



Master of Science Industrial Science : Informatics
Academic year 2013-2014

VOP Project Dossier

Stambomen

Submitted on XX XXXX 2014

Students:

Kenzo Clauw
Axl François
Lowie Huyghe
Sander Trypsteen
Jelle Verreth

Master of Science Industrial Science : Informatics
Academic year 2013-2014

VOP Project Dossier

Stambomen

Submitted on XX XXXX 2014

Student:

Kenzo Clauw

Axl François

Lowie Huyghe

Sander Trypsteen

Jelle Verreth

Abstract

Contents

1	Voorwoord	4
2	Introductie en situering	5
2.1	Introductie	5
2.2	Technische	6
2.3	Situering	7
3	Taak-verdeling	8
3.1	Methodologie	8
3.2	Sprint 1	10
3.2.1	Kenzo Clauw	10
3.2.2	Axl François	10
3.2.3	Lowie Huyghe	10
3.2.4	Sander Trypsteen	10
3.2.5	Jelle Verreth	10
3.3	Sprint 2	10
3.3.1	Kenzo Clauw	10
3.3.2	Axl François	10
3.3.3	Lowie Huyghe	10
3.3.4	Sander Trypsteen	10
3.3.5	Jelle Verreth	10
3.4	Sprint 3	10
3.4.1	Kenzo Clauw	10
3.4.2	Axl François	10
3.4.3	Lowie Huyghe	10
3.4.4	Sander Trypsteen	10
3.4.5	Jelle Verreth	10
3.5	Overzicht	10
4	Analyse	11

5	Interresantee Ontwerpbeslissingen	12
6	Interessante Implementatiekeuzes	13
6.1	Bibliotheken	13
6.1.1	Jersey	13
6.1.2	Swagger Jersey	13
6.1.3	SLF4J	14
6.1.4	Junit	14
6.1.5	Gedcom	15
6.1.6	Gedcom4j	17
6.1.7	Sardine	18
6.1.8	RestFB	18
6.1.9	JCalendar	18
6.1.10	Abego TreeLayout	18
6.1.11	Java FX	19
7	Kwaliteitscontrole	20
8	Gekende problemen	21
9	Verbeterpunten en uitbreidingsmogelijkheden	22
10	Nabeschouwing en besluit	23
	List of Figures	24
	List of Tables	25

Chapter 1

Voorwoord

Dit projectdossier is een schriftelijk neerslag van onze prestaties voor het vakoverschrijvend project¹. Hierin bespreken we de verschillende facetten van ons project. Allereest zullen wij beginnen met de opdracht voor te stellen en ze te situeren. Hierbij zullen we proberen van een zo volledig mogelijk overzicht te geven van alle technische en niet-technische onderdelen. Vervolgens krijgt u meer uitleg over de manier waarop onze takenverdeling tot stand komt en het volledig overzicht van onze takenverdeling. Daarna bespreken we de analyse van het project aan de hand van enkele diagrammen en use cases. Vervolgens lichten we enkele belangrijke aspecten van ons project toe. We gaan hier zowel in op ontwerpbeslissingen als op implementatiekeuzes. Dan komen we aan de kwaliteitscontrole voldoet onze applicatie aan de functionele en niet-functionele vereisten zoals deze zijn opgesteld door de klant? Uiteraard zijn er in elk project wel problemen en we zullen het niet na laten om deze te bespreken. Als voorlaatste geven we nog even onze visie op het verdere verloop van de applicatie. Waar kan het beter, welke uitbreidingen zijn mogelijk. Ten slotte eindigen we met een nabeschuiving en ons besluit over het VOP.

¹VOP

Chapter 2

Introductie en situering

2.1 Introductie

In deze paper kan u meer informatie vinden rond ons vop. Het onderwerp zoals u wellicht al gelezen hebt op het titel blad zijn stambomen. Het woord stambomen kan in verschillende contexten bekeken worden. We beperken ons echter enkel tot de genealogie. In genealogie zitten twee Griekse woorden verborgen namelijk Genea en Logos. Genea staat voor afkomst of afstamming en logos betekent wetenschap of kennis. Dus we bestuderen hier de afkomst, afstamming van een persoon. In dit project zijn er echter twee beperkingen:

Zonder dieper in te gaan op het onderwerp genealogie willen ook nog melden dat in dit project we ons slechts zullen beperken tot natuurlijk mogelijke afstammelingen. Hieronder verstaan wij dat er enkel man - vrouw relaties mogelijk zijn. We houden dus geen rekening met man - man of vrouw - vrouw relaties die dan kinderen zouden hebben.

Verder houden we ook geen rekening met de onderlinge relaties tussen echt-paren. Op zich zijn dat onze zaken niet maar wanneer een persoon een buitenechtelijke kind heeft dan zal hij dat niet kunnen invoeren in ons programma. Hieronder vallen dus ook half-broers of half-zussen. Deze zullen niet weergegeven worden in ons stamboom overzicht.

Nu de scope van de opdracht duidelijker is kunnen ingaan op wat er gevraagd is. Het is de bedoeling om een applicatie te maken waar een gebruiker stambomen kan ingeven en bekijken. De gebruiker moet deze stambomen kunnen raadplegen via een website en de bomen worden ingegeven via een desktop applicatie. Verder moet de gebruiker ook zijn bomen kunnen delen met zijn vrienden zowel binnen de applicatie als op sociale netwerk sites. Voor dit project zullen we ons qua sociale netwerk sites beperken tot Facebook.

2.2 Technische

We hebben op 10/02/2014 de inleidende presentatie van het vop gekregen. Hierin werden een aantal vereisten uitgelegd. Deze vereisten staan uitgebreid beschreven in VOP: de richtlijnen ¹. De belangrijkste hierin zijn dat er een docent de rol van klant zal spelen. Er zullen afspraken moeten worden gemaakt met de klant over wat al dan niet mogelijk is. Wat er moet gerealiseerd worden? We kwamen ook te weten dat het project in Java moest geschreven worden. Verder moest dit gebeuren door aan de hand van een drielagen architectuur.

Presentatie laag Deze laag bestond uit twee onderdelen namelijk een Java Desktop Client gemaakt in Java Swing en een website (HTML5, Javascript) die gebruikt maakt van Java Servlets. De nadruk uiteraard ligt hier bij het scheiden van onze logica van de presentatie.

Domein logica Hierbij was het de bedoeling om een REST-service in Java op te zetten waarvan de presentatie gebruik kan maken. We hebben hier van in het begin gekozen om Jersey te gebruiken.

Data tier Een relationele databank ging in staan voor de persistentie van onze gegevens. De databank die ons aangeboden werd door UGent is MySQL. We mochten in deze opdracht geen gebruik maken van ORM-tools².

¹ADD REFERENCE

²Object-Relational Mapping zoals Hibernate.

2.3 Situering

WAT?

Chapter 3

Taak-verdeling

3.1 Methodologie

Tijdens dit project zullen we gebruik maken van Scrum. Hierbij verdelen we het project in drie sprints. Aan het begin van elke sprint zitten we samen om te bespreken welke features we graag zouden opnemen. Hoe lang een bepaalde feature duurt om uit te werken. Na deze bespreking maken we een afspraak bij de klant.

Met de klant bespreken we dan welke features voor ons mogelijk zijn en welke features voor de klant noodzakelijk zijn. Hieruit volgt een contract, alle features die we tijdens een bepaalde sprint zullen realiseren.

Op het einde van de sprint kijken we dan welke features we gerealiseerd hebben en welke niet. Dankzij deze methode kunnen we inschatten of we tijdens het project moeten bijsturen. Het kan zijn dat we niet alle features gerealiseerd hebben dan moeten we harder werken tijdens de volgende sprint.

Tijdens elke sprint krijgen we van de docenten een voorgestelde lijst met features.

3.2 Sprint 1

3.2.1 Kenzo Clauw

3.2.2 Axl François

3.2.3 Lowie Huyghe

3.2.4 Sander Trypsteen

3.2.5 Jelle Verreth

3.3 Sprint 2

3.3.1 Kenzo Clauw

3.3.2 Axl François

3.3.3 Lowie Huyghe

3.3.4 Sander Trypsteen

3.3.5 Jelle Verreth

3.4 Sprint 3

3.4.1 Kenzo Clauw

3.4.2 Axl François

3.4.3 Lowie Huyghe

3.4.4 Sander Trypsteen

3.4.5 Jelle Verreth

3.5 Overzicht

Chapter 4

Analyse

Chapter 5

Interresantee Ontwerpbeslissingen

Chapter 6

Interessante Implementatiekeuzes

6.1 Bibliotheken

6.1.1 Jersey

Jersey is een JAX-RS referentie-implementatie die een eigen API aanbied door de JAX-RS toolkit uit te breiden met extra functies en hulpprogramma's om op een eenvoudige manier RESTful Webservices en client development aan te bieden. JAX-RS : Java API voor REST Web Services biedt ondersteuning bij het creëren van web services volgens het Representational State Transfer (REST) pattern. REST is een pattern die beschrijft hoe resources geadresseerd en gebruikt kunnen worden. Een resource kan aangesproken worden dankzij een gemeenschappelijke interface op basis van de HTTP standaard methodes. Een REST server zorgt ervoor dat de client een verbinding kan maken om de resources op te halen en te wijzigen. REST wordt veel gebruikt voor het bouwen van webservices en noemen we respectievelijk RESTful Webservices. Voor de applicatie hebben we gebruik gemaakt van Jersey die zorgt voor de REST server en een REST client.

6.1.2 Swagger Jersey

Swagger is een specificatie en compleet uitvoeringskader voor het beschrijven, het produceren , consumeren en visualiseren van REST webservices. Het doel van Swagger om client en documentatie systemen de mogelijkheid te geven

op hetzelfde tempo te werken als de server. De documentatie van de methoden, parameters en modellen zijn nauw geïntegreerd in de server-code, waardoor de API's altijd gesynchroniseerd zijn.

Door gebruik te maken van Swagger kunnen clients diensten consumeren en toegang tot de server code zonder in aanraking te komen de implementatie van de server. De interface van het framework zorgt ervoor dat er interactie mogelijk is met de API in een sandbox omgeving. Swagger ondersteunt JSON en XML en zal in de toekomst ook beschikbaar zijn in andere formaten.

6.1.3 SLF4J

SLF4J is een Java logging api die gebruik maakt van een facade pattern, dit is een software design pattern die gebruikt wordt in object-oriented programming om een complex systeem voor te stellen als een interface. De facade pattern is ideaal bij het werken met een groot aantal onderling afhankelijke klassen of klassen die meerdere methoden gebruiken, vooral wanneer deze te ingewikkeld zijn om te gebruiken of moeilijk te begrijpen. De onderliggende logging backend van SLF4J wordt bepaald op runtime door het toevoegen van de gewenste binding aan het classpath (java.util.logging). Om gebruik te maken van SLF4J moet je de slf4j-api-1.7.7.jar en slf4j-simple-1.7.7.jar plaatsen in de classpath van het project. Bij het volgende voorbeeld roepen we de loggerfactory op om dan vervolgens het toevoegen van een persoon te registreren :

```
private final Logger logger = LoggerFactory.getLogger(getClass());

public Person getPerson(int treeID, int personID)
{
    logger.info("[PERSON CONTROLLER] Getting person by id " + personID);
    return pc.getPerson(treeID, personID);
}
```

6.1.4 Junit

JUnit is een unit testing framework voor Java die gebruikt wordt in test-driven development die deel uit maakt van xUnit. Een unit test is een stukje code die een specifieke functionaliteit uitvoert om de code testen. Het percentage van de

code die wordt getest door unit tests wordt meestal de test coverage genoemd. Een unit test richt zich op een kleine eenheid van de code , bv een methode of een klasse.

6.1.5 Gedcom

GEDCOM is een speciaal tekstformaat die ontwikkeld is door de Kerk van Jezus Christus van de Heiligen der Laatste Dagen en is bedoeld als standaard voor communicatie tussen de Kerk en personen die genealogische data aanleveren. Dit formaat heeft zich nu ontwikkeld tot de standaard voor gegevensuitwisseling tussen de meeste genealogische programma's en systemen.

Stel dat we in de applicatie een persoon toevoegen dan zouden deze gegevens in een GEDCOM als volgt voorgesteld worden :

Detail	
Voornaam	Kenzo
Achternaam	Clauw
Geslacht	<input checked="" type="radio"/> Man <input type="radio"/> Vrouw
Geboortedatum	15-mrt-1991
Overlijdingsdatum	
Adres	
Gemeente	Oostende
Postcode	8400
Land	Belgie

Het eerste deel bevat de header met de gebruikte versie,character encoding (ANSI,UNICODE of ASCII) als de belangrijkste informatie.

6.1. BIBLIOTHEK CHAPTER 6. INTERESSANTE IMPLEMENTATIEKEUZES

```
0 HEAD
1 SOUR naam
2 VERS V4.0
1 DEST naam
1 DATE 10 MAY 2014
1 FILE C:\Users\admin\gedcom.ged
1 GEDC
2 VERS 5.5
1 CHAR ANSI
```

Het tweede deel bevat informatie over de personen.

```
0@I1@ INDI
1 NAME Kenzo/Clauw
1 SEX M
1 BIRT
2 DATE 15 MAY 1991
2 PLAC Oostende
1 FAMS @F1@

0@I2@ INDI
1 NAME Manon/Jonckheere
1 SEX F
1 BIRT
2 DATE 27 FEB 1992
2 PLAC Oostende
1 FAMS @F1@
```

Het laatste deel bevat de relaties tussen personen met TRLR die het bestand afsluit.

```
0 @F1@ FAM
1 HUSB @I1@
1 WIFE @I2@
0 TRLR
```

6.1.6 Gedcom4j

Gedcom4j is een gratis open-source Java library voor het laden (parsing) en opslaan van gegevens in Gedcom genealogie 5.5 of 5.51 bestanden naar een Java-object hiërarchie.

Om gebruik te maken van Gedcom4j moet je de gedcom4j.jar plaatsen in de classpath van het project.

Om toegang tot gegevens te verkrijgen maak je gebruik van de properties binnen de gp.gedcom structure waarbij de personen voorgesteld worden door het object Individual en de relaties door Family.

Laden van een gedcom bestand :

```
GedcomParser gp = new GedcomParser(); gp.load("sample/TGC551.ged");
```

Om de personen te overlopen :

```
for (Individual i : g.individuals.values()){
    System.out.println(i.formattedName());
}
```

Om de relaties met kinderen te overlopen :

```
for (Family f : g.families.values()){
    f.wife.formattedName();
    f.husband.formattedName();
    for (Individual c : f.children)
    {
        c.formattedName();
    }
}
```

6.1.7 Sardine

6.1.8 RestFB

6.1.9 JCalendar

6.1.10 Abego TreeLayout

ABEGO TreeLayout is een Efficiënt en aanpasbare Boom Layout algoritme voor Java. De TreeLayout creëert boom lay-outs voor willekeurige bomen. Het is niet beperkt tot een bepaalde productie of formaat, maar kan worden gebruikt voor elke vorm van tweedimensionale tekening. Voorbeelden zijn Swing gebaseerde componenten, SVG-bestanden, en nog veel meer.

Om de Treelayout te gebruiken moet je een instantie van de klasse TreeLayout voorzien van de knooppunten van de boom inclusief zijn kinderen samen met de hoogte tussen verschillende niveaus.

Eigenschappen : Op basis van deze informatie zorgt de TreeLayout voor een compacte layout met een overzichtelijke boom.

De indeling toont de hiërarchische structuur van de boom, namelijk de y-coördinaat van een knooppunt wordt bepaald door het niveau.

De randen kruisen elkaar niet en de nodes op hetzelfde niveau hebben een minimale horizontale afstand. De volgorde van de kinderen van een knooppunt wordt weergegeven in de tekening.

Het algoritme werkt symmetrisch, dat wil zeggen de tekening van de weerspiegeling van een boom is het gereflecteerde tekening van de oorspronkelijke boom

Om gebruik te maken van de Abego TreeLayout moet je de org.abego.treelayout.core.jar(de TreeLayout algoritme kern) en org.abego.treelayout.netbeans.jar(gebruikt van de NetBeans visuele API) plaatsen in de classpath van het project.

6.1.11 Java FX

Chapter 7

Kwaliteitscontrole

Om de kwaliteit van onze applicatie te garanderen wordt er gebruik gemaakt van Test-driven development die een onderdeel is van agile softwareontwikkeling.

Hoe werkt Test-driven development?

Voor dat er code geschreven wordt maak je eerst een geautomatiseerde test waarbij je rekening moet houden met alle mogelijkheden van invoer , errors en uitvoer. Op deze manier moet je nog geen rekening houden met code. De eerste keer dat een test uitgevoerd wordt moet deze een error produceren omdat er nog geen code aanwezig is. Na het afwerken van de test is het de bedoeling om op basis van de tests code te schrijven. Eens de code met succes de verschillende testen kan doorstaan kun je bepaalde bugs uit je applicatie al uitsluiten.

Wij hebben ervoor gekozen om aan de hand van JUnit de opgestelde use cases om uitgebreid te testen in onze applicatie aan de hand van een JUnit test suite.

Wat moet er getest worden?

In het algemeen is het veilig om bepaalde methoden zoals getters en setters die gewoon waarden oproepen of toewijzen te negeren. Het schrijven van tests hiervoor is tijdrovend en zinloos omdat je op deze manier de Java Virtual Machine zou testen waarvoor er al testen voorzien zijn.

Chapter 8

Gekende problemen

Chapter 9

Verbeterpunten en uitbreidingsmogelijkheden

Chapter 10

Nabeschouwing en besluit

List of Figures

List of Tables