Afstudeerverslag

Versie 1.0

Dit document beschrijft het proces wat gedurende de afstudeerstage is doorlopen en beschrijft de daarbij genomen keuzes en motivaties.

Robin Scholtes
18-3-2011

Referaat

Afstudeeropdracht – “VernieuwBouw Online Soccer Manager API”
Scholtes, R (06000827)
GameBasics, Zoetermeer november 2010 – maart 2011

Dit document is geschreven naar aanleiding van de door de student uitgevoerde afstudeeropdracht bij het bedrijf GameBasics. Dit afstudeerproject maakt onderdeel uit van de opleiding Informatica aan de Haagse Hoge School.

Gedurende de afstudeerperiode van zeventien weken is er binnen ASP.Net MVC een API ontwikkeld die het voor gebruikers mogelijk maakt op verschillende manieren de functionaliteiten van Online Soccer Manager te bedienen. Daarbij zijn er verschillende mogelijkheden geïmplementeerd waarmee de eindgebruiker data kan opvragen en tonen.

Descriptoren

- ASP.net MVC
- API
- Afstuderen
- Online Soccer Manager
- Serialisatie
- Autorisatie
- GameBasics
- Robin Scholtes
- RUP
- UML
- TDD

Voorwoord

Voor u ligt het afstudeerverslag van Robin Scholtes wat naar aanleiding van het afstuderen bij GameBasics geschreven is. Het afstudeerproject is uitgevoerd als onderdeel van de opleiding Informatica aan de Haagse Hoge School. Dit document is bedoeld voor zowel de student als de stage examinatoren.

Student, De student gebruikt dit verslag om de stappen en keuzes te beschrijven die hij tijdens het project gemaakt heeft en welke competenties hij hiermee behaald heeft.

Stage examinatoren, De examinatoren zullen aan de hand van dit document het proces wat de stagiair heeft doorlopen beoordelen aan de hand van de keuzes en motivaties die hier beschreven gaan worden.

Hierbij wil ik van de gelegenheid gebruik maken de personen te bedanken die het mogelijk hebben gemaakt dit project te voltooien.

Als eerste wil ik J. Derwort en F. Tijhuis voor de begeleiding en het bieden van de mogelijkheid om het afstudeertraject te doorlopen bij GameBasics.

Als tweede wil ik B. Kuiper en G.A. Mijnarends voor de tijd die gestoken is in het beoordelen en feedback geven op de producten opgeleverd tijdens het afstudeertraject.

Als derde wil ik G. Meuldijk bedanken voor het helpen bij kleine code problemen en het geven van nieuwe ideeën voor de API.

Robin Scholtes,
GameBasics BV.
Zoetermeer, Maart 2011

Inhoudsopgave

1 Inleiding

In dit hoofdstuk wordt de aanleiding beschreven van dit document en wordt er vervolgens kort beschreven wat er in het document staat en waar dit terug te vinden is.

1.1 Aanleiding afstudeerverslag

In het kader van de opleiding Informatica aan de Haagse Hogeschool is er bij het bedrijf GameBasics te Zoetermeer gedurende de periode november tot en met maart een afstudeerproject uitgevoerd. Het project dat in dit document beschreven zal is het implementeren van een API voor Online Soccer Manager met behulp van ASP.Net MVC en een aantal andere technieken die later beschreven zullen worden.

1.2 Doelstelling afstudeerverslag

Het doel van dit verslag is het verschaffen van inzicht in de door de student uitgevoerde werkzaamheden gedurende zijn afstudeertraject. Dit zodat de examinatoren een beoordeling kunnen geven en kunnen bepalen of de student de van te voren aangegeven competenties behaald heeft en of de keuzes die hij gemaakt heeft overwogen en gegrond zijn.

1.2 Structuur afstudeerverslag

Als eerste wordt er in dit document een beschrijving gegeven van het bedrijf (§ 2) en zullen de aanleiding(§ 3.1), probleemstelling (§ 3.2), en doelstelling (§ 3.3), die voor de opdracht relevant zijn beschreven worden. Tevens wordt er in het kort beschreven hoe de huidige situatie er bij ligt (§ 3.4).

Vervolgens wordt het plan van aanpak beschreven. Hier wordt niet zozeer de inhoud van het plan van aanpak besproken maar de totstandkoming van het document en waarom er gekozen is voor de zaken die opgenomen zijn in het plan van aanpak (§ 4).

Hierna worden de fases besproken die tijdens het project doorlopen zijn. Als eerste komt de Inception fase aan bod, de fase waarin het project opgestart wordt en de requirements opgesteld worden (§ 5). Vervolgens wordt de Elaboration fase beschreven, de fase waarin de API ontworpen en uitgedacht wordt (§ 6). Daarna wordt de Construction fase beschreven, de fase waarin het daadwerkelijk implementeren beschreven zal staan (§ 7). Tenslotte wordt de transition fase beschreven, dit is de fase waarin de overdracht plaatsvindt (§ 8).

Verder zal er na het beschrijven van het proces een evaluatie volgen over het uitgevoerde proces en over het opgeleverde product. In deze evaluaties wordt er opgenomen wat er goed en fout is gegaan en welke lessen er geleerd zijn voor toekomstige soortgelijke projecten(§ 9).

Aan het einde van het document is een glossary opgenomen met uitleg van technische begrippen, een literatuur lijst van de gebruikte bronnen bij dit project en alle documenten die tijdens het project opgeleverd zijn.

2 Bedrijf

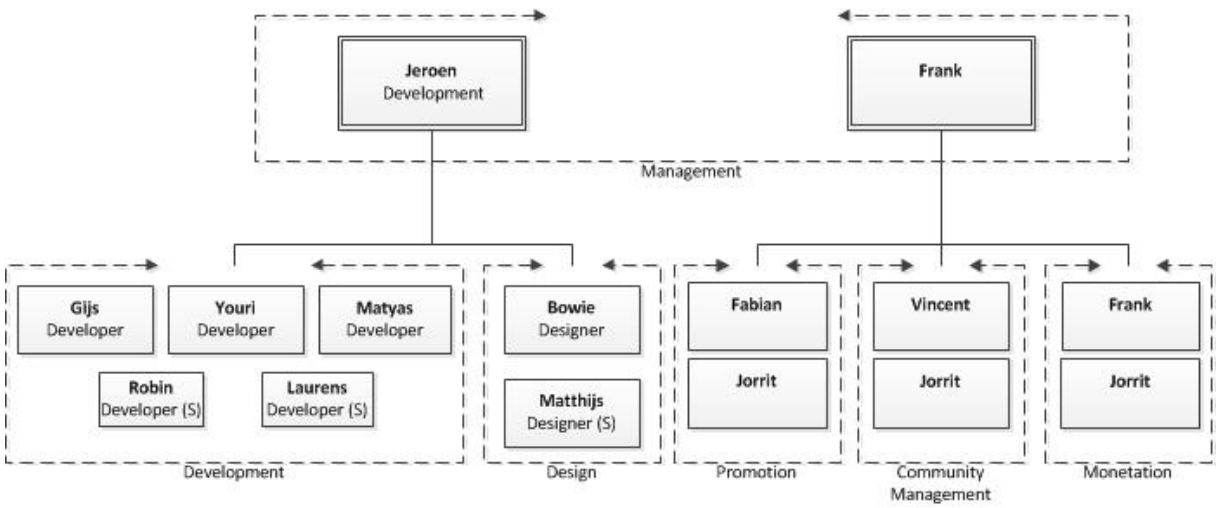
2.1 Organisatie Omschrijving

GameBasics BV. ontwikkelt en exploiteert online community games in Nederland en in het buitenland. Het doel van GameBasics is om een first-mover te zijn en een wereldwijd herkenbare speler op het gebied van online management webgames. De afgelopen 6 jaar is GameBasics aanzienlijk gegroeid en spelen meer dan 500.000 mensen hun voornaamste product, www.onlinesoccermanager.nl. GameBasics is bezig met het deployen van nieuwe versies in verschillende landen om zo in 2014 ongeveer 1,5 miljoen actieve spelers te hebben. De missie van GameBasics is als volgt:

Gamebasics wil zoveel mogelijk voetballiefhebbers wereldwijd in gelokaliseerde communities 10 minuten per dag het gevoel geven dat zij manager zijn van hún favoriete club. Dit willen we bereiken door te streven naar een optimale spelbeleving, waarbij je met je vrienden, op elk gewenst moment, via elk zinvol online platform je voetbalkennis kan etaleren.

2.1 Organisatie structuur

Op dit moment telt GameBasics 15 werknemers. Deze zijn werkzaam in administratie, development en marketing. De huidige twee directeuren zijn degene die het bedrijf hebben opgericht en de eerste versie van Online Soccer Manager geschreven hebben.



Figuur 1 - Organisatie Structuur

3 Opdracht: De OnlineSoccerManager API

In dit hoofdstuk wordt de opdracht beschreven die tijdens dit afstudeertraject is uitgevoerd door de student.

3.1 Aanleiding

In 2001 is de eerste versie van Online Soccer Manager ontwikkeld in Classic ASP en Visual Basic. Deze software was oorspronkelijk gebouwd voor een kleine set requirements en weinig gebruikers. Door de sterke groei van Gamebasics is de software steeds verder geëvolueerd en uitgebreid. De technische grenzen van wat er mogelijk is met de eenvoudige scripttechnologie zijn echter nagenoeg bereikt.

De complexe en uitgebreide structuur vraagt om een nieuwe en volwassen omgeving die er voor kan zorgen dat OnlineSoccerManager op meerdere servers in meerdere talen verschillende platformen kan bedienen.

3.2 Probleemstelling

GameBasics is een transitie project gestart om de website te herschrijven zodat deze weer aansluit bij de toekomst. Daarbij wordt gebruik maakt van innovatieve technologieën, zoals het ASP.NET MVC framework. Op dit moment zijn er verschillende versies van OnlineSoccerManager. Er is een versie die te bekijken is in mobiele web browsers en een versie die te bekijken is vanuit web browsers vanaf een normale desktop pc. Beide versies draaien op verschillende broncode, terwijl de achterliggende functionaliteiten hetzelfde zijn. De MVC technologie moet ervoor zorgen dat dezelfde codebase kan worden gebruikt voor verschillende versies van de software. Voor het communiceren met externe systemen, zoals sociale media en mobiele applicaties, is een API nodig die binnen het nieuwe framework is ontwikkeld.

3.3 Doelstelling

Zoals eerder beschreven is GameBasics een transitie project gestart om de oude versies uit te faseren en deze te vervangen met een nieuwe versie die gebruik maakt van de nieuwste technologieën. De doelstelling van dit project is om in ASP.Net MVC een “oplossing” te implementeren die kan worden ingezet om de functionaliteiten van OnlineSoccerManager te koppelen aan en beschikbaar maken voor Mobile Apps(toekomstige Android / IPhone applicaties) en Social Media(Twitter, Facebook Games). Het streven is om de functionaliteiten die op dit moment aanwezig zijn in de website zo volledig mogelijk beschikbaar te maken op andere platformen. Deze oplossing zal vanaf nu als API beschreven worden, omdat een API een Interface (tussenlaag) is waarmee het mogelijk is functionaliteiten van een systeem te benaderen.

3.4 Scope

Dit project zal zich richten op het implementeren van een API waarmee het voor applicaties mogelijk wordt om functionaliteiten en data van OnlineSoccerManager te benaderen. Daarbij moet er een oplossing verzonnen worden om er voor te zorgen dat een gebruiker kan inloggen met de API. Het toepassen van optimalisatie technieken valt buiten de scope van dit project, echter als er punten zijn waar het toegepast kan worden dienen hier wel notities van gemaakt te worden.

3.5 Uitgangssituatie

In dit hoofdstuk wordt de huidige situatie beschreven. Dit wordt gedaan door als eerste te beschrijven wat OnlineSoccerManager is, daarna wordt er verdere toelichting gegeven op hoe de architectuur van het systeem in elkaar zit en als laatste wordt het transitie project verder beschreven.

Wat is OnlineSoccerManager?

OnlineSoccerManager is een online voetbal manage spel, wat gebruikers de mogelijkheid biedt om hun favoriete team te managen. Gebruikers kunnen deelnemen aan reeds bestaande competities of kunnen samen met hun vrienden een nieuwe competitie starten om zo te kijken wie de beste manager is. Het spel is richt zich vooral op een jongere doelgroep en is zo opgezet dat het niet uitmaakt hoe vaak je online bent. Dit om het hersenloos klikken te voorkomen en het zogenaamd "botten" het niet waard te maken.

Na het registreren van een account krijgt de gebruiker de keuze een competitie te kiezen om vervolgens een team te kunnen kiezen. Nadat deze stappen doorlopen zijn kan een gebruiker zichzelf manager van een club noemen. Op dit moment krijgt een manager een aantal taken:

- Opstelling, Formatie en Tactiek bepalen.
  Een manager is vrij in het kiezen van zijn formatie en welke spelers hij opstelt, het is echter verstandig te kijken naar de vermoeidheid en het moraal van spelers voordat je deze opstelt.

- Spelers trainen
  Wanneer een speler niet opgesteld wordt kan deze getraind worden wat er voor zorgt dat deze speler zijn kwaliteiten kan verbeteren, wat terug te zien zal zijn in zijn statistieken.

- Personeel aannemen (doktoren, fysiotherapeuten, trainers etc.)
  Om er voor te zorgen dat je team fit blijft, getraind kan worden en dat geblesseerde spelers behandeld kunnen worden is het de verantwoordelijkheid van de manager het juiste personeel aan te nemen. Hoe meer personeel er aangenomen wordt, hoe meer geld dit echter kost.

- Stadion Managen
  Een manager krijgt de mogelijkheid om sponsors aan te nemen in het stadion om zo de inkomsten van de club te vergroten. Daarbij kan het stadion uitgebreid worden zodat er bijvoorbeeld meer toeschouwers in kunnen, wat extra inkomsten genereert, of er kan gekozen worden voor bijv. betere trainingsfaciliteiten.

- Transferbeleid
  Als manager ben je verantwoordelijk voor het aan en verkoop beleid van je club. Spelers die niet nodig of goed genoeg zijn kunnen verkocht worden en nieuwe spelers kunnen gescout en aangetrokken worden.

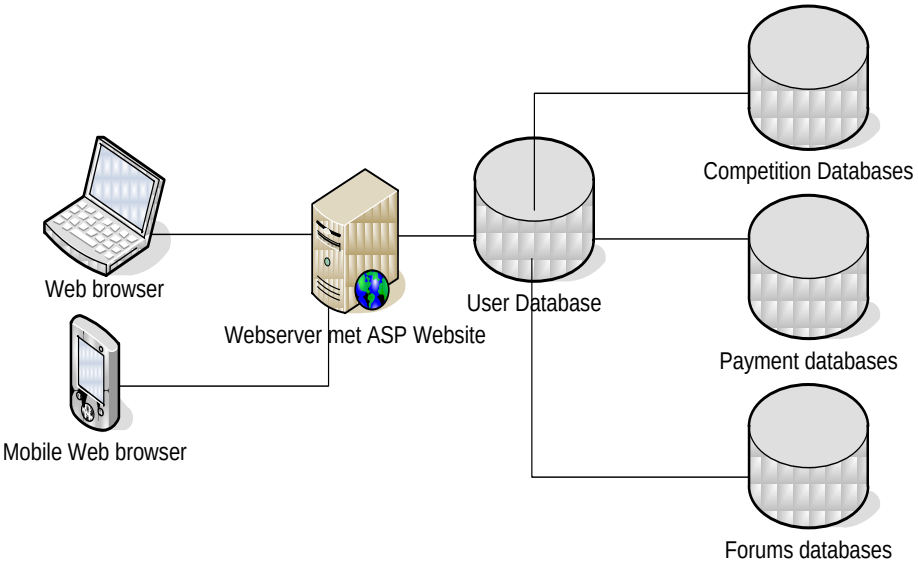
Figuur 2 - Overzicht competitietussenstand

Technische omschrijving

Het huidige systeem bestaat uit een aantal onderdelen:

- Desktop website.
- Mobiele website.
- Flash applicaties(widgets) die gebruikers in hun profiel of bijv. op Hyves kunnen integreren.
    - Wereldmap met overzicht in welke competitie doelstelling behaald is.
    - Profiel applicatie met competitie tussenstand, uitslagen en speelschema's.
- RSS Feeds
- IPhone app (eerste versie).

De Flash applicaties maken gebruik van pagina's die XML genereren met de benodigde data voor deze widgets. Zowel deze pagina's als de beide websites zijn geschreven in ASP. OnlineSoccerManager wordt op dit moment in bijna 20 talen aangeboden verspreid over verschillende webservers. Alle data wordt opgeslagen in verschillende MySQL databases (de grootste zijn opgenomen in figuur 3). Het plaatje hieronder beschrijft de huidige situatie. Op dit moment kan er vanaf een desktop en een mobiele browser verbinding gemaakt worden met een van de webservers. Deze maakt verbinding met de User Database om te bepalen bijv. welke competitie server de data voor de gebruiker staat.



Figuur 3 - Overzicht huidige architectuur

Transitie

Het transitie proces is ingegaan met het doel OnlineSoccerManager gebruik te laten maken van de nieuwste technieken. Dit betekent dat alle oude code uitgefaseerd wordt en vervangen wordt door nieuwe ASP.Net MVC code. Daarbij richt het transitie project zich op het samenvoegen van de code van de mobiele website en de normale website.

De scope van dit project is het maken van een API binnen de nieuwe ASP.Net MVC omgeving die het mogelijk maakt de functionaliteiten van OnlineSoccerManager aan te roepen vanuit externe applicaties. Deze ondersteuning bestaat in het huidige systeem nog niet en wordt ontwikkeld met oog op de toekomst. Naast het aanbieden van de functionaliteiten die beschikbaar zijn in de website krijgt de API ook de verantwoordelijkheid om de data aan te leveren die nodig is voor de Flash Widgets.

4 Plan van aanpak

In dit hoofdstuk wordt er beschreven hoe het plan van aanpak tot stand is gekomen en voor welke methodes en technieken er zijn gekozen.

4.1 Methoden en technieken

Hieronder volgt een beschrijving van de belangrijkste methoden en technieken die gebruikt zijn tijdens dit project en waarom deze gekozen zijn.

4.1.1 Methoden

- **RUP**
  Bij het kiezen van de project methode is er een overweging gemaakt tussen RUP, SCRUM en XP. De uiteindelijke keuze is RUP geworden omdat:
  - RUP gestructureerd documenteert en het gebruik van de verschillende fases duidelijkheid schept over wat er in een fase gaat gebeuren. Bij SCRUM wordt er per sprint bepaald wat er gaat gebeuren en wat er die sprint opgeleverd gaat worden. Aangezien dit project uit grote onderdelen bestaat is het moeilijk om iedere keer wat op te leveren.
  - Binnen de organisatie al ervaring met RUP wat de begeleiding vanuit het bedrijf makkelijker maakt.
  - XP (Extreme Programming) richt zich meer op puur programmeren dan op documenteren, wat het moeilijk maakt om competenties aan te tonen.
  - SCRUM en XP zijn deze methodes bedoeld voor projecten waarbij de requirements veel veranderen, de verwachting is dat dit binnen dit project weinig zal gebeuren, omdat de functionaliteiten die geïmplementeerd niet veranderen .
- **UML**
  Voor het modelleren van het eindproduct is er gekozen voor UML, dit omdat zich binnen UML een aantal structuur en gedragsdiagrammen bevinden die uitermate handig zijn voor het in kaart brengen van het eindproduct. Hierbij moet men vooral denken aan klassendiagrammen en sequentie diagrammen met betrekking tot de MVC structuur.
- **MoSCoW**
  Om de eisen een prioriteit te geven is er gekozen voor MoSCoW niet alleen omdat dit een van de meest gebruikte methodes is bij het analyseren van eisen voor software development, maar ook omdat het er voor zorgt dat er een duidelijk beeld geschetst kan worden met wat de belangrijkste eisen van een product zijn.

4.1.2 Technieken

- **C#**
  Het eindproduct dient geprogrammeerd te worden in Microsoft C#. Dit is een van de eisen die door de opdrachtgever aan het eindproduct gesteld is.
- **ASP.net MVC**
  De te implementeren API dient gebruik te maken van de functionaliteiten die ASP.Net MVC aanbiedt. Dit is besloten door de opdrachtgever, het gebruik van de MVC structuur zorgt voor een scheiding van verantwoordelijkheden binnen een systeem en is perfect voor uitbreidbaarheid.

- **JSON / XML**
  De data die een applicatie moet kunnen opvragen dient naast HTML ook aangeboden te worden in andere formaten. JSON en XML zijn de twee meest voorkomende formaten die door andere grote API's gebruikt worden en mede daarom een van de voornaamste eisen van de opdrachtgever. Bij grote API's moet gedacht worden aan de API's van bijvoorbeeld Twitter, Google of Hyves.
- **TDD**
  Er was al vastgelegd dat er gebruik gemaakt zou worden van unit testen om de kwaliteit van het product aan te tonen. Er is gekozen voor Test Driven Development omdat hierbij de unit tests worden opgesteld en de implementatie daarna pas volgt. Dit zorgt ervoor dat de eisen perfect in de applicatie verwerkt worden en men een duidelijk overzicht krijgt wat er al geïmplementeerd is en wat nog niet. Daarbij zit unit testen in de nieuwste versie van Visual Studio perfect geïntegreerd en is het zonde niet gebruik te maken van de aangeboden functionaliteiten.
- **GIT**
  Binnen Gamebasics wordt gebruik gemaakt van GIT, hier zal ik dus ook gebruik van moeten maken. GIT wordt gebruikt als versiebeheer voor de broncode. Binnen dit versie beheer zal voor dit project een aparte branch gemaakt worden. Op deze manier kunnen updates aan de broncode in deze branch worden gehaald, maar zal de "main" branch geen last hebben van de wijzigingen die dit project teweeg brengen tot op het moment dat de branches samen worden gevoegd.

4.2 Activiteiten

Hieronder volgt een beschrijving van de activiteiten die de student uit zal voeren tijdens het afstudeer traject en welke van de bovenstaande methodes of technieken er bij deze activiteiten gebruikt zullen worden.

- **Inlezen in de omgeving**
  Tijdens de Inception fase is er tijd ingepland voor de student om zich bekend te maken met de bestaande omgeving. Achter Online Soccer Manager draait een systeem wat meerdere databases host op verschillende servers, de werking hiervan begrijpen wordt nodig geacht om verdere problemen in het project te voorkomen. Daarbij is het werken met ASP.Net MVC nieuw en is er tijd gereserveerd om hiermee bekend te raken.
- **Plan van aanpak (Inception fase)**
  Aan het begin van het project wordt een plan van aanpak opgesteld om er voor te zorgen dat er duidelijkheid is tussen de opdrachtgever en de opdrachtnemer met betrekking tot welke activiteiten er wanneer uitgevoerd worden.
- **Definitie van eisen (Inception fase)**
  Om er voor te zorgen dat er duidelijk is wat er geïmplementeerd moet worden, wordt er aan het begin van het project in overleg met de opdrachtgever een lijst met functionele en technische eisen opgesteld en deze zullen een prioriteit krijgen.
- **Functioneel en technisch ontwerp (Elaboration fase)**
  Tijdens de Elaboration fase worden de eisen die in de Inception fase zijn opgesteld omgezet in een ontwerp.

- **Prototyping (Elaboration fase)**
  Aan het einde van de Elaboration fase zal er tijdens het ontwerpen gewerkt worden aan een prototype om te kijken of hetgeen wat ontworpen wordt goed werkt.
- **Testen (Construction fase)**
  Tijdens de gehele Construction fase zal de student zich bezig houden met het uitvoeren van unit tests over wat er geïmplementeerd is.
- **Implementatie (Construction fase)**
  Na goedkeuring van het ontwerp kan er begonnen worden met de implementatie. In de Elaboration fase wordt er besloten of er meerdere Construction fases doorlopen zullen worden met elk hun eigen eindproducten.
- **Overdracht en acceptatie (Transition fase)**
  Na de Construction fase zal het opgeleverde product aan een acceptatietest onderworpen worden. Deze test zal de eisen bevatten die in de Inception fase opgesteld zijn. Tevens zal er in deze fase een document opgesteld worden waarin de in productie name van het eindeproduct beschreven staat.

4.3 Op te leveren producten

Tijdens dit project zal een reeks aan producten worden opgeleverd. Hieronder staat per RUP fase besproken welke producten opgeleverd worden:

- Inception Fase.
  - Plan van Aanpak
  - Inception Document
- Elaboration Fase.
  - Elaboration Document
  - Prototype
- Construction Fase.
  - Construction Document
  - Testplan
  - Code
  - Handleiding
- Transition Fase.
  - Acceptatietest
  - Transition Document

4.4 Risicomanagement

Het managen en goed omschrijven van de risico's is een van de belangrijkste dingen binnen een project. Als de risico's van te voren niet duidelijk vastgelegd worden en er dus niet nagedacht wordt over wat er gedaan kan worden om risico's te voorkomen is er een veel grotere kans dat deze risico's ook daadwerkelijk voorkomen. Hieronder beschrijf ik kort de belangrijkste risico's die opgenomen zijn in het projectplan en waarom deze van groot belang zijn voor het succesvol afronden van het project.

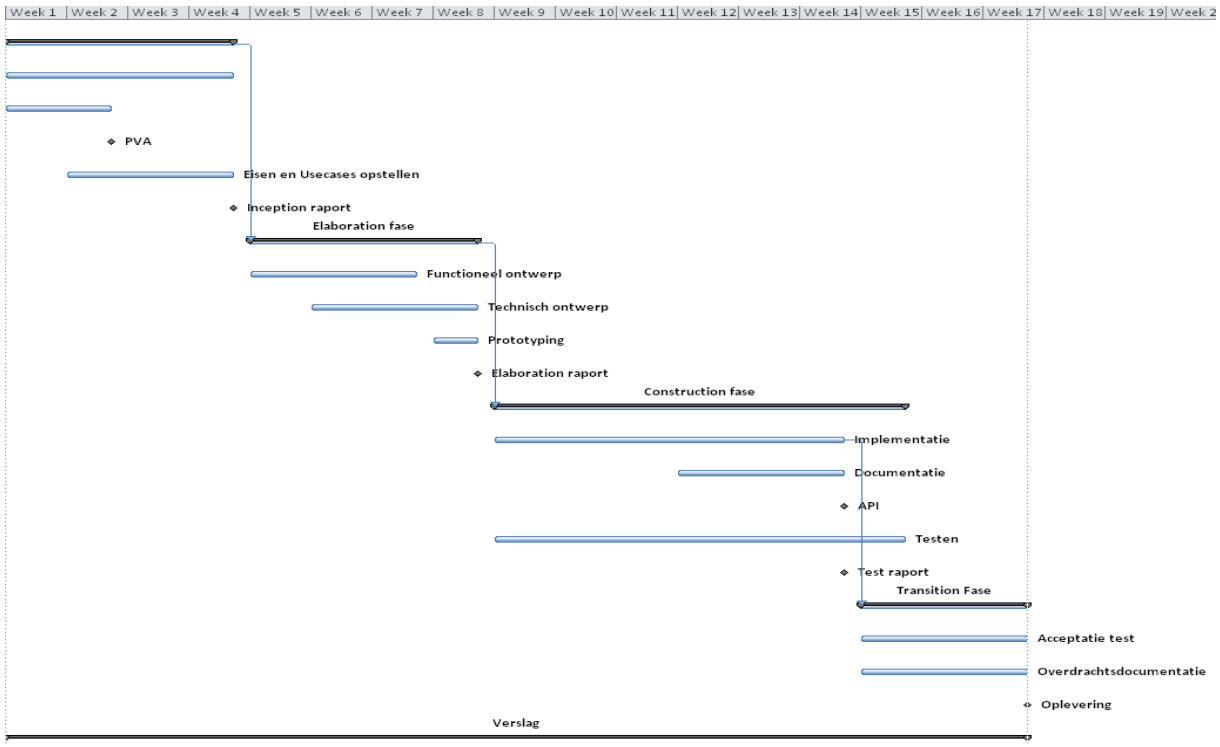
- **API voldoet niet aan de eisen**
  Een van de belangrijkste dingen om in het oog te houden is dat API ook daadwerkelijk voldoet aan de eisen van de opdrachtgever(voor alle eisen zie bijlage: Inception Rapport).

Het is dus van belang dit van te voren goed overeen te stemmen. Aan het eindproduct zitten veel eisen (veel functionaliteiten). Naast het van te voren duidelijk afstemmen van de eisen is het belangrijk wekelijks contact te houden met de opdrachtgever. Daarbij zal er volgens de RUP methode bij het ingaan van een nieuwe fase de Requirements opnieuw geëvalueerd worden.

- **API werkt niet met Flash Applicaties**
  Het is van belang dat de API de bestaande Flash applicaties goed ondersteund, aangezien deze door honderd duizenden spelers gebruikt worden. Het is dus nodig dat deze zorgvuldig getest worden en dat er vroeg aan de overdracht begonnen wordt.
- **Gebrek aan kennis**
  Een andere belangrijke factor om het project tot een goed einde te laten komen is dat de student voldoende kennis moet hebben van ASP.Net MVC en randzaken. Om deze reden is er ook voldoende tijd ingepland om er voor te zorgen dat deze kennis vergaard kan worden.

4.5 Planning

De planning die voor dit project is opgesteld is gemaakt in Microsoft Project, dit omdat het wijzigen van planningen hier zeer gemakkelijk is en het een duidelijk verloop genereert. De planning is al grotendeels vastgelegd in het afstudeerplan.



Figuur 4 - Planning

Hierboven een weergave van de planning conform het afstudeerplan. Deze planning wordt naarmate het project steeds gewijzigd indien nodig. De Construction fase is nu een grote fase omdat het zonder inzicht in ASP.Net MVC moeilijk te zeggen is hoe de Construction fase het beste opgedeeld kan worden.

5. Inception fase: Het beschrijven van de API

In dit hoofdstuk worden de activiteiten beschreven die uitgevoerd zijn tijdens de Inception fase en welke keuzes er gemaakt zijn.

5.1 Opstellen use cases

Hieronder volgt een beschrijving van de stappen die genomen zijn bij het opstellen van de use cases tijdens de Inception fase.

5.1.1 Eerste versie uce cases

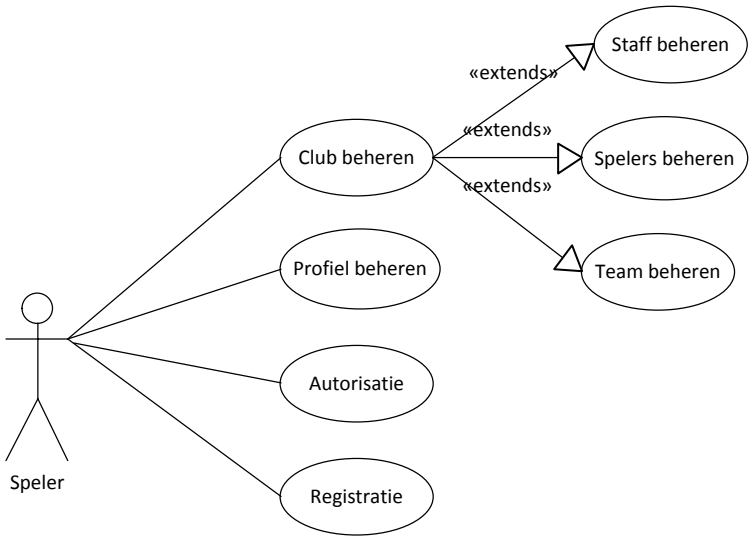
Bij het opstellen van de use cases is het van belang de bestaande functionaliteiten goed in kaart te brengen, dit om er voor te zorgen dat er een accurate planning gemaakt kan worden en er in het ontwerp rekening gehouden kan worden met alle functionaliteiten. Om alle functionaliteiten goed in kaart te brengen is er gekeken naar de verschillende versies van Online Soccer Manager:

- http://www.onlinesoccermanager.nl
- http://www.osmmini.nl (mobiele versie)

Bij het bekijken van de functionaliteiten werden de acties die gebruikers konden uitvoeren opgenomen in een use case model. Het probleem wat zich hier voor deed was dat de gebruiker veel kleine acties had en de use case dus groot en onduidelijk werd. In overleg met de opdrachtgever is er besloten om functionaliteiten te groeperen om zo meer duidelijkheid te krijgen.

5.1.2 Gespecificeerde use cases

Bij het opdelen van de Usecases in meer specifieke use cases is gekeken naar gemeenschappelijke functionaliteiten. Een voorbeeld hiervan is functionaliteiten die te maken hebben met het beheren van het team.



Figuur 5 - Algemene usecase

De use case hierboven beschrijft deze algemene functionaliteiten. Deze acties zijn in het Inception document terug te vinden en zijn in dit document uitgewerkt in gedetailleerde use cases. Hieronder

volgt een korte beschrijving welke functionaliteiten waar ingedeeld zijn zodat een duidelijk beeld gevormd kan worden bij deze use case.

- **Profiel beheren**
  Hieronder vallen alle functionaliteiten met betrekking tot het wijzigen van NAW gegevens en het instellen van de vakantiemode.

- **Autorisatie**
  Hieronder valt het inloggen. Dit is normaliter Authenticatie, maar bij de API log je niet echt in, maar wordt er gekeken of je rechten hebt om een functionaliteit uit te voeren.

- **Registratie**
  Het registreren van een account is een losse usecase omdat dit het enige moment is waarop een gebruiker niet ingelogd is of een account heeft.

- **Club beheren**
  Hieronder valt het kiezen of verlaten van een team, het aanwijzen van sponsors en het uitbreiden van het stadion.
  - *Staff beheren*
    Hieronder valt het aannemen van staff leden en het uitvoeren van hun specifieke functionaliteiten
  - *Spelers beheren*
    Hieronder valt het verkopen en kopen van spelers, het trainen van spelers en het bekijken van statistieken van spelers.
  - *Team beheren*
    Hieronder vallen alle functionaliteiten met betrekking tot de opstelling, tactiek en formatie.

Het opstellen van de use cases op deze manier heeft er toe geleid dat de functionaliteiten en acties duidelijk bij elkaar stonden en op elkaar aansloten. Bij iedere use case is een use case omschrijving toegevoegd waarin in het algemeen de actie beschreven staat, een uitgebreide beschrijving van de stappen die een Actor moet doen, welke aannames er gedaan worden en wat het resultaat is van een actie.

5.2 Technische en Functionele eisen opstellen

Tijdens de Inception fase is er gewerkt aan het opstellen van de functionele en technische eisen. Een aantal van deze eisen waren al duidelijk voor het project begon, het gebruik van een aantal technieken. Hetgeen belangrijk was voor de opdrachtgever was dat het met de API mogelijk was de website zo volledig mogelijk te bedienen. Dit op zichzelf is geen goed uitgewerkte eis. Met behulp van de opgestelde Use cases en overleg met de opdrachtgever zijn de eisen verder uitgewerkt (voor een complete lijst zie Inception Document). Hieronder worden kort de belangrijkste eisen samengenomen om een beeld te schetsen wat er met de API mogelijk zal zijn aan het einde van het project.

- Data moet in minimaal XML en JSON terug gegeven kunnen worden.

- API moet kunnen bepalen welke formaten verstuurd/aangevraagd moeten worden en of een aanvraag de verplichte parameters meestuurt.
- Het moet mogelijk zijn voor een gebruiker zijn team te managen (opstelling, formatie, tactiek, spelers trainen).
- Het moet mogelijk zijn voor een gebruiker personeel aan te nemen om hem te assisteren bij het managen van zijn team.
- De API moet de mogelijkheid bevatten om data in XML terug te sturen waarmee de Flash Applicaties (Legacy Widgets) overweg kunnen.
- Gebruiker moet zich kunnen Autoriseren.

5.3 Vooronderzoek

Tijdens de Inception fase is er tijd gereserveerd voor een vooronderzoek. Tijdens dit vooronderzoek is er gekeken naar de algemene werking van ASP.Net MVC en naar specifieke technische oplossing die voor de API van belang zijn en de motivatie waarom dit is.

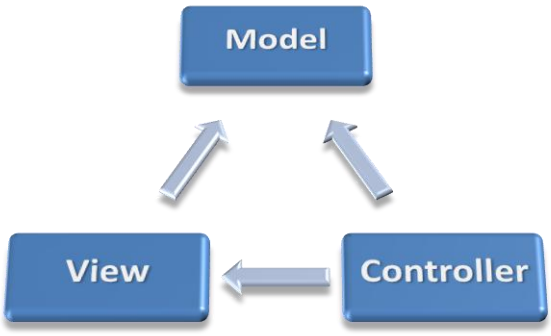
Motivatie planning

De keuze voor het plannen van het vooronderzoek aan het einde van de Inception fase, en niet ervoor, is gemaakt omdat eerst duidelijk moest worden wat er daadwerkelijk voor functionele en technische eisen gesteld zouden worden aan het eindproduct. Nadat deze eisen duidelijk waren kon er samen met de use cases een duidelijk beeld gevormd worden wat er mogelijk moet zijn binnen de API en welke technische kennis nodig zou zijn om de API te kunnen implementeren.

5.3.1 Inlezen ASP.Net MVC

Op de officiële Microsoft ASP.Net website staan verschillende series tutorials waarin de basis van MVC beschreven wordt tot aan geavanceerde onderwerpen over hoe je een complete webwinkel implementeert. Er is voor gekozen om deze tutorials te volgen, om zo bekend te raken met ASP.Net MVC. Daarbij staan er op de officiële site verschillende video tutorials uitgebracht door Microsoft MVP's, het bleek echter dat deze veel al dezelfde informatie boden dan de tutorials.

Om ervoor te zorgen dat er duidelijkheid was over hoe bepaalde "problemen" geïmplementeerd konden worden, is het nodig om eerst te begrijpen wat MVC is. Het MVC design pattern is een pattern waarbij een applicatie opgedeeld wordt in drie lagen met verschillende verantwoordelijkheden: data laag (Model), presentatie laag (View) en applicatielogica laag (Controller).



Figuur 6 - Interactie MVC componenten

- **Models**
  Models zijn de objecten die verantwoordelijk zijn voor de data, dit wil zeggen dat in het model de interactie met de databron zit, dit kan een database zijn maar ook bijvoorbeeld een XML bestand.
- **Views**
  Views zijn de componenten die verantwoordelijk zijn voor de weergave van de data afkomstig uit het model
- **Controllers**
  Controllers zijn de componenten die aan de hand van gebruikers acties bepalen welke views er gebruikt worden en welke models er aangesproken worden.

Het gebruik van MVC zorgt ervoor dat code leesbaarder en beter te begrijpen is omdat ieder onderdeel specifieke functionaliteiten heeft. Daarbij zorgt het ervoor dat code in hogere mate herbruikbaar is dan met bijvoorbeeld ASP.Net webpages, hierbij is er geen onderscheid in lagen van functionaliteiten.

Binnen MVC is het eenvoudig verschillende varianten van één website uit te rollen, denk hierbij aan een mobile en een vaste browser versie, omdat er alleen verschillende views gemaakt hoeven te worden. De logica hierachter kan voor beide versies hetzelfde blijven. Het opdelen van de applicatie zorgt ervoor dat alle onderdelen los goed testbaar zijn.

Binnen het kader van dit project zal er weinig gebruikt gemaakt worden van views in de vorm van HTML pagina's. Het doel is er voor te zorgen dat Online Soccer Manager ook bereikbaar wordt vanaf diverse sociale media en mobiele applicaties, hiervoor worden er dus geen echte views naar externe applicaties gestuurd maar bijvoorbeeld XML bestanden met data. Op deze manier kan ASP.Net MVC ingezet worden om zowel webpagina's te genereren maar ook om als Webservice gebruikt te worden. Op deze manier kan er voor bijvoorbeeld IPhone gebruik gemaakt worden van de ASP.Net MVC omgeving en kan de IPhone zelf de views regelen.

5.3.2 Technisch vooronderzoek

Het doel van dit vooronderzoek was het verkrijgen van kennis die gebruikt kon worden bij het opzetten van de API. Dit hield zich voornamelijk bezig met technieken binnen ASP.Net MVC, maar ook met Database interactie en Autorisatie. De volgende onderwerpen zijn bekeken:

- ViewData en ViewModels.
- ActionFiltering.
- ActionResults.
- Routing.
- Serialisatie en Deserialisatie van objecten.
- Het Repository Pattern.
- Autorisatie.
- Unit testen en TDD.

Hieronder volgt per onderzocht punt kort wat het inhoud en waarom hiernaar gekeken is en hoe dit van toepassing is op dit project. In de bijlage is extra uitleg met code voorbeelden over deze technische aspecten terug te vinden.

ViewData en ViewModel

Deze technieken zijn twee manieren waarop je data vanuit je controller naar je view kan sturen. Beide technieken hebben voor en nadelen. ViewData is een collectie waaraan je waardes kan toevoegen. ViewModels zijn objecten waarover je zelf controle hebt en kunnen daarom meerdere types tegelijk bevatten. Om de volgende redenen is er voor ViewModels gekozen:

- Controllers acties blijven schoner omdat je niet alle waardes los aan een collectie hoeft toe te voegen.
- Het geeft meer vrijheid in opbouw en ondersteuning van verschillend data.
- In het kader van de API zullen in ieder geval objecten naar XML of JSON omgezet moeten worden, wat met ViewModels zeer goed mogelijk is.

ActionFiltering

ActionFiltering is een techniek waarbij met een zogenaamd “Attribuut” extra functionaliteit aan een object of een methode toegevoegd kan worden. De ActionFilter bied de mogelijkheid om zowel voor als na het uitvoeren van een object of methode deze extra functionaliteiten toe te voegen. Het grote voordeel van deze methode is dat een attribuut makkelijk te hergebruiken is en het ervoor zorgt dat je code niet op meerdere plaatsen opnieuw hoeft te maken. Dit gaat in dit project gebruikt worden om:

- Het inlezen van een “API Call” en het terug sturen van de aangevraagde data zal voor iedere methode hetzelfde zijn. Deze functionaliteiten kunnen dus in een attribuut gegroepeerd worden waardoor de onderhoudbaarheid van de API goed blijft.
- Veel functionaliteiten hebben autorisatie nodig. Door ook deze functionaliteiten in een attribuut te zetten kunnen functionaliteiten simpel wel of niet van autorisatie voorzien worden. Daarbij zit alle logica en beslispunten voor het autoriseren van gebruikers op maar één locatie.

ActionResult

In ASP.Net MVC geven controllers objecten terug van het type ActionResult. Het is niet onmogelijk om objecten van andere types terug te geven, maar binnen de “flow” van ASP.Net MVC is het gemakkelijker ActionResults te gebruiken. Het is echter wel mogelijk om eigen implementaties van ActionResults te maken door middel van overerving. Binnen het kader van de API wordt er gedacht aan de volgende overgeërfde ActionResults:

- Een samengesteld result, dit zal een samenstelling zijn tussen een berichtgeving of het uitvoeren succesvol was en de reden daarvoor, en uitvoer van de methode.
- Een kaal result, dit wordt een lege container waar alleen een object in gedaan kan worden en waarbij geen extra bericht meegestuurd kan worden.

Routing

Routing is een techniek binnen ASP.Net MVC waarmee bepaald kan worden hoe een aanvraag door een applicatie naar de goede controller gestuurd kan worden. De standaard route in ASP.Net ziet er als volgt uit:

Controller/Action/Id (Bijv.: Krant/Artikel/3)

In de functionele eisen is opgenomen dat het mogelijk moet zijn in verschillende formaten data terug te geven aan gebruikers. Dit zou kunnen door in de naam van de methode het type op te nemen, maar dan zou men wanneer er een nieuw formaat bijkomt alle actions moeten wijzigen. Om deze reden is er gekeken hoe de volgende route gemaakt kon worden:

Controller/Action/Format/Id (Bijv.: Krant/Artikel/Xml/3)

In eerste instantie was er gekeken naar een route waarin je het formaat kon meegeven door middel van een "." in plaats van een "/", maar dit bracht problemen met zich mee binnen ASP.Net MVC. Het idee om het formaat mee te sturen in de route is afgekeken van de Twitter en Google API. Deze oplossing zorgt ervoor dat het altijd duidelijk is welk formaat er aangevraagd word en de verantwoordelijkheid voor het formaat ook bij de eindgebruiker ligt.

Serialisatie en Deserialisatie van objecten.

Serialisatie en Deserialisatie is het proces om een object in Code om te zetten naar een variant die over het web verstuurd kan worden. Dit wordt veel in XML en JSON gedaan. In de eisen functionele eisen stonden een aantal formaten, per formaat wordt hieronder de methode van Serialisatie en Deserialisatie besproken.

- JSon
  Binnen C# kan een `JavaScriptSerializer` van JSon een object maken en andersom, dit is de enige standaard methode die aanwezig is, maar voldoet aan de eisen.

- XML
  Binnen C# zijn er twee XML Serialisatie methodes, de standaard `XMLSerializer` en de nieuwere `DataContractSerializer`. De overweging die hierbij de doorslag heeft gegeven is het feit dat de standaard `XMLSerializer` alles serialiseert, tenzij je zegt dat hij dit niet moet doen. Bij de `DataContractSerializer` moet er specifiek voor ieder object ingesteld worden wat er geserialiseerd moet worden. De `DataContractSerializer` is wel 10% sneller, maar er is gekozen om dit met oog op de leesbaarheid van de code als acceptabel te beschouwen.

- Atom(RSS)
  Hiervoor is binnen C# geen standaard functionaliteit, de Serialisatie van dit formaat zal dus handmatig opgebouwd moeten worden.

Repository pattern

Het repository pattern is een design pattern wat gebruikt wordt om database functionaliteiten benaderbaar te maken. Een repository heeft ten minste standaard functionaliteiten om een bepaald model toe te voegen, te bewerken en te verwijderen. In de meeste gevallen zal een repository een "wrapper" zijn om een tabel. Deze wrapper wordt vanuit de controller aangesproken om een bepaald model uit de database terug te krijgen. Dit pattern dient gebruikt te gaan worden bij iedere database die op dit moment nog niet benaderbaar is als repository, op dit moment wordt voorzien minimaal de OAuth, Flash Widgets, Nieuws en Resultaten repositories moeten worden opgezet. Door het creëren van deze tussen laag is het voor het testen van de applicatie gemakkelijk mogelijk de applicatie af te koppelen van de database en te testen met waardes in het geheugen.

Figuur 7 - Plaatsing repository

Autorisatie

Binnen Online Soccer Manager zit veel van de functionaliteit gekoppeld aan een gebruikersaccount. Dit betekend dat er binnen de API een methode aangeboden moet worden waarmee gebruikers zich kunnen autoriseren. Het is wenselijk om bij een API de gebruikersnaam en wachtwoord van een gebruiker niet bij iedere gegevenstransactie mee te sturen, omdat dit soort transacties gevoelig zijn voor Phishing. Hierbij is er gekeken naar bestaande API's van Twitter en Google en is er gekeken hoe moeilijk het zou zijn een eigen implementatie te maken.

- OAuth
  OAuth is een open standaard voor het autoriseren van gebruikers. Met dit protocol geeft een gebruiker een applicatie van derden toegang tot hun account en wordt er hiervoor een token uitgegeven wat in de toekomst gebruikt kan worden als toegangsbewijs tot de data van een gebruiker.[1]

- OpenID
  OpenID is een gedecentraliseerd authenticatiemechanisme. Hierbij registreert een gebruiker zich bij OpenID en gebruikt vervolgens een token van hun om zich te kunnen authenticeren bij websites en maakt zo het gebruik van het versturen van gebruikersnamen en wachtwoorden overbodig.

- Eigen Implementatie
  Het bedenken van een eigen implementatie bleek al snel te ingewikkeld binnen de scope en geboden tijd van het project. De reeds bestaande technieken zijn dermate complex in de versleuteling van data en het genereren van één toegangstoken voor een gebruiker, dat er besloten is hier niet verder onderzoek naar te doen.

Uiteindelijk is er gekozen om gebruik te maken van OAuth omdat:

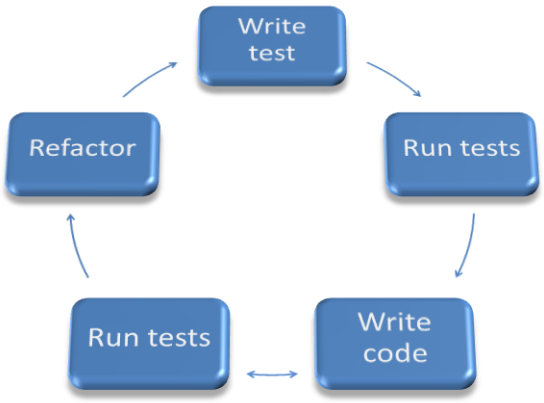
- OpenID verwacht dat een applicatie een account bij hun heeft om daarmee toegang te krijgen bij OnlineSoccerManager.
- De generatie van toegangsbewijzen gebeurt op de server, waardoor alle autorisatie intern binnen OnlineSoccerManager geregeld wordt.
- Voor een applicatie is het de makkelijkste methode, een gebruiker van een applicatie die gebruik maakt van de API, wordt een keer verzocht in te loggen op een speciale pagina. Deze pagina geeft een toegangsbewijs voor een bepaalde periode.

[1] Verdere informatie met betrekking tot de techniek van OAuth is opgenomen in de Elaboration Fase en Construction Fase.

Unit testen en Test Driven Development.

Het principe van Test Driven Development is gebaseerd op het uitvoeren van kleine iteraties. Bij Test Driven Development worden Unit Tests opgesteld voordat er überhaupt begonnen wordt met het implementeren. Het doel is om van deze methode gebruik te maken bij het programmeren van de API. Hieronder een basis flowchart van het proces wat doorlopen wordt bij Test Driven Development.

1. **Write a test**
   Bij TDD wordt als eerste een test opgesteld van de feature die geïmplementeerd gaat worden. Deze test faalt altijd.
2. **Run tests**
   Valideren dat de nieuwe test ook faalt en vorige tests nog steeds dezelfde resultaten geven.
3. **Write code**
   Code schrijven die ervoor zorgt dat de test niet meer faalt.
4. **Run tests**
   Kijken of alle tests succesvol doorlopen worden, zo ja naar Refactor, zo nee naar Write Code
5. **Refactor**
   Code opschonen, zodat het klaar is voor release, en voorzien van commentaar.



Figuur 8 - Flowchart Test Driven Development

Het constant uitvoeren van deze tests lijkt overbodig, maar het zorgt ervoor dat alle eisen goed getest kunnen worden en dat ook duidelijk te zien is wanneer er een functionaliteit niet meer werkt na het implementeren van een andere functionaliteit. Een van de veel beschreven nadelen van TDD is dat de programmeur zelf ook tester is. Aangezien dit geen bezwaar is voor dit afstudeer project, aangezien er aan competenties voldaan moet worden, is er gekozen om volgens de methode te werken.

Visual Studio biedt een uitgebreide unit test functionaliteit aan waarbij het mogelijk is snel en gemakkelijk unit tests op te stellen en de resultaten in een duidelijk overzicht te krijgen. Binnen ASP.Net MVC worden unit tests uitgevoerd op de controllers. Om bepaalde functies echter te testen moet er gebruik gemaakt worden van FakeImplementaties. Dit omdat testen op de productie omgeving traag kan zijn en in gevallen van database connecties niet mogelijk.

6. Elaboration fase: Ontwerpen van de API

In dit hoofdstuk worden de werkzaamheden die uitgevoerd zijn tijdens de Elaboration fase beschreven. Tijdens de Elaboration fase is er gewerkt aan het ontwerp voor API en is er gewerkt aan een Prototype.

6.1 Overzicht werkzaamheden

Tijdens de Construction fase zijn alle werkzaamheden van de API ontworpen. Dit viel uiteen in een aantal losse onderdelen die samen de uiteindelijke API vormen. Deze onderdelen zijn:

- De controllers met functionaliteiten voor de Flash Widgets.
- De controllers met functionaliteiten “om de website te bedienen”.
- De ApiBind ActionFilter, die het mogelijk maakt om Controller Actions toe te voegen aan de API.
- De autorisatie met OAuth.

Hieronder wordt per onderdeel kort uitgelegd hoe het ontwerp in elkaar zit en waarom hiervoor gekozen is.

6.1.1 ApiBind

Zoals eerder beschreven wordt er gebruik gemaakt van een ActionFilter die verantwoordelijk is om van een Controller Action een API Action te maken. Deze ActionFilter zal vanaf nu ApiBind genoemd worden. De ApiBind filter is verantwoordelijk voor de volgende functionaliteiten:

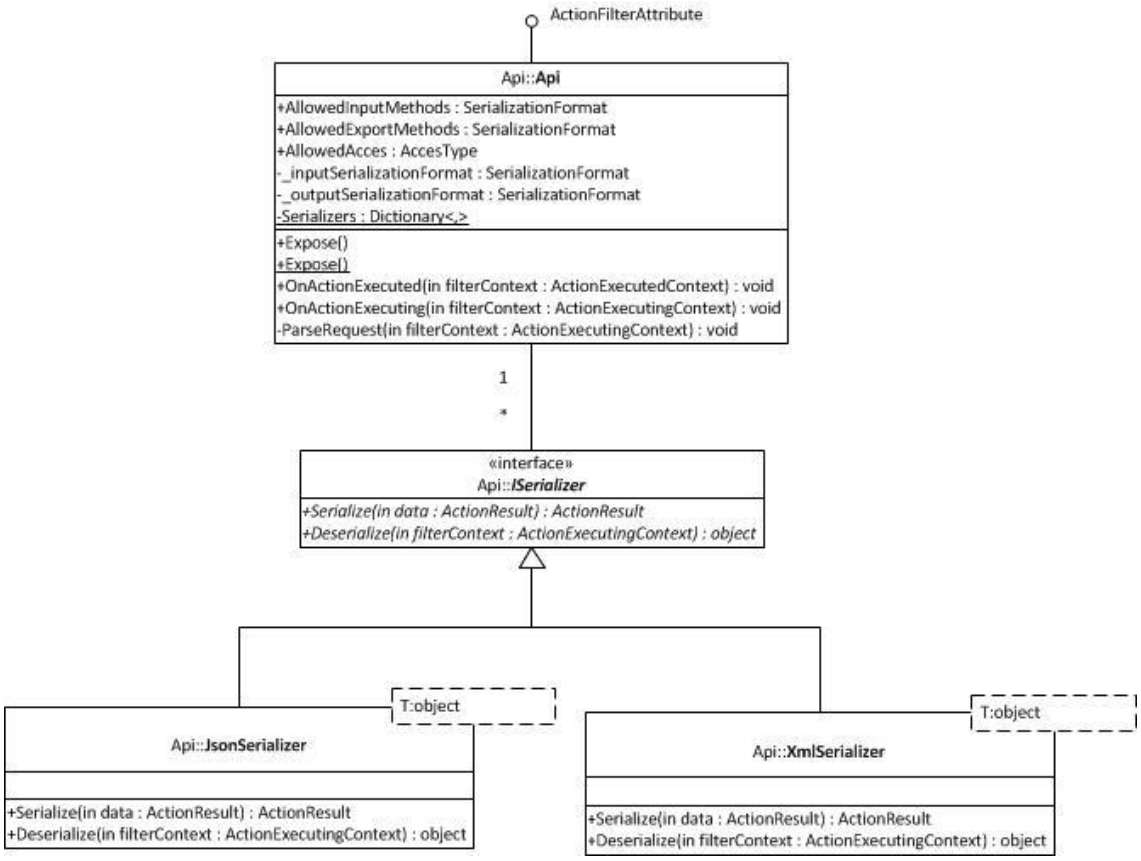
- Bepalen welk formaat er naar de API verstuurd wordt.
- Bepalen welk formaat de API terug moet sturen.
- Bepalen of een API Call valide is:
    - Zijn de formaten toegestaan?
    - Zijn alle verplichte parameters ingevuld?
- Het daadwerkelijk omzetten van Objecten naar XML/JSon varianten.

Doordat de ApiBind deze verantwoordelijkheden heeft is het zeer makkelijk om van een Controller Action een Api Action te maken. Namelijk door boven een Methode in een class `[ApiBind]` neer te zetten. Hiermee is de basis voor het aanmaken van API Calls gemaakt. Er is echter gekeken naar het instellen van deze filter om het mogelijk te maken om:

- Toegestane formaten voor invoer en uitvoer in te stellen.
- Aan te geven of data met Form Post verstuurd mag worden.
- Het koppelen van nieuwe Serialisatie methoden.

Hieronder wordt het ontwerp van deze filter weergegeven. Dit ontwerp wordt getoond omdat het de kern van het systeem omvat en het belangrijk is duidelijk te hebben hoe dit functioneert. Het is goed te zien dat de ApiBind (Hieronder nog Api genoemd, maar deze naam is later veranderd), meerdere Serialisatie methodes ondersteunt.

Ook al viel optimalisatie buiten de scope van dit project is er hier toch een optimalisatie toegepast. De Serialisatie methodes worden opgeslagen in een statische collectie, afgekeken van het Singleton Principe. Dit betekent dat iedere instantie van de ApiBind gebruik maakt van dezelfde Serialisatie objecten en dat deze dus maar 1 keer aangemaakt worden.



Figuur 9 - ApiBind model

6.1.2 ActionResults

Eerder werd verteld dat er gebruik gemaakt zou worden van een aantal mogelijke resultaten. Hieronder wordt kort gesommeerd welke dit zijn en waarom:

- CompositeActionResult
  Dit is een samengesteld object bestaand uit berichtgeving over de aanroep en het resultaat van de aanroep. Dit is het resultaat wat standaard terug gegeven gaat worden.

- PlainActionResult
  Dit resultaat wordt gebruikt om puur data terug te geven in XML of JSon. Dit object biedt geen mogelijkheid om een bericht terug te koppelen naar de applicatie. Dit result zal vooral gebruikt worden in de Controller Actions die voor de Flash Widgets omdat deze een specifiek formaat van XML verwachten.

- RssActionResult
  Dit resultaat wordt gebruikt op de punten waar RSS terug gegeven dient te worden. Er is gekozen om hier een los resultaat van te maken omdat het vrijwel niet mogelijk is om van ieder model een RSS feed te generen (velden als Entry, Date en Title moeten worden ingevuld). Daarom is er gekozen voor een RssActionResult wat RSS feeds kan genereren.

Er is voor bovenstaande resultaten gekozen omdat op deze manier op ieder gewenste manier data kan worden terug gestuurd.

6.1.3 Gebruik van Area.

Binnen ASP.Net MVC bestaat de mogelijkheid om een Area aan te maken. Een Area wordt gezien als een stuk code wat los staat van de rest van een project en in theorie hergebruikt moet kunnen worden. Het is hiermee echter ook mogelijk om controllers met dezelfde naam te herimplementeren. De reden dat er gekeken is naar een Area was omdat de Controllers met de API in hetzelfde project moesten komen als de website Controllers.

Het eerste doel was om Controller Acties zo op te zetten dat ze voor zowel API als Website toegangspunt gebruikt konden worden. In overleg met de opdrachtgever is er echter uiteindelijk besloten dat dit geen wenselijke oplossing zou zijn. Dit zou namelijk betekenen dat iedere wijziging in de website direct in de API zou komen. Dit zou problemen kunnen vormen voor bijvoorbeeld een IPhone App.

Door het gebruik van een Area kunnen Controllers opgezet worden die specifiek voor de API zijn. Dit zou kunnen gaan leiden tussen enige dubbele code tussen de Website en Api controllers, maar in dit geval bleek dit de enige oplossing.

6.1.4 Ontwerpen van ondersteuning Flash Widgets (Legacy)

Het lag vast in de eisen dat de API de mogelijkheid kreeg data te generen voor de Flash Widgets. Deze functionaliteiten vallen uit een in de volgende onderdelen:

- Het generen van een nieuws en resultaten overzicht in RSS.
- Een XML bestand voor de WorldMap Widget.
- Verschillende XML results voor profile, team en managergeschiedenis.
- De koppeling tussen widget en manager.
- Vertalingen van Widgets.

In de oude omgeving werd gebruik gemaakt van ASP pagina's die handmatig XML of RSS opbouwen. In C# wordt er gebruik gemaakt van Serialisatie. Met deze techniek is het mogelijk om van object XML of JSON te maken. Omdat er in de ApiBind al ondersteuning zit om van een willekeurig object naar XML of JSON te gaan was het in eerste instantie niet nodig handmatig XML op te bouwen.

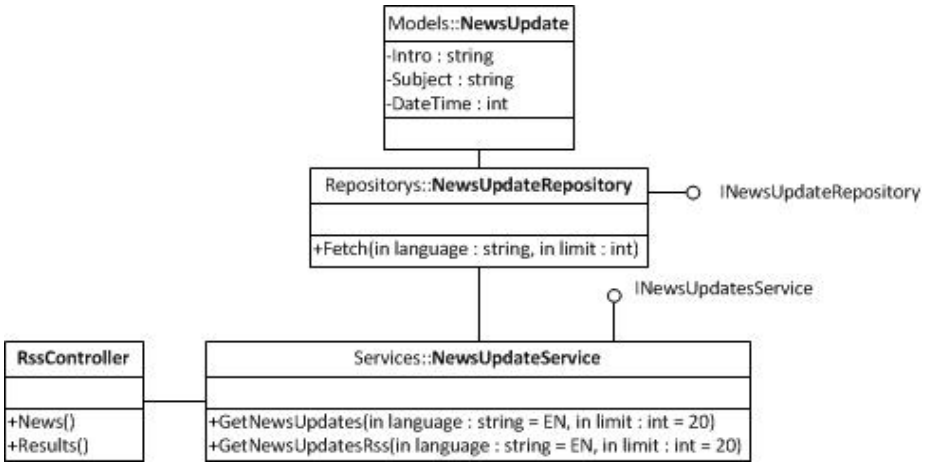
Het was de eis van de opdrachtgever dat de opbouw van de XML zo min mogelijk gewijzigd zou worden, om zo de integratie van de API in de Flash Widgets zo simpel mogelijk te maken. Op een aantal punten bleek dit echter niet mogelijk:

- In alle XML overzichten die voor de Flash Widgets gegenereerd werden stonden velden met HasNoTeam en NoResults. Buiten het feit dat dubbele ontkenningen niet handig zijn, worden in C# booleans standaard op false gezet. Dit leed ertoe dat wanneer een waarde niet ingevuld werd, deze dus false werd. Dit leed ertoe dat de output dan deed lijken dat er wel resultaten waren terwijl deze er niet waren. In overleg met de opdrachtgever is er besloten dat dit soort waardes naar HasTeam en HasResults omgezet mochten worden.
- Op sommige punten werden er in de XML ineens vertalingen opgenomen. Hiervoor was al een aparte ASP pagina die niet gebruikt bleek te worden. Dit gaf een probleem omdat bij bijvoorbeeld het team overzicht er spelers in de XML stonden. Dit speler object kon

statistieken bevatten, maar soms ook vertalingen. Binnen C# is er geen oplossing om een waarde soms een string te laten zijn en soms een int. In overleg met de opdrachtgever is er hier besloten om de vertaling weg te laten uit de XML en de verantwoordelijkheid hiervan in de Flash Widgets te leggen.

RSS

Er diende twee RSS feeds gegenereerd te kunnen worden. Zoals eerder beschreven is er gekozen voor een eigen ActionResult. Daarbij is er besloten om een controller te maken die deze RSS feeds maakt. Hieronder wordt het ontwerp getoond van de RSS feed voor het nieuws. Hierbij is duidelijk te zien hoe het repository pattern geïmplementeerd worden.



Figuur 10 - Implementatie Nieuws Rss Feed

WorldMap

De WorldMap Widget maakt gebruik van de AMMap library. Dit is een library die een bepaald formaat XML inleest om zo een wereldkaart met waardes erop te kunnen genereren. Er is gekozen om binnen de controller voor de WorldMap geen gebruik te maken van de ApiBind om de volgende redenen:

- De XML die AMMap verwacht is te complex, het maken van Objecten die geserialiseerd kunnen worden naar dat formaat maken het te moeilijk.
- AMMap kan alleen maar XML inlezen, het maken van een oplossing die meerdere formaten kan exporteren is dus niet nodig.

Vertalingen

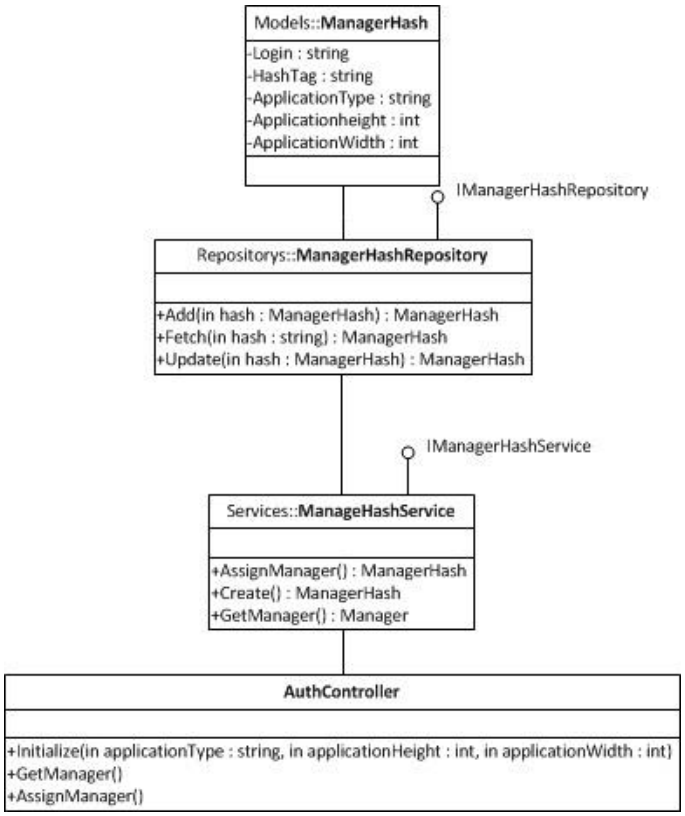
Bij het opzetten van de vertalingen is er gekeken naar hoe de vertalingen in de nieuwe website gemaakt worden. Deze worden gemaakt aan de hand van C# Resource files. Bij de Flash Widgets werden een aantal XML bestanden geleverd. Er is gekozen om deze om te zetten naar Resource files omdat het hiermee makkelijk is om tussen talen te wisselen. Er is gekozen om al deze XML files benaderbaar te maken vanuit één Translation Controller.

Koppelen van een Widget aan een Manager

De Flash Widgets maken gebruik van een eigen koppeling tussen een manager en een widget. Deze autorisatie gaat in de volgende stappen:

- Hash Aanvragen
- Inloggen en Hash aan Manager koppelen

Hieronder staat een schematische weergave van hoe deze manier van koppelen opgezet is. Daarbij is er ook weer duidelijk te zien hoe de repository geïmplementeerd wordt.



Figuur 11 - Overzicht Koppeling Widget aan Manager

6.1.5 Ontwerpen van overige functionaliteiten.

Bij het ontwerpen van de Controllers die het mogelijk moeten gaan maken de functionaliteiten van de website zo volledig mogelijk te benaderen is er gebruik gemaakt van de eerder opgestelde use cases. Daarbij is er gekeken hoe functionaliteiten opgebouwd waren in de oude website en hoe er binnen GameBasics aan de opzet van de nieuwe website gewerkt werd.

Volgens Microsoft en de algemene MVC standaarden hoort een controller maar met één model om te gaan en horen controllers weinig acties te hebben en de acties die er zijn horen weinig code te bevatten. Dit principe is bij het ontwerpen van de functionaliteiten niet gehandhaafd. Hierbij is er meer gekeken naar het indelen van functionaliteiten bij logische namen dan het creëren van compacte controllers. Dit is vooral gedaan om het voor Applicaties duidelijk te maken waar functionaliteiten terug te vinden zijn.

Hieronder wordt een korte lijst met de 4 belangrijkste controllers beschreven en welke functionaliteiten hierin zitten. Deze opzet zal zeer lijken op de Use cases in het Inception Document.

- Team
  - Lineup
  - Squad
  - Formation
  - Tactics
  - SecretTraining
  - TrainingsCamp

- Club
  - Sponsor
  - Stadium

- Staff
  - Hire
  - Sack
  - Overview
  - Doctor
  - Spy
  - Scout
  - CommercialManager

- Player
  - Overview
  - Buy
  - Sell

6.2 OAuth ontwerpen

OAuth is een open standaard die verschillende oude standaarden heeft opgenomen en verbeterd. OAuth wordt gebruikt wanneer applicaties van derde beschermde data willen aanroepen. Het standaard gebruikte voorbeeld van OAuth is een applicatie die gebruikt kan worden om op je social networking profile een foto share app te draaien. Deze applicatie heeft dan toegang tot je account nodig, wat met OAuth gerealiseerd kan worden.

Waarom OAuth?

OAuth is gekozen om ervoor te zorgen dat een applicatie kan aantonen dat het voldoende rechten heeft om op een bepaald account bewerkingen uit te voeren. Wanneer een gebruiker op de website inlogt wordt er een Sessie of een Cookie gemaakt wat er voor zorgt dat een gebruiker ingelogd blijft. Binnen een Api is dit niet mogelijk. OAuth is daarbij een methode die door veel grote API's zoals die van Yahoo, Google en Twitter ondersteund worden omdat dit een van de beste methodes is. OAuth maakt het mogelijk voor applicaties om met behulp van de API toegang te krijgen tot managers profiel en hiermee bijvoorbeeld bewerkingen uit te voeren op een team.

OAuth in het kort

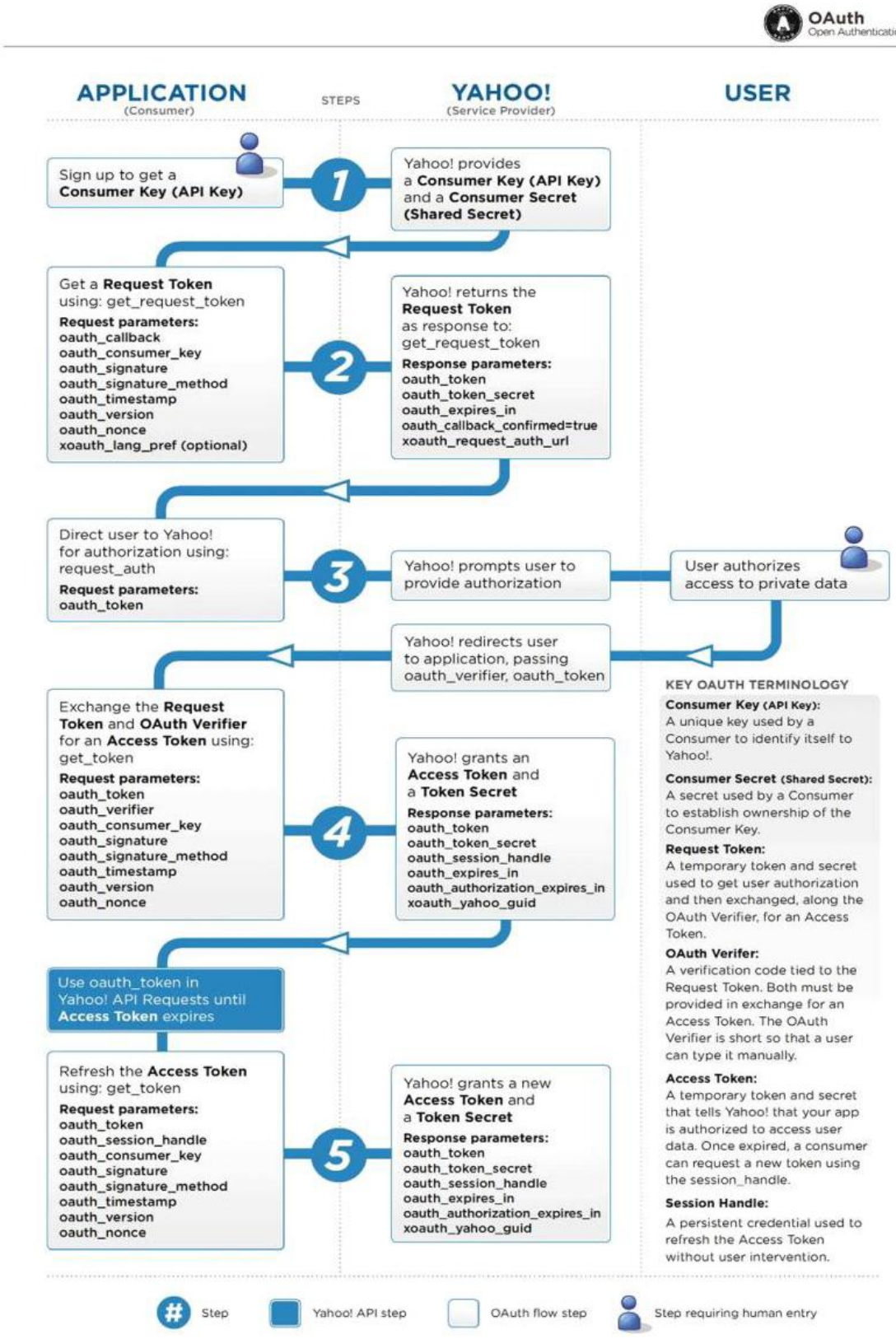
Binnen OAuth wordt gebruik gemaakt van twee combinaties van een sleutel en een key:

- Consumers
  Consumers zijn keys en sleutels die verleend worden aan applicaties van derden die gebruikt worden in het genereren van de OAuth handtekening.
- Tokens
  Tokens zijn de keys en sleutels die gebruikt worden om te bepalen voor welke gebruiker de OAuth handtekening is.

Wanneer een applicatie een aanvraag doet worden een aantal waardes samen genomen tot een handtekening, de server kan aan de hand van deze handtekening bepalen of de applicatie rechten heeft om de bewerking uit te voeren. Een valide aanvraag van OAuth bevat een aantal waarden, deze staan allen in onderstaande voorbeeld aanvraag:

http://voorbeeld.nl?oauth_timestamp=1296647328&oauth_nonce=bFNrtHT0GgNUiLxkS&oauth_version=1.0&oauth_signature_method=HMACSHA1&oauth_consumer_key=8cf4e795eec7af044b11d25385959c3704cee28b5&oauth_token=vsadsdg245t238ry0fahou9ghfiq3u2bg8702&oauth_signature=UVOqqO97yxFOrBzP%2BIP5rO3tmWo%3D

Het is de bedoeling dat voor iedere aanvraag alle OAuth waardes opnieuw gegenereerd worden en de handtekening opnieuw versleuteld wordt.



Figuur 12 - Autorisatie process met OAuth (Yahoo voorbeeld)

Validatie

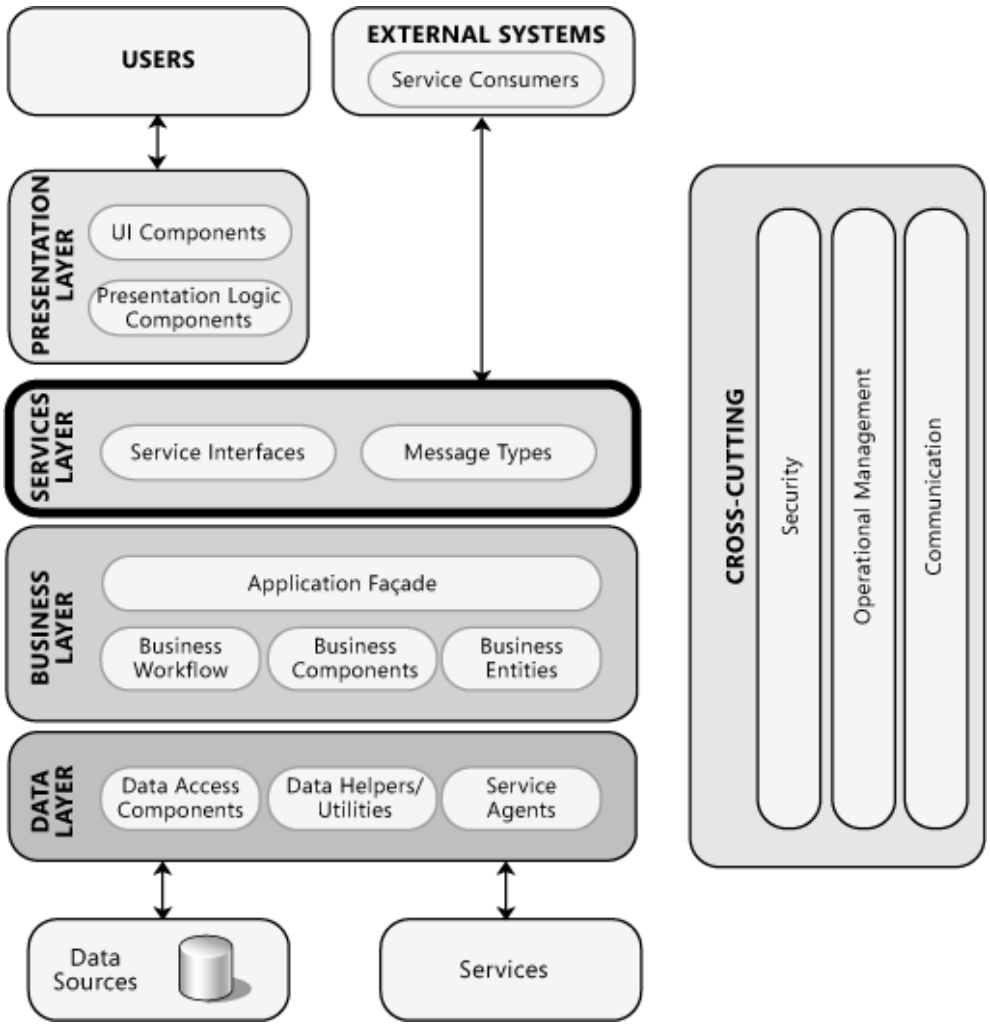
In het schema hierboven staat beschreven hoe het aanvragen van een OAuth Token gaat. Echter wanneer dit gedaan is moet er nog steeds bij iedere aanroep van de API gekeken worden of de OAuth aanroep valide is. Hiervoor is er gekeken naar een aantal technieken en wordt er vanuit gegaan dat de aanvraag opgesteld is zoals het eerder beschreven voorbeeld.

- Handtekening validatie
  Bij deze check wordt er gekeken of de consumer key en de token key bestaande waardes in de database zijn. Als dit zo is kan er met de waardes uit de aanvraag en de sleutels uit de database een handtekening nagemaakt worden, als deze hetzelfde zijn is de handtekening valide.
- Timestamp Check
  Bij deze check wordt naar twee tijden gekeken, als eerste of het token niet verlopen is (standaard is een token 1 jaar geldig), ten tweede wordt er naar de tijd van de aanroep gekeken. De timestamp van de aanroep mag niet meer dan 10 seconde oud zijn, dit om er voor te zorgen dat gebruikers niet maar één keer een handtekening generen en deze blijven gebruiken.
- Nonce validatie
  Bij deze check wordt er op de server een kleine cache aan gemaakt met Nonce waardes die reeds gebruikt zijn. Dit om er voor te zorgen dat ieder request wat gedaan wordt ook daadwerkelijk een nieuw request is.

De combinatie van bovenstaande methodes zorgt er voor dat het vrijwel niet mogelijk is te frauderen met de Tokens. Het was al voldoende geweest om alleen de Token Check te doen in combinatie met of de Timestamp Check of de Nonce check, maar voor de zekerheid is er gekozen om alle drie toe te passen.

6.3 Service Layer

Tijdens de Elaboration fase is er binnen GameBasics besloten om gebruik te maken van een zogenaamde Service Layer. Dit principe is afgeleid uit de Application Architecture Guide van Microsoft [2] . Onderstaand schema toont hoe de verschillende lagen verschillende verantwoordelijkheden hebben. Binnen het transitie project zal er gebruik gemaakt worden van een variant op deze Service Layer. Er is besloten om de Business Layer en de Service Layer in een laag te nemen, wat er toe leidt dat de Service Layer Business Logica bevat. De Service Layer is een extra laag die tussen de controllers en de data laag (repositories en models) komt te zitten. Dit betekent dat in controllers pure user interactie staat, in de services de business logica zit en in de repositories zit de database interactie.



Figuur 13 - Microsoft Architecture Guide voorbeeld

Aangezien het besluit om deze extra laag te implementeren tijdens de Elaboration fase genomen werd, kon dit ook meegenomen worden in het ontwerp voor de API. De richtlijn die opgesteld is, is dat iedere service minimaal verantwoordelijk is voor het benaderen van één repository.

[2]Microsoft Application Architecture Guide http://msdn.microsoft.com/en-us/library/ee658090.aspx

6.4 Prototyping

In de laatste weken van de Elaboration fase is er gewerkt aan een prototype. Binnen RUP is het gewoonlijk dat aan het einde van de Elaboration fase een werkend prototype opgeleverd wordt. Het prototype is niet alleen als proof of concept gebruikt maar ook om te kijken of het ontwerp überhaupt goed werkte. In het prototype zaten de volgende functionaliteiten:

- Aan de hand van de aanvraag bepalen in welk formaat er data verstuurd moet worden (Routing).
- Voorbeeld data kan opgevraagd worden, verstuurd (POST) worden niet.
- Voorbeeld Data terug sturen in XML formaat (Serialisatie).
- Basis versie van `ApiBind` (ActionFiltering).
- Test controller die gebruik maakt van `ApiBind` (ActionFiltering).

Er is gekozen om de bovenstaande functionaliteiten in het prototype op te nemen omdat dit de kern functionaliteiten zijn die van belang zijn. OAuth prototyping is niet gedaan omdat hier geen tijd meer voor was.

Hieronder wordt in het kort beschreven welke problemen het prototype aan het licht heeft gebracht.

Routing

Zoals beschreven maakt ASP.Net MVC gebruik van routes. Al deze routes staan in één grote collectie. Het gevolg hiervan was dat de route die gebruikt werd voor het prototype API de routes van de website ontregelde. Dit komt omdat in de websites HTML links opgebouwd worden met een ASP.Net MVC Utility. Er was echter geen rekening mee gehouden dat de route van de API ook gekozen zou worden door deze Utility. Dit bleek echter een veel voorkomend probleem te zijn en door gebruik te maken van een Area zou dit probleem zich niet meer voordoen.

Serialisatie

Bij het serialiseren was er gekozen voor een C# techniek die Generics genoemd wordt. Het bleek echter dat deze Generics niet om konden gaan met de manier waarop in de API bepaald werd welk type object er geserialiseerd werd. Om deze reden is Generics ook niet toegepast, maar is er gebruik gemaakt van het Type Object, dit is een C# Object met informatie over het type van het betreffende object.

Dependencies

Bij het bepalen van het ontwerp van de Controllers en de bijbehorende acties is er gebruik gemaakt van de Usecases, die op hun beurt weer gebaseerd waren op hoe de functionaliteiten in de oude ASP website waren ondergebracht. Het bleek echter al snel dat Controllers zeer veel dependencies naar verschillende services zouden krijgen. Aangezien optimalisatie buiten de scope van dit project viel is er niet veel aandacht aan besteed. De aanmaak van de service is geregeld bij de aanmaak van de Controller. Dit zal er voor zorgen dat mocht er in de toekomst besloten worden dat hier geoptimaliseerd kan worden, de code gemakkelijk aan te passen is.

Git

Een ander probleem waar tegen aangelopen is tijdens het prototypen is het verlies van data door het fout gebruiken van GIT. Er werd vanuit gegaan dat binnen GIT Commit dezelfde werking had als Commit binnen TortoiseSVN. Het bleek echter dat Commit in GIT de data niet naar de server

verstuurd, maar alleen opslaat in de lokale opslag. Om de data naar de server te krijgen dient het Push commando gebruikt te worden. Binnen de organisatie wordt gebruik gemaakt van GIT Extensions om de functionaliteiten van GIT te benutten, deze client werkt alleen niet altijd even vlekkelooos. Ergens tijdens het updaten van de broncode werd door een onverklaarbare persoonlijke of software fout een Rebase getriggered, waardoor alle locale Commits ongedaan werden. Normaliter zou dit geen probleem zijn, aangezien de gegevens op de server horen te staan. Dit was een harde les, maar hierna is deze fout niet meer gemaakt. Achteraf gezien is het maar goed dat het tijdens het prototypen gebeurde en niet tijdens de daadwerkelijke ontwikkelfase. Het beste was geweest als het helemaal niet was gebeurt, maar dit soort fouten kunnen optreden.

6.5 Testplan opstellen

Aan het einde van de Construction Fase is er een testplan opgesteld. In dit testplan stond opgenomen hoe Unit tests opgesteld moesten worden en wat er precies in Unit tests zou worden opgenomen.

Er is gekozen om niet de Controllers maar de Services te Unit testen. Dit omdat de Controllers geen logica meer bevatten maar dit allemaal in de services zit. Om de services succesvol te kunnen testen moet er gezorgd worden dat iedere dependency die een service heeft, dit wil zeggen verbindingen naar andere services of repositories, voorzien moeten worden van een FakeImplementatie.

De Unit tests worden volgens de TDD methodiek opgesteld voordat er aan implementatie begonnen kan worden. Bij het invullen van de tests moet er voldoende gekeken worden of de functie bedoeld gefunctioneerd heeft. Dit betekent dat er gekeken moet worden of de functie het gewenste resultaat geeft, eventueel of het aantal correct is en of het resultaat van het gewenste type is.

6.6 Planning

In de Inception fase is er besloten om de planning van de uit te voeren activiteiten in de Construction fase bij te stellen aan het einde van de Elaboration fase. Op deze manier kon er goed gekeken worden welke fasering het beste resultaat zou hebben. Na het afronden van alle ontwerpen is er gekeken naar twee mogelijke faseringen.

De eerste fasering was gebaseerd op de technische aspecten. Hierbij zouden er twee fases zijn:

- Construction Fase 1: een fase voor de “Model” laag
- Construction Fase 2: een fase voor de “Controller” laag.

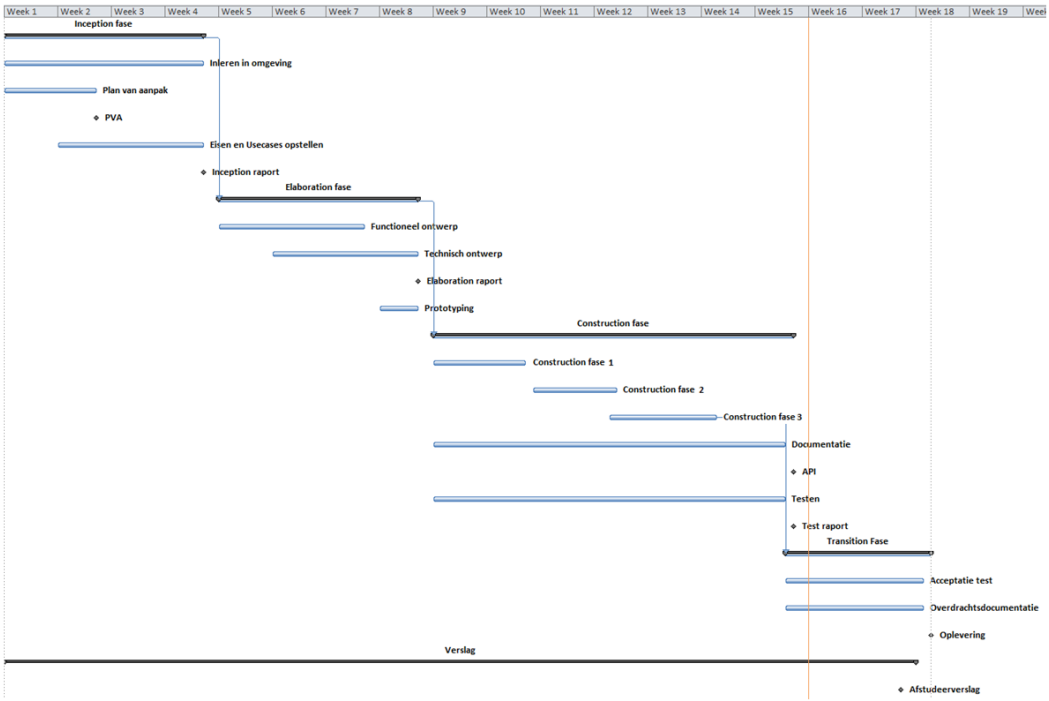
Op deze manier zou er een duidelijk onderscheid zijn tussen de verantwoordelijke lagen per fase.

De tweede fasering die overwogen was meer gebaseerd op de functionele aspecten van het eindproduct. Hierbij werd er gedacht om drie keer de Construction fase te doorlopen en per fase een los onderdeel af te ronden. Deze fases zouden zijn:

- Construction Fase 1: De ondersteuning voor de Flash Widgets (Legacy API’s)
- Construction Fase 2: De overige API functionaliteiten.
- Construction Fase 3: De OAuth autorisatie.

Uiteindelijk is er gekozen voor de tweede variant. Dit omdat dit voor het testen van de losse onderdelen beter leek en omdat er vanuit de opdrachtgever verteld werd dat bestaande Flash applicaties aangepast moesten worden door een derde partij om gebruik te maken van de API. Deze reden is doorslaggevend geweest bij het besluit om voor de planning met drie fases te kiezen.

Hieronder een screenshot van de planning met de nieuwe fasering.



Figuur 14 - Bijgestelde planning

7. Construction fase: Het bouwen van de API

In dit hoofdstuk worden de volgende Construction fases besproken:

- Construction Fase 1: De ondersteuning voor de Flash Widgets (Legacy API's)
- Construction Fase 2: De overige API functionaliteiten.
- Construction Fase 3: De OAuth autorisatie.

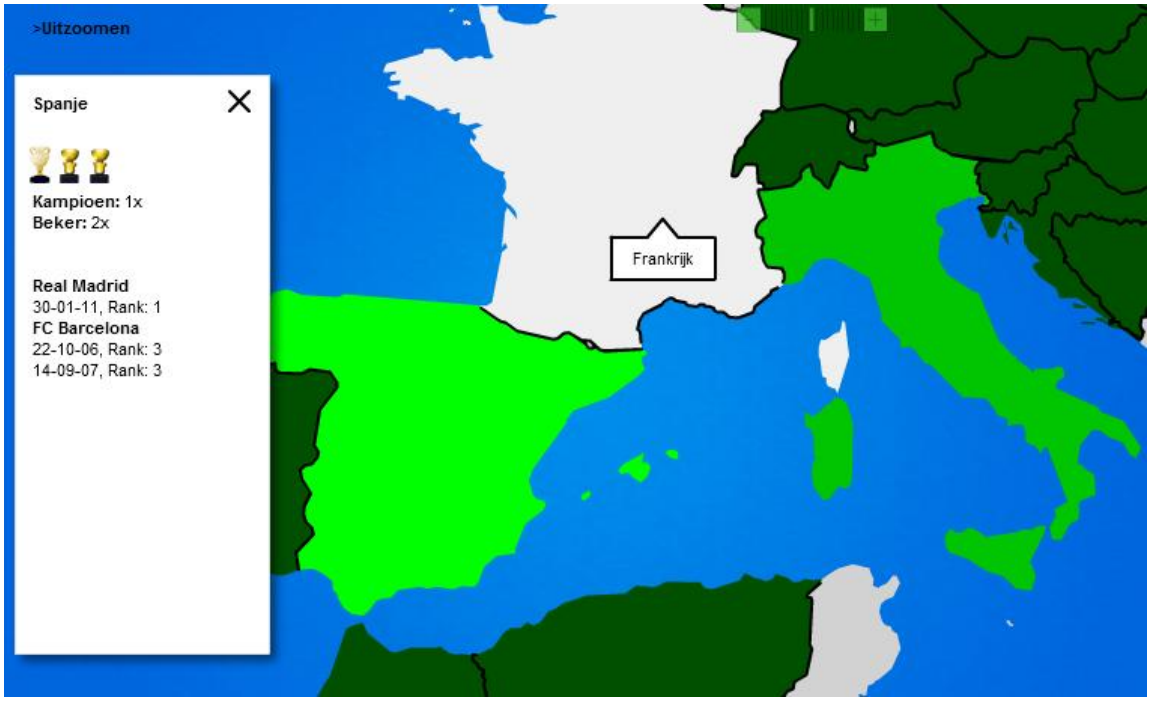
7.1 Fase 1: Ondersteuning voor de Flash Widgets

Tijdens deze fase is er gewerkt aan het maken van de API's voor de Flash widgets. Deze API's bestaan uit de Wereldkaart, de Profiel Widget, de RSS weergaves en de mogelijkheid om widgets aan managers te koppelen.

Als eerste is er tijdens deze fase begonnen met het opstellen van de unit tests volgens de in het Testplan opgestelde methodiek. Dit betekende dat voordat er aan het implementeren begonnen kon worden eerst lege classes gegenereerd moesten worden en de testvoorwaarden opgesteld moesten worden.

De Wereldkaart

Zoals eerder beschreven maakt deze applicatie gebruik van een externe plug-in en was het formaat van de XML niet vatbaar voor wijzigingen.



Figuur 15 - De Wereldkaart

Bij het implementeren van de WereldKaart API bleek dat de oude code meer dan 250 (!) database queries nodig had om alle data te verzamelen. Dit was in eerste instantie zo ook in de API opgenomen. Ook al vielen optimalisaties buiten de scope van dit blok, was hetzelfde resultaat ook te bereiken met twee database queries.

In de eerste situatie werden alle landen opgehaald, en per land een query gedaan of de speler daar een geschiedenis had. In de nieuwe versie worden er slechts twee aanvragen aan de repository gedaan, namelijk om alle landen op te halen en om de complete spelers geschiedenis op te vragen. Deze twee collecties kunnen dan met een techniek die LINQ heet aan elkaar gekoppeld worden.

```
01. var countryHistoryCollection = historyCollection.Where(x => x.CountryNr ==
country.CountryNr)
```

Bovenstaand stukje code is verantwoordelijk voor het zoeken van de juiste historie bij een land, maar aangezien dit niet meer met queries op de database gebeurt is de API vele malen sneller geworden en is de database belasting die deze API aanroep met zich meebrengt drastisch verlaagd.

Profiel widget

Deze flash applicatie bevat verschillende pagina's met informatie over een managers profiel. In de oude omgeving waren dit een aantal ASP pagina's die XML genereerde. In de nieuwe oplossing is er gekozen om een controller te maken die deze XML kan genereren.

Figuur 16 - Flash Applicatie

Zoals tijdens de ontwerp fase besproken zijn er een aantal wijzigingen gemaakt in de XML output om er voor te zorgen dat het mogelijk bleef om objecten te serialiseren en er voor te zorgen dat waardes logische namen hadden.

Koppeling tussen Widget en Manager

Om dit tot stand te brengen moest er eerst een Repository opgezet worden die bij de data in de database zou kunnen. Het opzetten van deze Repository zelf gaf geen problemen, het moeilijk was het verbinden met de goede database. Het feit dat er in een Nederlandse en Engelse test omgeving gewerkt wordt leverde hier een probleem op, aangezien de test manager die ik gebruikte zich binnen

het Engelse domein bevond maar de database die ik benaderde voor de koppeling bevond zich binnen het Nederlandse domein. Dit betekende dat data die ik wilde opvragen niet gevonden kon worden en het duurde een tijd voordat dit probleem duidelijk werd. Het voordeel was dat bij het opzetten van de overige repositories dit probleem zich niet meer voordeed.

ApiBind

Om er voor te zorgen dat de mogelijkheid er was om data naar XML / JSON om te zetten is er tijdens deze iteratie van de Construction fase gebruik gemaakt van de tijdens het prototypen opgestelde [ApiBind] filter. Tijdens deze iteratie is er voor gekozen om snel resultaat te behalen en is er gekozen om geen validatie of beslispunten in de ApiBind toe te voegen. De filter heeft een XML en een JSON Serialisatie mogelijkheid en kan alleen XML of JSON terug geven aan de hand van een in de code opgeslagen waarde. De keuze hiervoor is genomen omdat de bouw van dit onderdeel eigenlijk pas in de tweede Construction iteratie echt aan bod komt en dat de oude API voldoende kan functioneren met een kale versie van de filter. In de volgende Construction iteratie wordt er gekeken naar het afhandelen van verschillende formaten en het valideren van de aanroep.

Evaluatie TDD

Er is besloten om na deze fase deze manier van development te laten varen en gebruik te maken van normaal unit testen. Dit om de volgende redenen:

- Programmeren richt zich op het succesvol afronden van een test en dan pas een goed product opleveren.
- Er gaat te veel tijd inzitten.
  Het komt er op neer dat je alles bijna twee keer schrijft. Eerst programmeer je een versie die de test succesvol aflegt, vervolgens ga je de code helemaal herschrijven om te zorgen dat de code productie waardig is. Dit leed er in een aantal gevallen zelfs toe dat na het refactoren van de code de unit test weer faalde.

Het van te voren opstellen van de Unit tests zal nog wel gehanteerd worden, aangezien de tests in ieder geval opgesteld moeten worden. Het invullen van de Unit tests zal echter later gebeuren wanneer de functionaliteiten geïmplementeerd zijn.

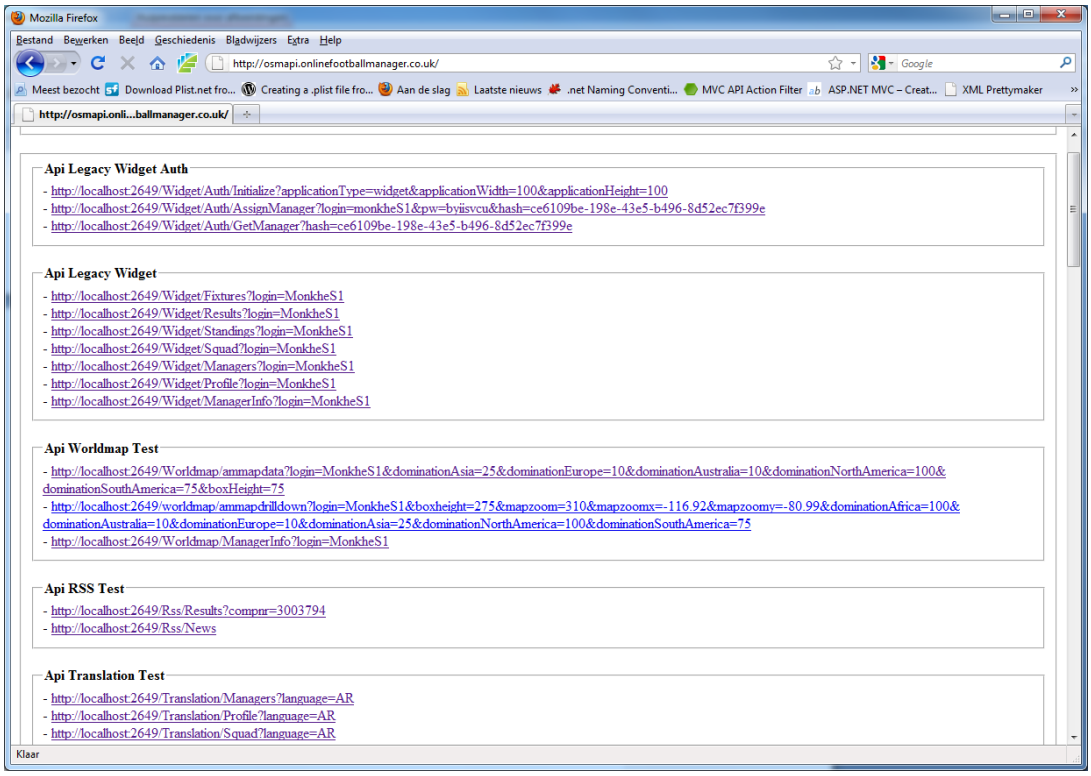
Result	Test Name	Project	Error Message	Result Storage
Passed	GetLegacyResultsTest	OSM.Data.Test		
Passed	GetLegacySquadTest	OSM.Data.Test		
Passed	GetLegacyStandingsTest	OSM.Data.Test		
Failed	GetManagerInfoTest	OSM.Data.Test	Test method OSM.Data.Test.Services.Api.LegacyManagerServiceTest.GetManagerInfoTest threw exception: ...	
Failed	GetManagerStandingsTest	OSM.Data.Test	Test method OSM.Data.Test.Services.Api.LegacyManagerServiceTest.GetManagerStandingsTest threw exception: ...	
Passed	GetManagerTest	OSM.Data.Test		
Passed	GetNewsUpdatesRssTest	OSM.Data.Test		
Passed	GetNewsUpdatesTest	OSM.Data.Test		

Figuur 17 - Resultaten Unit tests

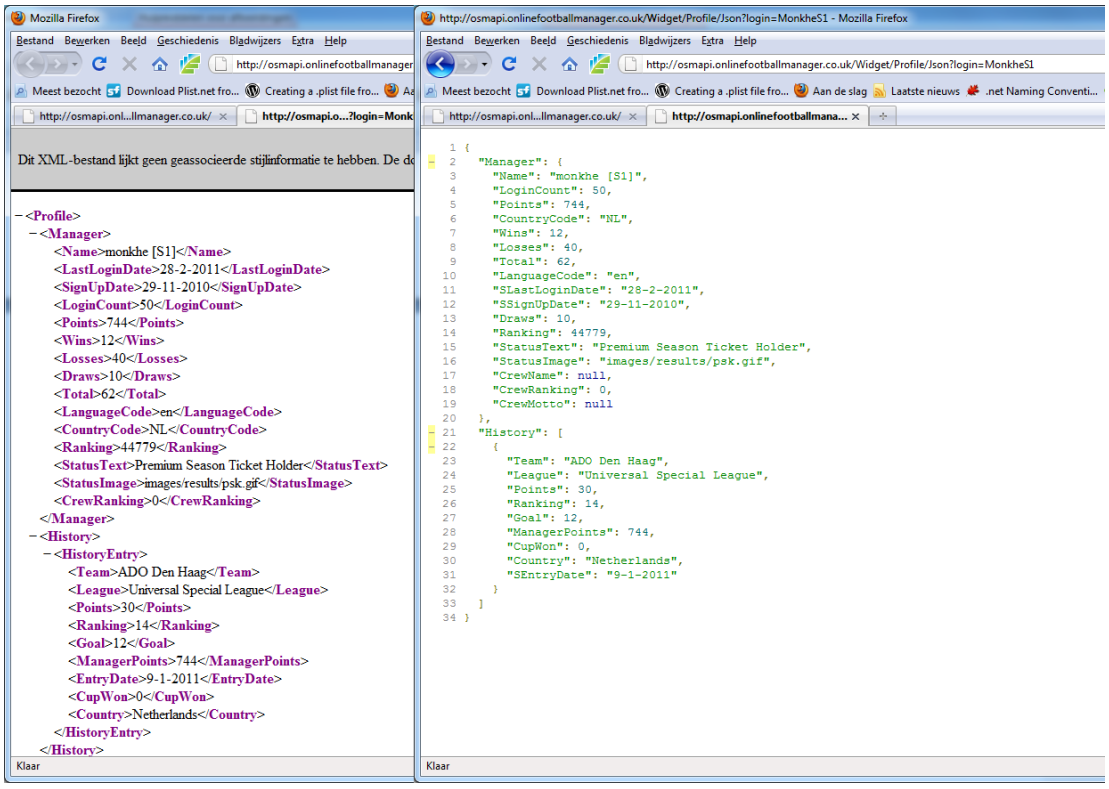
Bovenstaand plaatje laat de resultaten van de Unit tests zien die gebruikt worden bij de Widget API's. Hier valt duidelijk te zien dat 2 tests om een bepaalde reden niet succesvol zijn. In dit voorbeeld waren dit tests die functies aanriepen die nog niet geïmplementeerd waren, maar later wanneer alles wel af is en tests falen, kan dit zijn omdat afhankelijkheden van deze tests gewijzigd zijn. Op deze manier kun je de gevolgen van je wijzigingen makkelijk opsporen.

Naast de Unit tests wordt er gebruik gemaakt van een Mockup test. Hiervoor is een grote pagina gemaakt waarop alle API functionaliteiten staan en waarbij de uitvoer handmatig getest en bekeken kan worden. Hieronder een weergave van deze methode. Voor iedere functionaliteit die er gemaakt

is, is deze manier van opvragen van data mogelijk, dit is geld voor zowel het ophalen als posten van data. Deze twee methodes gecombineerd zorgen ervoor dat de functionaliteiten grondig getest worden wat de kwaliteit van het eindproduct hoog houd.



Figuur 18 - Mockup Pagina



Figuur 19 - Mockup Uitvoer van een API functionaliteit

7.2 Fase 2: De overige API functionaliteiten

Tijdens deze fase is er gewerkt aan het uitbreiden van de ApiBind (toevoegen van validatie en instellingen) en het implementeren van de controller acties die het mogelijk maken de functionaliteiten die in de website beschikbaar zijn ook beschikbaar te maken in de api.

Als eerste is er begonnen met het uitbreiden van de ApiBind. Het eerste doel was om zo vroeg mogelijk te bepalen of een API aanroep daadwerkelijk uitgevoerd kan/mag worden. Op het moment dat duidelijk wordt dat dit niet mag/kan moet een error bericht terug gegeven worden en kan de API aanroep afgebroken worden. De volgende controles zijn ingebouwd:

- Het valideren van het inputformaat.
- Het valideren van het outputformaat.
- Het valideren of alle verplichte waardes zijn ingevuld.

Paramater validatie

In eerste instantie was het plan deze validatie in de controllers zelf te doen, maar dit zou ertoe leiden dat iedere actie een reeks if en else constructies zou krijgen om te kijken of waardes ingevuld waren of niet. Om dit te voorkomen is er gekeken naar een techniek die in C# Reflection heet. Met deze techniek is het mogelijk om van een methode op te vragen welke parameters deze heeft en of deze parameters een standaard waarde hebben (Indien dit zo is, is de parameter niet verplicht). Op deze manier kan dus simpel per aanroep in de ApiBind gevalideerd worden of alle verplichte waardes zijn ingevuld. Deze methode is beter dan de handmatige validatie omdat:

- Nieuwe methodes automatisch parameter validatie krijgen
- Methodes waarvan de parameters aangepast worden, de validatie ook automatisch wordt bijgewerkt.
- De controller acties blijven schoon en bevatten alleen afhandeling van de actie en niet de controle ervan.

Wat kunnen we nu?

Het is nu mogelijk een aanroep te doen en dezelfde data in verschillende formaten op te vragen, daarbij wordt er gekeken of de aanroep alle verplichte parameters invult. Als we bijvoorbeeld de formatie opvragen van een team moeten we een competitie en team nummer meesturen in de aanvraag en geven we op in welk formaat we informatie terug willen zien. Dit leidt tot de volgende aanroepen met de bijbehorende uitvoer.

- http://osmapi.onlinefootballmanager.co.uk/team/formation/xml?CompNr=301&teamnr=3

```
<ApiResult>
        <MessageCode>API_SUCCES</MessageCode>
        <Result>
                <Nr>4</Nr>
                <Name>451</Name>
                <Detail>B</Detail>
                <Description>4-2-3-1</Description>
        </Result>
</ApiResult>
```

- http://osmapi.onlinefootballmanager.co.uk/team/formation/json?CompNr=301&teamnr=3

```
{
  "MessageCode": "API_SUCCES",
  "Result": {
    "Nr": 4,
    "Name": 451,
    "Detail": "B",
    "Description": "4-2-3-1"
  }
}
```

Op deze manier is dus voldaan aan de eis dat het mogelijk moest zijn meerdere formaten te ondersteunen en het flexibel en uitbreidbaar te houden.

Tijdens deze fase werd er door een andere werknemer gewerkt aan updates voor de IPhone. Er werd toen gevraagd om te kijken of het mogelijk was in de nieuwe API om het formaat PList te ondersteunen. Dit is het native format wat door de IPhone ondersteund wordt. Toen bleek dat er een bestaande C# library was die naar PList kon serialiseren is deze aan de lijst met Serialisatie methodes toegevoegd en bleek dat zonder enige moeite een derde Serialisatie methode kon worden toegevoegd (**Uitbreidbaarheid**).

- http://osmapi.onlinefootballmanager.co.uk/team/formation/plist?CompNr=301&teamnr=3

```
<plist version="1.0">
        <dict>
                <key>Nr</key>
                <integer>4</integer>
                <key>Name</key>
                <integer>451</integer>
                <key>Detail</key>
                <string>B</string>
                <key>Description</key>
                <string>4-2-3-1</string>
        </dict>
</plist>
```

Implementatie Controllers

Nadat het valideren van een aanroep en het vast stellen van hoe data geserialiseerd en gedeserialiseerd moest worden is er gewerkt aan het implementeren van de daadwerkelijk controllers. Bij deze acties is er een onderscheid te maken tussen controller acties die data ophalen(±75%) en controller acties die data opslaan(± 25%).

Er is gekozen om eerst te werken aan functionaliteiten die data ophaalde te implementeren. Hiervoor is gekozen omdat het opslaan van data het makkelijkst gecontroleerd kan worden wanneer de data ook opgevraagd kan worden. Inloggen op de website en hier kijken had ook gekund, maar aangezien zowel het ophalen als opslaan goed gecontroleerd moest worden, kon dat op de manier goed opgenomen worden in het proces.

Bij het implementeren van functionaliteiten is het globale lijstje (§ 6.1.5) met functionaliteiten aangehouden.

Het implementeren van de meeste aanroepen was vrij simpel, aangezien dit het aanspreken van een service was om data op te halen aan de hand van parameters waarvan al eerder besloten was dat ze valide waren. Een van de ingewikkeldste onderdelen was het implementeren van de staff. Bij aanroepen hiervan moesten er in vergelijking met andere methodes veel controles gedaan worden, is een staff lid in dienst, is dit staff lid al aan het werk?

Naarmate meer acties geïmplementeerd werden kwam er naar voren dat toch nog veel acties functionaliteiten bevatten die gedeeld waren (vanuit oogpunt van de gebruiker genomen):

- Heeft de gebruiker een team?
- Heeft de gebruiker staff lid X in dienst?
- Heeft de gebruiker een seizoenskaart?

Zoals eerder beschreven is ActionFiltering een mooi techniek om dit soort functionaliteiten naar een centrale locatie te halen om zo voor de aanroep van een methode al deze validatie uit te kunnen voeren. Om bovenstaande validatie te kunnen uitvoeren zijn de volgende ActionFilters extra gemaakt:

- ApiHasTeam ,validatie van het team
- ApiStaffMemberEployed(x), kijkt of een bepaald staff lid in dienst is waarbij x = (spy, spion, commercieel manager of doctor 1 -4).
- ApiHasTicket, validatie of een gebruiker een seizoenskaart heeft.

Op deze methode kan er per controller actie simpel ingesteld worden wanneer een seizoenskaart verplicht is of wanneer het hebben van een team verplicht is.

Nadat de GET functionaliteiten klaar waren is er gewerkt aan de POST aanroepen. Hierbij bleek al snel dat er data op 2 manieren gepost kan worden. De eerste manier is het versturen van de waardes in de url:

http://osmapi.onlinefootballmanager.co.uk/team/formation/json?CompNr=301&teamnr=3&formation=451a

de tweede manier is het versturen van de data als echte raw post data. Beide varianten zijn geïmplementeerd. Dit omdat de eerste variant het makkelijkst te gebruiken is voor applicaties, maar daarbij ook minder veilig. Deze methode is namelijk gevoelig voor Phishing. Dit maakt voor waardes zoals de formatie is 451A niet uit, maar wanneer het om wachtwoorden gaat is het wel wenselijk dat deze via raw Post data verstuurd kunnen worden.

Batching

Tijdens het implementeren van de functionaliteiten ontstonden er tussen de opdrachtgever en de opdrachtnemer gespreken over het maken van gecombineerde aanroepen om er voor te zorgen dat een Applicatie minder aanroepen zou hoeven te doen. Het zou handig zijn als een applicatie in één aanroep een team en de competitiestand van dat team zou kunnen opvragen.

Tijdens het bepalen van welke samengestelde resultaten er gemaakt zouden konden maken bleek dat dit eigenlijk oneindig veel mogelijkheden zou opleveren, want het hoeft niet bij twee aanroepen te blijven maar kunnen er ook drie of vier worden.

De API die door Google aangeboden wordt om hun data te benaderen maakt gebruik van een Batching Protocol. Omdat de tijd het binnen deze iteratie toeliet is er gekeken naar hoe dit geïmplementeerd kon worden en hoe dit toegepast kon worden. Uiteindelijk is het er op neer gekomen dat er een API call gedaan kan worden met een XML document met een groepering API Calls. Op deze manier is de applicatie die de API gebruikt vrij in het aantal aanroepen.

```
<?xml version="1.0" encoding="utf-16"?>
<BatchFeed xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Batch>
  <BatchEntry>
    <Url>http://osmapi.onlinefootballmanager.co.uk/Widget/profile?login=gebruiker</Url>
    <Method>GET</Method>
   </BatchEntry>
  <BatchEntry>
    <Url>http://osmapi.onlinefootballmanager.co.uk/Translation/Profile?language=EN</Url>
    <Method>GET</Method>
  </BatchEntry>
</Batch>
</BatchFeed>
```

Bovenstaand XML document is een voorbeeld van de beschreven Batching. Er worden twee aanroepen gegroepeerd, en deze worden vervolgens in een keer uitgevoerd. Dit zorgt ervoor dat degene die de API gebruikt vrij is in het samenstellen van de data naar zijn specificaties op de momenten dat hij het wilt. Voor bijvoorbeeld mobiele applicaties kan dit een groot voordeel zijn omdat nu in één batched aanroep de gehele updates van een team, de competitie en de resultaten opgehaald kunnen worden.

Testen

Bij het opzetten van de tests is op dezelfde manier gebeurd als tijdens de eerste iteratie van de Construction fase. Het invullen en uitvoeren van de Unit tests is gebeurde pas nadat een methode compleet geïmplementeerd was.

De nieuwe functionaliteiten werden steeds direct toegevoegd aan de test pagina om zo gemakkelijk de output van functies te kunnen opvragen en testen of het in alle formaten goed werkte.

Het opstellen van unit tests voor de ApiBind filter leverde enkele problemen op omdat deze gebruik maakte van C# http objecten. Om goed te unit testen moesten deze ook fake implementaties krijgen. Dit bleek moeilijker dan gedacht, daarom is er naar RhinoMocks en MockIt gekeken. Deze liepen echter tegen hetzelfde probleem aan als ik, namelijk dat het http object in kwestie een private constructor heeft en dus niet op deze manier een fake implementatie kon krijgen. Dit is uiteindelijk opgelost met een overerving waardoor het faken en uiteindelijk unit testen wel kon.

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML",
"|DataDirectory|\\TestData.xml", "OnActionExcecuting", DataAccessMethod.Sequential),
TestMethod]
public void OnActionExecutingTest()
{
    var apiFilter = new ApiBind();

    var aec = new ActionExecutingContext
    {
        HttpContext = new FakeHttpContextBase(TestContext),
        RequestContext = new FakeRequestContext(TestContext)
    };

    apiFilter.OnActionExecuting(aec);

    var output, input;

    if (TestContext.DataRow["format"].ToString().Equals("xml"))
        output = Library.Api.Api.SerializationFormat.Xml;
    if (TestContext.DataRow["format"].ToString().Equals("json"))
        output = Library.Api.Api.SerializationFormat.Json;

    if (TestContext.DataRow["contenttype"].ToString().Equals("text/xml") ||
TestContext.DataRow["contenttype"].ToString().Equals("application/xml"))
        input = Library.Api.Api.SerializationFormat.Xml;
    if (TestContext.DataRow["contenttype"].ToString().Equals("text/json") ||
TestContext.DataRow["contenttype"].ToString().Equals("application/json"))
        input = Library.Api.Api.SerializationFormat.Json;

    Assert.AreEqual(output,
PrivateMember.GetValue<Library.Api.Api.SerializationFormat>(apiFilter,
"_outputSerializationFormat"));
    Assert.AreEqual(input,
PrivateMember.GetValue<Library.Api.Api.SerializationFormat>(apiFilter,
"_inputDeserializationFormat"));
}
```

Bovenstaande code is een van de ingewikkeldste Unit tests. Deze maakt gebruik van een Data Injectie techniek vanuit XML. In dit geval staat in XML de mogelijke manieren van input en output opgeslagen. Alle mogelijkheden worden vervolgens in deze test uitgevoerd. Dit zorgt ervoor dat er met één test alle mogelijkheden en combinaties getest kan worden.

Resultaten

Uiteindelijk zijn alle testen succesvol, maar een aantal waren dit in eerste instantie niet. Hieronder wordt kort besproken wat de problemen waren en hoe deze opgelost zijn:

- PList
  Probleem: Output leeg.
  Oplossing: De gebruikte library voor PList verwachte dat een class gemarkeerd is met het Serializable attribuut, anders wordt het genegeerd.

- Trainingskamp
  Probleem: Deze test faalde de eerste testruns niet, maar later kon er ineens niet op trainingskamp gegaan worden (NO_OPPONENT error).
  Oplossing: Het probleem zat hem dat een trainingskamp gedaan werd voor de huidige speelronde en niet de volgende. Dit probleem deed zich voor bij een rustdag.

- Competitie afgelopen
  Probleem: Meerdere API functionaliteiten werkte niet meer toen van het testaccount de competitie afgelopen was.
  Oplossing: Consequent controleren of een gebruiker een team heeft.

7.3 Fase 3: De OAuth autorisatie

Tijdens deze fase is het OAuth autorisatie protocol geïmplementeerd. Deze fase zal zich vooral richten op het verstrekken en valideren van zogenaamde OAuth Tokens en het omzetten van deze tokens naar een "ApiSessie". Daarbij is er tijdens deze fase met een extern bedrijf gewerkt om de Flash applicaties te laten werken met de nieuwe API.

7.3.1 OAuth Implementeren

Voordat OAuth compleet geïmplementeerd was zijn een aantal stappen doorlopen.

Handtekening Genereren

Als eerste is er gewerkt aan het implementeren van het mechanisme wat de handtekening voor een aanvraag kan genereren. Hierbij zijn drie encryptie mogelijk:

- HMAC-SHA1
- RSA-SHA1
- Plaintext

Er is gekozen voor HMAC-SHA1 omdat deze encryptie standaard in C# zit en veiliger is dan Plaintext encryptie. RSA-SHA1 bevindt zich niet in C# standaard en het implementeren hiervan zou teveel tijd kosten.

In de Elaboration fase was er al onderzocht hoe handtekeningen in OAuth gegenereerd worden en welke waardes hiervoor nodig waren. Om te valideren of de gegenereerde handtekeningen goed waren is gebruik gemaakt van http://term.ie/oauth/example/?sig_method=HMAC-SHA1 op deze pagina kunnen handtekeningen gegenereerd worden. Door middel van *trial and error* is de methode van het genereren van de handtekeningen geperfectioneerd.

Controller

Nadat er handtekeningen gegenereerd konden worden was de volgende logische stap een toegangspunt voor applicaties maken binnen de API waar zij RequestTokens en AccesTokens kunnen aanvragen. Het ingewikkelde van dit proces is, is dat bij het aanvragen van een AccesToken gekeken moet worden of het bijbehorende RequestToken valide is en aan de hand daarvan een AccesToken generen en deze vervolgens aan een manager koppelen.

Het koppelen van managers kan op twee methodes:

- Directe POST login. Dit is wanneer gebruikers direct ingelogd moeten worden en validatie van de applicatie niet nodig is (Handig bij intern ontwikkelde pagina's).
- Redirect naar login. Dit is nodig wanneer applicaties van derden toegang tot een gebruikers account willen. De login vindt dan plaats op OSM hosts en gegevens zijn daarmee veilig.

ApiSessie

Bij websites is het gewoonlijk wanneer je inlogt er een sessie of cookie met gegevens wordt bijgehouden. Binnen een API is dit niet mogelijk, omdat nadat de data terug gestuurd wordt een gebruiker inweze weer uitlogt. In de sessie voor de website bevinden zich ±30 waardes, het hergebruiken hiervan bleek geen optie omdat het opbouwen van dit object per API aanroep te zwaar zou worden.

Eerder is beschreven hoe een ActionFilter gebruikt kon worden om te valideren of parameters voor een methode waren ingevuld. Het bleek dat een ActionFilter ook nieuwe parameters kon toevoegen. Dit betekend dat het mogelijk is om een OAuth ActionFilter te maken die een token valideert en omzet naar een “api sessie”. Door paramaters toe te voegen wordt het ineens mogelijk om in controllers informatie over de gebruiker op te vragen. Deze methode is op het moment van het schrijven van dit document nog niet geoptimaliseerd. Iedere aanroep worden de waardes van een token opnieuw uit de database gehaald. Het doel is om deze gegevens uiteindelijk voor een uur te cachen om zo de databaseload te verlagen.

7.3.2 Samenwerking NetBasics

Tijdens deze fase is er naast de OAuth implementatie ook gewerkt aan de transitie van de API voor de Flash applicaties. Deze Flash applicaties zijn ontworpen door het bedrijf NetBasics. Er is een afspraak met NetBasics gemaakt om door te spreken welke wijzigingen er in het XML formaat zijn gemaakt en wat er eventueel nog aangepast moet worden. Daarbij zou er een begin gemaakt worden aan de wijzigingen in de Flash applicaties.

Er is gezorgd dat voor deze afspraak de API vanaf de volgende locaties bereikbaar is:

- http://osmapi.onlinesoccermanager.nl
- http://osmapi.onlinefootballmanager.co.uk

In § 6.1.3 is er besloten op het API project in een Area te stoppen, om zo wel in het website project te blijven en gebruik te kunnen maken van eventuele gedeelde functionaliteiten. Het bleek echter dat een losse Area niet handig was omdat:

- De API niet los live gezet worden, op dit moment is de nieuwe website nog niet open voor publiek, toen de API live gezet werd, viel direct op dat de hele website openbaar was. Dit zou uiteindelijk niet een probleem zijn, maar voor dit moment wel.
- Het tweede punt is meer een kwestie van smaak dan een echt probleem, maar doordat de C# routes geen subdomeinen ondersteunen kregen de Urls voor de API dus twee keer een verwijzing mee dat het de API was, namelijk:
  http://osmapi.onlinefootballmanager.co.uk/api/

Er is toen gekeken naar hoeveel afhankelijkheden de API nog had met de website en dit bleek slechts op één functie aan te komen. Deze functie bepaald aan de hand van je domein de interne afhandeling welke databases er gekozen moeten worden. Hierdoor is er samen met de opdrachtgever besloten om van de API een los project te maken, om zo de bovenstaande problemen op te lossen.

Tijdens de eerste afspraak is er gewerkt aan de WereldKaart Flash applicatie. Het omzetten hiervan bleek slechts enkele uren werk te zijn. Binnen zowel de API als de Flash applicatie zijn wijzigingen gemaakt die zich vrijwel uitsluitend spitste op namen van velden. Omdat de API extern bereikbaar was is er besloten geen tweede afspraak te maken maar alle verdere wijzigingen via de e-mail te regelen.

Via de e-mail kwamen slechts twee aanvragen voor wijzigingen binnen. Het eerste was dat de API wanneer een team opgevraagd wordt een lijst met spelers teruggeeft. Het verzoek was om deze lijst met spelers op te delen in 4 lijsten, de aanvallers, de middenvelders, de verdedigers en de keepers. Het tweede verzoek was dat een set met vertalingen miste waardoor een deel van de Flash applicatie er blanco uit zag. Na deze wijzigingen is er een nieuwe build live gezet en kwam vanuit NetBasics het bericht dat alles werkte.

Buiten het feit dat de samenwerking met een externe partij een leerzaam proces was, is het de kwaliteit van het product ook ten goede gekomen. Op deze manier werd de werking van de API tijdens de ontwikkel fase door andere personen dan alleen de ontwikkelaar zelf getest.

7.4 Handleiding schrijven.

Het opstellen van een goede handleiding was een van de eisen van de opdrachtgever om er zo voor te zorgen dat het gebruik van de API voor mensen gemakkelijk is. Om deze reden is er in de handleiding niet alleen beschreven wat een functie doet maar ook in het kort wat voor functies een controller aanbied om het zo voor gebruikers makkelijk te maken informatie in de handleiding te vinden.

Per functie is er gekozen om de volgende zaken te bespreken:

- Algemene omschrijving.
- De URL voor de aanroep
- De Methode voor de aanroep (GET / POST).
- De Parameters die nodig zijn en welke waardes hierin toegestaan zijn
- Of deze functie autorisatie nodig heeft of niet.

8. Transition fase: Het in productie nemen van de API

Tijdens de laatste fase van het project is er gewerkt aan de in productie name van de API. Tevens stond er voor deze fase de acceptatie en gebruikerstest gepland. Tijdens het schrijven van dit verslag is hier echter nog niet aan toe gekomen en zal dit later uitgevoerd worden.

Tijdens de laatste Constructie fase is er al een groot deel van deze fase begonnen. Er is toen al gekeken hoe de API in productie genomen kon worden. Hieruit kwam toen ook naar voren dat een los project een betere oplossing was.

In het kader van de Transition Fase zal er een installatie handleiding geschreven worden die het voor toekomstige nieuwe builds van de API mogelijk maakt snel in productie genomen te worden.

Op dit moment draait de API nog lokaal en niet op een daadwerkelijke webserver. Dit wordt tegelijk met het live zetten van de nieuwe website omgezet.

Tenslotte is er tijdens deze fase wel een begin gemaakt aan het opstellen van de acceptatie test. Deze acceptatie test is gericht aan de opdrachtgever en bevat een simpele checklist met de functionele en technische eisen die opgesteld zijn tijdens de Inception Fase. Hiermee kan er geëvalueerd worden of de API aan de eisen van de opdrachtgever voldoet.

9. Evaluatie

Hieronder zal een evaluatie gedaan worden van het proces wat doorlopen is bij het afstuderen en het uiteindelijk opgeleverde product. Bij het proces zal er per RUP fase beschreven worden hoe het verlopen is en wat er goed of fout ging en wat er voor de toekomst voor lessen geleerd zijn. Bij de product evaluatie zal het uiteindelijk product worden geëvalueerd en alle documentatie van RUP die daarbij is opgeleverd.

9.1 Product evaluatie

Aan het einde van de afstudeerperiode is de OnlineSoccerManager API opgeleverd conform de eisen van de opdrachtgever. Bij deze oplevering zaten een aantal ondersteunende documenten vanuit RUP met betrekking tot het proces, een testrapport en een handleiding. Daarbij wordt er op dit moment gewerkt aan een document waarin beschreven staat hoe het eindproduct in productie genomen kan worden.

Vanuit GameBasics is naar voren gekomen dat zij tevreden zijn over de werking van het product en tevreden zijn over het feit dat de in productie name van het onderdeel voor de Flash applicaties zo vloeiend is verlopen.

Zelf ben ik tevreden over het eindproduct, van te voren waren er een aantal eisen gesteld aan het product waaraan ik duidelijk heb voldaan. Het product is zo opgezet dat het gemakkelijk uit te breiden is, dit is al gedemonstreerd door een nieuwe Serialisatie methode toe te voegen tijdens de ontwikkeling. Daarbij is het eenvoudig nieuwe functionaliteiten op te zetten en die te koppelen aan de API.

Een van de punten waar ik minder tevreden over ben is dat sommige code voor de legacy API's niet aan mijn persoonlijke standaarden kon voldoen omdat hier teveel gewerkt moest worden naar een bepaalde output. In de nieuwe API's was ik hier veel vrijer in en kon ik de uiteindelijke oplossing ook veel netter opzetten.

Het uiteindelijke product maakt het mogelijk om voor externe applicaties de functionaliteiten van OnlineSoccerManager te benaderen. Dit is al gedemonstreerd met de Flash applicaties, maar dit is slechts het ophalen van data. De API ondersteunt ook het wijzigen van data van gebruikers. De API draait op de productieomgeving nu en zal in de toekomst gebruikt gaan worden om nieuwe Android, IPhone en Windows Mobile applicaties te ontwikkelen. Daarbij zal de API ingezet worden om OnlineSoccerManager te koppelen aan bijv. Facebook.

9.2 Proces evaluatie

Tijdens dit afstudeerproject is er gebruik gemaakt van de RUP methodiek. Over het verloop hiervan ben ik zeer tevreden. Het goed gebruiken van RUP genereert alleen veel werk, aangezien iedere fase opnieuw wordt gekeken naar dingen die in de vorige fase al gedaan zijn. Dit zorgde er wel voor dat de eisen tijdens het project bekeken bleven worden om eventueel aangepast te kunnen worden.

Tijdens de Inception fase heb ik me vooral bezig gehouden met het instuderen in de ASP.Net MVC omgeving en de huidige OnlineSoccerManager. Het goed uitvoeren van deze fase bleek later in het project zeer handig, omdat door het hele project voor mij goed duidelijk was wat er gemaakt moest worden en welke eisen de opdrachtgever aan het eindproduct stelde. Ook heeft het wekelijkse overleg met de opdrachtgever bijgedragen aan het snel duidelijk krijgen van alle eisen.

In de Elaboration fase is er aan de hand van de eisen en de opgestelde usecases gewerkt aan een ontwerp. Naast het ontwerpen is er gekozen, om volgens de RUP methodiek, om een prototype te maken waarin een aantal basis zaken zouden werken. Dit bleek achteraf een wijs besluit omdat op deze manier snel duidelijk werd welke dingen in het ontwerp goed werkte en welke problemen er konden optreden. Het besluit om in deze fase de planning bij te stellen is achteraf gezien een goed besluit geweest. Op deze manier had ik genoeg kennis en inzicht om goed te kunnen indelen welke zaken los opgeleverd konden worden en samen in een Construction fase geplaatst konden worden. Het enige negatieve aan deze fase is het verlies van data geweest dat door een persoonlijke fout met GIT voorkwam.

Over het algemeen ben ik tevreden over het verloop van de Construction fases. Achteraf gezien is het een slimme zet geweest om van TDD af te stappen omdat dit gewoon teveel tijd zou kosten. Over de werkzaamheden die uitgevoerd zijn tijdens de verschillende fases ben ik ook zeer tevreden. Het werken in een aparte branch binnen GIT gaf soms problemen, omdat er soms wijzigingen werden gemaakt in onderdelen waarvan mijn werk afhankelijk was. Dit heeft echter nooit tot grote problemen geleid. Daarbij heb ik veel geleerd van het samenwerken met NetBasics en ben ik blij dat de integratie van mijn product zo soepel verliep.

Tijdens de Transition fase is er gewerkt aan het in productie nemen van de API. Een groot deel van de API draaide al op de live omgeving, het in gebruik nemen van het hele product was dan ook geen probleem. Tijdens deze fase had er nog een acceptatie test plaats moeten vinden en had het product onderworpen moeten worden aan een test door andere. Door uitloop is hier echter op het moment van het schrijven van dit verslag nog niet aan begonnen. Dit staat gepland voor een later moment.

Positieve punten:

- Goede fasering.
- Fases liepen goed in elkaar over (alle kennis was steeds aanwezig).
- Door goed verloop enkele extra's kunnen implementeren.

Negatieve punten:

- Dataverlies
- RUP genereert veel documentatie

9.3 Planning

De planning die tijdens dit project gemaakt is, makte gebruik van de RUP fasering, onderstaande fases zijn daarbij doorlopen:

- Inception Fase
- Elaboration Fase
- Construction Fase
- Transition Fase

De tijd die per fase besteed is aan de werkzaamheden kwam in bijna iedere fase overeen met de planning. In de toekomst zal ik echter de oriëntatie die tijdens de Inception fase gepland was als losse tussenfase plannen om de fases duidelijker af te kunnen bakenen.

De methode waarbij tijdens de Elaboration fase de rest van de planning verder uitgewerkt werd werkte voor mij zeer goed. De tijd was al vast gelegd in de planning, maar het verder specificeren van de planning pas doen op het moment dat de kennis daarvoor goed aanwezig is was voor mij persoonlijk een prettige werkwijze.

Positieve punten:

- Goed aan planning gehouden.
- Genoeg tijd per fase ingepland.
- Planning verder kunnen specificeren aan de hand van opgedane kennis in het verloop van het project.

Negatieve punten:

- Oriëntatie en Inception niet gescheiden houden.

9.4 Beroepstaken

Voordat het afstuderen begon is er een aantal competenties opgesteld in het afstudeerplan waaraan tijdens deze stage voldaan zou worden. Hieronder staan deze nogmaals beschreven en bij welke werkzaamheden er aan voldaan is.

- 2.2 Ontwerpen, bouwen en **bevragen** van een database.
  Deze competentie richt zich vooral op het bevragen, aangezien het bouwen van de database alleen voor de OAuth database gold. Het bevragen van de database vond plaats in een ingewikkelde omgeving, bestaande uit meerdere databases verspreid over verschillende servers. Voor het benaderen van de databases is gebruik gemaakt van het Repository design pattern. De volgende Repositories zijn opgezet:
  - Nieuws en Updates
  - OAuth
  - Hashes
  - Profiel Widget

- 3.2 Ontwerpen systeemdeel.
  Tijdens dit project is een complex opzichzelfstaand systeemdeel ontworpen. De API is opgezet binnen de ASP.Net MVC en maakt gebruik van het Service Layer Pattern en het Repository Pattern. Daarbij is de implementatie van het OAuth protocol opgenomen in het ontwerp.

- 3.3 Bouwen Applicatie.
  Tijdens de verschillende Construction fases is het ontwerp van de API uitgewerkt.

- 3.5 Uitvoeren van en rapporteren over het testproces.
  Tijdens de Elaboration fase is er een testplan opgesteld waarin beschreven stond volgens welke methode er getest zou worden, en wat testcases zouden moeten doen. Tijdens de eerste Construction fase is er uitvoerig gebruik gemaakt van Test Driven Development. Het unit testen is in de andere Construction fases nog wel gedaan maar niet via de TDD methodiek.

9.5 Conclusie

Terugkijkend op het hele proces kan ik concluderen dat het een succesvol afstudeerproject is geweest. Het product wat opgeleverd is werkt goed en voldoet aan de eisen van de opdrachtgever. Daarbij heb ik leren werken met verschillende technieken waar ik nog geen ervaring had. Hierbij wordt niet alleen op ASP.Net MVC gedoeld maar ook TDD en de OAuth techniek.

Het doorlopen van het complete software ontwikkel traject met het opstellen van de daarbij behorende planning is voor mijn gevoel ook zeer succesvol doorlopen en heeft mij veel inzicht gegeven op hoe een project van begin tot einde succesvol doorlopen kan worden.

Ik ben er blij mee dat het product op dit moment live staat en dat er al applicaties zijn (de Flash Widgets) die er gebruik van maken. Daarbij wordt er in de nabije toekomst gewerkt aan nieuwe Apps die gebruik gaan maken van het product wat er opgeleverd is.

Glossary

.Net Framework
Software framework voor Microsoft Windows systemen. In dit framework bevinden zich onder andere functionaliteiten voor dataconnecties, web functionaliteiten en user interfaces.

Api
Standaard interface die een gebruiker aan kan roepen om bepaalde functionaliteiten van een systeem van derden te gebruiken.

Authenticatie
Het vaststellen van de identiteit van een gebruiker.

Autorisatie
Het vaststellen of de gebruiker voldoende rechten heeft na Authenticatie.

Batching
Het groeperen van uit te voeren functionaliteiten om deze in één keer achter elkaar uit te voeren.

Dependency
Afhankelijkheid tussen twee elementen binnen een software architectuur.

Deserialisatie
Het creëren van een object uit een geserialiseerde vorm.

Generics
Het runtime bepalen van het type van een object. Wordt gebruikt voor methodes waarbij het type van input onbekend is.

HTML
Opmaaktaal voor specificatie van documenten, voornamelijk gericht op internet pagina's.

IntelliSense
Autocompletion tool die in Visual Studio zit. Wordt gebruikt voor het afmaken van wat de gebruiker in aan het typen is en het weergeven van documentatie/parameters voor functies.

JSON
Open source en textbased standaard om data uit te wisselen, veel gebruikt om data te serialiseren en over een netwerk te versturen.

LINQ
Language Integrated Query biedt een standaard methode voor omgang met data uit verschillende type databronnen (databases, XML, tekstbestanden, arrays).

MoSCoW
Methodiek om eisen binnen een project een prioriteit toe te kennen.

- **M**ust have this - deze eis *moet* in het eindresultaat terugkomen, zonder deze eis is het product niet bruikbaar;
- **S**hould have this if at all possible - deze eis is zeer gewenst, maar zonder is het product wel bruikbaar;

- **C**ould have this if it does not affect anything else - deze eis mag alleen aan bod komen als er tijd genoeg is;
- **W**on't have this but would like to have this in the future - deze eis zal in dit project niet aan bod komen maar kan in de toekomst, bij een vervolg project, interessant zijn.

PList
Formaat gebruikt door de IPhone, de ondersteuning hiervoor bevind zich native op dit platform en het is een variant op XML.

Query
Opdracht die aan een database gegeven wordt, die mogelijk data terug kan geven.

Reflection
Dit is een techniek waarbij een programma in staat is runtime zijn eigen structuur te observeren en wijzigen.

Rss
Really Simple Syndication (eenvoudige gelijktijdige publicatie), een groepering van XML varianten die vooral door bloggers en nieuws sites gebruikt wordt om berichten te publiceren in een standaard formaat. Dit formaat kan door zogenaamde Rss readers ingelezen en getoond worden.

Serialisatie
Het omzetten van een object zo dat het geschikt wordt voor verzending over het web, vaak wordt er naar JSON of XML geserialiseerd.

Strongly Typing
Methode die restricties plaatst aan de interactie van waardes, bij Strongly Typing kunnen ints en strings niet bij elkaar opgeteld worden, wat in Weakly Typing (tegenhanger van Strongly Typing) wel kan.

TDD
Test Driven Development, ontwikkelmethode waarbij het opstellen van de tests voor het implementeren al gebeurt.

XML
Standaard voor syntax waarmee men gestructureerde gegevens kan weergeven als tekst en daarmee versturen over het internet.

Literatuurlijst

1 Tutorials MVC. *Microsoft.* [Online]
http://www.asp.net/mvc

2 Plan van aanpak opzet. *Zbc Kennisbank.* [Online]
http://zbc.nu/ict/project-management/standaard-plan-van-aanpak/

3 MVC controller lifetime. *Persoonlijke blog*. [Online]
http://stephenwalther.com/blog/archive/2008/03/18/asp-net-mvc-in-depth-the-life-of-an-asp-net-mvc-request.aspx

4 OAuth officiële site. *Officiele Site*. [Online]
http://oauth.net/

5 OAuth voorbeelden [Boek]
Professional Twitter Development: With Examples in .NET 3.5, Daniel Crenna, 0470531320

6 ASP.Net Developers blog [Online]
http://weblogs.asp.net/scottgu/

7 PList creatie. *Community*. [Online]
http://www.inspiredbytechnology.com/index.php/2011/01/05/creating-a-plist-file-from-sql-using-xmltextwriter-c/

8 C# Code conventie. *Microsoft.* [Online]
http://10rem.net/articles/net-naming-conventions-and-programming-standards---best-practices

9 XmlSerialiser vs DataContractSerialiser. *Persoonlijke blog.* [Online]
http://www.danrigsby.com/blog/index.php/2008/03/07/xmlserializer-vs-datacontractserializer-serialization-in-wcf/

10 AMMap. *Officiële site*. [Online]
http://www.ammap.com/

Bijlagen

Versie 1.0

Dit document bevat de documenten die bij het afstudeerverslag horen.

Robin Scholtes
1-3-2011

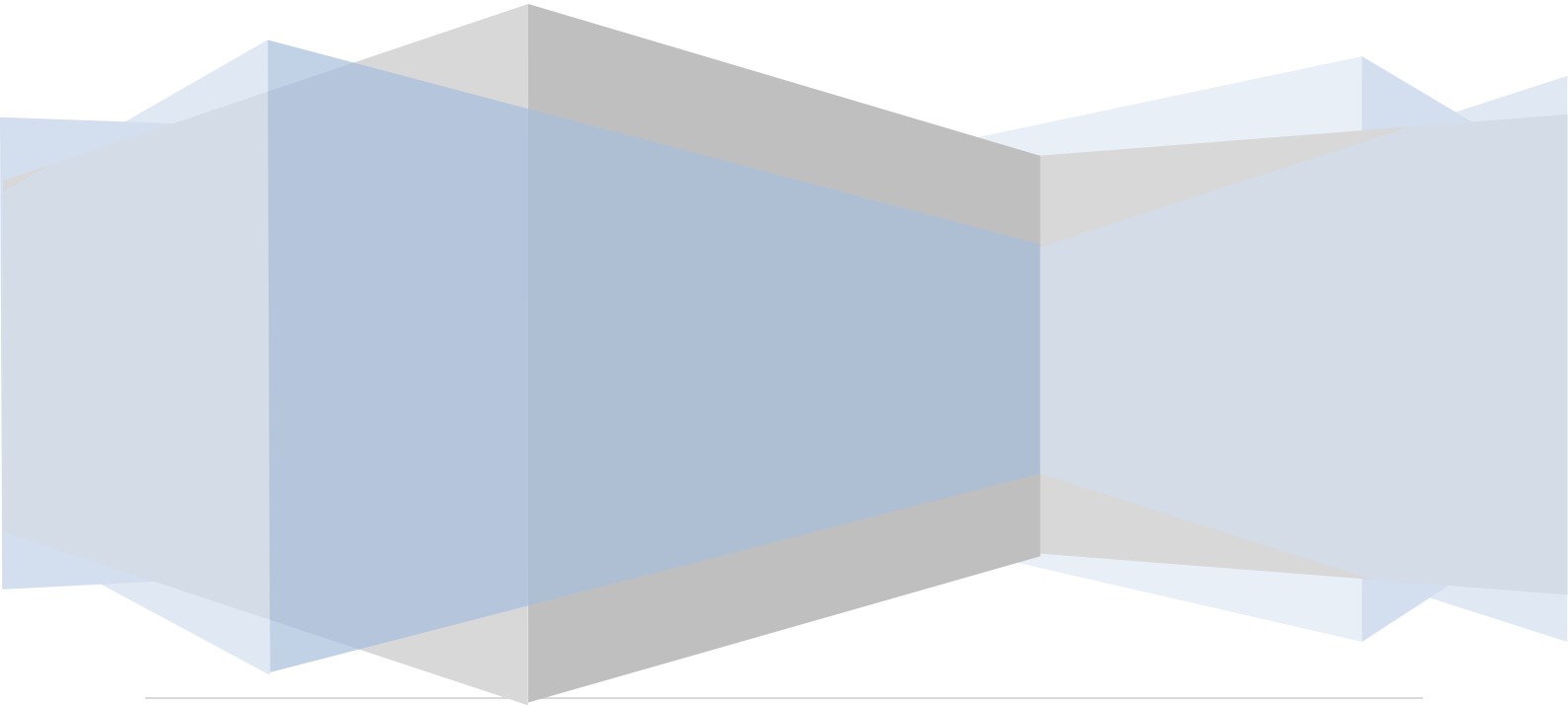
Inhoudsopgave

Plan van Aanpak

Versie 0.6

Dit plan beschrijft een project bij GameBasics gedurende 17 weken. In dit document worden afspraken gemaakt tussen de opdrachtgever en de opdrachtnemer over het wat, wanneer en hoe.

Robin Scholtes - 06000827
12-1-2011



Versie	Datum	Auteur(s)	Wijzigingen
0.1	15-11-2010	Robin	Initiële versie
0.2	17-11-2010	Robin	Wekelijkse afspraken opgenomen
0.3	22-11-2010	Robin	MSProject planning toegevoegd, gebruik van MoSCoW methode beschreven.
0.4	23-11-2010	Robin	Document opgemaakt, spelling en zinsconstructie gewijzigd.
0.5	25-11-2010	Robin	Omschrijving fases bijplanning toegevoegd.
0.6	12-1-2011	Robin	Nieuwe planning toegevoegd

Inhoudsopgave

1 Inleiding

In dit hoofdstuk wordt de aanleiding van het document beschreven. Daarbij wordt er in het kort besproken welke informatie er waar in het document te vinden is en welke regelingen er getroffen zijn met betrekking to het bijstellen en goedkeuren van wijzigingen binnen dit plan van aanpak.

1.1 Aanleiding

Dit document is geschreven om de uit te voeren werkzaamheden gedurende het project vast te leggen. Het opstellen van de werkzaamheden en eisen is voor zowel de opdrachtgever als de opdrachtnemer van belang. Dit om een duidelijke planning te kunnen maken en misverstanden te voorkomen.

1.2 Accordering en bijstelling

Dit document wordt in de eerste weken van het project opgesteld, maar zal naarmate het project vordert gewijzigd worden en eventueel verder ingevuld worden. Om eventuele onduidelijkheden of onenigheden te voorkomen zal het projectplan door de opdrachtgever goedgekeurd worden en zullen tevens alle wijzigingen door hem goedgekeurd worden.

1.3 Structuur document

Als eerste volgt hieronder een beschrijving van het bedrijf en de contact gegevens van de personen die bij dit project direct betrokken zijn. Vervolgens wordt de opdracht algemeen beschreven, hierbij wordt specifiek aandacht geschonken aan de aanleiding, probleemstelling, werkzaamheden, te gebruiken media en de scope van het project. Dit wordt gevolgd door een algemene beschrijving van de voorwaarden waaraan de opdrachtgever en de opdrachtnemer zich dienen te houden. In het volgende hoofdstuk is de planning van het project te zien met de bijbehorende mijlpalen. Tenslotte volgt er informatie over de geplande waarborging van kwaliteit binnen dit project.

2 Organisatie

In dit hoofdstuk volgt een korte beschrijving van alle betrokkenen bij dit project. Als eerste wordt het bedrijf besproken waarvoor het project uitgevoerd wordt. Dit wordt aangevuld met contactgegevens van de personen die actief aan dit project meewerken.

2.1 Bedrijf

GameBasics BV. ontwikkelt en exploiteert online community games in Nederland en in het buitenland. De missie van GameBasics is om een first-mover te zijn en een wereldwijd herkenbare speler op het gebied van online management webgames. De afgelopen 6 jaar is GameBasics aanzienlijk gegroeid en spelen meer dan 500.000 mensen hun voornaamste product, www.onlinesoccermanager.nl. Op dit moment telt GameBasics 11 werknemers. Deze zijn werkzaam in administratie, development en marketing. De huidige twee directeuren zijn degene die het bedrijf hebben opgericht en de eerste versie van Online Soccer Manager geschreven hebben.

Gamebasics B.V.	
Adres	Röntgenlaan 25
Postcode	2719 DX, Zoetermeer
E-mail	info@gamebasics.nl
Telefoon	+31 (0)79 341 4450

2.2 Projectleden

Opdrachtgever	
Naam	Jeroen Derwort
E-mail	Jeroen@derwort.nl
Telefoon	

Opdrachtnemer	
Naam	Robin Scholtes
E-mail	R.scholtes@hotmail.com
Telefoon	+31 (0)6 14 55 69 23

3 Project

In dit hoofdstuk volgt een algemene beschrijving over hoe de opdracht tot stand is gekomen en wat het doel van de opdracht is. Daarbij wordt er beschreven welke activiteiten er tijdens het project uitgevoerd worden, volgens welke methodes/technieken en welke producten hier uit volgen. Tenslotte wordt de opdracht afgebakend zodat voor zowel de opdrachtgever als opdrachtnemer er een duidelijk beeld is van wat het project omvat.

3.1 Aanleiding

In 2001 is de eerste versie van Online Soccer Manager ontwikkeld in Classic ASP en Visual Basic. Deze software was oorspronkelijk gebouwd voor een kleine set requirements en weinig gebruikers. Door de sterke groei van Gamebasics is de software steeds verder geëvolueerd en uitgebreid. De technische grenzen van wat er mogelijk is met de eenvoudige scripttechnologie zijn echter nagenoeg bereikt.

De complexe structuur van een applicatie die meertalig is, multi-platform, meerdere game werelden en servers omspant en communiceert met vele externe online systemen noopt tot een nieuwe, volwassen omgeving die deze requirements kan faciliteren.

3.2 Probleemstelling

GameBasics is een transitie project gestart om de website te herschrijven zodat deze weer aansluit bij de toekomst. Daarbij wordt gebruik maakt van innovatieve technologie, zoals ASP.NET MVC framework. Op dit moment draaien zowel de mobiele als de versie voor computer browsers op verschillende software, terwijl de achterliggende functionaliteiten hetzelfde zijn. De MVC technologie moet ervoor zorgen dat dezelfde codebase kan worden gebruikt voor verschillende versies van de software. Voor het communiceren met externe systemen, zoals sociale media en mobiele applicaties, is een API nodig die binnen het nieuwe framework is ontwikkeld.

3.3 Doelstellingen

De doelstelling is om in ASP.NET MVC een oplossing te implementeren die ingezet kan worden om Online Soccer Manager te koppelen aan Mobiele Apps en Social Media. De functionaliteiten van deze oplossing moeten het mogelijk maken de huidige website zo volledig mogelijk te bedienen. Met de MVC technologie is het mogelijk een model te ontwikkelen wat er voor zorgt dat verschillende media met de

functionaliteiten van Online Soccer Manager kunnen communiceren. De nadruk van de opdracht komt hierbij te voornamelijk te liggen bij het creëren van het Model(M) en de Controllers(C). De Views(V) worden gebruikt ter demonstratie van de werking van de achterliggende functionaliteiten.

3.4 Op te leveren producten

Binnen het project zijn een aantal producten die opgeleverd dienen te worden, deze worden hieronder kort besproken. In de planning is terug te vinden wanneer deze opgeleverd worden. Bij de verschillende producten die hieronder genoemd worden staat in het kort uitgelegd wat er zich in deze documenten bevind.

- Inception Document
  In dit document wordt de haalbaarheid en de scope van het project beschreven, de initiële use cases en worden de initiële functionele eisen opgesteld.
- Elaboration Document
  Dit documenten zullen bestaan uit:
  - Functionele eisen, dit is een beschrijving van de huidige functionaliteiten waaraan de uiteindelijke oplossing dient te voldoen.
  - Ontwerp documentatie, hierin wordt van te voren bepaald hoe de oplossing opgebouwd wordt en hoe de gewenste functionaliteiten functioneren
- Construction Document
  In dit document wordt de werking van de code gedocumenteerd en eventuele uitleg verleend over het gebruik van de code.
- Code
  Dit zal bestaan uit de geschreven code met de daarbij behorende projects en solutions van Visual Studio.
- Testrapport
  Tijdens de implementatie fase zal de code aan Unit test worden onderworpen om de kwaliteit van het opgeleverde product aan te tonen. Daarbij zal er gebruik gemaakt worden van Mockup tests om de werking van de functionaliteiten aan te tonen.
- Transition Document
  Dit document zal de acceptatie tests van de opdrachtgever bevatten samen met een beschrijving/advies voor de in product name van het opgeleverde project.

3.5 Activiteiten

- Inlezen in de omgeving
- Plan van aanpak (Inception fase)
- Definitie van eisen (Inception fase / Elaboration fase)
- Functioneel ontwerp (Elaboration fase)
- Technisch ontwerp (Elaboration fase)
- Testen (Construction fase)
- Implementatie (Construction fase)
- Overdracht en acceptatie (Transition fase)
- Onderzoek verslag

3.6 Randvoorwaarden en afbakeningen

Het is van belang dat zowel opdrachtgever als opdrachtnemer het eens zijn over wat er wel en niet gedaan wordt tijdens het project en welke voorwaardes er zijn om aan het project te kunnen beginnen.

3.6.1 Randvoorwaarden

- Er dient een werkende omgeving aanwezig te zijn met internet toegang.
- De opdrachtgever heeft minimaal een keer per week tijd om voortgang te bespreken.

3.6.2 Afbakeningen

- Het project zal zich uitsluitend bezighouden met de activiteiten beschreven in hfdst. 3.5, tenzij er in overleg anders besloten wordt.
- De opdrachtgever zal alleen de resultaten uit hfdst 3.4 ontvangen, tenzij er in overleg anders besloten wordt.
- Het project wordt uitgevoerd over een periode van 17 weken.

3.7 Methoden, technieken en software

Tijdens dit project zal er gebruik gemaakt worden van een aantal methodes, in dit hoofdstuk zal er een overzicht te vinden zijn van welke methoden dat zullen zijn. Daarnaast zal hier ook vermeld worden welk technieken gebruikt zullen worden tijdens het project en welke software er nodig is om het project te voltooien.

3.7.1 Methoden

- RUP
  Bij de fasering van het project wordt gebruik gemaakt van de RUP methodiek.
- UML
  Voor het modelleren van het eindproduct wordt gebruik gemaakt van de UML modelleer techniek.
- MoSCoW
  Bij het toekennen van een prioriteit aan een eis wordt gebruik gemaakt van de MoSCoW methodiek.

3.7.2 Technieken

- C#
  Dit is de programmeertaal waarmee gewerkt zal worden gedurende het project.
- ASP.net MVC
  De te implementeren API dient gebruik te maken van de functionaliteiten die ASP.net MVC aanbiedt.
- JSON / XML
  Technieken die gebruikt kunnen worden om data in een speciaal formaat naar de gebruiker te sturen en van de gebruiker te ontvangen.
- TDD
  Test Driven Development, dit is een techniek waarbij eerst de unit test opgesteld worden om vervolgens te implementatie te schrijven.

3.7.3 Software

- Microsoft Visual Studio 2010
- Microsoft Office / Open-Office
- Microsoft Visio
- Microsoft Project
- MySQL
- GIT (Versie beheer).

4 Project voorwaarden

Om het project zo soepel mogelijk te laten lopen en voor iedere persoon zijn verantwoordelijkheden zo duidelijk mogelijk te krijgen wordt in dit hoofdstuk de verantwoordelijkheden beschreven van alle betrokkenen bij dit project.

4.1 Project voorwaarden opdrachtgever

- De opdrachtgever is verantwoordelijk voor begeleiding van de student tijdens het project.
- De opdrachtgever biedt een plek aan waar de student kan werken.
- De opdrachtgever geeft feedback op de opgeleverde documenten en geeft zijn akkoord voor deze documenten.
- De opdrachtgever geeft de student tijd tijdens zijn stage te werken aan zijn eindverslag en daarmee niet bezig te zijn met werkzaamheden voor het bedrijf.

4.2 Project voorwarden opdrachtnemer

- De opdrachtnemer, is ingeschreven voor de opleiding Informatica aan de Haagse Hogeschool.
- De opdrachtnemer stelt een plan van aanpak op met een daarbij behorende planning en houd de opdrachtgever op de hoogte van voortgang.
- De opdrachtnemer werkt zelfstandig en verantwoordelijk.
- De opdrachtnemer documenteert iedere fase en is verantwoordelijk voor de overdracht van het eindproduct.

5 Planning

In dit hoofdstuk wordt er per fase beschreven van wanneer tot wanneer welke fase loopt en welke activiteiten er in de betreffende fase afgerond worden. Daarbij wordt er beschreven welke producten er wanneer opgeleverd worden. De grafische weergave van de hieronder beschreven planning is te vinden onder Bijlage A

5.1 Activiteitenplanning

- Inception fase

  De Inception fase zal plaatsvinden gedurende de eerste vier weken van het project. Tijdens deze fase wordt dit projectplan opgesteld en worden de eisen van het op te leveren product geanalyseerd. De eisen zullen volgens de MoSCoW methode een prioriteit toegekend krijgen, en samen met de Use cases opgenomen worden in het Inception Document. Tevens zal er gedurende de Inception fase tijd gemaakt worden om bekend te raken met de ASP.Net MVC omgeving en de reeds bestaande architectuur. Na het vaststellen van de prioriteiten en eisen zal het laatste deel van de Inception fase gebruikt worden om specifieke oplossingen en technieken te onderzoeken die van belang zijn voor het eindproduct.
- Elaboration fase

  In deze fase worden de functionele en technische eisen omgezet in een ontwerp. In de Inception fase is er al gekeken naar bepaalde technische aspecten, mochten er tijdens het ontwerpen nog onduidelijkheden zijn zullen deze naar voren komen tijdens het prototypen van het ontwerp. Uiteindelijk zal het ontwerp met de gemaakte keuzes opgeleverd worden in het Elaboration Document.

- Construction fase
  Tijdens de Construction fase zal de daadwerkelijke implementatie plaatsvinden. Het doel is om in deze fase te beginnen met het opstellen van de unit test van de te implementeren functionaliteiten. Gedurende de hele periode waarin daadwerkelijk geïmplementeerd wordt zal ook getest worden. Ten slotte wordt er aan het einde van deze fase de code gedocumenteerd en zal het Construction document opgeleverd worden waarin de werking en het gebruik van het product beschreven staat.
- Transition fase
  Tijdens de transition fase zal de acceptatie test door de opdrachtgever uitgevoerd worden en zullen er eventuele kleine laatste bijstellingen gedaan worden. Aan het einde van deze fase wordt het Transition document opgeleverd, in dit document staat beschreven hoe de geïmplementeerde oplossing in gebruik kan worden genomen.

6 Kwaliteitsborging

Om er voor te zorgen dat de kwaliteit van het project gewaarborgd wordt, word er in dit hoofdstuk beschreven op welke manieren er gekeken wordt naar de kwaliteit van het project en hoe er omgegaan wordt met bepaalde risico's.

6.1 Kwaliteit en zekerheid

De kwaliteit en zekerheid zal gemeten en beheerst worden aan de hand van de zogenaamde Gotick factoren (Geld, Organisatie, Tijd, Informatie, Communicatie en Kwaliteit). Deze zijn hieronder in een tabel uitgewerkt.

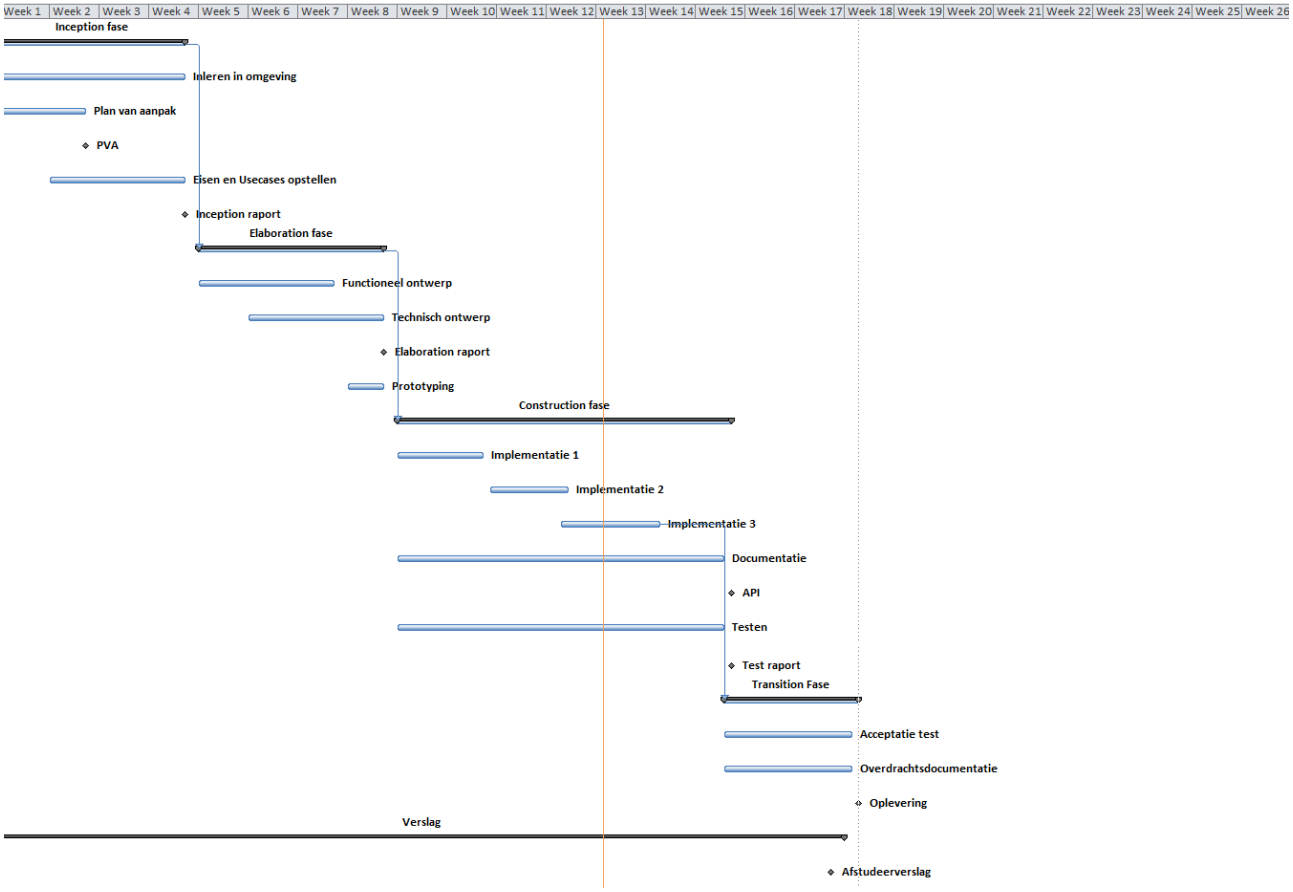
Geld	Het budget voor dit project beperkt zich tot de looptijd welke gegeven is om het gehele project te doorlopen. Deze looptijd omvat 17 weken.
Organisatie	Nvt.
Tijd	Er zal wekelijks overleg plaats vinden om te kijken of er nog volgens planning gewerkt wordt. Tijdens deze overleg voeringen zal er ook gekeken worden wat er zou moeten veranderen om weer volgens planning te kunnen werken indien dit niet meer het geval is.
Informatie	Er is afgesproken met de opdracht gever wekelijks schriftelijk een rapport op te leveren waarin de werkzaamheden van de afgelopen week staan met de status van deze werkzaamheden en de planning voor de volgende week.
Communicatie	Om te waarborgen dat de opdrachtgever op de hoogte blijft van het project zal er wekelijks een contact moment zijn, waarin onder andere de zaken genoemd in de tabel bij Tijd besproken worden. Hierbij is met de opdrachtgever afgesproken iedere maandag een voortgangsgesprek te hebben.
Kwaliteit	Om de kwaliteit van het project te waarborgen, wordt er wekelijks overleg gepleegd met de opdrachtgever om over de voortgang van het project te spreken en de eisen/planning evt. bij te stellen.

6.2 Risicomanagement

In dit hoofdstuk worden alle risico's beschreven die tijdens dit project voor kunnen komen. Bij de risico's wordt beschreven hoe ze te herkennen zijn, of ze te voorkomen zijn, wat er moet gebeuren mochten ze voorkomen en wat het impact op het project is.

- **Dataverlies**
  Herkenning: ontbrekende documenten of broncode.
  Voorkomen: gebruik van GIT voor back-ups en versiebeheer.
  Oplossing: Bij verlies van data is er dus een back-up aanwezig binnen versiebeheer.
  Impact: ++ (Bij daadwerkelijk verlies van gegevens kan er weken aan werk verloren gaan).
- **Gebrek aan kennis**
  Herkenning: Bij het programmeren en ontwerpen komt er naar boven dat een projectlid niet genoeg kennis heeft.
  Voorkomen: Op tijd beginnen met inlezen van de materie.
  Oplossing: Hulp vragen aan een expert.
  Impact: ++ (Als er te laat geconstateerd wordt dat er onvoldoende kennis is kan het eindresultaat moeilijk tot niet behaald worden).
- **Product voldoet niet aan de eisen**
  Herkenning: De acceptatietest worden niet met succes afgenomen.
  Voorkomen: Door goed te overleggen met de opdrachtgever wat er in het eindproduct moet komen en dit op papier vast te leggen.
  Oplossing: Overleg met de opdrachtgever.
  Impact: ++ (Als er te laat geconstateerd wordt dat het product niet aan de eisen voldoet kan het eindresultaat moeilijk tot niet behaald worden).
- **Product werkt niet naar behoren**
  Herkenning: Een van de uit te voeren tests faalt.
  Voorkomen: Fouten voorkomen kan moeilijk, het is noodzakelijk er vroeg bij te zijn zodat het probleem opgelost kan worden.
  Oplossing: Frequent testen.
  Impact: +.
- **Product is niet af / Product wordt te laat op geleverd**
  Herkenning: Bepaalde functionaliteiten zijn niet aanwezig in het eindproduct.
  Voorkomen: Door goed te plannen en duidelijk op papier op te stellen wat er gemaakt moet worden. Daarbij is het bespreken van voortgang met de opdrachtgever belangrijk.
  Oplossing: Uitloop.
  Impact: ++.
- **Product is niet of slecht gedocumenteerd**
  Herkenning: De documentatie is niet duidelijk voor de eindgebruiker of andere ontwikkelaars.
  Voorkomen: Door op tijd te beginnen met documentatie en deze voor te leggen aan mensen voor feedback.
  Oplossing: Feedback van mensen verwerken.
  Impact: +.

Bijlage A


Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Week 21
Week 22
Week 23
Week 24
Week 25
Week 26
Inception fase
Inleren in omgeving
Plan van aanpak
PVA
Eisen en Usecases opstellen
Inception raport
Elaboration fase
Functioneel ontwerp
Technisch ontwerp
Elaboration raport
Prototyping
Construction fase
Implementatie 1
Implementatie 2
Implementatie 3
Documentatie
API
Testen
Test raport
Transition Fase
Acceptatie test
Overdrachtsdocumentatie
Oplevering
Verslag
Afstudeerverslag

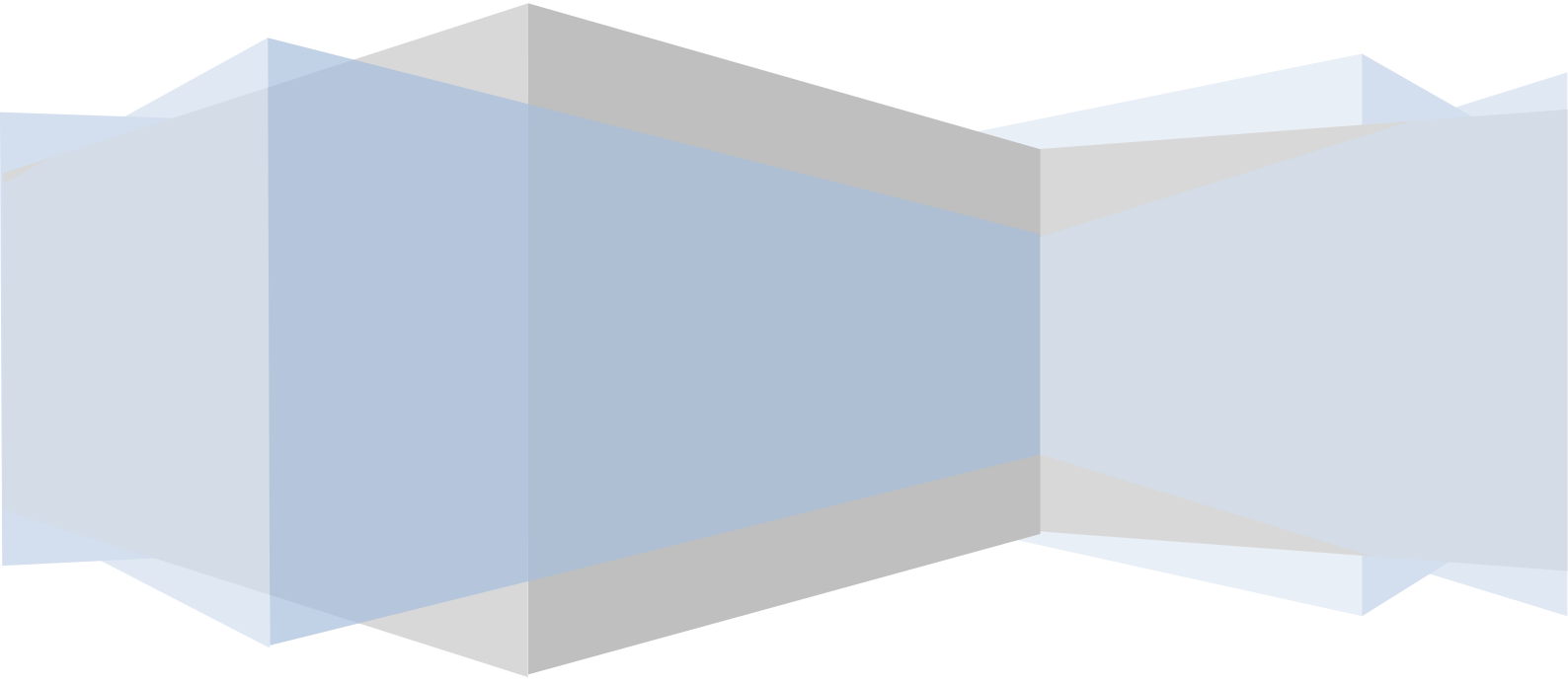
GameBasics

Inception Document

Versie 0.6

Dit document omschrijft welke functionaliteiten er nu aanwezig zijn en aan welke functionele en technische eisen het op te leveren product dient te voldoen.

Robin Scholtes
24-11-2010



Versie	Datum	Auteur(s)	Wijzigingen
0.1	17-11-2010	Robin	Initiële versie.
0.2	22-11-2010	Robin	Use cases en omschrijvingen toegevoegd.
0.3	22-11-2010	Robin	Begin gemaakt aan technische eisen.
0.4	23-11-2010	Robin	Document opgemaakt, spelling en zinsconstructie gewijzigd.
0.5	25-11-2010	Robin	Functionele eisen opgesteld.
0.6	29-11-2010	Robin	Functionele eisen afgerond.

Inhoudsopgave

1 Inleiding

Dit document is geschreven met als doel de functionaliteiten van het op te leveren product vast te stellen en de technieken die binnen het product gebruikt moeten worden of aangeboden moeten worden. Het is van groot belang om in dit document het op te leveren product zo volledig mogelijk te beschrijven. Hiermee wordt ervoor gezorgd dat de opdrachtgever en de opdrachtnemer op één lijn zitten wat betreft het eindproduct.

1.1 Structuur document

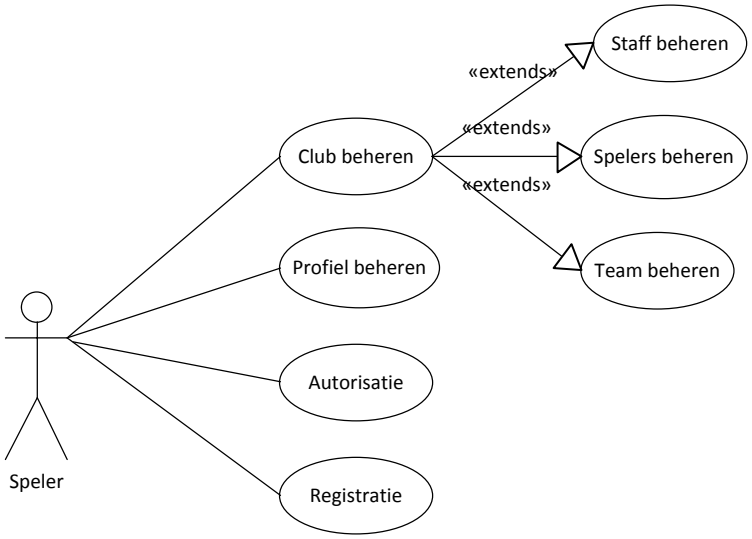
In dit document zullen daarom als eerste een aantal use cases opgesteld worden om de huidige functionaliteiten van Online Soccer Manager in kaart te brengen. Er is hier gekozen om te beginnen met een use case die de algemene “core” functionaliteiten beschrijft en om deze vervolgens in verdere use cases verder te specificeren, dit omdat er teveel functionaliteiten zich binnen het systeem vinden om in één diagram weer te geven.

De use cases worden opgevolgd door een lijst met functionele en technische specificaties die geïmplementeerd dienen te worden. Deze eisen zullen volgens de MoSCoW methode ingedeeld worden.

2 Use cases

In dit hoofdstuk worden de functionaliteiten van het bestaande Online Soccer Manager systeem in kaart gebracht en wordt er zo bepaald wat er in de verdere loop geïmplementeerd dient te worden.

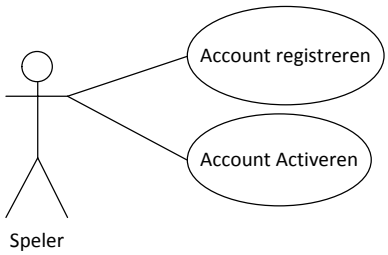
2.1 Algemene use case



Hierboven zijn de algemene functionaliteiten beschreven die binnen Online Soccer Manager mogelijk zijn. De cases beschreven in deze use case zullen in de rest van dit document verder uitgewerkt worden en verder beschreven worden. Deze use case dient slechts ter illustratie en verduidelijking van de algemene functionaliteiten.

2.2 Registratie

Use case



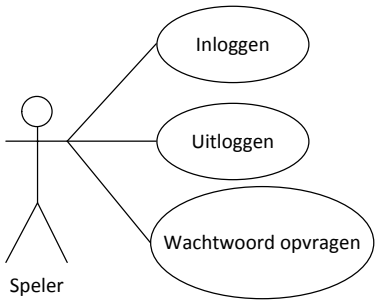
Use case omschrijving

Naam	Account registreren
Samenvatting	De actor voert zijn gegevens in om een account te registreren
Actoren	Speler
Aanname	De actor heeft nog geen account
Beschrijving	1. Actor voert gegevens in. 2. Actor verstuurt gegevens. 3. Systeem valideert gegevens. 4. Systeem verstuurt e-mail met gegevens naar de Actor.
Uitzonderingen	De Actor vult foute gegevens in waardoor registratie niet succesvol is.
Resultaat	De Actor heeft een account

Naam	Account activeren
Samenvatting	De actor activeert zijn account
Actoren	Speler
Aanname	De actor heeft zich geregistreerd en een activatie mailtje ontvangen
Beschrijving	1. Actor voert code in op activeringspagina
Uitzonderingen	
Resultaat	De Actor heeft een geactiveerd account

2.3 Autorisatie

Use case



Use case omschrijving

Naam	Inloggen
Samenvatting	De actor voert zijn gegevens in om in te loggen.
Actoren	Speler
Aanname	De actor heeft een geactiveerd account
Beschrijving	1. De Actor voert zijn gegevens in. 2. Het systeem autoriseert de actor.
Uitzonderingen	De actor voert incorrecte gegevens in en wordt doorverwezen naar een pagina met foutafhandeling.
Resultaat	De Actor is ingelogd

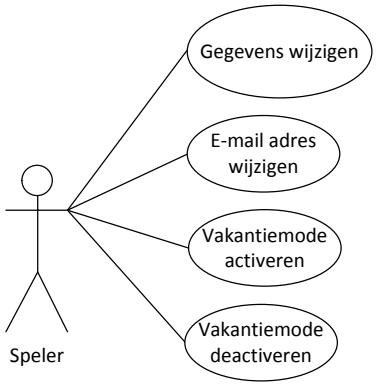
Naam	Uitloggen
Samenvatting	De actor logged uit.
Actoren	Speler
Aanname	De actor is ingelogd
Beschrijving	De actor drukt op uitloggen
Uitzonderingen	
Resultaat	De Actor wordt uitgelogd

Naam	Wachtwoord opvragen
Samenvatting	De actor heeft een account
Actoren	Speler
Aanname	De actor heeft een geactiveerd account en is in bezit van het email adres.
Beschrijving	1. De actor voert zijn e-mail adres in. 2. De actor voert de bevestigingscode in 3. De actor ontvangt een mailtje met zijn wachtwoord
Uitzonderingen	De Actor vult foutieve gegevens in of de bevestigingscode is fout, het

	wachtwoord opvragen mislukt.
Resultaat	De gebruiker heeft zijn wachtwoord opgevraagd

2.4 Profiel beheren

Use case



Use case omschrijving

Naam	Gegevens wijzigen
Samenvatting	De actor wijzigt zijn persoonlijke gegevens
Actoren	Speler
Aanname	De actor heeft een geactiveerd account en is ingelogd.
Beschrijving	1. De actor wijzigt zijn profiel. 2. De actor drukt op opslaan. 3. Het systeem slaat het profiel opnieuw op
Uitzonderingen	De gebruiker vult foutieve gegevens in waardoor wijzigen van profiel niet mogelijk is (verboden symbolen etc.)
Resultaat	De actor zijn profiel wordt gewijzigd.

Naam	E-mail adres wijzigen
Samenvatting	De actor wijzigt het e-mail adres van zijn profiel
Actoren	Speler
Aanname	De actor heeft een geactiveerd account en is ingelogd.
Beschrijving	1. De actor voert zijn nieuwe e-mail adres in. 2. De actor wordt uitgelogd. 3. De actor ontvangt een mailtje met een bevestigingscode, 4. De actor voert deze code in
Uitzonderingen	Het nieuwe e-mail adres is ongeldig waardoor het wijzigen niet mogelijk is.
Resultaat	De actor zijn e-mail adres is gewijzigd.

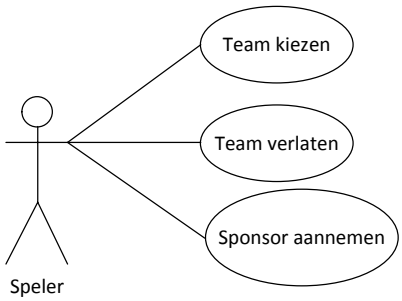
Naam	Vakantie mode activeren
Samenvatting	De actor zet zijn profiel in vakantie mode.
Actoren	Speler
Aanname	De actor heeft een geactiveerd account en is ingelogd.
Beschrijving	1. De actor drukt op vakantiemode 2. De actor bevestigt de meldingen. 3. De actor ontvangt een mailtje met instructies om de

	vakantiemode weer op te heffen.
Uitzonderingen	Als de Actor zijn account al in vakantie mode is, kan deze niet nogmaals in vakantiemode gezet worden.
Resultaat	De actor zijn profiel is in vakantiemode

Naam	Vakantie mode deactiveren
Samenvatting	De actor verandert de status van zijn profiel van vakantiemode naar actief.
Actoren	Speler
Aanname	De actor zijn account staat in vakantiemode en de actor heeft de instructiemailtje nog wat verstuurd is na het activeren van de vakantiemode.
Beschrijving	1. De actor gaat naar de link in het e-mailtje. 2. De actor logged weer in op zijn account.
Uitzonderingen	
Resultaat	De actor zijn account is weer actief.

2.5 Club beheren

Use case



Use case omschrijving

Naam	Team kiezen
Samenvatting	De actor selecteert een land en een team.
Actoren	Speler
Aanname	De actor heeft een geactiveerd account, is ingelogd en heeft nog geen team.
Beschrijving	1. De actor kiest een land 2. De actor kiest een competitie 3. De actor kiest een team
Uitzonderingen	De actor heeft niet recentelijk een team verlaten in dezelfde competitie.
Resultaat	De actor heeft een team.

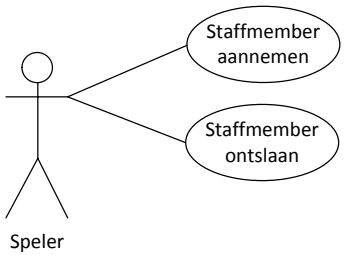
Naam	Team verlaten
Samenvatting	De actor verlaat zijn huidige team
Actoren	Speler
Aanname	De actor heeft een geactiveerd account, is ingelogd en heeft een team.
Beschrijving	1. De actor drukt op ontslag nemen. 2. De actor bevestigd.

Uitzonderingen	
Resultaat	De actor heeft geen team meer.

Naam	Sponsor aannemen
Samenvatting	De actor selecteert een sponsor.
Actoren	Speler
Aanname	De actor is ingelogd, heeft een team en heeft nog minimaal 1 lege sponsorplek.
Beschrijving	1. De speler drukt op sponsor aannemen 2. De speler selecteert een sponsor
Uitzonderingen	
Resultaat	De speler heeft een sponsor aangenomen.

2.6 Staff beheren

Use case



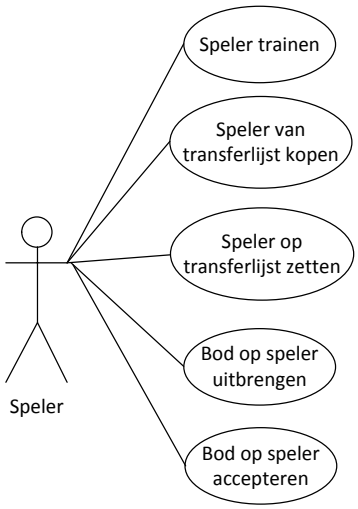
Use case omschrijving

Naam	Staffmember ontslaan
Samenvatting	De actor selecteert een member van de staff en ontslaat deze.
Actoren	Speler
Aanname	De actor is ingelogd, heeft een team en heeft minimaal 1 staffmember
Beschrijving	1. De actor selecteert een staffmember. 2. De actor drukt op ontslaan 3. De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft een staffmember ontslagen

Naam	Staffmember aannemen
Samenvatting	De actor selecteert een type staffmember en neemt deze aan.
Actoren	Speler
Aanname	De actor is ingelogd, heeft een team en heeft minimaal 1 vacature voor een staffmember
Beschrijving	1. De actor kiest een type staffmember 2. De actor drukt op aannemen 3. De actor bevestigt.
Uitzonderingen	Bepaalde staffmembers vereisen een seizoenskaart, wanneer deze niet actief is kan een Actor geen Scout of Commercieel Manager aannemen.
Resultaat	De actor heeft een staffmember aangenomen.

2.7 Spelers beheren

Use case



Use case omschrijving

Naam	Speler trainen
Samenvatting	De actor selecteert een speler om deze extra te trainen
Actoren	Speler
Aanname	De actor is ingelogd, heeft een team en de geselecteerde speler staat niet in de basis
Beschrijving	1. De actor selecteert de speler die getraind moet worden. 2. De actor drukt op bevestigen.
Uitzonderingen	Wanneer een speler geblesseerd is kan deze niet getraind worden.
Resultaat	De actor heeft de speler voor 1 speelronde getraind.

Naam	Speler van transferlijst kopen
Samenvatting	De actor koopt een speler die op de transferlijst staat.
Actoren	Speler
Aanname	De actor is ingelogd, heeft een team en heeft voldoende geld
Beschrijving	1. De actor selecteert een speler 2. De actor drukt op kopen en bevestigen.
Uitzonderingen	
Resultaat	De actor heeft een speler toegevoegd aan zijn selectie.

Naam	Speler op transferlijst zetten
Samenvatting	De actor selecteert een speler uit zijn team op te verkopen
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team.
Beschrijving	1. De actor selecteert een speler. 2. De actor voert in hoeveel de speler moet kosten. 3. De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft een speler op de transferlijst gezet

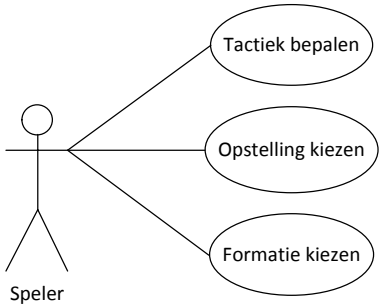
Naam	Bod op speler uitbrengen
Samenvatting	De actor selecteert een speler van een team en brengt een bod hierop

	uit
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team
Beschrijving	1. De actor selecteert de speler. 2. De actor voert in hoeveel hij op de speler wil bieden. 3. De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft een bod uitgebracht op een speler

Naam	Bod op een speler accepteren
Samenvatting	De actor accepteert een bod gemaakt op een speler uit zijn team
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team.
Beschrijving	1. De actor accepteert het bod.
Uitzonderingen	
Resultaat	De actor heeft een speler verkocht.

2.8 Team beheren

Use case



Use case omschrijving

Naam	Tactiek bepalen
Samenvatting	De actor bepaalt de tactiek die zijn team hanteert in de eerstvolgende wedstrijd.
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team.
Beschrijving	De actor stelt de opties voor tactiek in De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft zijn tactiek bepaald.

Naam	Opstelling kiezen
Samenvatting	De actor bepaalt welke spelers er op de bank zitten en welke er in de basis starten de volgende speelronde.
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team.
Beschrijving	Voor iedere positie kiest de actor een speler. De actor vult zijn veld selectie aan met spelers op de bank.

	De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft zijn opstelling voor de volgende speelronde bepaald.

Naam	Formatie kiezen
Samenvatting	De actor kiest de formatie waarin de volgende wedstrijd gespeeld wordt.
Actoren	Speler
Aanname	De actor is ingelogd en heeft een team.
Beschrijving	De actor selecteert een van de voorgedefinieerde formaties. De actor bevestigt.
Uitzonderingen	
Resultaat	De actor heeft de formatie van zijn team gewijzigd.

3 Functionele en technische eisen

In dit hoofdstuk worden de technische en functionele eisen besproken. Deze zullen vervolgens ingedeeld worden in verschillende prioriteiten aan de hand van de MoSCoW methode. De voornaamste eis van de opdrachtgever is het zo volledig mogelijk bedienen van de huidige functionaliteiten. Om hier meer duidelijkheid over te krijgen wordt hieronder puntsgewijs besproken wat er minimaal geïmplementeerd gaat worden en welke technische eisen hieraan gesteld worden.

3.1 Functionele eisen

- Gebruik moet zich kunnen autoriseren zodat er bewerkingen op zijn account uitgevoerd kunnen worden. **(M)**
- Gebruiker moet kunnen uitloggen. **(M)**
- Gebruiker moet zich kunnen registreren. **(S)**
- Gebruiker moet zijn profiel kunnen wijzigen. **(W)**
- Gebruiker moet zijn profiel in en uit vakantie mode kunnen halen. **(W)**
- Gebruiker moet gebeurtenissen kunnen opvragen per:
    - Team **(S)**
    - Competitie **(S)**
    - Filteren op: **(C)**
        - Stadion
        - Transfers
        - Fans
        - Players
        - Training
        - Board
        - Managers
        - Sponsors
        - Matches
        - Forum
- Gebruiker moet uitslagen kunnen opvragen per week. **(M)**
- Gebruiker moet competitietussenstand kunnen opvragen. **(M)**
- Gebruiker moet scoreboard kunnen bekijken. **(S)**
    - Indien seizoenskaart
- Gebruiker moet krant kunnen lezen **(M)**
- Gebruiker moet overzicht van team kunnen opvragen **(M)**
- Gebruiker moet formatie kunnen wijzigen **(M)**
    - 3 - 2 - 5
    - 3 - 3 - 4
    - 3 - 4 - 3
    - 3 - 5 - 2
    - 4 - 2 - 4
    - 4 - 3 – 3
    - 4 - 4 - 2
    - 4 - 5 - 1
    - 5 - 2 - 3
    - 5 - 3 - 2

    - 5 - 4 - 1
    - 6 - 3 – 1
- Gebruiker moet per linie kunnen instellen hoe deze zich gedraagt. **(M)**
- Gebruiker moet opstelling kunnen bepalen en wie er op de bank zitten. **(M)**
- Gebruiker moet specialisten kunnen aanwijzen: **(M)**
    - Teamcaptain
    - Penalty nemer
    - Vrije trap nemer
    - Corner nemer
- Gebruiker moet tactiek kunnen bepalen: **(M)**
    - Speelstijl :
        - Voorzichtig
        - Normaal
        - Hard
        - Extreem
    - Tactiek
        - Zoek de vleugels
        - Loer op de counter
        - Schiet van afstand
        - Opbouw van achteruit
        - Gebruik de lange bal
    - Mentaliteit
    - Baltempo
    - Pressie spelen
    - Dekking
    - Buitenspelval
- Gebruiker moet besloten training kunnen beleggen. **(M)**
- Gebruiker moet een trainingskamp kunnen beleggen. **(M)**
- Gebruiker moet overzicht van beoordelingen van spelers kunnen opvragen per team. **(M)**
- Gebruiker moet overzicht van boden op spelers kunnen zien en deze accepteren/afwijzen. **(M)**
- Gebruiker moet bod op speler kunnen uitbrengen. **(M)**
- Gebruiker moet speler op transferlijst kunnen zetten voor een zelf te bepalen bedrag. **(M)**
- Gebruiker moet speler van transferlijst kunnen kopen. **(M)**
- Gebruiker moet speler kunnen scouten(indien seizoenskaart) **(M)**
- Gebruiker moet per linie een speler kunnen trainen. **(M)**
- Gebruiker kan de transferlijst opvragen en deze filteren op linie. **(M)**
- Gebruiker kan advies vragen aan het bestuur. **(M)**
- Gebruiker kan overzicht van financiën opvragen. **(M)**
- Gebruiker kan personeel aannemen / ontslaan. **(M)**
- Gebruiker kan sponsors aannemen op noord/oost/zuid/west tribune**(M)**
- Gebruiker kan stadion uitbreiden (1 upgrade tegelijk): **(M)**
    - Capaciteit uitbreiden
    - Trainingsvelden aanleggen

    - Hotdog kraam
    - Verlichting aanleggen
    - Kwaliteit veld verbeteren.
- Gebruiker kan een overzicht van de competitie opvragen met de wijzigingen ten opzichte van vorige speelronde. **(M)**
- Gebruiker kan de stand van het bekertoernooi opvragen van de rondes die al gespeeld zijn. **(M)**
- Gebruiker kan per team het wedstrijdschema opvragen. **(M)**
- Gebruiker kan een overzicht opvragen van de volgende speeldag. **(M)**
- Gebruiker kan van server wisselen **(C)**
- Gebruiker kan een Competitie joinen **(C)**
- Gebruiker kan toegang tot een beschermde league aanvraag en zijn aanvraag terugtrekken**(C)**

3.2 Technische eisen

- Er dient gebruik gemaakt te worden van ASP.net MVC. **(M)**
- De gegevens dienen in de volgende formaten opgevraagd te kunnen worden:
    - XML**(M)**
    - JSON**(M)**
    - ATOM**(M)**
    - CVS**(C)**
- MYSQL **(M)**
- Autorisatie met behulp van **(M)**
    - oAuth
    - openId
    - eigen implementatie
- Code moet voldoen aan de Microsoft C# standaarden. **(S)**
- Code moet leesbaar en voorzien zijn van commentaar. **(M)**
- Database dient benadert te worden met repository pattern. **(M)**
- Code moet voorzien zijn van unit tests. **(M)**
- Correcte afhandeling van ActionResult. **(S)**
- Scheiding maken tussen externe en interne functionaliteiten. **(C)**

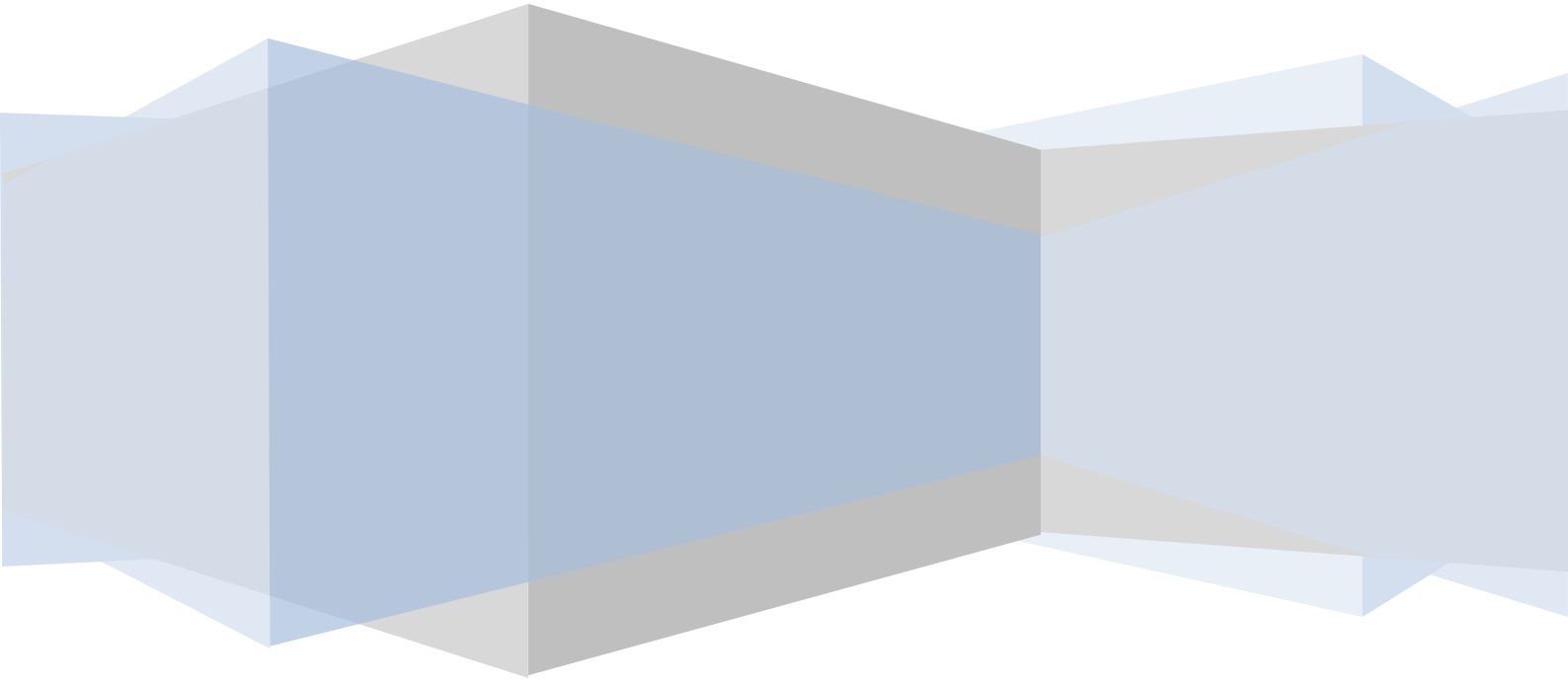
GameBasics

Vooronderzoek

Versie 1.0

Dit document beschrijft het vooronderzoek in meer technische details.

Robin Scholtes
15-3-2011



ViewData en ViewModel

Op het moment dat een gebruiker een aanvraag doet, wordt er binnen ASP.Net MVC automatisch bepaald welke controller er aangesproken dient te worden, de controller bepaalt op zijn beurt automatisch welke View er gebruikt moet worden. Dit is echter de eerste stap, het zou ook mooi zijn wanneer er data vanuit je Model in je View getoond kan worden. Hiervoor zijn er binnen ASP.Net MVC twee standaard manieren.

- **ViewData**
  Bij de ViewData techniek wordt er in de Controller direct tegen de View gezegd welke data er getoond wordt.

```
01. public ActionResult Voorbeeld()
02. {
03.     ViewData["Titel"] = "voorbeeld tekst";
04.     return View(ViewData);
05. }
```

  Bovenstaand stukje code zorgt er voor dat er in de View behorende bij de actie Voorbeeld de tekst Titel beschikbaar is. Deze manier is niet Strongly Typed. Dit betekent dat er in de HTML editor geen gebruik gemaakt kan worden van IntelliSense.
- **ViewModel**
  Bij ViewModels wordt de data die in een View nodig is in een zogenaamde Container gestopt en deze wordt vervolgens aan de View gegeven. Dit heeft als gevolg dat alle data die je nodig hebt in je View netjes bij elkaar verzameld is en dat deze data Strongly Typed is, waardoor het gebruik van IntelliSense wel werkt.

```
01. public ActionResult Voorbeeld()
02. {
03.     VoorbeeldViewModel ViewModel = new VoorbeeldViewModel();
04.     return View(ViewModel);
05. }
```

Er is gekozen voor het gebruik van ViewModels, niet alleen omdat dit ervoor zorgt dat in de View Editor gebruik gemaakt kan worden van de IntelliSense hulp, maar ook om er voor te zorgen dat je Controllers leesbaar blijven. Wanneer er voor het gebruik van ViewData gekozen zou worden krijg je lange lijsten met toekennen van data aan de Views, met ViewModels zit deze data verborgen in een aparte container.

ActionFilterAttribute

Binnen ASP.Net MVC is het mogelijk om “filters” te gebruiken op classes en functies. Dit betekent dat als je een filter aan een functie toekent je automatisch een extra actie voor en na het uitvoeren van de aangeroepen actie kan uitvoeren. Een filter ziet er zo uit:

```
01. [VoorbeeldFilter]
02. public ActionResult Voorbeeld()
03. {
04.     return View();
05. }
```

De bovenstaande code zorgt ervoor dat voor het aanroepen van de voorbeeld functie de `VoorbeeldFilter` filter aangeroepen wordt. Dit kan gebruikt worden om te kijken of een gebruiker bijvoorbeeld de rechten heeft om een functie aan te roepen. Er is gekozen om soortgelijke attributen te gebruiken bij functionaliteiten die geëxporteerd moeten kunnen worden naar formaten die te gebruiken zijn bij sociale media. Met het creëren van een `ActionFilterAttribute` wordt er voor gezorgd dat een functionaliteit op één plek geïmplementeerd kan worden en vervolgens op andere plaatsen hergebruikt kan worden. Dit is voor de kwaliteit van het eindproduct van groot belang, omdat dit voorkomt dat er grote hoeveelheden van code herhaald wordt door het hele systeem. De specifieke filter die bij het vooronderzoek bedacht is zorgt er voor dat er voor het uitvoeren van een functionaliteit gekeken wordt welk formaat van data er aangevraagd wordt en of er parameters bij het request zitten. Na het uitvoeren van de functie, kan de data omgezet worden in het aangevraagde formaat.

ActionResult

In ASP.Net MVC geven controllers altijd objecten terug die de interface `ActionResult` gebruiken. Zo worden `ViewResult` gegenereerd wanneer er een HTML pagina getoond moet worden en wordt er gebruik gemaakt van een `RedirectResult` wanneer de actie de gebruiker ergens naar toe moet sturen. Dit zijn enkel e standaard implementaties die zich bevinden binnen ASP.Net MVC. Er is gekozen voor het creëren van een eigen `ActionResult` om er voor te zorgen dat er voor de API zowel data als een actiebericht terug gekoppeld kunnen worden naar de gebruiker. Dit kan met de bestaande results niet. Daarbij zorgt het ervoor dat het voor developers makkelijker is om de API aan te roepen, omdat er gebruik gemaakt wordt van bekende types, die ook herkend worden binnen ASP.Net MVC. De belangrijkste reden is echter dat controllers niet alleen benaderbaar moeten zijn voor de “normale” website maar ook voor de API aanroepen, dit betekent dat er 1 type moet zijn waaraan alle teruggegeven waardes moeten voldoen. Aangezien `ActionResult` reeds gebruikt worden, was dit eigenlijk de enige oplossing. Het terug geven van gegevens die de interface `ActionResult` niet implementeren zou ervoor zorgen dat er niet gebruik gemaakt kan worden van één functie om gegevens op te vragen, maar dan zouden er twee functies moeten komen die exact dezelfde bewerkingen doen alleen verschillende types objecten terug geven.

Routing

Om te kijken hoe er bepaald kon worden wat voor soort request er door de gebruiker werd gedaan is er gekeken naar de Routing binnen ASP.Net MV. De routing is een expressie waarin een formaat wordt opgeslagen van een URL waarmee er bepaald kan worden welke acties op welke controllers uitgevoerd kunnen worden. De default Route in ASP.Net MVC ziet er als volgt uit:

```
01. routes.MapRoute(
02.         "Default", // Route name
03.         "{controller}/{action}/{id}", // URL with parameters
04.         new {  controller = "Home",
05.                action = "Index",
06.                id = UrlParameter.Optional } // Parameter defaults
07. );
```

De route die hierboven beschreven wordt zorgt ervoor dat bijvoorbeeld, http://voorbeeld.com/home/index/4 , doorverwezen wordt naar de Controller home en de actie Index wordt aangeroepen met parameter 4. Bij de API die ontwikkeld wordt dient het mogelijk te zijn om naast de normale HTML ook andere formaten op te vragen. API’s zoals Google en Twitter maken het mogelijk om bijvoorbeeld http://voorbeeld.com/home/index.xml/4 aan te roepen, wat aangeeft

dat XML terug gegeven moet worden. Dit kan binnen ASP.Net gerealiseerd worden met Routing. Om deze reden is de volgende Route opgesteld die naast de default route werkt.

```
routes.MapRoute(
       "Default2", // Route name
       "{controller}/{action}.{format}/{id}", // URL with parameters
       new {  controller = "Home",
              action = "Index",
              format = "HTML",
              id = UrlParameter.Optional } // Parameter defaults
);
```

ASP.Net MVC bepaalt achter de schermen zelf welke route er gekozen moet worden. Het bleek echter in eerste instantie dat deze route niet werkte, en dat er een Page Not Found error werd gegenereerd wanneer een URL werd opgevraagd met een toevoeging. Na lang onderzoek bleek dit probleem simpel op te lossen, het toevoegen van de routes moest omgedraaid worden. Het bleek dat ASP.Net MVC de URL aan route 1 kon matchen, door als action “index.xml” te kiezen. Deze functie bestaat echter niet in de controller en genereert daarom een Page Not Found. Het omdraaien van de Routes zorgt ervoor dat er eerst gekeken wordt of de route gematched kan worden met formaat, als dit het geval is zal deze route gekozen worden, zo niet dan zal hij terug vallen op de standaard. Een ander alternatief was geweest om van het request de header uit te lezen en te kijken welk formaat er mee werd gestuurd door de gebruiker, om zo te bepalen welk formaat er terug gestuurd dient te worden. Het probleem hiermee is echter dat je altijd data moet opsturen in het formaat dat je het terug wilt krijgen. Aangezien flexibiliteit hierin wenselijk was is er besloten om te kiezen voor extra Routing.

5.2.2 Repository pattern

Binnen de bestaande versie van Online Soccer Manager wordt er gebruik gemaakt van repositorys om de verschillende databases te benaderen. Het repository pattern creëert een extra laag tussen de daadwerkelijke database en Models. Een repository kan beschreven worden als een collectie in het geheugen waar aan objecten toegevoegd, van verwijderd en in bewerkt kunnen worden. Bij het implementeren van het repositorys moet er gelet worden op het aanbrengen van interfaces om de testbaarheid van de applicatie te garanderen. Dit betekent dat stel we hebben een Producten tabel waar producten in opgeslagen staan, de daadwerkelijke repository aan deze interface moet voldoen.

```
public interface IProductRepository
{
       Product GetById(int id);
       void Add(Product product);
       void Remove(Product product);
}
```

Door het gebruik van deze interface bij de repositorys kunnen er meerdere implementaties onder het repositorys gemaakt worden.

```
public class ProdcutRepository : IProductRepository
{
       Product GetById(int id) { /* voer database bewerkingen uit */ }
       void Add(Product product) { /* voer database bewerkingen uit */ }
       void Remove(Product product) ) { /* voer database bewerkingen uit */ }
}
```

```
07.
08. public class FakeProductRepository : IProductRepository
09. {
10.     Product GetById(int id) { return new Product("FakeProduct"); }
11.     void Add(Product product) { /* doet niets */ }
12.     void Remove(Product product) { /* doet niets */ }
13. }
```

Bovenstaand zien we de daadwerkelijke implementaties van de fictieve repository. De `ProdcutRepository` zal gebruikt worden voor productie omgeving en de `FakeProductRepository` zal gebruikt worden voor unit testen. Door tegen de Controller te zeggen welke repository hij moet gebruiken kan er makkelijk gewisseld worden tussen verschillende databronnen.

5.3.3 Serialisatie en Deserialisatie

De data die vanuit de models naar de gebruiker verstuurd wordt dient daarvoor eerst geserialiseerd te worden. Wanneer de gebruiker data in het formaat JSON wilt is er maar een mogelijkheid tot Serialisatie, namelijk de `JavaScriptSerializer`. Bij XML bleken er twee mogelijkheden te zijn waarmee er geserialiseerd kon worden.

XMLSerializer

De `XMLSerializer` zit in het .net Framework sinds versie 1.0 en maakt gebruik van opt-out Serialisatie. Dit betekent dat gegevens binnen een object geserialiseerd worden tenzij gezegd wordt dat dit niet moet. Daarbij geeft de `XMLSerializer` veel vrijheid in de manier waarop de XML opgemaakt wordt en kan er gebruik gemaakt worden van zogenaamde XSD's om XML te valideren.

DataContractSerializer

De `DataContractSerializer` is in .Net Framework 3 geïntroduceerd en richt zich voornamelijk op data Serialisatie binnen WCF(Windows Communication Foundation). Deze Serialisatie methode maakt gebruikt van opt-in, wat betekent dat je aangeeft wat je wel wilt serialiseren en wanneer het niet aangegeven wordt er dus ook niets geserialiseerd wordt. Daarbij is deze methode gemiddeld genomen 10% sneller dan de `XMLSerializer`.

Hoewel `DataContractSerializer` sneller is, is er toch gekozen om gebruik te maken van de `XMLSerializer`, om de code leesbaar te houden. Bij het gebruik van de `DataContractSerializer` moet er bij bijna alle gegevens aangegeven worden dat dit geserialiseerd moet worden wat er gewoonweg voor zorgde dat het overzicht binnen code kwijt raakte.

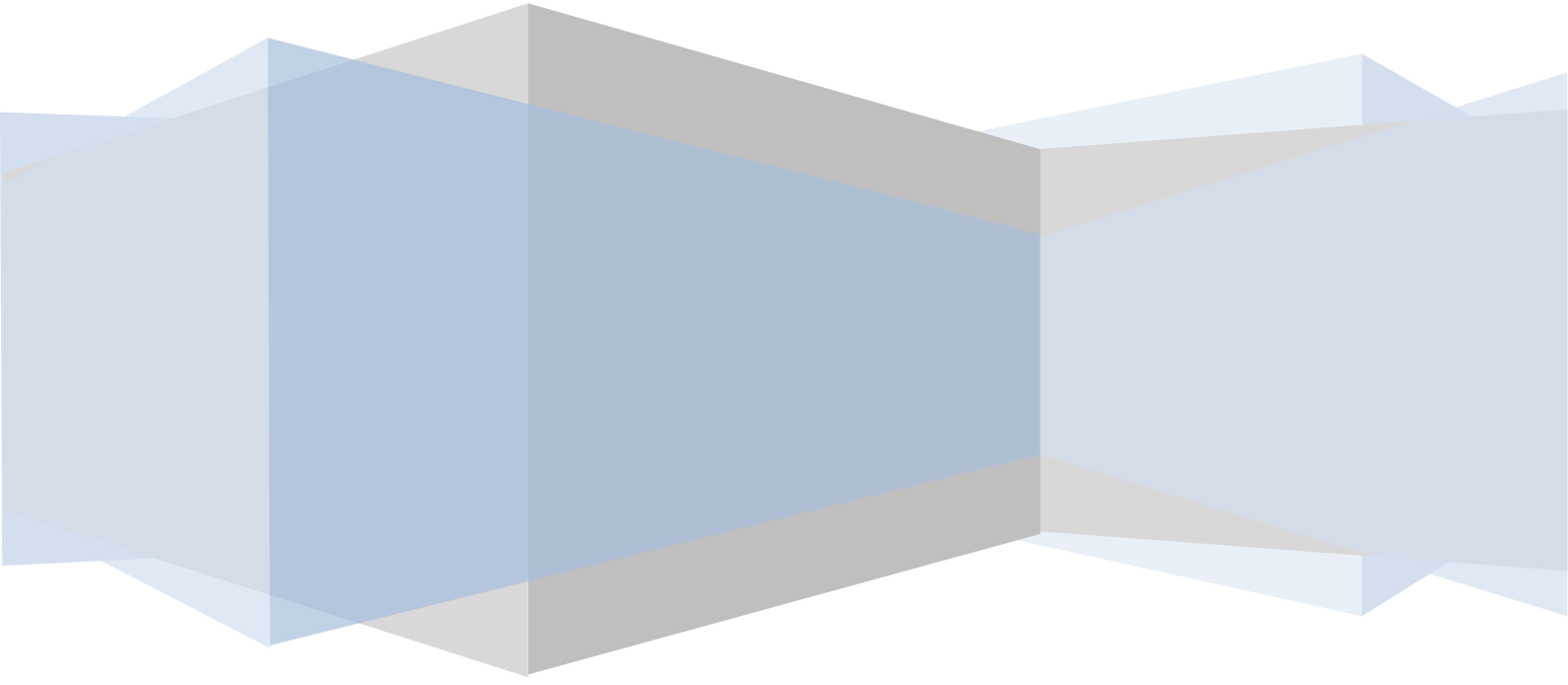
GameBasics

Elaboration Document

Versie 0.3

Dit document omschrijft op welke manier de API geïmplementeerd gaat worden en beschrijft de algemene flow van gegevens binnen het systeem.

Robin Scholtes
18-1-2011



Inhoudsopgave

1 Inleiding

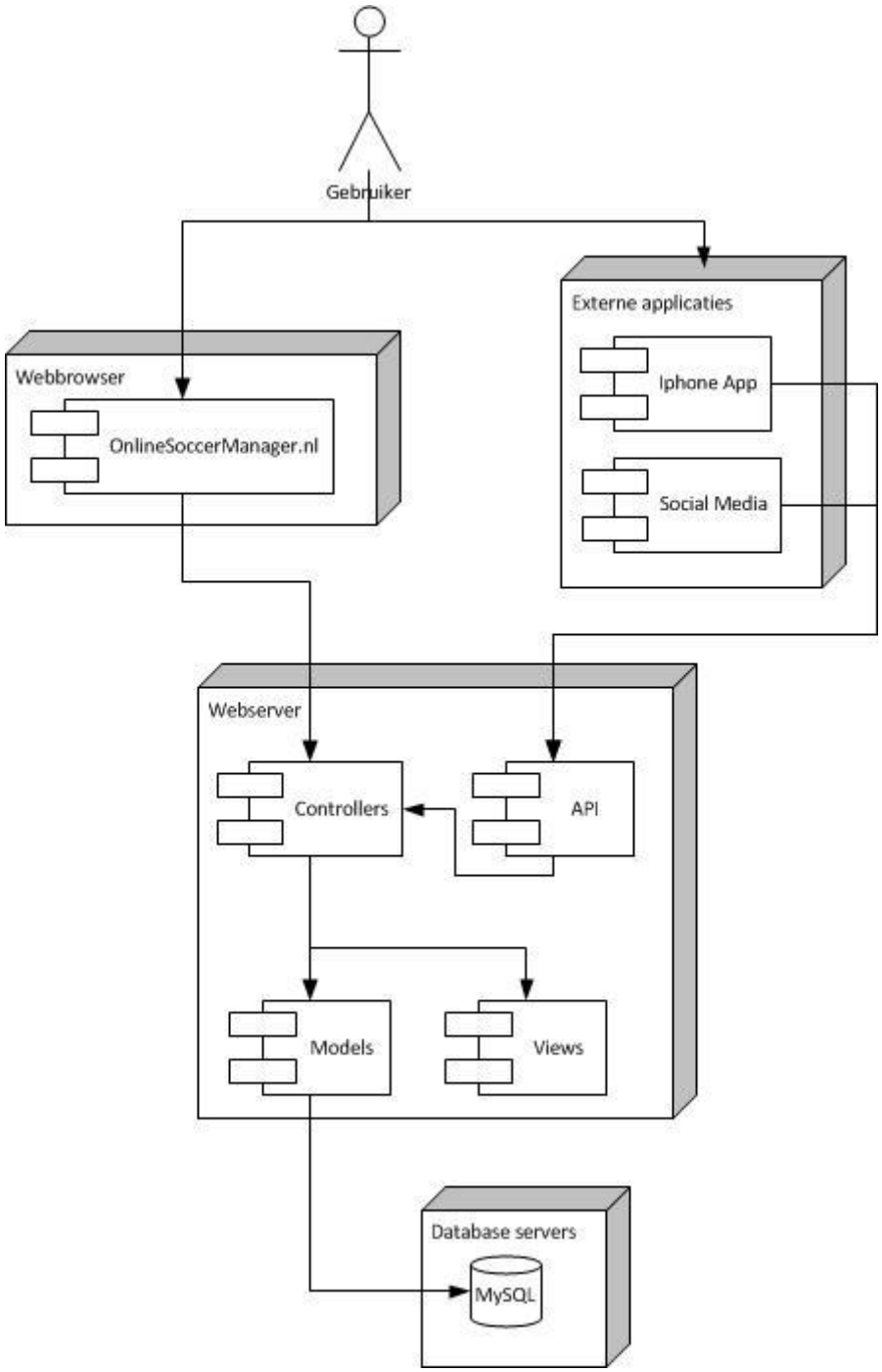
Dit document wordt geschreven met het doel te beschrijven welke functionaliteiten er op welke manier geïmplementeerd gaan worden. Het doel van dit document is om van te voren te bepalen op welke manieren de functionele en technische eisen gerealiseerd gaan worden.

1.1 Structuur document

In dit document zal met behulp van de UML methode de te implementeren oplossing beschreven worden. Er is gekozen om met behulp van Flowcharts de interne werking van de applicatie te beschrijven. Daarbij zal er Deployment diagram te vinden zijn met een algemene beschrijving van welke functionaliteiten waar aangeboden worden. Vervolgens zal er een Klassediagram te zien zijn met uitleg van het ontwerp. Dit klasse diagram zal ondersteund worden door verschillende sequentiediagrammen. In dit document zijn de functionele en technische eisen niet opgenomen, omdat deze in het Inception document zijn terug te vinden.

2 Deployment Diagram

Het Deployment Diagram wordt gebruikt om binnen een systeem de werking tussen verschillende componenten (nodes) te beschrijven.



Het bovenstaande model beschrijft de indeling van het systeem zoals het nu ontwikkeld wordt. Hieronder volgt een korte uitleg van de componenten en wat hun werkingen zijn.

- **Web browser**
  De Web browser is het middel wat de gebruiker gebruikt om de website te bezoeken. Het systeem is via verschillende url's bereikbaar om zo meerdere talen te ondersteunen.

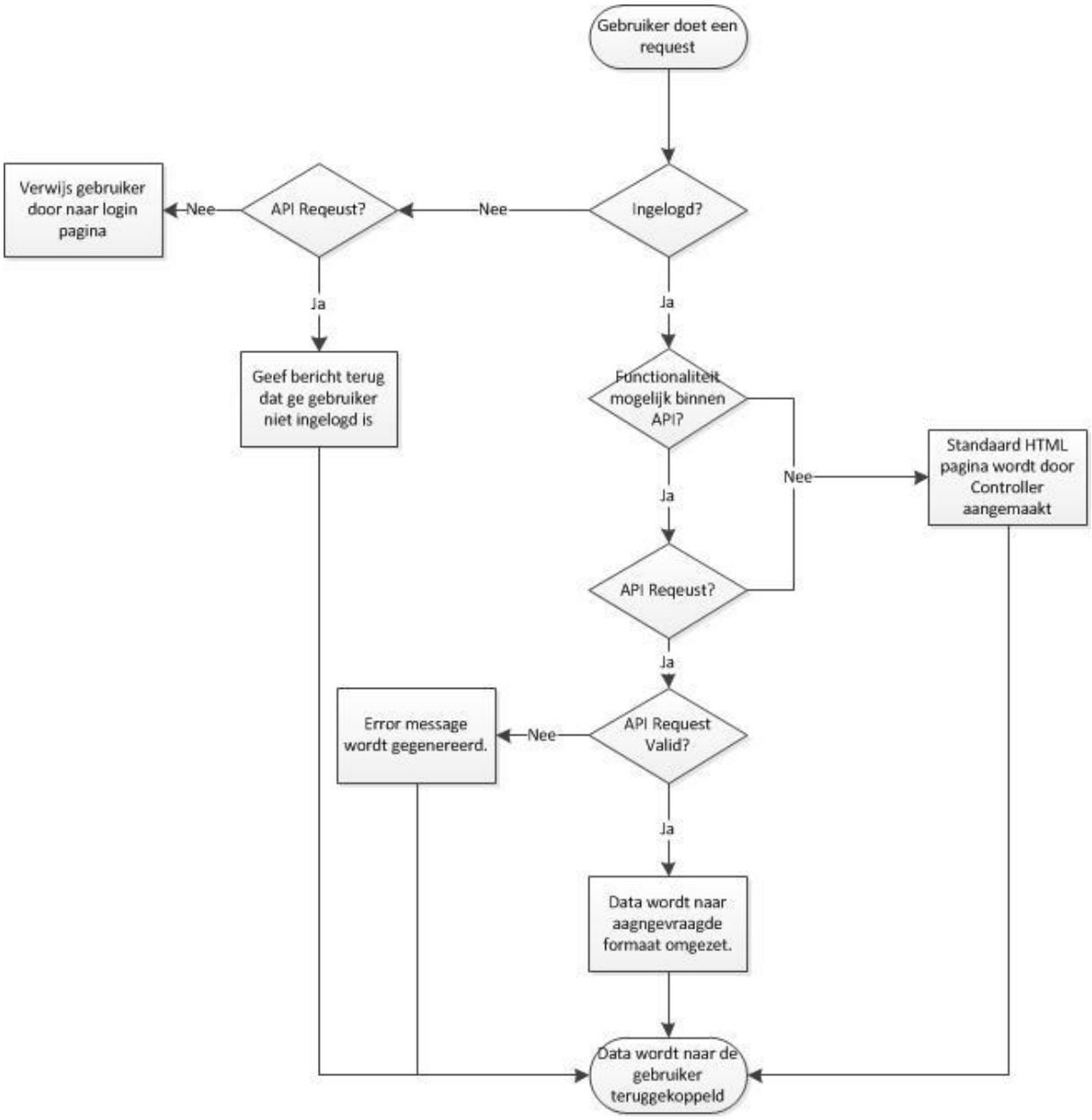
- **Externe Applicaties**
  Deze applicaties maken gebruik van de API laag om zo gegevens uit de bestaande databases te halen. Dezelfde informatie wordt verschaft aan de website.

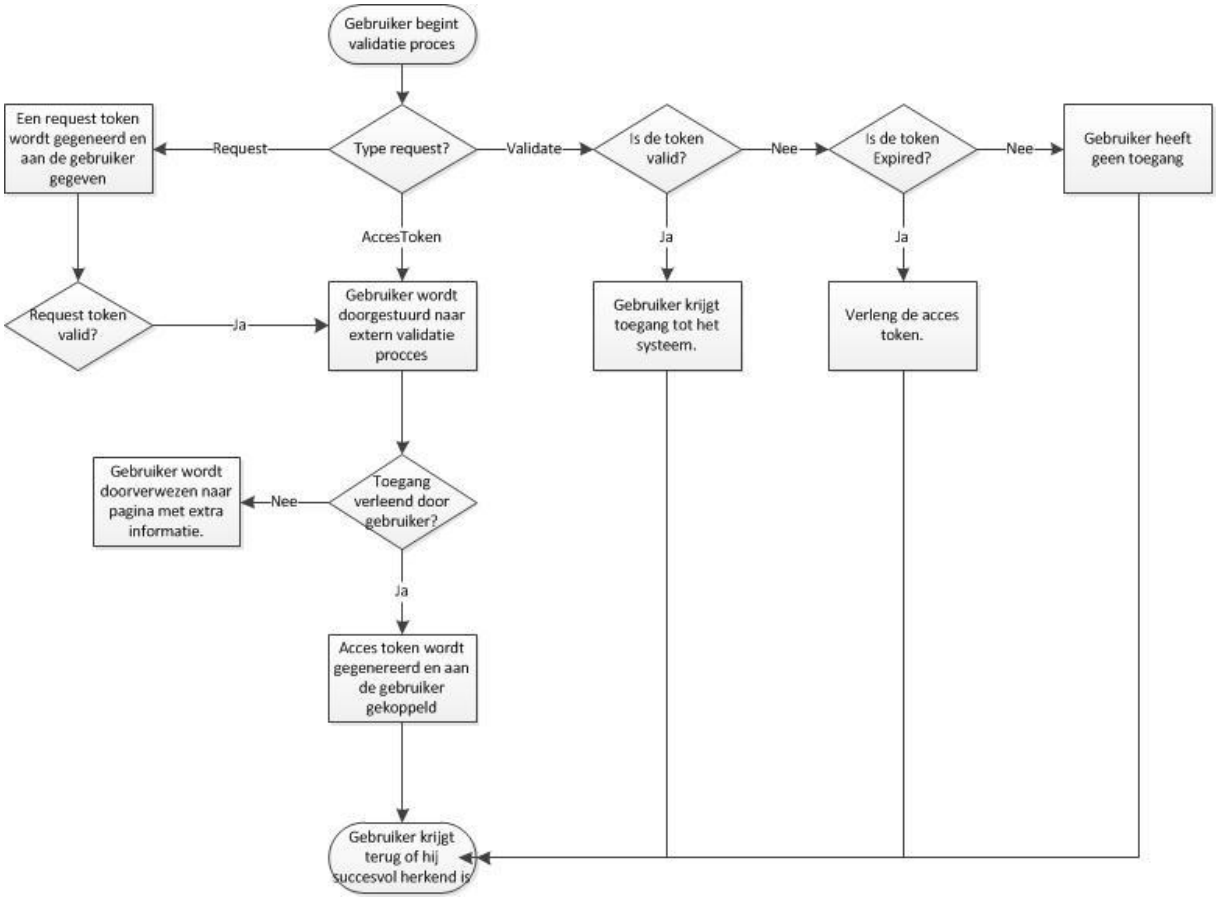
- **WebServer**
  De Webserver is het onderdeel waar de website fysiek gehost wordt. Dit onderdeel valt uiteen in de MVC structuur met een overkoepelende API voor externe toegang.

- **Database Servers**
  Het systeem maakt gebruik van meerdere database servers voor verschillende competities. Deze zijn benaderbaar via de repositorys.

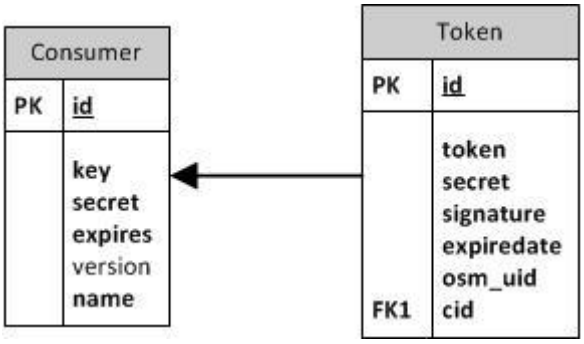
3 Flowcharts

In dit hoofdstuk wordt de flow van gegevens beschreven voor een request van een gebruiker en het autorisatie proces met OAuth.




Gebruiker begint validatie proces
Type request?
Request
Een request token wordt gegeneerd en aan de gebruiker gegeven
Request token valid?
Ja
Gebruiker wordt doorverwezen naar pagina met extra informatie.
Nee
Validate
Is de token valid?
Nee
Is de token Expired?
Nee
Gebruiker heeft geen toegang
AccesToken
Gebruiker wordt doorgestuurd naar extern validatie procces
Toegang verleend door gebruiker?
Ja
Acces token wordt gegenereerd en aan de gebruiker gekoppeld
Ja
Gebruiker krijgt toegang tot het systeem.
Ja
Verleng de acces token.
Gebruiker krijgt terug of hij succesvol herkend is

4 Database diagram



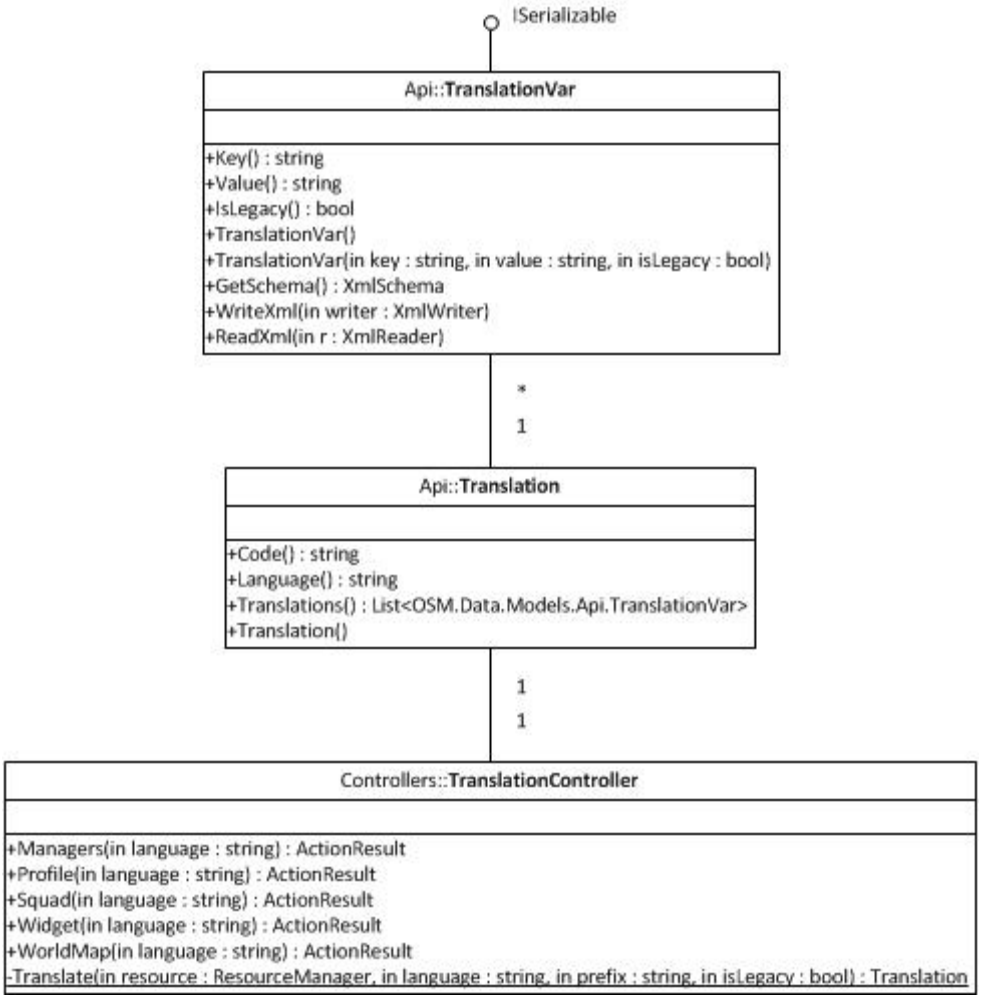
Voor het gebruik van OAuth is een aparte database opgesteld. In deze database worden Consumers opgeslagen en Tokens. Consumers zijn applicaties die binnen het systeem geregistreerd staan en toegang hebben tot de API. Een token is een user specifiek identificatie middel wat gekoppeld is aan een applicatie. Een user kan meerdere tokens hebben en een consumer kan meerdere tokens hebben. Een user kan echter maar 1 keer een token met een bepaalde consumer krijgen. Bij het token wordt het OSM_UID opgeslagen. Dit is het gebruikers id van de daadwerkelijke speler. Bij de authorisatie wordt de token gevalideerd en omgezet naar het gebruikersid.

5 Classdiagram

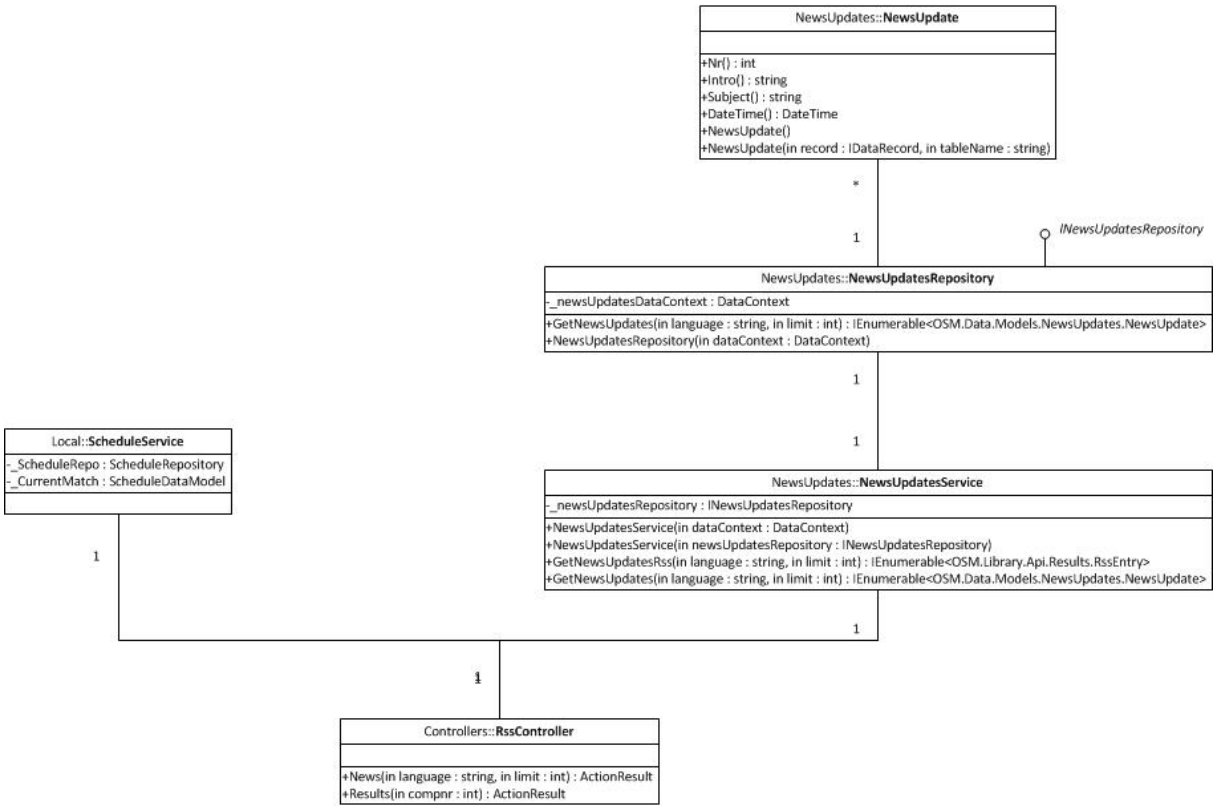
In dit hoofdstuk worden de classdiagrams beschreven van de API implementatie, de OAuth module en de implementatie binnen het systeem.

5.1 Flash Widgets

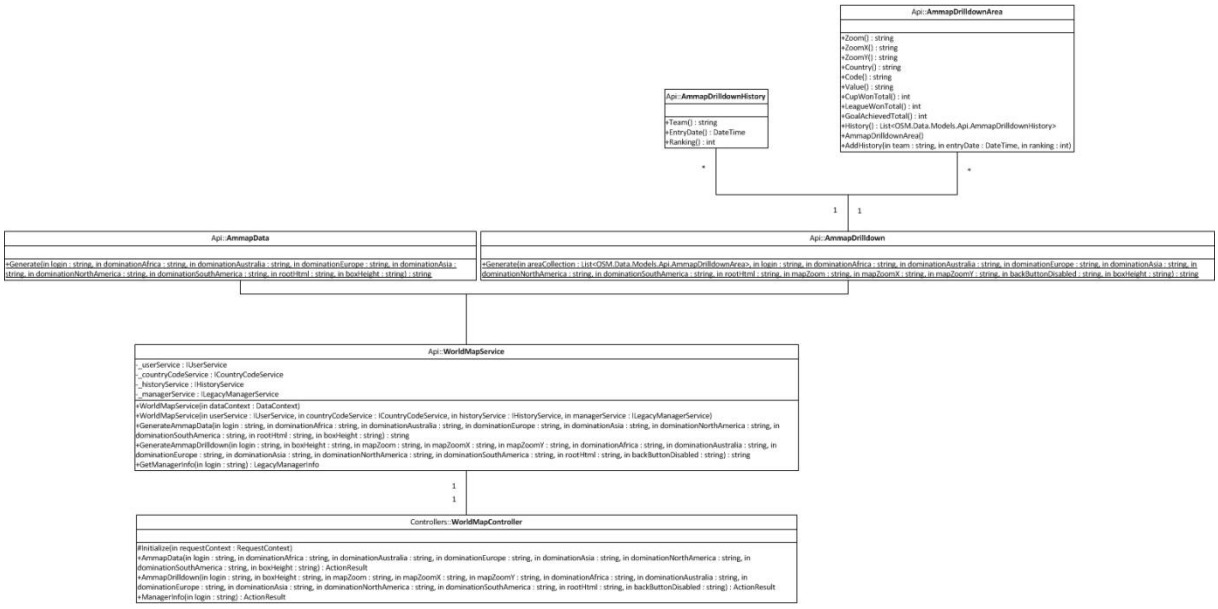
Translations



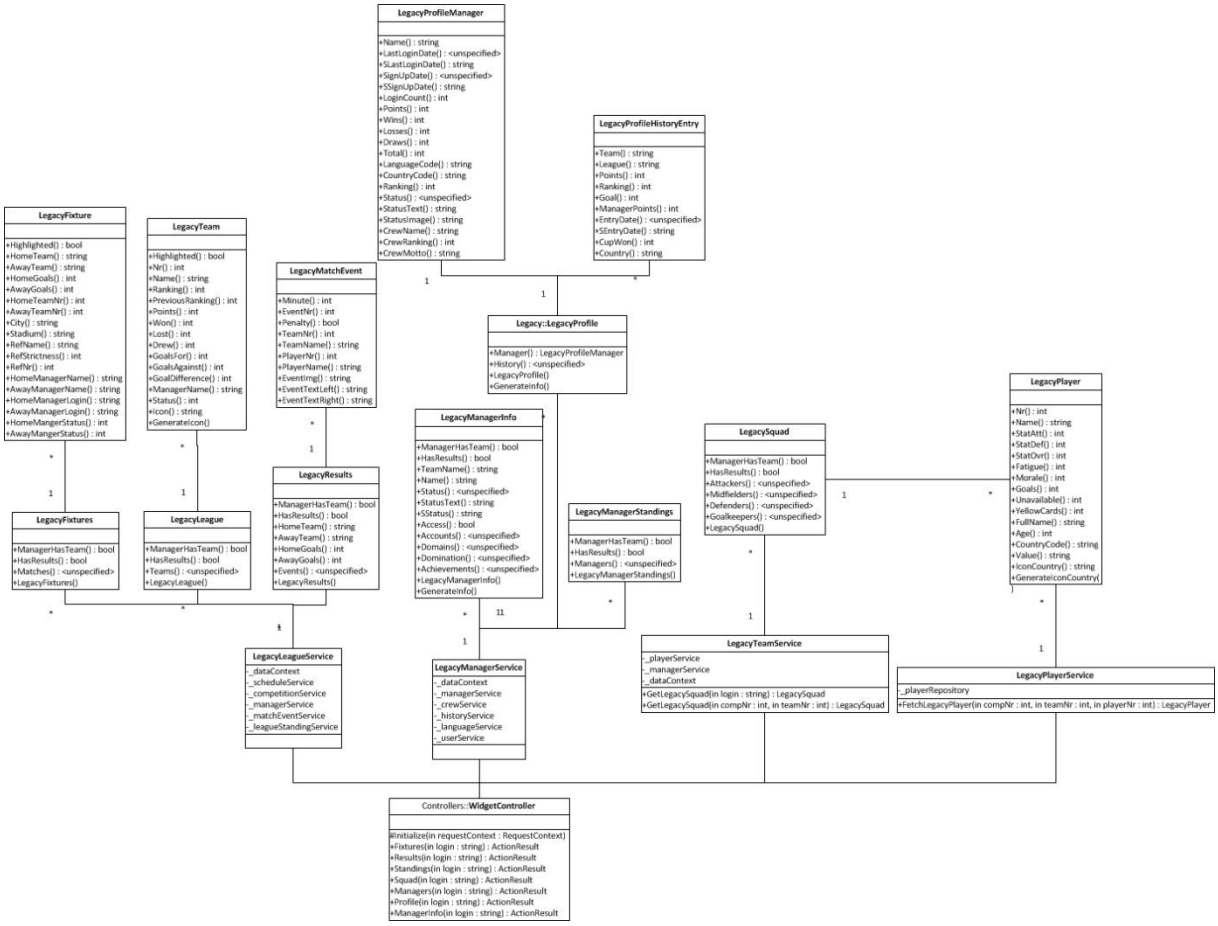
Rss



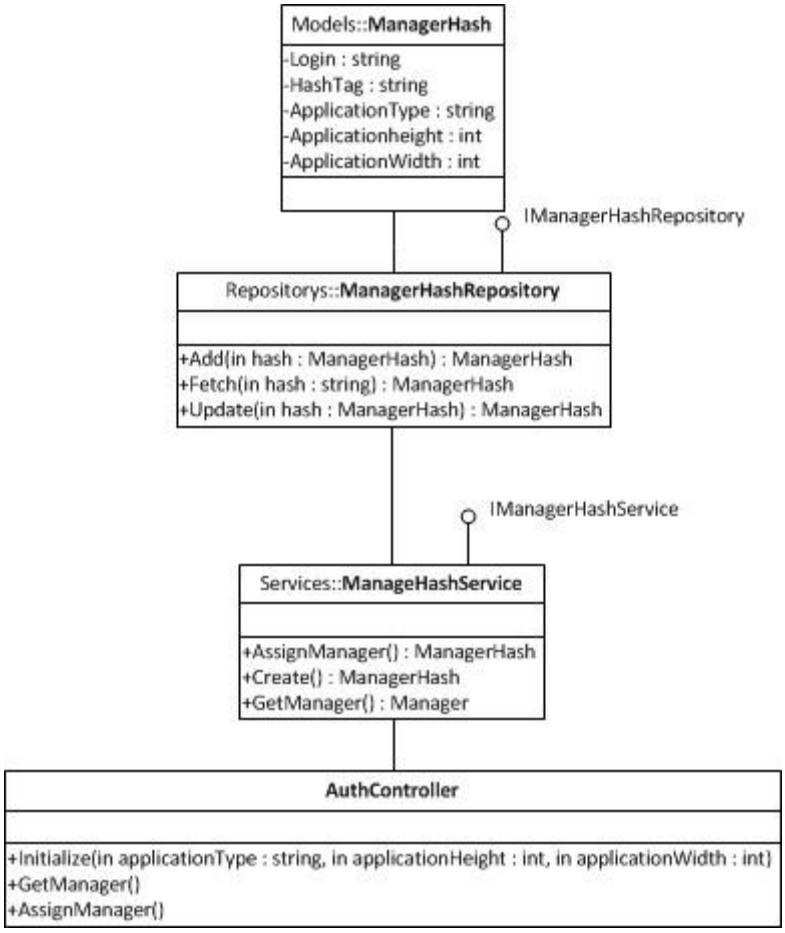
Worldmap



Widget

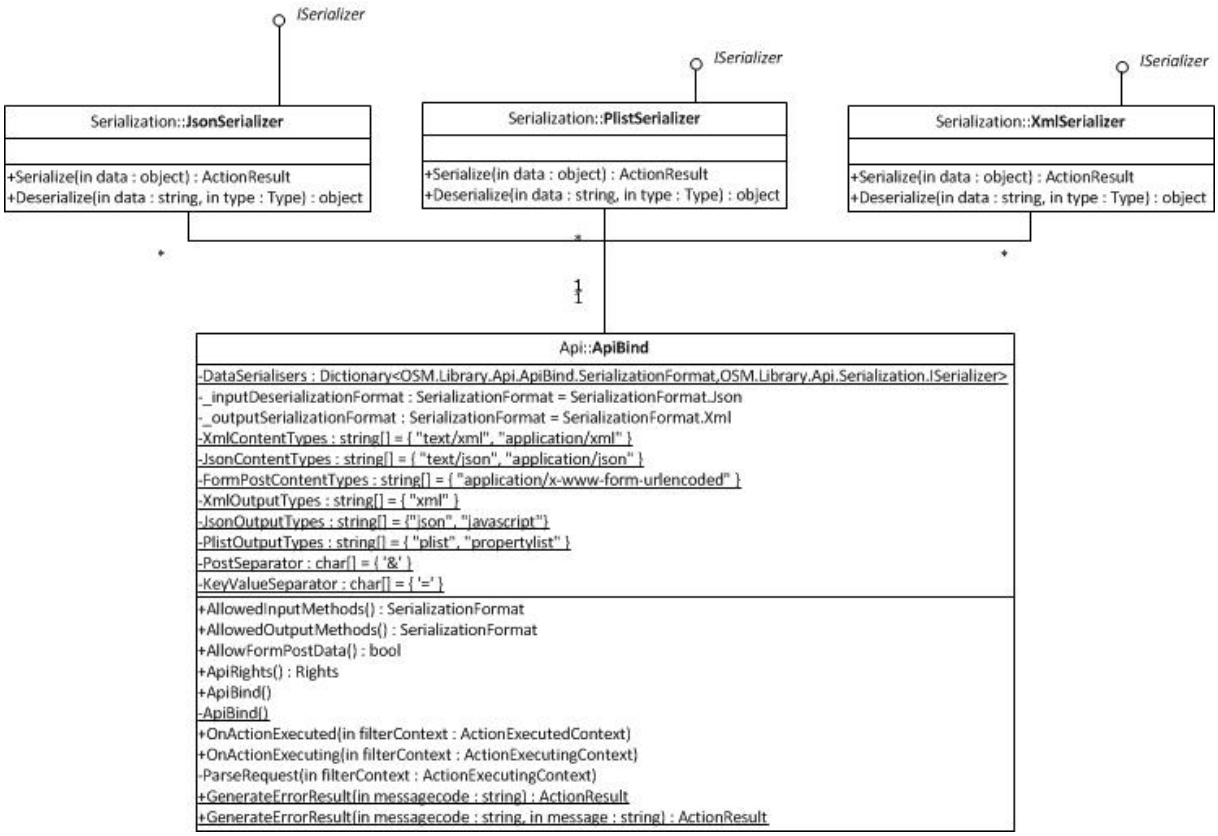

LegacyProfileManager
+Name() : string
+LastLoginDate() : <unspecified>
+SLastLoginDate() : string
+SignUpDate() : <unspecified>
+SSignUpDate() : string
+LoginCount() : int
+Points() : int
+Wins() : int
+Losses() : int
+Draws() : int
+Total() : int
+LanguageCode() : string
+CountryCode() : string
+Ranking() : int
+Status() : <unspecified>
+StatusText() : string
+StatusImage() : string
+CrewName() : string
+CrewRanking() : int
+CrewMotto() : string
LegacyProfileHistoryEntry
+Team() : string
+League() : string
+Points() : int
+Ranking() : int
+Goal() : int
+ManagerPoints() : int
+EntryDate() : <unspecified>
+SEntryDate() : string
+CupWon() : int
+Country() : string
LegacyFixture
+Highlighted() : bool
+HomeTeam() : string
+AwayTeam() : string
+HomeGoals() : int
+AwayGoals() : int
+HomeTeamNr() : int
+AwayTeamNr() : int
+City() : string
+Stadium() : string
+RefName() : string
+RefStrictness() : int
+RefNr() : int
+HomeManagerName() : string
+AwayManagerName() : string
+HomeManagerLogin() : string
+AwayManagerLogin() : string
+HomeMangerStatus() : int
+AwayMangerStatus() : int
LegacyTeam
+Highlighted() : bool
+Nr() : int
+Name() : string
+Ranking() : int
+PreviousRanking() : int
+Points() : int
+Won() : int
+Lost() : int
+Drew() : int
+GoalsFor() : int
+GoalsAgainst() : int
+GoalDifference() : int
+ManagerName() : string
+Status() : int
+Icon() : string
+GenerateIcon()
LegacyMatchEvent
+Minute() : int
+EventNr() : int
+Penalty() : bool
+TeamNr() : int
+TeamName() : string
+PlayerNr() : int
+PlayerName() : string
+EventImg() : string
+EventTextLeft() : string
+EventTextRight() : string
Legacy::LegacyProfile
+Manager() : LegacyProfileManager
+History() : <unspecified>
+LegacyProfile()
+GenerateInfo()
LegacyManagerInfo
+ManagerHasTeam() : bool
+HasResults() : bool
+TeamName() : string
+Name() : string
+Status() : <unspecified>
+StatusText() : string
+SStatus() : string
+Access() : bool
+Accounts() : <unspecified>
+Domains() : <unspecified>
+Domination() : <unspecified>
+Achievements() : <unspecified>
+LegacyManagerInfo()
+GenerateInfo()
LegacyResults
+ManagerHasTeam() : bool
+HasResults() : bool
+HomeTeam() : string
+AwayTeam() : string
+HomeGoals() : int
+AwayGoals() : int
+Events() : <unspecified>
+LegacyResults()
LegacyManagerStandings
+ManagerHasTeam() : bool
+HasResults() : bool
+Managers() : <unspecified>
+LegacyManagerStandings()
LegacySquad
+ManagerHasTeam() : bool
+HasResults() : bool
+Attackers() : <unspecified>
+Midfielders() : <unspecified>
+Defenders() : <unspecified>
+Goalkeepers() : <unspecified>
+LegacySquad()
LegacyPlayer
+Nr() : int
+Name() : string
+StatAtt() : int
+StatDef() : int
+StatOvr() : int
+Fatigue() : int
+Morale() : int
+Goals() : int
+Unavailable() : int
+YellowCards() : int
+FullName() : string
+Age() : int
+CountryCode() : string
+Value() : string
+IconCountry() : string
+GenerateIconCountry()
LegacyFixtures
+ManagerHasTeam() : bool
+HasResults() : bool
+Matches() : <unspecified>
+LegacyFixtures()
LegacyLeague
+ManagerHasTeam() : bool
+HasResults() : bool
+Teams() : <unspecified>
+LegacyLeague()
LegacyLeagueService
-_dataContext
-_scheduleService
-_competitionService
-_managerService
-_matchEventService
-_leagueStandingService
LegacyManagerService
-_dataContext
-_managerService
-_crewService
-_historyService
-_languageService
-_userService
LegacyTeamService
-_playerService
-_managerService
-_dataContext
+GetLegacySquad(in login : string) : LegacySquad
+GetLegacySquad(in compNr : int, in teamNr : int) : LegacySquad
LegacyPlayerService
-_playerRepository
+FetchLegacyPlayer(in compNr : int, in teamNr : int, in playerNr : int) : LegacyPlayer
Controllers::WidgetController
#Initialize(in requestContext : RequestContext)
+Fixtures(in login : string) : ActionResult
+Results(in login : string) : ActionResult
+Standings(in login : string) : ActionResult
+Squad(in login : string) : ActionResult
+Managers(in login : string) : ActionResult
+Profile(in login : string) : ActionResult
+ManagerInfo(in login : string) : ActionResult

Auth

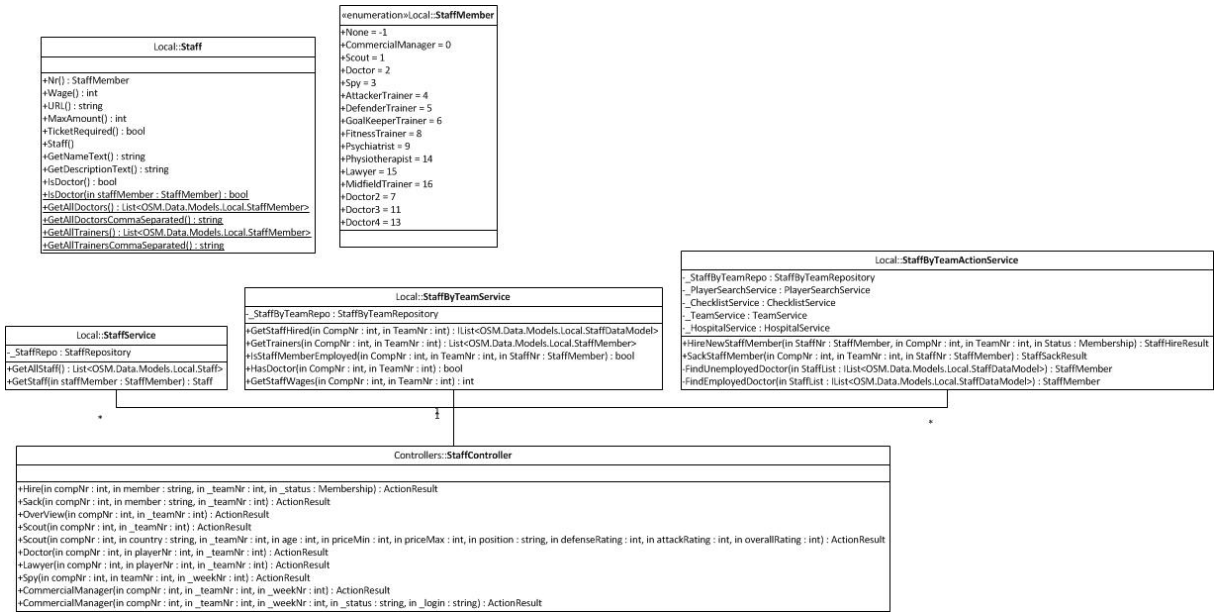

Models::ManagerHash
-Login : string
-HashTag : string
-ApplicationType : string
-Applicationheight : int
-ApplicationWidth : int
IManagerHashRepository
Repositorys::ManagerHashRepository
+Add(in hash : ManagerHash) : ManagerHash
+Fetch(in hash : string) : ManagerHash
+Update(in hash : ManagerHash) : ManagerHash
IManagerHashService
Services::ManageHashService
+AssignManager() : ManagerHash
+Create() : ManagerHash
+GetManager() : Manager
AuthController
+Initialize(in applicationType : string, in applicationHeight : int, in applicationWidth : int)
+GetManager()
+AssignManager()

5.2 Overige Functionaliteiten

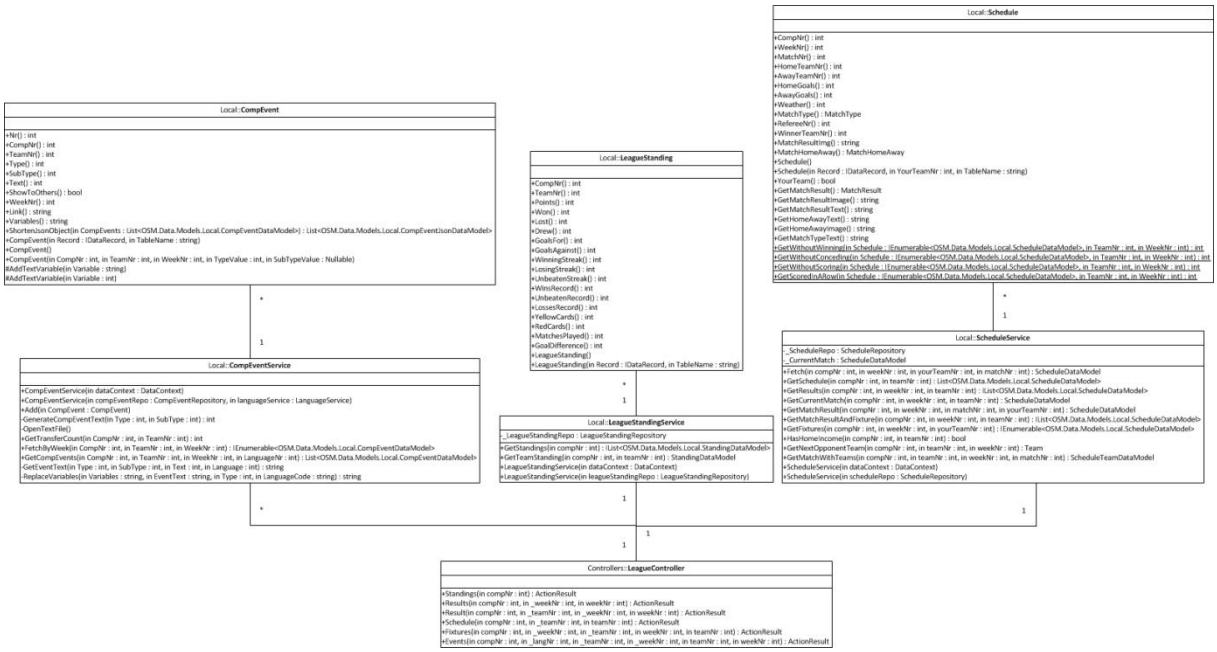
ApiBind



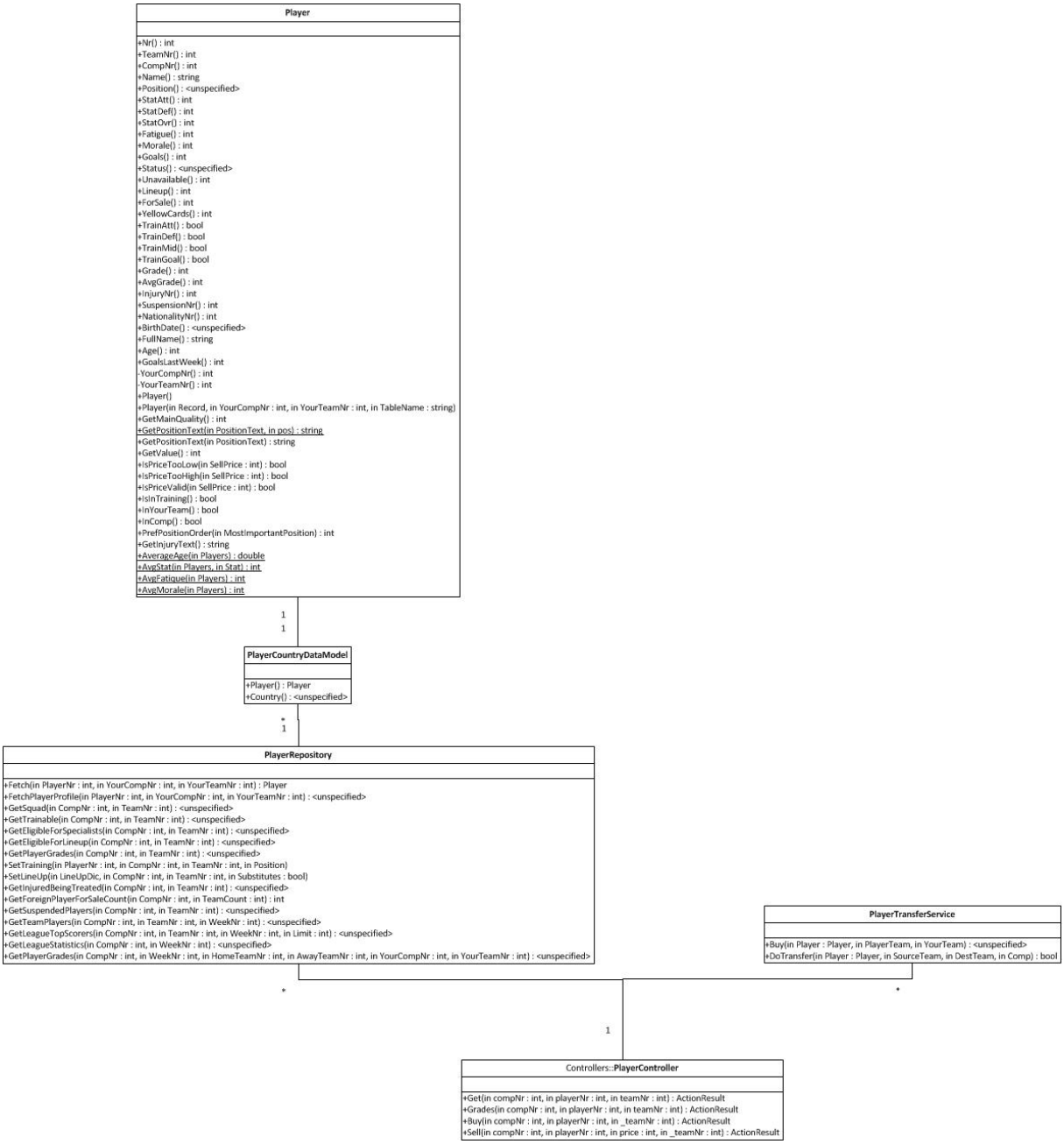
Staff



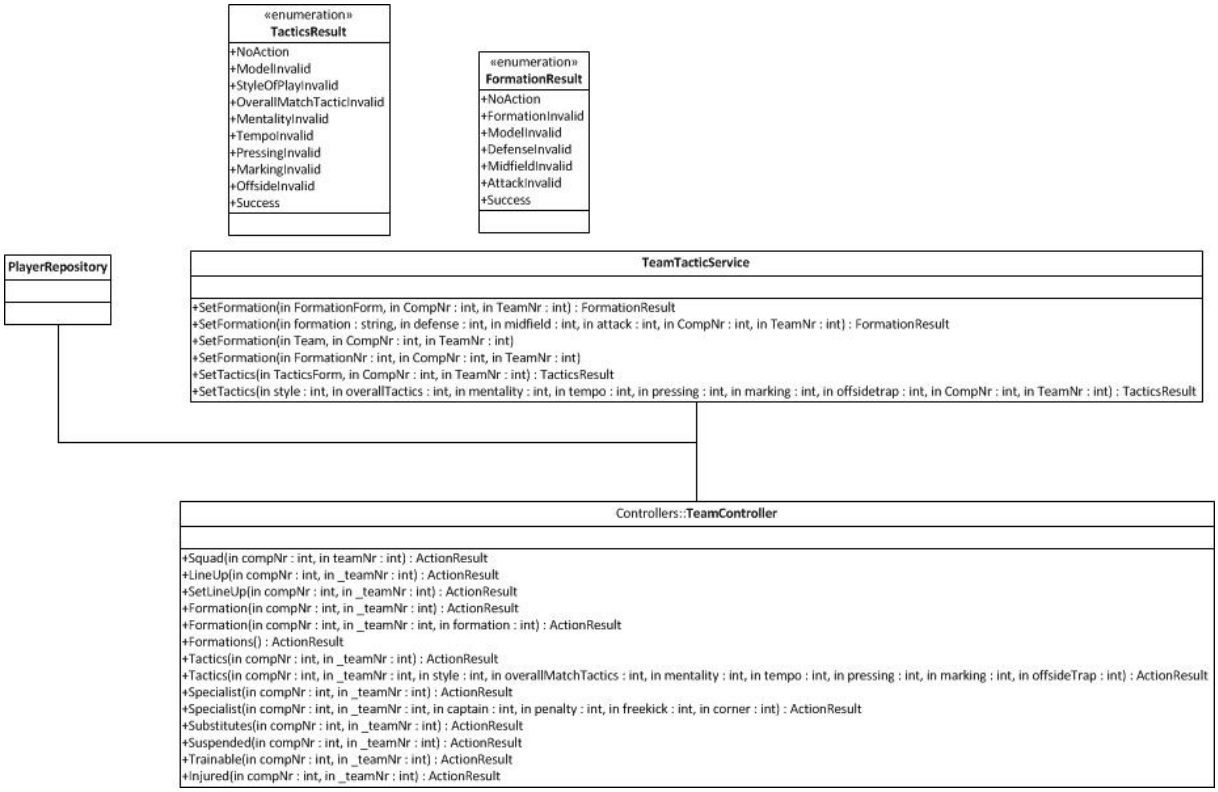
League


Local::CompEvent
+Nr() : int
+CompNr() : int
+TeamNr() : int
+Type() : int
+SubType() : int
+Text() : int
+ShowToOthers() : bool
+WeekNr() : int
+Link() : string
+Variables() : string
+ShortenJsonObject(in CompEvents : List<OSM.Data.Models.Local.CompEventDataModel>) : List<OSM.Data.Models.Local.CompEventJsonDataModel>
+CompEvent(in Record : IDataRecord, in TableName : string)
+CompEvent()
+CompEvent(in CompNr : int, in TeamNr : int, in WeekNr : int, in TypeValue : int, in SubTypeValue : Nullable)
#AddTextVariable(in Variable : string)
#AddTextVariable(in Variable : int)
Local::LeagueStanding
+CompNr() : int
+TeamNr() : int
+Points() : int
+Won() : int
+Lost() : int
+Drew() : int
+GoalsFor() : int
+GoalsAgainst() : int
+WinningStreak() : int
+LosingStreak() : int
+UnbeatenStreak() : int
+WinsRecord() : int
+UnbeatenRecord() : int
+LossesRecord() : int
+YellowCards() : int
+RedCards() : int
+MatchesPlayed() : int
+GoalDifference() : int
+LeagueStanding()
+LeagueStanding(in Record : IDataRecord, in TableName : string)
Local::Schedule
+CompNr() : int
+WeekNr() : int
+MatchNr() : int
+HomeTeamNr() : int
+AwayTeamNr() : int
+HomeGoals() : int
+AwayGoals() : int
+Weather() : int
+MatchType() : MatchType
+RefereeNr() : int
+WinnerTeamNr() : int
+MatchResultImg() : string
+MatchHomeAway() : MatchHomeAway
+Schedule()
+Schedule(in Record : IDataRecord, in YourTeamNr : int, in TableName : string)
+YourTeam() : bool
+GetMatchResult() : MatchResult
+GetMatchResultImage() : string
+GetMatchResultText() : string
+GetHomeAwayText() : string
+GetHomeAwayImage() : string
+GetMatchTypeText() : string
+GetWithoutWinning(in Schedule : IEnumerable<OSM.Data.Models.Local.ScheduleDataModel>, in TeamNr : int, in WeekNr : int) : int
+GetWithoutConceding(in Schedule : IEnumerable<OSM.Data.Models.Local.ScheduleDataModel>, in TeamNr : int, in WeekNr : int) : int
+GetWithoutScoring(in Schedule : IEnumerable<OSM.Data.Models.Local.ScheduleDataModel>, in TeamNr : int, in WeekNr : int) : int
+GetScoredInARow(in Schedule : IEnumerable<OSM.Data.Models.Local.ScheduleDataModel>, in TeamNr : int, in WeekNr : int) : int
Local::CompEventService
+CompEventService(in dataContext : DataContext)
+CompEventService(in compEventRepo : CompEventRepository, in languageService : LanguageService)
+Add(in CompEvent : CompEvent)
-GenerateCompEventText(in Type : int, in SubType : int) : int
-OpenTextFile()
+GetTransferCount(in CompNr : int, in TeamNr : int) : int
+FetchByWeek(in CompNr : int, in TeamNr : int, in WeekNr : int) : IEnumerable<OSM.Data.Models.Local.CompEventDataModel>
+GetCompEvents(in CompNr : int, in TeamNr : int, in WeekNr : int, in LanguageNr : int) : List<OSM.Data.Models.Local.CompEventDataModel>
-GetEventText(in Type : int, in SubType : int, in Text : int, in Language : int) : string
-ReplaceVariables(in Variables : string, in EventText : string, in Type : int, in LanguageCode : string) : string
Local::LeagueStandingService
-_LeagueStandingRepo : LeagueStandingRepository
+GetStandings(in compNr : int) : IList<OSM.Data.Models.Local.StandingDataModel>
+GetTeamStanding(in compNr : int, in teamNr : int) : StandingDataModel
+LeagueStandingService(in dataContext : DataContext)
+LeagueStandingService(in leagueStandingRepo : LeagueStandingRepository)
Local::ScheduleService
-_ScheduleRepo : ScheduleRepository
-_CurrentMatch : ScheduleDataModel
+Fetch(in compNr : int, in weekNr : int, in yourTeamNr : int, in matchNr : int) : ScheduleDataModel
+GetSchedule(in compNr : int, in teamNr : int) : List<OSM.Data.Models.Local.ScheduleDataModel>
+GetResults(in compNr : int, in weekNr : int, in teamNr : int) : IList<OSM.Data.Models.Local.ScheduleDataModel>
+GetCurrentMatch(in compNr : int, in weekNr : int, in teamNr : int) : ScheduleDataModel
+GetMatchResult(in compNr : int, in weekNr : int, in matchNr : int, in yourTeamNr : int) : ScheduleDataModel
+GetMatchResultAndFixture(in compNr : int, in weekNr : int, in teamNr : int) : IList<OSM.Data.Models.Local.ScheduleDataModel>
+GetFixtures(in compNr : int, in weekNr : int, in yourTeamNr : int) : IEnumerable<OSM.Data.Models.Local.ScheduleDataModel>
+HasHomeIncome(in compNr : int, in teamNr : int) : bool
+GetNextOpponentTeam(in compNr : int, in teamNr : int, in weekNr : int) : Team
+GetMatchWithTeams(in compNr : int, in teamNr : int, in weekNr : int, in matchNr : int) : ScheduleTeamDataModel
+ScheduleService(in dataContext : DataContext)
+ScheduleService(in scheduleRepo : ScheduleRepository)
Controllers::LeagueController
+Standings(in compNr : int) : ActionResult
+Results(in compNr : int, in _weekNr : int, in weekNr : int) : ActionResult
+Result(in compNr : int, in _teamNr : int, in _weekNr : int, in weekNr : int) : ActionResult
+Schedule(in compNr : int, in _teamNr : int, in teamNr : int) : ActionResult
+Fixtures(in compNr : int, in _weekNr : int, in _teamNr : int, in weekNr : int, in teamNr : int) : ActionResult
+Events(in compNr : int, in _langNr : int, in _teamNr : int, in _weekNr : int, in teamNr : int, in weekNr : int) : ActionResult

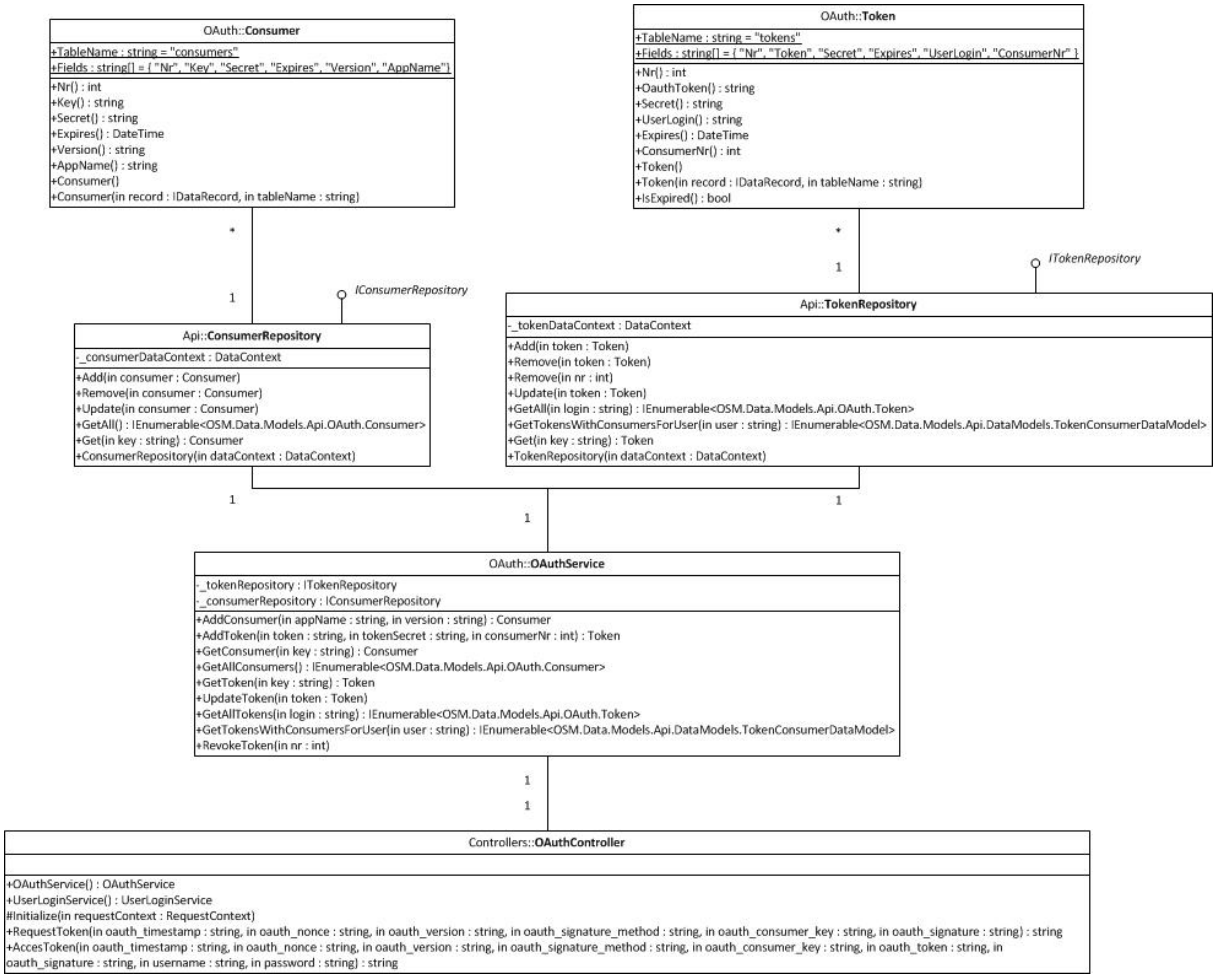
Player


Player
+Nr() : int
+TeamNr() : int
+CompNr() : int
+Name() : string
+Position() : <unspecified>
+StatAtt() : int
+StatDef() : int
+StatOvr() : int
+Fatigue() : int
+Morale() : int
+Goals() : int
+Status() : <unspecified>
+Unavailable() : int
+Lineup() : int
+ForSale() : int
+YellowCards() : int
+TrainAtt() : bool
+TrainDef() : bool
+TrainMid() : bool
+TrainGoal() : bool
+Grade() : int
+AvgGrade() : int
+InjuryNr() : int
+SuspensionNr() : int
+NationalityNr() : int
+BirthDate() : <unspecified>
+FullName() : string
+Age() : int
+GoalsLastWeek() : int
-YourCompNr() : int
-YourTeamNr() : int
+Player()
+Player(in Record, in YourCompNr : int, in YourTeamNr : int, in TableName : string)
+GetMainQuality() : int
+GetPositionText(in PositionText, in pos) : string
+GetPositionText(in PositionText) : string
+GetValue() : int
+IsPriceTooLow(in SellPrice : int) : bool
+IsPriceTooHigh(in SellPrice : int) : bool
+IsPriceValid(in SellPrice : int) : bool
+IsInTraining() : bool
+InYourTeam() : bool
+InComp() : bool
+PrefPositionOrder(in MostImportantPosition) : int
+GetInjuryText() : string
+AverageAge(in Players) : double
+AvgStat(in Players, in Stat) : int
+AvgFatigue(in Players) : int
+AvgMorale(in Players) : int
1
1
PlayerCountryDataModel
+Player() : Player
+Country() : <unspecified>
*
1
PlayerRepository
+Fetch(in PlayerNr : int, in YourCompNr : int, in YourTeamNr : int) : Player
+FetchPlayerProfile(in PlayerNr : int, in YourCompNr : int, in YourTeamNr : int) : <unspecified>
+GetSquad(in CompNr : int, in TeamNr : int) : <unspecified>
+GetTrainable(in CompNr : int, in TeamNr : int) : <unspecified>
+GetEligibleForSpecialists(in CompNr : int, in TeamNr : int) : <unspecified>
+GetEligibleForLineup(in CompNr : int, in TeamNr : int) : <unspecified>
+GetPlayerGrades(in CompNr : int, in TeamNr : int) : <unspecified>
+SetTraining(in PlayerNr : int, in CompNr : int, in TeamNr : int, in Position)
+SetLineUp(in LineUpDic, in CompNr : int, in TeamNr : int, in Substitutes : bool)
+GetInjuredBeingTreated(in CompNr : int, in TeamNr : int) : <unspecified>
+GetForeignPlayerForSaleCount(in CompNr : int, in TeamCount : int) : int
+GetSuspendedPlayers(in CompNr : int, in TeamNr : int) : <unspecified>
+GetTeamPlayers(in CompNr : int, in TeamNr : int, in WeekNr : int) : <unspecified>
+GetLeagueTopScorers(in CompNr : int, in TeamNr : int, in WeekNr : int, in Limit : int) : <unspecified>
+GetLeagueStatistics(in CompNr : int, in WeekNr : int) : <unspecified>
+GetPlayerGrades(in CompNr : int, in WeekNr : int, in HomeTeamNr : int, in AwayTeamNr : int, in YourCompNr : int, in YourTeamNr : int) : <unspecified>
PlayerTransferService
+Buy(in Player : Player, in PlayerTeam, in YourTeam) : <unspecified>
+DoTransfer(in Player : Player, in SourceTeam, in DestTeam, in Comp) : bool
*
*
1
Controllers::PlayerController
+Get(in compNr : int, in playerNr : int, in teamNr : int) : ActionResult
+Grades(in compNr : int, in playerNr : int, in teamNr : int) : ActionResult
+Buy(in compNr : int, in playerNr : int, in _teamNr : int) : ActionResult
+Sell(in compNr : int, in playerNr : int, in price : int, in _teamNr : int) : ActionResult

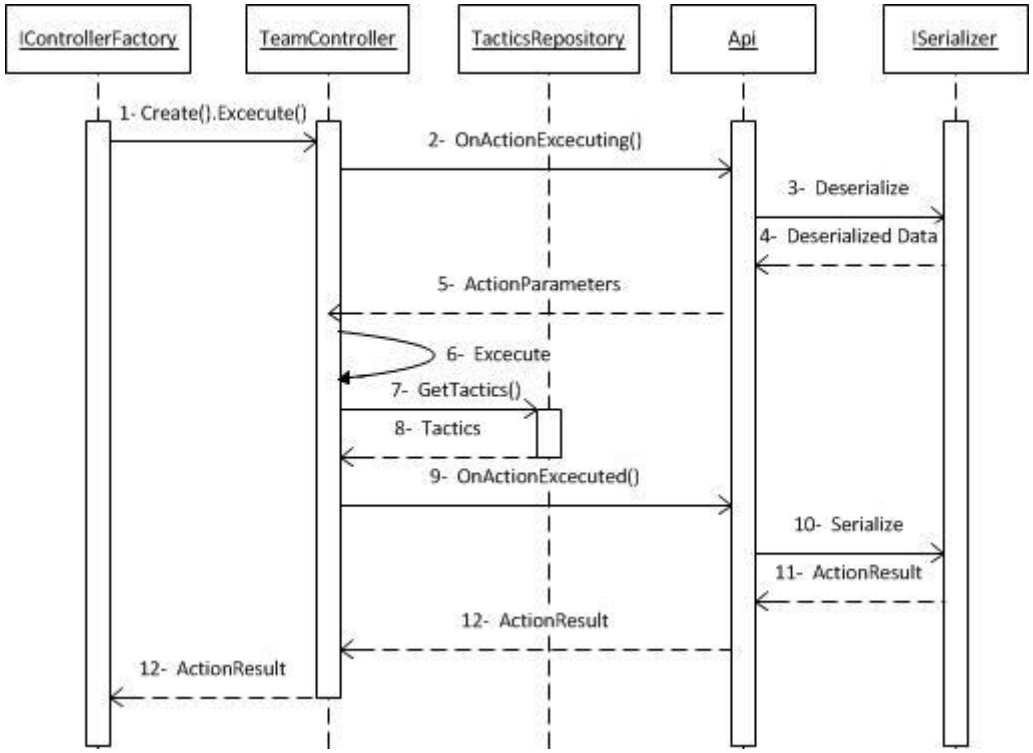
Team


«enumeration»
TacticsResult
+NoAction
+ModelInvalid
+StyleOfPlayInvalid
+OverallMatchTacticInvalid
+MentalityInvalid
+TempoInvalid
+PressingInvalid
+MarkingInvalid
+OffsideInvalid
+Success
«enumeration»
FormationResult
+NoAction
+FormationInvalid
+ModelInvalid
+DefenseInvalid
+MidfieldInvalid
+AttackInvalid
+Success
PlayerRepository
TeamTacticService
+SetFormation(in FormationForm, in CompNr : int, in TeamNr : int) : FormationResult
+SetFormation(in formation : string, in defense : int, in midfield : int, in attack : int, in CompNr : int, in TeamNr : int) : FormationResult
+SetFormation(in Team, in CompNr : int, in TeamNr : int)
+SetFormation(in FormationNr : int, in CompNr : int, in TeamNr : int)
+SetTactics(in TacticsForm, in CompNr : int, in TeamNr : int) : TacticsResult
+SetTactics(in style : int, in overallTactics : int, in mentality : int, in tempo : int, in pressing : int, in marking : int, in offsidetrap : int, in CompNr : int, in TeamNr : int) : TacticsResult
Controllers::TeamController
+Squad(in compNr : int, in teamNr : int) : ActionResult
+LineUp(in compNr : int, in _teamNr : int) : ActionResult
+SetLineUp(in compNr : int, in _teamNr : int) : ActionResult
+Formation(in compNr : int, in _teamNr : int) : ActionResult
+Formation(in compNr : int, in _teamNr : int, in formation : int) : ActionResult
+Formations() : ActionResult
+Tactics(in compNr : int, in _teamNr : int) : ActionResult
+Tactics(in compNr : int, in _teamNr : int, in style : int, in overallMatchTactics : int, in mentality : int, in tempo : int, in pressing : int, in marking : int, in offsideTrap : int) : ActionResult
+Specialist(in compNr : int, in _teamNr : int) : ActionResult
+Specialist(in compNr : int, in _teamNr : int, in captain : int, in penalty : int, in freekick : int, in corner : int) : ActionResult
+Substitutes(in compNr : int, in _teamNr : int) : ActionResult
+Suspended(in compNr : int, in _teamNr : int) : ActionResult
+Trainable(in compNr : int, in _teamNr : int) : ActionResult
+Injured(in compNr : int, in _teamNr : int) : ActionResult

5.3 OAuth


OAuth::Consumer
+TableName : string = "consumers"
+Fields : string[] = { "Nr", "Key", "Secret", "Expires", "Version", "AppName"}
+Nr() : int
+Key() : string
+Secret() : string
+Expires() : DateTime
+Version() : string
+AppName() : string
+Consumer()
+Consumer(in record : IDataRecord, in tableName : string)
OAuth::Token
+TableName : string = "tokens"
+Fields : string[] = { "Nr", "Token", "Secret", "Expires", "UserLogin", "ConsumerNr" }
+Nr() : int
+OauthToken() : string
+Secret() : string
+UserLogin() : string
+Expires() : DateTime
+ConsumerNr() : int
+Token()
+Token(in record : IDataRecord, in tableName : string)
+IsExpired() : bool
IConsumerRepository
ITokenRepository
Api::ConsumerRepository
-_consumerDataContext : DataContext
+Add(in consumer : Consumer)
+Remove(in consumer : Consumer)
+Update(in consumer : Consumer)
+GetAll() : IEnumerable<OSM.Data.Models.Api.OAuth.Consumer>
+Get(in key : string) : Consumer
+ConsumerRepository(in dataContext : DataContext)
Api::TokenRepository
-_tokenDataContext : DataContext
+Add(in token : Token)
+Remove(in token : Token)
+Remove(in nr : int)
+Update(in token : Token)
+GetAll(in login : string) : IEnumerable<OSM.Data.Models.Api.OAuth.Token>
+GetTokensWithConsumersForUser(in user : string) : IEnumerable<OSM.Data.Models.Api.DataModels.TokenConsumerDataModel>
+Get(in key : string) : Token
+TokenRepository(in dataContext : DataContext)
OAuth::OAuthService
-_tokenRepository : ITokenRepository
-_consumerRepository : IConsumerRepository
+AddConsumer(in appName : string, in version : string) : Consumer
+AddToken(in token : string, in tokenSecret : string, in consumerNr : int) : Token
+GetConsumer(in key : string) : Consumer
+GetAllConsumers() : IEnumerable<OSM.Data.Models.Api.OAuth.Consumer>
+GetToken(in key : string) : Token
+UpdateToken(in token : Token)
+GetAllTokens(in login : string) : IEnumerable<OSM.Data.Models.Api.OAuth.Token>
+GetTokensWithConsumersForUser(in user : string) : IEnumerable<OSM.Data.Models.Api.DataModels.TokenConsumerDataModel>
+RevokeToken(in nr : int)
Controllers::OAuthController
+OAuthService() : OAuthService
+UserLoginService() : UserLoginService
#Initialize(in requestContext : RequestContext)
+RequestToken(in oauth_timestamp : string, in oauth_nonce : string, in oauth_version : string, in oauth_signature_method : string, in oauth_consumer_key : string, in oauth_signature : string) : string
+AccesToken(in oauth_timestamp : string, in oauth_nonce : string, in oauth_version : string, in oauth_signature_method : string, in oauth_consumer_key : string, in oauth_token : string, in oauth_signature : string, in username : string, in password : string) : string

6 Sequentie diagram



1. Intern wordt door MVC de juiste controller uitgevoerd en aangeroepen.
2. Er wordt een functie aangeroepen voor de daadwerkelijke controller functie wordt uitgevoerd.
3. Er wordt gekeken of de gebruiker data heeft verstuurd en deze wordt indien nodig gedeserialiseerd.
6. De controllerer actie wordt intern uitgevoerd.
7. De voorbeeld functionaliteit om de tactiek van een team op te halen wordt vanuit de controller aangeroepen.
9. Na het uitvoeren van de actie wordt automatisch een post actie uitgevoerd.
10. De data die terug gegeven wordt aan de gebruiker wordt naar het gewenste formaat geserialiseerd.

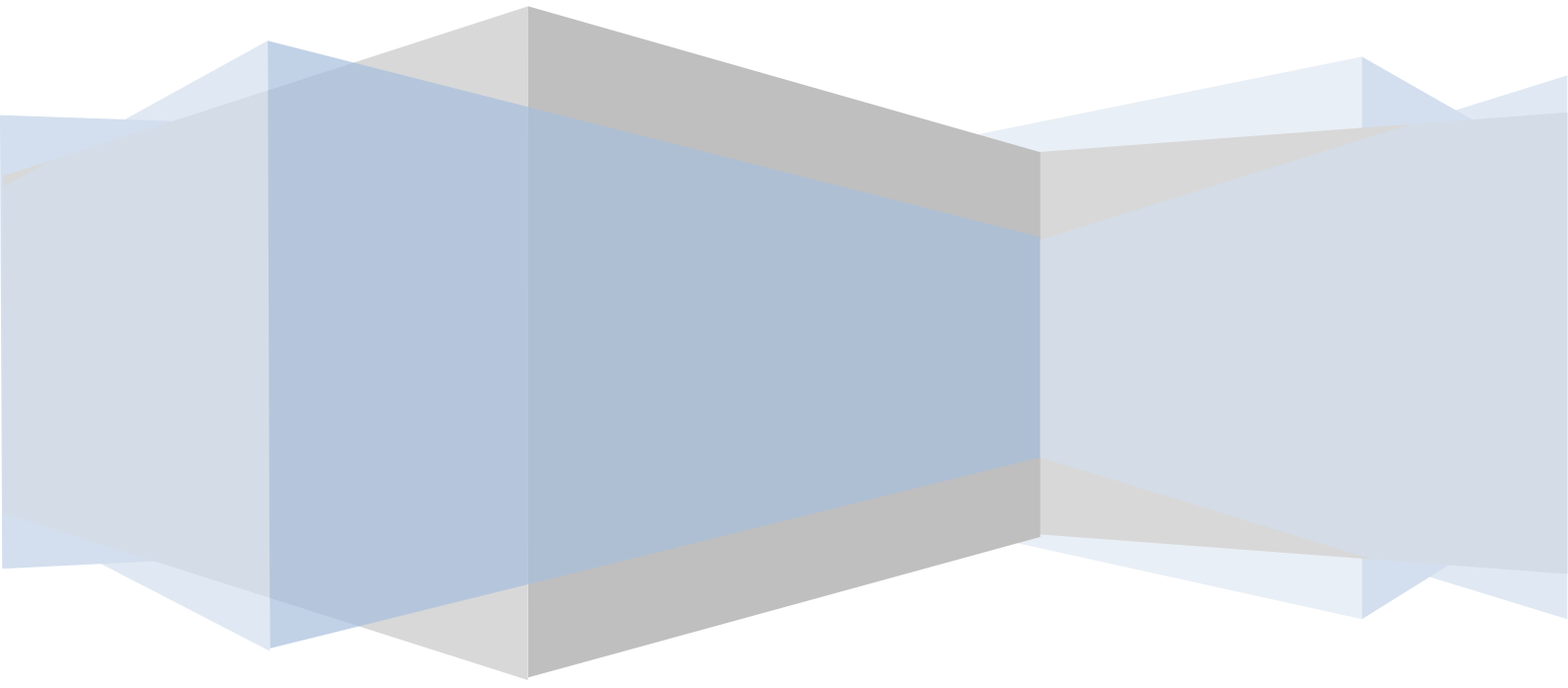
GameBasics

Testplan

Versie 1.0

Dit document omschrijft op welke manier de API getest zal worden.

Robin Scholtes
18-1-2011



1 Unit Tests

Tijdens dit project zal er gemaakt worden van Test Driven Development. Dit betekent dat voordat functionaliteiten geïmplementeerd worden eerst de Unit tests opgesteld worden. Binnen Visual Studio kan dit groot en deels geautomatiseerd. Hieronder wordt stapsgewijs de procedure besproken die doorlopen moet worden om van een ontwerp naar productie te gaan

1 Classes aanmaken

Alle classes die in het ontwerp zijn opgenomen dienen aangemaakt te worden.

2 Methodes aanmaken

Alle methodes worden aangemaakt met een standaard NotImplementedException

3 Tests Genereren

Binnen Visual Studio kunnen Unit tests en test omgevingen standaard voor functies gegenereerd worden. Deze dienen gegenereerd te worden voor:

- Alle services die gemaakt worden.
- Alle ApiBind gerelateerde functionaliteiten.

4 Dependencys Faken

Ieder object wat van unit tests voorzien wordt kan Dependencys hebben naar andere objecten. Als deze objecten door C# gegenereerd zijn of data interactie hebben moeten deze van een FakeImplementatie voorzien worden. Dit betekent dat er minimaal de volgende FakeImplementaties zullen zijn:

- FakeServiceX
- FakeRepositoryX
- FakeHttpContext

5 Invullen test voorwaarden

Iedere test wordt verzien van minimaal twee testvoorwaarden. Dit betekent dat er niet alleen getest wordt of een waarde voldoet aan een waarde maar ook of het type correct is en evt. andere voorwaarden. Het streven is ervoor te zorgen dat een test aan meerdere voorwaarden moet voldoen om zo te zorgen dat de test zoveel mogelijk uitzondering kan afvangen.

6 Tests uitvoeren

Eerste testrun, alle tests MOETEN falen voordat er begonnen mag worden met de implementatie.

7 Functies implementeren

Op dit moment mogen de functies echt geïmplementeerd worden.

8 Tests uitvoeren

Tweede testrun, als een test niet succesvol is terug naar 7.

9 Refactoren.

Wanneer een test succesvol is en de functionaliteit goed is wordt deze herschreven naar code die goed genoeg is en gedocumenteerd genoeg om in productie genomen te worden.

10 Validatie

Na het refactoren wordt een derde testrun gedaan om te valideren dat alle functionaliteiten nogsteeds succesvol de unit test doorlopen.

2 Test Pagina

Er zal een webpagina gemaakt worden waarop er voorbeeld aanroepen van alle API Calls komen te staan. Deze pagina kan gebruikt worden om snel te kijken of functies werken en wat de uitvoer is zonder dat er in Visual Studio gewerkt hoeft te worden. Deze pagina is gemaakt voor zowel het testen door de programmeur als wel als voor andere mensen binnen de organisatie. Het doel is dan ook deze pagina door meerdere mensen te laten bekijken om te kijken of hun fouten kunnen opsporten.

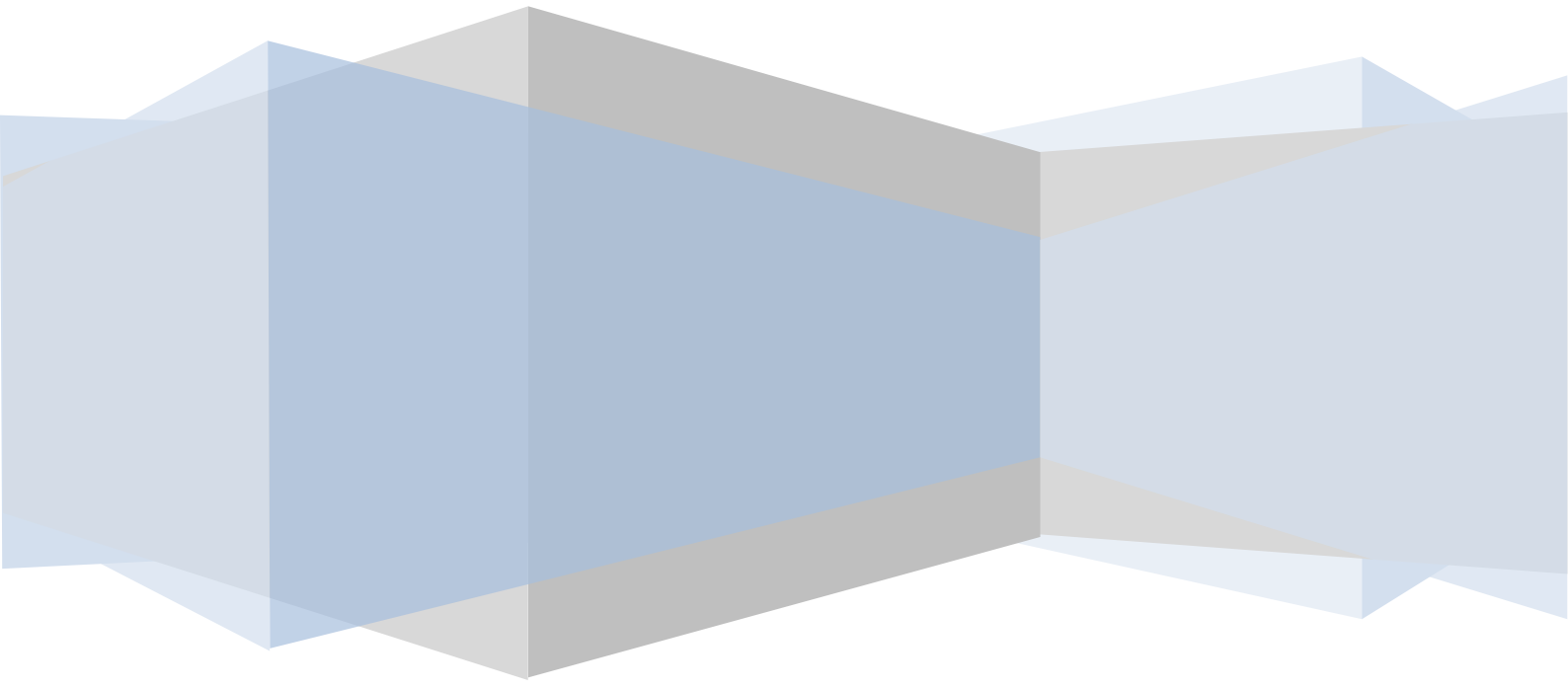
GameBasics

Construction Document

Versie 0.2

Dit document beschrijft de technische aspecten van de implementatie van de legacy API en de bijzonderheden tijdens deze iteratie van de Construction fase.

Robin Scholtes
2-2-2011



Inhoudsopgave

1 Inleiding

Dit document wordt geschreven met het doel om enkele specifieke technische zaken toe te lichten die in het ontwerp niet duidelijk gemaakt konden worden. Dit betekent dat dit document zich ook meer toe zal spitsen op de code.

1.1 Structuur document

Als eerste wordt hieronder de implementatie van de RSS feeds besproken. Vervolgens wordt er beschreven hoe de WorldMap API's zijn geïmplementeerd en tenslotte wordt er beschreven hoe de Widget API's zijn geïmplementeerd. Hierbij zal vooral aandacht geschonken worden aan oplossingen die onduidelijk kunnen zijn of technische complexiteit bevatten dat toelichting noodzakelijk is.

2 RSS Feeds

In de oude situatie waren er twee RSS feeds. Een van deze feeds gaf een X aantal berichten weer in een bepaalde taal. De andere RSS feed kon gebruikt worden om de resultaten van een bepaalde competitie op te vragen. Dit waren twee losse bestanden die grotendeels dezelfde functionaliteiten bevatten, namelijk het wegschrijven van de RSS. Binnen de nieuwe implementatie is er voor gekozen om een RSS Controller te maken waarbinnen alle RSS feeds opgevraagd kunnen worden. Dit zorgt er ook voor dat wanneer er nieuwe RSS feeds gemaakt worden deze hier toegevoegd kunnen worden. De RSS feeds zijn benaderbaar via de volgende URL's:

- http://osmapi.onlinefootballmanager.co.uk/rss/news
- http://osmapi.onlinefootballmanager.co.uk/rss/results

In het Elaboration rapport werd het ontwerp getoond van de API ActionFilter. Deze opzet was echter niet te gebruiken voor RSS omdat de data die naar de RSS feed gaat anders is dan de data die naar XML of JSON gaat. Hiervoor was de oplossing gekozen om een ActionResult te maken wat op moment van uitvoeren de data omzet naar een RSS feed. Dit ActionResult bestaat uit een Title/Description voor de feed en een List met RSSEntry's die stuk voor stuk een RSS entry beschrijven. Het opbouwen van de feed gebeurt op de volgende manier:

```
using (var writer = XmlWriter.Create(context.HttpContext.Response.OutputStream,
settings))
{
       // Begin structure
       writer.WriteStartElement("rss");
       writer.WriteAttributeString("version", "2.0");
       writer.WriteStartElement("channel");

       writer.WriteElementString("title", Title);
       writer.WriteElementString("description", Description);
       writer.WriteElementString("link",
       context.HttpContext.Request.Url.GetLeftPart(UriPartial.Authority));
}
```

Op deze wijze worden zowel entrys als de algemene feed informatie weggeschreven. Om er voor te zorgen dat de RSS Feed valide is moest er vooral gelet worden op de case van de variabelen, aangezien deze afwijkt van XML standaarden.

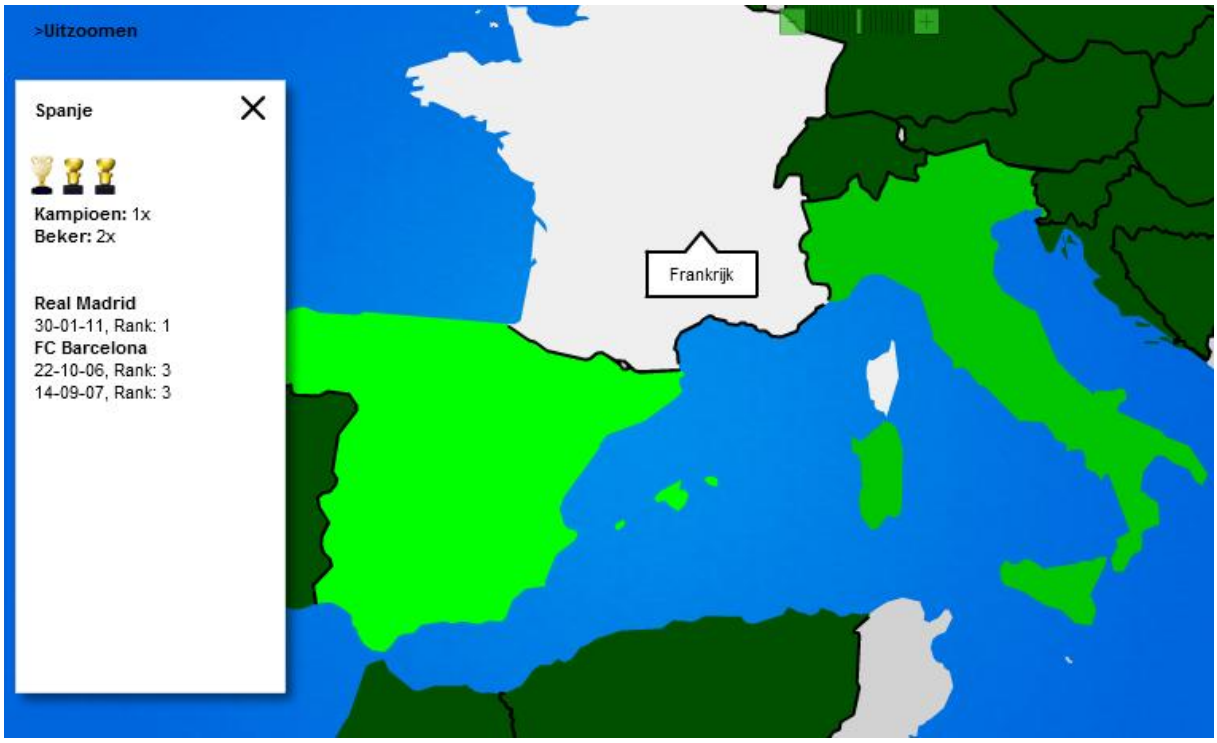
Wanneer de RSS feed voor het nieuws opgevraagd wordt en er specifiek 1 resultaat gevraagd wordt zal dit de output zijn:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>OnlineFootballManager.co.uk - The Football Manager</title>
    <description>The latest news of Online Football Manager, the world's
most fun football management game.</description>
    <link>http://osmapi.onlinefootballmanager.co.uk</link>
    <item>
      <title>Special offer</title>
      <description>Online Football Manager has a huge discount for you in
store. [url=/payment/payment.asp]When you order season tickets[/url] twice
(On the same day) with Allopass, you&amp;#39;ll receive double the amount
of season tickets with your second order!</description>
      <link>http://osmapi.onlinefootballmanager.co.uk</link>
```

```
      <pubDate>Tue, 15 Feb 2011 14:55:06 GMT</pubDate>
    </item>
  </channel>
</rss>
```

3 WorldMap

In het oude systeem werd er gebruik gemaakt van een WorldMap waarop te zien was in hoeveel landen je de doelstelling van het gekozen team behaalt hebt.



Deze WorldMap wordt getekend in flash door middel van een plug-in met de naam AMMap. Deze plug-in verwacht een bepaald formaat van XML, wat veel gebruikt maakt van XML Attributen. Omdat dit moeilijk na te maken is met Objecten is er bij dit element ook gekozen om de API ActionFilter niet te gebruiken en wederom de XML handmatig weg te schrijven. AMMap verwacht twee XML documenten , een voor een overzicht van de wereld met data van de continenten en een XML met data per land. De XML voor de continenten ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
  <map map_file="maps/continents.swf" zoom="92.0238%" zoom_x="5.68%"
zoom_y="3.72%">
    <areas>
      <area mc_name="africa" title="Africa&lt;br /&gt;%" zoom="250%"
zoom_x="-84.94%" zoom_y="-118.95%" url="!AMMAPDRILLDOWN" value=""/>
      <area value="100" />
    </areas>
  </map>
```

De XML voor de informatie per land ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
  <map map_file="maps/world.swf" zoom="%" zoom_x="%" zoom_y="%" tl_long="-
168.49" tl_lat="83.63" br_long="190.3" br_lat="-55.58">
    <areas>
      <area zoom="1043.4748%" zoom_x="-442.32%" zoom_y="-484.9%"
title="Spain" mc_name="ES" text_box_width="170" text_box_height=""
text_box_x="10" text_box_y="50" value = "100">
```

```
        <description>
          <![CDATA[
            <b></b><img src="/content/images/icons/worldmap/icon_champ.png"
width="15" height="25" hspace="0" vspace="0" align="left" /><img
src="/content/images/icons/worldmap/icon_cup.png" width="15" height="25"
hspace="20" vspace="0" align="left" /><img
src="/content/images/icons/worldmap/icon_cup.png" width="15" height="25"
hspace="40" vspace="0" align="left" /></p><br /><b></b><br /><br
/><b>Champion:</b><br /><b>Cup won:</b><br /><br /><b>FC Barcelona</b><br
/>22-10-2006, Rank: 3<br /><b>FC Barcelona</b><br />14-9-2007, Rank: 3<br
/><b>Real Madrid</b><br />30-1-2011, Rank: 1]]></description>
      </area>
    </areas>
    <movies>
      <movie file="home" x="15" y="18"
url="!/api/worldmap/ammap_data?login=Monkhe&boxheight=&mapzoom=250&mapzoomx
=-84.94&mapzoomy=-
118.95&dominationAfrica=&dominationAustralia=&dominationEurope=&dominationA
sia=&dominationNorthAmerica=&dominationSouthAmerica=">
      </movie>
    </movies>
    <labels>
      <label x="20" y="10"
url="!/api/worldmap/ammap_data?login=Monkhe&boxheight=&mapzoom=250&mapzoomx
=-84.94&mapzoomy=-
118.95&dominationAfrica=&dominationAustralia=&dominationEurope=&dominationA
sia=&dominationNorthAmerica=&dominationSouthAmerica=" remain="true"
color_hover="#CC0000" zoom_x="0%" zoom_y="0" zoom="100%">
        <text>
          <![CDATA[>
            <b>Back to World Map</b>
        </label>
      </labels>
    </map>
```

Bij het opbouwen van de geschiedenis per land is een grote optimalisatie uitgevoerd in de nieuwe versie ten opzichte van de oude versie. In de oude versie werden alle landen opgehaald, wat er ongeveer 250 zijn, vervolgens werd er per land een query naar de database gedaan om te kijken of de gebruiker hier een competitie heeft gespeeld en of hij zijn doelstelling gehaald heeft. Dit alleen resulteerde al in meer dan 250 query's naar de database. In de nieuwe versie worden er slechts twee query's naar de database gedaan. De eerste haalt de complete lijst met landen op en de tweede query haalt de complete geschiedenis van de gebruiker op. Met een techniek die binnen C# bekend staat als LINQ worden vervolgens de juiste records aan elkaar gekoppeld. Op deze manier wordt e database ontlast en wordt er meer verwerking van de data in de API zelf gedaan.

```
var countryHistoryCollection = historyCollection.Where(x => x.CountryNr ==
country.CountryNr);
```

Bij het opbouwen van de XML wordt ieder land ingevuld. Met bovenstaand commando wordt er in C# een selectie gedaan op de history met als voorwaarde dat het land van de history gelijk is aan het huidige land wat doorgelopen wordt. Met LINQ is niet alleen het groeperen en selecteren binnen collecties mogelijk, het is ook mogelijk om COUNT's te doen binnen collecties. Bij de history werd in de oude versie drie subquery's uitgevoerd die statistieken per land optelde. Dus binnen de 250 querys die al gedaan werden zaten ook nog verborgen subquerys.

Met LINQ wordt er op de history collectie een count uitgevoerd om te bepalen hoevaak een manager bijvoorbeeld kampioen is geworden in een bepaald land:

```
area.LeagueWonTotal = historyCollection.Count(x => (x.CountryNr == country.CountryNr
&& x.Ranking == 1 && x.ManagerPoints > 0));
```

Deze oplossing met LINQ zorgt voor minder belasting op de database en dat de pagina vele malen sneller wordt.

4 Widget

Het volgende punt waar in deze Construction Fase aan gewerkt is, is de ondersteuning om de Widget applicatie werkend te krijgen binnen de API. Deze widget bevat een kubus die rond kan draaien, met verschillende pagina's met data over een managers account en het team waarvan hij of zij op dat moment coach is.

Zoals te zien is, en al in de eerdere fases beschreven is kan een speler met deze widget zijn profiel bekijken, uitslagen, competitiestanden, speelschema's, teams en managers in de competitie bekijken. Deze functionaliteiten zitten ook in de website, en zullen uiteindelijk ook in de nieuw API komen.

Binnen C# bestaat een techniek die Reflection genoemd wordt. Deze techniek maakt het mogelijk om door een object heen te lopen en te kijken welke Public properties dit object bezit. Op deze manier kun je dus ook waardes van object A naar object B overzetten, zolang de veldnamen hetzelfde zijn. Deze techniek wordt ook gebruikt bij de API voor de widget. Er zijn models gemaakt in de Datalaag die specifiek voor deze widget zijn. Deze zijn geprefixed met Legacy, om er voor te zorgen dat er geen verwarring ontstaat. Om er voor te zorgen dat de Repositorys en de Service lagen niet te uitgebreid worden is er voor gekozen om van de Models die ook in de Website/API komen gebruik te maken en de data die hier in zit te "Kopiëren". In de gedeelde Library is een functie gemaakt die gebruik maakt van Generics om het type aan te geven waarnaar gekopieerd moet worden en Reflection om het daadwerkelijke kopiëren te realiseren.

```
var manager = _managerService.GetProfile(login, _dataContext.ServerNr);
var managerApi =
Toolbox.CopyObjectProperties<Data.Models.Api.LegacyManagerInfo>(manager));
```

Bovenstaand codevoorbeeld laat zien hoe deze functionaliteit gebruikt kan worden. Met deze aanroep worden alle waardes die zich in manager bevinden, naar een object van het type LegacyManagerInfo gekopieerd waarvan de naam van de variabele hetzelfde is. Hiermee worden grote stukken code versimpeld tot 1 regel. Velden die niet dezelfde naam hebben, of niet in een object voorkomen zullen met de hand alsnog moeten worden toegevoegd.

De eerder beschreven WorldMap en RSS feeds maakte geen gebruik van de API ActionFilter, de Widget API maakt dit wel. Dit is ook een van de redenen dat de data naar een nieuw object moet worden gekopieerd, zodat wanneer het object met de API geserialiseerd wordt, naar het goede formaat wordt omgezet wat begrepen kan worden bij de Widget.

```
[ApiBind(AllowedOutputMethods = ApiBind.SerializationFormat.Xml |
ApiBind.SerializationFormat.Json)]
public ActionResult ManagerInfo(string login)
{
	return new PlainActionResult(ManagerService.GetManagerInfo(login));
}
```

Hierboven is een van de implementaties weergegeven van de API Filter en een Widget functie bedoelt om de informatie van een manager op te halen. Het overzetten van de data gebeurt in de service, zo blijft je controller kort en krachtig.

5 Context

Zoals eerder beschreven draait heel OnlineSoccerManager en OnlineFootballManager op meerdere databases die verspreid zijn over meerdere servers. Binnen de website wordt er gebruik gemaakt van een eigen oplossing die WebContext heet. Deze WebContext bepaald aan de hand van de sessie op welke server we momenteel bewerkingen uitvoeren. Binnen de API bestaan er geen usersessies. Het bepalen van Servers en Domeinen moet dus ergens anders gebeuren. Om deze reden is er een ApiContext gemaakt.

```
public static DataContext Get(HttpRequestBase requestBase)
{
    int serverNr = 0;

    //decide what server we are on based on either the supplied login or competition
nr
    if(!string.IsNullOrEmpty(requestBase.QueryString["login"]))
    {
        var userRepository = new UserRepository(WebContext.Get);
        User user = userRepository.Fetch(requestBase.QueryString["login"]);

        if(user != null)
            serverNr = user.ServerNr;
    }
    else if (!string.IsNullOrEmpty(requestBase.QueryString["compnr"]))
    {
        serverNr =
OSM.Data.Models.Users.Server.GetServerNrFromCompNr(Convert.ToInt32(requestBase.QuerySt
ring["compnr"]), WebToolbox.GetWorld());
    }

    return new DataContext
    {
        WorldNr = WebToolbox.GetWorld(),
        DB = Constants.DB.Comp,
        ServerNr = serverNr
    };
}
```

Deze maakt in plaats van sessies gebruik van parameters in de URL en bepaalt zelf welke er nodig zijn. In de Widget en WorldMap werd de login gebruikt om het serverNr bij deze user te zoeken. Deze ondersteuning is wederom geïmplementeerd. Echter is er ook functionaliteit toegevoegd om via het Competitie nummer de server te bepalen. Dit omdat bij autorisatie met OAuth geen Login meer meegestuurd wordt, en er ook een aantal publiekelijke API´s zijn, denk hierbij aan het opvragen van een competitietussenstand, hierbij wordt alleen het competitie nummer meegstuurd.

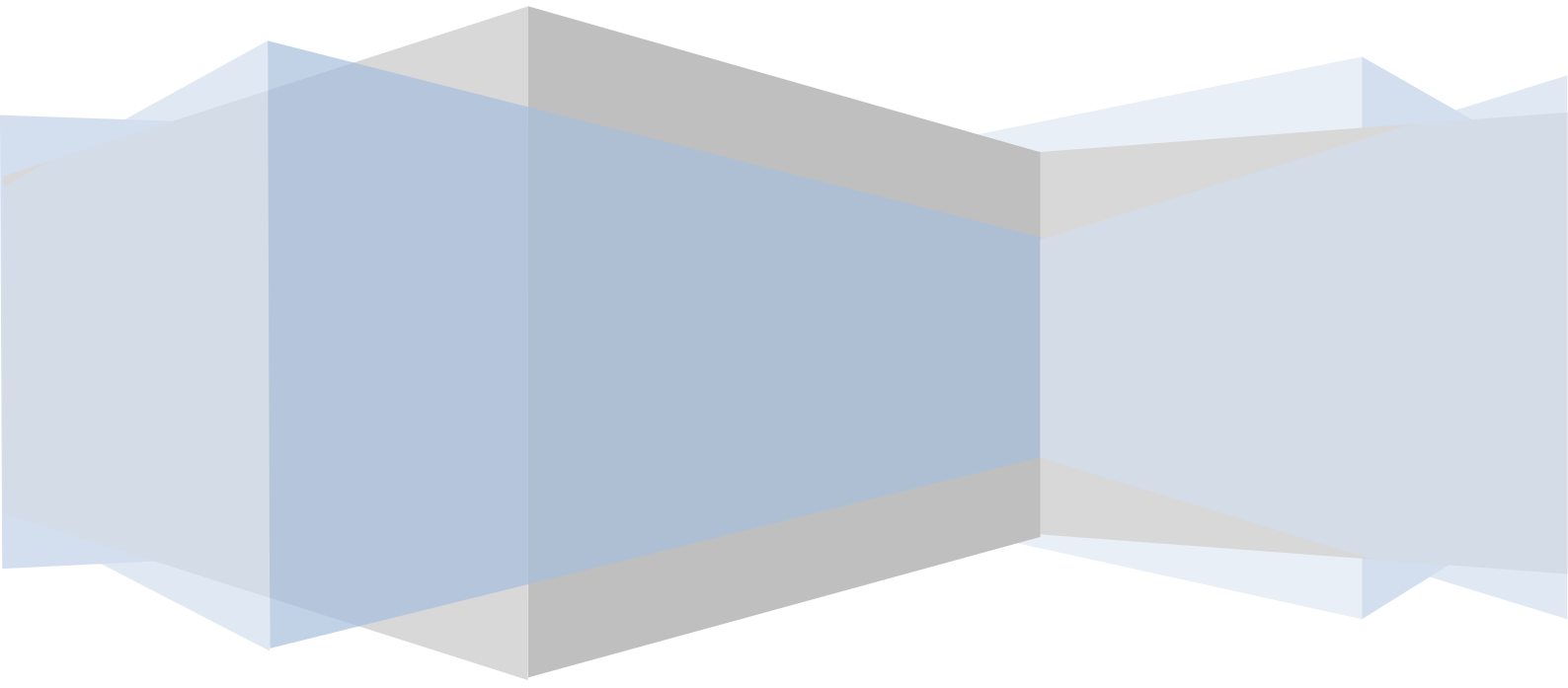
GameBasics

Construction Document

Versie 0.2

Dit document beschrijft de technische aspecten van de implementatie van de legacy API en de bijzonderheden tijdens deze iteratie van de Construction fase.

Robin Scholtes
22-2-2011



Inhoudsopgave

1 Inleiding

Dit document wordt geschreven met het doel om enkele specifieke technische zaken toe te lichten die in het ontwerp niet duidelijk gemaakt konden worden. Dit betekent dat dit document zich ook meer toe zal spitsen op de code.

1.1 Structuur document

Hieronder volgt een beschrijving van de API acties die speciale toelichting nodig hebben. Er wordt gekeken naar een aantal acties op het team en de vertaling om zo een inzicht te geven in hoe de functionaliteiten overal werken. Daarbij wordt de API ActionFilter hier toegelicht, omdat deze in deze fase afgerond is. De ontwikkeling hiervan is begonnen in de vorige Construction fase.

2 API ActionFilter

Zoals in het ontwerp naar voren kwam is de API ActionFilter er om er voor te zorgen dat data geserialeerd en gedeserialeerd kan worden. Binnen de API aanroepen is het noodzakelijk dat alle aanroepen goed gedaan worden. Om deze reden is er dan ook een uitgebreid validatie process.

De eerste stap in dit proces is kijken of de methode van evt. data input valide is en deze methode omzetten naar een interne waarde waar we verder mee kunnen.

```
if (FormPostContentTypes.Any(contentType.Contains)) _inputDeserializationFormat =
SerializationFormat.Post;

if ((_inputDeserializationFormat & AllowedInputMethods) !=
_inputDeserializationFormat)
{
    filterContext.Result =
ApiBind.GenerateErrorResult("INVALID_INPUT_FORMAT",string.Format(Resources.Messages.In
validInputFormat, contentType));
    return;
}
```

Bovenstaand stukje code is een klein stukje hiervan, er wordt gekeken of de input methode gelijk is aan een van de vooraf gedeclareerde post methodes, wanneer dit zo is wordt de input methode op Post gezet. De volgende stap is om te kijken of deze methode in de lijst met toegestane methodes voorkomt. Hiervoor wordt gebruik gemaakt van Bitwise callculations, in C# ook wel bekend als Flags.

Wanneer de input methode gevalideerd is, wordt de methode van output op dezelfde methode gevalideerd. Zoals er te zien is, kunnen er al twee manieren zijn die voorkomen dat de API call vroegtijdig wordt afgebroken.

De volgende stap is het valideren van de parameters. Binnen C# zijn is er de mogelijkheid variabelen een default waarde te geven. Wanneer dit het geval is, is het niet verplicht deze waarde in te vullen, anders wel.

```
List<ParameterDescriptor> emptyRequiredParameters = parameters.Where(parameter =>
parameter.DefaultValue == null)
    .Where(parameter => filterContext.ActionParameters[parameter.ParameterName] ==
null).ToList();

if(emptyRequiredParameters.Count > 0)
    filterContext.Result = ApiBind.GenerateErrorResult("INVALID_PARAMETER",
string.Format(OSM.Library.Api.Resources.Messages.EmptyRequiredParameter,
emptyRequiredParameters[0].ParameterName.ToLower()));
```

Bovenstaande code zorgt voor de validatie van input parameters. De parameters die een action nodig heeft zijn beschikbaar binnen MVC. Door met LINQ op te vragen welke van deze een default value van NULL hebben (deze mogen dus niet leeg zijn), en van deze collectie te kijken of de waarde ook daadwerkelijk leeg is. Is het mogelijk een error message te genereren wanneer een parameter niet is ingevuld. Dit betekent dat deze validatie niet meer in de Controllers wordt gedaan, maar gecentraliseerd door de API. Dit zorgt er voor dat API Calls niet worden uitgevoerd als alle velden niet zijn ingevuld. Als deze stappen allemaal doorlopen zijn wordt de Call daadwerkelijk uitgevoerd.

3 Team actions

Zoals eerder te zien viel in het Elaboration Document kan met de team Controller verschillende acties uitgevoerd worden op een team. Sommige van deze functionaliteiten zijn publiekelijk, andere vereisen een token. Aangezien er op het moment van het schrijven van dit document nog geen werkende autorisatie is, wordt dit buiten beschouwing gelaten en wordt er specifiek naar de functionaliteiten zelf gekeken.

Binnen de team controller is er ook weer gekozen om gebruik te maken van LINQ om subselecties te kunnen maken binnen teams. Dit is handig om bijvoorbeeld te kijken welke spelers geblesseerd zijn binnen een team, of welke spelers geschorst zijn.

```
var players = PlayerService.GetEligibleForLineup(compNr, _teamNr).Where(eligiblePlayer
=> (eligiblePlayer.Player.Lineup >= 12 && eligiblePlayer.Player.Lineup <=
18)).Select(x => x.Player).ToList();
```

Bovenstaande code wordt gebruikt om de wissels van een team te selecteren uit de collectie van spelers die speelgerechtigd zijn. Deze techniek zorgt ervoor dat er uit de database grotere collecties gehaald worden en dat de subselecties buiten de database gemaakt worden. Buiten het feit dat dit voor gemak in de code blijkt het in de praktijk ook sneller te zijn dan de query's in de database meer vergelijkingen te laten oplossen.

Naast het opvragen van informatie over een team, is dit een van de eerste controllers waarbij ook acties geïmplementeerd moeten worden waarmee we waardes kunnen setten/veranderen.

```
[ApiBind, HttpPost]
public ActionResult Specialist(int compNr, int _teamNr, int captain, int penalty, int
freekick, int corner)
{
    SpecialistResult result = SpecialistService.SetSpecialists(captain, freekick,
penalty, corner, compNr, _teamNr);
    return new CompositeActionResult
    {
        MessageCode = (result == SpecialistResult.Success) ? Messages.Succes :
Messages.Error,
        MessageString = result.ToString()
    };
}
```

Bovenstaand stukje code wordt gebruik om de sepcialisten van een team te setten. Dit zijn 4 posities in een team die extra verantwoordelijkheden krijgen toegewezen. Door het toevoegen van `HttpPost` reageer deze functie op input die gepost wordt. Dit betekent dat er duidelijk onderscheid in functies is welke bedoelt zijn om waardes op te halen en welke om waardes te veranderen.

4 Vertaling

OnlineSoccerManager en OnlineFootballManager is beschikbaar in verschillende talen. Ook binnen de Widgets wordt al gebruik gemaakt van een vertaling object. Er is in de Elaboration Fase besloten een losse controller te maken voor de vertalingen. Deze acties halen uit de C# Resx files de goede waardes en zetten deze om naar XML bestanden.

Binnen C# was het al mogelijk om met de ResourceManager subsets van resources te selecteren. Dit kon echter niet aan de hand van een taal. Daarom is er een zogenaamde Extension (uitbreiding opbestaand object) geïmplementeerd.

```
public static class ResourceManagerExtension
{
    public static ResourceSet GetResourceSet(this ResourceManager resourceManager,
string languagecode = "en")
    {
        return
resourceManager.GetResourceSet(CultureInfo.GetCultureInfo(languagecode), true, true);
    }
}
```

Deze methode maakt het mogelijk om aan de resourcemanager een String met een taal mee te geven. Aan de hand van deze waarde wordt dan bepaald welke vertalingen nodig zijn. Aangezien er intern ook met String´s gewerkt wordt om de taal aan te geven is dit een handig uitbreiding die het makkelijk maakt snel sets van vertalingen op te vragen.

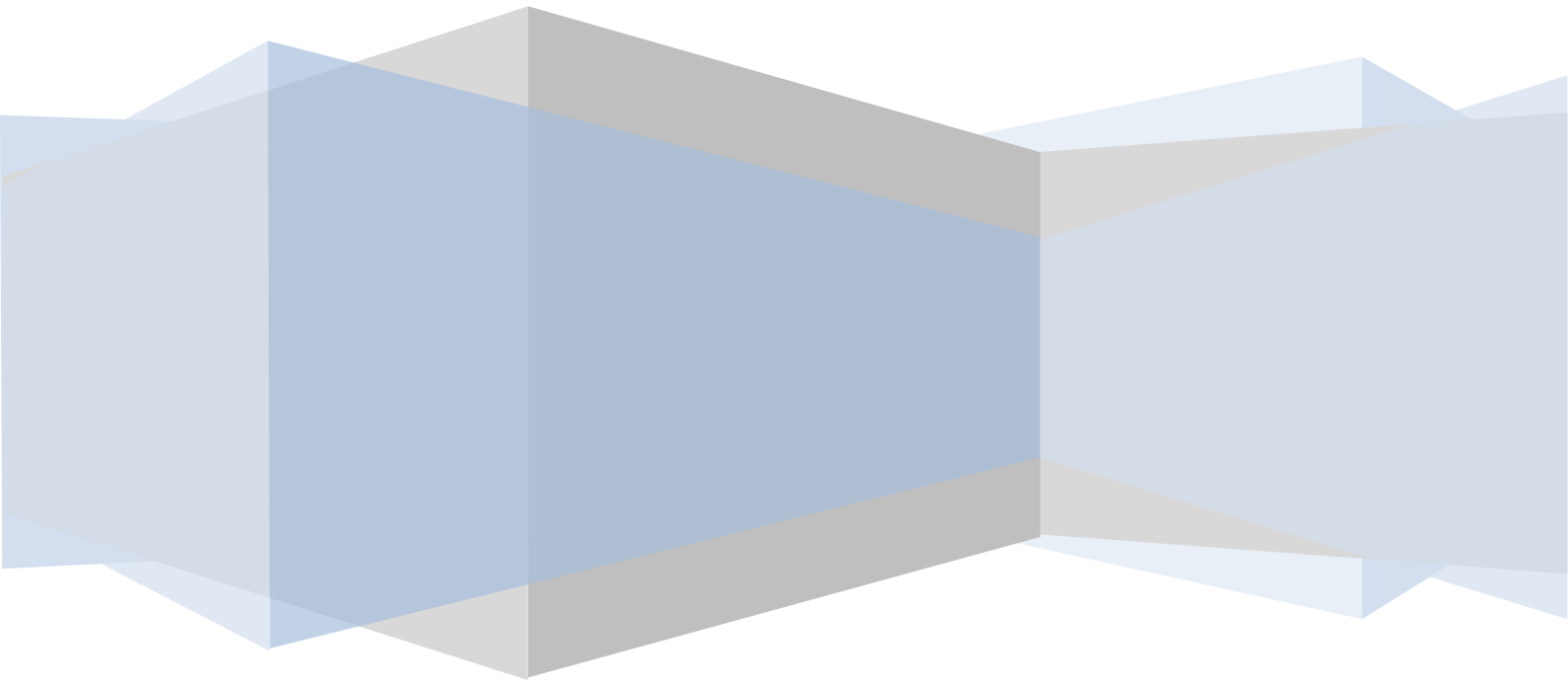
GameBasics

Construction Document

Versie 0.2

Dit document beschrijft de technische aspecten van de implementatie van de autorisatie binnen de API en de bijzonderheden tijdens deze iteratie van de Construction fase.

Robin Scholtes
15-12-2010



Inhoudsopgave

1 Inleiding

Dit document wordt geschreven met het doel te beschrijven welke functionaliteiten er op welke manier geïmplementeerd gaan worden. Het doel van dit document is om van te voren te bepalen op welke manieren de functionele en technische eisen gerealiseerd gaan worden.

1.1 Structuur document

In dit document worden de bijzondere technische details besproken die bij het implementeren van OAuth naar boven zijn gekomen. Daarbij wordt er niet alleen besproken hoe tokens verkregen/verstrekt worden, maar ook hoe functies van autorisatie voorzien kunnen worden.

2 OAuthController

De Oauthcontroller is de controller die het mogelijk maakt voor gebruikers om tokens aan te vragen en deze weer te valideren. Deze controller communiceert met de achterliggende service om Token en Consumer informatie op te halen.

Het opgeven van alle OAuth parameters is verplicht gemaakt om er voor te zorgen dat er een standaard aanroep gegenereerd wordt. In de API is een statisch object opgenomen wat aan de hand van de parameters OAuth tokens genereert. Dit gaat in een aantal specifieke stappen die er voor zorgen dat de OAuth versleuteling volgens de specificaties gebeurt. Als eerste worden de parameters van oAuth en de eventuele extra parameters genormaliseerd. Dit betekent dat alle parameters met een & gescheiden worden en dat de waarde en naam van de parameter weer gescheiden worden door een =

Key1=waarde&key2=waarde&key3=waarde

Deze waarde wordt samen genomen met de request methode (Post / Get) en de URL. Deze waarde wordt ook wel de Signature Base genoemd.

```
public static string CreateSignature(string signatureBase, string consumerSecret,
string tokenSecret)
{
    if (tokenSecret == null)
        tokenSecret = string.Empty;

    // URL encode key elements
    consumerSecret = Uri.EscapeDataString(consumerSecret);
    tokenSecret = Uri.EscapeDataString(tokenSecret);

    // initialize the cryptography provider
    var key = String.Concat(consumerSecret, "&", tokenSecret);
    var keyBytes = Encoding.UTF8.GetBytes(key);
    var signatureMethod = new HMACSHA1(keyBytes);

    // create a signature with the base and provider
    var data = Encoding.ASCII.GetBytes(signatureBase);
    var hash = signatureMethod.ComputeHash(data);
    var signature = Convert.ToBase64String(hash);

    // You must encode the URI for safe net travel
    signature = Uri.EscapeDataString(signature);
    return signature;
}
```

Bovenstaand stuk code is hetgeen wat er voor zorgt dat de signature van een token wordt gegenereerd. De signature wordt gebruikt om een request te valideren aan serverside. Binnen de signature base zitten de oAuth_timestamp en een random waarde. Deze signature base wordt samen met de consumersecret en tokensecret gehashed. Dit gebeurt door de twee secrets samen te voegen en van deze string de byte's door het HMACSHA1 algoritme te laten versleutelen. Op deze manier zitten de secrets versleuteld in de signature en hoeven deze waardes dus niet meegestuurd.

Wanneer een OAuth request token wordt aangevraagd, dit is het aller eerste token, wordt alleen de consumer en token secrets mee versleuteld. Wanneer een acces token wordt aangevraagd worden de request signature en secret ook mee versleuteld.

3 ApiAuthorize ActionFilter

In de website bestaat er een ActionFilter die gebruikt wordt voor autorisatie, deze valideert of er een geldige Sessie is. Binnen API Calls hebben we geen sessies en maken we gebruik van tokens. Om er voor te zorgen dat functies toch beveiligd kunnen worden is er een ApiAuthorize. Deze ActionFilter valideert het OAuth request en verleend wel of geen toegang tot de aangevraagde API call. Zoals eerder gezegd kon in de website, wanneer een gebruiker gevalideerd was, zijn informatie uit de sessie gelezen worden. In de API kan dit niet, hiervoor is er gekozen om gebruik te maken van de MVC ActionParameters. In de ActionFilter zijn de paramaters beschikbaar die de Action nodig heeft, deze zelfde collectie werd gebruikt om verplichte waardes eruit te halen en te kijken of deze gevuld waren. Aan deze collectie mogen echter waardes toegevoegd worden, wat het dus mogelijk maakt wanneer het token valide is, om extra parameters toe te kennen. De volgende waardes worden standaard gegenereerd:

- _login: De login van de gebruiker
- _status: De status van de gebruiker (seizoenskaarthouder of niet)
- _weekNr: De week waarin de competitie zich bevindt.
- _compNr: De competitie waarin de gebruiker zich bevindt.
- _langCode: De code van de taal van de gebruiker (EN, NL, AR etc.).
- _langNr: Het interne nummer voor een taal
- _isStaff: is de gebruiker lid van de staff (ja of nee).
- _name: De naam van de manager.

Om een request te valideren worden een aantal stappen doorlopen

```
 //get token and consumer from the service by the querystring paramaeters
Token token = oAuthService.GetToken(queryString["oauth_token"]);
Consumer consumer = oAuthService.GetConsumer(queryString["oauth_consumer_key"]);

///validat token and consumer
if (token == null)
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_TOKEN_ERROR", "The given
token was invalid");
else if (consumer == null)
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_CONSUMER_ERROR", "The
given consumer key was invalid");
else if (OauthFields.Any(x => !queryParameters.Contains(x)))
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_PARAM_ERROR", "The
required parameters where not set");
else if (token.Expires < DateTime.Now)
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_TOKEN_EXPIRED", "The
token supplied is no longer valid");
else if (Convert.ToInt32(queryString["oauth_timestamp"]) >
DateTime.Now.AddSeconds(15).ToFileTimeUtc())
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_TOKEN_EXPIRED", "The
token supplied is no longer valid");
```

De token en de consumer worden opgehaald, er wordt gekeken of deze ook daadwerkelijk gevonden worden. Vervolgens wordt er gekeken of alle OauthFields bestaan. Tenslotte worden er twee timestamps bekeken, de eerste kijkt of het Token niet is verlopen, de volgende check kijkt of het request in de afgelopen 15 seconde gedaan is, de laatste is er om er voor te zorgen dat het sniffen van urls niet mogelijk is, omdat requests een korte verlooptijd hebben.

Wanneer deze waardes gevalideerd zijn, wordt de signature gevalideerd, dit wordt gedaan door een signature te generen met de waardes uit het request.

```
var oauthParameters = OAuth.GetOAuthParameters(noOAuthParams, url, "GET",
consumer.Key, consumer.Secret, token.OauthToken, token.Secret,
queryString["oauth_timestamp"], queryString["oauth_nonce"]);

if (oauthParameters["oauth_signature"] !=
Uri.EscapeDataString(queryString["oauth_signature"]))
    filterContext.Result = ApiBind.GenerateErrorResult("AUTH_VALIDATION_ERROR", "The
supplied parameters did not match the token.");
```

Wanneer deze signatures gelijk zijn is het request goedgekeurd en kunnen we de gegevens van de gebruiker ophalen en dit doorsturen naar de Actie. Zoals eerder beschreven wordt er een reeks aan gegevens beschikbaar gesteld, het flexibele aan deze implementatie is dat deze parameters naar de actie gestuurd worden, maar de actie hoeft niets te doen met deze waardes, ze hoeven zelfs niet in de functie declaratie te staan.

```
[ApiBind, ApiAuthorize]
public ActionResult Buy(int compNr, int playerNr , int _teamNr)
```

Bovenstaande functie maakt gebruik van de ApiAuthorize, en gebruikt alleen het _teamNr, de andere waardes zijn hier niet nodig.