Ontwikkelen van een project management systeem
Afstudeerverslag

Gijs Meuldijk
20058100

versie 1.00

Referaat

Project management systeem
Urenregistratie
Rational Unified Process
User-centered Design
Unified Modeling Language
MySQL
PHP
AJAX
Zend Framework
REST
JSON
Web-applicatie
Klassediagram
Sequentiediagram
Interactie ontwerp

Voorwoord

Dit verslag is het resultaat van mijn afstudeeropdracht en vormt tevens de afronding van mijn studie informatica.

Ik wil mijn afstudeerbegeleider Bjorn van Spengen bedanken voor de begeleiding die hij heeft gegeven tijdens dit project. Naast mijn begeleider wil ik ook de medewerkers van Vurig bedanken voor hun feedback. Ook wil ik Alwine Lousberg en Cobie van der Hoek bedanken voor hun begeleiding van uit school.

Gijs Meuldijk

Inhoudsopgave

1 Inleiding

Dit afstudeerverslag heeft als doel inzicht(en) te geven in het afstudeerproces. In dit verslag zal naar voorkomen op welke manier er gewerkt is, welke keuzes er zijn gemaakt ook waarom deze keuzes zijn gemaakt.

In het tweede hoofdstuk wordt opdracht omschreven. Hierin komt ondermeer de beschrijving van het bedrijf, de organisatie, de aanleiding, de probleemstelling en de doelstelling naar voren.

In het derde hoofdstuk zal de inceptie fase aan bod komen. Hier staat ondermeer in met welke methoden en technieken ik heb gewerkt. Verder zijn hier ook de planning en requirements terug te vinden.

In het vierde hoofdstuk worden de activiteiten van de elaboratie fase beschreven. Hier zullen de use-case, het klassediagram, het implementatiemodel, het relationeel representatiemodel, de sequentiediagrammen en ten slotte het interactie ontwerp aan bod komen.

Het vijfde hoofdstuk omvat de beschrijving van alle activiteiten die ik tijdens de constructie fase heb uitgevoerd. Hier wordt uitgelegd hoe de webservice en de front-end zijn gebouwd. Ook bevat het de beschrijving van het testen.

Ten slotte wordt in het zesde hoofdstuk zowel het proces als product geëvalueerd.

2 Opdrachtomschrijving

2.1 Bedrijf
Vurig Media is een klein bedrijf dat zich bezig houdt met het bedenken, ontwerpen en ontwikkelen van interactieve oplossingen, die zowel grafisch als technisch geavanceerd zijn. Onder interactieve oplossingen wordt verstaan de interactie tussen de gebruiker en het systeem. Er worden back-ends ontwikkeld die veel functionaliteiten hebben maar daar tegenover staat dat er wordt gestreefd naar een zo simpel mogelijk gehouden front-end. Zoals bijvoorbeeld hun eigen ontwikkelde content management systeem (CMS). Dit CMS is zo generiek mogelijk opgebouwd dat het makkelijk te gebruiken is bij sites die verschillende maatwerken eisen hebben.

2.2 Organisatie
Vurig is een vrij kleine organisatie met drie medewerkers. Deze medewerkers zijn alle drie eigenaar van het bedrijf. De medewerkers hebben allemaal hun eigen specialisatie. Zo is er een ontwerper, flasher, ontwikkelaar. De plaats van de student valt onder de ontwikkelaar. Dit geldt voor 90% van het project voor de andere 10% zal de student onder de ontwerper vallen.

2.3 Aanleiding
Het bedrijf heeft geen volledig project management systeem (PMS) voor handen. Er zijn wel verschillende PMS-en op de markt beschikbaar maar deze hebben of teveel overbodige functies, of er ontbreekt iets. Zodoende wil het bedrijf een eigen maatwerk PMS bouwen.

Er is inmiddels wel al een basisstructuur ontwikkeld voor dit PMS maar deze moet verder worden uitgebreid met verschillende functionaliteiten. De basisstructuur bestaat uit niets meer dan een simpele urenregistratie voor de werknemers van Vurig. Ze kunnen hun uren per project opslaan. Door registratie van uren en een planning is er meer grip op projecten te krijgen en kan er in geval van uitloop of calamiteiten eerder worden ingegrepen. Verder biedt het de mogelijkheid om meer controle te hebben op het budget van een project.

Het is de bedoeling dat de student nieuwe functionaliteiten zal gaan toevoegen aan het bestaande systeem.

2.4 Probleemstelling
De probleemstelling bestaat uit twee delen.

Ten eerste kunnen de medewerkers van Vurig alleen hun uren invoeren, verder is er geen functionaliteit aanwezig in het PMS.

Ten tweede is er nog geen gebruiksvriendelijke interface beschikbaar voor het systeem.

2.5 Doelstelling

De doelstelling bestaat ook uit twee delen.

De doelstelling van deze opdracht is ten eerste het verder uitbreiden, ontwerpen, ontwikkelen en testen van de back-end van het PMS. Zodat Vurig hun project makkelijker kan gaan plannen en zodoende meer grip krijgt op hun projecten.

Ten tweede zal er voor de front-end een interactie ontwerp gemaakt worden, de bedoeling hiervan is dat er wordt nagedacht over de interactie met de gebruiker. Het is namelijk nog niet bekend op welke manier de gebruiker gebruik gaat maken van het PMS. De enige eis is dat het PMS het makkelijk en snel inplannen van projecten mogelijk maakt.

De nieuwe functionaliteiten die in het systeem mogelijk moeten zijn:
(gesorteerd naar prioriteit)

Planning maken
Het moet mogelijk zijn om een planning te maken. In deze planning kunnen ze werknemers en het aantal uur wat de werknemers tot hun beschikking hebben indelen per project. De werknemers kunnen zowel bij Vurig Media in dienst zijn als extern personeel zijn.

Het moet mogelijk zijn om alleen werknemers die beschikbaar zijn voor een project toe te wijzen. Dus indien iemand bijvoorbeeld op vakantie is dan mag het niet zo zijn dat diegene wel is toegewezen aan project op het moment dat hij/zij afwezig is.

De planning wordt ook op basis van het budget gemaakt. Als er meer uren zijn ingepland dan het budget toelaat moet het PMS hiervan een melding geven. Ook hebben ontwikkelaars en designers een eigen groep en hebben ze dus ook een eigen budget tot hun beschikking.

Autoriseren
Om het systeem te kunnen gebruiken, is het een vereiste dat de de gebruikers eerst in moeten loggen. Zodra ze zijn ingelogd zien ze een overzichtspagina met hun projecten, de activiteiten voor de komende dag en to-do’s.

Klanten login
Het moet voor projectleiders van de klant mogelijk zijn om in te loggen om te zien hoe het met project ervoor staat. Ze komen op een overzichtspagina terecht waar ze hun project(en) zien en ze hebben de mogelijkheid om tussen- en eindproducten te bekijken.

Het kan bijvoorbeeld ook mogelijk zijn dat er bij een klant meerdere projecten lopen en dat er verschillende projectleiders zijn toegewezen. Deze krijgen dan alleen inzicht in hun eigen project.

Factureren
Het zal in het PMS mogelijk moeten zijn om facturen op te stellen. Deze zullen als pdf moeten worden gegenereerd op basis van een template van Vurig. Deze facturen worden opgesteld door een projectleider. Deze moet ook inzicht kunnen krijgen of er een factuur nog openstaat.

Ik ben tot deze volgorde gekomen naar aanleiding van gesprekken die ik had met mijn opdrachtgever. In deze gesprekken kwam naar voren dat het belangrijkste gedeelte van het systeem het maken van planningen is. Ze willen een (centraal) systeem dat inzicht geeft in hun projecten en de daarbij behorende planningen. Omdat zoals in de aanleiding beschreven staat, meer grip te krijgen op (toekomstige) projecten.

Het autoriseren is daarna het meest belangrijkst voor Vurig omdat er nu verschil gemaakt kan worden in wat projectleiders en medewerkers in kunnen zien. Zoals bijvoorbeeld hun eigen activiteiten voor de komende dag en persoonlijke to-do's.
Nadat het voor intern personeel mogelijk is om in te loggen wordt het pas een prioriteit om dit ook voor klanten mogelijk te maken.

Als laatste heeft het factureren prioriteit. In de gesprekken kwam dit naar voren als minst belangrijke eis. Omdat opdrachten vaak toch voor een vast bedrag worden gefactureerd, is dit voor mijn opdrachtgever het minst van belang.

Deze eisen zijn vooraf geformuleerd naar aanleiding van gesprekken die ik heb gehad met Vurig. Deze eisen zijn ook terug te vinden in het afstudeerplan. De gedetailleerde eisen worden beschreven in de inceptie fase.

3 Inceptie fase
De inceptie fase is de fase waarmee het project begonnen is.

3.1 Aanpak
In dit hoofdstuk beschrijf ik de methoden en technieken die ik heb gebruikt voor het project. Hierna volgt de planning.

3.1.1 Methoden
Voor het project heb ik gebruik gemaakt van de methoden Rational Unified Process (RUP) en User-centered Design (UCD).

Rational Unified Process
Ik heb voor RUP gekozen omdat ik vond dat deze methode het beste aansluit bij het project. Een van de voordelen van RUP is dat het ontwerp zich richt op een component gerichte architectuur. Dit was voor mij de belangrijkste reden om voor RUP te kiezen aangezien ik het systeem met losse componenten (modulair) ga opbouwen. Ik ga namelijk eerst de module planning maken bouwen. Zie ook de eerder beschreven functionaliteiten in hoofdstuk 2.5.

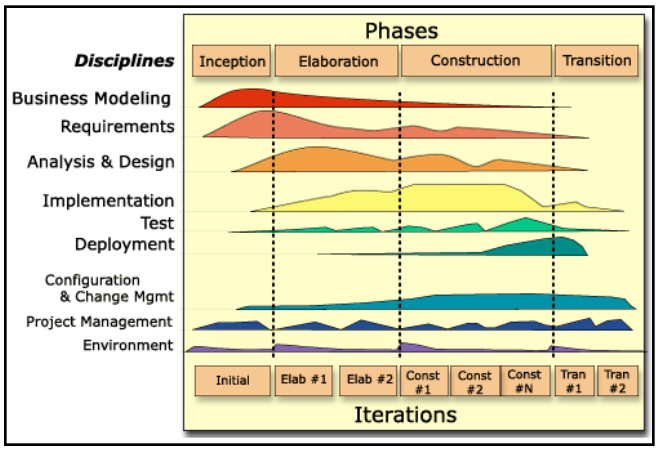
Naast de component gerichte architectuur heeft RUP ook nog andere voordelen zoals:

- Iteratief ontwikkelen
- Visueel modelleren
- Continu testen (fouten komen eerder aan het licht)

In eerste instantie wilde ik eigenlijk gebruik maken van eXtreme Programming (XP) maar omdat er bij XP veel minder gedocumenteerd wordt leek het me niet verstandig in verband met bewijsvoering.

Nadeel van RUP kan zijn dat je (te) veel aan het ontwerpen blijft t.o.v. XP. Om dit te voorkomen kan ik bijvoorbeeld tijdens elaboratie fase vaststellen hoeveel ontwerpen ik nodig heb om het systeem te kunnen bouwen. [1]

RUP kent 4 fases namelijk de inceptie, elaboratie, constructie en transitie fase. Er zal tijdens het afstuderen alleen gebruik worden gemaakt van de eerste drie fases. De transitie fase is de fase waarin het systeem in de productie omgeving terecht komt. Dit zal ik niet uitvoeren omdat alleen de beheerder van de systemen van Vurig hiervoor bevoegdheid heeft. Hoe het opgeleverd wordt zal ik mondeling overleggen met de beheerder.



Figuur 3.1 *De fases van RUP*

Het eerder genoemde voordeel het iteratief ontwikkelen zal ik naar mijn verwachting als volgt gaan toepassen:

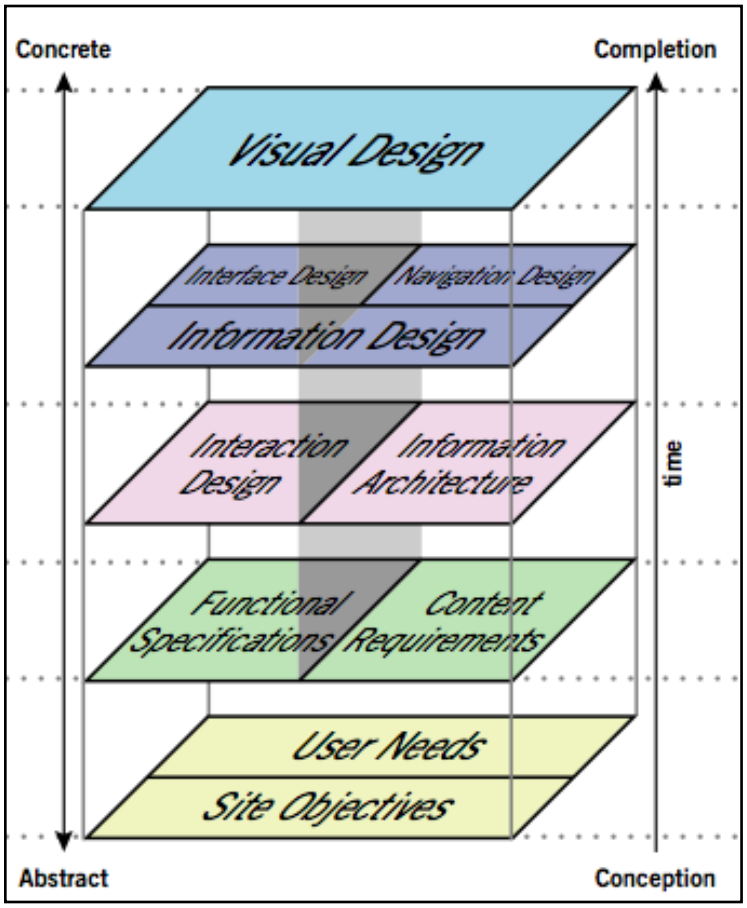
1. Het opleveren van een deelproduct
2. Feedback opdrachtgever
3. Feedback verwerken in iteratie

User-centered Design
Voor UCD heb ik gebruik gemaakt van *"The Elements of User Experience"* zoals dit beschreven is door Jesse James Garrett. Om twee redenen heb ik gekozen voor UCD. De eerste en de belangrijkste reden is dat deze methode diep genoeg ingaat op de verschillende fasen van het ontwerp traject, van het vaststellen van de eisen tot het grafische ontwerp.

De tweede reden is een persoonlijke reden. Deze methode heeft een lage instapdrempel. Dit is handig omdat dit voor mij een geheel nieuwe methode is. Hierdoor is het makkelijk(er) om de methode toe te passen. Doordat ik een beperkt aantal weken tot mijn beschikking had vond ik het belangrijk dat ik de methode snel en makkelijk kon toepassen.

Deze methode wordt naast RUP gebruikt omdat RUP zich meer richt op de back-end van systemen en niet zozeer usability. Vandaar dat er voor een methode is gekozen die zich ook meer richt op de front-end van het systeem.

In figuur 3.2 is te zien dat UCD verdeelt is in vijf fases en sommige van deze fases zijn ook nog opgesplitst. Dit omdat je het web op twee manieren kunt benaderen. Namelijk als software interface of als hypertext systeem. Software interface richt zicht op het grafische ontwerp. Dit zijn alle elementen die zich aan de linkerkant van het schema bevinden. Hypertext systeem richt zich op de content / informatie voorziening. Dit zijn alle elementen die zicht aan de rechterkant van het schema bevinden.



Figuur 3.2 *De fases van UCD*

Het verschil tussen software interface en hypertext systeem is als volgt. Bij het grafische ontwerp gaat het er vooral om welke handeling kan de gebruiker uitvoeren. Terwijl er bij het hypertext systeem wordt gekeken hoe informatie zo geordend en gepresenteerd kan worden dat het voor de gebruiker snel en makkelijk toegankelijk is.

De fases waarin UCD is verdeeld zijn:

1. Strategie
   - Vaststellen doelstellingen
2. Scope
   - Vaststellen functionele eisen
3. Structuur
   - Opstellen use-cases
4. Skelet
   - Ontwerpen van wireframes
5. Oppervlakte
   - Ontwerpen front-end systeem

Wegens tijdgebrek zal ik de front-end ontwerpen tot en met de vierde fase. Dit heeft twee redenen. Ten eerste is het niet de bedoeling dat het ontwerpen gaat overheersen, het moet wel een opdracht blijven die zicht richt op informatica. Ten tweede omdat het niet mag gaan overheersen is de vierde fase een mooi breekpunt. Op dit punt heb is het systeem vergenoeg ontworpen om inzicht te geven in het interactie ontwerp. Sommige van de activiteiten die te zien zijn bij de verdeling van de fases, komen overeen met RUP. Dit zijn activiteiten als:

- Vaststellen doelstellingen
- Vaststellen functionele eisen
- Opstellen use-cases

Het ontwerpen van de front-end zal plaatsvinden in de elaboratie fase. RUP is de leidende methode in het ontwerp en ontwikkel proces. Op de punten waar RUP te kort schiet voor het usability gedeelte, wordt er gebruik gemaakt van UCD.

3.1.2 Technieken

Unified Modeling Language
Als ondersteunde techniek voor RUP heb ik gekozen voor UML omdat RUP nauw samenwerkt met UML. Hoe ik deze techniek in de praktijk heb toegepast wordt beschreven in de elaboratie fase.

Zend Framework
Een van de eisen is dat ik het systeem bouw met behulp van het Zend Framework (ZF). ZF is een raamwerk met verschillende modules aan boord. Ook is het mogelijk om eigen extensies te schrijven voor het framework. Het ZF is zo op gesteld dat je makkelijk met het architecturaal patroon Model-View-Controller kan werken. Verdere uitleg is te vinden in de constructie fase.

Pre Hypertext Processor 5 (PHP) (Object Georiënteerd Programmeren (OOP))*
Door het gebruik van ZF moet je ook gebruik maken van PHP5 omdat er geen ondersteuning voor PHP4 is. Als er mogelijkheid was om zelf te kiezen, dan had ik sowieso ook voor PHP5 gekozen. Er zijn twee reden hiervoor. Ten eerste het PHP5 een minder volledige implementatie heeft van OOP dan PHP5. Ten tweede is er geen officiële ondersteuning meer voor PHP4.

** OOP*
Bij deze benadering wordt een systeem opgebouwd uit objecten, waarbij ieder object gemaakt is vanuit de definitie van een klasse.

MySQL5
Voor het database management systeem kon ik kiezen tussen MySQL5 en PostgreSQL. Uiteindelijk heb ik voor de eerste gekozen. Door voor- en nadelen tegen elkaar af te zetten kwam ik tot dit besluit.

MySQL5 heeft voordelen als:
- Snelheid
- Betrouwbaarheid

Nadelen:
- Geen referentiële integriteit

PostgreSQL heeft voordelen als:
- Volledige referentiële integriteit
- Betrouwbaarheid

Nadelen:
- Snelheid

Na het afzetten van de voor- en nadelen kwam ik tot de conclusie, dat het voor het project weinig verschil uit maakt of ik nu voor PostgreSQL of MySQL5 kies. Het maakt weinig verschil omdat de referentiële integriteit ook kan worden afgedwongen in het ontwikkelen van de back-end. Dus in dat geval neemt het systeem het afdwingen van referentiële integriteit over.

Daarom ben ik omdat ze bij Vurig met MySQL5 werken, voor MySQL5 gegaan.

Van de volgende technieken is ook gebruik gemaakt:

- *Asynchronous JavaScript and XML (AJAX)*
- *Javascript Object Notation (JSON)*
- *Representational State Transfer (REST)*
- *(X)HTML*
- *CSS*

In de constructie fase is beschreven op welke wijze deze technieken zijn toegepast.

3.2 Planning

Fase	Product	Startdatum	Einddatum	Tijd
Inceptie	Plan van aanpak Onderzoek Requirements	13 okt 2008	24 okt 2008	2 weken
Elaboratie	Use-cases Klassediagram Implementatiemodel Sequentiediagram	27 okt 2008	7 nov 2008	2 weken
Elaboratie	Interactie ontwerp	10 nov 2008	21 nov 2008	2 weken
Constructie	Back-end (Planning maken) en module test (plannen + uitvoeren)	1 dec 2008 15 dec 2008	5 dec 2008 19 dec 2008	2 weken
Constructie	Back-end (Autoriseren) en module test (plannen + uitvoeren)	22 dec 2008	24 dec 2008	0,5 week
Constructie	Back-end (Klanten login) en module test (plannen + uitvoeren)	25 dec 2008	31 dec 2008	1 week
Constructie	Itereren (Autoriseren) en module test (plannen + uitvoeren)	1 jan 2009	2 jan 2009	0,5 week
Constructie	Back-end (Factureren) en module test (plannen + uitvoeren)	5 jan 2009	9 jan 2009	1 week
Constructie	Front-end	19 jan 2009	23 jan 2009	1 week
Constructie	Testplan opstellen Testen (performance + systeem test)	26 jan 2009	30 jan 2009	1 week

De planning is verdeeld aan de hand van de fases van RUP. Het plannen van de constructie fase heb ik aan de hand van de eerdere gestelde prioriteiten gedaan. In overleg met mijn afstudeerbegeleider heb ik een iteratie ingepland bij de module autoriseren omdat ik verwachtte dat naar aanleiding van de klanten login het een en ander nog zou kunnen veranderen. In eerste instantie had ik ook een onderzoek ingepland. Met dit onderzoek zou ik gaan aantonen waarom de bestaande PMS-en voor Vurig niet voldeden. Dit onderzoek heb ik uiteindelijk niet gedaan omdat de uitkomst van het onderzoek al vast stond, heb ik het onderzoek achterwege gelaten.

Naderhand bleek deze planning te niet te voldoen, dit kwam pas naar voren ten tijde van de constructie fase. In de beschrijving van de constructie fase in dit verslag leg ik ook uit waarom dit zo is. Hieronder volgt de aangepaste planning.

Fase	Product	Startdatum	Einddatum	Week
Inceptie	Plan van aanpak Requirements	13 okt 2008	24 okt 2008	2
Elaboratie	Use-cases Klassediagram Implementatiemodel Sequentiediagram	27 okt 2008	7 nov 2008	4
Elaboratie	Interactie ontwerp	10 nov 2008	21 nov 2008	6
Constructie	Back-end (webservice)	1 dec 2008	5 dec 2008	8
Constructie	Back-end (webservice)	15 dec 2008	19 dec 2008	10
Constructie	Front-end	22 dec 2008	24 dec 2008	11
Constructie	Front-end	25 dec 2008	31 dec 2008	12
Constructie	Front-end	1 jan 2009	2 jan 2009	12
Constructie	Front-end	5 jan 2009	9 jan 2009	13
Constructie	Front-end	19 jan 2009	23 jan 2009	14
Constructie	Testplan opstellen Testen (performance + systeem test)	26 jan 2009	30 jan 2009	15

3.3 Requirements

Na het vaststellen van de planning ben ik de requirements gaan opstellen. In eerste instantie waren de requirements slecht meetbare requirements. Dit kwam omdat ik eerst alleen een globale omschrijving gaf van de modules die in het systeem moesten komen.

Zoals bijvoorbeeld het volgende:
“Het moet voor projectleiders van de klant mogelijk zijn om in te loggen om te zien hoe het met project ervoor staat. Ze komen op een overzichtspagina terecht waar ze hun project(en) zien en ze hebben de mogelijkheid om tussen- en eindproducten te bekijken.

Het kan bijvoorbeeld ook mogelijk zijn dat er bij een klant meerdere projecten lopen en dat er verschillende projectleiders zijn toegewezen. Deze krijgen dan alleen inzicht in hun eigen project.”

Uiteindelijk kwam ik tot de conclusie dat dit niet voldeed. Daarom heb ik de requirements als volgt opgezet. Ik heb per module beschreven wat er mogelijk moet zijn. Hieronder zijn de requirements terug te vinden. Deze zijn nu specifieker en hierdoor ook meetbaarder.

Tijdens het opstellen van de nieuwe requirements leek het me ook verstandig graphical user interface eisen op te nemen. Hierdoor komt er nu ook een overzichtspagina terug. Dit is geen module zoals planning maken. Dit is een pagina waar veel data wordt getoond die uit de andere modules wordt gelezen. De overzichtspagina is vooral gericht op de front-end van het systeem.

Deze requirements zijn tot stand gekomen door bestaande PMS-en zoals BasecampHQ en Copperproject te bekijken. Uit deze PMS-en kon ik afleiden wat er verwacht wordt van een PMS. Ook door middel van te brainstormen kwam ik op een lijstje met punten. Deze punten heb ik daarna voorgelegd aan mijn begeleider om te bepalen of Vurig het wel of niet in het systeem wilde hebben.

3.3.1 Functionele requirements

Planning maken

1. Projectleider moet een project kunnen maken
2. Projectleider moet klant kunnen koppelen aan het project
3. Projectleider moet een planning kunnen maken
4. Projectleider moet medewerkers (zowel extern als intern persoon zijn) kunnen in plannen
5. Projectleider kan alleen beschikbare medewerkers in plannen
6. Projectleider kan tussen- en eindproducten uploaden in het systeem, de klant wordt door middel van een e-mail op de hoogte gesteld.
7. Projectleider moet een budget in kunnen stellen
8. Projectleider moet milestones kunnen maken
9. Projectleider stelt een budget per onderdeel (ontwerp, video, animatie en ontwikkeling) in
10. Projectleider plant medewerkers in voor een onderdeel
11. Projectleider kan zichzelf inplannen voor een project
12. Projectleider moet projecten en planningen kunnen aanpassen
13. Projectleider moet het aantal gewerkte uren kunnen inzien

14. Projectleider moet een melding krijgen indien budget of einddatum wordt overschreden
15. Projectleider met administrator rechten moet alle projecten en planningen kunnen inzien

Uren registreren

16. Projectleider moet zijn uren kunnen registreren
17. Projectleider moet het project waar aan hij / zij gewerkt heeft kunnen registreren
18. Projectleider moet zijn uren van het project waar aan hij / zij gewerkt heeft kunnen registreren
19. Projectleider moet zijn activiteit van het project waar aan hij / zij gewerkt heeft kunnen registreren
20. Indien de projectleider zijn urenregistratie de vorige dag niet heeft uitgevoerd dan krijgt hij / zij hiervan een melding
21. Medewerker moet zijn uren kunnen registreren
22. Medewerker moet het project waar aan hij / zij gewerkt heeft kunnen registreren
23. Medewerker moet zijn uren van het project waar aan hij / zij gewerkt heeft kunnen registreren
24. Medewerker moet zijn activiteit van het project waar aan hij / zij gewerkt heeft kunnen registreren
25. Indien de medewerker zijn urenregistratie de vorige dag niet heeft uitgevoerd dan krijgt hij / zij hiervan een melding

Autoriseren

26. Projectleider moet kunnen inloggen
27. Medewerker moet kunnen inloggen

Administreren

28. Projectleider met administrator rechten moet gebruikers kunnen bekijken
29. Projectleider met administrator rechten moet nieuwe projectleiders kunnen toevoegen en bestaande projectleiders kunnen muteren
30. Projectleider met administrator rechten moet nieuwe medewerkers kunnen toevoegen en bestaande medewerkers kunnen muteren
31. Projectleider met administrator rechten moet nieuwe klanten kunnen toevoegen en bestaande klanten kunnen muteren

Klanten login

32. Klant (projectleider van klant) moet kunnen inloggen
33. Klant moet een overzichtspagina weergegeven krijgen
34. Klant moet op deze pagina de aan hem / haar bedeelde projecten kunnen zien
35. Klant kan van een project de tussen- en eindproducten bekijken
36. Klant kan van een project de start- en einddatum bekijken
37. Indien er meerdere projecten lopen bij dezelfde klant en er ook verschillende projectleiders (van de klant) zijn toegewezen dan krijgen deze projectleiders alleen inzicht in hun eigen project(en).

Factureren

38. Factuur genereren als pdf op basis van eigen template Vurig
39. Projectleider moet kunnen factureren

40. Projectleider moet een factuur kunnen maken op basis van een percentage van het budget van een project
41. Projectleider moet een factuur kunnen maken op basis van gewerkte uren in een periode van een project
42. Projectleider moet een factuur kunnen maken met een zelf op te geven bedrag
43. Projectleider moet aan kunnen geven of een factuur betaald is
44. Projectleider moet alle facturen kunnen zien in een bepaalde periode
45. Projectleider moet deze 2 bovenstaande parameters kunnen combineren
46. Projectleider met administrator rechten moet alle openstaande facturen kunnen inzien

3.3.2 Performance

47. De web-applicatie moet een response tijd hebben van minder dan 1 seconde
48. Het genereren van pdf's moet een response tijd hebben van minder dan 2 seconde

3.3.3 Onderhoudbaarheid

49. Unobtrusive Javascript en CSS (Javascript en CSS in losse bestanden niet direct in de HTML zetten)
50. HTML en CSS opstellen op aan de hand van de W3C standaard
51. Programmeren aan de hand van de Vurig standaard

3.3.4 Graphical User Interface (GUI)

Overzichtspagina

52. Projectleider moet een overzichtspagina weergegeven krijgen
53. Projectleider moet op deze pagina zijn / haar projecten kunnen zien
54. Projectleider moet op de pagina de ingeplande activiteiten voor de komende dag kunnen zien
55. Projectleider moet op deze pagina zijn / haar to-do lijst kunnen zien
56. Het moet mogelijk zijn voor projectleiders om een to-do lijst op te stellen met als extra dat projectleiders ook to-do's aan medewerkers kunnen toewijzen
57. Medewerker moet een overzichtspagina weergegeven krijgen
58. Medewerker moet op deze pagina de aan hem / haar bedeelde projecten kunnen zien
59. Medewerker moet op deze pagina de ingeplande activiteiten voor de komende dag kunnen zien
60. Medewerker moet op deze pagina ook zijn / haar to-do lijst kunnen zien
61. Het moet mogelijk zijn voor medewerkers om to-do lijst op te stellen

4 Elaboratie fase

In dit hoofdstuk worden alle stappen van de elaboratie fase van het project beschreven. Hier wordt uitgelegd hoe ik het PMS heb ontworpen en hoe ik het interactie ontwerp heb opgesteld.

De elaboratie fase tot en met punt 4.5 dient als bewijs voor de competentie OISF-7, punt 4.6 dient als bewijs voor de competentie OISF-11. Voor beiden geldt dit in combinatie met bijlage B.

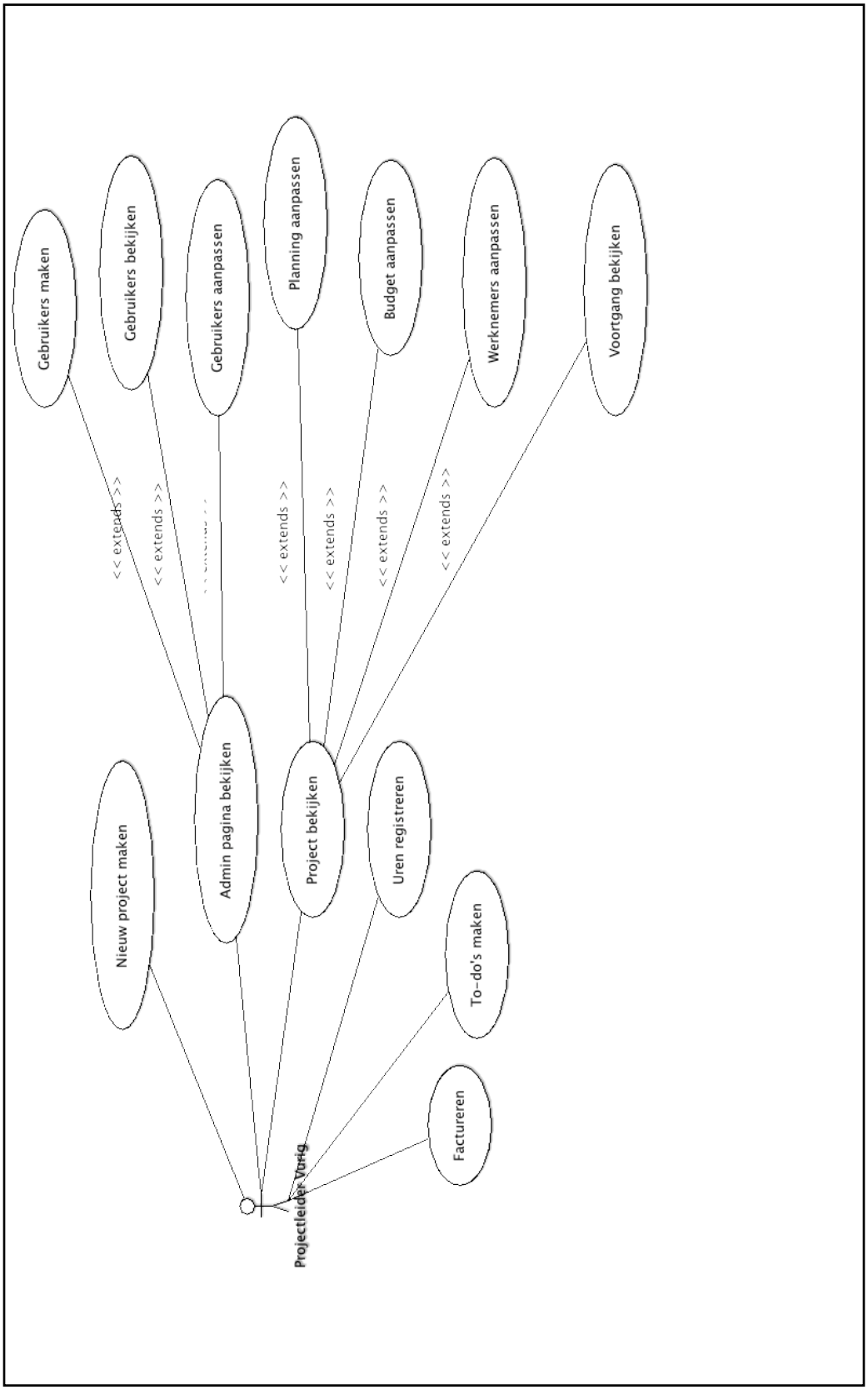
4.1 Use-cases

In dit hoofdstuk wordt beschreven hoe ik de use-cases heb opgesteld. Om de use-cases op te stellen heb ik gebruik gemaakt van de techniek UML. Eerst heb ik de use-cases in ArgoUML opgesteld en daarna heb ik de scenario's uitgeschreven.

4.1.1 Opstellen van use-cases

Het opstellen heb ik gedaan op basis van de requirements. Deze zijn terug te vinden in hoofdstuk 3.3. Doordat ik de requirements heb aangepast, had ik een veel beter beeld van wat het systeem in moest gaan houden, wat ten goede kwam aan het opstellen van de use-cases.

Allereerst heb ik de use-case voor het maken van de planning opgesteld. Ik heb hiervoor gekozen omdat zoals het in de requirements staat, dit het belangrijkste onderdeel van het systeem is.



Figuur 4.1 *Het use-case diagram voor projectleider*

Als actor voor deze use-case heb ik de projectleider genomen. Op basis van mijn actor was het mogelijk om in de requirements af te lezen wat zijn mogelijkheden moesten zijn. Namelijk:

- Vanuit de overzichtspagina moet het mogelijk zijn om alle andere modules aan te spreken
- Projectleider kiest er bijvoorbeeld voor om een project aan te maken
- Zodra hij dit doet moet hij ook een planning maken dit valt om deze reden onder dezelfde use-case

Dit is slechts een kleine greep uit het use-case diagram.

De reden hiervoor is dat een planning maken het belangrijkste is voor Vurig. Dus indien er een project gemaakt wordt is het noodzakelijk om hiervoor ook een planning te maken.

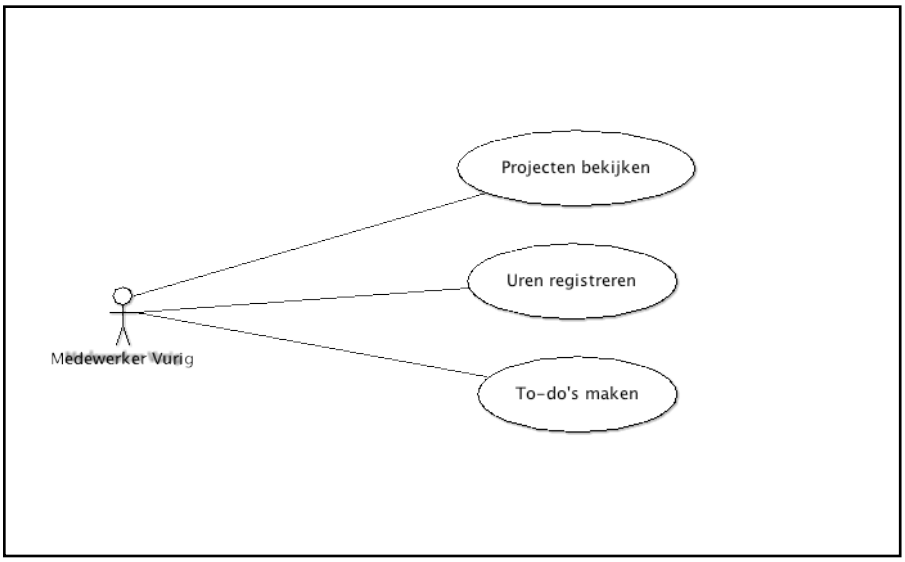
Er waren nog meer eisen namelijk:

- Projectleider moet projecten kunnen inzien
- Uren registreren
- To-do's maken

In de gesprekken met Vurig kwam naar voren dat ze ook een persoonlijke to-do lijst wilden. Deze staan staan los van de geplande activiteiten.

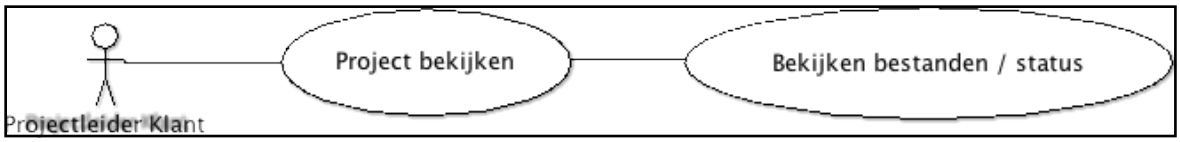
De laatste eis was factureren. Zoals eerder beschreven bij de functionaliteiten van het systeem is dit de minst belangrijkste voor Vurig.

De tweede use-case diagram die ik heb gemaakt is het PMS vanuit de medewerker gezien. Ik heb ervoor gekozen om hiervoor ook een use-case te maken om zo duidelijk aan te maken wat een medewerker moet kunnen in het PMS. Een medewerker moet ook kunnen inloggen en overzichtspagina krijgen en ook to-do's kunnen maken. Ook kan een medewerker zijn projecten bekijken.



Figuur 4.2 *Het use-case diagram voor de werknemer*

De derde en laatste use-case diagram die ik heb gemaakt is het PMS vanuit de projectleider van een klant gezien. Ondanks dat het een relatief simpele use-case diagram is vond ik het belangrijk om aan te tonen wat de klant moet kunnen met het systeem. Voor de klant is het alleen mogelijk om zijn eigen projecten te bekijken en de bestanden of de status hiervan in te zien.



Figuur 4.3 *Het use-case diagram voor de klant*

4.1.2 Beschrijven van de scenario's

Het beschrijven van de scenario's heb ik ook met behulp van de requirements gedaan en de use-cases. Voor elke use-case ben ik een scenario gaan opstellen. Allereerst heb ik het resultaat opgeschreven. Ik had nu een start- en eindsituatie. Hierdoor kon ik makkelijk de tussenliggende stappen beschrijven. In figuur 4.3 is een voorbeeld te zien van use-case scenario. De uitzonderingen die ik ben tegen gekomen in dit project zijn logische uitzonderingen. Zoals bijvoorbeeld:

- Gebruiksnaam bestaat al
- Login gegevens kloppen niet
- Budget van een project wordt overschreden

Naam	Projectleider / medewerker / klant maken Requirements nummers: 34, 35, 36
Samenvatting	De projectleider wil een projectleider / medewerker of klant maken.
Actor	Projectleider (met administrator rechten)
Aannamen	De projectleider is ingelogd en heeft de overzichtspagina voor zich.
Beschrijving	1. De projectleider kiest vanuit de overzichtspagina ervoor naar de admin pagina te gaan. 2. De projectleider kiest voor nieuwe gebruiker 3. De projectleider vult gegevens van de gebruiker in en geeft hier ook het type gebruiker aan.
Uitzonderingen	• *Gebruikersnaam bestaat al.* Het systeem geeft aan dat de projectleider een andere gebruikersnaam moet kiezen.
Resultaat	De projectleider heeft een projectleider / medewerker / klant gemaakt.

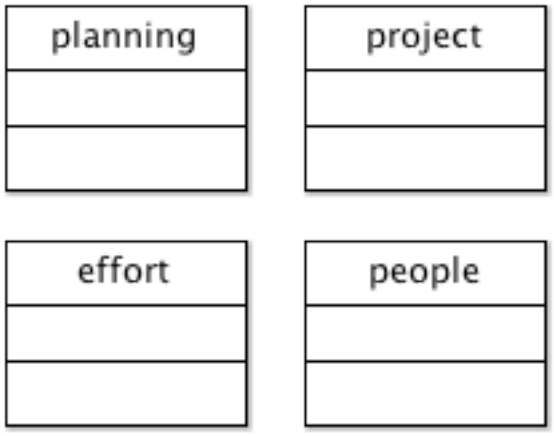
Figuur 4.3 *Voorbeeld van een use-case scenario*

4.2 Klassediagram

Na de use-cases ben ik het klassediagram gaan opstellen. Het klassediagram zorgt voor een schematische weergave van de klasses van het systeem en hun onderlinge relaties.

4.2.1 Opstellen van het klassediagram

Allereerst ben ik gaan bekijken hoe de basis die binnen Vurig beschikbaar was, was opgebouwd. In deze basis stonden de klasses project, planning, effort en people. Hierin waarin nog geen relaties tussen de klasses gedefinieerd. Vanuit deze basis ben ik gestart met de rest van het systeem. In figuur 4.4 is te zien hoe het conceptueel model er uit zag.



Figuur 4.4 *Conceptueel model*

Ik ben begonnen met het bepalen van de klasses die ik nodig zou hebben voor het systeem. Dit deed ik op basis van de requirements. Uit de requirements kon ik afleiden welke klasses ik nodig had. Ook de attributen kon voor het grootste gedeelte afleiden uit de requirements. Door middel van brainstormen kwamen er ook nog attributen naar boven. Het boek ‘Praktisch UML’ van Jos Warmer en Anneke Kleppe beschrijft namelijk twee technieken. De eerste is het onderstrepen van alle zelfstandige naamwoorden in bijvoorbeeld de probleemomschrijving. Deze klasses worden dan beschouwd als kandidaatklasses. Maar deze lijst was naar mijn inziens niet compleet genoeg vandaar dat ik de doelstelling en de requirements er ook in heb betrokken.

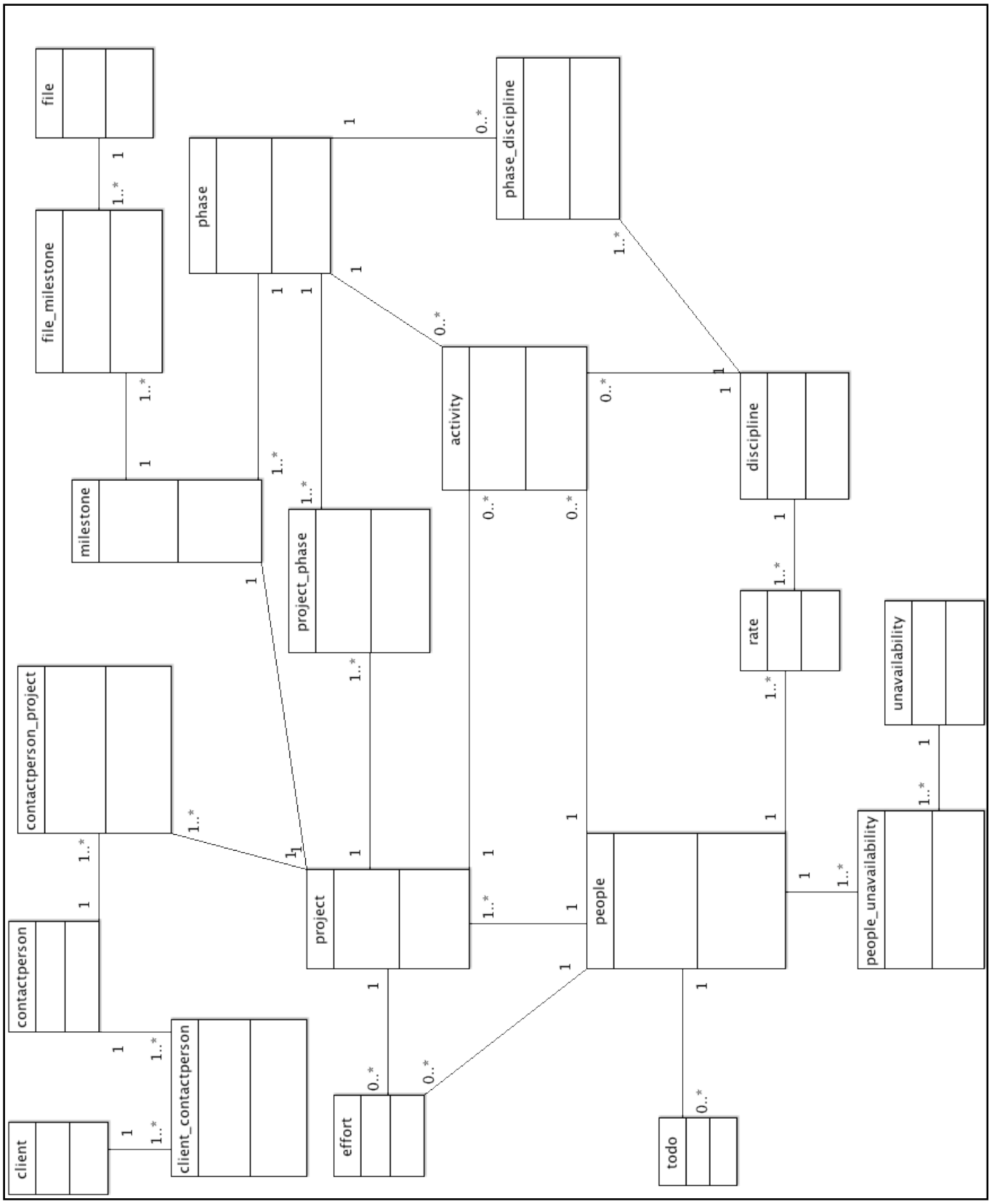
Hierna kon ik gaan bepalen welke kandidaatklasses ik daadwerkelijk nodig zou hebben. Dit deed ik door middel van beoordeling te geven aan de kandidaatklasses. Dit zijn beoordelingen als:

- Klasse
- Te vaag
- Attribuut

Dit leverde het volgende resultaat op *(gedeeltelijke weergave)*:

Kandidaatklasse	Beoordeling
To-do	Klasse
Budget	Attribuut
Bestand(File)	Klasse
Groep	Te vaag
Activiteiten	Klasse

Na het bepalen van de de klasses die in het klassediagram moesten komen, moesten de onderlinge relaties bepaald gaan worden. Logischerwijs draait het in het systeem om de planning. Deze klasse heb ik later veranderd in de klasse activiteit (activity in het model). Dit omdat een planning voortkomt uit 1 of meerdere activiteiten. Dit maakt het naar mijn idee een stuk duidelijker. De activiteit klasse speelt dan ook een centrale rol in het diagram. In deze klasse komen alle andere klasses zoals people, project, phase, discipline samen. Want op basis van deze gegevens kan ik bepalen wie, welk project, welke fase en welke discipline (een discipline is bijvoorbeeld: ontwikkelaar, flasher, ontwerper enz.) betrokken zijn een activiteit.



Figuur 4.5 *Het klassediagram (het volledige diagram met attributen is terug te vinden in het elaboratie rapport (bijlage B))*

4.3 Implementatiemodel
Na het afronden van het klassediagram ben ik het implementatiemodel gaan opstellen. Dit heb ik gedaan op basis van het klassediagram.

4.3.1 Opstellen van het implementatiemodel
Ik ben begonnen met de standaard methodes zoals de accessors en modifiers. Op basis van PHP heb ik bijvoorbeeld methodes als getClient en setClient gedefinieerd. Dit omdat je in PHP de overloading methodes __get en __set hebt, dit methodes die beschikbaar zijn voor alle attributen in de klasses. Dit heeft als voordeel dat ik in de klasse maar een __get en __set methode hoeft te maken.

De client en people klasse hebben allebei een login methode meegekregen dit omdat de gebruikers wel geautoriseerd moeten worden. Je wilt namelijk weten wie er is ingelogd en wat voor rechten hij / zij heeft. Doordat ik zowel de client als people klasse een login methode meegeef, wist ik van tevoren dat ik hiervoor een programmeer technische oplossing nodig had. Dit omdat ik moet gaan bepalen of iemand een klant of een gebruiker is. Hoe ik dit heb opgelost beschrijf ik in de constructie fase.

4.4 Relationeel representatiemodel
Nadat ik het implementatiemodel had opgesteld ben ik de gegevens voor het relationeel representatiemodel gaan vaststellen.

4.4.1 Vaststellen van het relationeel representatiemodel
Het vaststellen van de datatypes en de lengte van deze gegevens heb ik gedaan op basis van het eerdere systeem wat Vurig had staan. Het vaststellen van de primaire sleutels en de vreemde sleutels is gedaan op basis van het klassediagram.

Voor de start- en enddate is er gekozen voor een integer omdat de tijd zal worden opgeslagen als unix timestamp. Een unix timestamp is een timestamp van het aantal seconden wat verstreken is sinds 1 januari 1970 00:00:00. Het opslaan van een timestamp in plaats van een datum als string stelt mij in staat makkelijker te rekenen met tijd. Ook is het sorteren makkelijker.

people_unavailability
(people_unavailability_id, *people_id*, *unavailability_id*, start_time, end_time)

people_id is vreemde sleutel verwijst naar people_id in people, null niet toegestaan
unavailability_id is vreemde sleutel verwijst naar unavailability_id in unavailability, null niet toegestaan

people_unavailability_id - integer
people_id - integer
unavailability_id - integer
start_time - integer
end_time - integer

project
(project_id, serialnumber, year, name, description, start_date, end_date, budget, *projectleader*, deleted)

projectleader is vreemde sleutel verwijst naar people_id in people, null niet toegestaan

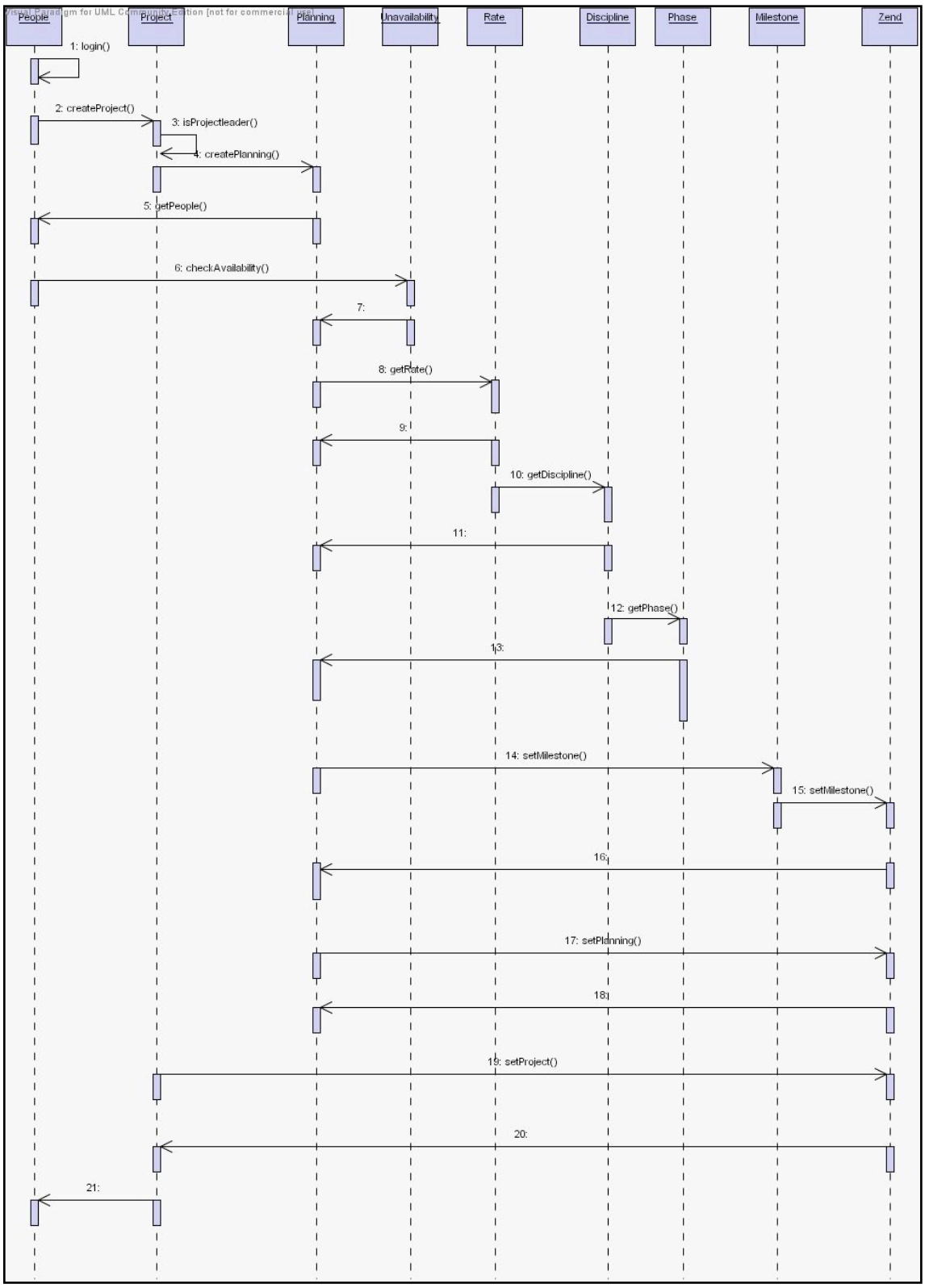
project_id - integer
serialnumber - tinyint
year - smallint
name - char (64)
description - tinytext
start_date - integer
end_date - integer
budget - integer
projectleader - integer
deleted - tinyint

Figuur 4.6 *Stuk van het relationeel representatiemodel*

4.5 Sequentiediagrammen
De volgende stap in het RUP proces is het opstellen van de sequentiediagrammen.

4.5.1 Opstellen van de sequentiediagrammen
Voor de weergave van de interactie tussen de objecten heb ik sequentiediagrammen opgesteld. In deze diagrammen komt naar voren hoe welke methodes er gebruikt worden door de verschillende klasses. Deze methodes zijn ook terug te vinden in het implementatiemodel.

Het doel bij het opstellen van de sequentiediagrammen was voor mij het inzicht krijgen welke klasses met elkaar communiceren. De laatste klasse waarmee ik communiceer is Zend maar eigenlijk communiceert Zend zelf nog door met zijn eigen klasses maar dit zou zo'n onoverzichtelijke sequentiediagram opleveren. Vandaar dat ik het op dat punt heb afgebakend.



Figuur 4.7 *Sequentie scenario maken van een planning, projectleider is actor (niet te zien op figuur 4.7)*

4.6 Interactie ontwerp

Met afronden van de sequentiediagram had ik de technische ontwerpen van het systeem af. Hierna ben ik begonnen met het interactie ontwerp voor het systeem.

4.6.1 Vaststellen strategie

Het vaststellen van de strategie, is de eerste fase in het UCD. Hier worden doelstellingen van het PMS en ook eventuele gebruikerswensen vastgesteld. De doelstellingen van het PMS waren al bekend dus die heb ik zo overgenomen uit het plan van aanpak.

Er waren in het begin weinig eisen gesteld door Vurig. Het moest mogelijk zijn om op een overzichtspagina snel in te zien welke activiteiten voor de komende dag en welke projecten zijn ingepland voor de gebruiker. De requirements die ik in hoofdstuk 3.3 beschrijf zijn meer inhoudelijk en niet zozeer gericht op grafische requirements. Door vragen te stellen over wat ze handig vonden bij de eerdere geteste PMS-en, kwam ik erachter wat ze qua interface handig achtte. Ze hadden al gebruik gemaakt van Basecamp HQ, die vonden ze qua interface fijn werken. Dat systeem heb ik als basis voor mijn ontwerp gebruikt. Verder liet ik nog wat voorbeeldjes zien van Copperproject, van Copperproject heb ik voornamelijk de detailpagina van de projecten gebruikt. Dit omdat het in veel gevallen als gebruiksvriendelijk wordt ervaren.

Het is gebruiksvriendelijk omdat bijvoorbeeld er een duidelijke indeling is gemaakt van het systeem. Zo zit aan de linkerkant de details van een onderwerp, zoals bijvoorbeeld projectdetails. Aan de rechterkant zitten bijvoorbeeld projecten om snel van project te wisselen en zit er een knop om snel een nieuw project te maken. Deze indeling wordt consistent door het systeem gebruikt. Dit maakt het toegankelijk en herkenbaar voor gebruikers.

4.6.2 Ontwerpen van geraamte

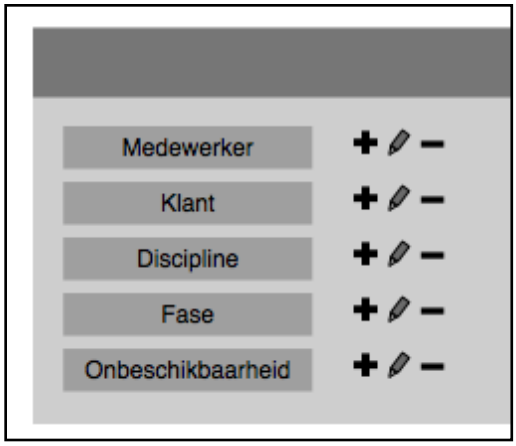
Het ontwerpen van het geraamte heb ik dus zoals ik hier boven bij punt 4.6.1 al heb beschreven op basis van Basecamp HQ en Copperproject gedaan. Dit was de essentie omdat de medewerkers van Vurig dit als gebruiksvriendelijke PMS-en ervaren. Ik ben begonnen met het bepalen van de content van de overzichtspagina in overleg met mijn begeleider en door gebruik te maken van de requirements.

Na het bepalen van de content, ben ik gaan bekijken hoe ik het op een logische manier kon gaan indelen. Dit heb ik gedaan door verschillende schetsen te maken. En deze daarna voor te leggen aan de medewerkers van Vurig. Voornamelijk zullen de drie medewerkers van Vurig met dit systeem gaan werken dus voor het grootste gedeelte heb ik het ontworpen op basis van hun input. Ik had het idee om aan de rechterkant van het scherm een rechthoekig blok te maken met daarin alle mogelijk handelingen. Zoals bijvoorbeeld het maken van een nieuw project. Het leek me handig om dat op een vaste plaats te zetten. Nadat ik mijn ontwerpen had laten zien aan de mensen van Vurig werd er besloten om van dat blok een to-do blok te maken. Dit heb ik gedaan omdat uit de feedback naar voren kwam dat ze zo altijd snel en makkelijk hun to-do’s kunnen inzien.

De structuur met de tabbladen die ik had opgezet naar aanleiding van Basecamp HQ werd goed bevonden. Doordat de structuur met tabbladen een goed overzicht geeft van waar je je bevindt binnen het systeem. Het is duidelijk zichtbaar is welke tabblad actief is. De structuur met tabbladen heb ik daarom ook behouden voor de detailpagina van een project. Er zijn 1 of meerdere tabbladen beschikbaar dit verschil is afhankelijk van je rechten als gebruiker. Ben je bijvoorbeeld een klant dan zie je alleen het dashboard, ben je een medewerker dan zie je het dashboard, projecten, activiteitenplanning en de urenregistratie. Ben je een projectleider dan zie je het dashboard, projecten, activiteitenplanning en de urenregistratie met het verschil dat een projectleider ook projecten kan maken.

De onderverdeling van dashboard, projecten, activiteitenplanning en urenregistratie. Heb ik zo opgezet omdat dit de belangrijkste onderdelen van het PMS vormen. Dit komt ook terug in de requirements. Daarin zie je dat ik de requirements heb verdeeld in verschillende onderdelen zoals overzichtspagina(dashboard), planning, uren registreren.

De projectleider kan ook to-do's naar medewerkers sturen. Tot slot heb je een projectleider met admin rechten deze heeft het dashboard, projecten, activiteitenplanning, urenregistratie en een admin pagina om gebruikers aan te maken tot zijn beschikking.

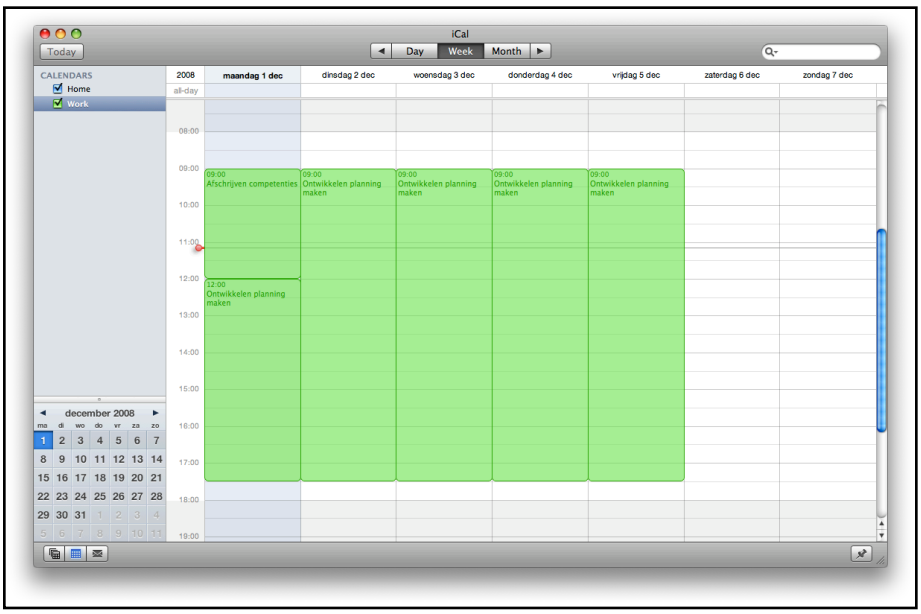


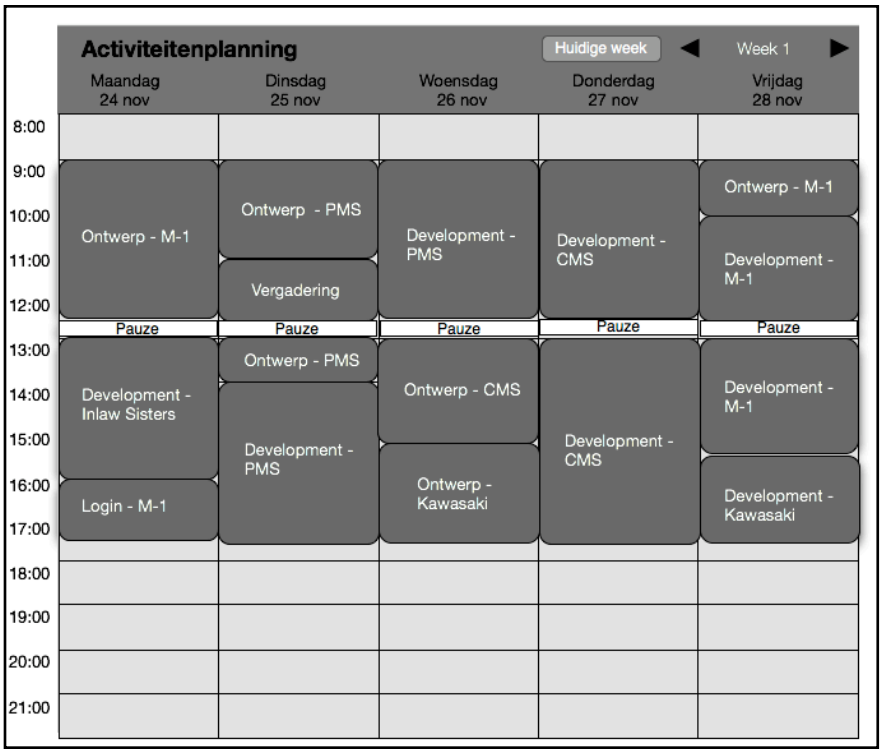
Figuur 4.8 *Knoppen*

Verder heb ik ook de knoppen om iets nieuw te maken, bewerken of verwijderen allemaal het zelfde gehouden. Deze knoppen zijn allemaal iconen geworden omdat het dit herkenbaar(der) maakt voor de gebruiker. Wanneer een gebruiker een + icoon tot zijn beschikking heeft betekend dat hij een nieuw item kan toevoegen. Indien hij het potlood icoon kan gebruiken betekend het dat hij iets kan bewerken. Indien - icoon beschikbaar is betekend het dat hij iets kan verwijderen. Bij de to-do's is gekozen om alleen het + icoon te laten zien. Als een gebruiker zegt dat hij iets heeft afgerond dan verdwijnt deze to-do na een dag uit het lijstje. De to-do blijft nog een dag staan omdat de gebruiker inzicht te geven wat hij / zij gedaan heeft op die dag. Het + icoon wordt ook altijd onder een blok gezet, dit altijd op dezelfde vaste plaats dit zorgt voor consistentie in het systeem wat als voordeel dat dat het systeem gebruiksvriendelijker maakt. Bij de to-do's kan hij niet onder het blok staan dus heb ik ervoor gekozen om hem onder het laatste to-do item te zetten. Zodat er nog steeds een consistentie is met de rest van het systeem.

Bij de admin pagina had ik eerst besloten om voor de tekstuele oplossing te gaan dus daar zou je knoppen krijgen met als tekst bijvoorbeeld: “Gebruiker toevoegen”. Uiteindelijk ben ik daar ook met de iconen gaan werken en heb deze grotendeels op dezelfde manier ingedeeld als de rest van het PMS. Alleen heb ik nu ook het + icoon het item naast gebruiker gezet. Dit qua consistentie niet de meeste correcte oplossing maar visueel komt het een stuk beter over dan als ik hem onder gebruiker zou zetten. Ook bij de projectdetail pagina heb ik gekozen voor deze oplossing omdat dit dan net iets lekkerder werkt en visueel sterker overkomt. Het komt lekkerder en visueel sterker over omdat het voor de gebruiker nog steeds het herkenbare icoon is.

Het weergeven en maken van een activiteitenplanning waren voor mij een van de lastigste punten om te ontwerpen. De activiteitenplanning is planning op basis van een week. Ik heb het ontwerp uiteindelijk op basis van iCal. Ik heb voor iCal gekozen om twee redenen. Ten eerste omdat het een overzichtelijke en duidelijke agenda applicatie is. Ten tweede zijn de medewerkers van Vurig bekend met iCal en zullen hierdoor ook makkelijk met deze planning over weg kunnen. Voor de algehele projectplanning geldt eigenlijk het zelfde.

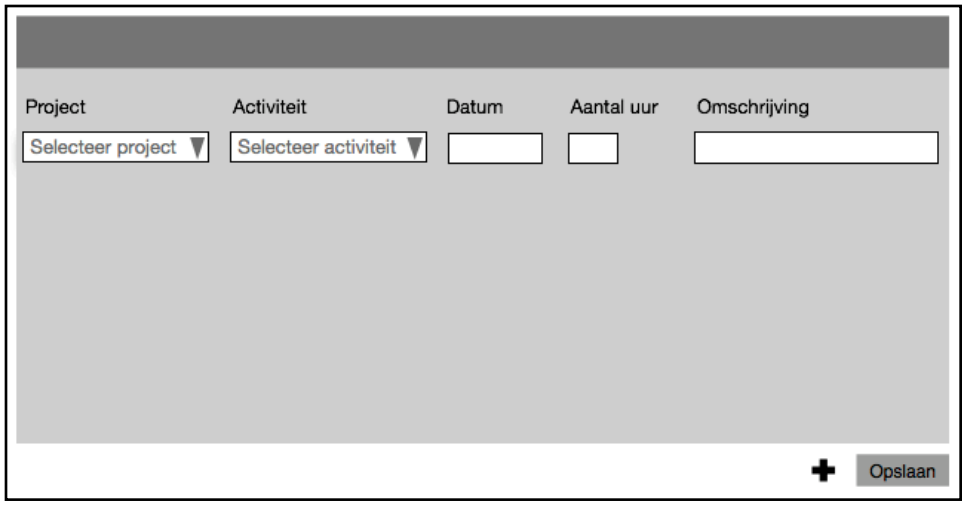




Figuur 4.9 *iCal vs. ontwerp*

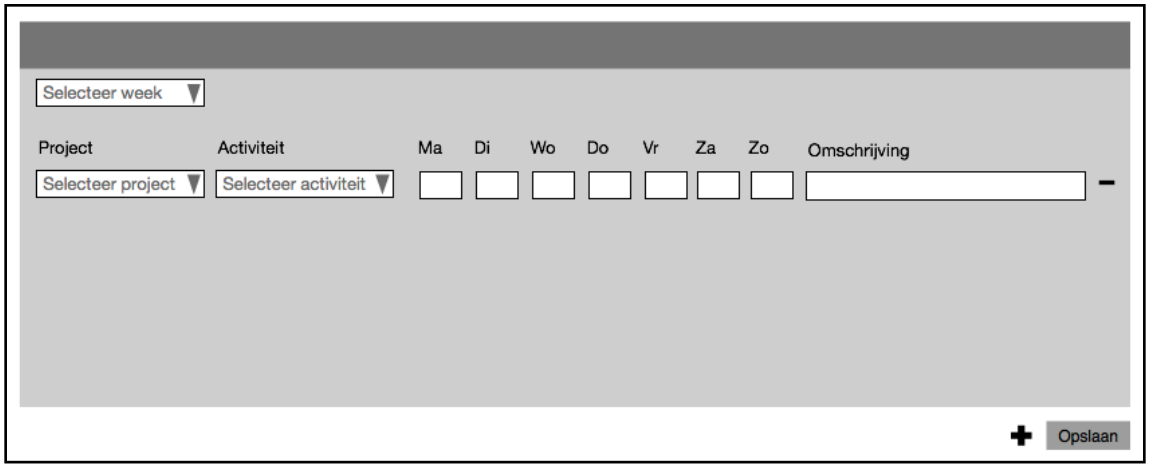
Voor de urenregistratie was er nog een discussie over of het nu per dag of per week moest gebeuren. Ik ben van mening dat het per dag beter is omdat je aan het eind van de dag nog goed weet hoeveel uren je kwijt ben geweest voor een bepaald project. Als je het aan eind van de week nog gaat invullen, ben je minder accuraat qua inschatting. Ik vind namelijk dat als je je planning zo accuraat mogelijk wil opstellen, ook een zo accuraat mogelijke urenregistratie moet hebben. Hier tegenover stond dat Vurig het makkelijker

vindt om het per week in te vullen en zou het ontwerp als in figuur 4.11 ook beter overzicht geven van hun uren. Ze waren bang dat ze het zouden vergeten om elke dag in te vullen. Maar ik bracht daar tegenover dat het systeem een melding zou geven indien ze het niet hebben ingevuld. En dat ze met mijn ontwerp nog steeds het per week in konden vullen. Zoals ook in figuur 4.10 te zien is, dit omdat het op basis van een datum gaat.



Figuur 4.10 *Mijn ontwerp voor de urenregistratie*

Het advies wat ik gegeven had namen ze niet over. Zodoende heb ik het ontwerp aangepast en ontworpen zoals deze in figuur 4.11 te zien is.

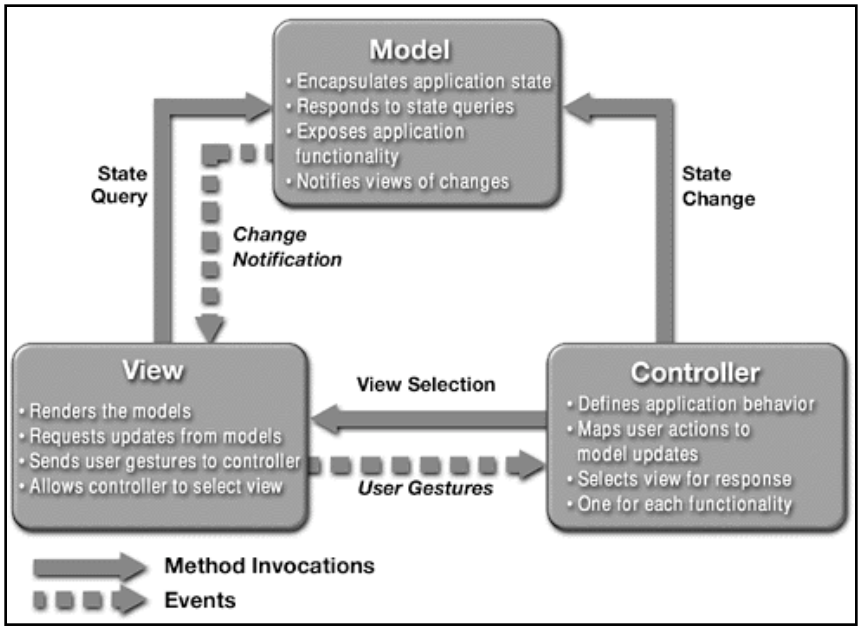


Figuur 4.11 *Het aangepaste ontwerp*

5 Constructie fase

In dit hoofdstuk wordt beschreven wat ik in de constructie fase heb uitgevoerd en hoe ik dat heb gedaan.

Deze fase dient tot en met punt 5.2 als bewijs voor de competentie OISF-8. Punt 5.3 geldt als bewijs voor de competentie IA-4 dit in combinatie met bijlage C.



Figuur 5.1 *Het MVC model*

Zoals beschreven bij techniek heb ik gebruikt gemaakt van het ZF. Het ZF is een raamwerk met daarin verschillende modules. Het framework is zo gemaakt om het bouwen van web-applicaties sneller en makkelijker te maken. Om het ZF zo eenvoudig mogelijk te houden is wordt er voornamelijk veel gebruikte modules aangeboden. Veel gebruikte modules zijn bijvoorbeeld:

Zend_Db (Database)
Zend_Rest (Webservice)
Zend_Mail (Mail)
Zend_Cache (Pagina caching)

Mochten de modules in het ZF niet voldoen, dan is het mogelijk om eigen modules te schrijven. Dit kan op basis van een bestaande module van het ZF of je kunt hem van de grond af aan opnieuw schrijven.

De structuur van het ZF is gebaseerd op het Model-View-Controller model zoals deze in figuur 5.1 te zien is. Dit een model dat de applicatie opdeelt in drie delen, waarbij elke deel andere taak heeft. Door gebruik te maken van dit model wordt het systeem flexibel en makkelijker uitbreidbaar.

Model
Het model bevat de applicatie data. Het model zal de queries bevatten die op de database worden uitgevoerd.

View
De view is de presentatie laag. De view geeft de data van het model weer. De informatie wordt verkregen door requests te doen aan het model.

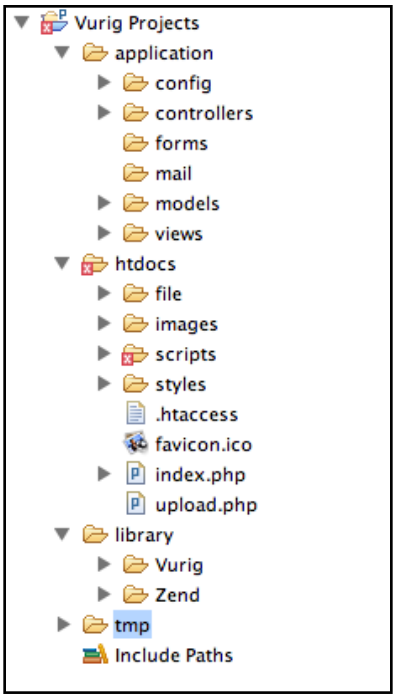
Controller
De controller is het hart van het MVC model deze zorgt voor de communicatie tussen het model en de view.

Het MVC model werkt als volgt:

1. Via de browser vraagt een gebruiker een applicatie iets te doen
2. De Controller beantwoord deze vraag door via het Model (dat met de database communiceert) data op te halen
3. De Controller roept de View aan om de opgehaalde data in een layout te plaatsen
4. De View (layout) wordt weergegeven in de browser

Op basis van het MVC model ben ik het systeem verder gaan uitwerken.

Dit model stelde mij in staat snel makkelijk de back-end van het PMS te bouwen. Allereerst heb ik de project structuur opgezet zoals het wordt aanbevolen door Zend. Dit betekend dat ik een application map heb waarin de controllers, models en views terug te vinden zijn. Verder is er een htdocs map waarin in alle direct te benaderen bestanden komen. En is er nog library map waarin het ZF staat en de extensies die door Vurig zelf zijn gemaakt. Het is ook mogelijk om naar je eigen inzicht de project structuur op te zetten maar ik vond het verstandig om het te doen op de manier die Zend aanraadt. Ik ben van mening dat dit een duidelijk structuur is waar goed mee te werken valt. Zo is er een duidelijk onderscheid tussen de publiekelijk toegankelijk map (htdocs) en de mappen wegens veiligheidsreden niet toegankelijk zijn voor het publiek. Doordat bij het ZF ook mogelijk is om eigen modules of varianten op bestaande modules te bouwen. Is er ook de library van Vurig en Zend apart gezet. Dit levert als voordeel namelijk, indien er een update is voor het ZF kan ik deze snel bijwerken en hoef ik niet bang te zijn dat eventueel eigen varianten op modules verloren gaan.



Figuur 5.2 *Project structuur*

** Apache is een webserver. Een webserver is een computer die is gekoppeld aan het internet. Op deze computer bevinden zich bestanden, afbeeldingen of webpagina's di e door derden via het internet geraadpleegd kunnen worden.*

Zend's front controller is een controller die alle requests voor het PMS afhandelt. Om hiervan gebruik te maken voer ik een rewrite uit. Hierbij wordt gebruik gemaakt van de Apache* module mod_rewrite. Deze rewrite rule zorgt ervoor dat alle requests die gedaan worden via de index.php worden gestuurd.

In de index.php wordt de zogenaamde bootstrap opgesteld. In dit bestand configureer je het ZF. Hier roep je de Zend Autoloader aan. Deze zorgt ervoor dat alle klasses uit de library worden in geladen, zonder dat je include of require hoeft te gebruiken. Ook zet je hier de paden naar applicatie, controllers en de library. Ook dispatch je hier de front controller. Deze voert de request uit. De front controller is een klasse die het Singleton pattern implementeert, dit houdt in dat er maar een instantie is van de Zend_Front_Controller bij elke request. Doordat een web-applicatie maar een request per keer kan doen is er ook maar een instantie nodig.

In de bootstrap roep je een config.xml aan. Hierin staat de database, hostname, de encryption key voor de authenticatie. Je kunt hier ook bijvoorbeeld meerder configuraties inzetten zoals bijvoorbeeld een production en een staging. Zodra je live gaat hoef je alleen in de bootstrap om te zetten dat je gebruik maakt van bijvoorbeeld production.

Na het maken van de bootstrap ben ik begonnen met het schrijven van controllers. De eerste controller die je maakt is normaal gesproken de index controller. De controllers is een extensie op de klasse Zend_Controller_Action. In de controller definieer je ook een methode indexAction(). Dit is de standaard actie die wordt gedaan zodra je de controller aanroept.

```php
<?php
class IndexController extends Zend_Controller_Action {

    private $_session = null;
    private $_frontend = null;

    public function preDispatch() {
        $this->view->headScript()->appendFile('/scripts/Vurig/home.js');
        $this->_session = Zend_Registry::get('projectsSession');
        $this->_frontend = Vurig_Projects_Frontend::getInstance();

        if (!$this->_frontend->authenticated()) {
            $this->_redirect('login');
        }
    }

    public function indexAction() {
        $this->view->projects = $this->_frontend->getProject(((bool)$this->_session->admin) ? 0 : $this->_session->userId);
    }

}
```

Figuur 5.3 *Voorbeeld van een indexController*

Na het maken van de controller is het tijd voor de view. Dit is het PHTML bestand die zorgt voor de weergave van de data. Vanuit de controller is het mogelijk om variabelen toe te wijzen aan de view. Door deze variabelen toe te wijzen is het mogelijk om de data door de view uit te laten lezen en weer te geven. In figuur 5.3 zie je dat ik in de indexAction() een variabele project toe wijs aan de view. Het is ook mogelijk dat een variabele een array is met als data bijvoorbeeld alle projecten. Dit kan je in view bijvoorbeeld uit laten lezen met een foreach loop.

Voor het project heb ik een webservice controller gemaakt die drie functies bevat de eerste is preDispatch(). Deze functie zorgt ervoor dat hetgeen wat in de preDispatch() functie staat eerst wordt uitgevoerd voor bijvoorbeeld de indexAction(). In het geval van de webservice controller geef ik aan dat ik geen renderer wil voor de view en een setNoController() deze zorgt ervoor dat render klasse niet gaat zoeken voor een script. Ik heb geen view nodig omdat de Rest klasse XML opbouwt en die heeft hierdoor geen view renderer nodig.

Verder zit er in webservice een functie restAction() in deze functie maak ik een nieuwe instantie van de klasse Vurig_Rest_Server. En geef ik de webservice klasse mee aan de webserver.

Als laatste bevat de webservice controller ook nog jsonAction() functie. In deze functie maak ik een nieuwe instantie van de Zend_Json_Server. Ook hier geef ik weer de webservice klasse mee. Indien er een directe call wordt gedaan op de jsonAction dan wordt er een service mapping description (SMD) gedownload. Hierin staan alle methodes die in de webservice beschikbaar zijn. Bij het bouwen van de webservice moet er bij de functies gedocumenteerd worden van welk datatype de parameters zijn en wat de return waarde is.

```
/**
 * Gets people details
 *
 * @param integer $peopleId
 * @return array
 */
public function getPeopleDetails($peopleId) {
    if ($this->_session->authenticated == true && $this->_session->usertype == 'people') {
        $db = $this->_getDatabase();

        $select = $db->select()
                    ->from('people')
                    ->where('people.people_id = ?', $peopleId)
                    ->query();

        return array(
                    'status' => self::SUCCESS,
                    'class' => 'peopledetails',
                    'result' => $select->fetchAll()
                    );
    } else {
        return array(
                    'status' => self::FAILED,
                    'message' => self::DENIED
                    );
    }
}
```

Figuur 5.4 *Functie in de webservice met documentatie*

5.1 Het bouwen van de webservice

Toen ik begon met het bouwen van de webservice, stond ik voor keuze hoe ik mijn data zou ophalen. Dit kan namelijk op verschillende manieren.

Ik had de keuze voor deze twee opties:

- Zend_Db_Select
- Zend_Db_Table

Ik ben eerst gaan kijken of deze twee opties ook zouden werken in combinatie met een webservice. Dit heb ik gedaan door een kleine test te schrijven. In deze test haalde ik wat gegevens op. Zend_Db_Table bleek niet goed te werken met een webservice waardoor ik dus met Zend_Db_Select ben gaan werken.

Bij beide opties zou het makkelijk geweest zijn om de database relationeel vast te leggen in PHP. Het verschil in de opties is eigenlijk dat het ophalen en bewerken van data bij Zend_Db_Table gemakkelijker is. Bij Zend_Db_Select schrijf je namelijk nog steeds queries al is het nog steeds in een simpelere vorm dan normale SQL queries.

Ander belangrijk punt om gebruikt te maken van Zend_Db_Select (of Zend_Db_Table indien dit dus had gewerkt) is dat deze module(s) veilig zijn. Door met behulp van Zend_Db_Select queries op te stellen is het systeem beveiligd tegen eventuele SQL injecties.

Het maken van de planning heb ik in een webservice gedaan. Ik heb een webservice gebouwd omdat ik hierdoor onafhankelijk ben van andere platformen. Hierdoor is het mogelijk om met PHP en AJAX de webservice te benaderen. In plaats van dat er voor PHP en AJAX een aparte code moet geschreven worden. Een webservice is universeel dus kan ik het met verschillende platformen benaderen. Ook zal de webservice als model gaan dienen in het MVC model.

Hierna ben ik de verschillende methodes gaan schrijven. Voor de meeste klasses gold dat ik de standaard add, get, update en remove methods schreef. Maar klasses zoals phase, milestone enzovoorts kon ik dat gelijk een join query doen bij het maken van andere klasse. Dit omdat het bijvoorbeeld gerelateerd is aan een project.

De opzet van de methodes is vrij simpel. Allereerst ben ik begonnen met de gemakkelijkste klasses en zodoende ben ik verder gaan werken naar de complexere. Ik heb hiervoor gekozen om zo rustig te wennen aan het ZF. Door deze manier van werken waren de complexere dingen beter uit te voeren omdat ik het ZF tegen die tijd beter begreep.

In de library heb ik een Projects klasse aangemaakt die als webservice geldt. Als eerste in de klasse heb ik de constanten gedefinieerd. Dit heb ik gedaan zodat ze gemakkelijk en snel aangepast kunnen worden. Deze constanten zijn bijvoorbeeld foutmeldingen die het systeem moet weergeven in bepaalde situaties. Ook heb ik een success en failed constanten gedefinieerd. Dit zijn statussen die ik telkens mee stuur in de resultset die ik terug krijg van de webservice. Na de constanten heb ik nog wat private variabelen gedefinieerd. Hierop volgde de constructor. Na de constructor heb ik de private functies

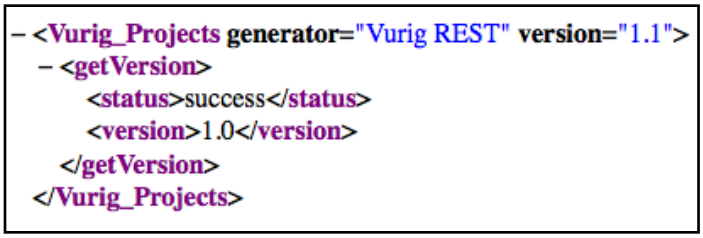
geplaatst en alle public functies hieronder. Dit gaf mij een duidelijk structuur waarmee ik te werk kon gaan. Zo waren de private of public functies makkelijk en snel terug te vinden.

Om te testen of de webservice klasse goed werk heb ik eerst een functie getVersion() gemaakt welke een array teruggeeft. Met het versie nummer welke in een constante gedefinieerd is en succes als result. Op deze manier wist ik dat het webservice een goede response teruggaf.

Ik roep de webservice op de volgende manier aan:

http://projects/webservice/rest?method=getVersion

projects is mijn base url. Als ik naar http://projects surf dan pakt Zend automatisch de index actie van de index controller.
/webservice door dit achter projects te zetten roep ik de webservicecontroller aan.
/rest door dit achter /webservice roep ik de restAction() aan.
?method=getVersion door dit achter /rest zetten roep ik de functie in de webservice klasse aan indien ik nog parameters in de functie zou hebben dan geef ik die meer door bijvoorbeeld &name=Vurig mee te geven achter getVersion.



Figuur 5.5 *XML Result*

Nadat ik deze functie gemaakt en getest ben ik de de private functie _getDatabase() gaan maken. Dit is een vrij eenvoudige functie waarin ik de volgende return had staan:

```
return Zend_Registry::get('database');
```

Door dit te doen haalde hij de configuratie van de database op uit de config.xml. Welke ik eerder al gezet heb in de bootstrap (index.php).

Nadat de functie voor de database was geschreven kon ik beginnen met het schrijven van de functies voor het systeem. Ik ben begonnen met de klasse client omdat dit een vrij eenvoudige klasse is. Hierdoor kon ik rustig wennen aan het ZF. Om te beginnen ben ik data gaan ophalen. Voor het ophalen heb ik twee functies gemaakt de eerste is getClientList en de tweede is getClientDetails. In het geval ik meer gegevens wil van een klant. De getClientsDetails heeft daarom nog een parameter met een clientId. Om te testen of het werkte heb ik zelf even met de hand wat test data in de database gezet.

```
$db = $this->_getDatabase();

$select = $db->select()
            ->from('client', array('client_id','name', 'short_name'))
            ->where('deleted = 0')
            ->query();
return array(
            'status' => self::SUCCESS,
            'class' => 'client',
            'result' => $select->fetchAll()
            );
```

Figuur 5.6 *De getClientList functie*

Zoals in figuur 5.6 te zien is het relatief eenvoudig om SQL op te halen.
Nadat dit werkte ben ik de functie voor het toevoegen van een client gaan maken. Deze functie vereiste wat meer werk omdat ik nu ook filters en validators ga gebruiken. Filters gebruik je om te kijken om er bijvoorbeeld HTML tags uit te filteren of alleen getallen of alleen letters. Hierna krijg je de validators hier geef je aan of het integers of strings zijn of iets anders. Ook geef je aan of de parameter benodigd is of dat de parameter ook leeg mag zijn. Nadat je dit gedaan hebt roep je de Zend_Filter_Input aan om te kijken of de resultaten valide zijn. Zo ja, dan roep ik weer _getDatabase aan. Hier op volgt een $db->beginTransaction() waarmee ik aangeef dat ik database transactie wil gaan uitvoeren. Door aan te geven dat ik een transactie wil starten is het mogelijk om met een try catch statement
data in de database te zetten indien dit niet lukt dan doet hij rollback van database en geeft een FAILED message als return value. Mocht de client naam en afkorting van de client al voor komen dan geeft hij hier een foutmelding van weer.

```
if ($input->isValid()) {
    if (!$this->_checkExistingClient($input->name) && !$this->_checkExistingShortName($input->shortName) ) {
            $client = array(
                        'name' => $input->name,
                        'short_name' => $input->shortName,
                        'visiting_address' => $input->visitingAddress,
                        'visiting_postal_code' => $input->visitingPostalCode,
                        'visiting_city' => $input->visitingCity,
                        'visiting_country' => $input->visitingCountry,
                        'mail_address' => $input->mailAddress,
                        'mail_postal_code' => $input->mailPostalCode,
                        'mail_city' => $input->mailCity,
                        'mail_country' => $input->mailCountry,
                        'administrative_address' => $input->administrativeAddress,
                        'administrative_postal_code' => $input->administrativePostalCode,
                        'administrative_city' => $input->administrativeCity,
                        'administrative_country' => $input->administrativeCountry
                        );
            $db = $this->_getDatabase();
            $db->beginTransaction();
            try {
                $db->insert('client', $client);
                $db->commit();
                return array(
                        'status' => self::SUCCESS
                        );
            } catch (Exception $e) {
                $db->rollBack();
                return array(
                        'status' => self::FAILED
                        );
            }
    } else {
        return array(
                'status' => self::FAILED,
                'message' => self::EXISTINGCLIENTSHORT
                );
    }
} else {
    return array(
                'status' => self::FAILED,
                'message' => $input->getMessages()
                );
}
```

Figuur 5.7 *Een gedeelte van addClient functie.*

Voor het updaten en deleten geldt dit grotendeels hetzelfde met als verschil dat je nu een clientId als parameter meegeeft en deze code uitvoert in plaats van een insert:

```
$db->update('client', $client, sprintf('client_id = %d', $input->clientId));
```

In het geval van verwijderen van een client zet ik deleted op 1. Dit heb ik pas tijdens de constructie fase ontworpen in het klassediagram. Door het attribuut op deleted op 1 te zetten geef ik aan dat bijvoorbeeld deze client verwijderd is. De standaard waarde zoals deze in database is gedefinieerd is 0.

Het kan nodig zijn om te weten welke contactpersoon er bijvoorbeeld verbonden was aan een project. Om deze reden wordt de data dus niet verwijderd uit de database. Deze attribuut gebruikte ik bij verscheidene tabellen. Bij de tussentabellen gebruik ik dit attribuut niet. Dat komt omdat ik geen data verwijder uit deze tussentabellen. Ik verwijder geen data uit de tussentabellen omdat de data die hierin staat relevant kan zijn voor verwijderde data, die dus niet “echt” uit systeem zijn verwijderd. Als ik de data uit de tussentabellen zou gooien dan weet ik niet meer waarmee de data een relatie had.

In het geval van tabel file heb ik geen attribuut deleted. Want ik wil geen bestand onnodig op het systeem laten staan. Dit zou teveel opslagruimte gaan kosten. Alhoewel er natuurlijk ook wat complexere functies waren. Deze functies kwam ik vooral tegen bij de planning en het bestanden gedeelte. Omdat er bij de planning veel gebruik wordt gemaakt van de tussentabellen, maakte het wat complexer dan normaal. Neem bijvoorbeeld phase zodra ik hier een nieuwe van maak wil ik deze ook gelijk kunnen koppelen aan een disicpline. Dit heb ik nu opgelost door een array mee te sturen waarmee ik gelijk meerdere disciplines aan een phase kan koppelen in de tussentabel phase_discipline.

Ten tijde van het bouwen van de back-end liep ik tegen wat complicaties aan. Of anders gezegd tegen beperkingen in mijn eerste ontwerp van het klassediagram. Bij het ophalen van de planning kwam ik tot de conclusie dat het phase_id niet uniek genoeg is om te bepalen bij welk project het hoort. Aangezien een phase_id bij meerdere projecten voor kan komen. Als oplossing hiervoor heb ik het project_id aan de activity klasse toegevoegd als vreemde sleutel. Nu weet ik welke activiteiten bij bijvoorbeeld project_id twee horen.

Nadat het gehele systeem in de webservice zat ben ik de authenticatie gaan bouwen. Er is gekozen voor een oplossing met sessies. Belangrijke reden om hiervoor te kiezen was dat ik makkelijk variabelen kan opslaan in een sessie. Zodoende kan ik dus makkelijk opslaan welke rechten een gebruiker allemaal heeft. De sessie wordt gestart in index.php. Er wordt gekeken of er nog geen sessie cookie bestaat anders wordt deze aangemaakt met de tijdsduur van een dag. Deze validatie heb ik er ingebouwd omdat het systeem anders meerdere cookies maakt en dit leverde bij de authenticatie problemen op.

Om klant en medewerker uit elkaar te houden, is er aan de functie login een parameter type meegegeven. Op basis hiervan weet ik of ik de data uit de client of people tabel moet lezen. In de sessie cookie wordt de volgende data opgeslagen:

- usertype
- admin
- userId
- authenticated

5.2 Het bouwen van de front-end

In het begin fase van het project heb ik meegeholpen met wat projecten die liepen binnen Vurig. Ik hielp mee met deze projecten omdat deze projecten ook gebruik maakte van de technieken die voor mijn project van belang waren. Zoals bijvoorbeeld:

- Dojo
- Zend Framework

Dit stelde mij instaat om me in te werken in de technieken. Dit heeft als voordeel gehad dat ik bij het bouwen van de front-end tegen minder problemen aanliep. Ik kon nu ook veel makkelijker dingen oplossen omdat ik nu veel meer inzicht had in de Dojo library. Hierdoor kwam ik tot de conclusie dat mijn huidige planning waar ik maar een week voor de front-end had ingepland niet zou voldoen. Op basis van de projecten waar ik had meegeholpen kwam ik er ook achter dat het bouwen van de back-end minder tijd zou gaan kosten dan ingepland. Dit leverde als voordeel op dat ik de tijd die ik oorspronkelijk voor de back-end had ingepland nu als tijd voor de front-end kon gebruiken.

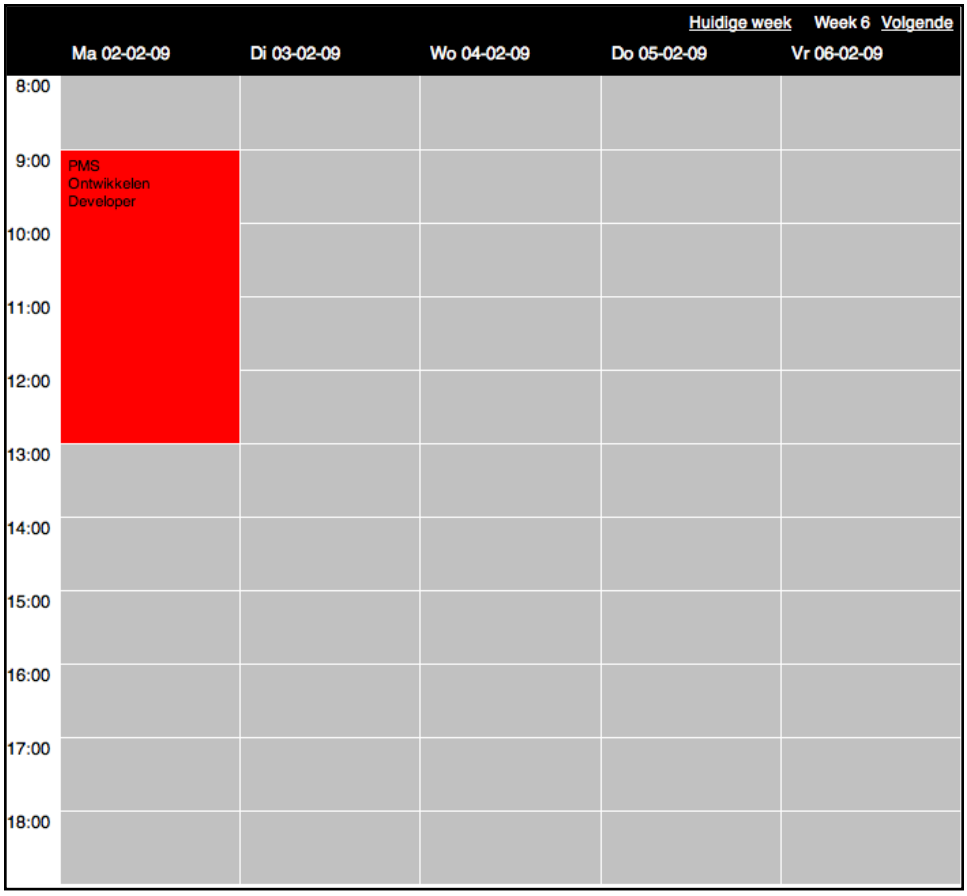
Als eerst ben ik de grafische opzet zoals ik die in de wireframes heb gemaakt, gebouwd met HTML en CSS. Met dit als basis is het makkelijker de andere delen van de front-end te bouwen. Omdat ik nu een grafische schil heb waar ik tegen aan kan werken. Dat maakt geheel een stuk overzichtelijker.

Nadat ik de basis van het front-end had staan ben ik begonnen met de module voor het maken van de planning. Dit omdat het zoals in de requirements beschreven staat de meeste prioriteit heeft.

Allereerst ben ik de activiteitenplanning voor een gebruiker op gaan halen. Het ophalen zelf was niet moeilijk. Maar het plaatsen van de activiteit daarentegen was complexer. In het klassediagram had ik bij de tabel activity de attributen time, start- en enddate gedefinieerd. Het feit dat er een enddate in stond maakte het weergeven van de activiteit complex. De startdatum is de startdatum van de activiteit bijvoorbeeld 10-01-2009 en de einddatum is 17-01-2009 en de totale beschikbare tijd is 7 uur. Hoe moet ik dat dan gaan weergeven in de planning? Om dit probleem op te lossen heb ik de attribuut enddate uit tabel activity gehaald. Nu weet ik op welke dag ik de activiteit moet weergeven en hoelang deze activiteit moet zijn.

Voor de activiteitenplanning voor een gebruiker moest ik weten wat de huidige week was. Dit doe ik door de huidige datum op te halen. Hierna moet ik bepalen wat de eerste dag van de week was waarbij wordt uitgegaan van de ISO 8601 standaard. Deze standaard zegt namelijk dat de eerste dag van de week op maandag valt. Om dit op te halen heb ik een functie geschreven welke ik in de Vurig library heb opgenomen namelijk Vurig_Measure_Time.

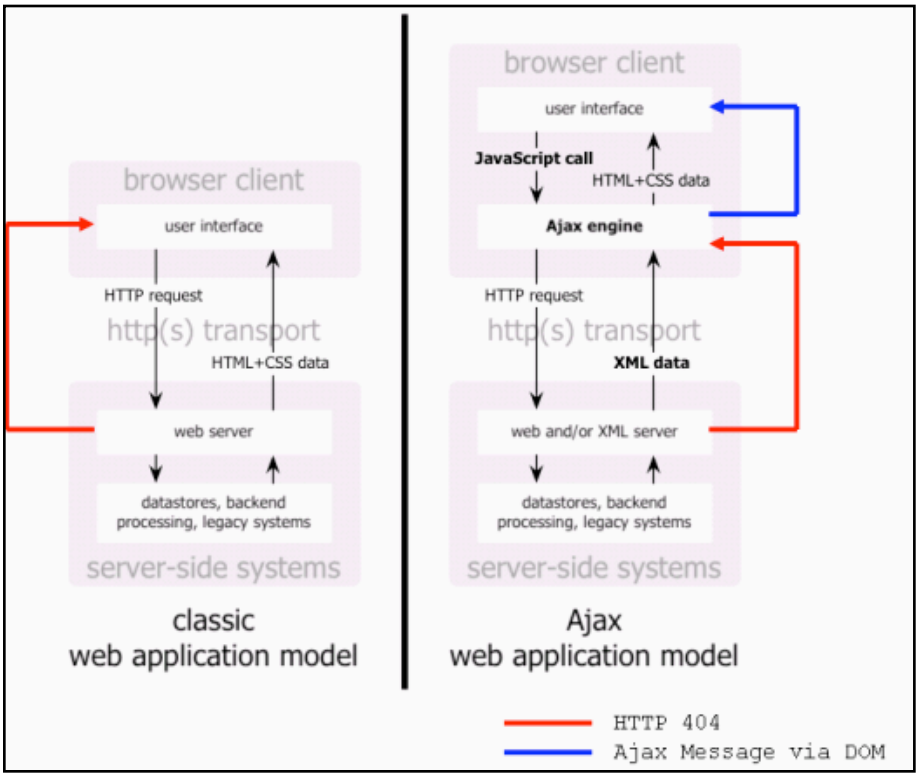
Ik heb hiervoor een speciale klasse geschreven omdat ik vaak van deze klasse gebruik zou maken. Op deze manier stond het op een centrale plek en was het makkelijk om aan te roepen. In deze klasse staan twee functies, getDaysInWeek om de dagen in de week op te halen en getFirstDayOfWeek. In deze laatste functie haal de eerste dag van de week op basis van een timestamp dit kan de huidige tijd zijn of zoals ik later zal uitleggen bij de projectplanning een timestamp van het project. Nu ik de eerste dag van de week weet is het makkelijk om de rest van de dagen ook te bereken omdat ik gebruik maak van een timestamp hoef ik alleen maar de timestamp van de eerste dag + 86400 te doen. 86400 is het aantal secondes om er een dag bij op te tellen. Zo kon ik ook makkelijk de datums voor de volgende week berekenen.



Figuur 5.8 *De activiteitenplanning*

In figuur 5.8 is te zien hoe de planning er uitziet. Om de plaatsing van activiteit te bereken is gebruik gemaakt van vaste hoogte en breedte per dag. Omdat ik dat weet een uur nu een vaste hoogte aan pixels heeft is het makkelijk bereken hoe lang een activiteit is. Door het aantal pixels te vermenigvuldigen met de time attribuut die uit database wordt gehaald. De activiteiten worden per dag opgehaald. Om de pagina zonder refresh van de pagina de data op te halen heb ik gebruik gemaakt van AJAX. De library hiervoor was Dojo. Omdat Dojo nauw samenwerkt met het Zend Framework is er gekozen voor deze library. Met behulp van Dojo kan ik callbacks maken op de server. Callbacks zijn functies die worden aangeroepen op het moment dat er een event plaats vindt zoals bijvoorbeeld

een onclick event. De functies die worden aangeroepen zijn functies afkomstig van de webservice die ik eerder heb gebouwd. De callbacks worden uitgevoerd op de achtergrond zonder dat de pagina gerefreshed moet worden.



Figuur 5.9 *Standaard web model vs AJAX web model*

De response die ik terug krijg van de webservice is een JavaScript Object Notation(JSON) response. JSON is een lichtgewicht formaat dat wordt gebruikt voor het uitwisselen van datastructuren. Groot voordeel van JSON is dat het volledig taal onafhankelijk is. JSON maakt het ook makkelijker om de response uit te lezen. In deze response welke ik dus als object terug krijg kan ik bijvoorbeeld data uit de database inlezen.

Zo staat in de response voor de activiteitenplanning een timestamp en de tijd die ik heb voor een activiteit. Hierdoor kan ik dus berekenen hoe hoog en waar precies de activiteit geplaatst moet worden. Ditzelfde verhaal geldt grotendeels ook voor de projectplanning.

Bij de projectplanning stond ik voor een visueel probleem. Namelijk wat gebeurd er als er twee activiteiten op hetzelfde tijdstip vallen en door twee verschillende personen wordt uitgevoerd. Hoe moet ik dit dan gaan weergeven? Hiervoor heb ik als oplossing bedacht dat activiteiten van het project per medewerker van het project wordt weergegeven. Rechts van de planning verschijnt een lijst met de medewerkers en de geselecteerde medewerker wordt dik gedrukt weergegeven.

Hierna ben ik gedeelte voor het inplannen gaan maken. Als je iemand gaat inplannen wordt er een callback gedaan, die checkt in people_unavailability en in activity of het geselecteerd persoon beschikbaar is. Dit heb ik opgelost met een SQL query.

Deze query zit als volgt in elkaar:

```
SELECT COUNT(people_id) AS `planning`,
        (SELECT COUNT(people_id) FROM people_unavailability
         WHERE people_unavailability.people_id = 15
         AND (1233835200 BETWEEN start_time AND end_time)
         OR people_unavailability.people_id = 15
         AND (1233849600 BETWEEN start_time AND end_time)) AS `unavailable`
FROM `activity`
WHERE (activity.people_id = '15')
AND ((1233835200 BETWEEN activity.start_date
AND activity.start_date + (3600 * activity.time)))
OR (activity.people_id = '15')
AND (1233849600 BETWEEN activity.start_date
AND activity.start_date + (3600 * activity.time))
```

Ik tel hoe vaak het people_id voorkomt, waar de toe te voegen activiteit z'n timestamp in dit geval *1233835200* ligt tussen de starttijd van de andere activiteiten en de eindtijd van deze activiteiten van people_id 15. Dit zelfde doe ik met de eindtijd van de toe te voegen activiteiten. Hetzelfde doe ik ook met de select waar ik kijk of de medewerker niet afwezig is. Indien ik resultaat vind dus de count is groter dan een. Dan weet ik dat diegene niet kan worden ingepland op dit tijdstip.

Na het bouwen van de planning ben ik het gedeelte voor de overzichtspagina gaan bouwen. Hier ben ik begonnen met de to-do lijst. Mijn grootste uitdaging lag in het toewijzen van een to-do aan andere medewerkers. Dit moet zoals in de requirements staat alleen mogelijk zijn voor projectleiders. Om te bepalen of iemand een projectleider is heb ik een callback gemaakt welke een true of false teruggeeft. Deze callback roept de functie isProjectleaderAdmin aan in de webservice. In de functie kijk ik in de sessie of de variabele is_projectleader of is_admin op 1 staat. Indien dit zo is geef ik een true waarde terug. Hierdoor weet ik dus dat degene die is ingelogd een projectleider of admin is.

Het was een wens van Vurig om de to-do's nadat ze zijn afgevinkt een dag te laten staan. Maar om dit technisch mogelijk te maken moest ik het klassediagram en het relationeel representatiemodel opnieuw aanpassen. In deze iteratie nam ik een attribuut completed_time op. Dit is een timestamp van de huidige tijd. Op deze manier kan ik simpel in SQL berekenen welke to-do's ik allemaal nog weer moet geven.

Verder zit er op de overzichtspagina een overzicht van de activiteiten van de komende dag. Ook is het mogelijk om door de dagen heen te bladeren. Op basis van de timestamp van de huidige dag en de gebruiker haal ik de data op. Het ophalen van de data wordt ook weer met een callback gedaan zodat de pagina niet onnodig gerefresht hoeft te worden.

Ook worden de projecten waar de medewerker is gekoppeld getoond op het dashboard. Indien de ingelogde gebruiker een projectleider met administrator rechten is dan ziet hij alle projecten die lopen. Deze informatie lees ik uit de sessie. Bij de projecten laat ik ook nog zien wat de status is van het project. De status wordt berekend op basis van twee gegevens. De eerste is het totaal aantal uur wat ingepland staat voor een project. Het tweede is het aantal uur wat geregistreerd is in de urenregistratie.

Dan krijg je de volgende formule:

geregistreerde uren / (totaal project / 100) = percentage status

15 / (20 / 100) = 75%

Indien het percentage groter is dan 100% zal er een css klasse aan worden gekoppeld die het percentage in het rood zet om duidelijk te maken dat het is overschreden.

Figuur 5.10 *De overzichtspagina*

Nadat de overzichtspagina af was, ben ik het projecten gedeelte gaan bouwen. Dit gedeelte is voornamelijk opgebouwd met de standaard *.phtml pagina's die gebruikt worden in het ZF voor de view. Zodra er op tabblad projecten geklikt wordt komt er een lijst met alle projecten ook de projecten die al zijn afgerond. Het verschil met de overzichtspagina is dat de afgeronde projecten dus ook worden getoond. In de view wordt dit getoond met de tekst afgerond bij de desbetreffende projecten.

Om de lijst met projecten niet oneindig door te laten lopen is er een navigatie ingebouwd. De lijst met project kan ongeveer 40 projecten op een pagina laten zien. De paginering heb ik gebouwd met behulp van de Zend module, Zend_Paginator.

Deze module zorgt ervoor dat een result zoals bijvoorbeeld een result set van een query, makkelijk gepagineerd kan worden. In de controller geeft ik aan dat ik een nieuwe instantie van Zend_Paginator aan wil maken met daar een Zend_Paginator_Array in deze array sla ik de result set van de query op. Daarna geef ik aan door middel van de functie setItemCountPerPage(40) hoeveel items ik per pagina wil laten weergeven. In een aparte .phtml laat ik de navigatie opbouwen. Voor de navigatie kan ik gebruik maken van de volgende variabelen:

Property	Type	Description
first	integer	First page number (i.e., 1)
firstItemNumber	integer	Absolute number of the first item on this page
firstPageInRange	integer	First page in the range returned by the scrolling style
current	integer	Current page number
currentItemCount	integer	Number of items on this page
itemCountPerPage	integer	Maximum number of items available to each page
last	integer	Last page number
lastItemNumber	integer	Absolute number of the last item on this page
lastPageInRange	integer	Last page in the range returned by the scrolling style
next	integer	Next page number
pageCount	integer	Number of pages
pagesInRange	array	Array of pages returned by the scrolling style
previous	integer	Previous page number
totalItemCount	integer	Total number of items

Door deze variabelen te gebruiken is het gemakkelijk de navigatie op te bouwen.

Hierna ben ik de detailpagina gaan maken voor een project. Deze pagina's waren niet meer dan wat standaard pagina waar wat data werd opgehaald. Een van die pagina's was het gedeelte voor de bestanden die bij een project horen. Het ophalen hiervan was niet zo'n probleem. Maar het het uploaden van een nieuwe bestand was daarentegen een stuk lastiger.

Het was de bedoeling dat dit een AJAX uploader zou gaan worden. Dit zorgde voor wat complicaties. Normaal gesproken wordt er een HTML formulier opgebouwd en met de commando dojo.formToObject(). Maak ik van het HTML formulier een object waar ik de data dus snel en gemakkelijk uit kan lezen. Maar in het geval van type filedata werkt dojo.formToObject() niet meer. Om hier een workaround voor te vinden heb ik eerst een ongecomprimeerde versie van Dojo gebruikt, in de ongecompromideerde versie wilde ik het zo aanpassen dat ik filedata wel kan gebruiken in de formToObject() functie. Maar dit werkte niet.

Uiteindelijk heb ik het opgelost met een iFrame. Een iFrame is een frame dat een ander gedeelte van een site kan bevatten. In dit geval bevatte het de response voor de uploader. Indien de webservice als response success terug geeft dan weet ik dat het uploaden is gelukt.

Figuur 5.11 *De AJAX uploader*

5.3 Opstellen testplan en uitvoering tests

Het testen van het systeem is op verschillende manieren uitgevoerd:

- Tijdens het bouwen van de webservice
- Tijdens het bouwen van de front-end
- Het uitvoeren van PHPUnit tests met behulp van de Zend_Test module
- Blackbox test met invoer en verwachte uitvoer

Het testen tijdens het bouwen van de webservice

Tijdens het bouwen van de webservice heb ik het systeem gelijk getest. Dit omdat ik tijdens het bouwen gelijk kijk of de functies die ik heb gebouwd ook werken. Ook test ik dan gelijk of de eerdere genoemde validators ook hun werken doen. Op deze manier ben ik er van verzekerd dat de functies hun werk goed doen en dat ik achteraf geen gekke problemen meer krijg.

Om de validators testen heb ik verschillende testdata gebruikt. Zoals bijvoorbeeld html tags, spaties, karakters als punt, komma, haakjes.

Het testen tijdens het bouwen van de front-end

Tijdens het bouwen van front-end testte ik gelijk of de werking van het systeem het deed zoals ik verwachtte. Indien dit niet het geval was paste ik het gelijk aan in de back-end van het systeem. Ook testte ik of er netjes foutmeldingen worden weergegeven.

Ik heb het getest door middel van acties uit te voeren, zoals bijvoorbeeld:

- **Actie**: Klikken op een project **Resultaat:** Detailpagina van een project weergeven
- **Actie:** Bestandsnaam wijzigen **Resultaat:** Bestandsnaam gewijzigd zonder pagina opnieuw te laden.

PHPUnit tests

De twee bovengenoemde tests zijn niet grondig genoeg om het systeem goed te testen maar het werkte wel om in grote lijnen een goed systeem neer te zetten. Het was lastig om te bepalen welke technieken ik zou gaan gebruiken voor het grondig testen van het systeem. Ik heb hier een lange tijd over nagedacht en uiteindelijk gekozen voor PHPUnit tests. Ik heb hiervoor gekozen omdat ten eerste een grondige manier is om het systeem te testen. Het heeft voordelen als:

- Unit tests zorgt ervoor dat er minder bugs in de code voorkomen
- Unit tests zijn geautomatiseerde tests die zo vaak als nodig kunnen worden uitgevoerd
- Door unit tests te gebruiken is het makkelijker om te testen of wijzigingen in de code problemen opleveren voor bestaande modules

Ten tweede is het een plus dat PHPUnit in het ZF zit ingebouwd.

Met de Zend_Test module is het mogelijk om controllers en hun acties te testen. Het enige nadeel van het gebruiken van de Zend_Test module is dat het bouwen van test controllers veel tijd kost.

Allereerst moet ervoor het testen een aparte bootstrap worden gemaakt. In deze bootstrap wordt de frontController niet gedispatched zoals dat normaal gesproken wel het geval is.

De volgende stap is het maken van een testController. Hierna maak je in de testController een methode setUp(). In deze methode maak je een nieuwe instantie van je bootstrap aan.

Nadat ik dit had gedaan ben ik de methodes voor het testen gaan schrijven. Een methode ziet er bijvoorbeeld zo uit:

```
public function testInvalidLoginCredentials()
{
    $request = $this->getRequest();
    $request->setMethod('POST')
            ->setPost(array(
                'username' => 'invalidusername',
                'password' => 'invalidpassword',
            ));
    $this->dispatch('/login');
    $this->assertNotRedirect();
    $this->assertQuery('form');
}
```

Figuur 5.12 *Voorbeeld van een test methode*

Hierna riep ik de test aan met behulp van command line tool. Ik zette het volgende in de command line:

`phpunit loginTestController`

loginTestcontroller is de class die je moet aanroepen.

En dit was het resultaat.

```
PHPUnit 3.3.0 by Sebastian Bergmann.
..
Time: 0 seconds
OK (2 tests, 2 assertions)
```

Voor elke test geeft PHPUnit iets terug:

.
Een punt geeft aan dat een test succesvol is.

F
Een F geeft aan dat de gefaald is.

E
Een E geeft aan dat er tijdens het testen een error is opgetreden.

S
Een S geeft aan dat een test is overgeslagen.

I
Een I geeft aan dat een test incompleet is.

Blackbox test
De laatste test die ik heb uitgevoerd is een blackbox test. In deze test geef ik de invoer, dit is een actie die moet worden uitgevoerd door de tester. Verder heb ik ook een verwachte uitvoer, hierin staat wat de verwachting is van de uitvoer op basis van de invoer. Uiteindelijk bevat het overzicht ook nog een actuele uitvoer, hierin staat wat er daadwerkelijk is uitgevoerd.

Als testpersoon heb ik iemand gekozen die affiniteit heeft met computers. Dit omdat de degenen die er gebruik van gaan maken sowieso affiniteit met computer hebben. Het testpersoon heb ik verschillende activiteiten laten uitvoeren die gedefinieerd staan in het constructie rapport. Dit zijn activiteiten als:

- Inloggen met juiste gegevens
- Inloggen met onjuiste gegevens

Het testpersoon heeft zijn bevindingen genoteerd en hiermee ben ik aan de slag gaan. Indien de actuele uitvoer niet overeen kwam met de verwachte uitvoer.

6 Evaluatie
In dit hoofdstuk wordt zowel het proces als product geëvalueerd.

6.1 Procesevaluatie
Voor het grootste gedeelte ben ik tevreden over het hele proces. Ik ben geen dingen tegen gekomen waar ik niet uit kwam.

Het grootste kritiek punt wat ik eigenlijk heb op het gevolgde proces is dat ik in het begin het proces niet formeel benaderde. Dit kwam voor een gedeelte door de informele sfeer in het bedrijf en voor gedeelte door mezelf omdat ik er iets te gemakkelijk over dacht. Ik onderschatte de omvang en diepgang van het systeem volledig. Nadat ik er al snel achterkwam dat het niet goed ging, ben ik het formeler gaan benaderen. Dit deed ik door planmatig de methodes te gaan volgen. In de toekomst zal ik daarom projecten formeler gaan benaderen.

Tijdens het proces heb ik ook de planning moeten aanpassen omdat ik zag dat de huidige planning van dat moment niet zou voldoen. Het bouwen van de back-end ging namelijk sneller dan verwacht. Zodoende had ik dus al snel een volledige back-end gebouwd. Ik heb Vurig ook meegeholpen met wat projecten die liepen. Dit om al technische kennis te vergaren over de technologieën die ik zou gebruiken voor het project. Hierdoor zag ik dat ik te weinig tijd had voor zien voor het bouwen van de front-end. Doordat ik het nu vroegtijdig wist en het voordeel had dat het bouwen van de back-end sneller ging dan verwacht, kon ik makkelijk de planning aanpassen. Had ik niet meegeholpen aan die projecten dan had ik het niet geweten en had ik een minder volledig product op kunnen leveren.

Achteraf gezien ben ik ook tevreden over de gekozen methoden. Met RUP was ik al bekend dus daar kwam ik niet voor grote verrassingen te staan. UCD daarentegen was helemaal nieuw voor me. De methode was in het boek van Jesse James Garrett helder en duidelijk omschreven. Ik ben ook content met de keuze voor een competentie die eigenlijk buiten mijn vakgebied ligt. Ik heb nu veel meer inzichten gekregen in het ontwerp proces. Hierdoor ben ik staat om het project in een nog grote geheel te zien. Dit kan van pas komen in toekomstige projecten waarin ik samen moet werken met een ontwerper.

Het enige wat ik jammer vind bij het proces is dat ik geen tijd had om het interactie ontwerp goed te testen. Dit kwam voornamelijk wegens de beperkt beschikbare tijd voor het project. Indien ik meer tijd had gehad dan had ik dit graag getest. Ook de laatste blackbox test van het RUP proces had ik eigenlijk wat uitgebreider willen doen. Het bouwen van de PHPUnit tests kostte teveel tijd waardoor de laatste blackbox test er een beetje bij inschoot.

6.2 Productevaluatie

Ik ben zeer tevreden met hetgeen wat ik uiteindelijk heb kunnen opleveren. Ook al zit in het systeem niet alles wat ik van tevoren gepland had. Ik ben van mening dat het systeem zoals ik het nu heb opgeleverd, voldoet om te gebruiken. Het enige wat nog uitgewerkt moet worden is het grafische ontwerp. Maar ook zonder dit ontwerp valt er al goed te werken met het systeem.

Over de opgeleverde producten ben ik tevreden met uitzondering van het constructie rapport. Ik vond zelf het testplan te weinig informatie bevatten omdat het bouwen van de PHPUnit tests met achteraf gezien teveel tijd hebben gekost. Misschien was het testen door middel van PHPUnit tests misschien wat overbodig en had ik me beter kunnen richten op een grondigere blackbox test. Ik ben van mening dat blackbox tests genoeg informatie opleveren om te zien of een systeem aan de verwachtingen voldoet. Bij een volgend project zal ik daarom eerder gebruik maken van deze blackbox tests.

De diepgang in het product zat het voor mij vooral in het MVC model en het ZF. Op deze wijze een systeem benaderen was nieuw voor mij. De heldere structuur in het MVC model maakt geheel overzichtelijk en onderhoudbaar. In de toekomst zal ik zeker meer projecten met behulp van het ZF gaan bouwen. Het MVC model in combinatie met het ZF biedt zoveel voordelen en maakt het bouwen van complex(ere) systeem toegankelijker.

Afkortingen en begrippen

AJAX	*Asynchronous Javascript And XML.* Is een techniek om interactieve web-applicaties te ontwikkelen.
CSS	*Cascading Style Sheets.* Is een techniek om de vormgeving van een of meerder webpagina's in een bestand vast te leggen.
Dojo	Is een AJAX framework met verschillende modules aan boord.
HTML	*HyperText Markup Language.* Is de opmaaktaal waarmee je documenten (webpagina's) maakt, die bekeken kunnen worden op het internet.
MySQL	MySQL is open source relationeel database management systeem.
PHP	*Hypertext PreProcessor.* Is een scripttaal waarmee het mogelijk is om dynamische websites te maken.
PMS	*Project Management Systeem.* Is een systeem waarin het mogelijk is om projecten en planningen te beheren.
REST	*Representational state transfer*. Is een beschrijving van hoe je data kan benaderen. Over het algemeen gebeurd dit met webservices.
RUP	*Rational Unified Process.* Is een ontwikkel methode dat zicht richt op iteratief ontwikkelen van software.
UCD	*User Centered Design.* UCD is een ontwerp methode die gebruikt wordt om grafische ontwerpen te ontwikkelen waarin de gebruiker centraal staat.
Webservice	Een webservice is een component die toegankelijk is via de standaard webprotocollen. Deze maakt het mogelijk om op afstand een dienst aan te vragen.
Zend Framework	Is een framework gemaakt met de gedachte om het ontwikkelen van sites zo simpel mogelijk te houden.

Bijlagen & bronnen

Bijlagen

A. Inceptie rapport
B. Elaboratie rapport
C. Constructie rapport
D. Beoordelingen

Bronnen

Ontwerp methoden

http://serg.telecom.lth.se/research/publications/docs/282_runeson_XP_RUP.pdf

The Elements of User Experience: User-centered design for the web, Jesse James Garrett
ISBN-10: 0735712026
ISBN-13: 978-0735712027

http://www.jjg.net/elements/

Praktisch UML, Jos Warmer en Anneke Kleppe
ISBN-10: 9043008125
ISBN-13: 9789043008129

Ontwikkelen

http://framework.zend.com
http://www.json.org
http://www.dojotoolkit.org

Testen

http://phpimpact.wordpress.com/2008/12/27/phpunit-testing-zend-framework-controllers/