ja	
Services & service interfaces	Nee	Meer RR berichten, maar nog steeds slechte interfaces.
Testbaarheid	Ja	Gesplitste domein API’s, web API’s als testpunt.
Verouderde frontend code	Ja	
Gemixte logica	Ja	

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Ja	
Ontkoppeling	Ja	
Makkelijk nieuwe componenten toevoegen	Ja	
Simpel en robuust	Ja	
Eén entiteitenmodel	Ja	
Veilig	Ja	
Personalisatie	Nee	

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Ja	
Web API wijzigingen	Ja	
Gedrag van (frontend) schermen	Ja	
Pagina inhoud en presentatie	Ja	

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	
Stubbing is makkelijk	Ja	
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	
De architectuur is gelaagd	Ja	
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	
Inversion of Control	Ja	
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	
Sandbox voor het deployen van content	Nee	ASP.NET/SiteCore.

C.8.1.2.2. Architectuur 2

Probleem	Opgelost?	Toelichting
Performance	Ja	Caching, RR direct naar de service.
Time To Market	Nee	
Hot deployment	Nee	
Kanaalonafhankelijke bediening	Ja	Web API's.
Customer journey	Nee	

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	
Versionering	Nee	
Services & service interfaces	Nee	
Testbaarheid	Nee	
Verouderde frontend code	Nee	
Gemixte logica	Nee	

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Ja	
Ontkoppeling	Ja	
Makkelijk nieuwe componenten toevoegen	Ja	
Simpel en robuust	Ja	
Eén entiteitenmodel	Ja	
Veilig	Ja	Web API proxy met authenticatie.
Personalisatie	Nee	

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Ja	
Web API wijzigingen	Ja	
Gedrag van (frontend) schermen	Ja	
Pagina inhoud en presentatie	Ja	

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	
Stubbing is makkelijk	Ja	
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	
De architectuur is gelaagd	Ja	
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	
Inversion of Control	Ja	
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	
Sandbox voor het deployen van content	Nee	SiteCore.

C.8.1.2.3. Architectuur 3

Probleem	Opgelost?	Toelichting
Performance	Ja	Caching, RR direct naar de service.
Time To Market	Nee	
Hot deployment	Nee	
Kanaalonafhankelijke bediening	Ja	Web API's.
Customer journey	Nee	

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	
Versionering	Nee	
Services & service interfaces	Nee	
Testbaarheid	Nee	
Verouderde frontend code	Nee	
Gemixte logica	Nee	

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Ja	
Ontkoppeling	Ja	
Makkelijk nieuwe componenten toevoegen	Ja	
Simpel en robuust	Ja	
Eén entiteitenmodel	Ja	
Veilig	Ja	
Personalisatie	Nee	

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Ja	
Web API wijzigingen	Ja	
Gedrag van (frontend) schermen	Ja	
Pagina inhoud en presentatie	Ja	

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	
Stubbing is makkelijk	Ja	
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	
De architectuur is gelaagd	Ja	
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	
Inversion of Control	Ja	
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	
Sandbox voor het deployen van content	Ja	

C.8.1.2.4. Architectuur 4

Probleem	Opgelost?	Toelichting
Performance	Nee	
Time To Market	Nee	
Hot deployment	Ja	
Kanaalonafhankelijke bediening	Ja	Web API's.
Customer journey	Nee	

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	
Versionering	Ja	
Services & service interfaces	Nee	
Testbaarheid	Nee	
Verouderde frontend code	Nee	
Gemixte logica	Nee	

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Nee	
Ontkoppeling	Ja	
Makkelijk nieuwe componenten toevoegen	Ja	
Simpel en robuust	Ja	
Eén entiteitenmodel	Ja	
Veilig	Ja	
Personalisatie	Nee	

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Ja	
Web API wijzigingen	Ja	
Gedrag van (frontend) schermen	Ja	
Pagina inhoud en presentatie	Ja	

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	
Stubbing is makkelijk	Ja	
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	
De architectuur is gelaagd	Ja	
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	
Inversion of Control	Ja	
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	
Sandbox voor het deployen van content	Nee	SiteCore.

C.8.1.2.5. Architectuur 5

Probleem	Opgelost?	Toelichting
Performance	Ja	Caching, directe service data aanroepen.
Time To Market	Nee	
Hot deployment	Nee	
Kanaalonafhankelijke bediening	Ja	Web API's.
Customer journey	Nee	

Architectuurprobleem	Opgelost?	Toelichting
Sterke koppeling	Ja	Introduceert mogelijk nieuwe koppeling.
Versionering	Ja	
Services & service interfaces	Ja	Vermijdt service interface aanroepen.
Testbaarheid	Nee	Mogelijk.
Verouderde frontend code	Nee	
Gemixte logica	Nee	

Eisen	Voldaan?	Toelichting
Herbruikbare, generieke code	Ja	
Ontkoppeling	Ja	Introduceert mogelijk nieuwe koppeling.
Makkelijk nieuwe componenten toevoegen	Ja	
Simpel en robuust	Nee	Directe service data aanroep is niet robuust.
Eén entiteitenmodel	Ja	
Veilig	Ja	
Personalisatie	Nee	

Wijzigingen	Ondersteund?	Toelichting
Service wijzigingen	Nee	Gevoelig voor aanpassingen van de service backend.
Web API wijzigingen	Ja	
Gedrag van (frontend) schermen	Ja	
Pagina inhoud en presentatie	Ja	

Aspect	Behouden?	Toelichting
Gebruik van generieke componenten	Ja	
Stubbing is makkelijk	Ja	
De frontend is ontkoppeld en kan onafhankelijk gedeployed worden	Ja	
De architectuur is gelaagd	Ja	Soort van dezelfde lagen-structuur, maar dan lager.
Transactie-laag onafhankelijk van SharePoint: generieke code	Ja	
Inversion of Control	Ja	
Facade (makkelijk services aan roepen, data-verrijking, business logica)	Ja	
Sandbox voor het deployen van content	Nee	SiteCore.