Departement N-Tech

EXTRANET VOOR ARTSEN

Sven Salemans
Raf Thewis

professionele bachelor
in de toegepaste informatica

Promotoren
Ing. S. De Block — **Wit-Gele Kruis**
Mevr. R. Lambrechts — **XIOS Hogeschool Limburg**

Bachelorproef academiejaar 2012-2013

WIT-GELE KRUIS LIMBURG

Departement N-Tech

EXTRANET VOOR ARTSEN

Sven Salemans
Raf Thewis

professionele bachelor
in de toegepaste informatica

Promotoren
Ing. S. De Block — **Wit-Gele Kruis**
Mevr. R. Lambrechts — **XIOS Hogeschool Limburg**

Bachelorproef academiejaar 2012-2013

WIT-GELE KRUIS LIMBURG

Dankbetuiging

In de afgelopen elf weken hebben we een erg boeiend ontwikkelingsproces doorlopen wat betreft onze informaticavaardigheden. Hierin hebben we zeer goede begeleiding en steun gekregen van een aantal belangrijke mensen. Zonder hen was dit project niet mogelijk geweest en we zouden dan ook even de tijd willen nemen om deze mensen te bedanken.

Ten eerste willen we onze stagebegeleider Steven De Block bedanken voor de vele uitdagingen waarmee hij ons geconfronteerd heeft. Deze zijn een waardevolle bijdrage voor onze professionele ontwikkeling. Ook willen we hem bedanken voor het mogelijk maken van deze stage en het vertrouwen dat hij in ons gesteld heeft.

Naast onze stagebegeleider heeft ook onze XIOS-promotor Rita Lambrechts veel voor ons betekend. We zijn zeer tevreden met de structurele ondersteuning die we van haar ontvangen hebben. Haar begeleiding in het planningsproces en het opvolgen van onze verslagen was ook zeer welkom.

Onze taallector, Kim Sleurs, willen we bedanken voor het nakijken van onze geschreven teksten en de feedback die ze gegeven heeft.

Glen Van De Sande van Cegeka heeft ons een goede technische begeleiding gegeven en we hebben veel van hem geleerd. In de toekomst zullen we zeker veel van zijn aangeraden best practices blijven toepassen. Hiervoor willen wij hem dan ook enorm bedanken.

Verder willen we graag onze talloze collega's van het Wit-Gele Kruis bedanken voor de fijne samenwerking en ondersteuning.

Als laatste willen we graag onze familie, vrienden en partner bedanken die altijd even begripvol waren als we te kampen hadden met werkstress.

Inhoudsopgave

1 Abstract

Titel:
EXTRANET VOOR ARTSEN

Door:
Sven Salemans
Raf Thewis

Promotoren:
Mr. Steven De Block — Bedrijfspromotor
Mevr. Rita Lambrechts — XIOS-promotor

Er is al een hele tijd een grote vraag bij artsen naar een systeem om gemakkelijker patiënteninformatie te kunnen raadplegen die bijgehouden wordt door het Wit-Gele Kruis. Er zijn hiervoor verschillende methodes, maar geen enkele is gebruiksvriendelijk en laagdrempelig in gebruik. Daarom maken wij Extranet voor Artsen (EVA). Dit extranet is een website waarop artsen met eID kunnen inloggen en gegevens van het Wit-Gele Kruis over hun eigen patiënten kunnen opvragen. Dit extranet moet veilig, laagdrempelig, gebruiksvriendelijk en nuttig voor artsen zijn.

Om het systeem beter te kunnen beveiligen en de backoffice van het Wit-Gele Kruis niet te overbelasten, maken we ook een databanksynchronisatietool. Enkel de data die nodig is op het extranet wordt 's nachts gesynchroniseerd van de backofficedatabank naar onze eigen databank. We maken hiervoor een custom tool omdat er custom behoeften zijn.

Het extranet zelf is geschreven in ASP.NET met het MVC4-framework van Microsoft. We werken dus volgens het mooie "Model-View-Controller" design pattern. Als framework voor "Inversion of Control" (dependency injection) gebruiken we Microsoft Unity.

De programmeertaal die we gebruiken voor de businesslogica van het extranet en voor de custom databanksynchronisatietool is C# met Microsoft .NET Framework 4.5. Op het extranet gebruiken we voor "Object Relation Mapping" met de databank het Entity Framework van Microsoft. Voor mocking bij het unit testen gebruiken we het Rhino Mocks-framework.

De custom databanksynchronisatietool synchroniseert data uit de Oracle 11g databank van de backoffice van het Wit-Gele Kruis naar een Microsoft SQL Server 2008 databank van ons extranet. De tool bestaat uit twee Windows Service applicaties: de ene draait in de backoffice en mag de databank niet te zwaar belasten; de andere draait op de server van het extranet en zorgt voor het up-to-date houden van de data in de databank van ons extranet. We hebben gekozen voor twee afzonderlijke applicaties om het principe van scheiding van verantwoordelijkheden na te streven. Dit heeft onder andere als voordeel dat als één van de servers down is, de andere gewoon verder kan gaan met

zijn werk. Als ze dan beide weer werken kan de pas opnieuw gestarte server zijn werk inhalen, zonder dat de andere extra belast wordt.

We hebben onze eigen Microsoft SQL en IIS Servers opgezet op onze laptops en mogen een testomgeving en acceptatieomgeving gebruiken op een Windows Server 2008 R2.

2 Lijst van afkortingen

CSRF	Cross-site Request Forgery
DI	Dependency Injection
EF	Entity Framework
eID	Elektronische Identiteitskaart
EVA	Extranet voor Artsen
EVD	Elektronisch Verpleegdossier
I/O	Input / Output
IIS	Internet Information Services
IoC	Inversion of Control
IT	Information Technology
LINQ	Language Integrated Query
MS	Microsoft
MVC	Model-View-Controller (design pattern)
MVC4	Model-View-Controller framework van Microsoft, versie 4
OWASP	Open Web Application Security Project
PC	Personal Computer
Regex	Reguliere Expressies / Regular Expressions
SQL	Structured Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
WGK	Wit-Gele Kruis
XIOS	eXpertisecentrum voor Industrie, Onderwijs en Samenleving
XML	Extensible Markup Language
XSS	Cross-site Scripting
SOLID	5 Principles hieronder vermeld:
SRP	Single Responsibility Principle
OCP	Open/Closed Principle
LSP	Liskov Substitution Principle
ISP	Interface Segregation Principle
DIP	Dependency Inversion Principle

3 Beschrijving van het bedrijf

Het Wit-Gele Kruis werd opgericht in 1937 en bestaat dus al een hele tijd. Het biedt een goede verpleegkundige zorg aan mensen die dit nodig hebben. De voornaamste dienst van het WGK is thuisverpleging, maar ook kraamzorg door vroedvrouwen en lactatiekundigen, gezinszorg, enzovoort behoren tot het uitgebreide aanbod.

Een bedrijf zoals het Wit-Gele Kruis met zo veel verantwoordelijkheden ten opzichte van haar klanten heeft natuurlijk ook een goede IT-ondersteuning nodig. De IT-dienst zorgt onder andere voor het elektronisch verpleegdossier (EVD), waarmee verpleegkundigen gemakkelijk en snel aan de gegevens van hun patiënten kunnen. Deze gegevens moeten natuurlijk altijd en overal beschikbaar zijn, daarom dat de IT-dienst alle gegevens over patiënten bijhoudt in gigantische databanken die op verschillende manieren benaderd kunnen worden.

Het Wit-Gele Kruis is ook veel bezig met innovatieve projecten. Met projecten zoals automatisch bloeddrukwaarden van patiënten verzenden naar hun kantoren komen ze in de pers.

De samenwerking van al deze verschillende aspecten maken van het Wit-Gele Kruis een heel interessant bedrijf voor een stage.

4 Nieuwe Technologieën

Hieronder een omschrijving van de nieuwe technologieën die we gebruikt hebben tijdens ons project. Deze technologieën hebben we niet of anders gezien op school.

4.1 MVC4

Om de code van ons extranet overzichtelijk en gestructureerd te houden hebben we gekozen om het principe van "scheiding van verantwoordelijkheden" na te streven met het "Model-View-Controller" design pattern als belangrijkste onderdeel. We hebben het MVC4 framework van Microsoft gebruikt om dit design pattern te implementeren. Meer informatie over onze specifieke implementatie is te vinden in het hoofdstuk "Implementatie".

4.2 Inversion of Control & Dependency Injection (Unity)

Een andere techniek die we gebruikt hebben om verantwoordelijkheden te scheiden, is "Inversion of Control". Onze klassen gebruiken vaak interfaces in plaats van zelf specifieke objecten te instantiëren. Dit doen we om gemakkelijk een andere implementatie van een interface te schrijven om functionaliteit toe te voegen of te wijzigen zonder daarvoor de klasse die het object gebruikt te moeten veranderen. Met het IoC-principe kunnen we dat zelfs nog verder doortrekken en de selectie van de specifieke implementatie van een interface doen in een speciale "bootstrapper"-klasse. Hierdoor wordt het selecteren van geschreven implementaties gecentraliseerd en heeft de rest van het project hier niets mee te maken. In deze "bootstrapper"-klasse worden specifieke implementaties gebonden aan een interface (binding). Dit "injecteren" van specifieke implementaties in interfaces heet Dependency Injection. Als framework voor IoC en DI gebruiken we Microsoft Unity. Meer informatie over onze specifieke implementatie van Unity is te vinden in het hoofdstuk "Implementatie".

4.3 eID

Voor de beveiliging van onze website hebben we gekozen voor eID. Onze motivatie hiervoor komt vanuit onze doelgroep zelf. We zijn samengekomen met een werkgroep

huisdokters en hier tot een consensus gekomen. Verdere info over het keuzeproces is terug te vinden in de paragraaf "Beveiliging" in het onderdeel "Implementatie Extranet".

4.4 Windows Service

De datasynchronisatietoepassing is opgesplitst in twee Windows Services. We hebben deze geschreven in C# met het Microsoft .Net Framework 4.5. Aan de projecten voegden we een `ProjectInstaller`-installatiebestand toe met de `ServiceProcessInstaller` en `ServiceInstaller` klassen. Met deze bestanden kunnen initiële configuraties ingesteld worden zoals de naam van de service en de standaard user account waarmee de service uitgevoerd zal worden. Deze bestanden zorgen ook voor een vlotte installatie van de services.

Het installeren zelf gebeurt met de command prompt utility "installutil.exe" die in de map van het .Net Framework te vinden is onder C:\Windows. Specifieker: het commando `installutil.exe /i <pad van de service>` kan gebruikt worden om de Service te installeren.

4.5 Rhino Mocks

We hebben Rhino Mocks gebruikt als ondersteunend mocking framework voor onze unit tests. Onze keuze ging uit naar Rhino Mocks omdat dit een open source framework is en we hier een opleiding over gekregen hebben van Cegeka. We gebruiken dit framework om al onze functies onafhankelijk van elkaar te unit testen. Dit zorgt ervoor dat we al onze unit tests kunnen uitvoeren zonder afhankelijk te zijn van externe klassen (zoals de databankconnectie).

5 Onderzoeksprobleem

5.1 Achtergrond

Het Wit-Gele Kruis heeft een gigantische databank met gegevens over zijn patiënten. Maandelijks komen verpleegkundigen samen met de huisartsen van deze patiënten om de huidige status te bespreken. Meetwaarden zoals bloeddrukken en temperaturen worden regelmatig doorgestuurd naar de dokter, maar op een onhandige manier die eerder als spam ervaren wordt. Dokters hebben echter nood aan deze informatie en vragen al jaren naar een gemakkelijkere manier om deze gegevens te raadplegen.

Om aan deze nood te beantwoorden, kreeg het Wit-Gele Kruis het idee om een extranet voor artsen te bouwen. Dit werd dan ook onze stageopdracht.

5.2 Doelstellingen

Het extranet voor artsen moet voldoen aan een aantal belangrijke criteria. Ten eerste moet het veilig zijn. We werken hier met gevoelige data, dus deze moet zo goed mogelijk afgeschermd worden voor hackers en andere onbevoegden. Enkel de huisarts van de patiënt mag de data van de patiënt kunnen opvragen.

Ten tweede moet het extranet enorm gebruiksvriendelijk zijn. Artsen moeten er gemakkelijk hun weg op vinden en de nodige informatie moet vlot opvraagbaar zijn. Ook al moet de website veilig zijn, toch moet de drempel laag blijven om het extranet te gebruiken en mag de inlogprocedure niet te omslachtig zijn.

Daarnaast moet het extranet ook aantrekkelijk zijn. Een mooie website is immers leuker om te gebruiken dan een ouderwetse, saaie pagina. De lay-out moet echter wel afgestemd worden met de dienst communicatie.

Een ander belangrijk aspect is de belasting van bestaande systemen. Het Wit-Gele Kruis gebruikt verschillende systemen die met een Oracle-databank werken in de backoffice. Deze Oracle-databank bevat alle gegevens over de patiënten, dus een deel van deze data zal beschikbaar gesteld worden op het extranet. Het extranet rechtstreeks laten

verbinden met deze databank is echter geen goed idee. Het zou de backoffice te veel kunnen belasten, waardoor kritische toepassingen te traag gaan werken. Ook in verband met beveiliging zou er zich een probleem vormen: wanneer hackers het systeem hacken, zou bij een rechtstreekse verbinding meteen de hele backoffice blootgesteld worden. Omwille van deze problemen kiezen we voor een aparte databank voor het extranet. De nodige data wordt gesynchroniseerd met deze databank met een custom databanksynchronisatietool. Deze tool ontwikkelen is ook een onderdeel van ons project.

5.3 Doelpubliek

Het doelpubliek bestaat uit huisartsen. In eventuele vervolgfases van het project kunnen hier ook nog specialisten of andere geneeskundigen bijkomen, maar dit valt buiten onze stage. Ook mensen binnen het Wit-Gele Kruis behoren voor ons tot het doelpubliek, omdat zij regelmatig oordelen of het extranet mooi vorm krijgt en of er nog functies bij moeten.

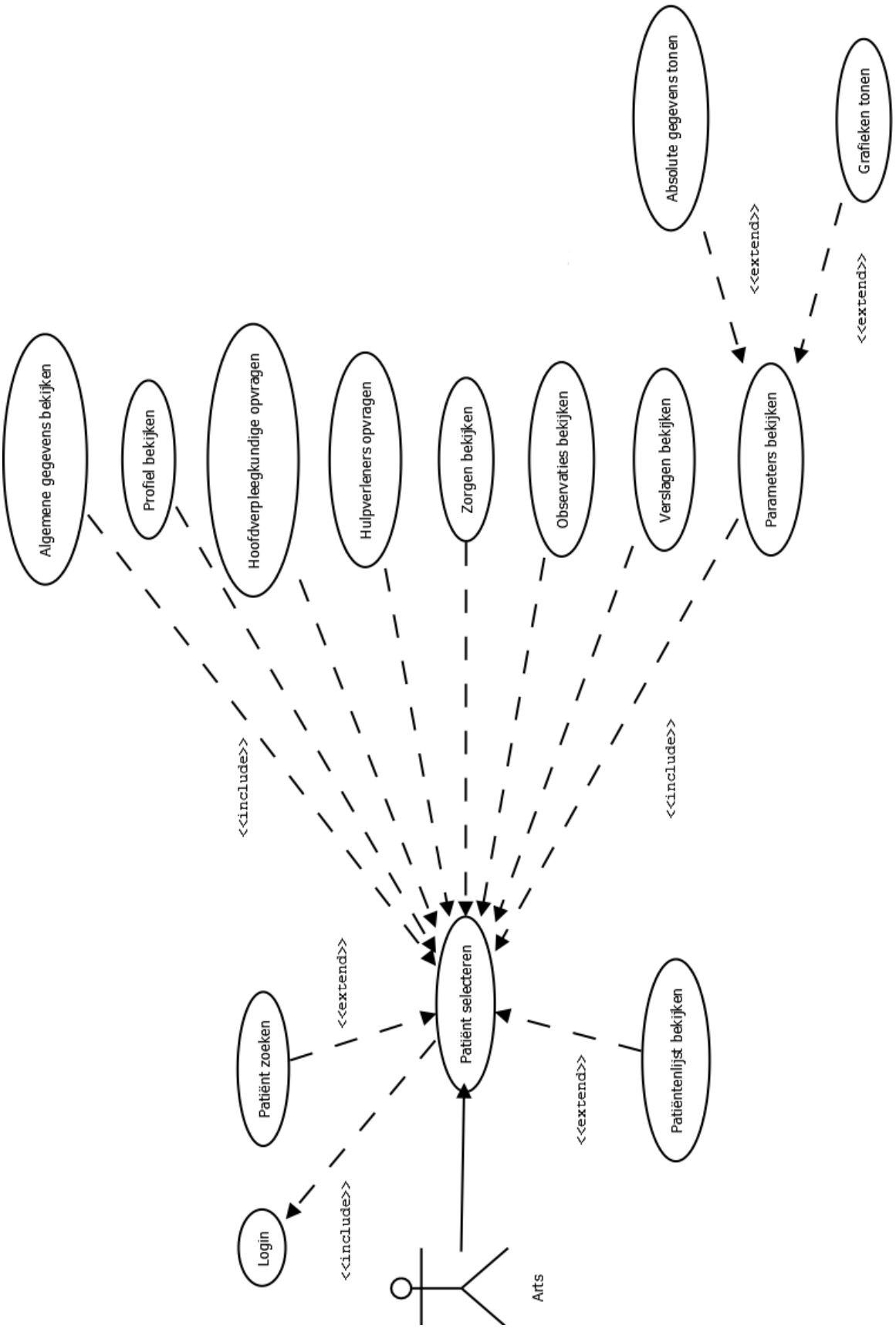
6 Design

6.1 Werkwijze

Aan het begin van onze stage kregen we uitgebreid toegelicht wat onze stageopdracht precies inhield. Het concept van het extranet werd ons duidelijk uitgelegd en er werd met ons besproken hoe we de datasynchronisatie zouden aanpakken. We besloten toen onze stage op te delen in “sprints”, volgens het agile principe. Deze sprints duren ongeveer een week en aan het begin van elke sprint wordt gepland wat er die week moet gebeuren. Deze planning wordt opgedeeld in taken die door middel van post-its visueel worden bevestigd op een “scrumboard”. Telkens een taak gedaan is, wordt deze verplaatst naar een van de categorieën “test” of “done” en wordt een volgende taak gekozen. Aan het einde van elke sprint tonen we het resultaat aan de hand van een demo aan de productowners. Hierdoor krijgen we al snel feedback over ons werk en weten we of we goed bezig zijn.

6.2 Usecasediagram

Na de discussies over functionaliteiten en aanpak met onze collega’s, maakten we een usecasediagram om de verschillende vereisten in kaart te brengen.


Absolute gegevens tonen
Grafieken tonen
<<extend>>
<<extend>>
Algemene gegevens bekijken
Profiel bekijken
Hoofdverpleegkundige opvragen
Hulpverleners opvragen
Zorgen bekijken
Observaties bekijken
Verslagen bekijken
Parameters bekijken
<<include>>
<<include>>
Patiënt zoeken
<<extend>>
Patiënt selecteren
<<extend>>
Patiëntenlijst bekijken
Login
<<include>>
Arts

De focus van de website ligt bij de patiënten. Wanneer een arts inlogt, krijgt hij een lijst met al zijn patiënten die geregistreerd zijn bij het WGK. Bij deze lijst staat een zoekvenster om specifieke patiënten snel te vinden.

Na het selecteren van een patiënt, opent een pagina met de algemene gegevens van deze patiënt. Het menu zorgt voor een gemakkelijke navigatie. In dit menu kan geselecteerd worden welke gegevens er getoond moeten worden. Dit kan gaan van verslagen die verpleegkundigen hebben geschreven over de patiënt, tot grafieken van specifieke meetwaarden zoals bloeddrukken en temperaturen.

6.3 Databank

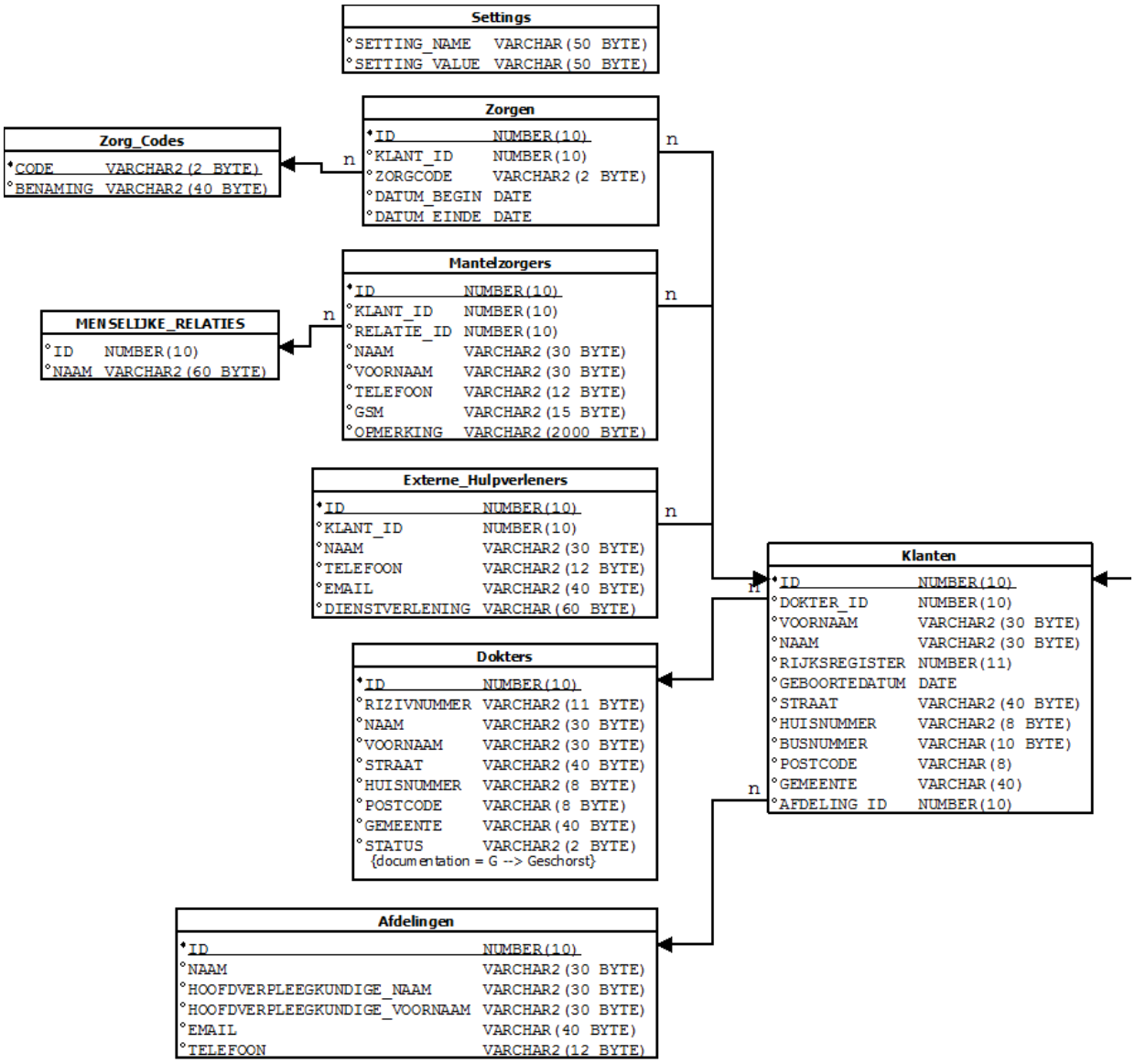
Zoals eerder al vermeld bij de doelstellingen van ons project hebben we gekozen voor een aparte databank voor ons extranet. De voornaamste reden is het minder belasten van de backoffice, waarop kritische applicaties draaien. Iedere nacht wordt de data van de backoffice gesynchroniseerd met de databank van het extranet.

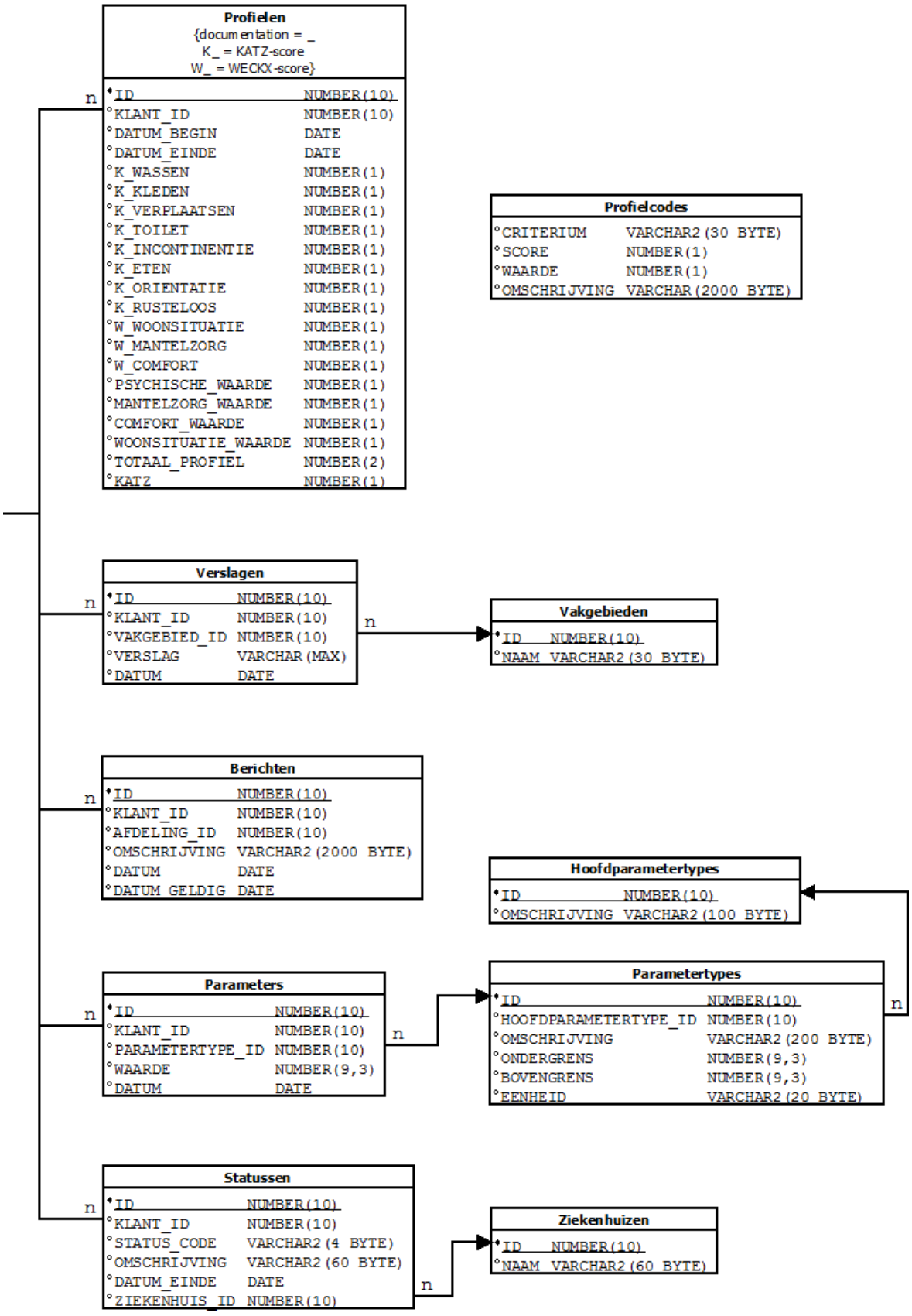
Een andere belangrijke reden is dat we de backoffice beter kunnen afschermen voor aanvallen van buitenaf. Als hackers in het slechtste geval het extranet kunnen hacken of een DDOS-aanval doen, zal dit geen effect hebben op de backoffice en blijft deze gewoon online. Ook zou niet alle informatie uit de backofficedatabank beschikbaar zijn voor de hacker, enkel de gesynchroniseerde data.

De databank van de backoffice is een Oracle Database 11g. Voor het extranet hebben we gekozen voor een MS SQL Server 2008. Deze keuze kan gestaafd worden met twee argumenten. Enerzijds werkt een Microsoft-databank gemakkelijker samen met een website geschreven in de Microsoft-programmeertaal ASP.NET dan een Oracle-databank. Anderzijds geeft het Wit-Gele Kruis de voorkeur aan een MS SQL Server omdat er plannen zijn om in de toekomst volledig over te schakelen op systemen van Microsoft. Deze twee redenen leken ons meer dan voldoende om te kiezen voor een MS SQL Server.

Na het uittekenen van het usecasediagram en het toetsen van onze mock-ups bij de productowners hebben we een UML-schema gemaakt van onze databank. Dit schema

is handig als documentatie bij ons project en gemakkelijk om op terug te vallen als iemand iets over de datastructuren achter het extranet moet weten. Het heeft echter wel een hele tijd geduurd eer we de structuur van de backofficedatabank onder de knie kregen. Deze databank gaat al lang mee en er zijn dan ook veel tabellen die niet meer gebruikt worden. Daardoor kan het geheel soms nogal verwarrend overkomen voor mensen die er niet mee vertrouwd zijn, zoals wij. Na het uitprinten van schema's van de databank en het uitdokteren van de structuur (we moesten immers weten welke data er nu precies beschikbaar was), konden we eindelijk onze eigen databankstructuur maken. Hieronder wordt het UML-diagram weergegeven. Voor de leesbaarheid is dit verspreid over twee pagina's.


Settings
°SETTING_NAME VARCHAR(50 BYTE)
°SETTING_VALUE VARCHAR(50 BYTE)
Zorgen
♦ID NUMBER(10)
°KLANT_ID NUMBER(10)
°ZORGCODE VARCHAR2(2 BYTE)
°DATUM_BEGIN DATE
°DATUM_EINDE DATE
Zorg_Codes
♦CODE VARCHAR2(2 BYTE)
°BENAMING VARCHAR2(40 BYTE)
n
n
Mantelzorgers
♦ID NUMBER(10)
°KLANT_ID NUMBER(10)
°RELATIE_ID NUMBER(10)
°NAAM VARCHAR2(30 BYTE)
°VOORNAAM VARCHAR2(30 BYTE)
°TELEFOON VARCHAR2(12 BYTE)
°GSM VARCHAR2(15 BYTE)
°OPMERKING VARCHAR2(2000 BYTE)
MENSELIJKE_RELATIES
°ID NUMBER(10)
°NAAM VARCHAR2(60 BYTE)
n
n
Externe_Hulpverleners
♦ID NUMBER(10)
°KLANT_ID NUMBER(10)
°NAAM VARCHAR2(30 BYTE)
°TELEFOON VARCHAR2(12 BYTE)
°EMAIL VARCHAR2(40 BYTE)
°DIENSTVERLENING VARCHAR(60 BYTE)
n
Klanten
♦ID NUMBER(10)
°DOKTER_ID NUMBER(10)
°VOORNAAM VARCHAR2(30 BYTE)
°NAAM VARCHAR2(30 BYTE)
°RIJKSREGISTER NUMBER(11)
°GEBOORTEDATUM DATE
°STRAAT VARCHAR2(40 BYTE)
°HUISNUMMER VARCHAR2(8 BYTE)
°BUSNUMMER VARCHAR(10 BYTE)
°POSTCODE VARCHAR(8)
°GEMEENTE VARCHAR(40)
°AFDELING_ID NUMBER(10)
n
Dokters
♦ID NUMBER(10)
°RIZIVNUMMER VARCHAR2(11 BYTE)
°NAAM VARCHAR2(30 BYTE)
°VOORNAAM VARCHAR2(30 BYTE)
°STRAAT VARCHAR2(40 BYTE)
°HUISNUMMER VARCHAR2(8 BYTE)
°POSTCODE VARCHAR(8 BYTE)
°GEMEENTE VARCHAR(40 BYTE)
°STATUS VARCHAR2(2 BYTE)
{documentation = G --> Geschorst}
n
Afdelingen
♦ID NUMBER(10)
°NAAM VARCHAR2(30 BYTE)
°HOOFDVERPLEEGKUNDIGE_NAAM VARCHAR2(30 BYTE)
°HOOFDVERPLEEGKUNDIGE_VOORNAAM VARCHAR2(30 BYTE)
°EMAIL VARCHAR(40 BYTE)
°TELEFOON VARCHAR2(12 BYTE)


Profielen
{documentation = _
K_ = KATZ-score
W_ = WECKX-score}
ID NUMBER(10)
KLANT_ID NUMBER(10)
DATUM_BEGIN DATE
DATUM_EINDE DATE
K_WASSEN NUMBER(1)
K_KLEDEN NUMBER(1)
K_VERPLAATSEN NUMBER(1)
K_TOILET NUMBER(1)
K_INCONTINENTIE NUMBER(1)
K_ETEN NUMBER(1)
K_ORIENTATIE NUMBER(1)
K_RUSTELOOS NUMBER(1)
W_WOONSITUATIE NUMBER(1)
W_MANTELZORG NUMBER(1)
W_COMFORT NUMBER(1)
PSYCHISCHE_WAARDE NUMBER(1)
MANTELZORG_WAARDE NUMBER(1)
COMFORT_WAARDE NUMBER(1)
WOONSITUATIE_WAARDE NUMBER(1)
TOTAAL_PROFIEL NUMBER(2)
KATZ NUMBER(1)
Profielcodes
CRITERIUM VARCHAR2(30 BYTE)
SCORE NUMBER(1)
WAARDE NUMBER(1)
OMSCHRIJVING VARCHAR(2000 BYTE)
Verslagen
ID NUMBER(10)
KLANT_ID NUMBER(10)
VAKGEBIED_ID NUMBER(10)
VERSLAG VARCHAR(MAX)
DATUM DATE
Vakgebieden
ID NUMBER(10)
NAAM VARCHAR2(30 BYTE)
Berichten
ID NUMBER(10)
KLANT_ID NUMBER(10)
AFDELING_ID NUMBER(10)
OMSCHRIJVING VARCHAR2(2000 BYTE)
DATUM DATE
DATUM_GELDIG DATE
Hoofdparametertypes
ID NUMBER(10)
OMSCHRIJVING VARCHAR2(100 BYTE)
Parameters
ID NUMBER(10)
KLANT_ID NUMBER(10)
PARAMETERTYPE_ID NUMBER(10)
WAARDE NUMBER(9,3)
DATUM DATE
Parametertypes
ID NUMBER(10)
HOOFDPARAMETERTYPE_ID NUMBER(10)
OMSCHRIJVING VARCHAR2(200 BYTE)
ONDERGRENS NUMBER(9,3)
BOVENGRENS NUMBER(9,3)
EENHEID VARCHAR2(20 BYTE)
Statussen
ID NUMBER(10)
KLANT_ID NUMBER(10)
STATUS_CODE VARCHAR2(4 BYTE)
OMSCHRIJVING VARCHAR2(60 BYTE)
DATUM_EINDE DATE
ZIEKENHUIS_ID NUMBER(10)
Ziekenhuizen
ID NUMBER(10)
NAAM VARCHAR2(60 BYTE)
n
n
n
n
n
n
n
n
n
n

6.4 Databanksynchronisatietool

Om de synchronisatie tussen de Oracle en SQL Server databanken vlot te laten verlopen, hebben we als onderdeel van ons stageproject een custom tool geschreven. Tijdens discussies met onze collega's en een consultant van Cegeka, hebben we besloten deze tool op te splitsen in twee Windows Services. De ene service exporteert alle gewijzigde data in de backoffice sinds de laatste synchronisatie naar XML-bestanden. De andere tool importeert deze XML-bestanden in de databank van het extranet.

Het opsplitsen in twee afzonderlijke services op verschillende servers heeft als voordeel dat als de ene server crasht, de andere gewoon verder kan gaan. Als de gecrashte server dan weer opgestart wordt, moet deze de achterstand wel inhalen, maar de andere server wordt hiervoor niet extra belast. We hebben de services zo opgevat dat ze volledig afzonderlijk kunnen werken. Hierover meer in het hoofdstuk "Implementatie".

6.5 Werkgroep artsen

Tijdens onze stage, op 8 mei 2013 om precies te zijn, hebben we een werkgroep met vijf huisartsen gehouden. Tijdens deze meeting hebben we eerst het concept van het extranet voorgesteld en aan de hand van een presentatie en korte demo een overzicht van de functies gegeven.

Na deze intro gaven we de artsen de kans om feedback te geven. Hier werd vlijtig gebruik van gemaakt. De artsen hadden meteen veel voorstellen voor extra functies of aanpassingen aan bestaande onderdelen. Na goed te communiceren over wat precies bedoeld werd en al deze opmerkingen te noteren, hebben we beslist wat tijdens onze stageperiode en met de huidige informatie in de backoffice van het WGK nog mogelijk was. De andere functies staan op de planning voor fase twee van het extranet, wat buiten onze stage valt.

7 Implementatie Extranet

Zoals al eerder vermeld hebben wij ons extranet ontwikkeld met behulp van MVC4. Elk onderdeel van dit framework hebben wij nog eens extra op een specifieke manier uitgewerkt. Hieronder volgt ons keuzeproces en de uitleg van de gekozen technieken.

7.1 Algemeen Proces bij opvragen pagina

Hier volgt een concreet overzicht van de structuur die doorlopen wordt bij de klik van een gebruiker die zo een action van een Controller oproept.

1 De gebruiker klikt op de link om de lijst van Patiënten weer te geven.
2 De `PatientController` wordt aangeroepen, Unity injecteert een `PatientRepository` en een `ViewModelFactory`-object in de parameters `IPatiëntRepository` en `IViewModelFacotory`.
3 De juiste methode van de `PatientController` wordt aangeroepen, in dit geval Index(int index) (De parameter index wordt gebruikt door de paginering om de juiste pagina weer te geven)
4 De methode `GetPatientsOverview(index, user_id)` van de `PatientRepository` wordt aangeroepen. Index is in dit geval 0 aangezien we de eerste pagina opvragen; `user_id` is het id van de dokter die ingelogd is. De `Repository` geeft een lijst van `IPatientModels` terug aan de `Controller`; de lijst is maximaal zo groot als het aantal patiënten dat gaat worden weergegeven.
5 De methode `GetCountPatientOverview(user_id)` van de `PatientRepository` wordt aangeroepen. Dit geeft het aantal patiënten van de huidig ingelogde dokter terug aan de controller.
6 In de `Controller` wordt de lijst van `IPatientModels` omgezet naar een lijst van `IPatientViewModels` met behulp van de `ViewModelFactory`.
7 De Controller maakt een `CustomPagedList` aan met als parameters de lijst van `IPatientViewModels`, het aantal patiënten van de ingelogde dokter en de huidige pagina.
8 De Controller geeft dit `CustomPagedList` object door aan de View.

9 In de View wordt dit object gepresenteerd op de daar gedefinieerde manier. De paginering is mogelijk dankzij de `CustomPagedList` (meer informatie hierover in de paragraaf "Paginering").

10 Het proces is afgerond; de gebruiker kan nu een andere action activeren.

7.2 Lay-out en Design

Voor de lay-out hebben we besloten een template te gebruiken. Deze template hebben we toegepast met behulp van de shared _Layout-view die MVC4 voorziet. Uiteraard hebben we deze template aangepast aan onze specifieke noden, zodat het perfect kon dienen voor de weergave van het extranet.

7.3 SOLID-designprincipes

We hebben gekozen voor het ontwikkelen van onze website met de SOLID-designprincipes. Hieronder volgt een korte beschrijving van deze principes. De praktische toepassing hiervan zal toegelicht worden tijdens de respectievelijke bespreking van de implementaties.

7.3.1 Single responsibility principle (SRP)

De S van SOLID staat voor Single responsibility principle. Dit houdt in dat elke klasse slechts één afgebakende verantwoordelijkheid mag hebben. Dit zorgt onder andere voor duidelijkere klassenamen en maakt de code meer gestructureerd.

7.3.2 Open/closed principle (OCP)

De O van SOLID staat voor het Open/Closed principle. Dit principe houdt in dat elke entity moet open staan voor uitbreiding, maar gesloten is voor aanpassingen. Dit is de basis van het "Keep all object variables private"-principe. Enkel bugfixes mogen toegevoegd worden. Als nadien extra functionaliteit nodig is, kan deze toegevoegd worden door een andere klasse te schrijven. Dankzij dit principe moet andere code die gebruikmaakt van deze klasse niet herschreven worden om de nieuwe klasse te gebruiken.

7.3.3 Liskov substitution principle (LSP)

De L van SOLID staat voor het Liskov substitution principle. Dit houdt in dat objecten van een programma vervangbaar moeten zijn door instanties van hun subtypes. De programmalogica dient nog op dezelfde manier afgehandeld te worden als dit gebeurt.

7.3.4 Interface-segregation principle (ISP)

De I van SOLID staat voor het Interface-segregation principle. Dit houdt in dat een interface enkel de methoden mag bevatten die verplicht worden aan de objecten die overerven. Indien een object van een interface zou overerven en niet elke methode hiervan nodig zou hebben, dient er een nieuwe interface gemaakt te worden zonder deze methode erbij (client-specific).

7.3.5 Dependency inversion principle (DIP)

De D van SOLID staat voor het laatste designprincipe: Dependency inversion. Dit houdt in dat een klasse hoort af te hangen van een abstractie van een object, niet van een concreet object. Een klasse mag dus niet afhankelijk zijn van (of 'vastgelast' zijn aan) een andere klasse.

7.3.6 Waarom SOLID-designprincipes?

We hebben gekozen om deze principes strikt te volgen bij de ontwikkeling van ons project om de uitbreidbaarheid maximaal te bevorderen. Ook is het gemakkelijk om bijvoorbeeld af te stappen van het Entity Framework aangezien we modulair werken (volgens LSP). Over het algemeen zijn deze principes ook Good Practices. De leesbaarheid en overzichtelijkheid van de code wordt hierdoor enkel beter. Het is zeer bevorderlijk voor de mensen die na ons het project gaan overnemen en opvolgen. Dankzij deze principes gaan zij gemakkelijk wijzigingen kunnen aanbrengen.

7.3.7 Implementatie van de SOLID-designprincipes

SRP: Geen enkele klasse van ons project heeft meer dan één verantwoordelijkheid. Indien dit het geval was, hebben wij telkens de klasse opgesplitst zodat dit principe gevolgd werd.

OCP: Dit hebben we gerespecteerd door `ViewModels` te maken van de `Models`. Zo kan er in de `ViewModels` gedefinieerd worden hoe de data gepresenteerd dient te worden op de `View`. Het Model dient dan niet aangepast te worden als de gewenste weergave zou veranderen.

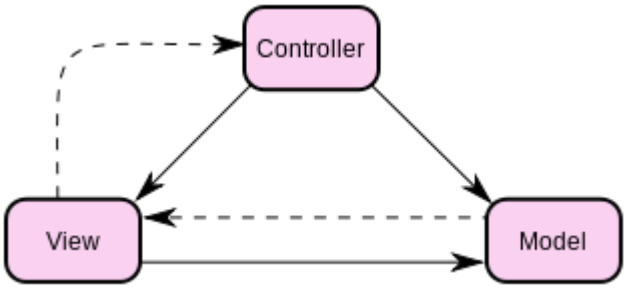
LSP: Dit designprincipe zorgt ervoor dat de mensen die ons project gaan opvolgen, na afloop van onze stage, gemakkelijk globale aanpassingen kunnen doen aan ons project. Zo zit de volledige logica van het Entity Framework dat wij gebruiken achter een Interface. Wanneer een andere databankcommunicatiemethode gebruikt moet worden, is het hierdoor mogelijk gewoon deze interface te implementeren met het nieuwe framework. Zo is onze applicatie modulair opgebouwd volgens het LSP.

ISP: Wij hebben voor elk `Model`, `ViewModel` en voor elke `Repository` een interface gemaakt waar enkel de methodes in staan die noodzakelijk zijn voor het functioneren van de programmalogica.

DIP: We hebben gebruikgemaakt van Unity (Dependency Injection) om dit designprincipe te respecteren. Door gebruik te maken van Unity hangt geen enkele klasse vast aan een concrete implementatie van een ander object. Meer uitleg over Unity vindt u onder het puntje “Unity”.

7.4 MVC4 Framework

Onze website is ontwikkeld binnen het MVC4 Framework van Microsoft. Dit houdt in dat alles opgesplitst wordt in 3 grote delen: `Models`, `Views` en `Controllers`.



Model: Dit is enkel de representatie van de informatie waarmee het programma werkt. In ons project zijn onze `Models` de representaties van de data van onze databank. Elk `Model` heeft een property voor elk veld van de respectievelijke tabel in de databank. Op deze manier mappen we de databankgegevens naar objecten (opgehaald uit de databank door de database layer).

View: De `View` zorgt voor de weergave van het model en andere data. Er gebeurt hier geen verwerking van gegevens. In ons project zijn de `Views` de pagina's. Elke `Controller` heeft een of meerdere `Views` die een `Model` met data krijgen doorgegeven van de `Controllers`.

Controller: De `Controller` reageert op input van users. Op basis van welke actie er aangeroepen wordt, zal de Controller aan de database layer de nodige data vragen, deze in een `ViewModel` steken (via een `ViewModelFactory`) en dit `ViewModel` dan doorgeven aan de `View`. De gebruiker kan dan in deze `View` die hij gepresenteerd krijgt terug een actie uitvoeren waarna dit proces zich herhaalt.

Database Layer: Deze laag zorgt voor de communicatie met de databank. De gevraagde data wordt opgehaald, in `Models` gestoken en vervolgens doorgegeven aan de `Controller`.

In de volgende paragrafen zullen we dieper ingaan op onze praktische toepassing van het framework. In de bijlagen is een codevoorbeeld te vinden van onze implementatie voor 1 pagina van het extranet, de pagina "Algemene Gegevens".

7.4.1 Models

Onze `Models` zijn gebaseerd op het Entity Framework. Dit houdt in dat elk `Model` een representatie is van de benodigde data uit een tabel van de databank. Met behulp van het Entity Framework kunnen we dan manueel de mapping toepassen tussen het model en het databankobject. Dit gebeurt via een `DatabaseContext` object, wat verder toegelicht zal worden in de Database Layer.

We hebben ervoor gekozen een extra laag tussen een `Model` en een `View` te voorzien: `ViewModels`. Elk `ViewModel` krijgt als parameter voor de constructor een `Model` mee. Het `ViewModel` zal instaan voor het formatteren van de data van het `Model` zodat het op een correcte manier zal worden weergegeven in de `View`. Zo kan een `Model` op verschillende manieren weergegeven worden in verschillende `Views` door meerdere `ViewModels` te maken van een `Model` indien nodig.

De `ViewModels` worden "aangemaakt" in de `Controller`. Indien we hier rechtstreeks een `ViewModel` zouden aanmaken, zouden we het DIP principe schenden. Daarom hebben we besloten om een `Factory` aan te maken die de creatie van alle `ViewModels` op zich neemt. Deze `Factory` wordt geïnjecteerd in de `Controller` met behulp van Unity. Uiteraard wordt er gewerkt met een interface van de `Factory`, zodat deze modulair kan vervangen worden.

7.4.2 Views

Onze `Views` bestaan uit webpagina's. Elke pagina bestaat uit de dataspecifieke `View`, die de masterpage-lay-out (shared `_layout View`) implementeert. Het resultaat is dat de volledige website een consistente look en feel heeft. Ook maken we gebruik van `Partial Views`, zodat ook de Views die herhaald worden overal consistent worden weergegeven (voor bijvoorbeeld het menu). De dataspecifieke `Views` geven dan de data van de meegegeven `ViewModels` weer op een in de `ViewModels` gedefinieerde manier.

7.4.3 Controllers

We hebben een `Controller` voorzien voor elk `Model`. Dit om het overzicht te bewaren. In elke `Controller` is er 1 method per `View` die als enige verantwoordelijkheid heeft een `ViewModel` door te geven aan de `View`. Elke `Controller` ontvangt 2 parameters bij de aanroep: een `IRepository` van het nodige `Model` en een `IFactory` van de `ViewModels`. Deze parameters worden geïnjecteerd door Unity. In een typische methode van de `Controller` worden de volgende zaken gedaan:

1. Vraag een lijst op van `Models` die nodig zijn (bv. een lijst van patiënten)
2. Maak een lijst van `ViewModels` van deze `Models` met behulp van de `Factory`
3. Geef de lijst van `ViewModels` door aan de `View`

Indien er ooit besloten zou worden om met een andere `Repository` te werken, dient er gewoon een nieuwe implementatie van de `IRepository` aangemaakt te worden die dezelfde methodes implementeert. De `Controller` zal gewoon werken. De hele bedoeling is dat de `Controller` niet op de hoogte is van HOE hij de data krijgt, zolang hij de benodigde data krijgt. Op deze manier kan de volledige Database Layer modulair vervangen worden.

7.4.4 Database Layer

Voor de communicatie met de databank hebben we gekozen voor het Entity Framework (EF). We hebben hiervoor gekozen omdat we hiermee al redelijk wat ervaring hadden van onze lessen op XIOS. Ook is dit een oplossing van Microsoft zelf en aangezien alle technologieën die we gebruiken van Microsoft zijn, leek dit ons de beste keuze. Een andere optie was het ontwikkelen van een web service. Dit hebben we echter niet gedaan omdat het niet de bedoeling was de data van EVA op een ander platform te kunnen tonen. Dit zou dus meer werk betekenen, zonder toegevoegde waarde.

Het Entity Framework kan op 3 manieren toegepast worden:

- Code First: Eerst de code schrijven (geen code laten genereren door EF).

- Model First: Eerst de `Models` maken en de databank genereren via EF met de gemaakte `Models`.
- Database First: Eerst de databank maken (of een bestaande databank gebruiken) en dan uit deze databank de `Models` genereren (dus code hiervoor laten genereren door EF).

Wij hebben gekozen voor een custom Code First Strategie, ondanks dat we eerst de databank gemaakt hebben. Onze motivatie hiervoor is dat wij de volledige controle over code en databank wilden behouden. De voordelen van onze strategie zijn als volgt:

- We verzorgen onze eigen mapping van model naar databank: dit zorgt ervoor dat we 'clean' `Models` kunnen gebruiken die ook hergebruikt kunnen worden indien een ander framework EF zou vervangen.
- We kunnen de `DBContext` afsplitsen van de `Models`. Zo hebben we nu door middel van een `ModelBuilder` alle mappings gedaan in de klasse `EvaDBContext`. Verder bevat deze klasse alle `DBContext` objecten van EF naar de databank.
- We hebben volledige controle over onze code.
- We hebben volledige controle over de structuur van de databank.
- We gebruiken alle voordelen van EF, zonder de automatische generatie van code. Dit zorgt ervoor dat we een optimaal overzicht over onze code behouden.

Concreet hebben we ons project als volgt uitgewerkt:

1 Databank maken met een zelfgemaakt SQL-script.
2 Manueel aanmaken van 'clean' models op basis van de databank.
3 Aanmaken klasse `EvaDBContext`.
4 Verzorgen mapping van `Models` naar databank via de `DbModelBuilder` van EF in de klasse `EvaDBContext`. Ook bevat deze klasse de `DbSet<Model>` objecten die aangesproken kunnen worden in de `Repositories`.
5 Voor elk model is er een `ModelRepository` voorzien (met bijhorende interface, gebruikt voor dependency injection met Unity).
6 Elke `Repository` krijgt het `EvaDBContext` object mee

7 Elke `Repository` voorziet de query logica voor het bijhorende model
8 De correcte `Repository` wordt met behulp van Unity geïnjecteerd in de constructor van de `Controllers`, zodat in de controller de juiste data opgehaald kan worden via deze `Repositories`.

Door deze manier van werken toe te passen, hebben we een zeer flexibel project gerealiseerd. Zo is het perfect mogelijk om een ander framework dan EF te gebruiken zonder de `Models` te moeten aanpassen. Er dienen enkel klassen aangemaakt te worden die overerven van de `IRepository` per `Model` en deze moeten geïnjecteerd worden via Unity in plaats van de EF `Repositories`. Dit is de hoofdreden waarom we voor deze manier van coderen gekozen hebben.

Als query language voor de Database Layer hebben we gekozen om LINQ te gebruiken. De hoofdreden hiervoor is dat het de nodige bescherming biedt tegen SQL injection en dat dit goed samenwerkt met EF.

Voor onze ingewikkelde query's hebben we besloten om views te voorzien in de database. Dit vereenvoudigt de LINQ-code en maakt deze overzichtelijker en leesbaarder voor de mensen die het extranet verder gaan ontwikkelen.

7.5 Paginering

Aangezien een dokter soms tientallen patiënten kan hebben bij het WGK, moesten we een manier uitwerken om paginering te voorzien op onze website. Het is namelijk niet wenselijk om een lijst van 100 patiënten op één pagina weer te geven.

De beschikbare libraries hadden echter telkens een performanceprobleem: elke keer dat een pagina wordt opgevraagd, gaat de library de volledige lijst opvragen en dan de records van de gevraagde pagina selecteren.

Dit was voor ons onaanvaardbaar, dus hebben we er zelf een oplossing voor ontworpen. We hebben zelf een klasse aangemaakt die overerft van de `PagedList`-klasse van MVC. Vervolgens hebben we de paginering verplaatst naar de serverside (de query selecteert enkel de nodige records). Dankzij onze custom implementatie van de

`PagedList` konden we nu veel performanter data pagineren. Het probleem en de oplossing was als volgt:

Probleemstelling: De `PagedList` haalt voor elke pagina opnieuw de volledige data uit de databank. Dit is niet performant. Bijgevolg pasten wij onze query aan zodat enkel de nodige records opgevraagd werden uit de databank. Het probleem dat zich nu stelde was dat de paginering van de library hierdoor niet meer werkte. De oorzaak hiervan was dat deze niet op de hoogte was van het aantal items dat gepagineerd diende te worden, doordat niet meer alle records geselecteerd werden.

Oplossing: We voeren nu een tweede (zeer lichte) query uit op de databank die het aantal records dat gepagineerd moet worden selecteert. Vervolgens hebben we een eigen klasse `CustomPagedList` gemaakt die overerft van `PagedList`, zodat we manueel het aantal te pagineren records konden instellen. Het probleem dat zich stelde was volledig opgelost dankzij deze aanpassing en het resultaat is een boost in performance voor ons project.

7.6 Unity en het DIP-principe

Zoals eerder vermeld, hebben we ervoor gekozen om de SOLID-designprincipes toe te passen in ons project. Bij het maken van de controllers kwamen we echter een probleem tegen dat op het eerste zicht moeilijk op te lossen was. Het probleem dat zich stelde was het volgende:

Initieel was onze `Controller` vastgelast aan de `Repository` met de databaselogic. Er werd dus een `Repository`-object aangemaakt in de `Controller`, om vervolgens dit object te gebruiken om te communiceren met de databank. Dit was echter niet volgens het DIP-principe. Idealiter zouden we een interface kunnen gebruiken van de `Repository`, zodat de `Controller` niet hoeft te weten welke instantie van deze interface gebruikt wordt.

Als oplossing voor dit probleem hebben we voor Unity gekozen. Unity zorgt ervoor dat er, tijdens runtime, wanneer de `Controller` opgeroepen wordt, automatisch een instantie van het `Repository`-object wordt toegekend aan de `IRepository` die als

parameter wordt meegegeven aan de `Controller`. Dit wordt geconfigureerd in Bootstrapper.cs. Indien later besloten zou worden om een andere `Repository` te gebruiken kan dit hier ook gemakkelijk worden aangepast.

Voor de creatie van onze `ViewModels` hebben we hetzelfde principe toegepast. Het was niet wenselijk (en in tegenspraak met het DIP-principe) dat er een instantie van een `ViewModel` aangemaakt werd in de `Controller`. Daarom hebben we ervoor gekozen een `ViewModelFactory` aan te maken, die we dan ook via Unity gaan injecteren in onze controllers. Op deze manier is de `Controller` niet afhankelijk van de `ViewModelklasse`. Dit draagt bij tot het modulair design en zorgt ervoor dat de SOLID-designprincipes gerespecteerd worden.

7.7 Beveiliging

Beveiliging is een enorm belangrijk aspect bij een project van dit kaliber. Een "Extranet voor Artsen" is een website, beschikbaar via het internet, die gevoelige informatie over patiënten van het WGK bevat. Daarom dat we ook heel wat tijd hebben gestoken in research naar verschillende beveiligingstechnieken. Beveiliging van een website moet gebeuren op verschillende niveaus. Hieronder gaan we verder in op ieder van die niveaus.

7.7.1 Entrypoint

Wanneer een dokter zijn patiënten wil raadplegen, moet hij zich eerst identificeren. Hier hadden we verschillende mogelijkheden voor. We hebben elk van deze mogelijkheden onderzocht en de voor- en nadelen tegen elkaar afgewogen. Uiteindelijk zijn we samen met onze collega's van het WGK en de huisartsen die aanwezig waren op de artsen-werkgroep tot een besluit gekomen.

- Een eerste mogelijkheid was een gewone gebruikersnaam en wachtwoord. Dit is misschien de makkelijkste en meest gebruikelijke manier om in te loggen op een website, maar voldoet helemaal niet aan onze veiligheidsvereisten. Deze mogelijkheid werd dan ook al snel verworpen.

- Een tweede mogelijkheid was een specifieke authenticator voor het extranet. Dit bleek na onderzoek een van de veiligste methoden en kwam dus in aanmerking om de uiteindelijke keuze te worden. Er zijn echter ook heel wat nadelen aan deze methode verbonden. Dokters hebben vaak al een hele reeks authenticators om op allerlei erg beveiligde websites in te loggen en zijn niet happig op nog een extra authenticator. Daarnaast komt nog de extra tijd en moeite die nodig is om in te loggen. De huisartsen op de artsenwerkgroep waarschuwden ons dan ook dat het extranet met een authenticator heel wat minder aantrekkelijk zou worden om te gebruiken.
- Een derde mogelijkheid was het gebruik van eHealth. Dit is een beveiligings-methode ontworpen door de regering. Ze is enorm veilig en zeker aan te raden, maar ook enorm omslachtig. eHealth is niet gekozen om in fase 1 van het extranet als loginmethode te worden gebruikt omwille van de omslachtigheid, maar staat zeker op de planning om toegevoegd te worden in een volgende fase, wanneer dokters meer vertrouwd zijn met het platform.
- De vierde mogelijkheid was eID. Een ander systeem ontworpen door de regering. Inloggen met een elektronische identiteitskaart is erg veilig en dankzij de kaartlezer ook nog eens eenvoudig. De meeste dokters hebben al zo'n kaartlezer aangesloten op hun PC, dus de drempel voor eID wordt weer wat lager. Daarnaast zijn dokters meteen te identificeren via hun identiteitskaart, in tegenstelling tot de eerste twee mogelijkheden, waarbij nog een koppeling gemaakt moest worden tussen gebruikersnaam of authenticator en dokter.

Uiteindelijk hebben we met onze collega's en de artsen op de artsenwerkgroep samen besloten dat eID het aantrekkelijkst was voor fase 1 van het extranet. In een latere fase, die buiten onze stage valt, zal zeker het eHealth-platform nog toegevoegd worden.

Aangezien beveiliging een gevoelig onderwerp is, heeft onze stageplaats ons verzocht om niet in detail te treden in verband met de implementatie van de eID-beveiliging van het extranet.

7.7.2 Coding best practices

Voor de beveiliging van onze website hebben we voornamelijk de guidelines van OWASP gevolgd. In eerste instantie hebben we de top 10 van meest gemaakte beveiligingsfouten overlopen. Deze hebben we allemaal zo goed mogelijk opgevangen in onze website. Het grootste gevaar voor een website is injectie. Dit hebben we beveiligd door overal LINQ te gebruiken waar de databank aangesproken wordt. Ook hebben we elk invoerveld van reguliere expressies (regex) voorzien, waardoor de invoer goed gefilterd wordt. Dit vormt een extra beveiligingslaag tegen injectie.

Verder hebben we onze sessies afgeschermd voor de gebruikers. Ten eerste werkt onze website met SSL-encryptie, wat het onderscheppen van netwerkverkeer van en naar onze website tegenhoudt. Ook wordt er geen gebruikersinformatie opgeslagen op de client; alles wordt bijgehouden in een sessie op de server. De cookies op de client bevatten enkel een geëncrypteerd sessie-id. Daarnaast worden onze sessies beheerd door Forms Authentication, een door Microsoft ingebouwd framework in .NET. We hebben ook uitvoerig onderzoek gedaan naar lekken die eventueel zouden bestaan in dit framework. Uit dit onderzoek bleek dat het zeer belangrijk is dat er een custom error page ingesteld wordt. Een hacker zou namelijk dankzij de feedback die hij krijgt van de errorpagina's van Forms Authentication deze beveiliging kunnen kraken. We hebben dan ook een custom error page gemaakt om dit tegen te gaan.

Een ander groot gevaar voor onze website is dat men een CSRF-aanval zou kunnen doen (Cross-site Request Forgery). Dit zou inhouden dat hackers het request naar de overheid zouden onderscheppen en zelf een eigen antwoord terugsturen naar onze website. Dit zou kunnen als ze een soort interface plaatsen tussen onze eID-authenticatie en onze website, waardoor ze hun identiteit zouden kunnen vervalsen. Dit houden we tegen door het instellen van een "realm" en het gebruik van een custom claims-validator. Deze validator zorgt ervoor dat enkel requests en responses van de overheid van en naar de website gestuurd kunnen worden.

Wat ook belangrijk was is de bescherming tegen XSS (Cross-site Scripting). XSS is het "embedden" van een script in een cookie, URL, request parameter, enzovoort. Dit script

wordt dan uitgevoerd op de client en kan voor veel beveiligingslekken zorgen. Cookie-XSS vangen we op door onze cookie http-only te maken. Dit zorgt ervoor dat er enkel html in de cookie mag staan. URL- en request-XSS vangen we op door elke actie die niet verwerkt kan worden (na onze filters uit te voeren) te redirecten naar een custom error-page. Ook de error achter de custom page wordt gescrambled zodat deze zelfs met een package-sniffer onleesbaar blijft.

7.7.3 Infrastructuur

Een derde beveiligingslaag is de infrastructuur. Servers en netwerken moeten goed beveiligd worden en aanvallen van hackers kunnen weren. Firewalls moeten juist ingesteld worden en alles moet grondig afgeschermd worden van de buitenwereld. De beveiliging van de infrastructuur behoort echter niet tot ons stageproject, dus hier zullen we niet verder op ingaan.

8 Implementatie Databanksynchronisatietool

Onze synchronisatietool bestaat uit twee Windows Services. Waarom we hebben gekozen voor twee afzonderlijke services, werd al vermeld in het hoofdstuk “Design”. Hoe deze precies afzonderlijk werken en welke stijl van coderen we hebben gebruikt, wordt besproken in dit hoofdstuk. Gemakkelijkheidshalve zal verder verwezen worden naar deze twee services met “EVA” voor de kant van ons Extranet voor Artsen, en “backoffice” voor de kant van de backoffice van het WGK. Naar de databank-synchronisatietool zal vanaf nu verwezen worden met “sync tool”.

8.1 Designprincipes

Zoals bij het extranet, hebben we bij de sync tool sterk gefocust op het Single Responsibility Principle. Elke klasse heeft één verantwoordelijkheid. Dit zorgt voor overzichtelijkheid in de code. Daarnaast kan de naam van een klasse duidelijker zijn omdat deze de verantwoordelijkheid van de klasse beter kan omschrijven.

Ook hebben we het principe nagestreefd dat alles in onze code losgekoppeld moet zijn van elkaar. Als een klasse een object gebruikt, “weet” deze klasse niet welke specifieke implementatie hij gebruikt, maar kent hij enkel de interface. Dankzij deze interfaces is alles vervangbaar en uitbreidbaar. Wanneer een klasse bijvoorbeeld iets moet kunnen wegschrijven naar een bestand, gebruikt hij gewoon een I/O-interface. Deze klasse moet echter niet weten of er bijvoorbeeld een `XmlWriter` of een `CsvWriter` geïnstantieerd is achter deze interface. Als er dan bijvoorbeeld overgeschakeld moet worden van het schrijven naar een XML-bestand, naar het schrijven naar een CSV-bestand, kan dit door enkel een nieuwe `CsvWriter`-klasse te schrijven. Omdat de klasse een I/O-interface gebruikt om weg te schrijven naar een bestand, moet deze klasse niet aangepast worden als de nieuwe `CsvWriter` deze interface implementeert.

8.2 Werking en keuzes

8.2.1 Generiek

De hele sync tool is generiek gebouwd. Er kunnen gemakkelijk tabellen of velden toegevoegd of verwijderd worden aan de synchronisatie zonder te hercompileren. Ook wisselen van serveradressen of dropfolderpaden kan zonder hercompileren dankzij configuratiebestanden.

Dankzij het veelvuldig gebruik van interfaces, kunnen gemakkelijk nieuwe implementaties van bijvoorbeeld databankcommunicatie of I/O-klassen toegevoegd worden. Dit werd al uitgelegd in de paragraaf “Design principles” hierboven en zorgt ook voor een soort van generiekheid.

8.2.2 Dropfolder

De data die geëxporteerd wordt aan de kant van de backoffice moet natuurlijk ook aan de EVA-kant geraken. Voor het doorsturen van de data waren er verschillende mogelijke werkwijzes:

- Een eerste mogelijkheid bestond uit het werken met een rechtstreekse `NetworkStream` tussen beide applicaties. Dit zou echter het nut van twee afzonderlijke applicaties volledig teniet doen. Als een van beide niet meer werkt, kan de andere immers ook niets doen. Ook kunnen er te veel dingen misgaan bij een rechtstreekse netwerkverbinding.
- Een andere mogelijkheid was het schrijven van web services. Dit is een propere manier om data over een netwerk te verzenden. De backoffice-kant van de applicatie zou dan als server dienen. De EVA-kant kan dan requests sturen naar de backoffice om nieuwe updates te ontvangen en de backoffice zou deze mooi kunnen verzenden. Het probleem dat beide services gelijktijdig moeten runnen blijft echter bestaan.
- De derde mogelijkheid werd ook meteen de gekozen optie. Iedereen was het erover eens dat bij onze requirements een dropfolder de beste keuze was. Enerzijds kan de backoffice-kant schrijven naar deze dropfolder op het netwerk

zonder dat de EVA-kant online moet zijn. Anderzijds kan de EVA-kant al geschreven bestanden onafhankelijk van de backoffice inlezen en importeren in de databank.

8.2.3 XML

Nadat we gekozen hadden voor een dropfolder met exportbestanden, moesten we een beslissing nemen over de structuur van de inhoud. CSV en een eigen formaat behoorden tot de mogelijkheden, maar we hebben uiteindelijk gekozen voor XML. Voor dit formaat bestaan er duidelijke en handige klassen in het .Net Framework. Ook is dit een bekend formaat, waardoor er veel informatie over te vinden is. Met een zelfgeschreven formaat moeten bugs natuurlijk ook volledig zelf opgelost worden. Voor een formaat zoals XML bestaan er vaak al oplossingen die te vinden zijn op het internet. Een derde voordeel is de leesbaarheid van het formaat: wanneer er handmatig iets aangepast moet worden of door een mens gelezen moet worden in het bestand, kan dit zonder al te veel moeite. Het formaat heeft een duidelijke structuur en is goed leesbaar.

8.2.4 Logging

Alles wat er gebeurt in onze service, wordt gelogd. We hebben twee soorten logs voorzien: informatieve logs en error logs. Elke dag wordt er een nieuw tekstbestand gemaakt voor beide soorten logs.

Bij gebeurtenissen zoals het importeren of exporteren van data, wordt er in het informatieve logbestand een lijn met timestamp toegevoegd bij het begin en het einde van de actie. Hiermee kan gemakkelijk nagekeken worden of alles gebeurd is wat nodig was. Ook performance kan hiermee getest worden door te kijken hoelang iedere actie duurt.

Bij mislukkingen van bepaalde acties wordt de fout weggeschreven naar het errorlog-bestand. Deze log bevat een error message en de stack trace van de `Exception` die gegooid werd.

8.2.5 Exporteren van data (backoffice)

Als er geen vorige synchronisatie gebeurd is, wordt een initiële export van de databank uitgevoerd. Dit wil zeggen dat alle data uit de nodige tabellen volledig weggeschreven wordt naar XML-bestanden. Om geheugen te sparen en XML's werkbaar te houden, worden grote tabellen automatisch opgesplitst in verschillende XML-bestanden.

Na iedere synchronisatie wordt de timestamp van het begin van de export weggeschreven naar het configuratiebestand van de applicatie. In dit configuratie-bestand kan deze datum ook handmatig gewijzigd worden, moest dit nodig zijn.

Wanneer er al een vorige synchronisatie gebeurd is, houdt de exporttool hier rekening mee. De tool gaat enkel nieuwe of geüpdatete data selecteren en exporteren naar XML. Dit is dus een incrementele synchronisatie.

Om de tool zo generiek mogelijk te houden, worden alle query's die nodig zijn voor het selecteren van data ingelezen uit een XML-bestand. De tool genereert automatisch juiste XML-structuren voor de verschillende tabellen.

8.2.6 Importeren van data (EVA)

De EVA-kant van de sync tool gaat kijken in de dropfolder of er nieuwe bestanden toegevoegd zijn. Als dit het geval is, gaat hij deze een voor een in chronologische volgorde inlezen en importeren in de databank. Na het importeren van een XML-bestand, schrijft hij de datum van dit bestand weg naar de databank als de datum van de laatste synchronisatie. Bestanden in de dropfolder met een datum voor deze laatste synchronisatiedatum worden niet meer ingelezen.

Fouten kunnen voorkomen en crashes zijn niet uit te sluiten. In de dropfolder zijn daarom drie submappen voorzien: failed, success en archive. Wanneer er iets misloopt met het inserteren van een bepaald record, wordt dit record in een apart XML-bestand bijgehouden met mislukte records in de map "failed". Gelukte records worden op hun beurt bijgehouden in de map "success". Nadat de volledige import gebeurd is, wordt de volledige XML in de map "archive" gearchiveerd en wordt er een notificatiemail

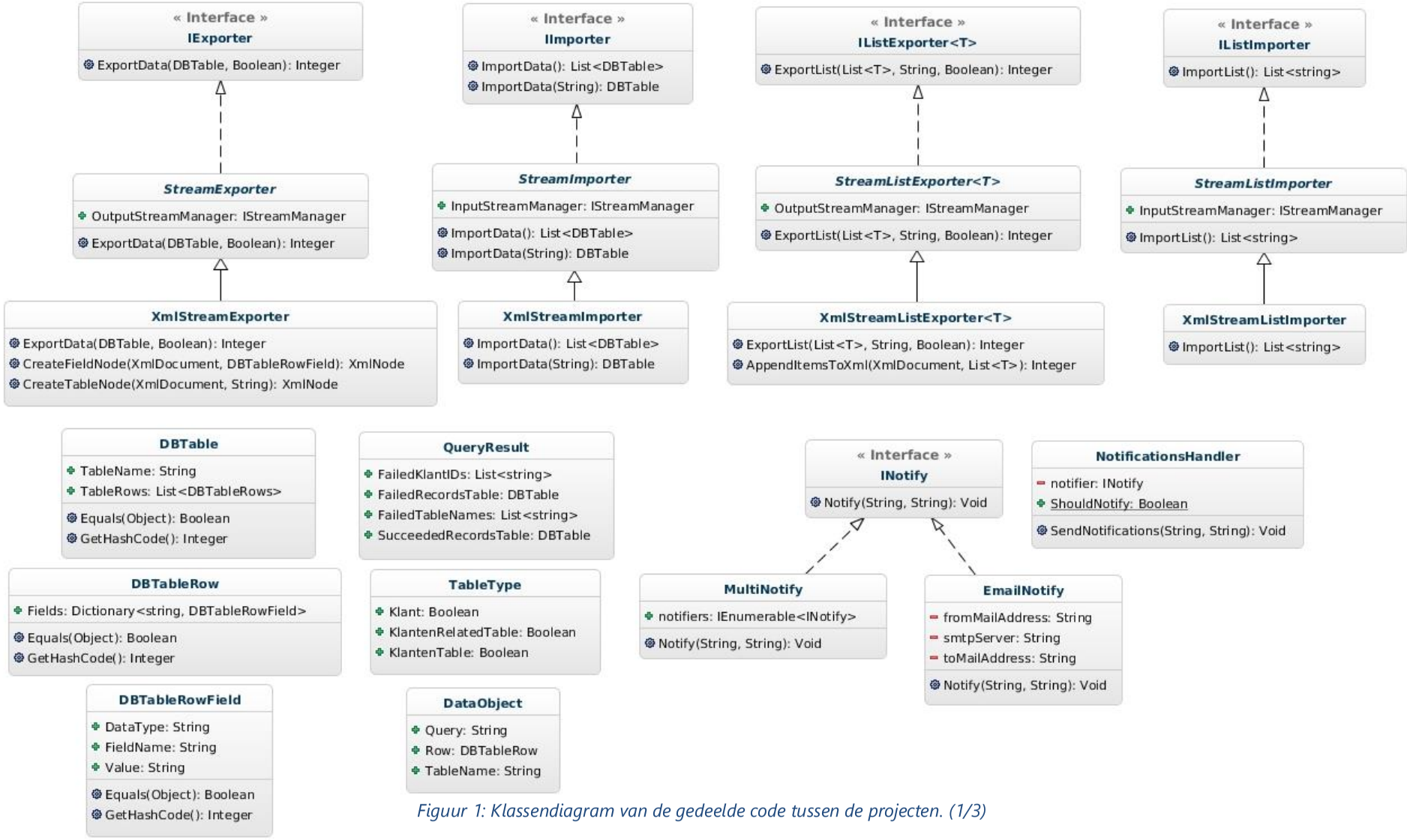
gestuurd naar de verantwoordelijke indien er fouten opgetreden zijn. Dit e-mailadres is natuurlijk weer in te stellen in het configuratiebestand.

Om de generiekheid van de tool te behouden, worden alle query's die nodig zijn voor het inserteren van data gegenereerd op basis van de ingelezen XML-bestanden uit de dropfolder.

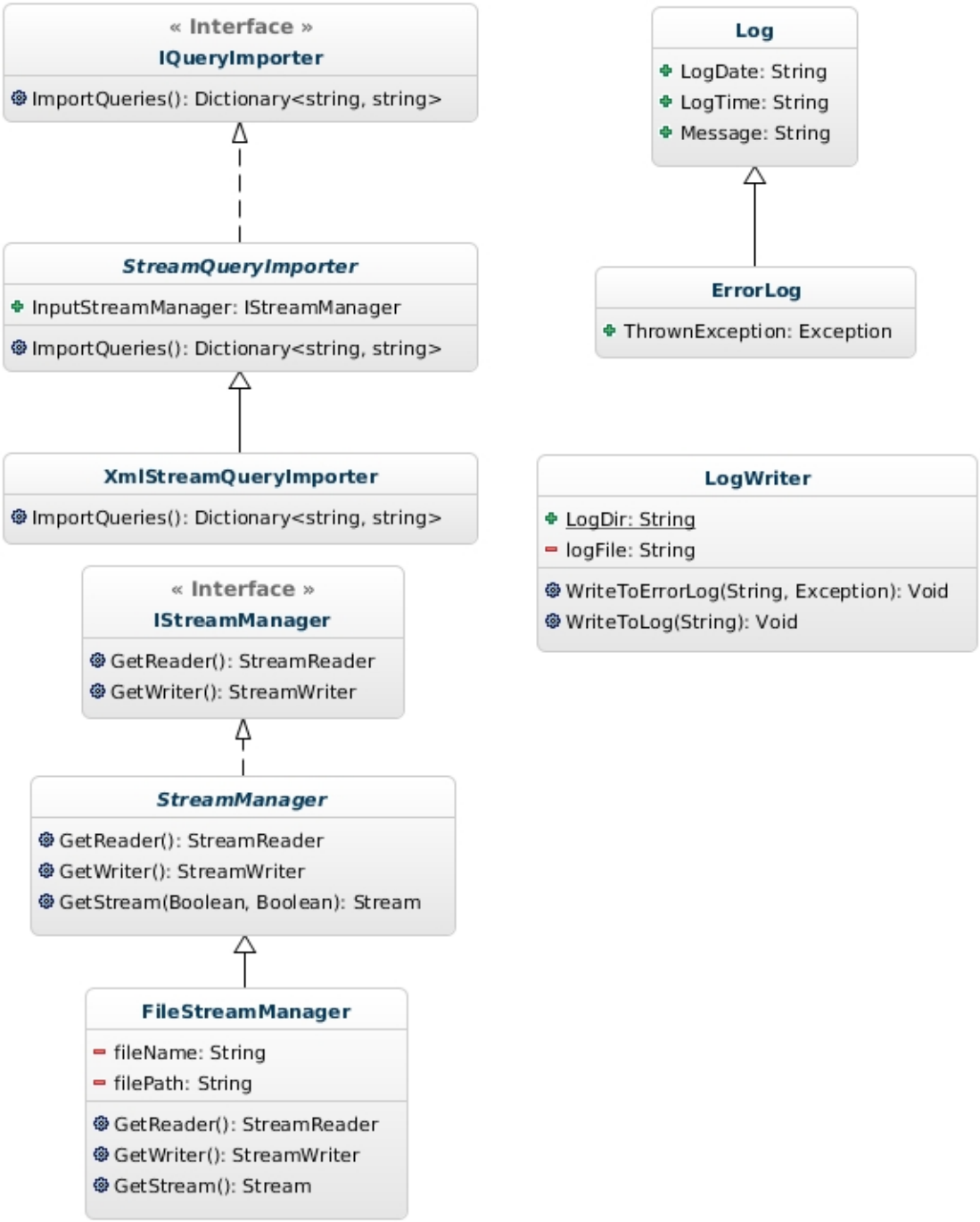
8.3 Opbouw en structuur

Naast de projecten voor de twee afzonderlijke Windows Services hebben we ook nog een shared library gemaakt voor alle code die te gebruiken is in beide projecten. De meeste code is zo generiek dat ze in het shared library zit. Enkel de specifieke code wordt in de projecten van de services zelf opgenomen. Deze specifieke code bestaat uit de flow van de export en de import, de communicatie met de databanken, het genereren van de insertquery's en de specifieke code voor het opstarten en afsluiten van de Windows Services.

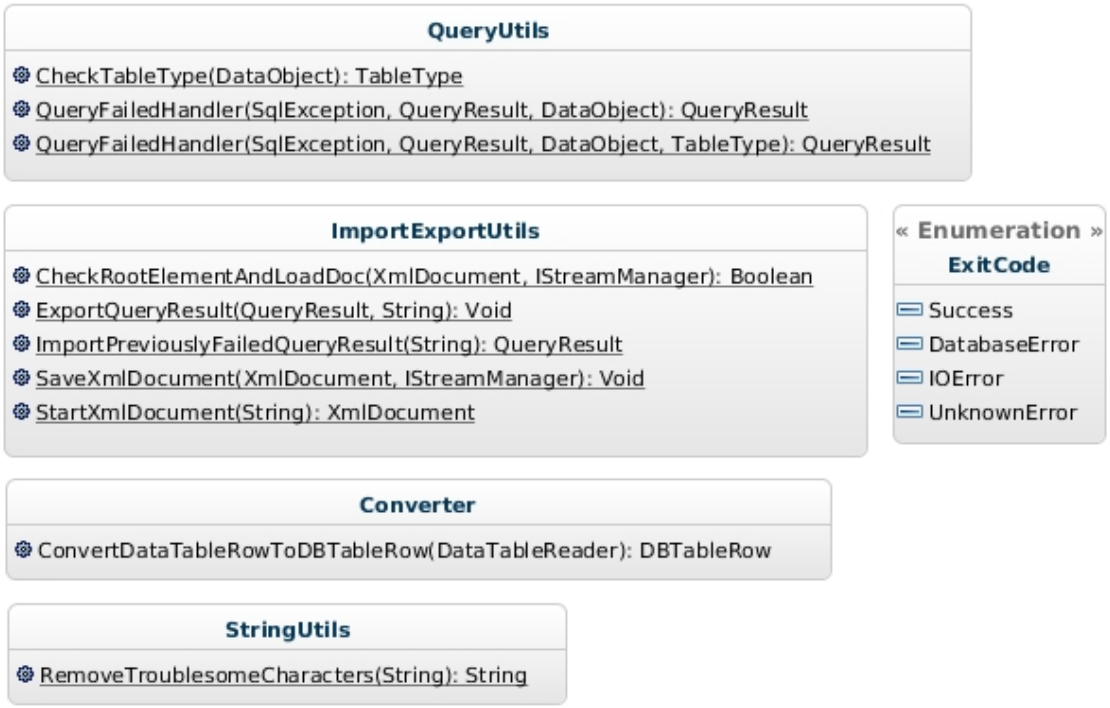
Hieronder de klassendiagrammen met een korte uitleg bij elke klasse.



Figuur 1: Klassendiagram van de gedeelde code tussen de projecten. (1/3)



Figuur 2: Klassendiagram van de gedeelde code tussen de projecten. (2/3)



Figuur 3: Klassendiagram van de gedeelde code tussen de projecten. (3/3)

8.3.1 Gedeelde code

IExporter

Interface om data te exporteren.

StreamExporter

Abstracte klasse die `IExporter` implementeert en een `IStreamManager` als property heeft. Dankzij de `IStreamManager` interface moet de klasse niet weten of er bijvoorbeeld een `FileStreamManager` of een `NetworkStreamManager` gebruikt wordt.

XmlStreamExporter

Een specifieke implementatie van `StreamExporter`. Deze klasse kan tabellen exporteren naar XML met de `StreamWriter` verkregen uit de `IStreamManager`.

IImporter

Interface om data te importeren.

StreamImporter

Abstracte klasse die `IImporter` implementeert en een `IStreamManager` als property heeft. Dankzij de `IStreamManager` interface moet de klasse niet weten of er bijvoorbeeld een `FileStreamManager` of een `NetworkStreamManager` gebruikt wordt.

XmlStreamImporter

Een specifieke implementatie van `StreamImporter`. Deze klasse kan XML-bestanden inlezen die gegenereerd zijn door de `XmlStreamExporter`. De ingelezen bestanden worden in een lijst van `DBTable` objecten gegoten.

IListExporter<T>

Interface om lijsten te exporteren.

StreamListExporter<T>

Abstracte klasse die `IListExporter<T>` implementeert en een `IStreamManager` als property heeft. Dankzij de `IStreamManager` interface moet de klasse niet weten of er bijvoorbeeld een `FileStreamManager` of een `NetworkStreamManager` gebruikt wordt.

XmlStreamListExporter<T>

Een specifieke implementatie van `StreamListExporter<T>`. Deze klasse kan lijsten exporteren naar XML met de `StreamWriter` verkregen uit de `IStreamManager`.

IListImporter

Interface om lijsten te importeren.

StreamListImporter

Abstracte klasse die `IListImporter` implementeert en een `IStreamManager` als property heeft. Dankzij de `IStreamManager` interface moet de klasse niet weten of er bijvoorbeeld een `FileStreamManager` of een `NetworkStreamManager` gebruikt wordt.

XmlStreamListImporter

Een specifieke implementatie van `StreamImporter`. Deze klasse kan XML-bestanden inlezen die gegenereerd zijn door de `XmlStreamListExporter`. De ingelezen bestanden worden in `List` objecten gegoten.

DBTable

`Model` voor een generieke tabel. Bevat een tabelnaam en een lijst van records.

DBTableRow

`Model` voor een generiek record van een tabel. Bevat een lijst van velden die elk een naam, een type en een waarde bijhouden.

DBTableRowField

`Model` voor een generiek veld van een record. Bevat een naam, een type en een waarde.

QueryResult

`Model` voor het bijhouden van de resultaten van uitgevoerde query's. Hierin worden onder andere gefaalde insertquery's bijgehouden.

TableType

`Model` voor het bijhouden van het type tabel waarmee gewerkt wordt.

DataObject

`Model` voor het bijhouden van een query en gerelateerde informatie.

INotify

Interface om notificaties te verzenden.

EmailNotify

Een specifieke implementatie van `INotify`. Deze klasse kan notificaties via e-mail verzenden.

MultiNotify

Een specifieke implementatie van `INotify`. Deze klasse kan meerdere `INotify` objecten bijhouden. Wanneer de functie `Notify()` wordt aangeroepen, doorloopt deze functie de lijst met `INotify` objecten en roept op elk van deze objecten de functie `Notify()` aan.

NotificationHandler

Wanneer de functie `SendNotifications()` van deze handler wordt aangeroepen, wordt er een notificatie verzonden met het meegegeven `INotify` object. Dit gebeurt echter enkel wanneer er tijdens de huidige synchronisatie fouten zijn opgetreden.

IQueryImporter

Interface om query's te importeren.

StreamQueryImporter

Abstracte klasse die `IQueryImporter` implementeert en een `IStreamManager` als property heeft. Dankzij de `IStreamManager` interface moet de klasse niet weten of er bijvoorbeeld een `FileStreamManager` of een `NetworkStreamManager` gebruikt wordt.

XmlStreamQueryImporter

Een specifieke implementatie van `StreamQueryImporter`. Deze klasse kan XML-bestanden inlezen die aan het XML-schema voor Query-XML's (zie hieronder) voldoen. De ingelezen bestanden worden in een `Dictionary` gegoten dat als keys de namen van de tabellen heeft en als values de query's zelf.

IStreamManager

Interface om een `Reader` of een `Writer` te verkrijgen om respectievelijk te lezen van of te schrijven naar een `Stream`. Als de `Stream` nog niet of niet meer bestaat zorgt deze manager ervoor dat de `Stream` opnieuw aangemaakt wordt.

StreamManager

Abstracte klasse die de `IStreamManager` implementeert. Daarnaast bevat deze klasse een abstracte, protected functie om een readable en/of writable `Stream` te verkrijgen.

FileStreamManager

Een specifieke implementatie van `StreamManager`. Deze klasse kan `FileStream` objecten aanmaken en deze in een `StreamWriter` of `StreamReader` teruggeven. Deze klasse zorgt dus voor een object dat kan interacteren met bestanden.

Log

Een modelklasse voor een log. Bevat properties voor een datum, een tijd en een bericht.

ErrorLog

Een modelklasse voor een error log. Erft over van `Log`. Bevat daarnaast nog een property om een gegooide `Exception` in bij te houden.

LogWriter

Klasse waarmee `Log` en `ErrorLog` objecten gegenereerd kunnen worden. Deze logs worden als leesbare logs weggeschreven naar tekstbestanden.

QueryUtils

Statische klasse met hulpmethodes voor het werken met query's.

ImportExportUtils

Statische klasse met hulpmethodes voor het importeren en exporteren van data.

Converter

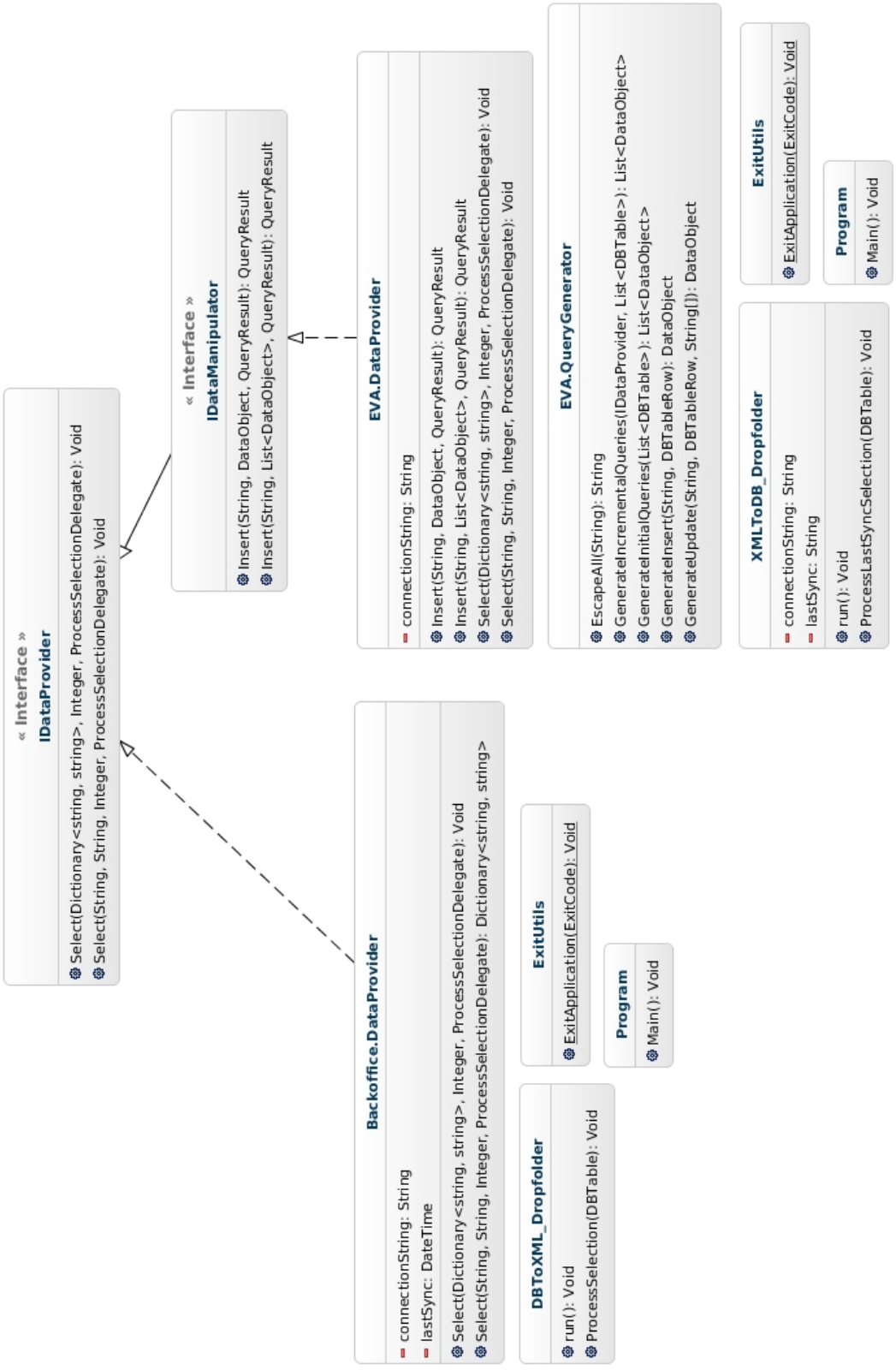
Klasse voor het converteren van data naar andere formaten.

StringUtils

Statische klasse met hulpmethodes voor het werken met tekst.

ExitCode

Enumeratie van verschillende afsluitcodes.



Figuur 4: Klassendiagram van de specifieke code van beide applicaties.

IDataProvider

Interface om data te selecteren.

IDataManipulator

Interface om data te inserteren/updaten. Implementeert ook `IDataProvider` voor het selecteren van data.

8.3.2 Backoffice

DataProvider

Kan selectquery's uitvoeren op de databank van de backoffice en met de meegegeven `IExporter` wegschrijven. De connection string voor de verbinding met deze databank kan gewijzigd worden in het configuratiebestand.

ExitUtils

Statische klasse waarmee de tool afgesloten kan worden. Stuurt notificaties voor het afsluiten. Gebruikt de enumeratie `ExitCode` om te specifiëren waarom er afgesloten werd.

DBToXML_Dropfolder

Bevat de flow van de applicatie als er van een databank naar XML-bestanden in een dropfolder geëxporteerd moet worden.

Program

Main entry point van de backofficetool. Maakt een `DBToXML_Dropfolder` object en roept hierop de functie `run()` aan.

8.3.2.1 EVA

SyncImportTool

Klasse die de opstart- en afsluitcode van de Windows Service bevat.

DataProvider

Kan select-, update- en insertquery's uitvoeren op de databank van het extranet. De connection string voor de verbinding met deze databank kan gewijzigd worden in het configuratiebestand.

QueryGenerator

Genereert de insert- en updatequery's voor het importeren van de ingelezen XML-bestanden in de databank. Zorgt ook voor het escapen van alle ongeldige tekens.

ExitUtils

Statische klasse waarmee de tool afgesloten kan worden. Stuurt notificaties voor het afsluiten. Gebruikt de enumeratie `ExitCode` om te specifiëren waarom er afgesloten werd.

XMLToDB_Dropfolder

Bevat de flow van de applicatie als er van XML-bestanden in een dropfolder naar een databank geïmporteerd moet worden.

Program

Main entry point van de backofficetool. Maakt een `XMLToDB_Dropfolder` object en roept hierop de functie `run()` aan.

8.3.3 XML-schema voor Query-XML's:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="SelectQueries"
    elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="queries" type="selectqueries" />

  <xs:complexType name="selectqueries">
    <xs:annotation>
      <xs:documentation>
        The list of queries to be used to select the data to be
        imported from the source database.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="select" type="selectType" />
    </xs:sequence>
  </xs:complexType>
```

```
  <xs:complexType name="selectType">
    <xs:annotation>
      <xs:documentation>
        A select query.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="table" type="xs:string" />
      <xs:element name="query" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

8.4 Problemen

Tijdens het project zijn we regelmatig op problemen gestoten. Gelukkig hebben we deze meestal vlot kunnen oplossen, maar soms haalden ze de planning wel een beetje overhoop.

Bij de databanksynchronisatietool zijn we begonnen met het werkend maken van de initiële synchronisatie. Wanneer de flow klaar was en we al een aantal keer getest hadden met een paar tabellen uit de databank, begonnen we met het testen van een volledige initiële synchronisatie. Dit viel echter tegen: er waren blijkbaar heel wat bugs in onze code geslopen. Het heeft ons iets meer dan drie dagen gekost om alle bugs eruit te halen en de initiële synchronisatietesten foutloos te laten verlopen. Hier hadden we niet op gerekend, maar doordat deze bugs nu verholpen waren, kon de incrementele synchronisatie ook vlotter gecodeerd worden. Het enige verschil tussen de initiële en incrementele synchronisaties is immers het rekening houden met de laatste synchronisatietimestamp.

Een ander probleem waar we op gestoten zijn, was de ingewikkeldheid van de backofficedatabank. Zoals beschreven in het hoofdstuk “Design”, is dit een databank

die al lang mee gaat en waar veel tabellen en velden in aanwezig zijn die niet meer gebruikt worden. Dit kan natuurlijk voor veel verwarring zorgen, waardoor we een hele tijd zijn bezig geweest met het uitdokteren van de structuur.

Het opzetten van testomgevingen voor de MS SQL Server en de IIS Server kostte ook wat tijd. We zijn programmeurs en niet enorm ervaren in servers opzetten, maar het is ons uiteindelijk toch gelukt.

9 Testen

9.1 Agile

In het hoofdstuk “Design”, in de paragraaf “Werkwijze”, hebben we al aangegeven dat we volgens het agile-principe hebben gewerkt. Dit gaf ons de mogelijkheid om aan het einde van elke sprint, wekelijks dus, ons tot dan afgewerkte deel te demonstreren aan en eventueel te laten testen door de productowners. Hier hebben we gebruik van gemaakt en we kregen al snel feedback over ons project. Gelukkig was deze feedback voornamelijk heel positief, maar hier en daar werd nog een extra feature gevraagd. Dankzij de vroege feedback hadden we nog tijd om deze extra features toe te voegen. Een voorbeeld van zo’n feature zijn de grafieken van de meetwaarden naast de tabellen met de absolute waarden.

9.2 Werkgroep artsen

De werkgroep met de huisartsen die beschreven staat in het hoofdstuk “Design” is ook een vorm van testen. De artsen hebben het extranet niet zelf gebruikt, maar hebben het wel gezien en feedback hierover gegeven. Dit was een zeer nuttige test om na te gaan of we voldeden aan de noden van de artsen.

9.3 Finale testen

Aan het einde van onze stage werd nog eens alles grondig door ons en door de productowners getest. De laatste bugs konden eruit gehaald worden en er werd gekeken of het project aan alle requirements voldeed. De feedback die we na deze testen kregen was enorm positief. Iedereen was zeer tevreden met het resultaat en het project was klaar om binnenkort in een gesloten bèta gelanceerd te worden (zie volgende paragraaf).

9.4 Closed beta

Ons extranet zal in september, na onze stage, starten als een pilootproject bij een beperkt aantal artsen, een soort van gesloten bèta-test. Hierdoor kan dit groepje artsen

de website al eens grondig testen en feedback geven over de gebruiksvriendelijkheid en inhoud van het extranet. Dit kan zeer nuttig zijn. Als er problemen zouden zijn met de gebruiksvriendelijkheid van de website voor artsen, zouden veel gebruikers al snel afgeschrikt kunnen worden. Door een test bij een klein groepje artsen, kunnen de eventuele foutjes in de userinterface er al uitgehaald worden voordat alle artsen toegang krijgen tot het extranet.

9.5 Unit Testing

We hebben ervoor gekozen om gebruik te maken van unittests in onze code. Zoals al eerder vermeld, gebruiken we het ingebouwde Unit Testing-framework van .Net in combinatie met Rhino Mocks. De uitleg over Rhino Mocks kan u terugvinden onder het hoofdstuk “Nieuwe Technologieën”. We hebben zowel in het extranet als in de datasynchronisatietool gebruikgemaakt van deze unittests.

Onze ervaring met unittesten is dat het maken van deze testen wel wat meer tijd kost, maar dat je deze investering dubbel en dik terugverdient door het aantal bugs dat je vermijdt dankzij deze tests. Het zorgt er ook voor dat je onmiddellijk ziet of je andere code nog werkt wanneer je ergens iets aanpast in een bestaande klasse. Op deze manier vermijd je dat iemand die een klasse moet aanpassen, eerst moet uitzoeken of hetgeen hij verandert niet de volledige programmalogica in de war stuurt. Het hoofddoel van onze unittests is dan ook een vangnet te creëren voor de mensen die de ontwikkeling van het extranet na ons overnemen.

We hebben er ook voor gezorgd dat de unittests gebruikt kunnen worden zonder actieve databankconnectie. Elke dependency wordt namelijk gemockt met het Rhino Mocks mockingframework. Elke unittest test exact één functie, en elke functie heeft slechts één verantwoordelijkheid. Op deze manier is het mogelijk om alle unittests onafhankelijk van elkaar uit te voeren.

Onze conclusie omtrent unittests is dat deze enorm waardevol zijn voor het ontwikkelen en debuggen van een project. Unittests verzorgen een uitstekend vangnet voor codeerfouten en foutieve codeaanpassingen.

10 Conclusie

Onze stage is ons enorm goed bevallen. We kregen veel vrijheid, mochten zelf ideeën voorstellen, kregen alles wat we nodig hadden en hebben veel bijgeleerd.

We hebben tijdens onze stage een goede kijk gekregen op het bedrijfsleven. We werden behandeld als collega's en dit gaf ons een goed gevoel. Ook hebben we enorm veel bijgeleerd op technisch vlak. Met onder andere MVC4, Unity, Rhino Mocks en eID hadden we nog nooit eerder gewerkt. Nu hebben we deze frameworks echter goed onder de knie, waardoor we ze in volgende projecten gemakkelijk opnieuw kunnen gebruiken.

We hebben ons bijna elke week aan de planning kunnen houden, met hier en daar een kleine vertraging die de week erna weer ingehaald werd. Maar ook hierover hebben we veel geleerd tijdens onze stage. Telkens een taak afgewerkt was, vergeleken we de geplande tijd met de effectieve tijd die we voor deze taak nodig hadden. Dankzij deze methode werden onze wekelijkse sprintplanningen telkens beter.

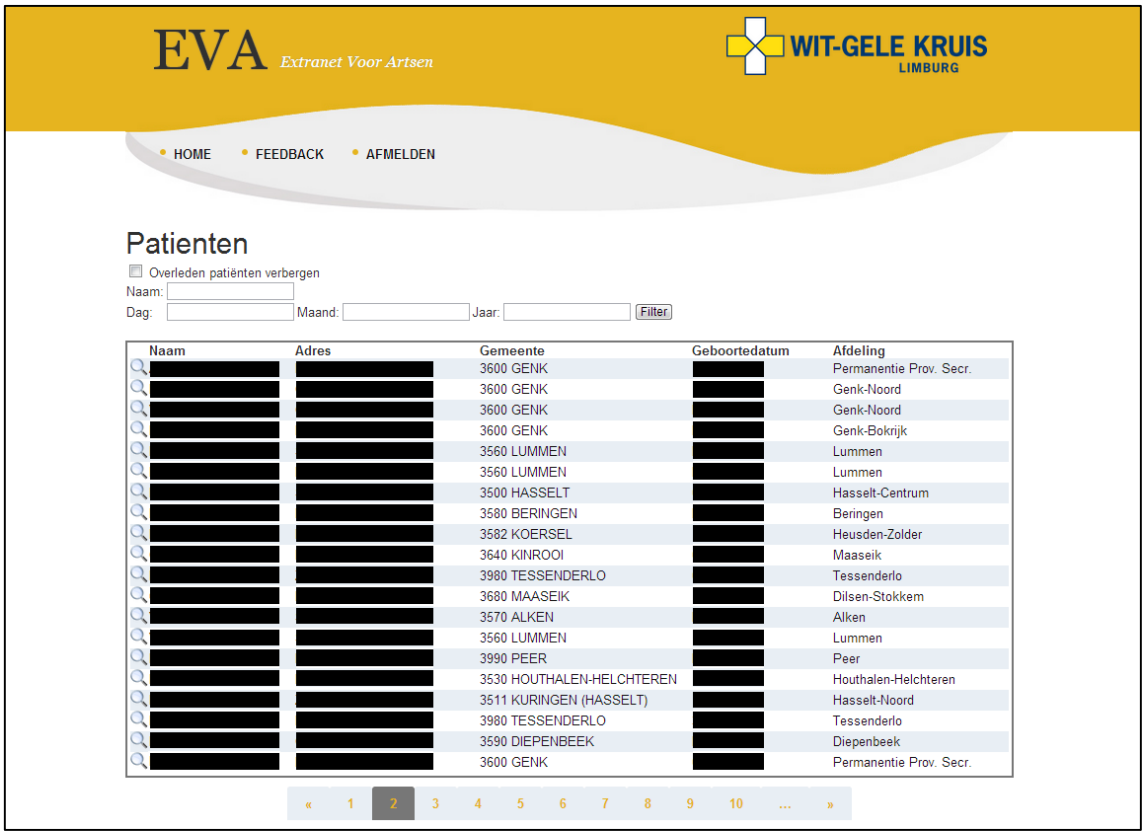
De enige problemen die we ondervonden tijdens ons project, waren technische problemen. Dit soort problemen zijn altijd op te lossen, dus dat hebben we gedaan. Bij het oplossen van deze problemen werden we zeer vrij gelaten. We mochten zelf een lijst met voor- en nadelen van mogelijke oplossingen maken. Uiteindelijk besloten we dan samen met onze bedrijfspromotor welke oplossing het beste zou zijn.

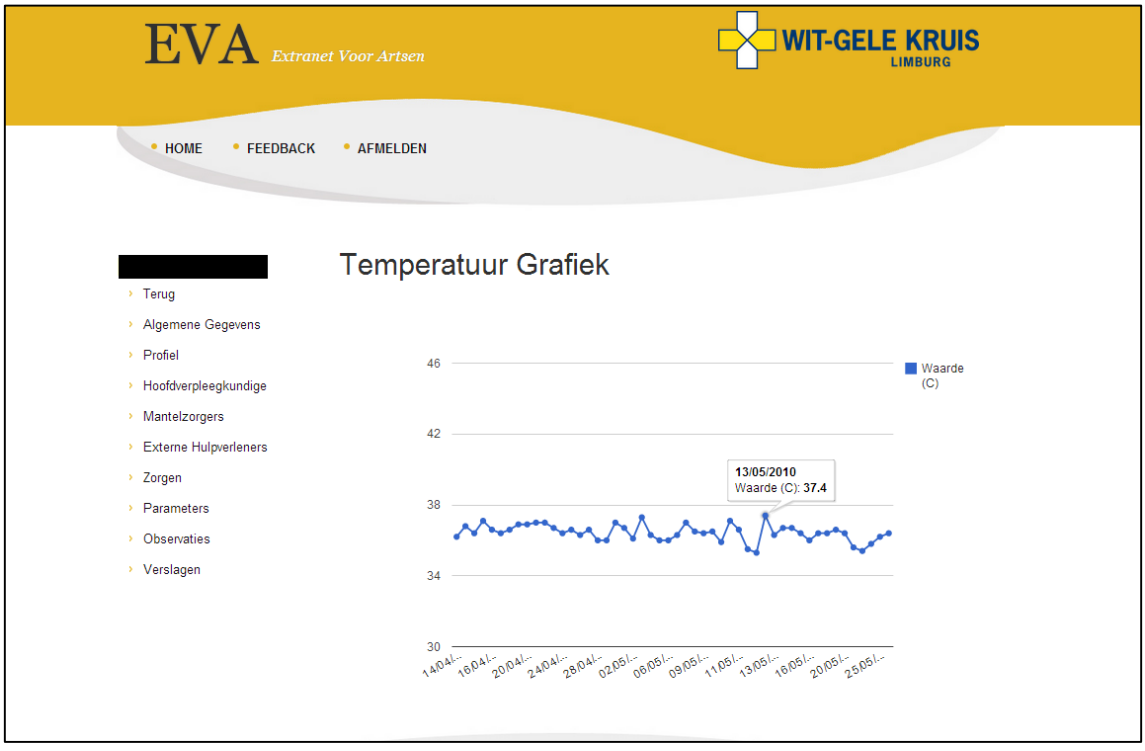
Zelf zijn we zeer tevreden met het resultaat van onze stage. We hebben ons project mooi kunnen uitwerken en alles was af voor de deadline. We voelen ons ook heel vereerd dat we ons project al hebben mogen voorstellen aan een aantal dokters en dat er een bèta gepland staat voor september.

We weten zeker dat deze stage een hele goede invloed gaat hebben op ons toekomstig werk. We zijn al vertrouwd met de manier van werken in het bedrijfsleven en we kunnen plannen en programmeren zoals het hoort. Deze stage zullen we nooit vergeten.

11 Bijlagen

11.1 Screenshots





EVA *Extranet Voor Artsen*

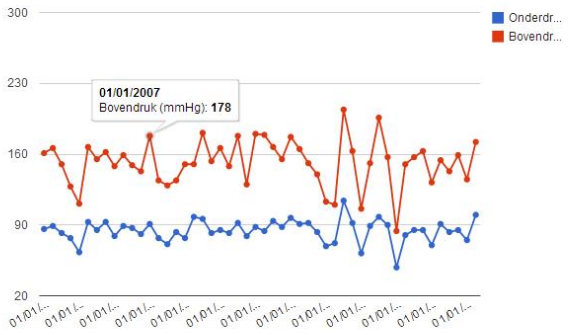
WIT-GELE KRUIS LIMBURG

HOME FEEDBACK AFMELDEN

[redacted]

- Terug
- Algemene Gegevens
- Profiel
- Hoofdverpleegkundige
- Mantelzorgers
- Externe Hulpverleners
- Zorgen
- Parameters
- Observaties
- Verslagen

Bloeddruk Grafiek



EVA *Extranet Voor Artsen*

WIT-GELE KRUIS LIMBURG

HOME FEEDBACK AFMELDEN

[redacted]

- Terug
- Algemene Gegevens
- Profiel
- Hoofdverpleegkundige
- Mantelzorgers
- Externe Hulpverleners
- Zorgen
- Parameters
- Observaties
- Verslagen

Zorggeschiedenis

Code	Naam	Period
PF	perfusie intraveneus	01/05/2013 tot 31/05/2013
IC	subcutane inspuiting	17/08/2012 tot 15/09/2012
T	toilet	10/05/2013 tot N/A
PF	perfusie intraveneus	28/05/2013 tot N/A

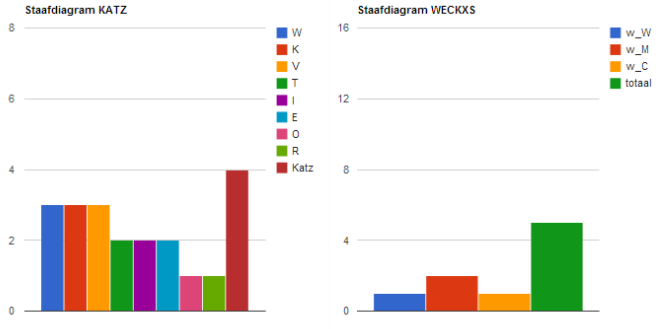
1

[redacted]

- Terug
- Algemene Gegevens
- Profiel
- Hoofdverpleegkundige
- Mantelzorgers
- Externe Hulpverleners
- Zorgen
- Parameters
- Observaties
- Verslagen

Details Profielaanvraag

Code	Waarde	Score	Beschrijving
wassen	W	3	gedeeltelijke hulp onder en boven de gordel
kleden	K	3	gedeeltelijke hulp onder en boven de gordel
verplaatsen	V	3	hulp van derden is noodzakelijk
toilet	T	2	gedeeltelijk hulp van derden om naar toilet te gaan of zich te reinigen
incontinentie	I	2	accidenteel incontinent voor urine of faeces (incl. blaassonde en stoma)
eten	E	2	voorbereidende hulp
orientatie	O	1	geen probleem
rusteloos	R	1	geen probleem
KATZ	/	4	Totale KATZ score
woonsituatie	w_W	1	met beschikbare valide persoon
mantelzorg	w_M	2	er is soms (mits afspraken) iemand beschikbaar
comfort	w_C	1	ingerichte badkamer
Totaal	/	5	Totale score van WECKXS en KATZ



11.2 Codefragmenten Pagina “Algemene Gegevens”

11.2.1 Models

```
public interface IPatient
{
    string Afdeling { get; set; }
    int? Dokter_id { get; set; }
    string Gemeente { get; set; }
    string Huisnr { get; set; }
    int ID { get; set; }
    string Afdeling_id { get; set; }
    string Naam { get; set; }
    string Postcode { get; set; }
    Int64? Rijksregister { get; set; }
    string Straat { get; set; }
    string Voornaam { get; set; }
    string Status_code { get; set; }
    string Status_omschrijving { get; set; }
    string Ziekenhuis { get; set; }
    DateTime? Geboortedatum { get; set; }
}

public class Patient : Eva.Models.IPatient
{
    public int ID { get; set; }
    public string Afdeling_id { get; set; }
    public int? Dokter_id { get; set; }
    public string Naam { get; set; }
    public string Voornaam { get; set; }
    public string Straat { get; set; }
    public string Huisnr { get; set; }
    public string Postcode { get; set; }
    public string Gemeente { get; set; }
    public Int64? Rijksregister { get; set; }
    public string Afdeling { get; set; }
    public string Status_code { get; set; }
    public string Status_omschrijving { get; set; }
    public string Ziekenhuis { get; set; }
    public DateTime? Geboortedatum { get; set; }
}
```

11.2.2 ViewModels

```
public interface IPatientViewModel
{
    string Adres { get; }
    string Afdeling { get; }
    string Gemeente { get; }
    string Naam { get; }
    string Rijksregister { get; }
    string Status { get; }
    string Ziekenhuis { get; }
    int Id { get; }
    string Afdeling_id { get; }
    string Geboortedatum { get; }
}
```

```
public class PatientViewModel : Eva.ViewModels.IPatientViewModel
{
    private readonly IPatient patient;

    public PatientViewModel(IPatient patient)
    {
        if (patient == null)
        {
            throw new ArgumentNullException("PatientIsNull");
        }
        this.patient = patient;
    }

    public string Adres
    {
        get
        {
            if (string.IsNullOrEmpty(this.patient.Straat))
                return "N/A";
            else
                return this.patient.Straat + " " + this.patient.Huisnr;
        }
    }

    public string Gemeente
    {
        get
        {
            if (string.IsNullOrEmpty(this.patient.Postcode))
                return "N/A";
            if (string.IsNullOrEmpty(this.patient.Gemeente))
                return this.patient.Postcode;
            else
                return this.patient.Postcode + " " +
                       this.patient.Gemeente.ToUpper();
        }
    }

    public String Rijksregister
    {
        get
        {
            if (string.IsNullOrEmpty(this.patient.Rijksregister.ToString()))
                return "N/A";
            else
                return this.patient.Rijksregister.ToString();
        }
    }

    public string Geboortedatum
    {
        get
        {
            if (this.patient.Geboortedatum != null)
                return String.Format("{0:dd/MM/yyyy}",
                       this.patient.Geboortedatum);
            else
                return "N/A";
        }
```

```
        }

        public int Id
        {
            get
            {
                return this.patient.ID;
            }
        }

        public string Afdeling_id
        {
            get
            {
                return this.patient.Afdeling_id;
            }
        }

        public string Naam
        {
            get
            {
                return this.patient.Naam + " " + this.patient.Voornaam;
            }
        }

        public string Afdeling
        {
            get
            {
                if (string.IsNullOrEmpty(this.patient.Afdeling))
                    return "N/A";
                else
                    return this.patient.Afdeling;
            }
        }

        public string Status
        {
            get
            {
                if (String.IsNullOrEmpty(this.patient.Status_omschrijving) &&
                    String.IsNullOrEmpty(this.patient.Status_code))
                    return "N/A";
                return this.patient.Status_omschrijving + " (" +
                       this.patient.Status_code + ")";
            }
        }

        public string Ziekenhuis
        {
            get
            {
                if (String.IsNullOrEmpty(this.patient.Ziekenhuis))
                    return "n.v.t.";
                return this.patient.Ziekenhuis;
            }
        }
    }
```

11.2.3 DBContext

```
public class EvaDBContext : DbContext
{
    public EvaDBContext()
        : base("EvaDBContextConnectionString")
    { }

    public DbSet<Patient> Patienten { get; set; }
    public DbSet<HoofdVPK> HoofdVPKs { get; set; }
    public DbSet<Profiel> Profielen { get; set; }
    public DbSet<Mantelzorger> Mantelzorgers { get; set; }
    public DbSet<ExterneHulpverlener> ExterneHulpverleners { get; set; }
    public DbSet<Zorg> Zorgen { get; set; }
    public DbSet<Observatie> Observaties { get; set; }
    public DbSet<Verslag> Verslagen { get; set; }
    public DbSet<Vakgebied> Vakgebieden { get; set; }
    public DbSet<Parameter> Parameters { get; set; }
    public DbSet<Dokter> Dokters { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        modelBuilder = buildPatient(modelBuilder);
        modelBuilder = buildHoofdVPK(modelBuilder);
        modelBuilder = BuildProfielen(modelBuilder);
        modelBuilder = BuildMantelzorgers(modelBuilder);
        modelBuilder = BuildExterneHulpverleners(modelBuilder);
        modelBuilder = BuildZorgen(modelBuilder);
        modelBuilder = BuildObservaties(modelBuilder);
        modelBuilder = BuildVerslagen(modelBuilder);
        modelBuilder = BuildVakgebieden(modelBuilder);
        modelBuilder = BuildParameters(modelBuilder);
        modelBuilder = BuildDokters(modelBuilder);
        base.OnModelCreating(modelBuilder);
    }
```

11.2.4 Eén voorbeeld van modelbuilderfunctie

```
    private DbModelBuilder buildPatient(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Patient>().ToTable("PatientenOverzichtView");
        modelBuilder.Entity<Patient>().HasKey(p => p.ID);
        modelBuilder.Entity<Patient>().Property(p =>
              p.ID).HasColumnName("klant_id");
        modelBuilder.Entity<Patient>().Property(p =>
              p.Afdeling_id).HasColumnName("afdeling_id");

        modelBuilder.Entity<Patient>().Property(p =>
              p.Naam).HasColumnName("naam");
        modelBuilder.Entity<Patient>().Property(p =>
              p.Voornaam).HasColumnName("voornaam");
        modelBuilder.Entity<Patient>().Property(p =>
              p.Straat).HasColumnName("straat");
        modelBuilder.Entity<Patient>().Property(p =>
              p.Huisnr).HasColumnName("huisnummer");
        modelBuilder.Entity<Patient>().Property(p =>
              p.Geboortedatum).HasColumnName("geboortedatum");
```

```
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Rijksregister).HasColumnName("rijksregister").IsOptional();
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Afdeling).HasColumnName("afdeling");
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Gemeente).HasColumnName("gemeente");
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Postcode).HasColumnName("postcode");
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Dokter_id).HasColumnName("dokter_id");

            modelBuilder.Entity<Patient>().Property(p =>
                  p.Status_code).HasColumnName("status_code");
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Status_omschrijving).HasColumnName("status_omschrijving");
            modelBuilder.Entity<Patient>().Property(p =>
                  p.Ziekenhuis).HasColumnName("ziekenhuis");
            return modelBuilder;
        }
```

11.2.5 Ifactory

```
public interface IViewModelFactory
{
    IHoofdVPKViewModel CreateHoofdVPKViewModel(
          Eva.Models.IHoofdVPK hoofdvpk);
    IPatientViewModel CreatePatientViewModel(Eva.Models.IPatient patient);
    IProfielViewModel CreateProfielViewModel(Eva.Models.IProfiel profiel);
    IMantelzorgerViewModel CreateMantelzorgerViewModel(
          Eva.Models.IMantelzorger mantelzorger);
    IPatientenOverzichtViewModel CreatePatientenOverzichtViewModel();
    IExterneHulpverlenerViewModel CreateExterneHulpverlenerViewModel(
          Eva.Models.IExterneHulpverlener hulpverlener);
    IZorgViewModel CreateZorgViewModel(Eva.Models.IZorg zorg);
    IObservatieViewModel CreateObservatieModel(
          Eva.Models.IObservatie observatie);
    IVerslagViewModel CreateVerslagViewModel(
          Eva.Models.IVerslag verslag);
    IParameterViewModel CreateParameterViewModel(
          Eva.Models.IParameter parameter);
    IBloeddrukViewModel CreateBloeddrukViewModel(
          Eva.Models.IParameter onderdruk,
          Eva.Models.IParameter bovendruk);
}
```

11.2.6 Fragment van factory

```
public class ViewModelFactory : Eva.ViewModels.Factory.IViewModelFactory
{
    public IPatientViewModel CreatePatientViewModel(IPatient patient)
    {
        IPatientViewModel patientViewModel = new PatientViewModel(patient);
        return patientViewModel;
    }
    …
}
```

11.2.7 Unity bootstrapper

```
public static class Bootstrapper
{
    public static IUnityContainer Initialise()
    {
        var container = BuildUnityContainer();

        DependencyResolver.SetResolver(
                    new UnityDependencyResolver(container));

        return container;
    }

    private static IUnityContainer BuildUnityContainer()
    {
        var container = new UnityContainer();

        RegisterTypes(container);

        return container;
    }

    public static void RegisterTypes(IUnityContainer container)
    {
        container.RegisterType<IViewModelFactory,
              Eva.ViewModels.Factory.ViewModelFactory>();

        container.RegisterType<IPatientRepository,
              Eva.Repositories.EF.PatientRepository>();
        container.RegisterType<IHvpkRepository,
              Eva.Repositories.EF.HvpkRepository>();
        container.RegisterType<IProfielRepository,
              Eva.Repositories.EF.ProfielRepository>();
        container.RegisterType<IMantelzorgerRepository,
              Eva.Repositories.EF.MantelzorgerRepository>();
        container.RegisterType<IExterneHulpverlenerRepository,
              Eva.Repositories.EF.ExterneHulpverlenerRepository>();
        container.RegisterType<IZorgRepository,
              Eva.Repositories.EF.ZorgRepository>();
        container.RegisterType<IObservatieRepository,
              Eva.Repositories.EF.ObservatieRepository>();
        container.RegisterType<IVerslagRepository,
              Eva.Repositories.EF.VerslagRepository>();
        container.RegisterType<IParameterRepository,
              Eva.Repositories.EF.ParameterRepository>();
        container.RegisterType<IDokterRepository,
              Eva.Repositories.EF.DokterRepository>();
        container.RegisterType<IPatientProtection,
              Eva.Utils.PatientProtection>();
    }
}
```

11.2.8 Patient Details View

```
@model Eva.ViewModels.IPatientViewModel
```

```
@{
    ViewBag.Title = "Algemene gegevens";
}

<h2>Algemene gegevens</h2>

<fieldset>
    <legend>@Html.DisplayFor(model => model.Naam)</legend>
    <table class="table_patient_details">
        <colgroup>
            <col class="col1" />
            <col class="col2" />
            <col class="col3" />
            <col class="col4" />
            <col class="col5" />
            <col class="col6" />
            <col class="col7" />
            <col class="col8" />
        </colgroup>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Naam)
            </td>
            <td>
                @Html.DisplayFor(model => model.Naam)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Adres)
            </td>
            <td>
                @Html.DisplayFor(model => model.Adres)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Gemeente)
            </td>
            <td>
                @Html.DisplayFor(model => model.Gemeente)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Geboortedatum)
            </td>
            <td>
                @Html.DisplayFor(model => model.Geboortedatum)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Rijksregister)
            </td>
            <td>
                @Html.DisplayFor(model => model.Rijksregister)
            </td>
        </tr>
        <tr>
            <td>
```

```
                @Html.DisplayNameFor(model => model.Afdeling)
            </td>
            <td>
                @Html.DisplayFor(model => model.Afdeling)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Status)
            </td>
            <td>
                @Html.DisplayFor(model => model.Status)
            </td>
        </tr>
        <tr>
            <td>
                @Html.DisplayNameFor(model => model.Ziekenhuis)
            </td>
            <td>
                @Html.DisplayFor(model => model.Ziekenhuis)
            </td>
        </tr>
    </table>
</fieldset>

<p>
    @Html.ActionLink("Terug naar patientenoverzicht", "Index")
</p>

@section sidebar {
        @Html.Partial("_SubMenuPartial")
}
```

11.2.9 PatientController

```
    [Authorize]
    public class PatientController : Controller
    {
        private readonly int NUMBEROFPATIENTSPERPAGE =
              Properties.Settings.Default.RecordsPerPage;

        private IPatientRepository patientRepository;
        private IViewModelFactory factory;
        private IPatientProtection protection;


        public PatientController(IPatientRepository patientRepository,
              IViewModelFactory factory, IPatientProtection protection)
        {
            this.patientRepository = patientRepository;
            this.factory = factory;
            this.protection = protection;
        }

        public ActionResult Index(int? i, string naam, string dag, string maand,
              string jaar, bool? overledenFiltered)
        {
            int index = i ?? 1;
            int? user_id = this.protection.getCurrentUser();
            if ( user_id == null)
```

```
            return RedirectToAction("UnauthorizedAttempt",
                   "EidAuthentication");

        string message = null;
        message = getFeedbackFromSearchEngine(dag, maand, jaar);

        IList<IPatient> patienten =
              this.patientRepository.GetPatientenOverviewPage(naam,
              this.FactorYearOrDay(dag), this.FactorMonth(maand),
              this.FactorYearOrDay(jaar), overledenFiltered,
              NUMBEROFPATIENTSPERPAGE, index, (int)user_id);

        int count =  this.patientRepository.GetCount(naam,
                     this.FactorYearOrDay(dag), this.FactorMonth(maand),
                     this.FactorYearOrDay(jaar), overledenFiltered,
                     (int)user_id);
        IList<IPatientViewModel> models = new List<IPatientViewModel>();
        IPatientenOverzichtViewModel model =
              this.factory.CreatePatientenOverzichtViewModel();
        model.Patienten = null;

        if (count == 0 && message == null)
            ViewBag.Message = "Geen resultaten gevonden voor deze
                              zoektermen";
        else
        {
            foreach (IPatient patient in patienten)
            {
                IPatientViewModel patientModel =
                      this.factory.CreatePatientViewModel(patient);
                models.Add(patientModel);
            }
            model.Page = index;
            model.Patienten = new CustomIPagedList<IPatientViewModel>(
                  models, index, NUMBEROFPATIENTSPERPAGE, count);
            model.Dag = this.FactorYearOrDay(dag);
            model.Maand = this.FactorMonth(maand);
            model.Jaar = this.FactorYearOrDay(jaar);
            model.Naam = naam;
            model.OverledenFiltered = overledenFiltered;
            ViewBag.Message = message;
        }
        return View(model);
    }

    // GET: /Patient/Details/1
    public ActionResult Details(int id)
    {
        if (!this.protection.CheckIfUserIsLinkedToPatient(id))
            return RedirectToAction("UnauthorizedAttempt",
                   "EidAuthentication");
        int? user_id = this.protection.getCurrentUser();
        ViewBag.PatientName = this.protection.getPatientName(id);
        ViewBag.PatientId = id;

        IPatient patient = patientRepository.getDetails(id, (int)user_id);

        IPatientViewModel patientModel =
              this.factory.CreatePatientViewModel(patient);
        return View(patientModel);
    }
```

```
        private string getFeedbackFromSearchEngine(string dag, string maand,
              string jaar)
        {
            string message = "";

            int test;
            if (!string.IsNullOrEmpty(dag) &&
                !(Int32.TryParse(dag, out test) && test > 0 && test < 31))
                message += "Verkeerde dag van de maand ingegeven <br />";

            int? factoredMaand = FactorMonth(maand);
            if (!string.IsNullOrEmpty(maand) &&
                !(factoredMaand != null &&
                factoredMaand > 0 &&
                factoredMaand < 13))
            {
                message += "Verkeerde maand ingegeven <br />";
            }

            if (!string.IsNullOrEmpty(jaar) &&
                !(Int32.TryParse(jaar, out test) &&
                test > 1800))
            {
                message += "Verkeerd jaartal ingegeven <br />";
            }

            if (message.CompareTo("") == 0)
                return null;
            else
                return message;
        }
    }
```

11.2.10 IPatientRepository

```
    public interface IPatientRepository
    {
        System.Collections.Generic.IList<Eva.Models.IPatient>
              GetPatientenOverviewPage(string naam, int? dag, int? maand, int?
              jaar, bool? overleden, int pagesize, int page, int dokter_id);
        Eva.Models.IPatient getDetails(int id, int dokter_id);
        int GetCount(string naam, int? dag, int? maand, int? jaar, bool?
              overleden, int dokter_id);
        string getNameById(int id);
    }
```

11.2.11 PatientRepository

```
    public class PatientRepository : IPatientRepository
    {
        private EvaDBContext evaDBContext;

        public PatientRepository()
        {
            this.evaDBContext = new EvaDBContext();
        }
```

```
public IList<IPatient> GetPatientenOverviewPage(string naam, int? dag,
      int? maand, int? jaar, bool? overledenFiltered, int pagesize, int
      page, int dokter_id)
{
    var query = from x in this.evaDBContext.Patienten select x;

    if (!String.IsNullOrEmpty(naam))
    {
        string[] splittedname = naam.Split(' ');
        foreach (string namePart in splittedname)
        {
            query =  query.Where(x => x.Naam.Contains(namePart) ||
                     x.Voornaam.Contains(namePart));
        }
    }
    if (dag != null)
    {
        query =  query.Where(x => x.Geboortedatum != null &&
                 ((DateTime)x.Geboortedatum).Day == dag);
    }
    if (maand != null)
    {
        query =  query.Where(x => x.Geboortedatum != null &&
                 ((DateTime)x.Geboortedatum).Month == maand);
    }
    if (jaar != null)
    {
        query =  query.Where(x => x.Geboortedatum != null &&
                 ((DateTime)x.Geboortedatum).Year == jaar);
    }
    if (overledenFiltered == true)
    {
        query =  query.Where(x => x.Status_code.CompareTo("OV") != 0 ||
                 x.Status_code  == null);
    }

   return  query.OrderByDescending(x => x.ID).Skip(pagesize * page -
           pagesize).Take(pagesize).ToList<IPatient>();
}

public IPatient getDetails(int id, int dokter_id)
{
    var query = from x in this.evaDBContext.Patienten select x;
    query = query.Where( x =>x.ID == id);

    List <IPatient> patient = query.ToList<IPatient>();
    if (patient.Count > 0)
        return patient[0];
    else
        return null;
}

public int GetCount(string naam, int? dag, int? maand, int? jaar, bool?
      overleden, int dokter_id)
{
    var query = from x in this.evaDBContext.Patienten select x;
```

```
            if (!String.IsNullOrEmpty(naam))
            {
                string[] splittedname = naam.Split(' ');
                foreach (string namePart in splittedname)
                {
                    query =  query.Where(x => x.Naam.Contains(namePart) ||
                             x.Voornaam.Contains(namePart));
                }
            }
            if (dag != null)
            {
                query =  query.Where(x => x.Geboortedatum != null &&
                         ((DateTime)x.Geboortedatum).Day == dag);
            }
            if (maand != null)
            {
                query =  query.Where(x => x.Geboortedatum != null &&
                         ((DateTime)x.Geboortedatum).Month == maand);
            }
            if (jaar != null)
            {
                query =  query.Where(x => x.Geboortedatum != null &&
                         ((DateTime)x.Geboortedatum).Year == jaar);
            }
            if (overleden == true)
            {
                query =  query.Where(x => x.Status_code.CompareTo("OV") != 0 ||
                         x.Status_code == null);
            }
            return query.Count();
        }

        public string getNameById(int id)
        {
            string name = "";

            var query =  from x in this.evaDBContext.Patienten
                         select new {x.ID,x.Naam, x.Voornaam};
            query = query.Where(x => x.ID == id);

            foreach (var item in query)
            {
                name = item.Naam + " " + item.Voornaam;
            }
            return name;
        }
    }
```

11.2.12 PagedList:

```
    public class CustomIPagedList<T> : PagedList.IPagedList<T>
    {

        private IEnumerable<T> myList;

        public CustomIPagedList(IEnumerable<T> myList, int pageNumber, int
              pageSize,int max)
        {
            this.myList = myList;
```

```
            this.PageNumber = pageNumber;
            this.PageSize = pageSize;
            this.TotalItemCount = max;
            this.Count = max;

            if ( (max%pageSize) == 0)
                this.PageCount = max/pageSize;
            else
                this.PageCount = (max/pageSize) + 1;

            if (pageNumber == 1)
            {
                this.HasPreviousPage = false;
                if (PageCount > 1)
                {
                    this.IsFirstPage = true;
                    this.HasNextPage = true;
                }
                else
                    this.HasNextPage = false;
            }
            else
                this.HasPreviousPage = true;

            if (pageNumber == this.PageCount)
            {
                this.IsLastPage = true;
                this.HasNextPage = false;
            }
            else if ( pageNumber != 1 )
                this.HasNextPage = true;
        }

        …
    }
```

12 Literatuuropgave en referenties

[FI130516] fedict, *The Electronic Identity Card (EID) – Developers Guide*, [online] available http://eid.belgium.be/en/binaries/UPD_Developers_Guide_tcm406-112228.pdf , Downloaded 16/05/2013

[HT111219] Troy Hunt, *OWASP Top 10 for .NET developers*, [online] available https://asafaweb.com/OWASP%20Top%2010%20for%20.NET%20developers.pdf , Release 19/12/2011

[MS121115] Microsoft, *How To Implement Forms-Based Authentication in Your ASP.NET Application by Using C#.NET*, [online] available http://support.microsoft.com/kb/301240 , Last Review 15/11/2012

[MS130507] Microsoft, *ASP.NET Web Application Security*, [online] available http://msdn.microsoft.com/en-us/library/330a99hc(v=vs.100).aspx , Downloaded 07/05/2013

[MS130508] Microsoft, *Walkthrough: Using Forms Authentication in ASP.NET MVC*, [online] available http://msdn.microsoft.com/en-us/library/ff398049(v=vs.100).aspx , Downloaded 08/05/2013

[MS130522] Microsoft, *How to: Add Installers to Your Service Application*, [online] available http://msdn.microsoft.com/en-us/library/ddhy0byf(v=vs.110).aspx , Downloaded 22/05/2013