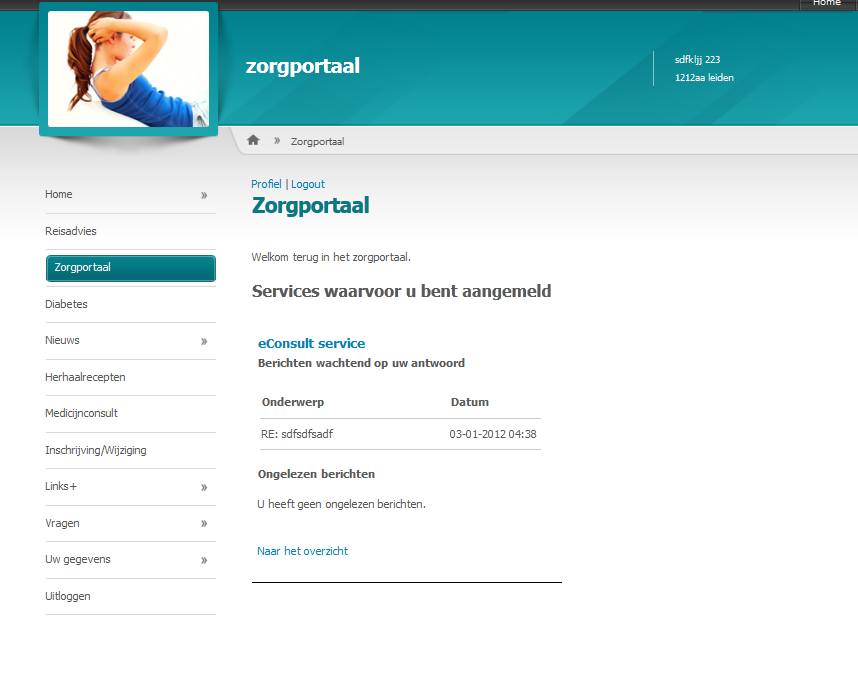
**Afstudeerdossier**



## Samenbrengen van zorgservices in het Zorgportaal

*Bob Brinks  
25 mei 2011  
Amsterdam*

# Referaat

B. Brinks

Titel: Ontwikkeling van een Zorgportaal voor de verschillende services van Pharmeon

Opleiding: HBO, Informatica

In de periode van 21 november 2011 tot 25 maart 2012 heb ik, Bob Brinks, student Informatica aan de Haagse hogeschool afgestudeerd bij Pharmeon. Tijdens de afstudeerperiode heb ik een systeem ontwikkeld dat de verschillende zorgservices van Pharmeon samen brengt in een portaal, het Zorgportaal. Dit verslag verantwoordt het proces en de aanpak van de afstudeerperiode.

Trefwoorden:

* Zorgportaal
* .NET
* C#
* UML
* Integratie

# Voorwoord

Met dit afstudeerproject sluit ik mijn opleiding Informatica aan de Haagse Hogeschool af. Ik wil de volgende mensen graag bedanken voor hun ondersteuning tijdens dit project:

* Erik-Jan de Keizer in zijn rol als bedrijfsmentor.
* Wouter Saarloos voor het regelen van de afstudeerplek.
* Alle leden van SCRUM team A bij Pharmeon voor hun ondersteuning en prettige samenwerking.
* Pharmeon als geheel voor het aanbieden van dit afstudeerproject en de leuke werksfeer.
* De begeleiding vanuit de Haagse Hoge school.

Bob Brinks  
25 mei 2012

Inhoud

[1. Inleiding 1](#_Toc325029645)

[2. Beschrijving organisatie 3](#_Toc325029646)

[3. Beschrijving situatie 4](#_Toc325029647)

[3.1. Probleemstelling 4](#_Toc325029648)

[3.2. Doelstelling 5](#_Toc325029649)

[4. Bespreking oplossingsmethoden en aanpak 6](#_Toc325029650)

[4.1. Scrum 6](#_Toc325029651)

[5. Beschrijving van de werkzaamheden 8](#_Toc325029652)

[5.1. Oriëntatie 8](#_Toc325029653)

[Opstellen plan van aanpak 8](#_Toc325029654)

[Onderzoeken ICT-omgeving Pharmeon 9](#_Toc325029655)

[Opstellen User Stories 11](#_Toc325029656)

[Opstellen architectuur voor het Zorgportaal systeem 12](#_Toc325029657)

[5.2. Sprint 1: Overzicht van beschikbare zorgservices 19](#_Toc325029658)

[Ontwikkelen van middleware 20](#_Toc325029659)

[Ontwikkelen LIPU plugin 25](#_Toc325029660)

[5.3. Sprint 2: Het registreren en inloggen van gebruikers 26](#_Toc325029661)

[Webformulieren 27](#_Toc325029662)

[Userservice 31](#_Toc325029663)

[Sessiemanager 31](#_Toc325029664)

[5.4. Sprint 3: Het gebruik van zorgservices in het Zorgportaal 36](#_Toc325029665)

[Bouwen van eConsult API 37](#_Toc325029666)

[Het uitbreiden van de Zorgportaal LIPU Plugin voor het weergeven van widgets 39](#_Toc325029667)

[Slack time: Ontwikkeling nieuwe template handler 40](#_Toc325029668)

[5.5. Sprint 4: Authenticatie met de beveiligde omgeving (SSIM) 44](#_Toc325029669)

[De CommCommunicatorAPIProxy 45](#_Toc325029670)

[SsimAuthController 46](#_Toc325029671)

[De Configuration tool 47](#_Toc325029672)

[Slack time 49](#_Toc325029673)

[5.6. Sprint 5: Bouwen Webagenda API 50](#_Toc325029674)

[5.7. Sprint 6: Backlog afwerken 51](#_Toc325029675)

[Verhogen code coverage 51](#_Toc325029676)

[6. Evaluatie van het opgeleverde product 55](#_Toc325029677)

[7. Proces evaluatie 57](#_Toc325029678)

[7.1. Planning 57](#_Toc325029679)

[7.2. Ervaring met Test Driven Developement 57](#_Toc325029680)

[7.3. Ervaring met Scrum 57](#_Toc325029681)

[8. Evaluatie beroepstaken 58](#_Toc325029682)

[8.1. 2.1 Opstellen gegevensmodel voor een database 58](#_Toc325029683)

[8.2. 3.1 Ontwerpen softwarearchitectuur 58](#_Toc325029684)

[8.3. 3.2 Ontwerpen systeemdeel 58](#_Toc325029685)

[8.4. 3.3 Bouwen applicatie 59](#_Toc325029686)

# Inleiding

Het doel van dit afstudeerproject is het samenbrengen van de verschillende zorgservices die Pharmeon aanbied. Hiervoor zal er een systeem bedacht en geïmplementeerd worden om de informatie uit de verschillende zorgservices in één overzicht te tonen. Dit overzicht zal weergegeven worden op een webpagina, deze webpagina heet het **Zorgportaal**. Dit nieuwe overzicht maakt het opzoeken van informatie makkelijker voor gebruikers van de zorgservices van Pharmeon.

Dit verslag is primair geschreven voor de twee examinatoren van de Haagse Hogeschool en de gecommitteerde. Daarnaast is het bedoeld als documentatie voor de ontwikkelaars van Pharmeon. Dit verslag is geschreven voor lezers die al enige kennis van software ontwikkeling hebben.

In dit verslag zal ik mijn werkzaamheden en keuzes beschrijven zodat u inzicht krijgt in de manier van werken tijdens dit afstudeerproject.

**Leeswijzer**

In hoofdstuk 2 en 3 wordt het bedrijf en de situatie aan het begin van de opdracht omschreven.

In hoofdstuk 4 wordt de keuze voor een projectbeheer methode omschreven.

In hoofdstuk 5 worden de uitgevoerde werkzaamheden omschreven.

In hoofdstuk 5.1 wordt de oriëntatiefase omschreven.

In hoofdstuk 5.2 wordt sprint 1: “Overzicht van beschikbare zorgservices” omschreven.

In hoofdstuk 5.3 wordt sprint 2: “Het registreren en inloggen van gebruikers” omschreven.

In hoofdstuk 5.4 wordt sprint 3: “Het gebruik van zorgservices in het Zorgportaal” omschreven.

In hoofdstuk 5.5 wordt sprint 4: “Authenticatie met de beveiligde omgeving (SSIM)” omschreven.

In hoofdstuk 5.6 wordt sprint 5: “Bouwen Webagenda API” omschreven.

In hoofdstuk 5.7 wordt sprint 6: “Backlog afwerken” omschreven.

In hoofdstuk 6 wordt het gemaakte product geëvalueerd.

In hoofdstuk 7 wordt het proces geëvalueerd.

In hoofdstuk 8 worden de beroepstaken geëvalueerd.

# 

# Beschrijving organisatie

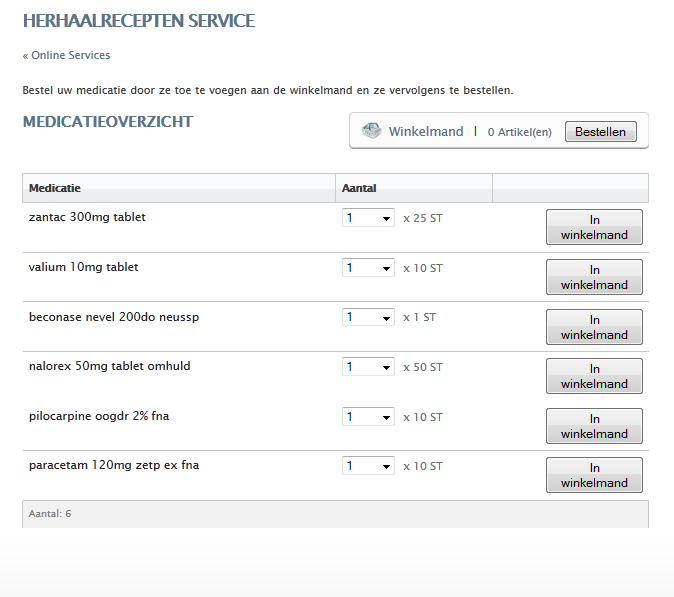
De dienst van Pharmeon bestaat uit het leveren, onderhouden en beheren van websites voor bedrijven in de gezondheidszorg. Deze bedrijven bestaan uit apothekers, artsen tandartsen, dierenartsen en praktijken. Pharmeon beschrijft zichzelf als volgt aan klanten:

“Pharmeon is een full service internet onderneming gespecialiseerd in het verlenen van internetdiensten binnen de gezondheidszorg. De internetoplossingen van Pharmeon ondersteunen medische professionals bij het bieden van online zorg aan de consument.

Door zorgverleners online te brengen voorziet Pharmeon in de groeiende behoefte van consumenten aan continue deskundige en betrouwbare informatie over gezondheid en behandelingsmogelijkheden.”

Via de websites bied Pharmeon enkele zorgservices aan. Het samenbrengen van deze zorgservices is de focus van dit afstudeerproject.  
Deze zorgservices zijn bijvoorbeeld:

* Herhaalreceptservice geeft patiënten de mogelijkheid online hun medicijnen te beheren.
* Pilservice wordt gebruikt om de anticonceptie pil automatisch te bestellen.
* Diabetesservice wordt gebruikt om medische voorraden tegen diabetes te bestellen.
* eConsult service maakt het mogelijk online vragen te stellen aan huisartsen.
* Webagenda maakt het mogelijk om online afspraken te maken met huisartsen of andere medische experts.

De herhaalrecepten service wordt hieronder als voorbeeld weergegeven.   


Figuur 1: De herhaalreceptenservice geeft patiënten een online overzicht van de medicijnen die zij gebruiken en de mogelijkheid om deze medicijnen te bestellen

**Bedrijfsorganisatie**  
Bij Pharmeon werken ongeveer 40 medewerkers verdeeld over de afdelingen Research & Developement en Customer support.

Zelf zit ik bij de afdeling Research & Developement in Scrum team A waar IT systemen worden ontwikkeld en onderhouden. Mijn samenwerking met dit Scrum team wordt in 4.1 verder toegelicht.



Figuur 2: Organigram Pharmeon

# 

# Beschrijving situatie

*Beschrijving van de situatie bij de aanvang van het afstuderen*

## Probleemstelling

In de huidige situatie zijn de zorgservices die Pharmeon aanbiedt losse web applicaties. Iedere zorgservice heeft zijn eigen database en programmatuur. Onderlinge communicatie tussen de verschillende zorgservices en hun databases bestaat niet. De integratie tussen de zorgservices is slecht. Deze slechte integratie betekend voor patiënten dat ze per zorgservice moeten inloggen en ze hebben niet de mogelijkheid een totaaloverzicht van hun medische informatie te zien.



Figuur 3: De zorgservices als losse systemen zonder onderlinge interactie

De huidige situatie is ontstaan doordat de zorgservices die Pharmeon levert over een tijdspanne van 10 jaar als losse systemen ontwikkeld zijn. De zorgservices zijn ontwikkeld in verschillende talen; ASP (classic) en .NET. De verschillende zorgservices zijn niet ontworpen en geïmplementeerd volgens best practices en design patterns, zo ontbreekt de scheiding tussen model, view en controller (MVC).

Pharmeon wil graag een nieuwe zorgservice ontwikkelen, het Zorgportaal. De bedoeling van dit Zorgportaal is het aanbieden van een totaaloverzicht van medische informatie van een patiënt. Dit houdt in dat het Zorgportaal data uit alle zorgservices moet ophalen.   
  
De huidige situatie zal aangepast moeten worden om het Zorgportaal te kunnen realiseren omdat de huidige situatie zorgt voor de volgende twee problemen:

* In de huidige situatie staat data van één patiënt verspreid over meerdere databases, hierdoor is het niet mogelijk om een totaaloverzicht van een patiënt te krijgen.
* Door het ontbreken van samenhang tussen de datamodellen van de verschillende zorgservices weet het systeem niet dat data van de verschillende zorgservices mogelijk data van dezelfde patiënt zijn.

## Doelstelling

De doelstelling voor de opdracht is het ontwikkelen van het Zorgportaal waarmee een totaaloverzicht van een patiënt weergegeven en aangepast kan worden.

Bij het ontwikkelen van het Zorgportaal ligt de nadruk op de backend. Deze backend heeft als doel het verbeteren van de integratie tussen de verschillende zorgservices.

**Randvoorwaarde**  
De huidige zorgservices moeten als standalone kunnen blijven draaien, deze zijn namelijk al in productie op veel websites van Pharmeon. De bestaande zorgservices en hun databases moeten dus niet aangepast worden voor de ontwikkeling van het Zorgportaal.

# 

# Bespreking oplossingsmethoden en aanpak

*Bespreking van mogelijke oplossingsmethoden en verdediging van de gekozen aanpak*

## Scrum

In het project wordt de ontwikkelmethode “Scrum” gebruikt. Ik heb voor Scrum gekozen omdat het de ontwikkelmethode is waar Pharmeon gebruik van maakt. Tijdens het project werk ik samen met een Scrum team. Dit Scrum team werkt niet aan mijn afstudeerproject, maar aan andere projecten. De rol van het team in mijn project is het beoordelen van door mij gemaakte producten en keuzes, en het bieden van ondersteuning.  
Als ik voor sommige taken niet de benodigde vaardigheden of kennis heb kan ik voor deze taken ondersteuning vragen aan het Scrum team. Voorbeelden van deze taken zijn: het grafisch vormgeven van de applicatie en het uitrollen van de applicatie.

In het figuur hieronder staat het Scrum proces weergegeven en worden enkele belangrijke termen binnen Scrum genoemd. Vervolgens wordt toegelicht hoe Scrum binnen mijn project geïnterpreteerd wordt.



Figuur 1: Het Scrum proces (bron: http://nl.wikipedia.org/wiki/Scrum\_(softwareontwikkelmethode))

**Product backlog**Binnen Scrum worden de eisen (requirements) in de product backlog bijgehouden, dit is een geprioriteerde lijst van eisen. Ik zal aan het begin van mijn project de (belangrijkste) eisen van het Zorgportaal verzamelen en in mijn eigen product backlog plaatsen. Zoals gebruikelijk binnen Scrum zullen de eisen worden beschreven als User Stories. Wat User Stories zijn wordt beschreven in hoofdstuk 5 onder “Opstellen van de User Stories”.

**Sprints**Scrum is een iteratieve ontwikkelmethode, de iteraties worden binnen Scrum ‘sprints’ genoemd. Binnen mijn project hebben sprints de volgende eigenschappen:

* Iedere sprint heeft een vaste lengte, voor mijn project zullen alle sprints 2 weken duren.
* Iedere sprint heeft een lijst van User Stories die binnen die 2 weken gerealiseerd zullen worden. Deze lijst wordt de Sprint backlog genoemd.  
  De User Stories in de sprint backlog komen uit de product backlog en worden geselecteerd op prioriteit aan het begin van de sprint.
* De Sprint backlog wordt op een manier gerealiseerd waardoor er aan het einde van de sprint een werkend product is. Dit product kan in een demo gepresenteerd worden aan stakeholders.
* Als bepaalde User Stories niet binnen de twee weken van de sprint gerealiseerd kunnen worden, worden ze terug in de product backlog geplaatst. De sprint wordt niet verlengd om de User Stories alsnog af te ronden. De User Stories kunnen in latere sprints weer opgepakt worden als ze nog steeds hoge prioriteit hebben.

De sprints binnen mijn project bestaan uit 10 werkdagen. Elk van deze dagen begint met een daily Scrum meeting. In deze korte vergadering met het Scrum team vertellen de leden wat ze de dag ervoor gedaan hebben, wat ze op de dag gaan doen en of er problemen zijn.  
**Scrum rollen**Binnen mijn project gebruik ik de volgende Scrum rollen.

* **Product owner**Een persoon die alle stakeholders representeert. Stakeholders zullen niet direct met mij praten maar met de product owner. De product owner stelt ook de lijst van eisen van het systeem op en prioriteert deze, deze lijst van eisen wordt de **product backlog** genoemd. Tom Meintser is product owner voor het Zorgportaal project.
* **Scrum Master**De persoon die het Scrum proces waarborgt. Voor het Zorgportaal project is de afstudeerder de Scrum Master.
* **Het team**Een groep van personen die de software uiteindelijk echt implementeert, ontwerpt, test, onderhoudt etc. Het team voor het Zorgportaal project bestaat uit de afstudeerder en voor bepaalde sprints ook uit andere medewerkers van Pharmeon.

# Beschrijving van de werkzaamheden

In dit hoofdstuk worden de werkzaamheden van het project beschreven, de werkzaamheden zijn per sprint beschreven.

## 

## Oriëntatie

De eerste drie weken van mijn afstudeerproject heb ik een oriëntatie op het project gedaan en dus nog geen sprints gedaan. Ik noem deze oriëntatie periode geen sprint omdat het geen bruikbaar product (software) oplevert en volgens Scrum is dit bij een sprint wel het geval. Ik heb deze weken gebruikt om de huidige situatie binnen Pharmeon duidelijk te krijgen en de architectuur van het Zorgportaal en de bijbehorende middleware op te zetten. Tijdens deze oriëntatie periode heb ik de volgende activiteiten uitgevoerd:

* Opstellen plan van aanpak
* Onderzoek ICT-omgeving Pharmeon
* Opstellen User Stories
* Opstellen architectuur van het Zorgportaal

### Opstellen plan van aanpak

Het plan van aanpak heb ik voor een groot deel opgesteld in de aanloop naar het project (afstudeerplan). In de eerste week van mijn afstudeerproject heb ik het aangevuld met de uitgangsituatie, risico’s en te hanteren methoden en technieken, deze zijn terug te vinden in de bijlage: “Plan van aanpak”.  
Ook heb ik een globale planning voor de sprints geschreven. Ik heb er voor gekozen om mijn sprints synchroon met die van de rest van de teams binnen Pharmeon te laten lopen. Dit houdt dus in dat ik sprints van 2 weken heb ingeplant vanaf projectweek 4. De eerste sprint duurt hierbij een week, deze sprint is als oriëntatie op Scrum bedoeld en zorgt voor het synchroon lopen met Pharmeon sprints na de Oriëntatie fase. Ik ga zeven sprints uitvoeren en ik verwacht binnen deze sprints het zorgportaal systeem op te kunnen leveren met integratie van in ieder geval twee van de zorgservices.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Project week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Iteratie | Kalenderweek | 48 | 49 | 50 | 51 | 52 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Oriëntatie |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sprint 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tabel 1: Globale planning

Welke User Stories in welke sprint behandeld worden staat in deze globale planning nog niet vast, dit zal per sprint bekeken worden.

### Onderzoeken ICT-omgeving Pharmeon

Omdat ik binnen dit afstudeerproject de bestaande ICT-omgeving van Pharmeon uitbreid is het belangrijk om te weten hoe deze omgeving er uit ziet. Daarom heb ik na het opstellen van mijn Plan van aanpak de ICT-omgeving binnen Pharmeon onderzocht.  
In deze paragraaf wordt het component LIPU omschreven. Dit component wordt omschreven omdat het Zorgportaal er veel gebruik van zal maken.

#### LIPU

De websites van Pharmeon zijn geprogrammeerd in de sterk verouderde taal ASP (Classic). Om bij het door ontwikkelen van de websites gebruik te kunnen maken van de voordelen van de moderne programmeertaal **C#.net** heeft Pharmeon LIPU ontwikkeld.

LIPU maakt het mogelijk HTTP requests die binnen komen in de ASP omgeving af te handelen met .NET code. LIPU is door Pharmeon ontwikkeld omdat de websites van Pharmeon in ASP draaien terwijl Pharmeon wil overstappen naar .NET. Omdat de websites en alle bijbehorende componenten een erg groot systeem zijn, is het niet mogelijk om dat hele systeem binnen overzienbare tijd om te zetten naar .NET. Daarom is LIPU bedacht als tussenstap tussen ASP en .NET.

LIPU draait als een web service die vanuit ASP aangeroepen kan worden. Als antwoord op aanroepen geeft de LIPU web service HTML (of in het geval van AJAX requests XML) terug. Deze HTML (of XML) wordt gegenereerd door een LIPU plug-in. Dit is een in C# geprogrammeerde plug-in die de LIPU web service uitbreidt met functionaliteit, voor het Zorgportaal is dit het Zorgportaal LIPU plug-in.



Figuur 2: De LIPU webservice wordt aangeroepen vanuit ASP om een HTTP request van de gebruiker af te handellen.

Bij aanvragen naar de LIPU Web service wordt een LIPU\_Handler object meegegeven, dit object bevat data uit de HTTP request die bij de ASP omgeving binnen is gekomen, bijvoorbeeld POST en GET data, de URL en de sessie (session) van de gebruiker die de request heeft gedaan.  
Deze LIPU\_Handler wordt door de LIPU plug-ins gebruikt om de juiste HTML te genereren.

### Opstellen User Stories

Als derde activiteit in de oriëntatie fase heb ik de User Stories (eisen) voor het Zorgportaal opgesteld.  
Na een gesprek met de Product Owner (Tom Meintser) en het hoofd van de technische afdeling (Jody de Vries) over hun verwachtingen van het Zorgportaal heb ik User Stories opgesteld. Bij het opstellen heb ik gebruikt gemaakt van het artikel: “User Stories, een verhaal apart” door **Dennis Doomen.**

De User Stories heb ik daarna met de Product Owner doorgenomen en verbeterd. Hierna heeft de Product Owner de User Stories goed gekeurd en prioriteiten toegewezen.   
Met deze stappen ben ik tot de volgende User Stories en prioriteiten gekomen:

1. Als patiënt wil ik een overzicht van de zorgservices die een website aanbied kunnen zien.
2. Als een ongeregistreerde patiënt wil ik bij het inschrijven bij een zorgverlener ( voor offline administratie) ook de mogelijkheid hebben om meteen een online account aan te maken. Zodat ik niet twee keer dezelfde informatie hoe in te vullen.
3. Als patiënt wil ik maar eenmaal registreren bij een website zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef te registreren.
4. Als patiënt wil ik per bezoek aan een website maar eenmaal inloggen zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef in te loggen.
5. Als patiënt wil ik bij het navigeren naar een zorgservice kunnen inloggen zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef in te loggen.
6. Als ingelogde patiënt wil ik mij via het Zorgportaal kunnen aanmelden bij zorgservices zodat ik niet op de website hoef te zoeken naar de verschillende zorgservices die de website aanbied.
7. Als ingelogde patiënt wil ik een overzicht kunnen zien van de zorgservices waar ik gebruik van maak (voor aangemeld ben).
8. Als ingelogde patiënt wil ik een overzicht kunnen zien van mijn status in de zorgserivce waar ik gebruik van maak, zodat ik in een overzicht kan zien of ik actie moet ondernemen in een of meerdere van de zorgservices.
9. Als ingelogde patiënt wil ik per bezoek maar eenmalige een sessiecode invoeren zodat ik bij het switchen van zorgservices minder tijd kwijt ben met het invoeren van sessiecodes.
10. Als ingelogde patiënt wil ik dat gegevens die ik eerder in het systeem heb in gevoerd, niet opnieuw ingevoerd hoeven te worden in web formulieren, zodat ik geen tijd verspil met het opnieuw invoeren van al bekende gegevens.
11. Als patiënt wil ik vanuit elke pagina op de website altijd terug kunnen naar het Zorgportaal zodat ik nooit op een pagina ‘vast’ kom te zitten.
12. Als een beheerder van een website wil ik kunnen aanpassen welke zorgservices mijn website aanbied binnen het Zorgportaal.
13. Als Pharmeon wil ik statistieken kunnen inzien over het gebruik van de zorgservices en het Zorgportaal.

De bovenstaande lijst bevat de verkorte versies van de User Stories, de volledige versie zijn uitgeschreven volgens het bovengenoemde artikel van Dennis Doomen. Een volledige lijst van uitgeschreven User Stories is terug te vinden in de bijlage: “User Stories”.

|  |  |
| --- | --- |
| Als een | Patiënt |
| Wil ik | Een overzicht van de zorgservices die een website aanbiedt kunnen zien |
| Zodat | Ik weet wat de website mij biedt op het gebied van zorgservices |
| Storyotype | New Functionality |
| Story points | 13 |

*Acceptation criteria:*

  Alleen zorgservices die geactiveerd zijn

  Alleen de volgende zorgservices: herhaalrecept, Pil, Diabetes, eConsult, Digitaal medisch dossier en Webagenda

### Opstellen architectuur voor het Zorgportaal systeem

In de oriëntatie periode heb ik een oplossingsrichting gekozen om de slechte integratie tussen de verschillende zorgservices die Pharmeon aanbiedt te verbeteren. Deze oplossing zal als rode draad door de verschillende sprints van mijn project lopen.   
Binnen Scrum is het gebruikelijk systeemonderdelen pas te ontwerpen wanneer ze geïmplementeerd worden binnen een sprint. Omdat het Zorgportaal op verschillende manieren tussen de bestaande systemen van Pharmeon geïmplementeerd kan worden heb ik er voor gekozen om voor de eerste sprint alvast een architectuur/oplossingsrichting voor het project op te stellen.

Keuze oplossingsrichting  
Ik heb naar enkele oplossingen gekeken die de data en functionaliteit van de verschillende zorgservices beschikbaar maken voor het Zorgportaal, namelijk:

* **De data direct uit de database halen**
* **Point-to-point connecties tussen het Zorgportaal en elk van de zorgservices**
* **Middleware als mediator tussen de zorgservices en het Zorgportaal**

**De data direct uit de database halen**  
Deze oplossing is de meest directe manier om data uit de verschillende zorgservices te halen. Het voordeel van data direct uit database halen is dat de ontwikkeling niet ingewikkeld is. Als er ergens in het Zorgportaal data nodig is kan de ontwikkelaar die data altijd direct uit de database halen met behulp van SQL.  
Naast dit voordeel van makkelijke ontwikkeling zijn er ook enkele nadelen aan data direct uit de database halen.

Figuur 4: Data direct uit de databases halen

* De verschillende zorgservices implementeren business logica die de data uit de database bewerkt voordat deze aan de gebruiker getoond wordt (of op een andere manier gebruikt wordt). Door de data direct uit de database te halen wordt deze business logica omzeild en is het mogelijk dat de opgehaalde data niet juist is.  
  Ook de business logica voor het opslaan of bewerken van data wordt omzeild.
* Elke functionaliteit die wordt geboden door een zorgservice die ook in het Zorgportaal beschikbaar moet zijn, moet opnieuw geïmplementeerd worden binnen het Zorgportaal. Dit kost tijd en kan leiden tot ‘gekopieerde’ code en is daardoor slecht voor de onderhoudbaarheid van het systeem als geheel.
* Veranderingen in de databasestructuur moeten in de business logica van de zorgservices zelf en in het Zorgportaal aangepast worden (coupling).

**Point-to-point connecties tussen het Zorgportaal en elk van de zorgservices**Deze point-to-point connecties houden in dat er voor iedere zorgservice waar het Zorgportaal gebruik van wilt maken een connectie geïmplementeerd wordt. Via deze connectie kan het Zorgportaal de business logica van een zorgservice aanroepen.



Figuur 5 Voorbeeld van een point-to-point connectie

De problemen die de oplossing “De data direct uit de database halen” met zich meebrengt zijn dus niet van toepassing op de Point-to-point oplossing. Maar deze Point-to-point oplossing brengt ook enkele problemen met zich mee en hoewel deze problemen buiten de scoop van de ontwikkeling van het Zorgportaal liggen heb ik ze toch meegenomen in de keuze voor een oplosrichting.  
Problemen buiten de scope:

* In het geval dat de zorgservices ook gebruik willen maken van data of functionaliteit uit een andere zorgservice moet hiervoor ook een point-to-point connectie geïmplementeerd worden. Als dit vaak voor komt zullen er in het geval van Pharmeon snel een groot aantal connecties geïmplementeerd moeten worden.   
  Deze vele connecties zorgen voor hoge coupling en dependencies, dit zorgt er voor dat verandering in het systeem grote impact hebben op de rest van het systeem.

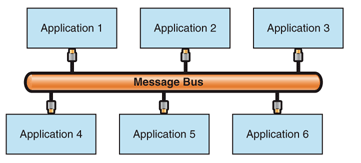


Figuur 6 Voorbeeld van een groot aantal connecties

* Om iedere zorgservice met iedere andere zorgservice te integreren moeten *n(n -1)/2 (n = aantal zorgservices, bron: http://en.wikipedia.org/wiki/Enterprise\_application\_integration)* connecties geïmplementeerd worden. Met 6 zorgservices komt dit op 15 connecties. Als er dan nog een nieuwe zorgservice wordt toegevoegd komt het op 21 connecties. Het ontwikkelen van al deze connecties kan een tijdrovend proces zijn.

Binnen de scope van het Zorgportaal project worden er alleen connecties tussen het Zorgportaal en de verschillende zorgservices gemaakt. De problemen met coupling en dependencies zijn in dat geval nog te overzien.

**Middleware als mediator tussen de zorgservices en het Zorgportaal**Met een middleware-systeem als mediator tussen de zorgservices en het Zorgportaal wordt per zorgservice één connectie geïmplementeerd naar het middleware-systeem. Het middleware-systeem biedt dan een API waar alle andere applicaties (incl. Het Zorgportaal en de verschillende zorgservices) gebruik van kunnen maken. Deze API biedt alle data en functionaliteit die via zorgservice-connecties aangeboden wordt. De problemen die Point-to-point en data direct uit de database halen met zich meebrengen zijn niet van toepassing op een middleware oplossing.  
Wel neemt het ontwerpen en implementeren van een middleware oplossing voor een laag aantal zorgservices meer tijd in beslag dan de andere 2 oplosrichtingen. En door de hogere complexiteit kan een middleware-systeem tot zwaardere server belasting leiden.



Figuur 3: Een messagebus (middleware) als mediator tussen 6 applicaties heeft maar 6 connecties (bron: http://msdn.microsoft.com/en-us/library/ff647328.aspx)

**Uiteindelijke keuze**  
De oplossing om de integratie tussen de verschillende zorgservices te realiseren die ik heb gekozen is een middleware als mediator tussen de zorgservices en het Zorgportaal. De point-to-point oplossing was voor dit project ook geschikt. De keuze voor het middleware-systeem boven point-to-point komt vooral voort uit de vele connecties tussen zorgservices die point-to-point buiten de scope van dit project veroorzaakt.

Het middleware-systeem  
Om de data en functionaliteit van de verschillende zorgservices beschikbaar te maken voor het Zorgportaal heb ik ervoor gekozen een middleware-systeem te ontwikkelen. Omdat er binnen Pharmeon geen wens is om een bestaand middleware-systeem aan te schaffen heb ik geen onderzoek gedaan naar standaard pakketten voor middleware.  
De middleware heeft als doel de data en functionaliteit van de verschillende zorgservices beschikbaar maken voor de andere zorgservices waaronder het Zorgportaal. De middleware lost hiermee het probleem van de slechte integratie tussen de zorgservices op.  
**Communicatie met het middleware-systeem**  
Het onderstaande diagram laat zien hoe het Zorgportaal via het middleware-systeem kan communiceren met de verschillende zorgservices.



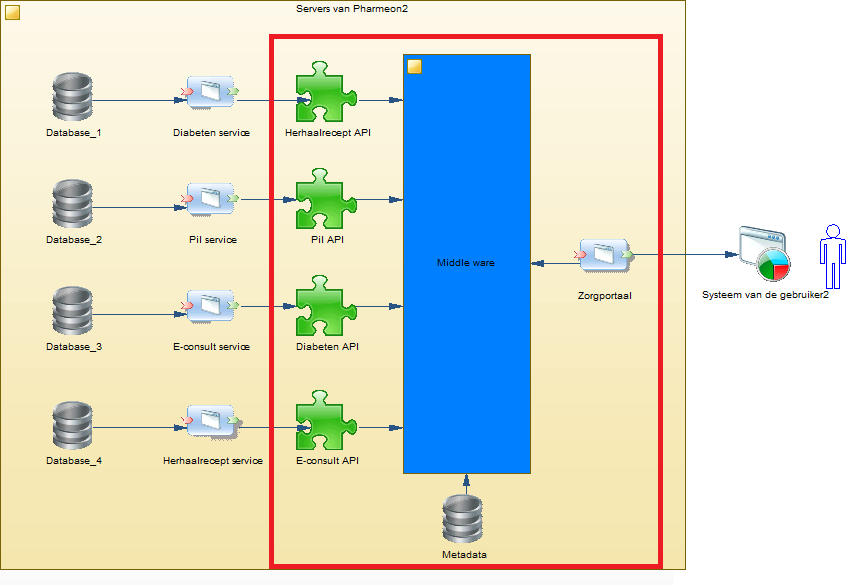
Figuur 4 Communicatie tussen de middleware en de verschillende zorgservices

De zorgservices die in dit diagram in het Zorgservices blok (de zorgservices zijn als één blok weergegeven om het diagram leesbaar te houden) weergegeven zijn, kunnen zelf ook gebruik maken van de Zorg API en zo data uit andere zorgservices ophalen. Hoewel deze functionaliteit bestaat valt het buiten de scope van de ontwikkeling van het Zorgportaal en dus dit project.

De diagrammen op de volgende pagina laten zien hoe de structuur uit het bovenstaande figuur 4 de oude situatie binnen Pharmeon verandert.



Figuur 5 De oude situatie



Figuur 6: In het rode blok staan de systeemdelen die ontwikkeld moeten worden.

**Eisen van het middleware-systeem**Na ervoor gekozen te hebben een middleware-systeem te ontwikkelen heb ik onderzocht hoe een dergelijk middleware-systeem zou moeten werken. De belangrijkste informatie heb ik gevonden in Microsofts omschrijving van een Message Bus (middleware) op MSDN (<http://msdn.microsoft.com/en-us/library/ff647328.aspx>). Dit artikel beschrijft de theorie achter de Message Bus dat integratie tussen meerdere applicaties kan realiseren. De oplossing uit het artikel, de Message Bus, is niet de oplossing voor het integratie probleem van de zorgservices. Uit het artikel heb ik wel de volgende belangrijke punten kunnen halen voor de ontwikkeling van een Middleware-systeem:

* De verschillende applicaties (in het geval van Pharmeon de zorgservices) moeten zichzelf bij het middleware-systeem kunnen registreren. Zodat hun informatie/ functionaliteit beschikbaar gesteld kan worden door het middleware-systeem.
* Het middleware-systeem moet een infrastructuur bieden om berichten van de ene applicatie naar de andere te kunnen versturen.
* De middleware moet informatie in een formaat versturen dat door alle applicatie in ieder geval qua syntax begrijpelijk is.

Op basis van deze belangrijke punten heb ik zelf een lijst van eisen voor het middleware-systeem opgesteld:

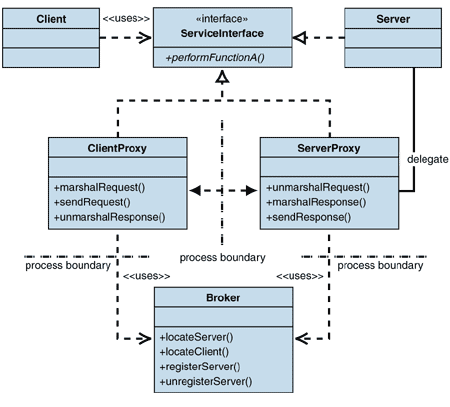
* De middleware houdt een lijst bij van beschikbare zorgservices
* De middleware houdt bij hoe de verschillende zorgservices aangeroepen kunnen worden (infrastructuur).
  + Draait de zorgservice in dezelfde runtime of als web service of biedt de zorgservices zijn data op een andere manier aan?
  + Welk communicatieprotocol wordt gebruikt voor het ophalen van de data?
* De middleware houdt bij welke gebruiker is ingelogd (authenticatie en autorisatie).
  + Welke data mag de ingelogde gebruiker inzien/aanpassen?
  + Welke functies mag de ingelogde gebruiker gebruiken?
* De middleware houdt bij welke data en functionaliteiten beschikbaar zijn bij welke service.

**Architectuur van het middleware systeem**  
Op basis van deze eisen heb ik het onderstaande componentendiagram opgesteld, dit diagram beschrijft de componenten en onderlinge relaties die ik voor de middleware moet implementeren.



Figuur 8: Interne werking middleware

De manier waarop de middleware de lijst van beschikbare zorgservices bijhoudt is gebaseerd op het broker design-pattern. De Zorgservices (Server in het broker design pattern) registreren zichzelf bij de Service pool (Broker in het broker design pattern) met een service proxy (ServerProxy in het broker design pattern). Er worden binnen het Zorgportaal geen client proxies bijgehouden zoals dat wel kan in het Broker pattern, de Zorg API (ServiceInterface in het broker design pattern) wordt namelijk altijd lokaal aangesproken.



Figuur 7: Het broker design pattern

Naast de Zorgservice, Service Proxy en Servicepool componenten staat er in het diagram ook nog een Configuration Tool. Deze tool stelt in welke zorgservices zich bij het middleware-systeem registreren. Bij die registratie worden de volgende taken uitgevoerd:

* Bij het registreren vraagt de middleware welke data en functionaliteit een zorgservice biedt, deze informatie wordt opgeslagen in de Common datamodel manager.
* In de Service pool wordt een proxy van de geregistreerde zorgservice bijgehouden.  
  Met deze proxy kan de zorgservice worden aangeroepen waar deze ook draait.

De zorgservices kunnen aan de middleware vragen om functionaliteit en data die door de geregistreerde zorgservices geboden wordt. Als de gevraagde functionaliteit of data niet beschikbaar is geeft de middleware een exception terug.

* De zorgservice vraagt data of functionaliteit aan bij de Zorg API, deze request wordt afgehandeld door de request processor.
* De request processor vraagt aan het Common datamodel management in welke zorgservices (of waar) de data of functionaliteit wordt aangeboden.
* In het Common datamodel is per functionaliteit of data ook bekend welke ‘permissions’ (toestemmingen) er voor nodig zijn.
* De Sessionmanger vertelt de Request processor of een gebruiker ingelogd heeft en welke permissions hij/zij heeft.

## 

## Sprint 1: Overzicht van beschikbare zorgservices

Sprint 1 heb ik als een test sprint opgezet, het belangrijkste doel van deze sprint was de ontwikkel methode Scrum in de praktijk ervaren, leren hoe de bestaande systemen van Pharmeon werken, hoe de rest van het Scrum team ontwikkelt en welke standaarden Pharmeon hanteert.

In sprint 1 is één user story opgenomen:

1. Als patiënt wil ik een overzicht van de zorgservices die een website aanbiedt kunnen zien.

Deze User Story biedt niet direct functionaliteit voor de klant, maar zoals genoemd was dit niet de bedoeling van deze sprint. De definition of done[[1]](#footnote-1) voor deze sprint is als volgt: “Er kan een overzicht getoond worden van de zorgservices die een website aanbiedt”.

Om de definition of done te bereiken heb ik in deze sprint de volgende taken uitgevoerd, uiteraard binnen de scope van deze sprint:

1. Ontwikkelen van een deel van de middleware
2. Ontwikkelen LIPU plugin

Hoe ik deze taken heb uitgevoerd beschrijf ik in dit hoofdstuk. Daarnaast beschrijf ik ook hoe ik gebruik het gemaakt van Test Driven Developement (TDD).

### Ontwikkelen van middleware

Binnen deze sprint heb ik de eerste delen van het middleware-systeem geïmplementeerd. Deze delen zijn nodig voor het weergeven en bijhouden van een lijst van zorgservices. De scope van deze implementatie is bepaald door User Story 1: “**Overzicht van informatie uit verschillende zorgservices in het Zorgportaal”**. Deze versie van de middleware moet dus een overzicht kunnen aanbieden van de zorgservices die een website aanbiedt. Het ontwerp is gebaseerd op de architectuur die ik in de oriëntatiefase heb opgesteld. Het ontwerp valt binnen de groene aangegeven componenten in de middleware (Figuur 8). Voor de overige componenten van de middleware heb ik binnen deze sprint dummy data en/of dummy implementaties gebruikt, zoals gebruikelijk binnen SCRUM.



Figuur 8: Te realiseren componenten (groen)

**Gemaakte keuzes voor de zorgservice pool**  
De lijst van zorgservices in de service pool moet tussen verschillende page requests blijven bestaan. Het Zorgportaal systeem draait als LIPU plug-in en binnen LIPU plug-ins zijn er geen persistente objecten (objecten die blijven bestaan tijdens meerdere page requests). Om toch gebruik te kunnen maken van persistente objecten heb ik twee oplossingen bekeken, de service proxy objecten geserialiseerd opslaan in een bestand of een aparte applicatie bouwen volgens het Object pool design pattern.

De Object pool applicatie zou continu draaien en de service pool zou de objecten die hij nodig heeft (Service proxies) kunnen aanvragen bij deze Object pool applicatie. Het ontwikkelen van deze applicatie was binnen de tijdspanne van deze sprint niet haalbaar. Daarom heb ik ervoor gekozen de Service proxies in een XML bestand op te slaan.

De Service proxies worden geserialiseerd[[2]](#footnote-2) opgeslagen in dit bestand en als er een aanvraag binnenkomt wordt de Service proxy gedeserialiseerd[[3]](#footnote-3) tot een object dat gebruikt kan worden om de aanvraag af te handelen.  
Het gebruik van XML bestanden heeft wel enkele nadelen:

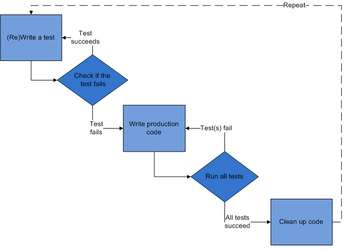
* Het zorgt voor extra I/O requests bij elke page load. Andere oplossingen die geen I/O request per page load doen zullen betere performance hebben.
* Als er gevoelige data in de Service proxy zou staan is het mogelijk deze uit te lezen uit het geserialiseerde object in het XML bestand. Dit is een mogelijk security probleem.

Als de performance of security problemen te groot worden in toekomstige sprints zal er tijd gemaakt moeten worden om de Object pool applicatie alsnog te ontwikkelen.

**Gemaakte keuzes voor service proxies**Binnen de scope van deze sprint is het niet nodig om echte data of function calls naar de verschillende zorgservices te doen. Om deze reden heb ik alleen een ‘direct call’ proxy geïmplementeerd, deze implementatie roept zorgservices die in dezelfde runtime draaien direct aan. De IServiceAPIProxy interface kan in toekomstige sprints op andere manieren geïmplementeerd worden om data van buiten de runtime op te halen. Alle IServiceAPI implementaties binnen deze sprint waren dummy implementaties die test XML data terug gaven. Het onderstaande klasse diagram beschrijft de uiteindelijk geïmplementeerde classes van het middleware-systeem binnen Sprint 1.

**Implementeren met behulp van Test Driven Developement (TDD)**

Voor de ontwikkeling van de middleware heb ik gebruik gemaakt van TDD. TDD is een software ontwikkelmethode waarbij eerst de (unit)tests worden geschreven en daarna pas de code.



Figuur 9: Een flowchart van de ontwikkelcyclus van TDD (bron: http://en.wikipedia.org/wiki/Test-driven\_development)

Om duidelijk te maken hoe ik gebruik heb gemaakt van TDD bij het ontwikkelen van de middleware zal ik een ontwikkelcyclus van TDD toelichten. Binnen deze cyclus heb ik de servicepool de mogelijkheid gegeven om nieuwe zorgservices toe te voegen.  
Het toevoegen van een nieuwe zorgservice houdt in dat er een nieuw XML element met de beschrijving van een zorgservice wordt toegevoegd aan het XML bestand van de servicepool.

Als eerste stap in de TDD ontwikkel cyclus heb ik de volgende test geschreven.

[TestMethod()]

public void registerServiceTest()

{

//Setup test file manager

TestFileManager testFilemanager = new TestFileManager();

FileManagerFactory.setTestingFileManagaer(testFilemanager);

testFilemanager.xmlToReturn = "<services />";

//Setup test variables

string fileName = string.Empty;

CServicePoolXML target = new CServicePoolXML(fileName);

string serviceName = "test";

IServiceAPIProxy serviceProxy = new SerializableTestProxy();

//Run the method

target.registerService(serviceName, serviceProxy);

string actual = testFilemanager.xmlWriten;

//Set expected XML

XmlHandler expected = new XmlHandler();

expected.LoadXml(@"De verwachte xml is in dit voorbeeld weggelaten”);

//Asserts

Assert.AreEqual(expected.GetRawXml(), actual);

}

**De test (Write a test)**  
Deze test gaat er vanuit dat de service pool de zorgservice (SerializableTestProxy) omzet naar XML en wegschrijft naar het servicepool XML bestand. Het omzetten naar XML wordt gedaan met een.NET component. Dat component is al ontwikkeld en getest en wordt door deze Unittest dus niet mee getest.

Om te controleren wat de CServicePoolXML wegschrijft naar het bestand en of weggeschreven gegevens gelijk zijn aan de verwachte waarde, maak ik gebruik van een FileManagerFactory. De TestFileManager wordt met behulp van een Factory (design pattern) in de CServicePoolXML gebruikt. De TestFileManager leest en schrijft geen bestanden op de harde schijf, in tegenstelling tot de normale FileManager implementatie. De TestFileManager kan geconfigureerd worden om bepaalde XML terug te geven (testFilemanager.xmlToReturn = "<services />";) en slaat weggeschreven XML op in het geheugen (testFilemanager.xmlWriten).

Door gebruikt te maken van deze TestFileManager is de unittest niet meer afhankelijk van bestanden op de harde schijf, hierdoor kan de test overal gedraaid worden en heeft de test geen impact op bestanden die mogelijk door andere tests of zelfs de ontwikkel/productie omgevingen gebruikt worden. Dit weghalen van afhankelijkheid (depencies) van code op externe systemen (bijvoorbeeld de harde schijf) is een eis voor goede Unittests.

**De code (Write production code)**

Na het schrijven van de test heb ik, volgens TDD, de volgende code geschreven om de test te laten slagen.

public void registerService(string serviceName, IServiceAPIProxy serviceProxy)

{

//Het nieuwe service element aanmaken

XmlElement serviceElement = file.XmlDoc.CreateElement("service");

serviceElement.setAttribute("name", serviceName);

serviceElement.InnerText = ObjectSerializer.Serialize(serviceProxy);

//Het nieuwe service element toevoegen aan het XML bestand

file.AddElement(serviceElement, file.XmlDoc.SelectSingleNode("services"));

filemanager.saveXML(fileName, file);

}

De bovenstaande code maakt een XML element aan voor de te registreren zorgservice. Dit XML element wordt daarna toegevoegd aan de bestaande XML in het servicepool bestand. Hiermee is voldaan aan eerste de Unittest.

Nadat aan de eerste Unittest voldaan is, heb ik andere Unittests geschreven en de code aangevuld om ook aan die Unittests te voldoen. Enkele voorbeelden van scenario’s die de latere Unittests testen zijn:

* Er wordt een exception gethrowt als er geen bestand gevonden kan worden. Voor het throwen van deze exception heb ik de TestFileManager uitgebreid met de mogelijkheid een exception te throwen.
* Als er al een service met de zelfde naam bestaat, wordt deze overschreven.

Nadat alle unittests voor de registerService methode geschreven waren heb ik waar nodig de productie code nog gerefactord (**Clean up code**).

**TDD door de rest van het project**

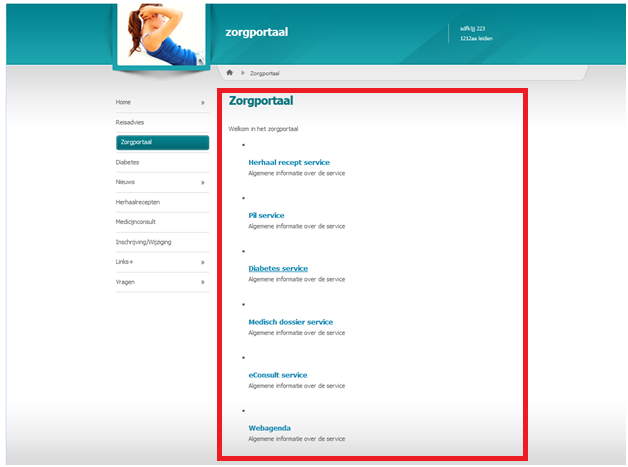
In de rest van mijn afstudeerproject heb ik geprobeerd zo veel mogelijk TDD te gebruiken. Dit is voor de middleware goed gelukt. Andere onderdelen van het Zorgportaal systeem (de LIPU plug-in en de verschillende zorgservice API’s) waren lastiger te testen omdat ze gebruik maken van legacy componenten waar veel ontestbare code door ontstaat. Een voorbeeld hiervan is de GeneralDB klasse, een component ontwikkelt binnen Pharmeon waarmee connecties naar verschillende databases gemaakt worden. Dit component moet ik gebruiken om databases aan te kunnen spreken maar er is geen Interface voor die geïmplementeerd kan worden voor test doeleinde zoals ik dat met de FileManager klasse wel kon doen.

Voor methodes die afhankelijk zijn van legacy componenten is het niet mogelijk Unit tests, of een andere vorm van geautomatiseerde en herhaalbare tests, te schrijven. Hierdoor was TDD voor het ontwikkelen van deze methodes niet bruikbaar. Deze methodes heb ik daarom ontwikkeld zoals ik voor TDD ook ontwikkelde, namelijk in een ontwikkelomgeving de code draaien en met error guessing tests tijdens het programmeren testen. Daarna is deze code nog getest door het test team van Pharmeon.

### Ontwikkelen LIPU plug-in

In deze sprint heb ik een basale LIPU plug-in geïmplementeerd. Deze plug-in haalt data uit de middleware in de vorm van XML en genereert met behulp van een XLST template de HTML die weergegeven wordt.   
Deze eerste versie van het Zorgportaal LIPU plug-in roept de benodigde functies procedureel aan. Ik heb binnen deze sprint dus nog niet nagedacht over een structuur voor de LIPU plug-in (abstractie, herbruikbaarheid e.d.).

Aan het einde van de sprint kon de LIPU plug-in binnen websites van Pharmeon het volgende overzicht weergeven. De weergegeven zorgservices zijn nog dummy implementaties en kunnen dus nog geen echte data laten zien. De zorgservices zijn opgehaald met het in deze sprint geïmplementeerde deel van de middleware.



Figuur 10: Het Zorgportaal laat een lijst van beschikbare services zien, alles binnen het rode blok wordt gegenereerd door de Zorgportaal LIPU plug-in.

## 

## Sprint 2: Het registreren en inloggen van gebruikers

Binnen deze sprint zijn de User Stories behandeld die gaan over het inloggen en registreren van gebruikers binnen het Zorgportaal:

1. Als een ongeregistreerde patiënt wil ik bij het inschrijven bij een zorgverlener ( voor offline administratie) ook de mogelijkheid hebben om meteen een online account aan te maken. Zodat ik niet twee keer dezelfde informatie hoe in te vullen.
2. Als patiënt wil ik maar eenmaal registreren bij een website zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef te registreren.
3. Als patiënt wil ik per bezoek aan een website maar eenmaal inloggen zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef in te loggen.
4. Als patiënt wil ik bij het navigeren naar een zorgservice kunnen inloggen zodat ik bij het switchen naar een andere zorgservice niet opnieuw hoef in te loggen.

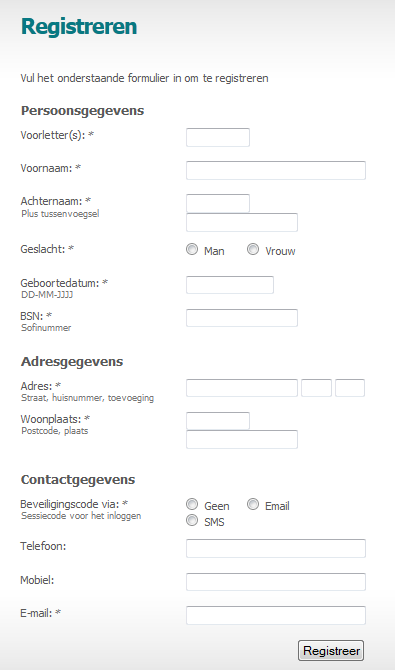
De definition of done voor deze sprint was als volgt:

* “Een patiënt kan zich registreren binnen het Zorgportaal.”
* “Een patiënt kan na het registreren in het Zorgportaal, inloggen in het Zorgportaal en de rest van de website.”
* “Een ingelogde patiënt kan gebruik maken van meerdere zorgservices.”

Om de definition of done binnen deze sprint te realiseren heb ik de volgende taken uitgevoerd:

* Web formulieren module ontwikkelen
* User service ontwikkelen
* Sessionmanger aan de middleware toegevoegd
* De LIPU Plugin herstructurering

### Web formulieren

Een groot deel van de te bouwen functionaliteit was het maken en afhandelen van web formulieren zoals de registratie en inlog formulieren. Om de beste manier van het maken en afhandelen van web formulieren te vinden heb ik binnen Pharmeon rondgevraagd naar de manier waarop zij formulieren maken en afhandelen. Bij het rondvragen zijn mij enkele problemen met de standaard werkwijze rondom formulieren binnen Pharmeon opgevallen.

Figuur 11: Een voorbeeld van een web formulier

**Problemen**  
Web formulieren binnen Pharmeon worden per stuk ontwikkeld. Ieder web formulier wordt in vier delen ontwikkeld; de HTML, de clientside validatie, de serverside validatie en de afhandeling van het formulier. Deze delen beschrijf ik hieronder samen met de problemen die zich voordoen bij de huidige manier van formulieren ontwikkelen.

* **De HTML** wordt in een XSLT template geschreven. Dit gebeurd wel volgens standaarden maar er lopen meerdere standaarden door elkaar heen (in tables of in lists bijvoorbeeld). Bij de HTML wordt ook CSS ontwikkeld, deze kan vaak hergebruikt worden van eerdere formulieren.  
  **Problemen**
  + Als een dergelijke ‘standaard’ veranderd zou worden moet per form (en dat zijn er veel 100+) de template aangepast worden.
  + Het schrijven van de templates is veel knip en plak werk en lijdt daarom tot dubbele code, aanpassingen of bugfixes moeten hierdoor op meerdere plaatsen gedaan worden.
  + De geschreven template is natuurlijk nog niet getest en dus moet per template getest (integratie tests) worden. Dit kan tijdrovend zijn.
* **De clientside validatie** wordt gedaan met jQuery validation. De configuratie voor deze jQuery plug-in wordt per form geschreven.  
  **Problemen**
  + De geschreven configuratie is niet getest en moet daarom per formulier getest (integratie tests) worden.
  + Als de jQuery validation plug-in door een update veranderd is, zal het voor Pharmeon moeilijk zijn deze nieuwe versie ook te gaan gebruiken. Het is namelijk mogelijk de configuratie die per form geschreven is niet werkt met de nieuwere versie. In dat geval zal voor elke form de configuratie bewerkt en opnieuw getest moeten worden, dit is te tijdrovend en er zal gekozen worden om geen update naar de nieuwe versie van de validatie te doen.

**Mogelijkheid**De berichten die door de jQuery validation worden weergegeven zijn geschreven in één taal (meestal Nederlands soms Engels). Door clientside validatie beter te regelen is het makkelijker om lokalisatie (sites in meerdere/andere talen) of de formaliteit (u vorm of jij vorm) van de berichten in te voeren.

* **De serverside validatie** wordt per form geschreven en is vaak onderdeel van de afhandeling van een form.  
  **Problemen**
  + Omdat serverside validatie per form schrijven tijdrovend is (bij het ontbreken van een gestandaardiseerde manier), wordt er voor gekozen om alleen de belangrijkste waardes serverside nog te controleren. Dit kan mogelijk zorgen voor security issues.
  + Er ligt waarschijnlijk te veel verantwoordelijkheid bij clientside validatie.

**Mogelijkheid**Clientside en serverside configuratie voor validatie op de zelfde plek configureren zodat er geen dubbel werk gedaan wordt.

* **Afhandeling van formulieren** wordt per formulier geschreven.  
  **Problemen**
  + Veel formulieren heb dezelfde afhandeling van de ingevoerde data, bijvoorbeeld het versturen van de mail of het vullen van een database met de ingevoerde data. Deze afhandelingen worden nu per form geschreven en getest. Hier zit ook veel knip en plak werk bij wat weer leidt tot dubbele code (slecht voor de onderhoudbaarheid).

**Mogelijkheid**Door voor de veel voorkomende afhandelingen van formulieren één keer te schrijven (gebruikmakend van de herbruikbare formulieren module) kan veel tijd bespaard worden met implementatie en testen.

**Samenvatting problemen**Kort samengevat komen de volgende problemen naar voren bij de huidige manier van web formulieren maken binnen Pharmeon.

* + Veel dubbele code en dus verloren tijd.
  + Veel opnieuw testen per formulier en dus verloren tijd.
  + Geen hergebruik van geteste herbruikbare code.
  + Geen standaardisatie waardoor veranderingen in de wensen of de omgeving een grote impact hebben (er moet op veel plaatsen code aangepast worden).

**Oplossing: Herbruikbare formulieren**Als oplossing voor de problemen die ik in de ontwikkeling van formulier binnen Pharmeon heb gevonden, ben ik een herbruikbare formulieren module gaan bouwen. Deze module heeft als doel het ontwikkelen van web formulier binnen Pharmeon te standaardiseren en formulieren op te bouwen uit zo veel mogelijk herbruikbare code.

Ik heb de verantwoordelijkheden van de herbruikbare formulieren module als volgt afgebakend.  
De herbruikbare formulieren module is verantwoordelijk voor:

* Het bieden van een eenduidige manier om een formulier aan te maken.
* Het genereren van de HTML van een formulier.
* Het genereren van configuratie voor clientside validatie (met jQuery validation).
* De serverside validatie van ingevoerde gegevens.
* Het bieden van een dataholder voor de ingevoerde gegevens. Deze dataholder biedt de ingevoerde gegevens op een gestandaardiseerde wijze aan zodat deze dataholder aan meerdere afhandelingen van een formulier meegegeven kan worden (voorbeelden van afhandelingen van een formulier zijn het vullen van een database met de ingevoerde gegevens of het versturen van een mail met de ingevoerde gegevens).
* De herbruikbare formulieren module moet in ieder geval 80% van de formulieren binnen Pharmeon kunnen aanbieden (20%-80% scenario).

In het onderstaande diagram is de functionele werking van de formulieren module weergegeven. De technische uitleg van de formulieren module staat in de bijlage: “Herbruikbare Formulieren”.

****

Figuur 12: In de herbruikbare formulieren module kunnen alle web formulieren in C# omschreven worden, het bovenstaande schema laat zien hoe het formulier aan de gebruiker getoond en ingevuld kan worden.

De herbruikbare formulieren module zorgt er voor dat client- en serverside validatie, de HTML en de afhandelingen van formulieren op een herbruikbare manier geïmplementeerd kunnen worden.

### Userservice

Nadat ik de formulieren module af had heb ik de Userservice gebouwd. De User service gebruikt de informatie uit de formulieren om een gebruiker bijvoorbeeld in te loggen of te registreren. Deze service bestaat als een nieuwe service naast de bestaande zorgservices van Pharmeon, hoewel deze service geen User Interface (webpagina) heeft. De service is verantwoordelijk voor handelingen van gebruikers van het Zorgportaal en de verschillende zorgservices.

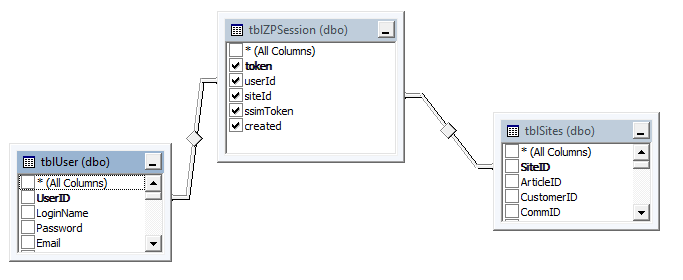
De Userservice maakt gebruik van een bestaande tabellen structuur in de database van Pharmeon, deze tabellen worden omschreven in het ‘Shared tables’ hoofdstuk van de Architectural Description. De business logica is door mij geïmplementeerd, deze bestond voorheen namelijk alleen in ASP. Onder deze business logica valt; het registreren van nieuwe gebruikers en het controleren van inloggegevens van bestaande gebruikers.

### Sessiemanager

Om de gebruikers die de Userservice hebben gebruik om in te loggen bij te houden is de middleware uitgebreid met een sessie manager. Als een gebruiker inlogt, wordt er een nieuwe sessie aangemaakt waar zijn unieke id (userId) wordt bijgehouden.   
Voor deze sprint heb ik een sessiemanager gebaseerd op de database gemaakt. Er is gekozen voor een database gebaseerd ontwerp ten opzichte van een XML door de grote hoeveelheid lees, schrijf en verwijder bewerkingen die bij sessies komt kijken. De XML file zou vaak tegelijkertijd worden gelezen en bewerkt worden wat tot concurrency problemen kan leiden. Bij de ServicePool en Commondatamodel was dit geen probleem, omdat de XML files van deze componenten zelden aangepast hoeven te worden.  


Figuur 13: Session manager in de middleware (groen)

Zoals genoemd slaat de sessiemanager actieve sessies op in de database hiervoor heb ik de tabel tblZPSession toegevoegd in de databases van Pharmeon.



Figuur 14: De tblZPSession tabel houdt de actieve sessies van gebruikers bij.

Deze nieuwe tabel verwijst na de bestaande tblUser en tblSites tabellen die informatie over gebruikers en sites van Pharmeon bijhoud. Meer informatie over deze bestaande tabellen is terug te vinden in de Architectural Description in de Information Viewpoint onder Shared tables.

#### Herstructurering van de LIPU plug-in

In de eerste sprint hoefde de Zorgportaal LIPU plug-in slechts één pagina weer te geven, een pagina met het overzicht van de beschikbare zorgservices. Binnen deze sprint komen daar pagina’s bij voor het registreren en inloggen van gebruikers. Voor het weergeven van meerdere pagina’s is **routing** nodig.   
Om deze routing op een gestructureerde manier te doen heb ik de LIPU plug-in uit de eerste sprint uitgebreid met routing onderdelen

Voor het routen van aanvragen aan de Zorgportaal LIPU Plug-in heb ik de volgende structuur bedacht, deze structuur is gebaseerd op de Controller/Action structuur die in veel frameworks (vooral binnen PHP) terugkomt.



Figuur 15: Klassendiagram LIPU Plug-in Zorgportaal

De LIPU plug-in klasse, waar de request binnenkomt, gaat op zoek naar de juiste Controller. Welke controller de juiste is wordt bepaald aan de hand van de URL van de request. De controller kiest daarna op basis van de Action, die ook bepaald wordt door de URL, de juiste afhandeling van de request.

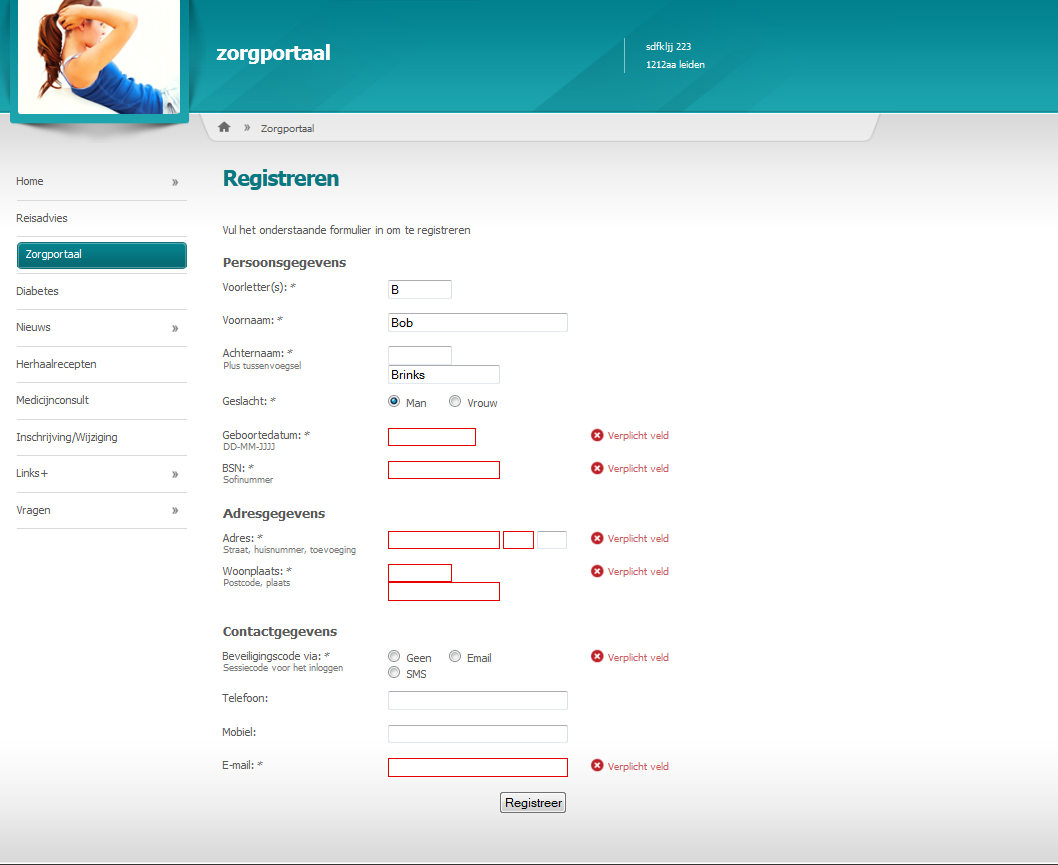
De verschillende controllers zijn implementaties van de abstracte klasse AController. In sprint 1 had ik al logica geschreven die elke controller implementatie kan hergebruiken. Een voorbeeld hiervan is het aanroepen van de template handler, die met behulp van XML en XSLT de HTML genereerd, de logica hiervoor heb ik naar de SetHtml methode verhuisd.

Voor deze sprint heb ik de volgende implementaties van de AController abstracte klasse geïmplementeerd:

* PageController  
  Deze controller is verantwoordelijk voor het weergeven van algemene pagina’s. Het overzicht van zorgservices pagina is binnen deze sprint de enige pagina die deze controller afhandelt. De geïmplementeerde logica voor deze pagina uit sprint 1 heb ik naar deze controller verhuisd.
* ErrorController  
  Deze controller wordt gebruikt als er een exception voordoet in het afhandelen van de action, en deze exception wordt bij deze afhandeling niet afgevangen. De ErrorController genereert op dat moment een foutmelding en toont deze aan de gebruiker.
* UserController  
  Deze controller is verantwoordelijk voor het afhandelen van acties waarin de gebruiker (User) centraal staat. Binnen deze sprint zijn dat de volgende acties: registreren, inloggen en inlogstatus opvragen.

Voor het weergeven van de login en registratie formulieren is de genoemde herbruikbare formulieren module gebruikt. De definitie van deze formulieren en afhandeling vind plaats in een methode van een controller. Voor het registratie formulier bevindt de definitie zich samen met de afhandeling in de UserController. Voor het login formulier bevindt de definitie zich in de AController abstracte klasse, de rede hiervoor is dat het login formulier op meerdere pagina’s weergeven moet kunnen worden. Bijvoorbeeld op een pagina waarvoor de gebruiker ingelogd moet zijn, in het geval dat de gebruiker niet ingelogd heeft. De afhandeling van het login formulier bevindt zich wel in de UserController.

Met de nieuwe structuur van de LIPU plug-in Zorgportaal kon aan het einde van deze sprint de registratie en login formulieren, error pagina’s voor niet afgevangen exceptions en het overzicht van services worden weergegeven.



Figuur 16: Het registratie formulier binnen het Zorgportaal, wordt weergegeven met behulp van de herbruikbare formulieren module

## 

## Sprint 3: Het gebruik van zorgservices in het Zorgportaal

Binnen deze sprint heb ik aan de volgende User Stories gewerkt:

1. Als ingelogde patiënt wil ik mij via het Zorgportaal kunnen aanmelden bij zorgservices zodat ik niet op de website hoef te zoeken naar de verschillende zorgservices die de website aanbied.
2. Als ingelogde patiënt wil ik een overzicht kunnen zien van de zorgservice waar ik gebruik van maak (voor aangemeld ben).
3. Als ingelogde patiënt wil ik een overzicht kunnen zien van mijn status in de service waar ik gebruik van maak, zodat ik in een overzicht kan zien of ik actie moet ondernemen in een of meerdere van de zorgservices.

Voor deze sprint was besproken om twee zorgservices toe te voegen in het Zorgportaal (Tot nu toe werd er gewerkt met dummy zorgservices). De Product Owner heeft ervoor gekozen de eConsult service en de webagenda toe te voegen omdat deze de hoogste prioriteit.  
Voor de eConsult service heb ik in deze sprint dan ook de API gebouwd. Voor de webagenda heb ik ervoor gekozen nog te wachten met het bouwen van de API omdat het Scrum team waar ik mee samen werk in sprint 5 zou zijn de webagenda totaal te herbouwen. De huidige implementatie had geen goede uitbreidbaarheid. Ik zal daarom in sprint 5 pas de webagenda API bouwen.

De definition of done voor deze sprint was als volgt:

* “Een patiënt kan zich via het Zorgportaal aanmelden voor de eConsult service.”
* “Een patiënt kan een lijst zien van services waar hij zich voor aangemeld heeft.”
* “Het Zorgportaal geeft in de lijst van services extra patiënt specifieke informatie.”

Binnen deze sprint heb ik de volgende activiteiten uitgevoerd om de definition of done te behalen:

* Bouwen eConsult API
* Uitbreiden Zorgportaal LIPU plugin
* Bouwen nieuwe template handler
* Weergeven van de service widgets in het Zorgportaal
* Refactoring

### Bouwen van eConsult API

Voor het bouwen van de eConsult API heb ik gekeken naar het hergebruiken van de eerder geïmplementeerde business logica. De eConsult service is ontwikkeld door tijdens een afstudeer project drie jaar geleden. In die tijd draaide elke zorgservice van Pharmeon nog volledig in ASP dus de eConsult service is ook ontwikkeld in ASP.

Deze business logica was niet herbruikbaar in het Zorgportaal om de volgende redenen:

* Geen scheiding volgens MVC  
  Per pagina is voor eConsult een ‘displayPage’ methode gedefinieerd.  
  Het ophalen van de te tonen informatie, het bewerken van data in de database, het definiëren van de HTML en autorisatie van gebruikers gebeurt allemaal in deze displayPage methodes.  
  Hierdoor is het niet mogelijk alleen de informatie op te vragen via de geïmplementeerde business logica, er wordt namelijk altijd een volledige HTML pagina terug gegeven.

Er zijn wel SQL query’s gedefinieerd die herbruikbaar zijn, deze zullen uit de displayPage methodes gehaald moeten worden voordat ze gebruikt kunnen worden.

Om de data uit de eConsult service toch te kunnen weergeven in het Zorgportaal heb ik 3 oplossingsrichtingen bekeken:

1. Het schrijven van een parser in C#  
   Deze parser zou de HTML die de eConsult service genereerd moeten opvragen via HTTP requests en deze HTML daarna zo parsen dat de benodigde data in een bruikbaar formaat wordt doorgegeven aan de middleware van het Zorgportaal (bijvoorbeeld XML). De parser zou ook geschreven kunnen worden aan de ASP kant, wordt de geparste data via HTTP requests aangeboden (als een Web service) aan de middleware van het Zorgportaal.  
     
   Deze oplossing heb ik niet gekozen, het parsen van de HTML naar bruikbare data neemt veel processortijd in beslag en is dus slecht voor de performance van het systeem.  
   Ook is het mogelijk dat er verschillende templates op verschillende sites gebruikt worden waar de parser rekening mee kan houden. Omdat er veel verschillende sites zijn die eConsult gebruiken kan dit aantal templates zo groot zijn dat het bouwen van een parser die rekening houd met alle templates te veel tijd in beslag neemt.  
   Daar komt nog bij dat er ook een oplossing moet komen voor het ophalen van data waar een gebruiker voor ingelogde moet zijn. Dit zou moeten gebeuren met het doorsturen van sessies of het delen van sessies tussen de C# en ASP omgevingen. Ook dit kost veel ontwikkeltijd en brengt mogelijk security kwesties met zich mee.
2. Het refactoren van de ASP code van de eConsult service  
   Door met deze refactoring scheiding volgens MVC toe te passen is het mogelijk de data uit de eConsult service in een bruikbaar formaat, bijvoorbeeld XML, aan te bieden.  
     
   Deze oplossing zou heel goed zijn als ASP nog relevant was, maar Pharmeon wil ASP uitfaseren en nieuwe code in ASP toevoegen helpt dit uitfasering project niet.
3. Het herschrijven van de business logica in C#  
   Om de feature “Bouwen API voor eConsult” binnen de scope van User Story 8 “Weergeven van informatie uit verschillende zorgservices in het Zorgportaal” te realiseren heb ik ervoor gekozen de business logica voor het ophalen van data in de eConsult service te implementeren in C#, deze implementatie zal gebaseerd zijn op de SQL Query’s uit de ASP implementatie.  
     
   Door deze business logica te implementeren in C# wordt er zo min mogelijk ontwikkeltijd verspeeld. De geïmplementeerde business logica zal namelijk hergebruikt kunnen worden wanneer de ASP eConsult service wordt omgezet naar C#.

Ik heb bij het herschrijven van de business logica alleen de classes geïmplementeerd die nodig zijn om een overzicht van eConsult berichten weer te kunnen geven in het Zorgportaal. Deze drie classes maken samen het model op van de eConsult service:

1. EConsult  
   Een klasse die een instantie (per site) van eConsult representeert.
2. EConsultUser  
   Een klasse die een gebruiker (per site per userId) van eConsult representeert
3. EConsultMessage  
   Een klasse die een eConsult bericht representeert.

Naast het model het ik de EConsultAPI geschreven die de eConsult berichten per gebruiker doorstuurt naar de Middleware. Dit doorsturen gebeurt op de volgende wijze:

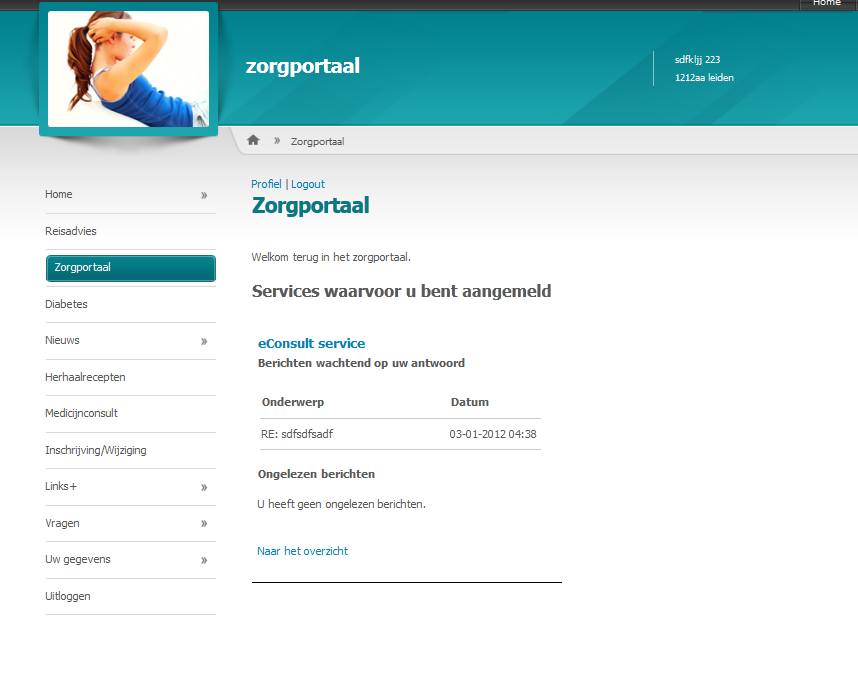


Figuur 17: Het ophalen van data uit de eConsult service door de middleware

### Het uitbreiden van de Zorgportaal LIPU Plug-in voor het weergeven van widgets

De berichten uit de eConsult service moeten in het Zorgportaal weergegeven kunnen worden. Hiervoor heb ik een widget structuur bedacht. Iedere zorgservice in het Zorgportaal zal zijn eigen widget hebben.   
Deze widget bestaat uit een widget klasse in C# en XSLT die de XML uit die klasse weergeeft. Voor het ophalen van de juiste widget klasse en XSLT per zorgservice maak ik gebruik van een WidgetFactory klasse die op basis van service naam opzoek gaat naar de bijbehorende widget en XSLT.

De uitgebreide Zorgportaal LIPU Plug-in kan aan het einde van deze sprint het volgende overzicht weergeven met daarin de eerste zorgservice in het Zorgportaal, de eConsult service.



Figuur 18: De eConsult widget in het Zorgportaal

### Slack time: Ontwikkeling nieuwe template handler

Binnen Scrum is het gebruikelijk de sprints niet voor 100% vol te plannen maar ruimte te laten voor slack time. Deze tijd kan door Scrum teamleden gebruikt worden bestaande systemen te refactoren/ te verbeteren of nieuwe onderdelen toe te voegen. Deze verbeteringen hoeven niet direct gerelateerd te zijn aan User Stories/ eisen van de klant.

Tijdens deze sprint heb ik mijn slack time gebruikt om een nieuwe template handler te ontwikkelen. Dit heb ik gedaan omdat de oude template handler van Pharmeon performance issues had en onnodige handelingen bij het instellen vereiste. Ook ondersteunde de oude template handler niet alle XSLT tags. Vanuit Pharmeon was al een wens om een nieuwe template handler te ontwikkelen, er had ook al een lid van een ander Scrum team (Wouter Saarloos) naar gekeken.

#### Eisen van de template handler

Alle data die op pagina’s van websites van Pharmeon getoond wordt staat in de vorm van XML. Deze XML wordt met behulp van XSLT omgezet naar HTML die door een browser getoond kan worden. Dit omzetten van XML naar HTML gebeurd in de template handler.

Bij het omzetten van XML naar HTML moet de template handler op zoek gaan naar de juiste templates (XSLT). Hierbij moet het rekening houden met de volgende gegevens:

* De LIPU plug-in die de pagina weergeeft.
* De pagina die weergegeven moet worden. Dit wordt aangegeven met de naam van de template die bij de betreffende pagina hoort.
* De site of community specifieke weergave van de pagina. Iedere site van Pharmeon valt binnen een community, per community kunnen andere templates gebruikt worden voor de zelfde pagina’s. Dit kan ook nog verschillen op site niveau.



Figuur 19: Werking template handler

#### Problemen met de bestaande template handler

De bestaande template handler zoek de juiste templates op via een database. In deze database staat bij welke LIPU plug-in, pagina, community en website een template hoort. Voor sommige oudere templates staan ook de XSLT in de database, voor nieuwere templates staat de XSLT in files op de harde schijf waar naar gerefereerd wordt in de database.  
Het gebruik van een database voor het kiezen van de juiste template brengt twee problemen met zich mee.

* Voor het ophalen van de juiste template moet er één of meerdere database query’s gedraaid worden. (impact op de performance van de website)
* Voordat een LIPU plug-in gebruik kan maken van een bepaalde template moet hiervoor een record in de database aangemaakt worden.

Binnen XSLT is het gebruikelijke templates op te splitsen over meerdere files om de templates onderhoudbaar te houden. Veelvoorkomende stukken template staan in losse files en heel grote templates worden opgesplitst over meerdere files. Header, content en footer scheiding komt ook veel voor. Dit gebruik binnen XSLT is met de huidige template handler moeilijk te realiseren.

Om met al deze losse XSLT files de template voor een pagina op te bouwen wordt binnen XSLT gebruik gemaakt van import en include tags. Deze tags worden door de huidige template handler niet ondersteund. Om templates toch op te kunnen splitsen maakt de huidige template handler gebruik van naar speciale Pharmeon include tags.

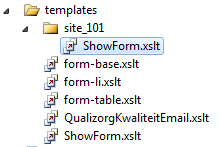
De opgesplitste delen van een template kunnen per website of community verschillen en er moet dus in de database opgezocht worden welke versie van het onderdeel gebruikt moet worden. Om dit te kunnen realiseren heeft de huidige template handler een functie die op zoek gaat naar de speciale Pharmeon include tags. Op basis van de templatenaam die in deze speciale include tags staan kan de functie de juiste template onderdelen uit de database halen. Het is hierbij niet mogelijk ‘nested-includes’ te hebben, een geinclude template kan zelf geen includes hebben.

In de praktijk zorgt dit ervoor dat per LIPU plug-in één ‘main’ template aangemaakt wordt en alle andere templates bij die plug-in in deze ‘main’ template geinclude worden. Deze situatie brengt ook weer drie problemen met zich mee.

* De template onderdelen missen volgens Visual Studio template onderdelen (includes), het is dus niet mogelijk met Visual Studio de XSLT te laten valideren. Deze functionaliteit van Visual Studio wordt binnen Pharmeon dan ook niet gebruikt, terwijl het gebruik wel kan leiden tot minder fouten in de XSLT.
* Omdat de ‘main’ template alle template onderdelen van een LIPU plug-in include, worden er vaak ongebruikte delen XSLT ingeladen en door de XslTransform klasse ( een .NET klasse voor het parsen van XML met XSLT) geparst. Dit zorgt ook voor performance issues omdat deze XSLT soms enorm groot groeit bij uitgebreide LIPU plug-ins.
* Voor elke include wordt een database query uitgevoerd, dit zorgt voor een lastige keuze voor de ontwikkelaars. Of meer XSLT in één template bestand plaatsen, waardoor de onderhoudbaarheid van de templates lager wordt, of de templates beter opsplitsen waardoor de performance van de templates slechter wordt.

#### De nieuwe template handler die al in ontwikkeling was

Om de problemen met de oude template handler op te lossen was een lid van een ander Scrum team (Wouter Saarloos) al begonnen met het bouwen van een nieuwe template handler. Deze template handler laat het database gedeelte van de oude template handler weg en het zoeken van de juiste templates op een andere manier oplossen.   
De nieuwe oplossing maakt gebruik van een vaste mappenstructuur voor de templates. Zoals in de onderstaande afbeelding is weergegeven zullen templates voor specifieke websites of communities in submappen staan. Deze submappen hebben de naam site\_%siteId% of com\_%communityId%.



Figuur 20: Mappen structuur nieuwe template handler

De nieuwe template handler zoekt op basis van de templatenaam de submappen door naar site of community specifieke templates. Door gebruik te maken van deze mappenstructuur hoeft de juiste template niet meer in de database opgezocht te worden.

De nieuwe template handler maakte nog steeds gebruik van de speciale Pharmeon include tags om templates te kunnen includen. En ondersteuning voor ‘nested-includes’ kostte veel moeite en was dus ook niet geïmplementeerd in deze nieuwe template handler. De performance problemen met alle includes in een ‘main’ template waren door deze nieuwe template handler dan ook nog niet opgelost.

#### Mijn nieuwe template handler

Ik heb de ontwikkeling van de nieuwe template handler overgenomen om problemen van ‘nested-includes’ op te lossen. Ik heb er voor gekozen om de speciale include tags en de bijbehorende functie die deze include tags behandelde niet uit te breiden met de mogelijkheid van ‘nested-includes’. In plaats daarvan heb ik gekeken naar de werking van de .NET XslTransform klasse.

XslTransform klasse:  
“Transformeerd XML data aan de hand van een Extensible Stylesheet Language for Transformations (XSLT) style sheet.”

In de documentatie van de XslTransform klasse heb ik gevonden dat deze gebruik maakt van een XmlUrlResolver om te includen XSLT bestanden te vinden.   
Ik heb mijn eigen implementatie van de ResolveURI methode geschreven in een subklasse van de XmlUrlResolver klasse. Deze implementatie gaat zoek naar de juiste template bestanden op basis van een siteId, een communityId en de vaste mappen structuur opgezet door Wouter Saarloos.  
Met deze XmlUrlResolver voldoet de XslTransform klasse aan alle eisen voor de Pharmeon template handler en lost het ook de problemen op die de oude template handler had.

Ook kan er met deze nieuwe template handler gebruik gemaakt worden van de standaard XSL include en import tags. Hierdoor wordt de geschreven XSLT automatisch gevalideerd door Visual Studio.

De oude templates van de vele verschillende projecten van Pharmeon moeten nog omgezet worden. Deze maken namelijk nog gebruik van de oude manier van includen en staan nog niet in de nieuwe mappen structuur.  
De nieuwe template handler kan naast de oude gebruikt worden en is al in meerdere projecten die tegelijkertijd met mijn afstuderen draaiden gebruikt.

## 

## Sprint 4: Authenticatie met de beveiligde omgeving (SSIM)

In deze sprint heb ik aan de volgende User Stories gewerkt.

1. Als ingelogde patiënt wil ik per bezoek maar eenmalige een sessiecode invoeren zodat ik bij het switchen van zorgservices minder tijd kwijt ben met het invoeren van sessiecodes.
2. Als ingelogde patiënt wil ik dat gegevens die ik eerder in het systeem heb in gevoerd, niet opnieuw ingevoerd hoeven te worden in web formulieren, zodat ik geen tijd verspil met het opnieuw invoeren van al bekende gegevens.
3. Als patiënt wil ik vanuit elke pagina op de website altijd terug kunnen naar het Zorgportaal zodat ik nooit op een pagina ‘vast’ kom te zitten.
4. Als beheerder van een website wil ik kunnen aanpassen welke zorgservices mijn website aanbied binnen het Zorgportaal.

Voor User Story 9 wordt er gebruik gemaakt van een beveiligde omgeving, de **SSIM omgeving**. In deze SSIM omgeving staan gevoelige gegevens van patiënten opgeslagen. Voordat patiënten en andere gebruikers deze gegevens mogen inzien moeten ze een **sessiecode** invoeren. Deze sessiecode wordt verzonden via e-mail en/of sms.

Om deze User Stories te implementeren heb ik de bestaande registratie en login processen van het Zorgportaal uitgebreid om gebruikers in de SSIM omgeving te kunnen authentiseren. Hiervoor heb ik de volgende taken uitgevoerd:

* De CommCommunicatorAPIProxy ontwikkelen  
  Een nieuwe zorgservice proxy ontwikkelen voor communicatie met de SSIM.
* De SsimAuthController ontwikkelen  
  De Zorgportaal LIPU plug-in uitbreiden met een nieuwe controller.
* De Configuration tool ontwikkelen

In de slack time van deze sprint heb ik ook nog de manier waarop widgets informatie uit de zorgservices ophalen op een andere manier geïmplementeerd.

### De CommCommunicatorAPIProxy

In alle voorgaande sprints was het mogelijk om alle zorgservices in dezelfde runtime te draaien, hierdoor heb ik altijd gebruik gemaakt van de DirectCallAPIProxy klasse. Binnen deze sprint moet er gecommuniceerd worden met de SSIM omgeving. De SSIM omgeving draait niet in de zelfde runtime als de middleware. Hiervoor moet er dus een nieuwe zorgservice proxy ontwikkeld worden, namelijk de CommCommunicatorAPIProxy. Deze proxy communiceert met de webserver waar de SSIM omgeving als web service op draait.



Figuur 21: De DirectCallAPIProxy communiceert met zorgservices binnen dezelfde runtime



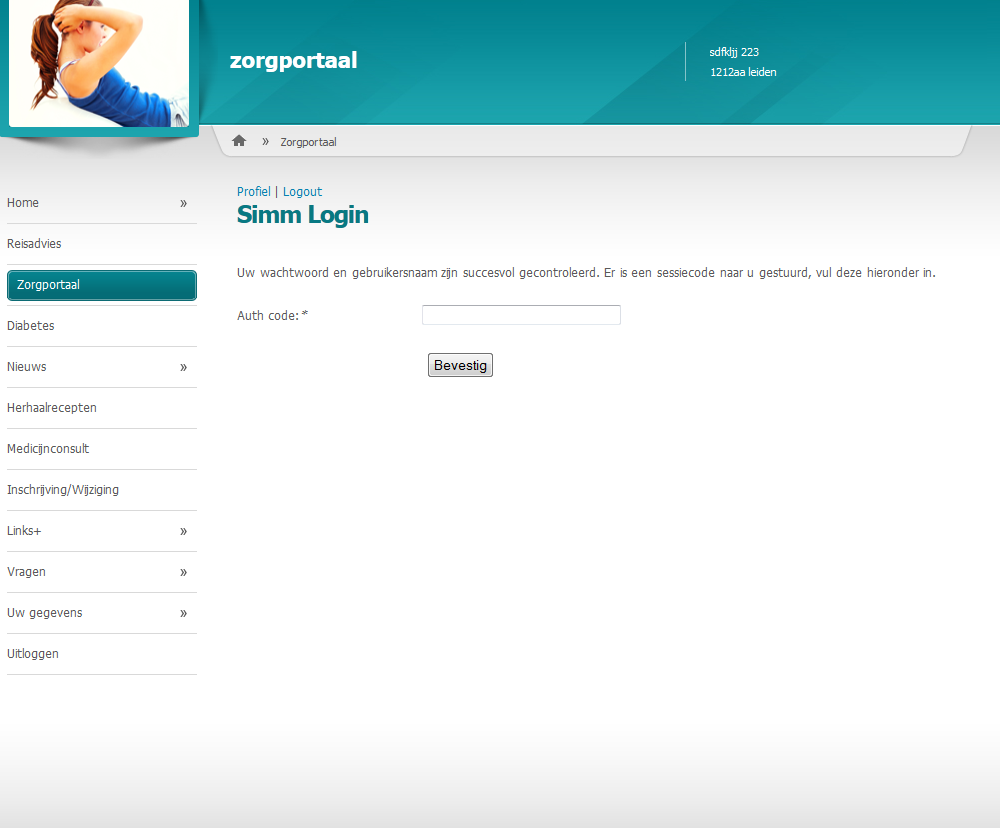
Figuur 22: De CommCommunicatorAPIProxy communiceert met de SSIM web service op de SSIM web service

### SsimAuthController

Binnen deze sprint heb ik de bestaande Zorgportaal LIPU Plug-in uitgebreid met een nieuwe controller: “de SsimAuthController”. Deze controller doet de afhandeling van acties die een gebruiker moet doen om bij de SSIM omgeving te authentiseren. Om deze acties af te handelen wordt gebruik gemaakt van de CommCommunicatorAPIProxy om met de SSIM omgeving te communiceren.

De twee acties die in de SsimAuthController geïmplementeerd worden zijn:

* Het verifiëren van het e-mailadres of het telefoonnummer van de gebruiker.
* Het authentiseren bij de SSIM omgeving. Dit gebeurd door het versturen van een e-mail of SMS met een sessiecode. Deze sessie code wordt dan vergeleken met een waarde in de SSIM omgeving.

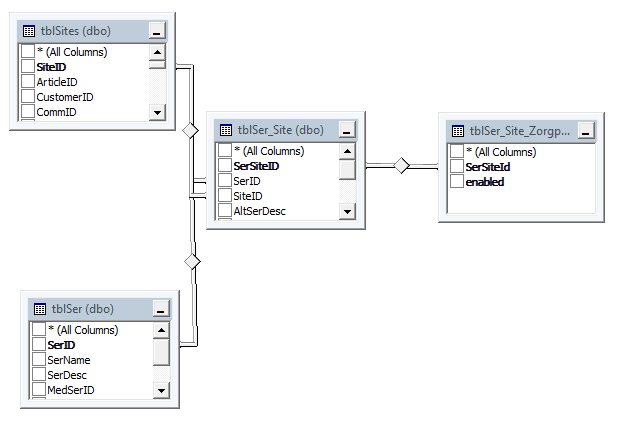


Figuur 23: De gebruiker moet de sessiecode die vanuit de SSIM omgeving verstuurd is invullen

### De Configuration tool

Om de beheerder van de website de mogelijkheid te geven zorgservices aan het zorgportaal toe te voegen en weg te halen heb ik in deze sprint een Configuration tool geïmplementeerd.

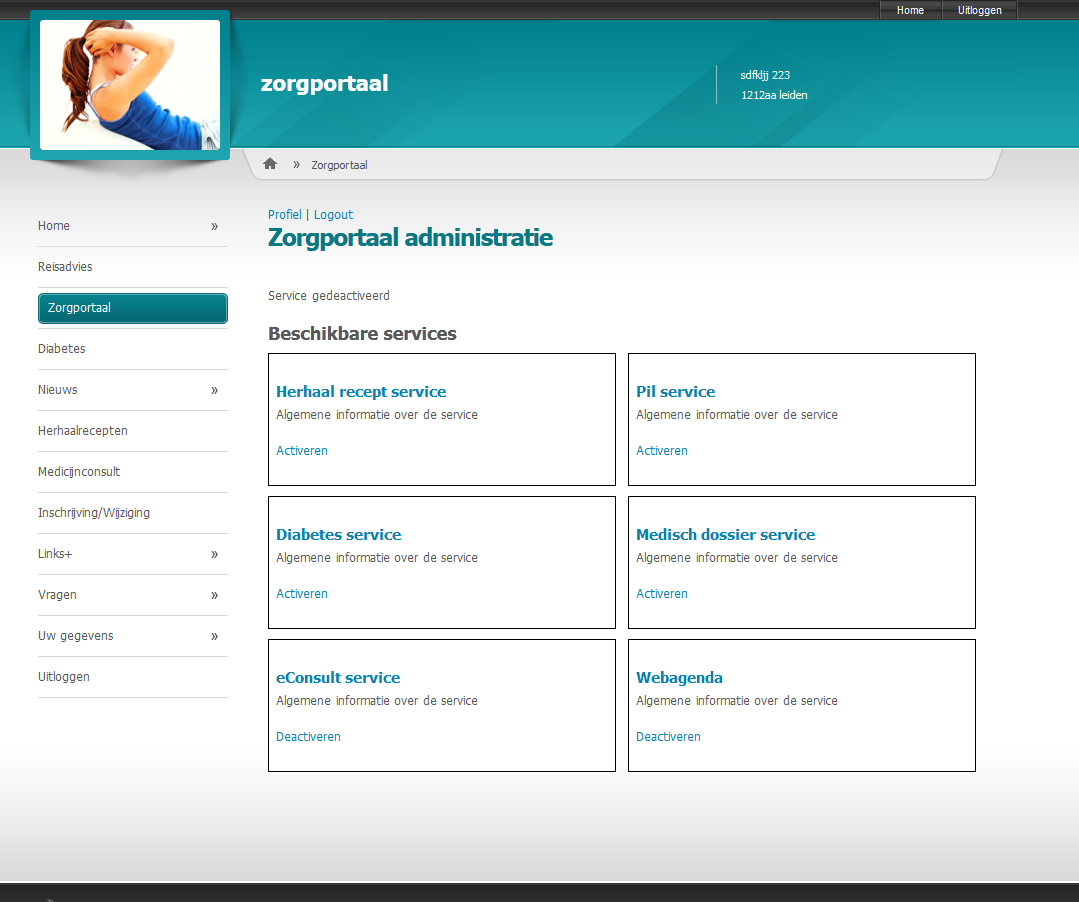
Om de Configuration tool de mogelijkheid te geven services aan of uit te zetten heb ik de functionaliteit van de service pool uitgebreid. Door deze uitbreiding houdt de service pool ook bij welke status (aan of uit) een service heeft, deze status wordt opgeslagen in de database. Hiervoor heb ik een nieuwe tabel toegevoegd, namelijk de tblSer\_Site\_Zorgportaal tabel.



Figuur 24: De tblSer\_Site\_Zorgportaal geeft aan welke services per site geactiveerd zijn.

Deze nieuwe tabel verwijst naar de koppeltabel tblSer\_Site. De tblSer\_Site geeft aan welke services per site beschikbaar zijn, de nieuwe tabel geeft aan welke van de beschikbare services ook in het zorgportaal weergegeven worden.

Naast de uitbreiding van de service pool bestaat de Configuration tool uit een nieuwe controller in de Zorgportaal LIPU Plug-in, namelijk de Admin (administratie) controller. Deze controller kan de onderstaande pagina laten zien en de status van services in de service pool aanpassen.



Figuur 25: De Configuration tool waarin de beheerder van een site services aan of uit kan zetten

### Slack time

De slack time van deze sprint heb ik gebruikt om een fout in het ontwerp van de Widgets en zorgservice api’s te verbeteren.

De getNotification() methode was opgenomen als een standaard onderdeel van de zorgservice API en dus verplicht bij de definitie van elke zorgservice. De widgets van het Zorgportaal gebruiken de methode voor het ophalen van de informatie die zij moeten tonen. Iedere zorgservice API implementeerde de methode op een manier waardoor de benodigde informatie terug gegeven kon worden aan de widgets.

Door de informatie voor de widgets op deze manier op te halen is de logica voor het ophalen van deze informatie in het model beland. (Het model als onderdeel van de MVC scheiding). Het model moet eigenlijk alleen de informatie terug geven van objecten die binnen het domein van die bepaalde model vallen. Het plaatsen van de getNotification() methode in het model is een onjuiste toepassing van MVC.

Voorbeeld: Het model van de herhaalreceptservice moet alleen data over herhaalrecepten bevatten namelijk: welke herhaalrecepten bestaan er, welke recepten horen bij welke patiënt en welke medicijnen horen bij welke recepten.  
Notificaties vallen buiten het domein van de herhaalreceptservice, het zelfde geld ook voor de andere zorgservices.

De logica uit de getNotification() methodes is verplaats naar de controller. De rol van de controller wordt binnen het Zorgportaal project ingenomen door de LIPU plug-in Zorgportaal. De widgets zijn in deze nieuwe structuur zelf verantwoordelijk voor het ophalen van de juiste data uit de zorgservices in plaats van het aanroepen van de getNotification() methodes.

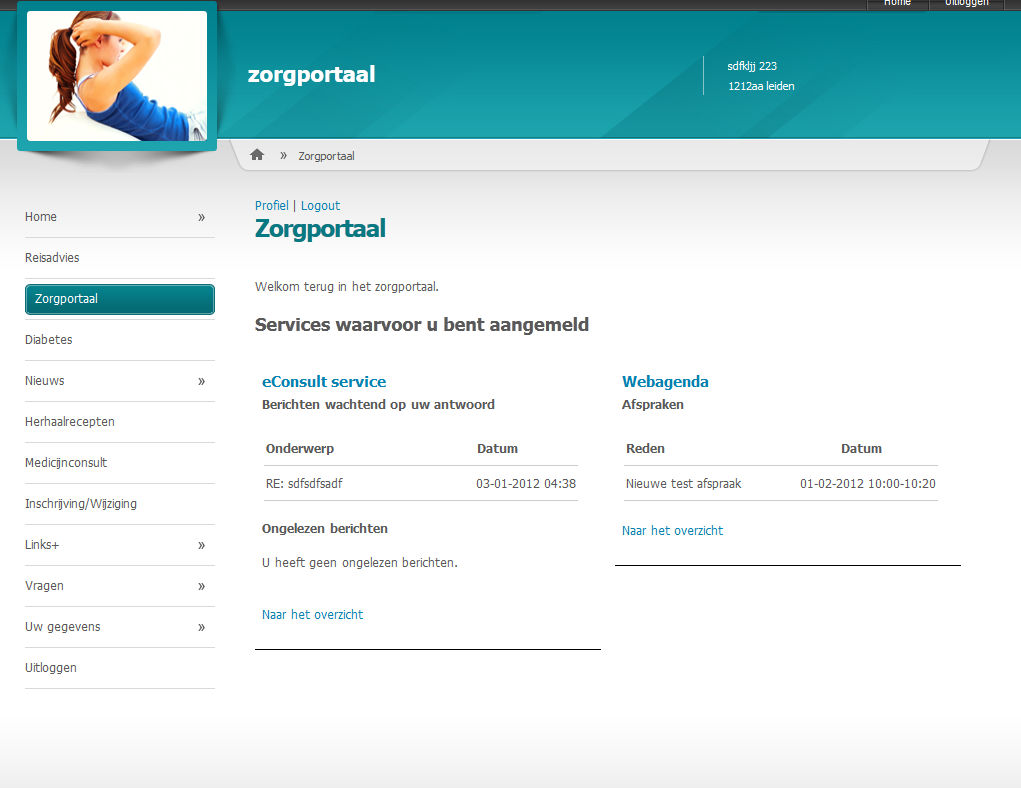
## 

## Sprint 5: Bouwen Webagenda API

Binnen deze sprint help ik mijn Scrum team bij het bouwen van een nieuwe versie van de Webagenda en ontwikkel ik tegelijkertijd de Webagenda API.

De oude webagenda wordt met de nieuwe versie opnieuw gebouwd om beter onderhoudbaar te zijn en om een logische scheiding tussen MVC aan te brengen. Ook zal de nieuwe versie van de webagenda gebruik maken van de nieuwe template handler die ik in sprint 3 ontwikkeld heb.

Door de goede scheiding tussen MVC was het implementeren van Webagenda API simpel. Ik heb methodes geschreven die de afspraken per gebruiker kunnen ophalen. De widget voor de webagenda gebruikt de middleware dan om deze afspraken op te halen en geeft deze weer.



Figuur 26: De webagenda widget geeft afspraken uit de webagenda van de gebruiker weer met behulp van de webagenda API

## 

## Sprint 6: Backlog afwerken

In deze laatste sprint heb ik de taken opgepakt die nog in de product backlog stonden en veel slacktime gebruikt om de systeemdelen uit eerdere sprints te verbeteren. De taken uit de product backlog bestonden uit het verhogen van de code coverage van de Unittests, performance testing en refactoring van bestaande code.

### Verhogen code coverage

Een groot deel van het zorgportaal is ontwikkeld met behulp van Test Driven Developement en heeft daardoor hoge code coverage. Voor sommige delen was het niet mogelijk gebruik te maken van Test Driven Developement door het gebruik van bestaande (ontestbare) componenten. Door deze ongeteste delen was de code coverage voor het gehele project 32%.  
Pharmeon is bezig met het invoeren van een minimum eis voor code coverage van 50%, hierom heb ik het verhogen van de code coverage opgenomen in mijn product backlog.

Om de code coverage te verhogen heb ik bekeken hoe ik de ontestbare code ofwel testbaar kon maken ofwel kon minimaliseren.

**Ontestbare code testbaar maken**Ontestbare code ontstaat vaak door afhankelijkheid (dependencies) van een stuk code op andere componenten of systemen, bijvoorbeeld afhankelijkheid van het bestandssysteem, een database of andere (legacy) code. Deze afhankelijkheden heb ik voor het grootste gedeelte weg kunnen nemen door het gebruik van “stubs” en voor databases het gebruik van “mocking”. (Stubs en mocking technieken volgens “The art of unit testing” van ‘Roy Osherove’).

Aan de hand van het voorbeeld van afhankelijkheden van databases (GeneralDB) zal ik toelichten hoe ik gebruik heb gemaakt van stubs en mocking.

Het GeneralDB component van Pharmeon wordt gebruikt om verbinding te maken met databases en SQL query’s te draaien. Om voor dit component een stub implementatie te kunnen schrijven heb ik eerst een interface (IDbHandler) gedefinieerd die de bestaande GeneralDB klasse implementeert. En in de code die ik wil testen heb ik het gebruik van de GeneralDB klasse vervangen door deze interface (zie voorbeeld).

//GeneralDB db = new GeneralDB();

IDbHandler db = GeneralDBFactory.getGeneralDB();

Om te kunnen Unittesten moet in het bovenstaande voorbeeld een stub versie van de GeneralDB klasse geinitialiseerd worden, hiervoor zijn binnen Unittesten enkele technieken. Ik heb ervoor gekozen om een Factory klasse (GeneralDBFactory) te schrijven.

De Factory klasse (hieronder weergegeven) initialiseert de juiste implementatie van IDbHandler, voor de productie/ontwikkel omgeving is dit de GeneralDB implementatie en voor de (Unit)testomgeving kan dit een stub of mocking implementatie zijn. Het kiezen van de juiste implementatie van een object in een Injector (in dit geval de Factory klasse) is een voorbeeld van Dependency Injection (http://en.wikipedia.org/wiki/Dependency\_injection).

public static class GeneralDBFactory

{

private static IDbHandler dbHandler;

public static IDbHandler getGeneralDB()

{

if (GeneralDBFactory.dbHandler == null)

{

GeneralDBFactory.dbHandler = new GeneralDB();

}

return GeneralDBFactory.dbHandler;

}

public static void setTestingDatabaseHandler(IDbHandler dbHandler)

{

GeneralDBFactory.dbHandler = dbHandler;

}

}

Voor verschillende Unittest heb ik gebruik gemaakt van verschillende implementaties van de IDbHandler die voor die specifieke Unittest gewenste waardes terug geven.

Het gebruik van Factories om afhankelijkheden weg te halen heb ik ook gebruik voor veel andere componenten, zoals de Sessionmanger, de filemanager, de templatehandler, de connectionstringmanager en het middleware-systeem. Met het gebruik van deze Factories heb ik Unittests kunnen schrijven en de code coverage van het project naar 69% gebracht.

**Unittesting binnen Pharmeon in een workshop**  
De Factories en interfaces die ik geschreven heb voor mijn Unittests zullen in het vervolg ook gebruikt worden door de andere ontwikkelaars binnen Pharmeon om er voor te zorgen dat er binnen Pharmeon meer Unittestbare code geschreven wordt. Hiervoor heb ik een Unittest project in Visual studio aangemaakt met documentatie hoe er getest kan worden met behulp van deze Factories. Ook ga ik samen met Wouter Saarloos (een collega bij Pharmeon) een workshop geven aan de ontwikkelaars binnen Pharmeon over Unittesten en het schrijven van testbare code.   
In deze workshop zal ook de herbruikbare formulieren module en de templatehandler waar het Zorgportaal gebruik van maakt toegelicht worden.

# 

# Evaluatie van het opgeleverde product

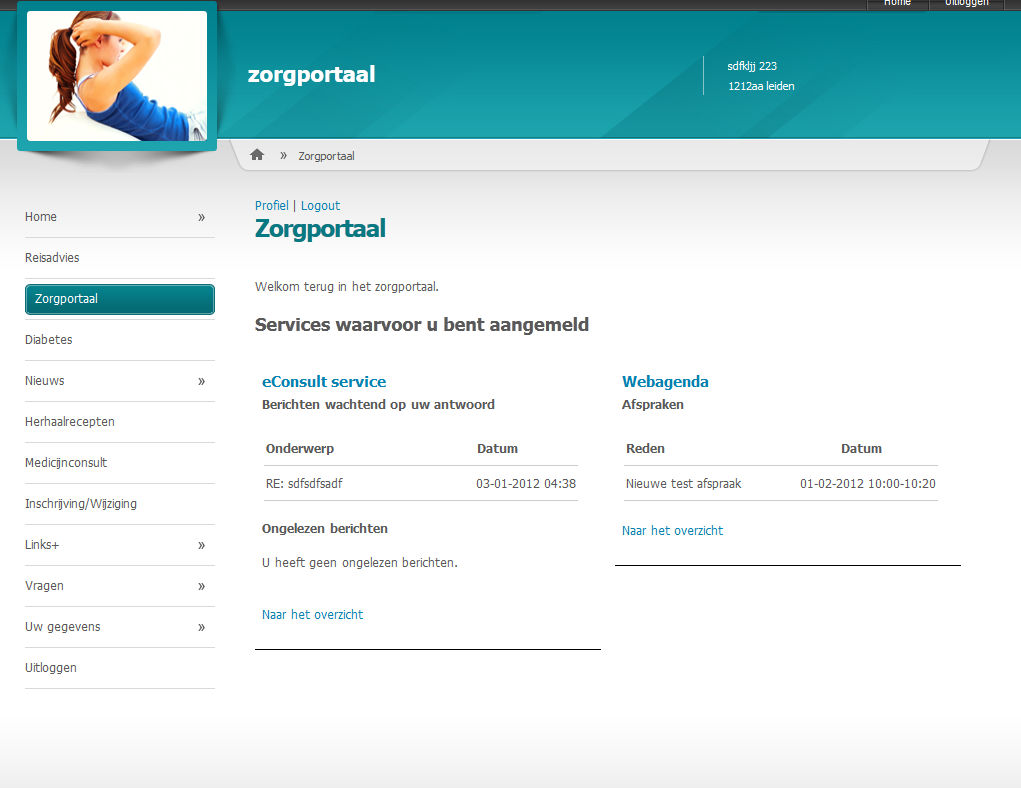
In dit hoofdstuk beschrijf ik of het zorgportaal systeem aan de doelstelling van het project voldoet.

**Is de doelstelling behaald?**  
De doelstelling van het project was als volgt:

*De doelstelling voor de opdracht is het ontwikkelen van het zorgportaal waarmee een totaaloverzicht van een patiënt weergegeven en aangepast kan worden.*

*Bij het ontwikkelen van het zorgportaal ligt de nadruk op de backend.*

Het zorgportaal systeem is in staat informatie uit meerdere zorgservices te tonen in een totaal overzicht van een patiënt. In het opgeleverde product zijn twee zorgservices beschikbaar binnen het zorgportaal (zie screenshot). Hiermee voldoet het zorgportaal aan het resultaat dat beschreven wordt in het afstudeerplan.



Het opgeleverde product voldoet aan alle User Stories behalve 13, het bijhouden van statistieken over het gebruik van het zorgportaal. Deze Userstory is komen te vervallen omdat het een te lage prioriteit had.   
Het opgeleverde product is geunittest en bij het afronden van deze afstudeerperiode worden acceptatie tests uitgevoerd waarna het product live gezet zal worden.

Met de functionaliteiten die het opgeleverde product bied vind ik dat het zorgportaal project geslaagd is.

**De integratie tussen de zorgservices en de toekomst van het zorgportaal**Het grote probleem wat het zorgportaal project moest oplossen was de slechte integratie tussen de verschillende zorgservices. De middleware die is ontwikkeld in dit project bied de mogelijkheid alle zorgservices op de zelfde manier te integreren en lost hiermee het probleem van de integratie op. Deze middleware voldoet ook aan alle eisen die er aan het begin van het project aan gesteld werden.

Deze verbeterde integratie zal voor Pharmeon in de toekomst enkele mogelijkheden bieden. Allereerst zullen alle zorgservices een API gaan bieden voor communicatie met de middleware.   
Hierna zijn er plannen om de middleware uit te breiden met een web service die de zorgAPI van de middleware aanbied. Deze web service zou als data bron kunnen bieden voor toekomstige projecten van Pharmeon zoals:

* Mobiele applicaties zoals app-versies van de zorgservices en het zorgportaal.
* Zorgwidgets op websites die niet door Pharmeon beheerd worden.
* Synchronisatie met huisarts, tandarts en/of apotheek systemen.

## Slacktime producten

Tijdens mijn afstudeerproject heb ik binnen enkele sprints extra tijd genomen om de componenten Herbruikbare web formulieren en de nieuwe template handler uitgebreider te implementeren dan nodig voor het Zorgportaal project. Deze componenten worden binnen Pharmeon nu voor alle projecten gebruikt.

Deze slacktime producten lossen enkele structurele problemen binnen de software van Pharmeon op. Zo zorgt de nieuwe template handler ervoor dat de ontwikkelaars binnen Pharmeon nu de XSLT 1.0 standaard kunnen aanhouden in de templates en de templates kunnen makkelijker georganiseerd worden binnen de verschillende projecten.

Het herbruikbare web formulieren component lost een boel problemen op. Vooral de onderhoudbaarheid en ontwikkeltijd (inclusief testing) van web formulieren binnen Pharmeon is verbeterd.

# 

# Proces evaluatie

## Planning

Zoals beschreven is er in het plan van aanpak een globale planning opgesteld. In deze globale planning heb ik 7 sprints opgenomen en daarbij geschat dat alle User Stories binnen die 7 sprints gehaald konden worden. Deze globale planning is goed aangehouden en alle User Stories die nog prioriteit hadden zijn binnen de 7 sprints afgerond (Userstory 13 is niet afgerond omdat deze geen prioriteit meer had).

Aan het begin van iedere sprint heb ik uit het product backlog een aantal User Stories gekozen om tijdens die sprint te realiseren. In iedere sprint ben ik in staat geweest de gekozen User Stories af te ronden en in een aantal sprints was er ook tijd over voor slacktime.

De planning over het hele project is soepel verlopen.

## Ervaring met Test Driven Developement

Over het algemeen vind ik Test Driven Developement een prettige ontwikkel methode. Voor de systeemdelen die ik zelf ontwikkeld heb (de middleware, de formulieren module en de LIPU plug-in) is het ook gelukt om er goed gebruik van te kunnen maken.

Voor de systeemdelen die gebruik maakte van bestaande systemen is TDD niet bruikbaar omdat er te veel ontestbare code in de bestaande systemen zat. Hierdoor heb ik geleerd dat TDD een goede ontwikkel methode is maar je moet bij onderhoudt en/of doorontwikkeling van bestaande software snel door hebben waar je het kan gebruiken en waar niet.

Door het gebruik van TDD heb ik veel geleerd over Unittesten, Dependency Injection en Mocking. Deze nieuwe vaardigheden heb ik in de laatste sprint kunnen gebruiken om de gemaakte code voor 75% code coverage te Unittesten. Hier ben ik zeer tevreden over.

## Ervaring met Scrum

Ik heb Scrum als prettig ervaren. Het heeft ervoor gezocht dat ik op elk moment wist op welke Userstory of taak ik mij moest focussen.

Ik ben wel blij dat ik vooraf de architectuur van het systeem al opgezet had. Dit is niet gebruikelijk binnen Scrum.  
Zonder dit vooraf opzetten had ik mogelijk niet de juiste oplossing gekozen of de middleware niet op dezelfde manier geïmplementeerd.

Sprints binnen Scrum in combinatie met User Stories heb ik ervaren als een goede manier om gefocust te blijven op de dingen die op dat moment belangrijk is. Je programmeert de code die nodig is voor de User Story waar je me bezig bent, kan je de actie die in de User Story uitvoeren dan ben je klaar en ga je verder met de volgende User Story. Met deze werkwijze weet jij (en je team leden) altijd waar je mee bezig bent en je minimaliseert de hoeveelheid onnodig werk.  
Ook de snelle feedback aan het einde van elke sprint heb ik als positief ervaren.

# 

# Evaluatie beroepstaken

## 2.1 Opstellen gegevensmodel voor een database

Deze beroepstaak is matig aanbod gekomen tijdens het afstudeerproject. Er zijn wel gegevensmodellen voor de bestaande databases opgesteld, terug te vinden in het AD. Ook zijn er enkele tabellen aan de bestaande databases toegevoegd.

Binnen het afstudeerproject is deze beroepstaak niet op een lastig niveau bewezen.

## 3.1 Ontwerpen softwarearchitectuur

Deze beroepstaak is bewezen door het opstellen van de architectuur van het Zorgportaal inclusief de middleware. Deze architectuur wordt beschreven in de architectural description.  
Het zorgportaal systeem wordt in deze architectuur opgedeeld in componenten en er wordt beschreven dat de architectuur gedistribueerde componenten ondersteund.  
De eisen voor deze architectuur zijn voort gekomen uit gespreken met de Product Owner en het hoofd van de technische afdeling.  
Ook de juiste keuze maken voor de oplossingsrichting (middleware) weegt mee voor deze beroepstaak.

Door het uitvoeren van deze activiteiten is deze beroepstaak behaald op een Lastig of mogelijk Complex niveau.

## 3.2 Ontwerpen systeemdeel

Deze beroepstaak is bewezen door het ontwerpen van de verschillende componenten en systeemdelen van het Zorgportaal systeem. Hieronder vallen:

* De herbruikbare formulieren module
* De Zorgportaal LIPU Plugin
* De Template handler
* De verschillende componenten van de middleware: de service pool, de Sessionmanger, de zorgservice api’s en proxies en de Configuration tool.

Bij het ontwerpen van deze componenten heb ik gebruik gemaakt van design patterns:

* Factory
* Singleton
* Proxy
* Brooker
* **Interpreters**

De ontwerpen van de verschillende componenten hebben rekening gehouden met de testbaarheid en afhankelijkheid van onderliggende systemen voor het gebruik van Test driven developement en voor Unit testen.

Met de bovenstaande activiteiten is deze beroepstaak op een Complex niveau behaald.

## 3.3 Bouwen applicatie

Deze beroepstaak is behaald door het bouwen, documenteren en Unit testen van de volgende systeemdelen:

* De herbruikbare formulieren module
* De Zorgportaal LIPU Plugin
* De Template handler
* De verschillende componenten van de middleware: de service pool, de Sessionmanger, de zorgservice api’s en proxies en de Configuration tool.

De systeemdelen en Unit tests zijn geïmplementeerd in C# met Visual studio. De herbruikbare formulieren module is voor dit project het beste voorbeeld van rekening houden met herbruikbaarheid van de geschreven code. De middleware houd door zijn lage coupling rekening met toekomstige wijzigingen en hun impact. De gebouwde systeemdelen sluiten aan op bestaande software waar onder LIPU en de SSIM omgeving. Er is gebruik gemaakt van een testomgeving voor het testen van de gemaakte componenten. De versies van de geschreven code zijn door Subversion beheerd.

Met deze activiteiten is deze beroepstaak behaald op een Complex niveau.

1. De definition of done geeft binnen Scrum aan wanneer iets klaar is en er niet meer verder aan gewerkt wordt. [↑](#footnote-ref-1)
2. Serialiseren is het omzetten van een object in het geheugen van een programma naar een string, een binaire reeks of een ander formaat dat op een harde schijf, over een netwerk etc. weggeschreven kan worden. [↑](#footnote-ref-2)
3. Deserialiseren is het omgekeerde van serialiseren, hierbij wordt een object in het geheugen van een programma opgebouwd vanaf een string, binaire reeks of een ander formaat. [↑](#footnote-ref-3)