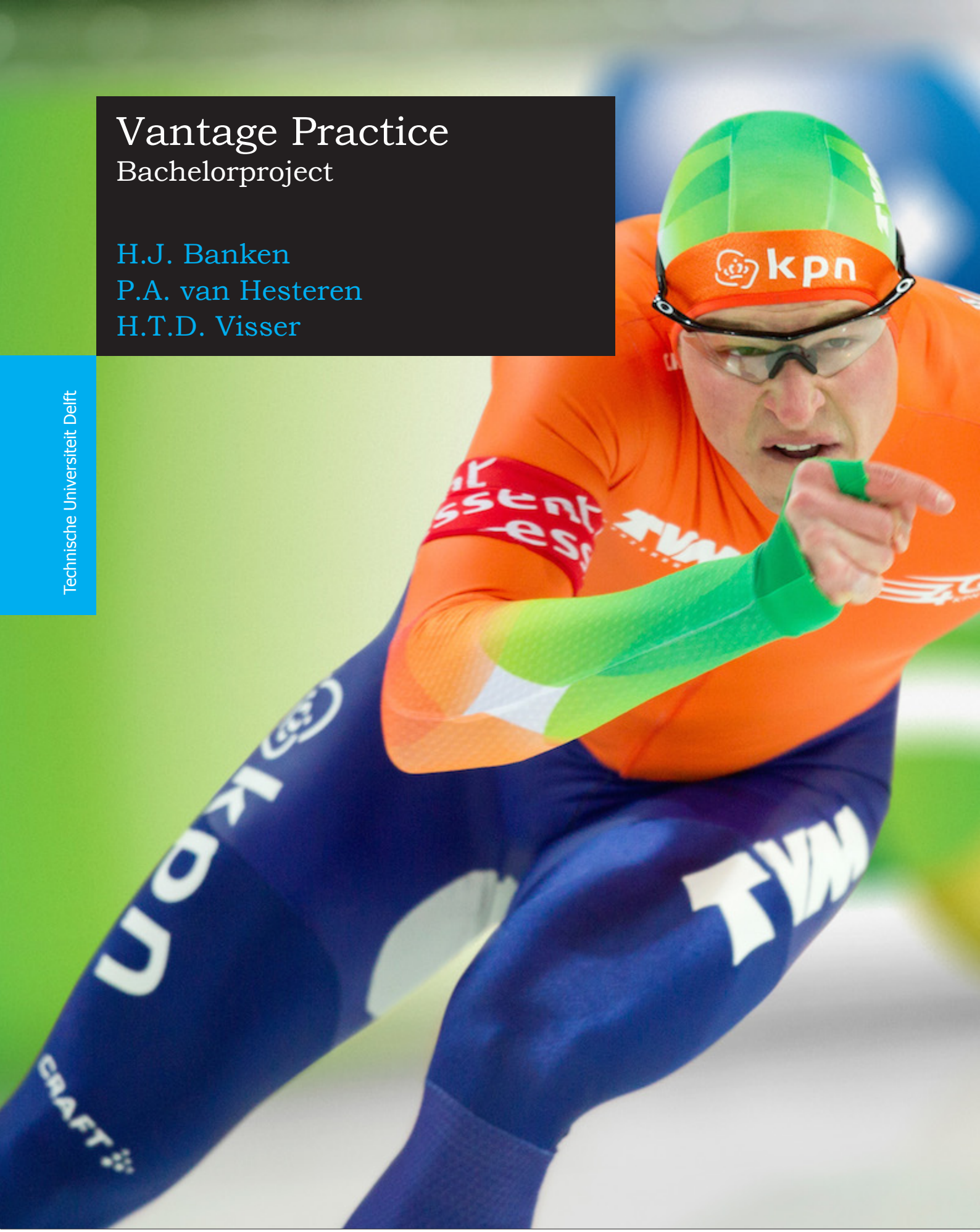
Vantage Practice

Bachelorproject

H.J. Banken
P.A. van Hesteren
H.T.D. Visser

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

Vantage Practice

Trainingsapp voor Baansporten

Bachelorproject

door

Herman Banken

Patrick van Hesteren

Hylke Visser

TU Coach: Dr.ir. C. Bezemer
Projectbegeleider: J. Stokking

emando

TU Delft Delft University of Technology

Voorwoord

Dit document is het eindverslag van het bachelorproject "Trainingsapp voor Baansporten", van de opleiding Technische Informatica op de TU Delft, uitgevoerd door Herman Banken, Patrick van Hesteren en Hylke Visser.

We willen graag de volgende personen bedanken voor hun steun en begeleiding gedurende het project:

Coach (TU Delft): dr. ir. Cor-Paul Bezemer

Als TU Coach heeft Cor-Paul een grote rol gespeeld binnen het traject van het project. Naast het toezicht houden op het proces, heeft hij ons wekelijks voorzien van feedback op de voortgang van het project. Tevens heeft bij het schrijven van dit document en het voorbereiden van de gebruikerstest voorzien van tips en feedback.

Projectbegeleider (Eigenaar Emando): Johan Stokking

Als projectbegeleider heeft Johan een belangrijke rol gespeeld binnen het ontwikkeltraject van het project. Naast het toezichthouden op de kwaliteit en voortgang van het project, heeft hij een actieve rol gespeeld binnen zowel het implementatie- als het ontwerp-proces van de applicatie.

Daarnaast willen we Martha Larson, dr. ir. Johan Pouwelse en Marleen Banken bedanken voor de tijd en moeite die zij in ons project hebben gestoken.

H.J. Banken
P.A. van Hesteren
H.T.D. Visser
Delft, 19 Juni 2014

Inhoudsopgave

1 Inleiding **1**

2 Opdracht en Omschrijving **5**
2.1 Opdrachtsamenvatting . . . 5
2.2 Doelgroep en Gebruikers . . . 6
2.3 Reeds bestaande vergelijkbare systemen . . . 6
2.4 Programma van eisen . . . 6

3 Projectmethodologie **9**
3.1 Scrum . . . 9
3.2 Stand up meetings . . . 10
3.3 Team Foundation Server . . . 10
3.3.1 Verantwoordelijkheden . . . 11
3.3.2 Taakverdeling . . . 11
3.4 Code reviews . . . 11
3.5 Projectkeuzes . . . 11

4 Systeemontwerp en -implementatie **13**
4.1 Datalaag . . . 14
4.1.1 Ruwe inkomende data . . . 14
4.1.2 Entiteiten . . . 14
4.1.3 Azure SQL & Entity Framework . . . 15
4.1.4 Azure Table Storage . . . 15
4.2 Businesslaag . . . 17
4.2.1 Aggregatiestructuur . . . 17
4.2.2 Dataconsistentie en parallellisering . . . 20
4.2.3 Workflows . . . 20
4.3 Servicelaag . . . 21
4.3.1 SignalR . . . 22
4.3.2 WebApi . . . 22
4.3.3 OWIN . . . 22
4.3.4 Platform-onafhankelijke client . . . 22
4.3.5 Lokale opslag met Akavache . . . 23
4.4 Presentatielaag . . . 23
4.4.1 ViewModels en Reactive UI . . . 23
4.4.2 Grafische user interface . . . 24
4.5 Testomgeving . . . 26
4.5.1 Integratietesten van client en server . . . 26

5 Evaluatie **29**
5.1 KNSB Meetings . . . 29
5.2 Enquête & gebruikerstest . . . 29
5.2.1 Enquête . . . 29
5.2.2 Resultaten enquête . . . 30
5.2.3 Gebruikerstest . . . 30
5.2.4 Resultaten gebruikerstest . . . 30
5.3 SIG feedback . . . 31

6 Discussie **33**
6.1 Product . . . 33
6.1.1 Hoe “af” is de applicatie? . . . 33
6.1.2 Verbeteringen . . . 33
6.1.3 Android . . . 33
6.1.4 Is de applicatie een toevoeging op de reeds bestaande markt? . . . 33
6.2 Proces . . . 34
6.2.1 SCRUM overhead . . . 34
6.2.2 Burndown & afhankelijkheden . . . 34
6.2.3 Verwachtingen verschillende betrokkenen . . . 35
6.2.4 Cutting-edge technieken . . . 35

7 Conclusie **37**
7.1 Oplevering . . . 37
7.2 Doorontwikkeling . . . 37
7.3 Toekomst . . . 38

A Plan van Aanpak **39**
A.1 Introductie . . . 39
A.2 Projectopdracht . . . 39
A.3 Specifieke proceseisen . . . 40
A.4 Tijdsplanning . . . 40
A.5 Projectinrichting . . . 40
A.6 Kwaliteitsborging . . . 41

B Planning **43**

C Oriëntatieverslag **45**
C.1 Introductie . . . 45
C.2 Soortgelijke oplossingen . . . 45
C.3 Doelgroep & gebruikers . . . 45
C.4 Back-end . . . 45
C.5 Client . . . 47
C.6 Entiteiten . . . 49
C.7 Database . . . 50
C.8 Architectuur . . . 51

D Interface Ontwerpen **53**

E Testresultaten **57**

F Aggregatie Flow **61**

Bibliografie **63**

Samenvatting

In baansporten is de laatste jaren een ontwikkeling gaande om tijdregistratie te digitaliseren door het gebruik van transponders en detectielussen in de baan. Daardoor zijn nieuwe mogelijkheden ontstaan om ook naast het wedstrijdmoment de sportprestaties in te zien.

Het huidige gebruik van transponders - buiten wedstrijden - is voornamelijk achteraf, terwijl zowel sporter als coach juist *tijdens* de training het meest bezig zijn met de prestaties. Het is daarom wenselijk om de resultaten in realtime door te geven aan coaches en sporters zelf. De bestaande oplossingen bieden slechts beperkt inzicht in trainingen, hetgeen de aanleiding voor dit Bachelor Eindproject is geweest.

Gedurende dit project is met hulp van de agile ontwikkelmethode Scrum in combinatie met Team Foundation Server gewerkt aan een applicatie die op een overzichtelijke manier inzicht geeft in trainingen van eigen, maar ook de prestaties van andere sporters. Daarnaast bevat de applicatie een sociale component, waarmee sporters gemotiveerd worden om te blijven of te gaan sporten.

Binnen de applicatie kunnen gebruikers namelijk een transponder toekennen aan hun account en het is mogelijk om groepen aan te maken en/of hier deel in te nemen. De groepen bieden bijvoorbeeld een coach, zijn pupillen en hun thuisfront c.q. aanhang de mogelijkheid om naast de eigen resultaten ook de resultaten van andere groepen in te zien.

Daarnaast bevat dit sociale component ook de mogelijkheid om leaderboards, een soort ranglijst, per verschillende context (baan of groep) in te zien. Zo heeft iedere gebruiker zijn eigen records, maar heeft hij ook een positie op de ranglijst van de banen waarop hij gesport heeft en in de groepen waarvan hij deel uitmaakt.

Om sporters reeds tijdens hun training inzicht te geven in hun prestaties dient alle verkregen data realtime verwerkt en verstuurd te worden naar de mobiele telefoons. Om dit mogelijk te maken draait er in de Microsoft Azure Cloud een server die doorkomsten van de transponders verwerkt. De doorkomsten komen binnen vanuit MyLaps, de transponder- en detectielussenleverancier. Vervolgens groeperen we doorkomsten op trainingssessies en ronden, en onderscheiden we de rustperiodes. Het verwerken van deze data is bijzonder complex: de beschikbare data wordt in elke stap een beetje meer verrijkt en continu naar de mobiele apps verstuurd, zodat gebruikers altijd de laatst beschikbare informatie zien.

De applicatie is ontwikkeld met behulp van cross platform ontwikkeltechnieken, waardoor deze relatief eenvoudig uit te brengen is op andere mobiele platformen. De back-end van de applicatie is sport-agnostisch, waardoor deze breed inzetbaar is voor allerlei verschillende soorten baansporten.

Uit de enquêtes, die gedurende het project afgenomen zijn onder de doelgroep van de applicatie, is gebleken dat er veel vraag naar een dergelijke applicatie is en dat de applicatie veel potentie heeft. Ook is er tijdens meetings tussen Emando, de opdrachtgever, en de Koninklijke Nederlandsche Schaatsenrijders Bond (KNSB) naar voren gekomen dat de KNSB veel interesse heeft in de applicatie en de doorontwikkeling ervan.

Momenteel is de iPhone applicatie klaar om de markt op te gaan, maar uit de enquêtes, gebruikerstest en vanuit de KNSB is naar voren gekomen dat er nog veel functionaliteit is die toegevoegd zou kunnen worden aan de applicatie. Onze aanbeveling is dan ook om de applicatie verder te ontwikkelen en de doelgroep te vergroten door de applicatie ook uit te brengen voor andere platformen.

1

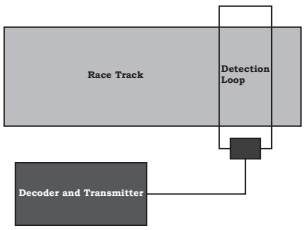
Inleiding

In baansporten is de laatste jaren een ontwikkeling gaande om tijdregistratie te digitaliseren door het gebruik van transponders zoals de MyLaps ProFlex transponder in figuur 1.1 en detectielussen die in de baan zijn verwerkt, wat te zien is in figuur 1.2. Door deze ontwikkeling zijn nieuwe mogelijkheden ontstaan om ook naast het wedstrijdmoment de sportprestaties in te zien. Het constateren van de hierna genoemde nieuwe mogelijkheden vormde de aanleiding voor dit project.

Figuur 1.1: MyLaps ProFlex transponder, op schaal

Een grote speler op de tijdregistratiemarkt is MyLaps[1]. Dit bedrijf installeert en beheert detectielussen en is actief bij diverse sporten zoals schaatsen, wielrennen, zwemmen, atletiek en diverse motorsporten. Bij sporten met permanente banen liggen de detectielussen het gehele jaar in de baan. Er bestaat de mogelijkheid om op de website van MyLaps doorkomsttijden in te zien. De informatie die uit deze tijden is af te leiden, wordt door sporters die zich willen blijven verbeteren als erg waardevol gezien, waardoor er steeds meer getraind wordt met deze transponders.

Voor aanvang van het seizoen installeert MyLaps een aantal detectielussen in de baan. De lussen bevinden zich in het ijs of onder het houten oppervlak van een baanwielrenbaan. Elke lus heeft een elektromagnetisch veld. Wanneer een sporter met transponder over dat veld heen rijdt, wordt de transponder geactiveerd en stuurt deze een unieke puls, die vervolgens door de lus wordt opgevangen. De MyLaps X2 Server die aan de detectielussen zit aangesloten stuurt vervolgens het signaal door naar de MyLaps Cloud.



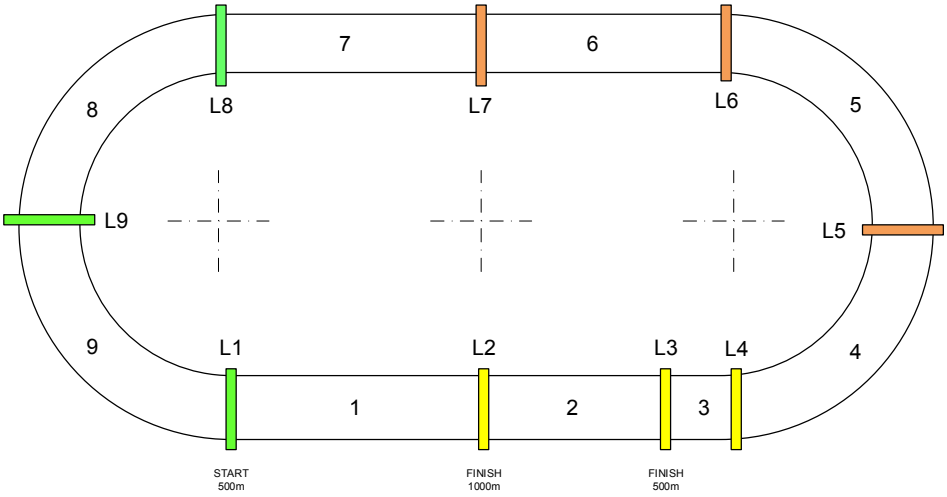
Figuur 1.2: Een schema van een detectielus en en decoder

Het huidige gebruik van transponders - buiten wedstrijden - is voornamelijk achteraf, terwijl juist tijdens de training zowel sporter als coach het meeste bezig zijn met de prestaties. Het is daarom wenselijk om de resultaten in realtime door te geven aan coaches en sporters zelf.

Op veel banen zijn meerdere detectielussen geïnstalleerd, terwijl er op de website van MyLaps slechts één wordt ontsloten omdat MyLaps dit niet ziet als haar 'core business'. In Thialf, de schaatsbaan in Heerenveen, Friesland, liggen bijvoorbeeld twaalf detectielussen, in Haarlem (figuur 1.3) negen en op de andere schaatsbanen liggen er tenminste twee. Door de data van meerdere lussen te combineren is een betere indicatie te maken van de snelheid van sporters. In dit document zullen we de specifieke toepassing bij schaatsen toelichten, maar het principe is vrijwel hetzelfde voor elke denkbare baansport.

Een gemiddelde schaatstraining, zoals te zien in figuur 1.4 bestaat uit losse trainingsonderdelen.

[1] http://www.mylaps.com



Figuur 1.3: L1 tot en met L9 zijn de negen MyLaps detectielussen in Haarlem

Allround schaatsers moeten tijdens zo'n training bijvoorbeeld 4 ronden warmrijden, dan 6 keer 2 ronden sprinten, dan 4 keer een sprintje van 200 meter, 2 keer een glijstart en dan 2 keer een echte start bij de coach, vanuit de zijkant van de baan. Zin als volgt veranderen: Tussen de opdracht door is er rust, waarin het gebruikelijk is om tenminste 400 meter uit te glijden.

Figuur 1.4: Een typisch schaats-trainingsschema

Veel trainingselementen bestaan dus uit korte opdrachten, waar het juist om snelheid gaat. Een enkele lus is dan niet afdoende, omdat de rust voor of na de opdracht mee wordt gewogen. Schaatsers starten en stoppen namelijk niet precies boven de lus, maar starten vaak na de bocht en stoppen afhankelijk van de afstand van de opdracht op een willekeurig punt.

Andere trainingen van bijvoorbeeld lange afstands- of marathonschaatsers kunnen bestaan uit één element: de hele training lang schaatsen, zonder overeind te komen. Wanneer er op tactische punten detectielussen geïnstalleerd zijn, is het bijvoorbeeld ook mogelijk om onderscheid te maken tussen bochten en de rechte stukken. Bij sporten met een ronde baan verschilt de snelheid in de bocht namelijk erg met die op het rechte eind. Deze vergelijking wordt nu al in Thialf gedaan, maar eigenlijk alleen in het professionele circuit en tijdens wedstrijden op televisie.

Topsporters zijn enorm prestatiegericht en als je al aan de top zit, dan kunnen kleine aanpassingen aan je techniek, ademhaling et cetera, allesbepalende verschillen maken. Coaches van professionele teams houden zich daarom bezig met allerhande analyses. Naast ademanalyse en hartslag monitoring, is het in Thialf bijvoorbeeld ook mogelijk om (van maximaal 20 schaatsers) continu de positie te bepalen met een in-door positioning system (IPS) ontwikkeld door InnoSportLab[2].

[2] http://www.innosportlabthialf.nl

Al deze geavanceerde analyses zijn echter duur en kosten te veel tijd om aantrekkelijk te zijn voor recreatieve schaatsers. MyLaps X2 kan in combinatie met de applicatie die tijdens dit bachelorproject is gebouwd, recreatieve schaatsers een manier bieden om analyses uit te voeren en hun prestaties naar een hoger niveau te tillen. Het is dan ook de bedoeling dat iedereen met een iPhone vanaf dit schaatsseizoen de Vantage Practice Applicatie kan downloaden vanuit de App Store.

Opbouw Verslag

In dit verslag zal allereerst in hoofdstuk 2 de opdracht en het domein van de opdracht worden toegelicht. Vervolgens zal in hoofdstuk 3 de methodologie worden behandeld die tijdens het proces is aangehouden. Hierna zal in hoofdstuk 4 gedetailleerd worden toegelicht op welke manier het systeem is ontworpen, welke implementatiekeuzes er zijn gemaakt en hoe deze keuzes zijn uitgewerkt. In hoofdstuk 5 wordt behandeld hoe de applicatie wordt geëvalueerd met behulp van tests en feedback van verschillende betrokken partijen en mogelijke eindgebruikers. Ten slotte zal in hoofdstuk 6 onder andere worden besproken in hoeverre het project een succes is geweest, wat er moet gebeuren om ook een Android applicatie te kunnen maken en welke invloed bepaalde keuzes hebben gehad op het project. Ten slotte zal in de conclusie, in hoofdstuk 7 worden aangegeven hoe de applicatie is opgeleverd, zal er een visie gegeven worden op de doorontwikkeling van de applicatie en zal worden besproken welke toekomst wij zien voor de applicatie.

2

Opdracht en Omschrijving

Emando B.V. is een jong bedrijf dat maatwerksoftware maakt. Momenteel ontwikkelt Emando voor de Koninklijke Nederlandsche Schaatsenrijders Bond (KNSB) een nieuw licentie-aankoop- en -beheersysteem, een nieuw wedstrijdadministratiesysteem en een nieuwe schaatstijdendatabase. Naar aanleiding van eerder contact met de KNSB over toegang tot de schaatstijdendatabase zijn wij in telefonisch contact gekomen met Johan Stokking van Emando. Tijdens het oriënterende gesprek bleek dat zowel wij als Emando er geïnteresseerd in waren dat wij ons Bachelor Eindproject bij Emando zouden uitvoeren, onder begeleiding van de heer Stokking.

2.1. Opdrachtsamenvatting

In de weken voorafgaand aan het Bachelor Eindproject is in samenspraak met Emando de opdrachtomschrijving gemaakt. Bij het brainstormen zijn de minimale eisen en de mogelijke extra functies bedacht:

> *Het ontwerpen en ontwikkelen van een systeem voor het weergeven en analyseren van (live) recreatieve en trainingsdata in de context van baansporten (bijvoorbeeld schaatsen en baanwielrennen). Dit systeem zal realtime gegevens van de baan, maar ook opgeslagen gegevens uit het verleden, inzichtelijk maken voor sporters, coaches en eventuele toeschouwers (volgers). Daarnaast zal er de mogelijkheid zijn om een sociale component en een competitie element toe te voegen, en een mogelijkheid om aggregatie op data uit te voeren.*
>
> *Het systeem ondersteunt het sporten: sporters hoeven hun training niet te onderbreken, doordat audio cues en sport-specifieke schermen geïmplementeerd worden. De sociale component bestaat uit het volgen van medesporters c.q. familieleden of vrienden en het delen van resultaten. Het competitie-element zal een virtuele competitie tussen vrienden omvatten.*
>
> *Technisch gezien zal er een scheiding gemaakt worden tussen de diverse toepassingen (clients) en de API. De API biedt toegang tot realtime data en de diverse modulaire (sport-specifieke) aggregaties, waarbij de mogelijkheid bestaat voor bijvoorbeeld coaches om zich te abonneren op meerdere sporters. De API zal gekoppeld worden op bestaande systemen die data van transponders ontsluiten.*

Om richting te geven aan het toekomstige applicatie, is de opdrachtomschrijving ambitieus opgesteld. We hebben vanaf het begin de instelling gehad om eerst een minimale versie te maken die werkt, om vervolgens de eerste versie uit te bouwen tot een volwaardige applicatie.

Zoals te lezen is in de opdracht zal er zowel een mobiele applicatie als een back-end systeem ontwikkeld worden. In sectie 2.4 worden de elementen uit de opdrachtomschrijving uitgesplitst en omgezet in functionaliteiten met de bijbehorende prioriteit.

2.2. Doelgroep en Gebruikers

Schaatsers

De voornaamste doelgroep van onze applicatie bestaat uit recreatieve en amateurschaatsers. Zij moeten met behulp van onze applicatie meer inzicht krijgen in hun prestaties en trainingen. Om ervoor te zorgen dat deze groep gebruikers onze applicatie verkiest boven de reeds bestaande (minder uitgebreide) systemen, dienen zij snel en gemakkelijk met de applicatie aan de slag te kunnen gaan.

Bij een té ingewikkeld systeem zullen zij afhaken en de overstap naar onze applicatie wellicht niet willen maken. Omdat zij de belangrijkste doelgroep zijn van de applicatie, is het belangrijk dat zij gemotiveerd worden om de applicatie te blijven gebruiken en hen tevens te motiveren om ook anderen aan te sporen dit te doen. Met behulp van sociale componenten zoals trainingsgroepen, het volgen van andere schaatsers en leaderboards kan onze applicatie hen hierin stimuleren.

Coaches

De tweede doelgroep bestaat uit de coaches van de schaatsers. Dit zijn mensen die de schaatsers begeleiden bij hun trainingen en wedstrijden. Coaches die zelf niet schaatsen beschikken niet altijd over een transponder, maar zij willen graag snel en gemakkelijk inzicht in de data van de schaatsers die zij coachen. Het moet voor hen dus gemakkelijk zijn om enerzijds snel tussen schaatsers te kunnen switchen en anderzijds de prestaties zowel onderling als per training te vergelijken.

Overige gebruikers

Tot slot is er de groep gebruikers die zelf geen deel uitmaakt van de schaatswereld. Zij zijn geen schaatsers of coaches en zijn met name geïnteresseerd in de mogelijkheden van de applicatie en het volgen van kennissen of bekende schaatsers. Ook zij moeten, weliswaar in mindere mate, in staat zijn om de applicatie te gebruiken.

2.3. Reeds bestaande vergelijkbare systemen

Er bestaan al enkele oplossingen die functionaliteit bieden die vergelijkbaar is met die van de applicatie die wij gaan bouwen. Strava (figuur 2.1(a)) en RunKeeper (figuur 2.1(b)) maken gebruik van GPS om onder andere hardlopers en wielrenners van realtime informatie te voorzien. MyLaps Practice (figuur 2.1(d)) is een website waar je na je training je prestaties kan terugkijken. Er bestaat ook een iPhone App voor MyLaps Practice (figuur 2.1(e)). Coach Watch (figuur 2.1(c)) is een iPad applicatie waarmee coaches hun teams kunnen volgen.

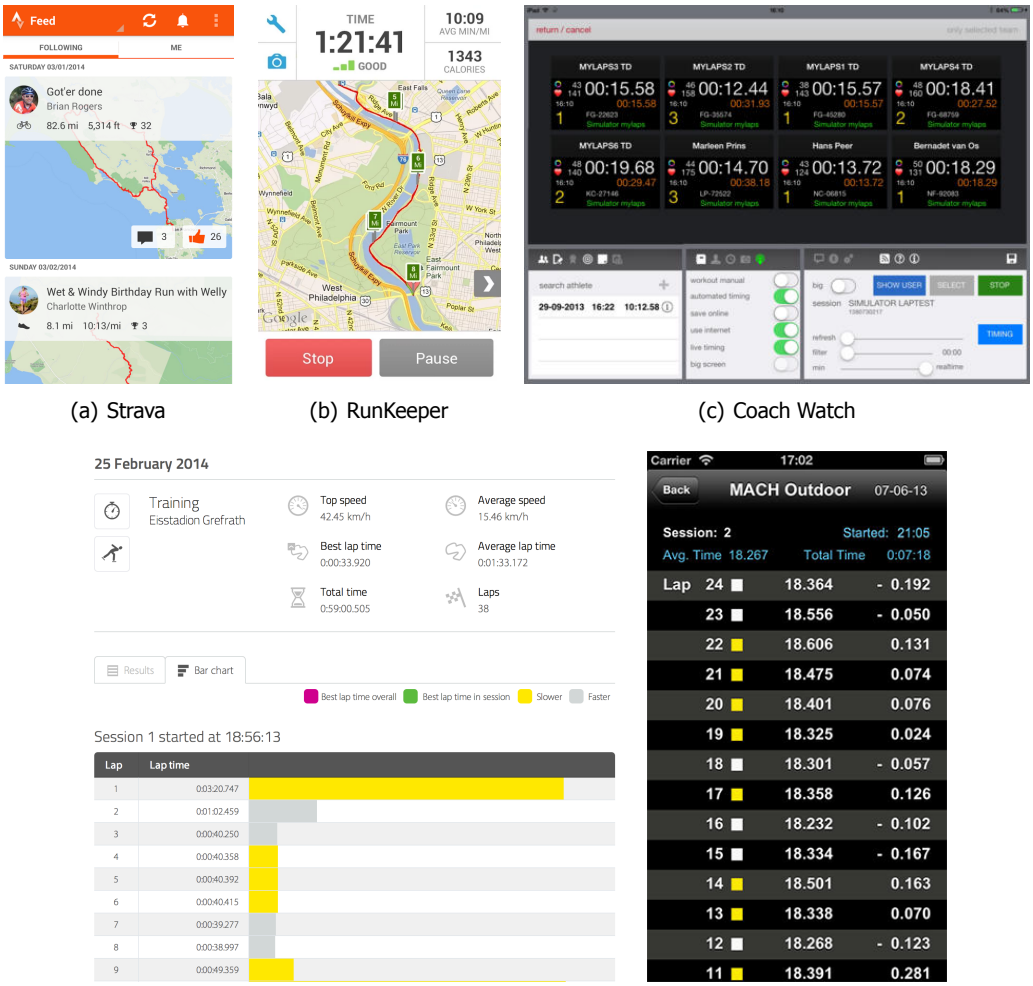
Strava en RunKeeper zijn voor baansporten, zoals bijvoorbeeld schaatsen, niet goed bruikbaar, aangezien GPS op de banen slecht presteert, waardoor de informatie niet nauwkeurig genoeg is. MyLaps Practice en Coach Watch maken wel gebruik van de detectielussen in de banen. De MyLaps Practice website kan alleen gebruikt worden voor het achteraf bekijken van sportprestaties en deze de applicatie is vooral gericht op motorsporten. CoachWatch is een (vrij dure) applicatie die vooral gericht is op coaches en dus minder geschikt voor trainingen of voor amateurschaatsers.

Emando ziet de mogelijkheid om een applicatie te bieden die gebruikmaakt van de detectielussen in de banen, die live gegevens kan tonen op een smartphone en die ook nog eens goedkoop of gratis is. Onze applicatie richt zich dus op het invullen van de stukken die missen in de bestaande oplossingen.

2.4. Programma van eisen

Traditioneel gezien wordt bij software ontwikkeling het MoSCoW [1, 2] model gebruikt. Het MoSCoW model onderscheidt functionaliteiten op basis van prioriteiten. De verschillende niveaus zijn must-, should-, could- en won't-haves. De vertaling van deze niveaus spreekt voor zich.

Wij gebruiken het in combinatie met het in SCRUM veel gebruikte Minimum Viable Product (MVP) model [3]. Hierbij wordt gedefinieerd wat het product minimaal moet kunnen om bruikbaar te zijn. In die zin komt onze MVP dus overeen met de must-haves uit het MoSCoW model. Eventuele tegenslagen mogen er niet toe lijden dat het MVP niet gemaakt wordt, vandaar is de planning om het MVP al in de



(a) Strava (b) RunKeeper (c) Coach Watch

(d) MyLaps Practice (e) MyLaps Practice App

Figuur 2.1: Soortgelijke oplossingen

vierde week van het tien weken durende project af te hebben. Gebruikers kunnen op dat moment met de applicatie spelen en de kernfunctionaliteit beoordelen.

Het MVP biedt op zich zelf nog niet alle features die zowel wij als de opdrachtgever graag geïmplementeerd zouden zien. Het eindproduct zal over enkele bijzondere functies moeten beschikken, de zogenaamde 'killer-features', om het product populair te maken. Deze features zijn in die zin dus vergelijkbaar met de should-haves uit het MoSCoW model.

Na het MVP en de features die het verschil maken zijn er ook nog een aantal features die niet noodzakelijk zijn en geen groot verschil zouden maken. Deze features zijn te vergelijken met could-haves uit het MoSCoW model. We zullen zeker niet alle could-haves implementeren, en wellicht komen enkele should-features ook niet aan bod. Wellicht volgt uit een user study dat de door ons bedachte could-haves door users erg gewenste features zijn. In overleg met onze begeleiders kunnen we besluiten deze features te implementeren. Bovendien bieden de could-haves een goede leidraad bij toekomstige ontwikkeling na afloop van ons project.

Must-haves (MVP)

- Bruikbaar als applicatie op smartphone(s)
- Architectuur is sport-agnostisch
- De sportspecifieke weergave van tijden is voor schaatsbanen geïmplementeerd

- Mogelijkheid om een account aan te maken
- Realtime transponderdoorkomsten tonen van geselecteerde sporters
- Historische doorkomsten van een persoon, gegroepeerd per dag of training
- De voor bovenstaande features ontwikkelde API, moet uitbreidbaar zijn en aangepast kunnen worden.

Should-haves

- Mogelijkheid om account te koppelen aan Facebook
  - Mogelijkheid om prestaties en records te delen op Facebook
- Audio-cues geven aan sporter over doorkomsten van een geselecteerde transponder
- Leaderboards tonen met sporters gerangschikt naar prestaties en disciplines (virtuele competitie), zoals:
  - rondetijd (actuele/gemiddelde/snelste)
  - snelheid (gemiddelde/snelste)
  - cumulatieve afstand (per seizoen/week/training)
  - rust/intensief-ratio (hoeveelheid rondjes, ratio)
  - achievements (verbeter jezelf 10%, voor 7 uur op de baan, 2 trainingen per week)
- Groepen-functionaliteit
  - Mogelijkheid om een lege groep te maken
  - Mogelijkheid om sporters uit te nodigen voor een groep
  - Mogelijkheid om op uitnodiging in een groep te gaan
  - Mogelijkheid om uit een groep te gaan
- Zowel leaderboards als realtime doorkomstschermen kunnen worden gefilterd op groep of baan
- Privacy-instellingen bieden aan gebruikers
  - Mogelijkheid om een account publiekelijk of anoniem te laten indexeren
  - Mogelijkheid om eigen data te delen met anderen

Could-haves

- Suggesties krijgen om met soortgelijke schaatsers te gaan schaatsen
- Training kunnen exporteren naar RunKeeper[1]

Won't-haves

Het door Emando ontwikkelde systeem Vantage zal de huidige systemen voor wedstrijduitslagen vervangen. Koppelingen op het huidige systeem zouden dus snel niet meer werken en het nieuwe systeem is niet af tijdens ons project.

- Wedstrijdloting en uitslagen bekijken
- Persoonlijke records tonen

[1] Fitness logboek platform, http://www.runkeeper.com

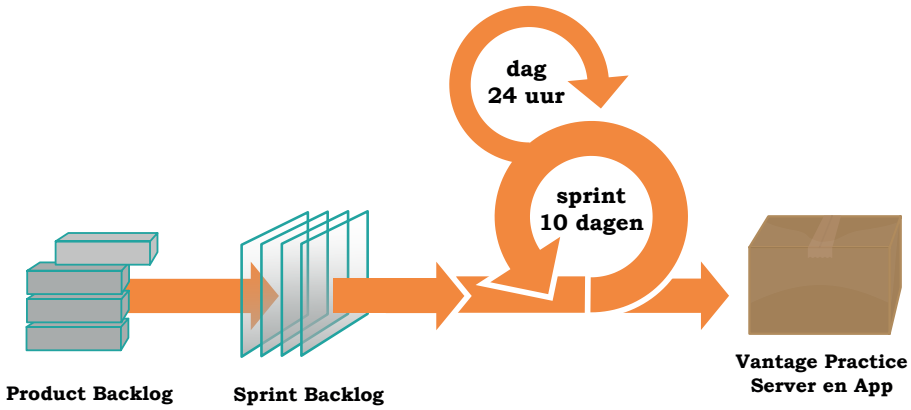
3

Projectmethodologie

Emando zit in het BizSpark[1] programma van Microsoft. Dit programma biedt startups gratis software en ondersteuning om hun startup te laten slagen. Dit heeft tot gevolg dat alle ontwikkeltools van Microsoft gratis gebruikt worden, waardoor het Emando dan ook op Microsoft technologie leunt. Onderstaande methodologieën kunnen eenvoudig geïntegreerd worden in Microsoft's ontwikkelomgeving Visual Studio. Tijdens het bachelorproject is dan ook niet afgeweken van de methodologieën die Emando al gebruikt, omdat ons deze omgeving wordt aangeboden én om onderhoudbaarheid in de toekomst te garanderen.

3.1. Scrum

In overleg met de opdrachtgever is er gekozen voor een Scrum [4] aanpak voor het project. Scrum is een flexibele manier om software te ontwikkelen, zie ook figuur 3.1. Hierbij wordt gewerkt met een multidisciplinair team dat in korte sprints met een vaste lengte (in dit geval tien werkdagen), werkende software oplevert en geleidelijk stabiele functionaliteit toevoegt. Een Scrum aanpak heeft als voordeel dat er na iedere sprint een werkend product is, waarna de eisen en doelstellingen gemakkelijk bijgesteld kunnen worden. Met Scrum kunnen na de gebruikerstest, halverwege het project, de doelstellingen bijgesteld worden aan de hand van de uitkomst van deze test. Door deze agile [5] methode toe te passen sluit het eindproduct altijd zo goed mogelijk aan op de wensen van de opdrachtgever en de end-users.



Figuur 3.1: Overzicht van het Scrum proces

[1] http://www.microsoft.com/bizspark/

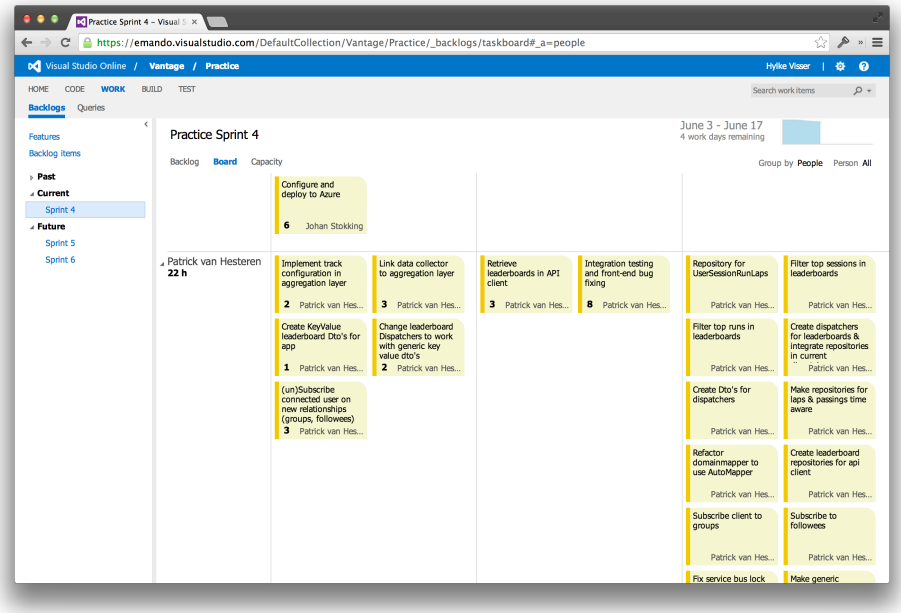
Aan het begin van het project is een lijst met product backlog items opgesteld. Deze items representeren de globale functionaliteiten van de applicatie en worden aan een teamlid toegewezen dat daarvoor verantwoordelijk is. Voor elke sprint wordt bepaald welke van deze backlog items er in de tien dagen durende sprint geïmplementeerd zullen worden. Elk backlog item bevat een aantal taken die voltooid moeten worden om het backlog item af te krijgen. Aan deze taken wordt een aantal uren toegekend en vervolgens worden de taken verdeeld over de drie teamleden, afhankelijk van ieders expertise en de hoeveelheid beschikbare uren.

3.2. Stand up meetings

Om het overzicht binnen het project te waarborgen, zullen er aan het begin van iedere werkdag stand up meetings met alle projectleden plaatsvinden. Bij deze stand up meetings wordt kort besproken waar momenteel aan gewerkt wordt, in hoeverre dit op schema ligt en waar de uitdagingen liggen voor de komende dagen. De opdrachtgever is ook minstens twee keer per week aanwezig bij deze meetings.

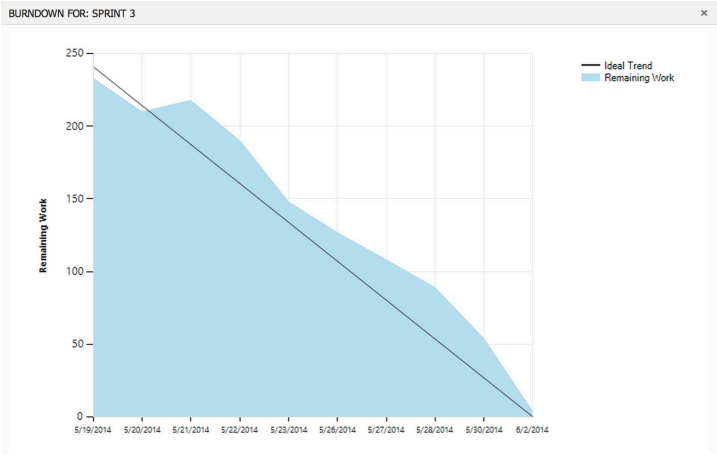
3.3. Team Foundation Server

Emando gebruikt Team Foundation Server (TFS) voor het versiebeheer van broncode, het bijhouden van het ontwikkelproces, het tracken van issues en het visualiseren van de voortgang in projecten. De gemakkelijke integratie van TFS met Visual Studio zorgt ervoor dat taken gemakkelijk toegewezen kunnen worden aan teamleden, waarna deze geopend kunnen worden en hiermee aan de slag kunnen binnen Visual Studio. De status van deze items kan zowel vanuit Visual Studio als online aangepast worden en bij het afvinken van items kunnen deze gekoppeld worden aan versienummers, zodat deze achteraf makkelijk vindbaar zijn. Daarnaast bestaat er een handig overzicht, te zien in figuur 3.2, waarin in één oogopslag de status van alle taken bekeken kan worden.



Figuur 3.2: Overzicht van de activiteiten op het Scrum Board in TFS

De visualisatie van de zogenaamde burndown van het project, zie figuur 3.3, garandeert het inzicht van de opdrachtgever. De voortgang van het project kan namelijk in één oogopslag gevolgd worden. Daarnaast biedt TFS inzicht in de tijd die ieder issue en backlog item kost, hetgeen een goede indicatie is voor de scrum-sprint.



Figuur 3.3: Burndown grafiek van Sprint 3 in TFS

3.3.1. Verantwoordelijkheden

Bij de taakverdeling hebben wij er voor gekozen om aan ieder backlog item een verantwoordelijke toe te kennen. Door de verantwoordelijkheden duidelijk te maken binnen het project, is het altijd duidelijk wie van welke functionaliteit kennis heeft en weet hoe de voortgang hiervan is. Hiermee is het ook mogelijk om slechts met een deel van de teamleaden en de opdrachtgever implementatiedetails door te spreken. Binnen TFS is het mogelijk om de verantwoordelijkheden van taken en verantwoordelijkheden te verdelen over verschillende personen.

3.3.2. Taakverdeling

Om overlap van werk en functionaliteiten te voorkomen hebben we er voor gekozen om iedere functionaliteit uit te splitsen in verschillende backlog items en deze op hun beurt weer uit te splitsen in verschillende taken. Door deze taken klein te houden, is het makkelijker om de precieze invulling van functionaliteiten te definiëren. Daarnaast heeft dit als voordeel dat kleine taken sneller af zijn dan grote taken, waardoor de burndown van het project accurater is.

In grove lijnen ziet de verantwoordelijkheid en taakverdeling er als volgt uit:

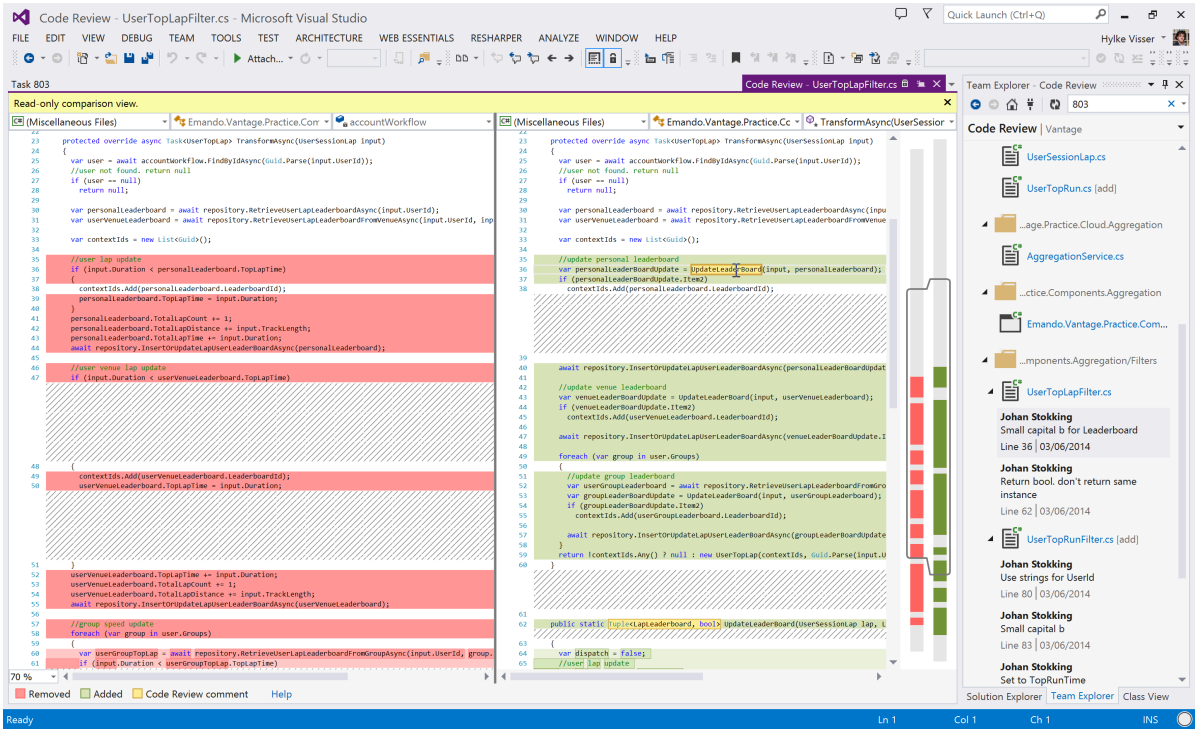
1. Patrick van Hesteren: Back-end en Cloud
2. Hylke Visser: Grafische User Interface en Testen
3. Herman Banken: Applicatie en SignalR

3.4. Code reviews

Om de kwaliteit van het project te kunnen waarborgen, is voor de back-end van het project gekozen om met code reviews te werken. Omdat de back-end de core van het project vormt en tevens in de cloud draait - waardoor er directe kosten zijn verbonden aan de kwaliteit van de code - is het belangrijk dat deze efficiënt en naar behoren werkt. Bij het implementeren van nieuwe of aanpassen van bestaande functionaliteit kunnen binnen Visual Studio Code Reviews aangevraagd worden aan projectleden. In ons geval is onze opdrachtgever de persoon geweest die de Code Reviews gedaan heeft. Bij zo'n Code Review, te zien in figuur 3.4, worden de wijzigingen in code gecontroleerd op fouten. Er kan feedback geplaatst worden bij deze wijzigingen, zodat de ontwikkelaar makkelijk kan zien welk deel van de code verbeterd dient te worden.

3.5. Projectkeuzes

Qua talen, frameworks en libraries moesten al vroeg enkele afwegingen gemaakt worden. Uiteindelijk hebben we gekozen om te werken met onder andere C#, Xamarin en SignalR. De redenatie achter



Figuur 3.4: Code Review in Visual Studio

deze keuzes is te lezen in het oriëntatieverslag (sectie C.4 en C.5).

4

Systeemontwerp en -implementatie

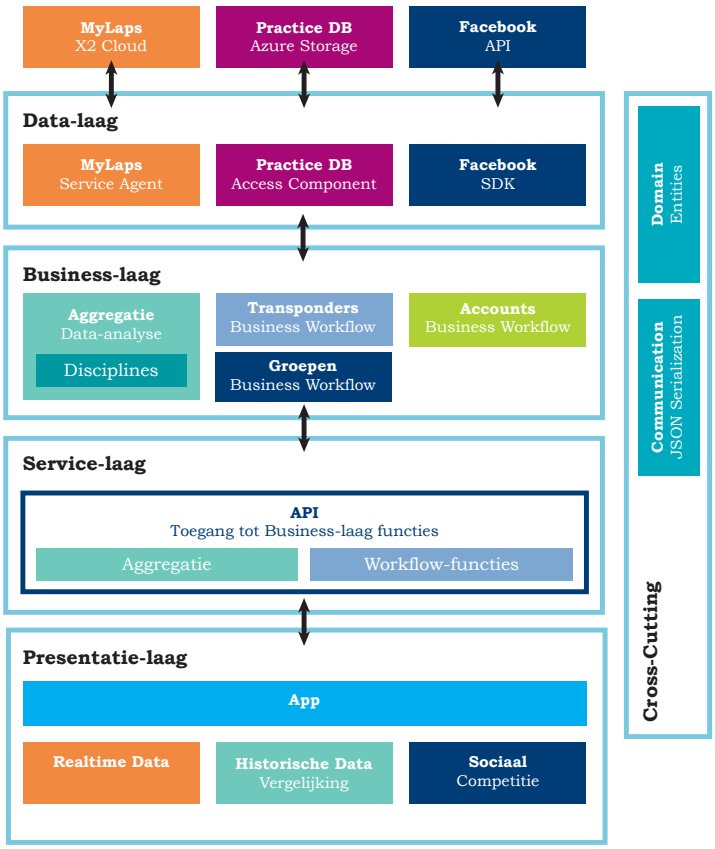
De applicaties die Emando ontwikkelt worden opgedeeld in vier lagen. Deze opzet zullen wij ook aanhouden.

De presentatielaag verzorgt de weergave. De presentatielaag bevindt zich in de code van de mobiele applicatie en bevat zowel platform-onafhankelijke View-Models en hulpmiddelen als platformspecifieke User Interface elementen.

De servicelaag is de interface tussen de server en de mobiele apps. Deze gebruikt de SignalR- en WebApi-technieken.

De businesslaag is verantwoordelijk voor het ophalen en verwerken van de data. Deze laag bevat een verbinding met de service- en datalaag. Met deze verbindingen is deze laag is staat met behulp van logica data op te halen, te verwerken, eventueel op te slaan en door te sturen naar de servicelaag.

De datalaag is verantwoordelijk voor alle verschillende databronnen: Entity Framework, Table Storage en MyLaps. Deze laag bevat zelf geen verbindingen met andere lagen, maar de data kan met behulp van repositories vanuit de businesslaag opgevraagd worden.



Figuur 4.1: Diagram van de lagen van de applicatie

Naast deze vier lagen is er ook nog een deel van de code die gedeeld wordt door alle andere lagen: **"cross cutting"**. In de crosscutting-laag bevinden zich voornamelijk de domein-entiteiten, zodat alle serverlagen deze entiteiten kunnen gebruiken. Naast de entiteiten wordt ook de JSON serialisatie van de entiteiten in de cross cutting laag geregeld: er zijn speciale te serialiseren afgeleide modellen,

met zo weinig afhankelijkheden van frameworks dat ze in een Portable Class Library (PCL) kunnen. Hierdoor kunnen de mobiele applicaties dezelfde modellen gebruiken als de servicelaag, en kunnen de modellen via de Websockets- en HTTP-verbinding van SignalR en WebAPI verstuurd worden. Deze entiteiten bevatten allen een context Id en user Id, welke aangeeft voor welke context (groep, baan of gebruiker) bedoelt is en van wie deze entiteit is. Op deze manier is SignalR in staat om deze entiteiten te versturen naar de juiste gebruikers.

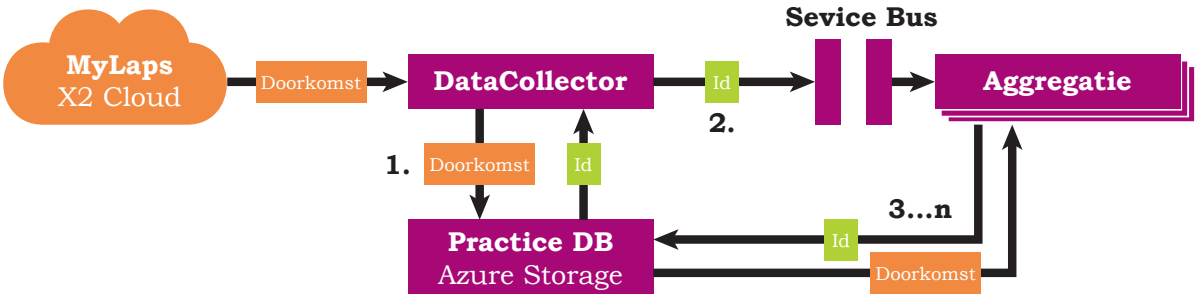
4.1. Datalaag

De applicatie bevat data uit verschillende bronnen, en slaat zijn data in verschillende bronnen op. Deze data en de verbindingen met de bronnen vormen de datalaag.

4.1.1. Ruwe inkomende data

De transponderleverancier MyLaps heeft een SDK beschikbaar gesteld aan Emando, waarmee het mogelijk is om de doorkomsten van transponders realtime binnen te krijgen. Om deze databron te benaderen bevat de datalaag een Data Collector, geïmplementeerd als Azure Cloud Worker Role, welke non-stop draait op één cloud instance. Dit proces onderhoudt de verbinding met de MyLaps SDK.

Vanuit de MyLaps SDK ontvangt de data laag realtime de doorkomsten van iedere transponder die langs een lus komt. Deze doorkomsten bevatten een tijdstip, een transpondernummer en een plaats op de baan. Zodra een van deze doorkomsten binnen komt, wordt deze direct opgeslagen voor toekomstig gebruik in een doorkomstentabel in Azure Table Storage. Het kan namelijk voorkomen dat de transponder van de zojuist binnengekomen doorkomst nog niet gekoppeld is aan een gebruikersaccount. Wanneer dit het geval is, kunnen er (nog) geen aggregaties uitgevoerd worden op deze data.



Figuur 4.2: De werking van de Data Collector en de Service Bus

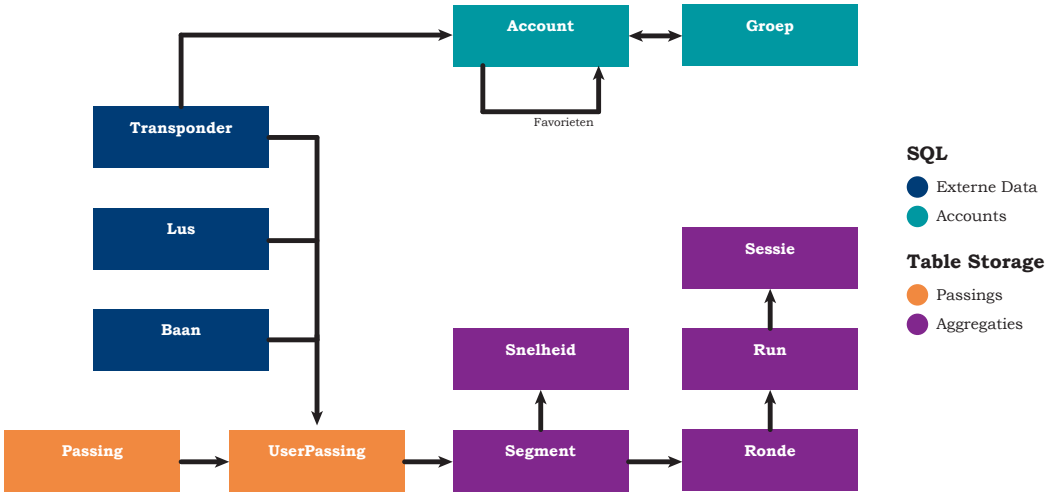
Door deze data op te slaan, is het mogelijk om later functionaliteit in te bouwen om alsnog over deze data te kunnen aggregeren.

Nadat de data opgeslagen is, wordt via een Azure Service Bus het Id van de doorkomst doorgestuurt naar het aggregatieproces in de businesslaag. Het aggregatieproces raadpleegt bij het binnenkrijgen van dit Id opnieuw de doorkomst, om zo minder data over de zogenaamde service bus te versturen en altijd over up to date data te beschikken. Dit process wordt weergegeven in figuur 4.2.

4.1.2. Entiteiten

In figuur 4.3 wordt een deel van de entiteiten die de applicatie gebruikt getoond, inclusief de onderlinge relaties. De entiteiten die te maken hebben met externe data en accounts worden opgeslagen met behulp van Azure SQL, zoals ook besproken in sectie C.7 van het oriëntatieverslag. De entiteiten die te maken hebben met doorkomsten en aggregaties worden opgeslagen in Azure Table Storage.

Hoewel de daadwerkelijke structuur van de data in Azure Table Storage anders is dan in de figuur, geeft de figuur toch een goed beeld van deze entiteiten en hun relaties.



Figuur 4.3: De entiteiten en hun relaties.

4.1.3. Azure SQL & Entity Framework

Azure SQL is de relationele SQL database van Azure en omdat de relaties in onze entiteiten worden beschreven door het Entity Framework [6, 7], gebruikt onze applicatie de Entity Framework driver voor Azure SQL. Entity Framework is de de facto standaard voor relationele datalagen in ASP.NET. Het kan bijvoorbeeld relaties tussen entiteiten automatisch opzoeken. Ook hoeft geen SQL code geschreven te worden. Om entiteiten op te kan kan platformonafhankelijke LINQ code geschreven worden.

4.1.4. Azure Table Storage

De geaggregeerde data kan "plat" worden opgeslagen met behulp van NoSQL. Door alleen passings (doorkomsten), rondes en leaderboard-waarden entiteiten op te slaan in de aggregatiedatabase, hoeven onderlinge relaties niet bijgehouden te worden en kunnen gegevens goedkoop opgevraagd worden. Gegevens zoals "rust/run"-ronde en sessies zijn onderdeel van rondes. De snelheid en segmenten worden niet opgeslagen, maar direct verstuurd naar de applicatie.

Vanuit Emando ging de voorkeur voor het opslaan van de geaggregeerde data uit naar de NoSQL Azure Table Storage. De Azure Table Storage integreert eenvoudig met de reeds bestaande infrastructuur van het bedrijf. Bovendien biedt een gepartitioneerde en gesorteerde Azure Table Storage tabel een groot performancevoordeel: het goedkoop en snel kunnen selecteren van de laatste entiteiten per context.

Bij het gebruik van NoSQL is het belangrijk dat de Partition- en Row Key efficiënt gekozen worden, aangezien dit de snelheid van het zoeken bepaalt. Verder is het belangrijk dat de combinatie van Partition en Row Key uniek is. In figuren 4.4, 4.5, 4.6 en 4.7 is te zien hoe deze keys gekozen zijn en wat de structuur van deze tabellen is.

Vanuit de Data Collector worden de doorkomsten die realtime binnen komen opgeslagen in Azure Table Storage als backup. De structuur van deze tabel is terug te vinden in figuur 4.4. Verderop in het aggregatieproces worden deze doorkomsten gekoppeld aan gebruikers. Deze aggregaties worden vervolgens opgeslagen in een aparte tabel binnen Azure Table Storage. De structuur van deze tabel is terug te vinden in figuur 4.5.

Verderop in het aggregatieproces worden doorkomsten van gebruikers gekoppeld aan sessies, rondes en run/rust periodes. Deze aggregaties worden opgeslagen in de rondetabel binnen Azure Table Storage. De structuur hiervan is terug te vinden in figuur 4.6.

Van snelheden, rondes, run/rust periodes en sessies worden leaderboards bijgehouden in Azure Table Storage. Deze hebben elk een eigen corresponderende tabel, zodat aggregaties parallel aan elkaar leaderboards kunnen updaten. Alle leaderboards zijn voorzien van een leaderboard Id en een

user Id. Het leaderboard Id houdt bij voor welke context het leaderboard geldig is. Dit kunnen banen, groepen, maar ook gebruikers zijn. Wanneer een gebruiker een ronde rijdt op een baan X, moeten zijn records voor de baan, zijn groepen en zijn eigen totalen geüpdatet worden. Een voorbeeldstructuur van deze leaderboards is te vinden in figuur 4.7.

PartitionKey	RowKey	Timestamp	What	Strength	IsFinisl	ApplianceName	Applianc	ApplianceEventIc	How	Wh	When	Sent	IsResen	Received
T_MYLAPS Proc...	2014-06-08T15:...	6/8/2014 13:57:36	-1081215809	-96.5327125725...	False	MYLAPS Gener...	Thialf	267336	Transponder	20	6/8/2014 13:57:16	6/8/2014 13:57:17	False	6/8/2014 13:57:35
T_MYLAPS Proc...	2014-06-08T15:...	6/8/2014 13:57:37	-1081215809	48.5025853842975	False	MYLAPS Gener...	Thialf	992512	Transponder	0	6/8/2014 13:57:23	6/8/2014 13:57:24	False	6/8/2014 13:57:37
T_MYLAPS Proc...	2014-06-08T15:...	6/8/2014 13:57:38	-1081215809	-33.7105193565...	True	MYLAPS Gener...	Thialf	581447	Transponder	1	6/8/2014 13:57:30	6/8/2014 13:57:31	False	6/8/2014 13:57:38

Figuur 4.4: De doorkomstentabel met voorbeeld records. Doorkomsten bevatten 2 kernelementen, waar op gefilterd wordt: e transponder (bestaande uit een type en een nummer) en het tijdstip waarin de doorkomst binnen kwam. Door de keys als volgt in te richten kan daarom efficiënt gezocht worden binnen de data: de Partition Key is het type transponder + transpondernummer en de Row Key het tijdstip waarop de doorkomst binnen kwam.

PartitionKey	RowKey	Timestamp	UserId	VenueId	TrackLength	SessionId	IsFinish	When	Where
U_a56245ea-2c72-4aa2-9c2c-146bad34565d	2014-06-06T13:21:03.5463526Z	6/6/2014 13:21:08	a56245ea-2c72-...	f669f93f-16ce-4...	400	90cbff75-7bf6-4...	True	6/6/2014 13:21:03	1
U_a56245ea-2c72-4aa2-9c2c-146bad34565d	2014-06-06T13:21:05.9031299Z	6/6/2014 13:21:08	a56245ea-2c72-...	f669f93f-16ce-4...	400	90cbff75-7bf6-4...	True	6/6/2014 13:21:05	2
U_a56245ea-2c72-4aa2-9c2c-146bad34565d	2014-06-06T13:21:06.0161388Z	6/6/2014 13:21:27	a56245ea-2c72-...	f669f93f-16ce-4...	400	90cbff75-7bf6-4...	True	6/6/2014 13:21:06	0
U_a56245ea-2c72-4aa2-9c2c-146bad34565d	2014-06-06T13:21:07.1410603Z	6/6/2014 13:21:37	a56245ea-2c72-...	f669f93f-16ce-4...	400	90cbff75-7bf6-4...	True	6/6/2014 13:21:07	0

Figuur 4.5: De doorkomstentabel met voorbeeldrecords. Gebruikers doorkomsten bevatten 2 kernelementen, waar op gefilterd wordt: de gebruiker en het tijdstip waarin de doorkomst binnen kwam. Door de keys als volgt in te richten kan daarom efficiënt gezocht worden binnen de data: de Partition Key is het account Id met een prefix voor gebruikers en de Row Key het tijdstip waarop de doorkomst binnen kwam.

PartitionKey	RowKey	Timestamp	UserId	VenueId	TrackLength	SessionId	LapId	StartTime	EndTime	IsRest
a56245ea-2c72-...	a7280855-7801-...	6/8/2014 13:57:44	a56245ea-2c72-...	f669f93f-16ce-4...	400	57669ef2-ed3e-...	a7280855-7801-...	6/8/2014 13:57:34	6/8/2014 13:57:34	False
a56245ea-2c72-...	a77e1bfc-3f0a-...	6/6/2014 13:21:13	a56245ea-2c72-...	f669f93f-16ce-4...	400	90cbff75-7bf6-4...	a77e1bfc-3f0a-...	6/6/2014 13:21:03	6/6/2014 13:21:05	False
a56245ea-2c72-...	a97b6970-78dd...	6/8/2014 13:59:03	a56245ea-2c72-...	f669f93f-16ce-4...	400	57669ef2-ed3e-...	a97b6970-78dd...	6/8/2014 13:59:01	6/8/2014 13:59:01	False
a56245ea-2c72-...	b00429b1-e863-...	6/8/2014 13:57:52	a56245ea-2c72-...	f669f93f-16ce-4...	400	57669ef2-ed3e-...	b00429b1-e863-...	6/8/2014 13:57:37	6/8/2014 13:57:37	False
a56245ea-2c72-...	b0e65dc8-fff6-4...	6/8/2014 13:57:48	a56245ea-2c72-...	f669f93f-16ce-4...	400	57669ef2-ed3e-...	b0e65dc8-fff6-4...	6/8/2014 13:57:35	6/8/2014 13:57:35	False

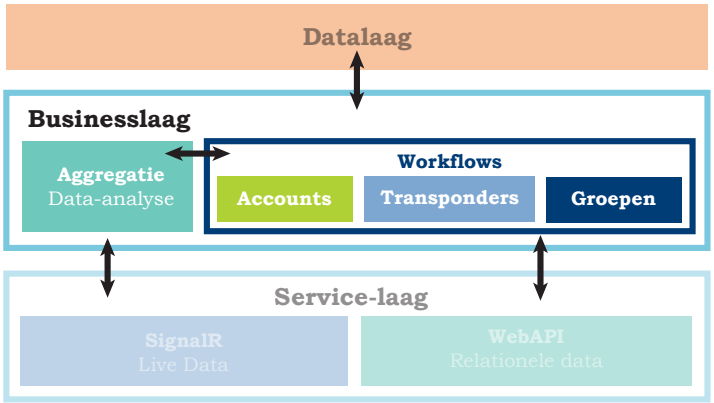
Figuur 4.6: De rondetabel met voorbeeldrecords. Gebruikersdoorkomsten bevatten 2 kernelementen, waar op gefilterd wordt: de gebruiker en het rondenummer zelf. Door de keys als volgt in te richten kan daarom efficiënt gezocht worden binnen de data: de Partition Key is het account Id en de Row Key het ronde Id.

PartitionKey	RowKey	Timestamp	UserId	LeaderboardId	TotalLapTime	TotalLapDistance	TopLapTime	TotalLapCount
a56245ea-2c72-4aa2-9c2c-146bad34565d	a56245ea-2c72-4aa2-9c2c-146bad34565d	6/8/2014 13:59:11	a56245ea-2c72-...	a56245ea-2c72-4aa2-9c2c-146bad34565d	3312996310	15200	1130089	38
ab32fbdd-3972-452e-b12e-d4f5e59be025	a56245ea-2c72-4aa2-9c2c-146bad34565d	6/8/2014 13:59:11	a56245ea-2c72-...	ab32fbdd-3972-452e-b12e-d4f5e59be025	3312996310	15200	1130089	38
f669f93f-16ce-43dd-99f4-2785445fed3e	a56245ea-2c72-4aa2-9c2c-146bad34565d	6/8/2014 13:59:11	a56245ea-2c72-...	f669f93f-16ce-43dd-99f4-2785445fed3e	3312996310	15200	1130089	38

Figuur 4.7: De ronde-leaderboardtabel, een voorbeeld van een leaderboardtabel. Leaderboards bevatten 2 kernelementen, waar op gefilterd wordt: de gebruiker en de context waar het leaderboard voor is (groep, baan of gebruiker). Door de keys als volgt in te richten kan daarom efficiënt gezocht worden binnen de data: de Partition Key het account Id en de Row Key het context Id (groep, baan of account Id).

4.2. Businesslaag

De businesslaag bestaat uit meerdere grote onderdelen. De belangrijkste onderdelen van de businesslaag zijn de aggregatie en de workflows.



Figuur 4.8: Opbouw en interactie van de businesslaag

De businesslaag is de plek waar de communicatie met de datalaag en de servicelaag plaats vindt. Vanwege de keuze om de data realtime door te geven aan de applicatie is het belangrijk dat deze laag snel en vloeiend werkt, ook bij een hoge load. Om deze reden hebben we gekozen voor een Cloud-oplossing in Windows Azure, welke draait op een of meerdere Cloud instances. Bij een hogere load worden er automatisch voldoende extra instances aangemaakt en wordt de load evenwichtig verdeeld. Tevens biedt een Cloud-oplossing de mogelijkheid om verbeteringen aan bestaande of juist nieuwe functionaliteit toe te voegen aan de back-end, zonder dat de applicatie hierbij een update vereist.

Hoewel er meerdere soortgelijke Cloud-oplossingen bestaan, ging vanwege de bestaande infrastructuur van Emando de voorkeur uit naar Windows Azure. Een bijkomend voordeel van Windows Azure is de eenvoudige integratie met Visual Studio, waardoor het deployen naar de cloud- of testomgeving gemakkelijk is.

4.2.1. Aggregatiestructuur

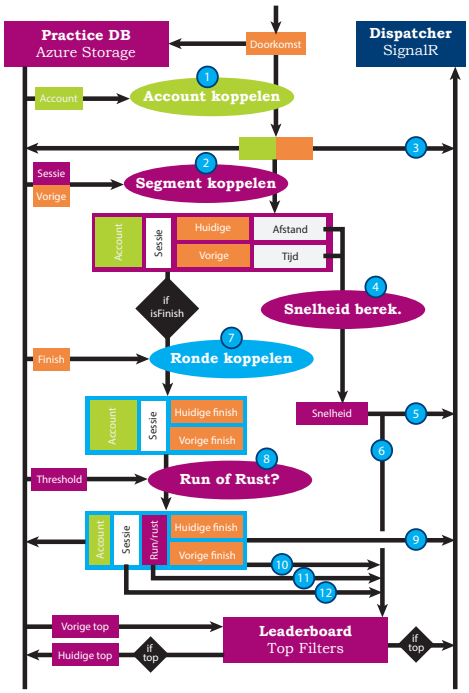
Omdat de ruwe doorkomsten niet direct betekenis hebben voor gebruikers, dient deze data verrijkt te worden om zo inzicht te geven in trainingen van gebruikers. Dit proces noemen we de aggregatie van data.

Zodra een doorkomst de datalaag binnenkomt, wordt het Id hiervan naar de aggregatielaag overgestuurd via de Azure Service Bus. Zodra de aggregatielaag deze ontvangt, wordt de bijbehorende doorkomst opgevraagd. Hierna start het aggregatieproces, dat bestaat uit de volgende onderdelen, zoals ook te zien is in figuur 4.9:

- **Reguliere filters.** Deze filters vergelijken reeds bestaande geaggregeerde en ruwe data met de nieuwste binnenkomst, om zo nieuwe data te aggregeren. Deze data wordt vervolgens doorgegeven aan de dispatchers en top filters. Voorbeeld: Het opmaken van rondetijden aan de hand van verschillende doorkomsten op een baan.
- **Top filters.** Deze filters vergelijken reeds bestaande geaggregeerde data met de zojuist geaggregeerde data, om zo te kijken of de zojuist geaggregeerde data een record bevat voor de gebruiker. Deze filters verwerken de nieuwe records in de leaderboards voor een gebruiker op gebruikers-, baan- en groepsniveau. Vervolgens worden deze leaderboards doorgegeven aan de dispatchers. Voorbeeld: Een gebruiker heeft een nieuwe ronde geschaatst, waarin zijn rondetijd verbeterd is, maar ook zijn totale afstand op de baan toegenomen is.
- **Dispatchers.** De dispatchers ontvangen alle geaggregeerde data en zijn verantwoordelijk voor het doorgeven van deze data aan de juiste applicaties. De dispatchers zorgen er voor dat deze

data realtime binnen komt bij alle geïnteresseerden van deze updates. Voorbeeld: Live doorkomsten van gebruikers van een groep/baan.

De filters zijn opgebouwd volgens een pipeline-structuur. Na het uitvoeren van een filter worden de filters aangeroepen die de filteruitvoer als invoer verwachten. Als een filter geen aggregatie uit kan voeren, stopt de aggregatie bij dit filter, aangezien er zonder de uitvoer van dit filter, geen andere filters aangeroepen kunnen worden.



Figuur 4.9: Flow-diagram van het aggregatieproces
N.B. Deze afbeelding is in groot formaat opgenomen in Appendix F op pagina 62.

1. **Account koppelen aan doorkomsten.**

   Allereerst wordt bij een doorkomst aan de hand van het transponder nummer een account gezocht. Als er een bestaande gebruiker is die dit transpondernummer momenteel aan zijn account gekoppeld heeft, wordt dit account gekoppeld aan deze doorkomst. Dit wordt opgeslagen in een aparte tabel in Azure Table Storage.

2. **Doorkomsten koppelen aan segmenten.** Een doorkomst maakt deel uit van een training, soms gekoppeld aan een trainingsschema. Een training kan bestaan uit een of meerdere sessies. Een segment van doorkomsten houdt in dat er twee doorkomsten van dezelfde gebruiker binnen 30 minuten hebben plaatsgevonden op dezelfde baan. Wanneer de vorige doorkomst van deze gebruiker en de huidige doorkomst binnen een segment vallen, vallen deze doorkomsten onder dezelfde trainingssessie. Wanneer dit het geval is, krijgt de doorkomst dezelfde sessie Id toegewezen als zijn vorige doorkomst. Wanneer dit niet het geval is of wanneer er geen vorige doorkomst is, kan er geen segment gevonden worden.

3. **Dispatchen van live doorkomsten.** Zodra er een gebruiker gevonden is bij een doorkomst, moet deze doorgegeven worden aan alle "geïnteresseerden": de gebruiker zelf, gebruikers op dezelfde baan, leden van dezelfde groep als deze gebruiker en alle gebruikers die geabonneerd zijn op deze gebruiker.

   Met behulp van SignalR wordt de doorkomst doorgegeven aan de juiste personen.

4. **Snelheid berekenen.** Tussen twee doorkomsten in een segment ligt een bepaalde afstand. Een gebruiker heeft deze afstand afgelegd in een tijdsinterval. Aan de hand van deze gegevens wordt de snelheid binnen dit segment berekend.

5. **Dispatchen van snelheden.** Zodra er een snelheid is berekend voor een gebruiker op een bepaald segment, moet deze doorgegeven worden aan alle eerder genoemde geïnteresseerden.

   Met behulp van SignalR wordt de doorkomst doorgegeven aan de juiste personen.

6. **Snelheidsleaderboard updaten en dispatchen.** Zodra er een snelheid berekend is over een bepaald segment, kan dit gevolgen hebben voor het snelheidsleaderboard. De gereden snelheid wordt vergeleken met de eerdere persoonlijke records van de gebruiker, zijn records op deze baan en zijn records in al zijn groepen. Daarnaast wordt altijd de gereden afstand op de baan, tijdsinterval etc. bijgewerkt in de totalen binnen dit leaderboard. Wanneer de zojuist gereden snelheid een record blijkt te zijn voor een van zojuist genoemde contexten, wordt deze up to date gebracht in de leaderboards. Daarna wordt het gehele leaderboard voor deze context via SignalR naar alle geïnteresseerden verstuurd.

7. **Doorkomsten koppelen aan rondes.** De gevonden sessie bij het segment kan gebruikt worden om rondes te vinden. Wanneer de huidige doorkomst afkomstig is van de finish-lus en de gebruiker in de huidige sessie een eerdere doorkomst op de finish-lus heeft, is er sprake van een ronde in een training. Wanneer de sessie Ids niet overeenkomen, of de huidige doorkomst niet op de finish lijn ligt is het niet mogelijk om een ronde te vinden.

8. **Filteren van run- en rustperiodes.** Afhankelijk van het type sporter, wordt er al dan niet getraind aan de hand van een trainingsschema. Deze schema's bestaan vaak uit run- en rustperiodes waarin respectievelijk hard getraind wordt of langzaam wordt uitgereden. Omdat de lengte van rondes kan variëren op verschillende banen, wordt niet aan de hand van tijden, maar snelheden gefilterd. Wanneer de gemiddelde snelheid in een ronde hoger is dan een vooraf ingestelde snelheid, wordt deze ronde beschouwd als een run-ronde. Anders wordt deze beschouwd als onderdeel van een rustperiode. Nu alle ronde data compleet is, wordt deze ronde opgeslagen in een aparte tabel in Azure Table Storage voor latere aggregaties.

9. **Dispatchen van rondes.** Zodra er een ronde is berekend voor een gebruiker, wordt deze opnieuw met behulp van SignalR naar de eerder genoemde geïnteresseerden verstuurd.

10. **Rondeleaderboard updaten en dispatchen.** Zodra er een ronde is gevonden voor een gebruiker, kan dit gevolgen hebben voor het rondeleaderboard. De gereden ronde wordt vergeleken met de persoonlijke records voor een gebruiker, zijn records op deze baan en zijn records in al zijn groepen. Daarnaast wordt altijd de gereden afstand op de baan, tijdsinterval etc. bijgewerkt in de totalen binnen dit leaderboard. Wanneer de zojuist gereden rondetijd een record blijkt te zijn voor een van zojuist genoemde contexten, wordt deze up to date gebracht in de leaderboards. Daarna wordt het gehele leaderboard voor deze context naar alle geïnteresseerden gestuurd via SignalR.

11. **Run/rustleaderboard updaten en dispatchen.** Zodra er een ronde is gevonden voor een gebruiker, kan dit gevolgen hebben voor het run/rustleaderboard. De gereden rondetijd wordt toegevoegd aan de desbetreffende run/rustperiode en aan de hand hiervan wordt het totaal hiervan vergeleken met de persoonlijke records voor een gebruiker, zijn records op deze baan en zijn records in al zijn groepen. Daarnaast wordt altijd de gereden afstand op de baan, tijdsinterval etc. bijgewerkt in de totalen binnen dit leaderboard. Wanneer de zojuist gereden rondetijd een record blijkt te zijn voor een van zojuist genoemde contexten, wordt deze up to date gebracht in de leaderboards. Daarna wordt het gehele leaderboard voor deze context naar alle geïnteresseerden gestuurd via SignalR.

12. **Sessie leaderboard updaten en dispatchen.** Zodra er een ronde is gevonden voor een gebruiker, kan dit gevolgen hebben voor het sessie leaderboard. De gereden rondetijd wordt toegevoegd aan de desbetreffende sessie en aan de hand hiervan wordt het totaal hiervan vergeleken met de persoonlijke records voor een gebruiker, zijn records op deze baan en zijn records in al zijn groepen. Daarnaast wordt altijd de gereden afstand op de baan, tijdsinterval etc. bijgewerkt in de totalen binnen dit leaderboard. Wanneer de zojuist gereden rondetijd een record blijkt te zijn voor een van zojuist genoemde contexten, wordt deze up to date gebracht in de leaderboards. Daarna wordt het gehele leaderboard voor deze context naar alle geïnteresseerden gestuurd via SignalR.

4.2.2. Dataconsistentie en parallellisering

Vanwege de meeschalende Cloud-oplossing, kan het voorkomen dat aggregaties van één persoon over meerdere instances berekend worden. Het is belangrijk dat deze aggregaties parallel lopen om gebruikers van realtime updates te kunnen voorzien. Dit brengt echter wel het probleem met zich mee dat sommige data afhankelijk is van andere data en dat deze data consistent moet blijven met elkaar. De eerder genoemde pipeline-architectuur zorgt er voor dat bij ieder filter bekend is welke afhankelijkheden hierop berusten. Zodra het filter klaar is met zijn aggregatie, wordt gekeken of de aggregatie bruikbaar is voor zijn afhankelijkheden en geeft zijn aggregatie aan de hand hiervan door. Deze afhankelijkheden kunnen op hun beurt over verschillende instances verdeeld worden, zonder dat er opstoppingen ontstaan.

Omdat onze aggregatielaag berust op de MyLaps SDK en een service bus, kan het in theorie voorkomen dat een eerdere doorkomst pas later bij deze laag binnenkomt dan een latere doorkomst. Ook kan het voorkomen dat het berekenen van een aggregatie van een eerdere doorkomst langer duurt dan die van een latere doorkomst, waardoor de updates hiervan door elkaar heen gaan lopen. Dit is te zien in figuur 4.10. In deze figuur is te zien hoe doorkomst 3 later gereden wordt, maar eerder geaggregeerd wordt dan doorkomst 2 welke eerder gereden is. Doordat de aggregatie werkt met de vorige doorkomst (doorkomst 1) wordt doorkomst 3 gekoppeld aan doorkomst 1 in een segment waarover de snelheid berekend wordt. Hieruit komt een correcte waarde.

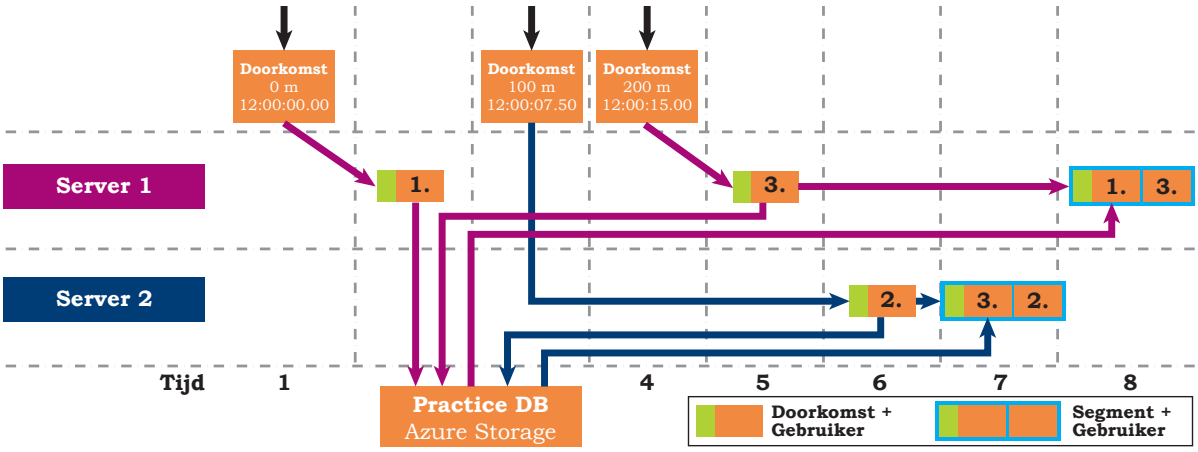
Wanneer doorkomst 2 echter geaggregeerd wordt, vraagt deze de vorige doorkomst op. Omdat doorkomst 3 al geaggregeerd was, is doorkomst 3 zijn vorige doorkomst. Hierbij wordt doorkomst 2 gekoppeld aan doorkomst 3. In dit segment blijkt nu -100 meter gereden te zijn. Er zou nu een verkeerde aggregatie ontstaan!

Daarom is het belangrijk dat de data te allen tijde consistent is, ook over meerdere Cloud instances. Om deze reden zijn alle repositories calls voorzien van een synchronisatietijdstip. Dit is het tijdstip waarop de doorkomst doorgekomen is. Deze calls geven daarmee alleen (aggregatie)data terug die op het moment van de doorkomst beschikbaar was. In het bovengenoemde geval zou doorkomst 2 daarom óók gekoppeld worden aan doorkomst 1, vanwege het feit dat doorkomst 3 later binnen gekomen is dan doorkomst 2. Op deze manier worden inconsistenties in de data voorkomen.

Voor de leaderboards is gekozen om deze te verspreiden over vier aparte tabellen. Leaderboards bestaan uit data die vanuit verschillende filters aangepast worden. Hierbij wordt eerst het leaderboard opgevraagd en daarna de update teruggeschreven naar de tabel. Bij het updaten is het mogelijk om een merge-operatie uit te voeren, waarbij alleen de beschikbare data (bijvoorbeeld een snelheidsleaderboard) bij te werken zonder daarbij de bestaande data aan te passen. Dit kan echter alleen wanneer de ETag-eigenschap van dit leaderboard ondertussen niet gewijzigd is. Deze ETag-eigenschap is een onderdeel van Azure Table Storage dat bijhoudt of het record in de tussentijd al dan niet gewijzigd is. In de praktijk komen tussentijdse wijzigingen vaak voor, doordat er meerdere filters parallel aan elkaar hetzelfde record bij proberen te werken voor andere leaderboard eigenschappen. Om deze reden is beter om de leaderboards te spreiden over meerdere tabellen.

4.2.3. Workflows

Een ander onderdeel van de businesslaag zijn de workflows. Workflows zijn afgebakende taken, die eenvoudig door de servicelaag en andere businesslaag-processen aan te roepen zijn, waarbij de ser-



Figuur 4.10: Flow-diagram van hoe de data stroom verkeerd kan gaan.

vicelaag niet hoeft te "weten" hoe deze taken worden uitgevoerd, als ze maar het beoogde resultaat opleveren. In onze applicatie zijn er op drie gebieden workflows, zoals ook te zien is in figuur 4.8:

- **Gebruikersaccounts beheren.** De AccountWorkflow biedt mogelijkheden om accounts aan te maken met zowel gebruikersnaam en wachtwoord als met een Facebook authenticatiesleutel ("access token"). Verder kunnen accounts worden opgezocht aan de hand van hun e-mailadres of unieke nummer. De servicelaag biedt deze workflow rechtstreeks aan via de API.

- **Transponders beheren.** Gebruikers kunnen per tijdseenheid maar één transponder hebben en deze kan niet door meer mensen tegelijk geregisteerd zijn. De AccountWorkflow is verantwoordelijk voor de logica van dit proces, zodat door een servicelaag-proces transponders kunnen worden toegevoegd aan en kunnen worden ontkoppeld van hun account (en daar eventueel de juiste foutmeldingen bij te geven). Daarnaast moet de businesslaag snel kunnen opzoeken welke gebruiker bij een transponder hoort, op het moment dat een doorkomst binnenkomt. De TransponderWorkflow bevat de logica om deze checks te doen en de transponders op te zoeken.

- **Groepen beheren.** Ook het aanmaken en beheren van groepen zijn specifieke taken waar, in dit geval, een GroupWorkflow voor bestaat. Deze workflow zorgt er voor dat de actieve gebruiker wordt toegevoegd aan de aangemaakte groep en dat er alleen wijzigingen plaatsvinden die mogen worden uitgevoerd. Deze logica wordt voornamelijk door de servicelaag gebruikt om de groepen functionaliteit aan te bieden. Daarnaast wordt beschikbare nieuwe informatie doorgestuurd naar de gebruikers uit dezelfde groepen. Ook hiervoor wordt de workflow gebruikt.

4.3. Servicelaag

Onze applicatie bevat data die op verschillende momenten van cruciaal belang, of juist onbelangrijk is. Tijdens het sporten zijn vooral doorkomsten interessant evenals informatie over de actuele training. Ná het sporten zijn analyses op de training en vergelijkingen met andere sporters interessant. Zowel SignalR[1] als WebApi[2] zijn technieken om data door te sturen, maar SignalR is gemaakt voor live data en WebApi voor het toegankelijk maken van al bestaande data via een REST API [3]. In onze applicatie gebruiken we dan ook beide technieken naast elkaar. Een derde techniek die we gebruiken is OWIN [4], een brug tussen de webserver en de webapplicatie. OWIN verzorgt het opstarten en configureren van de webapplicatie.

[1] http://www.asp.net/signalr
[2] http://www.asp.net/web-api
[3] http://en.wikipedia.org/wiki/Representational_state_transfer
[4] http://www.asp.net/aspnet/overview/owin-and-katana

4.3.1. SignalR

SignalR onderhoudt een continue verbinding tussen server en client waarbij het zelf zorgt voor het juiste onderliggende protocol. SignalR werkt bij voorkeur over WebSockets, maar werkt ook met oudere browsers door gebruik te maken van "polling". Bij polling houdt de server de verbinding zo lang mogelijk open. Als er een bericht afgeleverd moet worden, wordt het bericht verstuurd, de verbinding gesloten, en gelijk een nieuwe verbinding gemaakt. Door gebruik te maken van SignalR hoeven we ons niet bezig te houden met de werking van deze ingewikkelde protocollen, en kunnen we eenvoudig een verbinding opzetten.

Naast het beheren van de individuele verbindingen, onderhoudt SignalR ook virtuele groepen. Een nieuw bericht kan gestuurd worden naar individuele gebruikers, maar ook naar deze groepen. Wij gebruiken de groepen als interesse-context: in onze applicatie kunnen gebruikers een scherm open hebben van een baan of een (vrienden)groep, ze zijn dan geïnteresseerd in de updates van alle gebruikers in die context. Wanneer één van de leden van de context aan het schaatsen is, en er moet een update gestuurd worden, dan sturen we met één commando aan iedereen in die context een update.

4.3.2. WebApi

De API die we gemaakt hebben met WebApi 2.0 ontsluit data die niet "live" is. Onder andere de registratie en het ophalen van profieldata, groepen, favorieten en banen gebeuren via WebApi. WebApi heeft als voordeel dat het te gebruiken is zónder een SignalR client en ook dat het makkelijker te debuggen is dan SignalR, omdat er geen websocketverbinding gemaakt hoeft te worden. Door een API aan te bieden kan ons platform in de toekomst ook gemakkelijk gebruik worden voor andere applicaties.

4.3.3. OWIN

Zowel SignalR als WebApi zijn als 'middleware' geconfigureerd in het opstartscript van OWIN. OWIN zorgt er voor dat een binnenkomend verzoek wordt afgehandeld door die middleware die het verzoek kan afhandelen. Zo worden verzoeken naar de API doorgestuurd naar WebApi en verzoeken naar de url /signalr worden doorgestuurd naar SignalR.

In OWIN hebben we ook OAuth geconfigureerd. OAuth is een authenticatiestandaard waarbij een gebruiker kan inloggen met zijn gebruikersnaam en wachtwoord of via een externe dienst zoals Facebook. De client krijgt dan een "token", een sleutel, die meegestuurd moet worden met opvolgende verzoeken. Hiermee is de authenticatie centraal geregeld, aangezien zowel SignalR als WebApi hiermee overweg kunnen.

4.3.4. Platform-onafhankelijke client

Om in de mobiele applicatie de data van zowel SignalR als WebApi te ontsluiten is er een platform-onafhankelijke client gemaakt. Deze client bevat alle interactie met SignalR en WebApi zodat andere delen van de applicaties geen verbinding hoeven op te zetten en te onderhouden.

De client bevat dus methodes voor zaken als inloggen en registreren, welke zijn aangesloten op WebApi. Deze methodes zijn asynchroon en retourneren taken. Met de asynchrone taalelementen van C#[5] kunnen eenvoudig de functies worden aangeroepen vanuit knoppen in de applicatie en worden taken afgewacht, zodat óf de voortgangs-indicator kan worden verborgen óf een foutmelding getoond kan worden.

De data die vanuit SignalR binnenkomt wordt als "Observable" weergegeven. Observables zijn taalelementen van C# uit Reactive Extensions (Rx) en kunnen worden gezien als objecten die functies uitvoeren als er nieuwe data binnenkomt, door een functie te "subscriben" op een Observable. De data vanuit SignalR heeft het karakter van bijvoorbeeld *de actuele snelheid van gebruiker X*. De Observable heet dan CurrentSpeed en een voorbeeld van een functie is er een die de snelheid op het telefoonscherm aanpast aan de net binnengekomen snelheid.

[5] http://msdn.microsoft.com/en-us/library/hh191443.aspx

4.3.5. Lokale opslag met Akavache

Mobiele applicaties moeten anders omgaan met data dan desktop-applicaties in die zin dat mobiele applicaties elk moment gesloten kunnen worden. Telefoons hebben een beperkte hoeveelheid geheugen, en als het OS van de telefoon detecteert dat er te weinig geheugen vrij is kan een niet zichtbare applicatie abrupt worden afgesloten. Als een gebruiker onze applicatie even onderbreekt om een andere applicatie te bekijken dan kan de applicatie dus zomaar worden afgesloten. Om bij het terug komen in de applicatie na het afsluiten niet een leeg scherm te hoeven tonen, moeten we data opslaan op het permanente geheugen van de telefoon. Hiervoor gebruiken we Akavache[6].

Akavache is een library voor asynchrone en persitente key-value opslag voor C# applicaties. Met Akavache is eenvoudig te checken of een model al geladen is, het eventueel nog te downloaden en dan op te vragen.

Aangezien de ViewModels, die ook later nog aan bod komen in sectie 4.4.1, niet opgeslagen kunnen worden (sommige eigenschappen kunnen niet worden geserialiseerd) moet er nog een vertaalslag plaatsvinden tussen de opgeslagen entiteiten en de ViewModels. Onze platform-onafhankelijke client doet dit voor alle entiteiten en zorgt er voor dat de ViewModels in stand blijven gedurende de hele tijd dat de applicatie draait. Eventuele wijzigingen worden aangepast in de ViewModels, maar de ViewModels worden nooit vervangen.

4.4. Presentatielaag

De presentatielaag bevindt zich in de mobiele applicatie. Hierna komen enkele karakteristieke onderdelen aan bod.

4.4.1. ViewModels en Reactive UI

Voor de koppeling tussen views en data gebruiken we speciale ViewModels die een 'lifetime' hebben die langer is dan de views die deze modellen gebruiken. Controllers vragen de ViewModels op en geven ze door aan de views. Elke view bindt zich vast aan het ViewModel en als een veld van het model verandert, dan updaten de views automatisch. Dit werkt met ReactiveUI [7], een UI framework dat gebruik maakt van Rx. Elk veld van de ViewModels stuurt een notificatie naar alle luisterende views als het wordt aangepast. ReactiveUI levert hiervoor zowel het notificatiesysteem als het koppelsysteem voor views. Een versimpeld voorbeeld is als volgt:

```
public class SpeedViewModel : ReactiveObject {
  private double _speed = 0;
  public double Speed {
    get { return _speed; }
    set { RaiseAndSetIfChanged(ref _speed, value); }
  }
}

public class View : UIView, IViewFor<SpeedViewModel> {
  public View(SpeedViewModel model) {
    OneWayBind(
      model, vm => vm.Speed,
      view => view.speedLabel.Text,
      field => field.ToString("F") + " km/h"
    );
  }
}
```

[6] https://github.com/akavache/Akavache
[7] http://www.reactiveui.net/

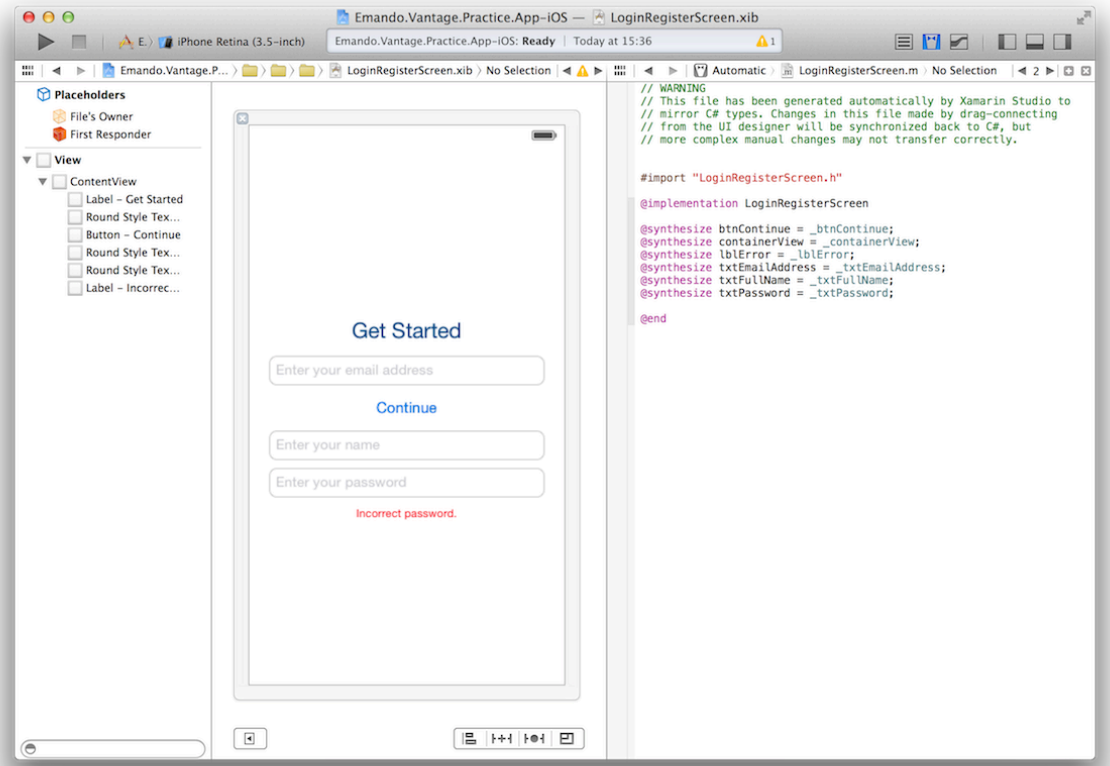
Wanneer de controller nu een ViewModel aanpast, of er via SignalR een update binnenkomt, dan wordt de view direct aangepast.

Naast het updaten van alleenstaande modellen, gebruiken we ook lijsten van modellen. Hiervoor gebruiken we ReactiveLists, uit ReactiveUI, welke notificaties kunnen sturen wanneer er een element wordt toegevoegd of wordt verwijderd uit de lijst. We updaten dan de bestaande tabel en hoeven niet de hele tabel opnieuw te tekenen.

Door gebruik te maken van ReactiveUI wordt het heel simpel om views te maken die 'live' zijn en het zorgt er voor dat de code overzichtelijk blijft, omdat het invullen en updaten van de views met dezelfde code gebeurt en deze erg simpel is.

4.4.2. Grafische user interface

De gebruikersinterface van de applicatie staat in principe los van de rest van de applicatie. Hoewel het grootste deel van de code gebruikt kan worden op verschillende platforms, moet de gebruikersinterface voor elk platform apart ontwikkeld worden. Tijdens dit bachelorproject is ervoor gelozen om alleen een specifieke gebruikersinterface te maken voor iPhones. Hiervoor is gekozen omdat het grootste deel van de KNSB-top een iPhone gebruikt. Wanneer de KNSB besluit dat de applicatie ook daadwerkelijk gebruikt gaat worden, kunnen er nog gebruikersinterfaces gebouwd worden voor andere populaire platforms.



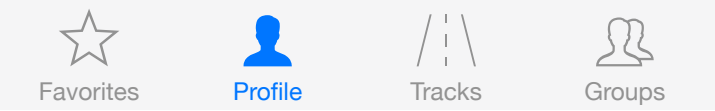
Figuur 4.11: De Xcode Interface Builder

Door het gebruik van Xamarin gaat de implementatie van de iOS user interface iets anders dan met reguliere iOS applicaties. Er kan deels gebruikgemaakt worden van de Interface Builder (figuur 4.11) die ingebouwd zit in Xcode, de officiële ontwikkelomgeving voor iOS.

Na het opslaan van de gebruikersinterface in Xcode, genereert de Xamarin-omgeving automatisch code zoals hieronder weergegeven, waarmee de native interface kan worden aangesloten op C#.

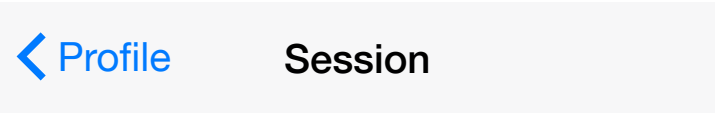
```
[Register ("LoginRegisterScreen")]
partial class LoginRegisterScreen
{
  [Outlet]
  MonoTouch.UIKit.UIButton btnContinue { get; set; }
  [Outlet]
  MonoTouch.UIKit.UILabel lblError { get; set; }
  [Outlet]
  MonoTouch.UIKit.UITextField txtEmailAddress { get; set; }
```

De gebruikersinterface van de applicatie is in twee lagen gestructureerd. Allereerst zijn er de verschillende tabs waarmee de gebruiker kan wisselen tussen de verschillende contexten: persoonlijk, favorieten, groepen en banen. Deze onderverdeling is geïmplementeerd met behulp van de in iOS ingebouwde TabBarController, te zien in figuur 4.12. Voor elk van deze contexten hebben we geprobeerd een icoontje te ontwerpen dat overeen komt met de stijl die in iOS 7 wordt gehandhaafd, maar tegelijkertijd ook duidelijk genoeg aangeeft wat er met het icoontje bedoeld wordt. Voor de gebruikers, favorieten en groepen was dit redelijk simpel, maar voor de banen was dit een grotere uitdaging, aangezien de baan er voor verschillende sporten anders uitziet. Daarom hebben we uiteindelijk besloten om een simpel symbool van een weg te gebruiken voor het representeren van banen.



Figuur 4.12: De iOS TabBarView

De tweede laag van de interface-structuur bestaat eigenlijk voor een groot deel uit overzicht- en detailschermen. Het profielscherm van een gebruiker bevat bijvoorbeeld een overzicht van al zijn sessies, een sessiescherm is weer een overzicht van alle runs, met daarin weer de afzonderlijke rondes. Deze structuur, die ook voorkomt bij de favorieten, groepen en banen, wordt geïmplementeerd met behulp van een NavigationController (figuur 4.13). Een overzicht van de interface-schetsen van de schermen die in de applicatie bestaan, is te vinden in appendix D.



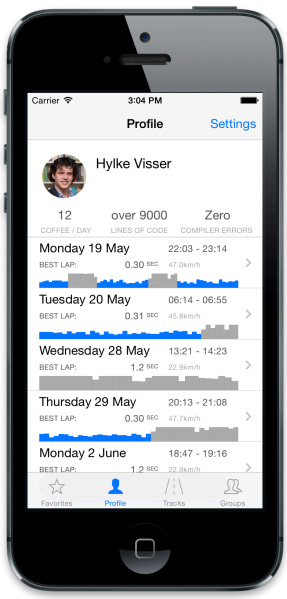
Figuur 4.13: De iOS NavigationView

Binnen ongeveer alle schermen wordt er gebruikgemaakt van tabel-views. Om deze views realtime te kunnen voorzien van actuele data, zijn er speciale tabel-viewcontrollers ontwikkeld die aansluiten op de reactive ViewModels. Telkens als er iets wordt aangepast in de ViewModels zal deze tabel-viewcontroller de nodige tabelcellen invoegen, verwijderen, verplaatsen of bijwerken.

Een ander belangrijk deel van de gebruikersinterface is de functionaliteit die ervoor zorgt dat er grafiekjes getekend worden. Dit wordt gedaan met behulp van de CoreGraphics functies in iOS. Voor elke ronde wordt een balkje getekend, waarbij de hoogte van het balkje correspondeert met de bijbehorende rondetijd. De kleur van het balkje geeft aan of de ronde een run-ronde of een rust-ronde is. De balkjes worden geschaald zodat uiteindelijk een overzicht van de volledige sessie te zien is.

Bij het tekenen van de grafieken is het van groot belang dat de grafieken automatisch bijgewerkt worden als er nieuwe data beschikbaar is, maar dat dit niet te veel impact heeft op de performance van de applicatie. Dit is gedaan door gebruik te maken van de Reactive UI (zie Sectie 4.4.1). Als er een item aan de databron wordt toegevoegd, wordt er automatisch een nieuwe balk toegevoegd. Wanneer er een rondetijd zou veranderen, wat momenteel overigens niet kan voorkomen, wordt de hoogte van het corresponderende balkje direct aangepast.

Om ervoor te zorgen dat de applicatie nog steeds snel aanvoelt, worden de grafiekjes die waarschijnlijk niet meer zullen veranderen, omgezet in afbeeldingen en gecached, zodat ze niet telkens opnieuw getekend hoeven te worden.



Figuur 4.14: Het profielscherm met het sessieoverzicht

4.5. Testomgeving

Bij elke applicatie is het van belang dat er goed getest wordt. In het geval van onze applicatie is het toch een behoorlijke uitdaging om effectief te testen. Allereerst werkt de applicatie met realtime data van een externe leverancier. Deze data kan gesimuleerd worden, maar het is lastig om unit-tests te maken die realtime en asynchrone datastromen kunnen testen. Daarnaast is het testen van een iOS App vrij lastig en tijdrovend, aangezien het veel tijd kost om vingerbewegingen te simuleren, te controleren of deze de gewenste gevolgen hebben en of de juiste data getoond wordt.

Gezien het feit dat de iOS App lastig te testen is, terwijl het grootste gedeelte van de iOS App de data vanuit de server direct overneemt in de views, hebben we besloten de iOS App handmatig te testen. De server daarentegen wordt wel uitgebreid getest met behulp van automatische unit-tests.

Binnen het servergedeelte van de applicatie worden alle transponderdoorkomsten realtime verwerkt. De filters die te maken hebben met het verwerken van deze doorkomsten worden elk afzonderlijk getest om er zeker van te zijn dat ze correct functioneren. Dit testen wordt gedaan met een set unit-tests die elk van deze filters aanroept met bepaalde input en controleert of de uitvoer overeenkomt met de verwachte uitvoer van deze filters.

Dit houdt in dat in elke test een nep-omgeving wordt gecreëerd. Deze omgeving bevat de gebruikers, groepen en relaties die tijdens de test voor het betreffende filter gebruikt worden. Alle hulpklassen voor het filter worden gemockt, wat inhoudt dat er een gesimuleerd object wordt gemaakt dat het gedrag van de oorspronkelijke klasse nabootst. Tijdens de tests wordt gecontroleerd of de juiste methoden van de hulpklassen worden aangeroepen en of de juiste uitvoer wordt geproduceerd.

Al deze tests worden uitgevoerd in de testomgeving van Visual Studio, te zien in figuur 4.15. In deze omgeving kan men zien welke tests slagen of falen en kan in debug-modus achterhaald worden wat de oorzaak is van falende tests.

4.5.1. Integratietesten van client en server

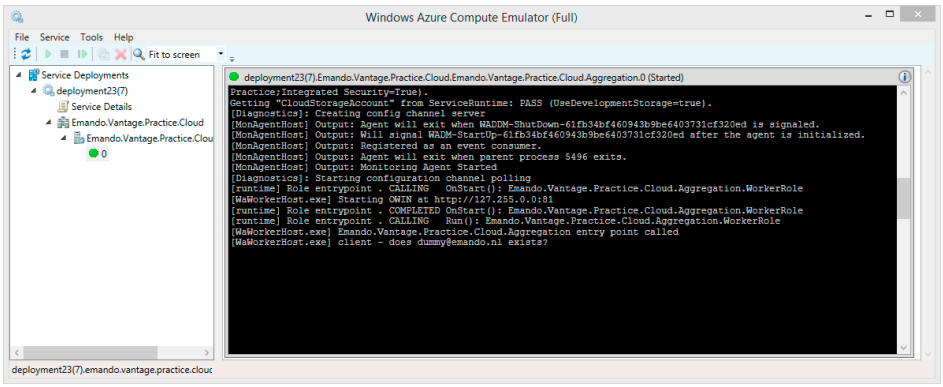
Een bijkomend voordeel van het hebben van een platform-onafhankelijke client is dat een deel van de iOS App ook uit te voeren is op hetzelfde platform als waar de server draait. Dit heeft tot gevolg dat er toch integratietests uitgevoerd kunnen worden om te controleren of de iOS App aansluit op de server. Ook maakt dit het debuggen makkelijker, aangezien in Visual Studio door de programma's heen kan 'gestept' kan worden (het regel voor regel door de lopende applicatie heen stappen).

In figuur 4.16 zijn de schermen te zien die onderdeel uitmaken van het integratietestproject. Het scherm bij (a) is de Azure Emulator, hierin worden de datalaag, businesslaag (de aggregatie) en servicelaag (de API) uitgevoerd. Bij (b) kunnen er met behulp van het toetsenbord transponderdoorkomsten worden gesimuleerd, die vervolgens in Azure worden verwerkt. Ten slotte wordt in (c) getoond welke data de ApiClient binnen krijgt.

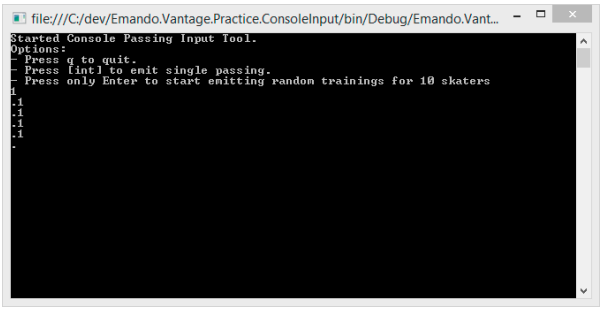
Het is niet mogelijk een test van het hele platform te automatiseren op zo'n manier dat alles op een testserver draait. Voor het uitvoeren van het Azure project is een Azure Service Bus nodig die slechts in de Cloud bestaat. Azure is alleen te bereiken met de Azure SDK die exclusief voor Visual Studio beschikbaar is, en dus alleen op Windows werkt, terwijl de Xamarin projecten voor iOS alleen op een MacBook kunnen opstarten. Om een speciale Mac test-server op te zetten met daarop een virtuele machine met Windows voert te ver voor dit project, dus worden integratie-tests uitgevoerd op onze

Figuur 4.15: De testomgeving van Visual Studio

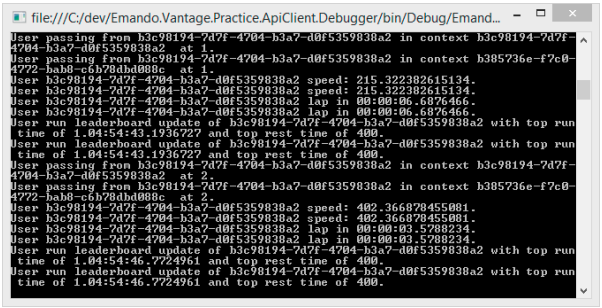
persoonlijke MacBooks, waar we deze set-up wel hebben geïnstalleerd.



(a) De Azure Emulator



(b) De Passing Input Console



(c) De Client Console

Figuur 4.16: Het integratieproject voor het testen van de API Client

5

Evaluatie

Om ervoor te zorgen dat de applicatie op zowel implementatie- als functioneel niveau goed in elkaar steekt, hebben we gedurende het project op verscheidene manieren getoetst of de applicatie voldoet aan de vooraf gestelde eisen. De resultaten van deze toetsingen zijn meegenomen in de opeenvolgende sprints om de applicatie zo goed mogelijk af te stemmen op de wensen van alle betrokken partijen.

5.1. KNSB Meetings

Voorafgaand aan dit project is er geen direct contact geweest met de KNSB. Gaandeweg het project is er vanuit Emando contact opgenomen met de KNSB om hun interesse te peilen. Er bleek veel interesse te zijn vauit de KNSB, De KNSB heeft dan ook aangegeven dat een doorontwikkeling van de applicatie wenselijk is. Ook is daarbij aangegeven dat een Android applicatie wenselijk is, om de doelgroep te vergroten.

5.2. Enquête & gebruikerstest

Om het gebruik van de applicatie onder onze doelgroep te bevorderen is het belangrijk om de applicatie aan te laten sluiten bij de verwachtingen en eisen van onze doelgroep. Om deze reden is het uiterst belangrijk dat er in een vroeg stadium gecontroleerd wordt of de functionaliteit die gebouwd wordt, aansluit op de wensen van de gebruikers. Om deze reden is er vlak na onze tweede sprint een gebruikersenquête afgenomen. Met behulp van deze enquête hoopten we inzicht te krijgen in de wensen en verwachtingen van onze doelgroep, maar ook in de reactie van gebruikers bij het zien van de eerste versie van de applicatie.

Aan het eind van het ontwikkelproces vond er een gebruikerstest plaats om te kijken of de functionaliteiten die gebouwd zijn, ook in de praktijk bruikbaar zijn. Aan de hand van de uitkomsten van deze test kunnen er ook beslissingen genomen over de toekomst en eventuele doorontwikkeling van de applicatie.

5.2.1. Enquête

Omdat de enquête in een vroeg stadium van het ontwikkelproces plaats vond vinden, was nog niet alle functionaliteit binnen de applicatie beschikbaar. Bij deze enquête werden daarom screenshots van een eerste versie van de applicatie getoond en ontwerpen van de overige functionaliteit. De gebruikers beantwoordden aan de hand van die beelden de vragenlijst.

In de enquête zelf werden er vragen gesteld om te achterhalen hoe belangrijk/waardevol mensen bepaalde functionaliteiten vinden. Dit werd gedaan worden met behulp van de Likert Scale [8].

De enquête is te vinden op pagina 59 en 60. De resultaten van de enquête zijn te vinden op pagina 57.

5.2.2. Resultaten enquête

Er vallen een paar dingen op in de resultaten van de enquête. Allereerst is het marktaandeel van Android erg groot. We hebben er in overleg met de opdrachtgever voor gekozen voor iOS te ontwikkelen, maar de markt voor een Android versie is er dus zeker.

Daarnaast valt op dat de basis functionaliteit, het tonen van doorkomsten en het inzien van trainingen, erg hoog scoort. De sociale functionaliteit scoort lager, al zijn leaderboards en groepen toch ook erg gewild. Het delen via Facebook en RunKeeper is volgens de respondenten niet belangrijk. Zoals ook uit de overige opmerkingen blijkt is de audio feedback echt een functie die er voor kan zorgen dat de applicatie ook bruikbaar is voor schaaters tijdens de training. Het bekijken van de telefoon tijdens het (hard) schaatsen is waarschijnlijk niet veilig.

Verder merken enkele respondenten op dat het jammer zou zijn als er alleen met transponders gewerkt kan worden. Ze noemen GPS als alternatief, maar uit ervaring weten we dat GPS niet werkt in afgesloten ruimtes zoals een schaatsbaan. Door ondersteuning in te bouwen voor GPS is echter wel de bruikbaarheid uit te breiden naar weg-wielrennen, skeeleren en veel andere sporten. Op de markt van GPS-trackers zijn echter al vele partijen actief, waardoor wij hebben besloten nog geen GPS ondersteuning in te bouwen. Een belangrijke feature die onze applicatie echter wel biedt is het analyseren van rondes, iets wat bij de meeste GPS-trackers geen optie is. In de toekomst kan er dus zeker gekeken worden of GPS geïntegreerd kan worden, ook met het oog op natuurijs en de wensen van de KNSB.

5.2.3. Gebruikerstest

Om een echte gebruikerstest te doen, moest er ijs liggen. Gelukkig voor ons werd de verbouwing van Thialf een zomer uitgesteld en lag er zomerijs in Thialf. De baan in Friesland is als enige baan van Nederland gedurende de zomer open en dit jaar lag er ijs vanaf 14 juni. Op 18 juni heeft er een gebruikerstest plaatsgevonden tijdens twee reguliere trainingen. Van 10.15 tot 11.45 was het topsport-uur en daarna was er tot 16.00 gelegenheid voor recreanten om te schaatsen.

Tijdens het topsport-uur hebben we transponders uitgedeeld aan de Activia-ploeg. Annette Gerritsen, Roxanne van Hemert en Laurine van Riessen reden met onze transponders. Daarnaast hebben met hun coach Jac Orie de applicatie doorgenomen. Er was voor team Activia helaas geen tijd om een nabespreking te doen na de training. Wel vonden ze vooraf het idee van de applicatie leuk en waren ze te spreken over de al aanwezige functionaliteit in de applicatie. De functionaliteit om gebruikersprofielen af te schermen werd nog genoemd door een coach van een ander team. Deze functionaliteit hadden wij ook al opgenomen in onze Should-have lijst, maar was nog niet gebouwd.

Na de topsport-training hebben we een grotere groep schaatsers transponders gegeven en heeft Herman ook geschaatst. Voorafgaand aan de test werden de testers kort gebrieft worden over de bedoeling van de test. Gedurende de training van de deelnemers namen we steeds een deelnemer aan de kant om de applicatie en zijn schaats-resultaten te laten zien op een test-telefoon. We hadden op deze telefoons de applicatie al geïnstalleerd en zo ingesteld dat de account gekoppeld was aan de juiste transponder, zodat de gebruikers direct aan de slag konden met de applicatie.

5.2.4. Resultaten gebruikerstest

Zowel de topsporters als recreatieve schaatsers bijzonder te spreken over de applicatie. Velen van de recreanten die wij spraken beschikten (nog) niet over een transponder. Ze gaven aan dat dit kwam door de beperkte mogelijkheden. Het merendeel van hen gaf aan het aanschaffen van een transponder met de komt van de applicatie te heroverwegen. Bij het testen van de applicatie hebben we veel enthousiaste reacties ontvangen. Veel testgebruikers waren verbaasd over de uitgebreide mogelijkheden en het gedetailleerde niveau waarop de applicatie inzicht gaf in hun trainingsdata. Een aantal van hen noemde de MyLaps practice website, waarin zij naar eigen zeggen "wel eens" naar hun trainingsdata keken en gaven daarbij aan dat deze velen malen minder inzichtelijk was dan onze applicatie.

Hoewel de respondenten en testers enthousiast waren over de applicatie, bleek uit de test wel dat gebruikers mogelijkheden zagen om de applicatie uit te breiden. De belangrijkste en meest genoemde

toevoegingen zijn hieronder genoemd:

- Profielen/groepen/trainingsdata onvindbaar maken voor anderen (handig voor topsporters).
- Audio-cues, live audio feedback over je training
- Integratie met hartslagmeter
- Schatting voor calorieën verbruik
- Integratie met instelbare trainingsschema's en suggesties hiervoor aan de hand van training resultaten
- Mogelijkheid om te trainen met GPS, i.p.v. transponders

5.3. SIG feedback

Om de code kwaliteit van het project in de kaart te brengen heeft de Software Improvement Group (SIG) onze code anderhalve week voor onze eindpresentatie geëvalueerd:

> De code van het systeem scoort bijna 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Interfacing en Unit Size.
>
> Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. In dit systeem valt op dat er op meerdere plekken twee 'DateTime' objecten worden meegegeven. In sommige gevallen representeren deze twee objecten een 'periode', maar in andere gevallen lijken de parameters 'time' en 'sent' iets anders te representeren. Om bij toekomstige aanpassingen duidelijker te maken wat er precies meegegeven moet worden aan deze methodes is het aan te raden een specifiek type te introduceren voor deze concepten.
>
> Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'Run'-methode binnen de class 'Program', zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// Create a new passing' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.
>
> Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Uit deze evaluatie komt naar voren dat de code goed onderhoudbaar is. De genoemde voorbeelden over meerdere DateTime objecten die hetzelfde lijken te representeren, representeren een tijdstip waarop de doorkomst ontvangen is en een tijdstip waarop de transponder langs de lus ging. In de praktijk zijn dit compleet verschillende dingen, omdat er sprake kan zijn van een latency op de MyLaps X2 Cloud omgeving.

Vanuit Emando wordt gebruik gemaakt van Dependency Injection. Bij het ontwikkelen van de verschillende repositories en controllers is bewust gekozen voor een opsplitsing van belangen volgens het

Seperation of Concerns principe [9, 10]. Classes maken dus gebruik van relatief "kleine" interfaces, maar hebben er daardoor meerdere interfaces nodig in hun contructor. Door het gebruik van Dependency Injection is het niet nodig om bij het gebruik van deze classes de constructors aan te roepen met al hun parameters.

De genoemde Unit Size betreft de run methode van het Console Input Test project. Hoewel deze methode inderdaad aan de lange kant is, is dit geen onderdeel van de productie code. Dit project wordt enkel gebruikt om doorkomsten te genereren en integratie testen uit te voeren.

Ook het genereren van nieuwe doorkomsten in deze run methode is niet opgesplitst naar een aparte methode, omdat het aanmaken van een doorkomst geen aparte functionaliteit is en dit tevens niet meerdere malen gebeurd. Door het genereren van een doorkomst te abstraheren naar een aparte methode, ontstaat een methode met een grote hoeveelheid parameters (Passing bevat een grote hoeveelheid properties), wat de leesbaarheid van de code niet ten goede zou komen.

De bovengenoemde problemen zijn niet direct van toepassing op de productiecode en daarom is in overleg met Emando er voor gekozen om de verkregen feedback van SIG niet direct te verwerken in de bestaande productiecode. Wel is de feedback meegenomen binnen het verdere ontwikkeltraject van de applicatie en hebben we er op toegezien om verdere functionaliteit van de applicatie conform de genoemde standaarden te ontwikkelen.

6 Discussie

Voorafgaand aan het project zijn zowel op proces- als productniveau een aantal beslissingen genomen. Deze beslissingen hebben een directe invloed gehad op de werkwijze van het project en het uiteindelijke resultaat. In dit hoofdstuk worden een aantal van deze keuzes geëvalueerd aan de hand van de bevindingen en resultaten gedurende het project.

6.1. Product

6.1.1. Hoe "af" is de applicatie?

Bij de oplevering van het project staat er een werkende applicatie die sporters op een gemakkelijke en overzichtelijke manier inzicht geeft in hun trainingen, maar ook in die van anderen. Daarnaast bevat de applicatie een sociale component die sporters kan motiveren om tot het uiterste te gaan. Deze sociale component zorgt er ook voor dat sporters de applicatie willen blijven gebruiken. De applicatie berust op een robuuste en uitbreidbare API welke goed gedocumenteerd is. Het is daarmee relatief eenvoudig om extra functionaliteit toe te voegen.

6.1.2. Verbeteringen

Hoewel de applicatie voldoet aan de door ons gedefiniëerde MVP, zijn er enkele verbeterpunten welke toevoegingen kunnen zijn aan de applicatie. Zo zou het mogelijk kunnen zijn om realtime trainingsinformatie door te geven aan sporters met behulp van audio cues. Verder zou de geaggregeerde data gecombineerd kunnen worden met trainingstijden om gebruikersprofielen te creëren, welke gebruikt kunnen worden om gebruikers suggesties aan te kunnen bieden om met andere sporters van gelijkwaardig niveau te gaan sporten. Ook is het mogelijk om de sociale component verder uit te breiden naar een achievement-systeem.

Tot slot zou ook het delen van prestaties en records en het exporteren van deze data naar RunKeeper een toevoeging zijn voor de applicatie.

6.1.3. Android

Momenteel is de applicatie slechts beschikbaar voor één mobiel platform: iOS. Naast iOS is ook Android een groot platform waar veel gebruikers van onze doelgroep mee werken. Dit heeft de KNSB reeds aangegeven in een van de meetings met Emando. Om het bereik van de applicatie te vergroten, is het aan te raden om de applicatie ook uit te rollen voor Android. Vanwege de cross-platform ontwikkeling met Xamarin, is het slechts nodig om de platform-specifieke views te schrijven voor Android. Om deze reden kan dit in relatief weinig tijd.

6.1.4. Is de applicatie een toevoeging op de reeds bestaande markt?

Hoewel er zeker ruimte is voor toevoegingen van en verbeteringen aan de huidige applicatie, bevat de applicatie tal van functionaliteiten welke trainingen voor sporters op een overzichtelijke manier

inzichtelijker kunnen maken. Naast het inzien van eigen data, is het ook mogelijk om andermans data in te zien, hetgeen een toevoeging is voor trainingsgroepen en coaches. Hoewel deze twee aspecten an sich niet nieuw zijn, is de eenvoud en combinatie er van in één applicatie dat wel. Verder is de sociale component niet terug te vinden in vergelijkbare reeds beschikbare applicaties.

Uit de enquêtes en gebruikerstest is gebleken dat de applicatie veel potentie heeft en er zeker vraag naar is onder sporters. Ook kwam naar voren dat er veel vraag is naar extra functionaliteit (met name de audio cues), maar de ondervraagden gaven hierbij aan de applicatie met de huidige functionaliteit ook al graag in gebruik te nemen.

6.2. Proces

6.2.1. SCRUM overhead

Omdat de precieze vormgeving van de applicatie naast het MVP niet vast stond en veel cutting-edge technieken gebruikt werden, bleek de flexibele ontwikkelmethode SCRUM een goede oplossing voor ons project. Hiermee was het gedurende het project mogelijk om de doelstellingen en implementatie-details per sprint bij te stellen. Ondanks het feit dat iedere sprint opnieuw gepland moest worden en dagelijkse stand-ups nodig waren om de projectvoortgang en planning door te spreken, heeft SCRUM veel flexibiliteit en overzicht geboden binnen het project. Het gebuik van SCRUM raden wij dan ook ten zeerste aan bij een volgend (soortgelijk) project.

Effectiviteit TFS

Bij de planning van het project en het werken in sprints hebben we veelvuldig gebruik gemaakt van TFS. Hoewel dit een handige tool is, welke binnen Emando de standaard was bij het plannen van sprints, zit er bij het plannen veel overhead. Omdat iedere functionaliteit opgesplitst moest worden in verantwoordelijkheden, backlog items met verantwoordelijkheden en corresponderende taken met een tijdsbesteding, zat er veel tijd in het op orde houden van de planning en voortgang binnen TFS. Ondanks het feit dat Visual Studio goede integratie met TFS biedt, dienen taken ook handmatig op orde gehouden te worden om bij de dagelijkse stand-ups een goed beeld van de voortgang van het project te verkrijgen.

Omdat onze opdrachtgever een controlerende rol heeft aangenomen binnen het project, bood TFS een uitkomst om op ieder moment gemakkelijk inzicht te verkrijgen in de projectvoortgang en afhankelijkheden. Ook tijdens het project was het voor ons erg prettig om in TFS altijd een overzicht te hebben van alle taken en prioriteiten.

Ondanks het feit dat het inplannen van sprints in TFS relatief veel tijd heeft gekost, heeft het gebruik van TFS ons in combinatie met de dagelijkse stand-ups veel tijd bespaard doordat het een helder overzicht geeft van de projectvoortgang en afhankelijkheden. Het gebruik van TFS sprint planning tools als TFS raden wij dan ook ten zeerste aan bij een volgend project.

6.2.2. Burndown & afhankelijkheden

In TFS is het mogelijk om aan de hand van de gemaakte sprint planning een burndown-grafiek te genereren, waarin te zien is hoe de projectvoortgang in een sprint verloopt. Bij het plannen van de sprint dienen daarbij een exact aantal uren per taak aangegeven te worden. Door het gebruik van veel cutting-edge technieken die nieuw waren voor ons, hebben sommige taken veel meer en andere veel minder tijd gekost dan aanvankelijk gedacht. Vanwege de onderlinge afhankelijkheden was het soms lastig om verder te gaan met taken die berustten op andere taken die vertraging op liepen. Omdat deze taken op TFS bij een vertraging niet afgevinkt konden worden, bevat de burndown-grafiek daarom veel pieken en dalen en week het merendeel van de dagen de burndown-grafiek af van de ideale werklijn.

Ook bleek bij gedurende sprint regelmatig dat er voor bepaalde taken op de achtergrond extra taken nodig waren om deze naar behoren te laten functioneren. Bij het aanmaken van nieuwe taken binnen TFS met corresponderende uren ontstond vanzelfsprekend een stijgende lijn in de burndown-grafiek.

6.2.3. Verwachtingen verschillende betrokkenen

Bij het project waren aanvankelijk drie partijen betrokken: de opdrachtgever, de begeleider vanuit de TU Delft en de projectleden. Later is door het contact tussen Emando en de KNSB ook de laatste betrokken bij het project.

Omdat de verwachtingen van alle betrokkenen van elkaar afweken, is gezien de tijdsdruk gekozen voor een middenweg waarin iedereen zich kon vinden. Daarbij is gekozen voor een applicatie die gemakkelijk uitbreidbaar is, voldoet aan alle eisen van MVP (en waar mogelijk meer), voorzien is van een robuuste en stabiele back-end en een goede documentatie van het project en zijn voortgang.

Contact met opdrachtgever

Om goed contact te onderhouden met de opdrachtgever is er voor gekozen om vier dagen per week op het kantoor van Emando in Amsterdam te werken. Dit maakte het mogelijk om de verwachtingen, projectvoortgang en -planning regelmatig af te stemmen in de dagelijkse stand-ups. Ook zorgde dit er voor dat implementatiedetails en code reviews gemakkelijk doorgesproken konden worden, zodat alle implementaties volgens de bedrijfsstructuur van Emando gerealiseerd konden worden.

6.2.4. Cutting-edge technieken

Bij het maken van implementatiekeuzes is regelmatig bewust besloten om cutting-edge technieken te gebruiken. Hoewel het gebruik van de nieuwste technologieën veel mogelijkheden met zich meebrengt, kost het meestal meer tijd dan het gebruiken van conventionele technieken. Dit heeft te maken met de grote hoeveelheid (nog niet gedocumenteerde) mogelijkheden die nieuwe technologieën met zicht meebrengen.

Bij het maken van deze afweging was het niet alleen de technische uitdaging, maar ook het het toekomstperspectief van de applicatie dat de doorslag gaf voor het gebruik van deze technologieën. De weliswaar steilere leercurve heeft ons er niet van weerhouden om deze veelbelovende technieken te gaan gebruiken.

7 Conclusie

De applicatie is in het kader van het Bachelor Eindproject "Trainingsapp voor baansporten" ontwikkeld. Omdat de projectduur van dit project vele malen korter is dan het traject van een volledige applicatie, is de applicatie niet compleet. Zoals beschreven is in de resultaten van de gebruikerstest en de discussie in hoofdstuk 6, zijn er tal van functionaliteiten waarmee de applicatie later uitgebreid kan worden. Het is daarom belangrijk dat het project goed opgeleverd wordt, zodat de applicatie door ontwikkeld kan worden.

7.1. Oplevering

Alle code die ontwikkeld is binnen het project is terug te vinden binnen Source Control van het project in TFS. Om eventuele doorontwikkeling van het project te bevorderen bevatten is alle code, waar nodig, voorzien van commentaar. Ook zijn er voor de API's documenten gemaakt waarin hun werk- en gebruikswijze gedocumenteerd is.

Naast deze documentatie voor de code, is er ook een aantal documenten geschreven waarin beschreven staat hoe andere ontwikkelaars aan de slag zouden kunnen met het project. Hierin staat de volgorde van stappen beschreven die ondernomen dient te worden om alle afhankelijkheden en cloud instances draaiende te krijgen.

7.2. Doorontwikkeling

Hoewel de applicatie momenteel geschikt is om uit te brengen op de markt, zijn er nog tal van mogelijkheden om de applicatie door te ontwikkelen. Zoals uit de enquêtes en de meetings met de KNSB gebleken is, is er veel vraag naar de applicatie, maar ook naar verdere functionaliteit. Enkele van deze gewenste functionaliteiten zijn:

- Profielen/groepen/trainingsdata onvindbaar maken voor anderen (handig voor topsporters).
- Audio cues, live audio feedback over je training
- Integratie met hartslagmeter
- Schatting voor calorieën verbruik
- Integratie met instelbare trainingsschema's en suggesties hiervoor aan de hand van training resultaten
- Mogelijkheid om te trainen met GPS, i.p.v. transponders

Ook een vergroting van de doelgroep door de applicatie uit te brengen op andere platformen is wenselijk. Binnen TFS zijn backlog items aangemaakt voor alle functionaliteit die later wenselijk is,

maar nog niet geïmplementeerd is. Deze backlog items zijn gesorteerd op volgorde van prioriteit en kunnen gebruikt worden als leidraad bij de doorontwikkeling van het project.

In het theoretische geval dat er een Azure Cloud instance crasht kan het voorkomen dat er aggregaties meerdere malen berekent. Dit resulteert niet in "verkeerde" data, maar kan wel zorgen voor dubbele berekeningen over intervallen. Dit probleem is te verhelpen door aggregaties niet te laten werken met een synchronisatietijdstip, maar in plaats daarvan aggregaties die vertraagd raken, de "te vroeg" uitgerekende aggregaties te laten verwijderen en opnieuw uit te laten rekenen. In overleg met Emando is er voor gekozen dit probleem niet op te lossen tijdens dit Bachelorproject omdat de kans dat een dergelijke Cloud instance crasht bijzonder klein is en het oplossen van dit probleem op technisch vlak een grote hoeveelheid tijd zou kosten. Dit is een probleem dat in de doorontwikkeling zeker verholpen kan worden.

7.3. Toekomst

De KNSB heeft in overleg met Emando besloten de mogelijkheden van huidige applicatie mee te nemen in de nieuwe strategie voor het seizoen 2014-2015 om actieve sporters te benaderen. De KNSB denkt momenteel na over een complete applicatie die naast het trainingsgedeelte ook andere schaatsgerelateerde functionaliteiten zou bevatten. Deze functionaliteiten zouden onder andere informatie over verenigingen, informatie over (veilig) natuurijs, openingstijden en adressen van schaatsbanen etc. kunnen omvatten. Het is nog onduidelijk of de huidige applicatie uitgebreid zal worden met functionaliteit en platformen of dat er een complete schaats-applicatie komt waar de gebouwde functionaliteit onderdeel van zal uitmaken.

Het is in ieder geval zeker dat de KNSB veel potentie ziet in de gebouwde functionaliteit en deze functionaliteit ook graag wil aanbieden aan de Nederlandse schaatsers.

A

Plan van Aanpak

A.1. Introductie

Aan de hand van de projectopdracht, te lezen in Sectie A.2, en gesprekken met Johan en Cor-Paul hebben we dit Plan van Aanpak opgesteld. Dit Plan van Aanpak zal worden gebruikt als houvast bij het uitvoeren van het project en wordt daarom ook goedgekeurd door alle betrokken partijen.

In dit document zal eerst de projectopdracht aan bod komen en zal worden toegelicht hoe deze tot stand is gekomen. Vervolgens wordt toegelicht op welke wijze we te werk zullen gaan en wat onze tijdsplanning is. Hierna wordt iets dieper ingegaan op de procesmatige inrichting van het project. Tot slot zal er nog aandacht worden besteed aan kwaliteitsborging.

A.2. Projectopdracht

A.2.1. Opdrachtomgeving

Emando is een bedrijf dat zich momenteel onder andere richt op het ontwikkelen van Vantage. Een onderdeel van Vantage is een systeem voor tijdregistratie tijdens wedstrijden in baansporten. Een groot deel van de door Vantage gebruikte infrastructuur zou ook gebruikt kunnen worden voor de recreatieve- of amateurvariant van de sport. Hiervoor zou het project "Trainingsapp voor Baansporten" verschillende mogelijkheden moeten bieden.

A.2.2. Doelstelling project

Het project is bedoeld als pilot. Dit houdt in dat het een mogelijke toevoeging zou kunnen zijn aan het bestaande Vantage platform, dat momenteel vooral gericht is op wedstrijden. Emando kan met dit project ook een dienst aanbieden voor recreatieve sporters en voor trainingen om hun prestaties in te zien en te vergelijken.

A.2.3. Opdrachtformulering

In overleg met Emando is de projectopdract als volgt gedefinieerd:

> *Het ontwerpen en ontwikkelen van een systeem voor het weergeven en analyseren van (live) recreatieve en trainingsdata in de context van baansporten (bijvoorbeeld schaatsen en baanwielrennen). Dit systeem zal realtime gegevens van de baan, maar ook opgeslagen gegevens uit het verleden, inzichtelijk maken voor sporters, coaches en eventuele toeschouwers (volgers). Daarnaast zal er de mogelijkheid zijn om een sociale component en een competitie element toe te voegen, en een mogelijkheid om aggregatie op data uit te voeren.*
>
> *Het systeem ondersteunt het sporten: sporters hoeven hun training niet te onderbreken, doordat audio cues en sport-specifieke schermen geïmplementeerd worden. De sociale component bestaat uit het volgen van medesporters c.q. familieleden of vrienden en het*

delen van resultaten. Het competitie-element zal een virtuele competitie tussen vrienden omvatten.

Technisch gezien zal er een scheiding gemaakt worden tussen de diverse toepassingen (clients) en de API. De API biedt toegang tot realtime data en de diverse modulaire (sport-specifieke) aggregaties, waarbij de mogelijkheid bestaat voor bijvoorbeeld coaches om zich te abonneren op meerdere sporters. De API zal gekoppeld worden op bestaande systemen die data van transponders ontsluiten.

A.2.4. Op te leveren producten en diensten

Het project zal uiteindelijk resulteren in een applicatie waarmee (live) recreatieve- en trainingsdata van baansporten wordt geanalyseerd en getoond aan sporters, coaches of toeschouwers. Verder zullen gedurende het project de volgende documenten worden opgeleverd:

- Plan van Aanpak
- Oriëntatie-verslag (onderdeel van verslag)
- Programma van eisen (onderdeel van verslag)
- Eindverslag
- Reflectie aangaande de gang van het proces

A.2.5. Specifieke producteisen

De specifieke producteisen zijn eerder behandeld in Sectie 2.4.

A.3. Specifieke proceseisen

Gedurende het project zal er per teamlid 40 uur per week gewerkt worden aan het project, zoals overeen gekomen in de stageovereenkomst. De projectwerkzaamheden zullen voor het grootste deel plaatsvinden op het kantoor van Emando te Amsterdam. De opdrachtgever zal ervoor zorgen dat de teamleden een werkplek hebben wanneer zij hun werkzaamheden uitvoeren op het kantoor van Emando. Om goed contact met de begeleider aan de TU Delft te behouden, zal er tevens gewerkt worden op de Technische Universiteit Delft.

Het ontwikkelproces zal *agile* zijn. Dat wil zeggen dat de applicatie zal worden opgeleverd in verschillende iteraties, waarin telkens functionaliteit aan de applicatie wordt toegevoegd. Er zal gewerkt worden met de Minimum Viable Product-aanpak. Er wordt in eerste instantie een applicatie opgeleverd met precies de kern-functionaliteiten en niet meer. In een latere stadium kan er op een iteratieve wijze extra functionaliteit worden toegevoegd.

De teamleden zullen zorg uitdragen om de projectwerkzaamheden naar beste weten en kunnen uit te voeren. Daarbij worden echter geen garanties gedaan over de verwachte resultaten, zoals deze omschreven zijn in de projectopdracht.

A.4. Tijdsplanning

Iedere week zal er gefocust worden op bepaalde aspecten van het proces en functionaliteiten van de applicatie. De planning is opgenomen als Appendix B.

A.5. Projectinrichting

A.5.1. Proces

Naast de aspecten uit de tijdsplanning zullen we elke werkdag beginnen met een stand-up: elk teamlid vertelt gedurende maximaal tien minuten wat hij afgelopen dag bereikt heeft en wat hij van plan is die dag te doen. De opdrachtgever heeft aangegeven hier zo vaak als voor hem mogelijk is bij aanwezig te zijn. Aan het eind van elke week is er een demo. Door elke dag de problemen en uitdagingen te bespreken kunnen vertragingen zo vroeg mogelijk gedetecteerd of voorkomen worden.

De voortgang van het proces zal worden bijgehouden door middel van een issue tracker en een (digitaal) SCRUM[1] bord. De teamleden zullen met behulp van deze tools de voortgang van het proces inzichtelijk maken en een overzicht kunnen geven van het project.

Zoals per stagecontract afgesproken, zal er per teamlid 40 uur per week gewerkt worden aan het project, waarbij teamleden zorg zullen uitdragen om de projectwerkzaamheden naar beste weten en kunnen uit te voeren. Er is in die zin een gedeelde verantwoordelijkheid, waar teamleden elkaar zonodig bijsturing geven.

A.5.2. Resources

De opdrachtgever stelt gedurende het project een werkplek ter beschikking in het kantoor aan het Leidseplein in Amsterdam. We zullen gedurende het project op onze eigen laptops werken. Via AcademicDownload[2] verschaft de TU Delft studenten licenties voor Windows 8.1 Enterprise. Overige licenties voor ontwikkeltools zullen door de opdrachtgever worden verzorgd.

Via MyLaps zal de opdrachtgever ons toegang verschaffen tot de transponder-data. Overige data van onder andere de KNSB zal ook via het netwerk van de opdrachtgever worden verkregen.

A.6. Kwaliteitsborging

De kwaliteit van de code kan gewaarborgd worden door het aanhouden van best practices en conventies die worden gebruikt binnen Emando. Daarnaast kan gebruik gemaakt worden van de Code Review functionaliteit die in de ontwikkeltool beschikbaar is. Hiermee kunnen teamleden elkaars code beoordelen en waar nodig verbeteren. Ten slotte zal de code voor analyse worden opgestuurd naar de SIG. Afhankelijk van de SIG resultaten, kan de code verbeterd worden waar nodig.

De kwaliteit van de opgeleverde applicatie zal worden gewaarborgd door veelvuldig te testen. Dit zal niet alleen gedaan worden door middel van unit-tests, maar ook door tijdens het proces enkele integratie-tests en user-tests uit te voeren. De resultaten uit de user-tests kunnen eventueel ook gebruikt worden om het ontwerp van de applicatie beter af te stemmen op de gebruikers.

De kwaliteit en efficiëntie van het proces zal hoog worden gehouden door het gebruik van een online projectbeheeromgeving. De opdrachtgever kan via het SCRUM bord op deze omgeving de voortgang in de gaten houden en zal bij de dagelijkse stand-ups op de hoogte worden gesteld van de voortgang van het project en eventuele problemen of uitdagingen. Deze dagelijkse stand-ups zorgen er ook voor dat misverstanden tussen de opdrachtgever en het team worden voorkomen of in een vroeg stadium kunnen worden opgelost.

[1] http://en.wikipedia.org/wiki/Scrum_(software_development)
[2] http://academicdownload.com

B

Planning

Week 1 21 april - 25 april Research	Opstellen plan van aanpak Orientatie en literatuurstudie
Week 2 28 april - 2 mei Research	Orientatie en literatuur studie Verwerken resultaten literatuurstudie Orientatieverslag afmaken Begin opzet architectuur Regelen benodigdheden user-test
Week 3 5 mei - 9 mei Development	Verwerken resultaten literatuurstudie Opzet architectuur data analyse server Opzet architectuur API Opzet architectuur client Verwerken opzet architectuur
Week 4 12 mei - 16 mei Development	Ontwikkeling data analyse server Ontwikkeling API Ontwikkeling client Voorbereiden user-test Verwerken opzet architectuur Verwerken ontwikkelingsproces Verwerken opzet user-test **Deadline:** Minimum Viable Product
Week 5 19 mei - 23 mei Development	Ontwikkeling data analyse server Integratie-tests Eerste user-test

Week 6 26 mei - 30 mei Development	Toevoegen functionaliteit Verwerken inzichten uit user-test Regelen benodigdheden demo bij eindpresentatie
Week 7 2 juni - 6 juni Development	Toevoegen functionaliteit Afronden eindverslag
Week 8 9 juni - 13 juni Development & Testing	Begin test-proces Tweede user-test Afronden product Afronden eindverslag **Deadline:** Opsturen broncode naar de SIG
Week 9 16 juni - 20 juni Development & Testing	Verwerken feedback user-test Verbeteren laatste problemen Afronden product Afronden eindverslag Voorbereiden presentatie Verbeteren broncode aan de hand van feedback van de SIG
Week 10 23 juni - 27 juni Testing & Presentatie	Presentatie Terugblik project Overdracht code & product **Deadline:** Inleveren hardcopy eindverslag **Deadline:** Opsturen broncode naar de SIG

C

Oriëntatieverslag

C.1. Introductie

Aan de hand van de projectopdracht, ons Plan van Aanpak (te vinden in Appendix A) en gesprekken met Johan en Cor-Paul hebben we dit Oriëntatieverslag opgesteld. Dit verslag biedt inzicht in de mogelijke technieken voor het ontwikkelen van mobiele applicaties en server back-ends, geschikt voor onze toepassing. Ook zullen in dit verslag de keuzes voor het gebruik van bepaalde technieken worden toegelicht.

C.2. Soortgelijke oplossingen

In Sectie 2.3 is een overzicht gegeven van soortgelijke oplossingen.

C.3. Doelgroep & gebruikers

In Sectie 2.2 is behandeld wat de doelgroep van de applicatie is en welke verschillende soorten gebruikers er verwacht worden.

C.4. Back-end

Afhankelijk van welk back-end platform we kiezen, zijn er verschillende mogelijkheden voor de servers waarop de back-end kan draaien. Ook is er de keuze tussen traditionele server clusters en een cloud-oplossing, zoals een Platform as a Service (PaaS). Er zijn verschillende redenen om wel of niet voor een cloud oplossing te gaan. De voor- en nadelen van een cloud oplossing staan hieronder opgesomd.

Bij het vergelijken van een cloud oplossing met een traditioneel serverpark zijn we er vanuit gegaan dat dit serverpark zelf beheerd wordt en niet uitbesteed is wordt een externe partij. Dit houdt ook in dat het onderhoud hiervan door ons zelf gedaan zou worden.

C.4.1. Schaalbaarheid

Cloud oplossingen bieden de mogelijkheid om gemakkelijk en (automatisch) vrijwel onbeperkt te schalen bij een hoge load. Sommige cloud oplossingen zoals Windows Azure bieden de mogelijkheid om dit automatisch op te laten schalen aan de hand van de load. Bij een traditioneel serverpark is dit niet het geval. Hierbij dienen handmatig servers aangesloten te worden op het moment van een hoge load.

C.4.2. Besturingsssyteem

In tegenstelling tot bij een traditioneel serverpark, ligt de verantwoordelijkheid van het beheren van de besturingssystemen niet bij ons, maar bij de beheerder van de cloud service. Het instellen en optimaliseren van alle besturingsssystemen voor alle frameworks en programmeertalen kan veel tijd kosten. Door deze verantwoordelijkheid bij de beheerder van de cloud service te leggen, kan veel tijd bespaard worden.

	Azure	App Engine	AWS	Heroku	AppFog
.NET	×		×		
Java	×	×	×	×	×
Scala	×	×	×	×	×
Python	×	×	×		×
Node.JS	×		×	×	×
PHP	×	×	×		×
Ruby	×		×	×	×
Overig		Go		Closure	Erlang

Tabel C.1: Vijf grote PaaS providers en de talen en frameworks die ze ondersteunen

C.4.3. Onderhoud & Uptime

Cloud oplossingen worden beheerd en onderhouden door de service provider, waar dit bij een traditioneel serverpark bij ons zelf zou liggen. Ook bieden cloud oplossingen een gegarrandeerde hoge update, zonder dat we hier zelf iets voor hoeven doen.

C.4.4. Kosten

Bovenop de kosten van die de service provider van de cloud oplossing maakt, wordt een winstmarge geheven. Dit is bij het traditionele serverpark niet het geval. Bij een cloud oplossing zijn de onderhoudskosten (voor ons) vele malen lager, waardoor deze kosten zowel hoger als lager uit kunnen vallen in vergelijking tot de kosten van een traditioneel serverpark.

C.4.5. Beschikbaarheid frameworks/programmeertalen

Doordat cloud oplossingen door de service provider beheerd worden is er doorgaans minder vrijheid in het kiezen van frameworks/programmeertalen. Alleen de bekendste en meest gebruikte frameworks/programmeertalen worden ondersteund. Bij een traditioneel serverpark hebben we de mogelijkheid om deze volledig zelf in te richten en hebben we daarmee meer vrijheid. Vanwege onze keuze voor C# en .NET, is dit geen probleem voor ons.

Het nadeel dat de prijs hoger kan liggen kan worden afgezwakt en mogelijk worden omgebogen: om grote hoeveelheden gebruikers aan te kunnen moet een traditioneel server cluster groot zijn, terwijl een PaaS automatisch kan schalen en dus op dal-momenten significant goedkoper kan zijn. We verwachten dat het gebruik van onze applicatie een piek zal vertonen in de avond en in het weekend, omdat er voornamelijk buiten werk en schooltijd gesport wordt. Er is voor onze applicatie dus meer dal- dan piek-belasting en we verwachten dus ook dat een PaaS oplossing goedkoper zal zijn dan een cluster.

In tabel C.1 ziet u een overzicht van de verschillende grote cloud platformen die er momenteel zijn, en de talen en frameworks die ze ondersteunen [11–13]. Zoals in de tabel te zien is ondersteunen de meeste PaaSś vele talen en frameworks. Microsoft Azure en Amazon AWS ondersteunen de 4 grote platformen .NET, Java, Python en NodeJS.

De bestaande systemen van Emando zijn gemaakt met het .NET framework en draaien voornamelijk in Microsoft Azure als Cloud Service. Aangezien MyLaps voor ons een belangrijke databron is en de MyLaps SDK alleen voor C#/.NET beschikbaar is, kan er veel werk bespaard worden door ook deze programmeertaal en runtime te gebruiken.

Bovendien heeft Emando een laag over de MyLaps SDK heen geschreven waardoor de SDK bruikbaar is met Reactive Extensions (Rx). Rx - mede ontwikkeld door prof.dr. H.J.M. Meijer [14] - is een manier om LINQ te gebruiken voor observeerbare collecties en asynchroon programmeren, twee eigenschappen waar we binnen de applicatie mee te maken gaan krijgen.

Bij onze keuze hebben ook de andere platformen overwogen, maar door de beperkte beschikbaarheid van de MyLaps SDK (alleen binnen .NET) en het gebruik van .NET door Emando waren dit minder bruikbare opties.

C.4.6. Realtime

De trainingsapplicatie moet zonder dat de gebruiker de applicatie handmatig ververst de laatste doorkomsten binnenkrijgen en tonen. Ook moet de getoonde rondetijd up to date zijn, en niet een ronde tijd van een vorig rondje. De communicatie tussen de back-end en de clients moet dus realtime zijn. Realtime wil zeggen dat de server een update naar de client kan sturen. In 'normaal' web verkeer kan de server niet op elk moment een bericht sturen naar de client. De server kan alleen reageren op een verzoek van de client, omdat de client geen vast IP-adres heeft en er standaard geen poort open staat. Pas wanneer de client een uitgaand verzoek heeft gaat er een poort open.

Een oplossing hiervoor is WebSockets: bij WebSockets blijft de verbinding open en dus de poort en kan de server op elk moment een bericht sturen, als er bijvoorbeeld een schaatser voorbij een detectielus komt. Een verbinding openhouden kost geheugen op de server en de server heeft dus een maximum aantal verbindingen dat hij kan openhouden. Er bestaan verschillende WebSocket frameworks die je het aanmaken en onderhouden van de verbindingen uit handen nemen. Een hiervan is SignalR voor .NET. Emando heeft SignalR[1] succesvol gebruikt tijdens de Olympische Spelen van 2014 om `http://live.schaatsen.nl` te voorzien van live data. SignalR is inmiddels een officiele ASP.NET library en wordt ontwikkeld en ondersteund door Microsoft ontwikkelaars.

C.5. Client

Er zijn tegenwoordig diverse manieren om mobiele applicaties te maken, voor diverse platformen (iOS, Android, Windows Phone). Deze verschillende methodes vereisen verschillende programmeertalen en architecturen. Een groot verschil tussen de verschillende methodes is de hoeveelheid code die kan worden hergebruikt tussen verschillende platformen. De uitersten hiervan gebruiken een compleet gescheiden broncode ten opzichte van een volledig gedeelde broncode. De volgende methodes zijn meegenomen in onze ontwerpkeuze:

Native applicatie: Een native applicatie is een applicatie die ontworpen is voor een specifiek platform. Het design van dergelijke applicaties ligt (meestal) in het verlengde van de richtlijnen voor dit platform en het gebruik van platform specifieke handelingen (gestures) wordt gestimuleerd en gehanteerd binnen de applicatie. Vanwege de richtlijnen en het gebruik van deze gestures, zijn dergelijke applicaties zelfverklarend voor eindgebruikers. Een ander voordeel is dat door de native implementatie, de user interface performance van dergelijke applicaties doorgaans erg goed is.

Een nadeel van native applicaties is dat applicatie code vaak in een platform specifieke programmeertaal wordt ontwikkeld en dat er daardoor 3 complete applicaties ontwikkeld moeten worden om de drie grootste platformen te ondersteunen (iOS, Android en Windows Phone). Het ontwikkelen hiervan kost daardoor vaak veel tijd.

Web applicatie: Een web applicatie is een applicatie die ontworpen is om op alle (mobiele) platformen te draaien. Deze applicatie is benaderbaar via het web (m.b.v. browsers) en heeft als groot voordeel dat deze maar 1x ontwikkeld hoeft te worden. Ook zijn veel ontwikkelaars bekend met de programmeertalen waarmee gewerkt moet worden (HTML, CSS en JavaScript).

Er zijn veel verschillende frameworks om mobiele applicaties te maken en daar ligt ook meteen een groot gevaar: elk framework heeft zijn eigen voordelen en wisselen tussen frameworks kan vaak betekenen dat grote delen van de applicatie opnieuw geschreven moeten worden. Ook tijdens ons project zijn diverse frameworks (Meteor, Angular, Backbone, Ember) ter sprake gekomen en voor elk genoemd framework wist er een projectlid voor- en nadelen te noemen.

Frameworks: jQuery Mobile [15], Sencha Touch [16], Meteor [17], AngularJS [18], Backbone.js [19], Ember.js [20], Ionic [21], React [22]

Native applicatie met WebView: Een ander belangrijk nadeel van web applicaties is het feit dat deze niet vindbaar zijn in de app stores. Native applicaties met een WebView[2] zijn applicaties

[1] `http://www.asp.net/signalr`
[2] Een WebView is een soort van browser binnen een applicatie, zonder de mogelijkheid zelf een url in te voeren.

waarin met behulp van een webview een reeds bestaande web applicatie geladen wordt. Hiermee kan met relatief weinig tijd worden meegeleund op het model van de applicatie winkels en is de applicatie in een klap vindbaar op de plek waar gebruikers zoeken naar applicaties.

Een nadeel van deze applicaties is hun performance. De applicatie laadt steeds webpagina's in en bij het wegvallen van de verbinding kan de applicatie doorgaans geen of weinig informatie of user interface tonen. Ook kan er doorgaans geen of weinig gebruik gemaakt worden van gestures. In sommige implementaties van deze applicaties wordt een onderscheid gemaakt tussen de offline user interface en de online web app, wat de performance verbetert en het probleem met het wegvallen van de verbinding oplost. De prestaties, vormgeving en gestures zijn echter vele malen minder praktisch ingericht dan bij een native applicatie. Het uitbrengen van een applicatie binnen de applicatie winkels brengt echter wel verwachtingen met zich mee. Doorgaans gaan gebruikers er vanuit dat de bovenstaande beschreven eigenschappen wel verwerkt zitten binnen deze applicaties. **Frameworks:** PhoneGap/Cordova [23], Titanium [24]

Cross Compiling: Cross Compiling is een techniek om broncode om te zetten naar uitvoerbare bestanden voor de verschillende platformen. Hiermee kan vaak de business- en servicelaag van de applicatie gedeeld worden tussen platformen, en sommige 'cross-compilers' ondersteunen ook het delen van de presentatielaag. Vaak is dit echter niet wenselijk, omdat juist op de presentatielaag veel verschil gemaakt kan worden tussen platformen, om beter aan te sluiten bij de verwachtingen van de gebruiker. Met cross compiling wordt tijd bespaard op de business- en servicelaag, welke gebruikt kan worden om een presentatielaag te maken voor meerdere platformen. Applicaties die gemaakt zijn door middel van Cross Compilation zijn niet te onderscheiden van native applicaties (gemaakt in de programmeertaal van het platform) want de uitvoerbare bestanden zijn binair vrijwel gelijk.

Een nadeel van Cross Compiling is dat het opzetten van de architectuur direct op een juiste manier dient te gebeuren, hetgeen tijd extra tijd kan kosten. Het later aanpassen van de architectuur heeft tot gevolg dat er voor alle platformen een update nodig is voordat het project weer succesvol compileert. Ook moet er in tegenstelling tot een web app nog steeds voor meerdere platformen ontwikkeld worden.

Frameworks: Xamarin [25], Adobe AIR [26], XMLVM [27], Apportable [28]

C.5.1. Xamarin

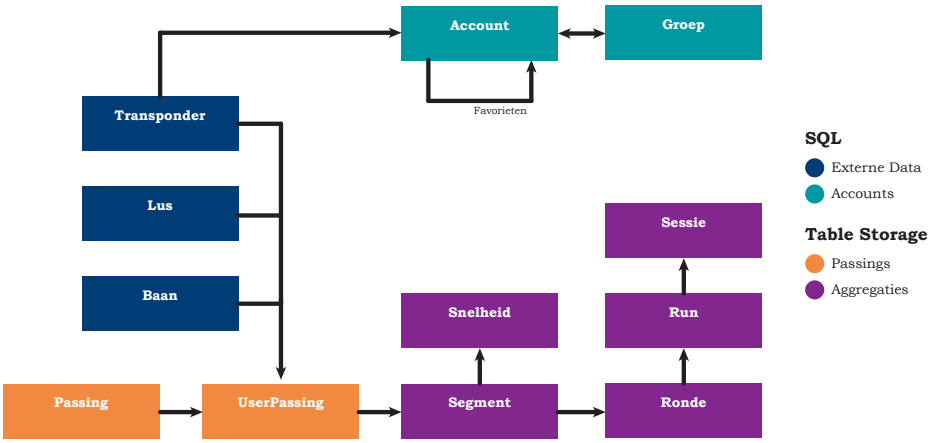
Wisselen tussen methodiek betekent het opnieuw ontwikkelen van de applicatie. Omdat onze applicatie niet alleen een proof-of-concept is, maar bedoeld is voor echt gebruik en doorontwikkeling, is het economisch voordeliger om direct voor de uiteindelijke methodiek te kiezen.

Omdat performance en responsetijd een belangrijke rol spelen binnen onze applicatie, ligt het voor de hand om voor een native of gelijkwaardig presterende aanpak te gaan. Een implementatie met WebView biedt niet de performance waar we naar op zoek zijn.

In overleg met de opdrachtgever is besloten de applicatie te ontwikkelen voor één platform met een Cross Compiling-implementatie. Het ontwikkelen van deze applicatie (met de gestelde eisen) voor meerdere platformen kost meer tijd dan ontwikkelen voor een enkel plaftform, en zou ten koste gaan van het MVP. Door een goede business- en servicelaag op te zetten, is het later mogelijk om de applicatie uit te rollen naar andere platformen, door enkel de platform specifieke views nog te ontwikkelen.

We hebben voor Xamarin boven andere Cross Compiling implementaties gekozen omdat Xamarin werkt met het .NET framework,vanwege de .NET SDK van MyLaps en de .NET implementaties van Emando.Dit brengt een paar belangrijke voordelen met zich meebrengt:

- Het platform .NET biedt LINQ en Reactive Extensions (Rx). Met LINQ en Rx kunnen eenvoudig streams aan binnenkomende data efficient afgehandeld worden. Aangezien dit een belangrijk onderdeel is van onze applicatie is dit een groot voordeel.



Figuur C.1: Entiteiten binnen de applicatie

- Aangezien we onze server zullen inrichten met SignalR is het fijn dat er al een SignalR component bestaat voor Xamarin. Dit maakt de communicatie met de server een 'no-brainer'.
- Emando gebruikt Team Foundation Server (TFS) en de integratie van Xamarin met Visual Studio (en daarmee TFS) zal latere doorontwikkeling en aansluiting op de bestaande werkwijze ten goede zou komen.

C.5.2. iOS

Vanwege de projectduur is besloten de applicatie aanvankelijk voor slechts 1 platform uit te rollen en dusdanig in te richten zodat deze later gemakkelijk uitgebreid kan worden naar andere platformen.

Omdat er voor slechts één platform een applicatie uitgerold wordt, is de keuze voor dit platform niet onbelangrijk. Het kiezen voor het platform is gebeurd aan de hand van de volgende criteria:

- Aantal schaatsers die het platform gebruiken
- Moeilijkheidsgraad om naar platform uit te rollen

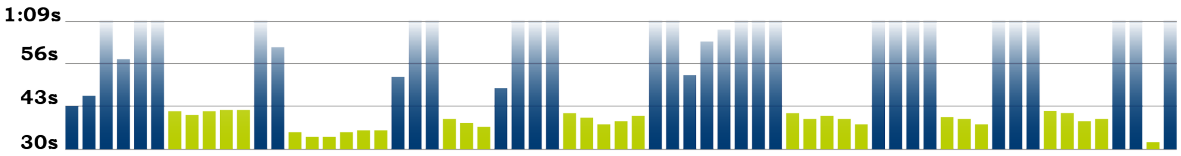
De moeilijkheidsgraad om naar een specifiek platform uit te rollen bleek bij alle platformen nagenoeg even hoog. Platformen als *Windows Phone*[3] en *BlackBerry*[4] hebben niet genoeg market share en aantal gebruikers om het uitrollen van de applicatie rendabel te maken. *Android*[5] en *iOS*[6] hebben een grote market share en veel actieve gebruikers. Van de medewerkers van de schaatsbond beschikt de meerderheid over een mobiel aparaat dat *iOS* draait, hetgeen de doorslaggevende factor is geweest om aanvankelijk de applicatie eerst voor *iOS* te ontwikkelen.

C.6. Entiteiten

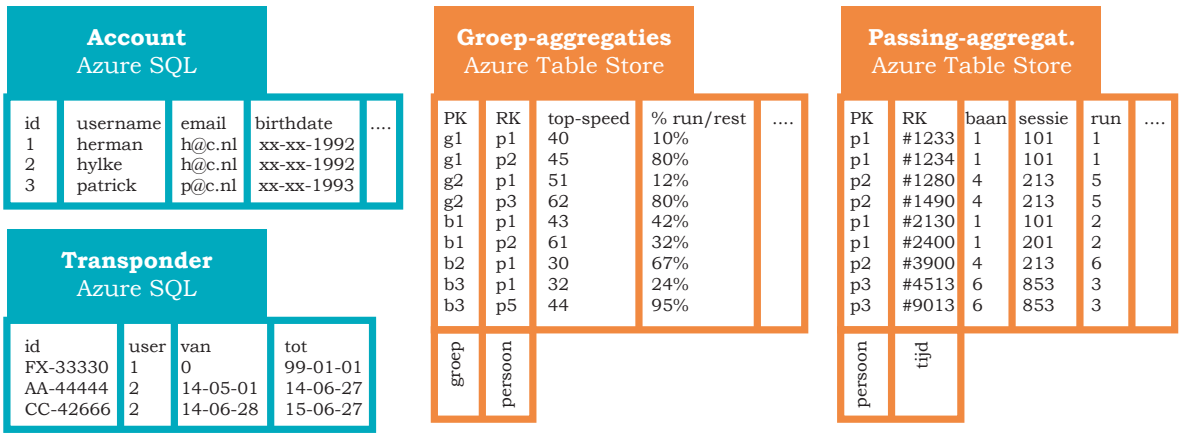
De data binnen onze applicatie bestaat uit verschillende entiteiten. In figuur C.1 is de relatie tussen de entiteiten afgebeeld. De entiteit op het kleinste niveau is een *passing*, de doorkomst bij een detectielus. Een doorkomst heeft velden voor het moment van registratie, welke lus detecteerde, welke baan de lus staat en welke transponder er bij hoort. passings komen uit de MyLaps SDK en zijn dus externe data, die we wel zelf moeten opslaan. De MyLaps SDK is namelijk niet bedoeld als database om passings te selecteren, maar slechts om realtime passings te verkrijgen.

Wanneer een passing ons systeem binnenkomt proberen we deze te koppelen aan een gebruiker. Als passing bij een voor ons onbekende transponder hoort slaan we de passing op en verwerken

[3] http://www.windowsphone.com/
[4] http://www.blackberry.com
[5] http://www.android.com/
[6] http://www.apple.com/nl/ios/



Figuur C.2: De rondetijden tijdens een typische training; specifiek: training van Herman van 3 maart 2014



Figuur C.3: Verschillende data, verschillende database engines

we hem later, als we de betreffende transponder wel kennen. Dit kan gebeuren als een gebruiker met terugwerkende kracht een transponder koppelt aan zijn account. Elke gebruikersaccount heeft dus een lijst met transponders en vanaf wanneer tot wanneer deze transponder in het bezit van de gebruiker was. Een transponder kan namelijk ook worden uitgeleend. Per tijdseenheid kan er per gebruiker maar één transponder zijn.

Aan de hand van een lijst met passings kunnen we enkele berekeningen doen. Met deze berekeningen kunnen we bijvoorbeeld bepalen wanneer een training begint en eindigt. Een groep van de passings van een training heet een Session. Binnen een training zijn weer Runs te onderscheiden. Runs zijn groepjes van passings waartussen een sporter niet uitreed. Een training in het baansporten kan er bijvoorbeeld bestaan uit het 5 keer rijden van 4 ronden, met tussendoor 2 ronden rust. Er zijn in dat geval binnen de sessie 4 runs te onderscheiden. In figuur C.2 is een training van Herman te zien. Bij Herman zijn rondetijden langzamer dan 40 seconden geen actieve onderdelen, maar zijn uitrijdrondjes. Dit is per sporter en sport verschillend, dus moet deze grens dynamisch bepaald worden of instelbaar zijn. In de figuur zijn de runs groen gekleurd.

C.7. Database

Zoals in figuur C.1, is een deel van onze data relationeel. De gebruikers zijn gekoppeld aan transponders en transponders zijn gekoppeld aan passings. Gebruikers kunnen deel uitmaken van groepen en kunnen een lijst met favoriete gebruikers hebben, welke ze willen volgen. Een deel van de data zullen we opslaan in een relationele database. Data die relatief weinig wordt aangepast zoals de gebruikeraccounts en bijbehorende transponders leent zich hier uitstekend voor. Met een *join* is eenvoudig een lijst van transponders van een gebruiker op te halen. In figuur C.3 is (een abstractie en deel van) de relationele data weergegeven in het blauw. De user-id's uit de account-tabel komen terug in de transponder-tabel.

Naast de relationele data, genereren we met aggregaties heel veel nieuwe data, zoals de snelste rondetijd, de langste run, of de hoogste snelheid per persoon. Deze data is gebonden aan de groepen waarin een gebruiker zit en de banen waarop de passing plaatsvond. In samenspraak met de opdrachtgever is besproken om hiervoor Azure Table Storage te gebruiken. Azure Table Storage heeft

twee keys per rij, de Partition Key en de Row Key [29]. Wij slaan de (virtuele) groep op in de Partition Key (PK) en de gebruikers in de Row Key (RK). Op die manier is snel voor een groep alle data op te halen, voor bijvoorbeeld een Leaderboard. Ook kan bij het opslaan van de aggregaties snel op de Row Key gefilterd worden. De verschillende aggregaties worden opgeslagen in verschillende kolommen. Azure Table Storage biedt de namelijk mogelijkheid arbitraire objecten te koppelen aan een Partition Key en Row Key (de combinatie van de twee keys is uniek). In figuur C.3 is (abstractie van) de platte data weergegeven in het oranje. In de passing aggregatie tabel is de unieke key gebaseerd op de transponder en de timestamp van de passing. Een transponder kan niet twee keer gedetecteerd worden op hetzelfde moment zonder dat het dezelfde passing is. Aggregaties per passing zoals bij welke run en sessie de passing hoorde slaan we hier in op.

Deze data wordt realtime verwerkt en doorgestuurd naar de client wanneer nodig. Om deze aggregaties snel en onafhankelijk van elkaar te laten verlopen, moet rekening worden gehouden met de mogelijkheid om bij pieken meerdere instanties van deze aggregatie-pijpen te draaien. Elke instantie ontvangt dan andere passings. Om te zorgen dat de client altijd de laatste versie ontvangt wordt elke aggregatie vergezeld van de *timestamp* van de laatste passing. Als een client een aggregatie ontvangt met een oudere timestamp dan degene die hij al ontvangen heeft dan kan hij deze negeren.

C.8. Architectuur

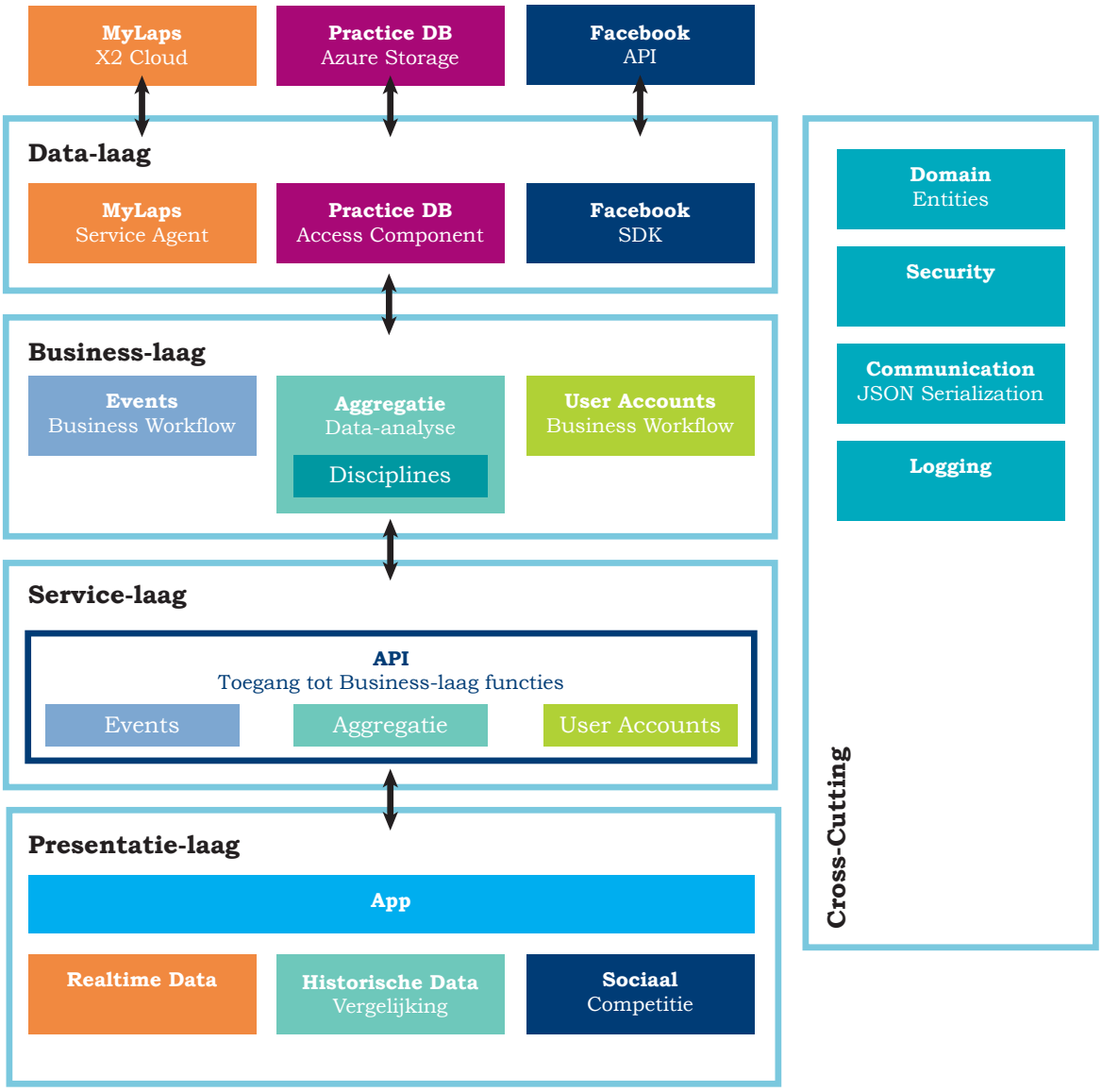
In overleg met de opdrachtgever is gekozen voor een multi-layered architectuur met vier lagen: de datalaag, businesslaag, servicelaag en de presentatielaag. Deze architectuur garandeert een scheiding tussen verschillende stukken logica, zodat alle logica in de verantwoordelijke laag wordt uitgevoerd. In figuur C.4 zijn alle layers en hun componenten weergegeven.

De datalaag bevat componenten die een verbinding kunnen maken met de databases en andere services zoals MyLaps Cloud en de Facebook API. De datalaag zet ruwe data uit de database om in domein entiteiten en levert deze af bij de businesslaag.

De businesslaag bevat alle workflows die op de server uitgevoerd moeten worden. De workflows zijn in drie categorieen op te delen. Als eerste de Aggregatie componenten die diverse aggregaties kunnen uitvoeren op data uit de Practice Database. Ten tweede is er de Event Workflow die bepaald wat er moet gebeuren als er een event binnenkomt zoals bijvoorbeeld een doorkomst vanaf MyLaps. In het geval van een doorkomst moet deze bijvoorbeeld worden opgeslagen in de Practice Database en moeten de aggregaties een iteratie doorlopen. Als derde is er het User Account component dat acties voor gebruikersacccounts mogelijk maakt. Zo is dit component belast met het opslaan van nieuwe users, het inloggen met wachtwoord of met Facebook, het beheren van groepen en het ophalen van vriendenlijsten.

De servicelaag bevat de API. De diverse aanroepen die de clients nodig hebben worden in de servicelaag gekoppeld aan de goede workflow uit de businesslaag. De servicelaag zorgt ook voor Caching, waar nodig.

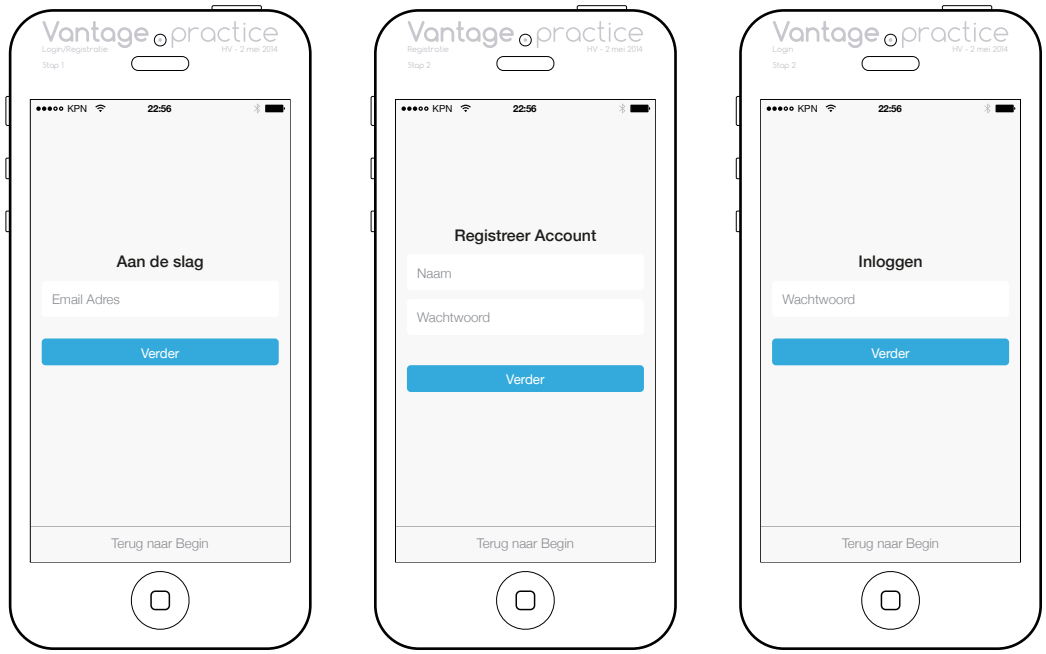
De presentatielaag bevindt zich in de mobiele applicaties. Deze toont de informatie uit de servicelaag. De presentatielaag bevat bijvoorbeeld User Interface componenten, weergaves voor diverse data types en menu's. Naast de weergave zit er in de apps ook een stuk servicelaag om te kunnen communiceren met de servicelaag op de server.



Figuur C.4: Systeem architectuur

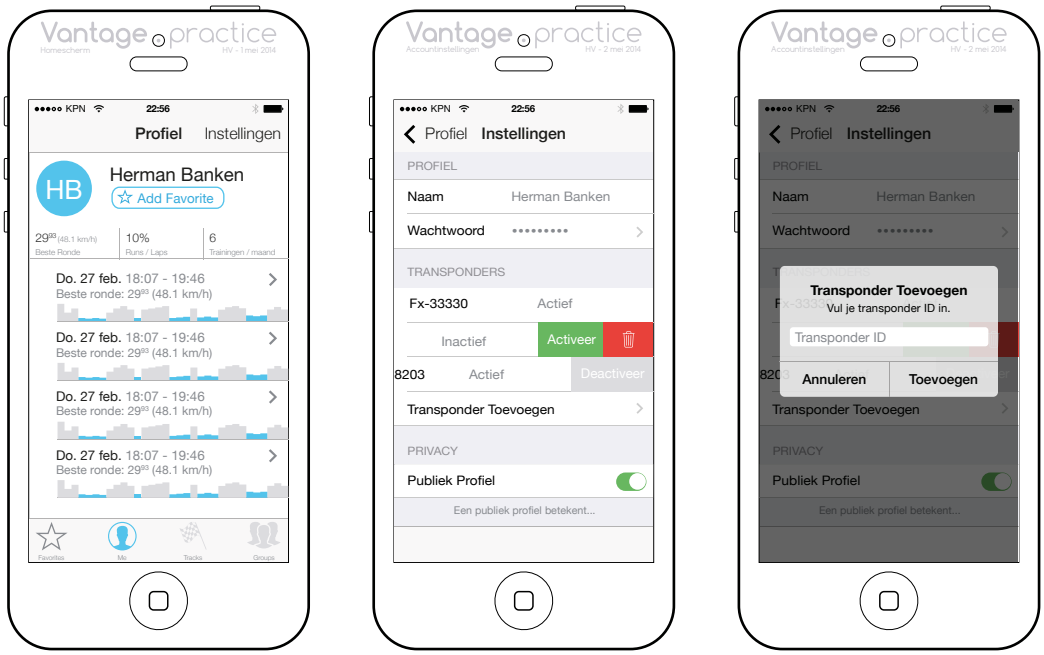
D

Interface Ontwerpen



(a) Beginscherm (b) Registratiescherm (c) Loginscherm

Figuur D.1: Login en registratie



(a) Profielscherm (b) Accountinstellingen (c) Transponder Toevoegen

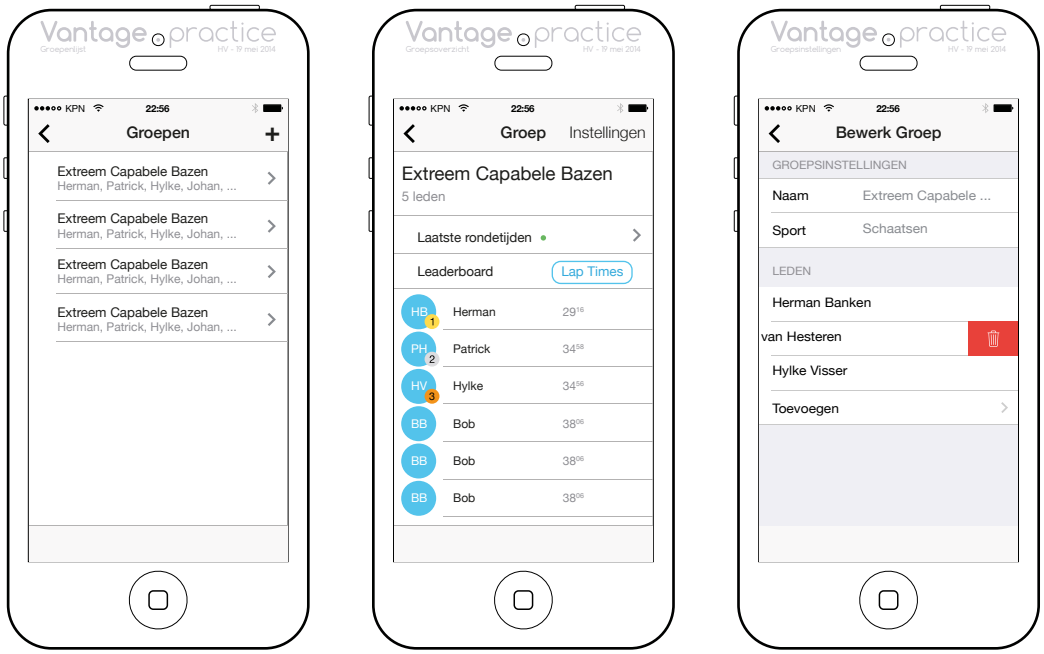
Figuur D.2: Profiel en accountinstellingen

(a) Sessieoverzicht (b) Sessieoverzicht met een opengeklapte run

Figuur D.3: Sessie-schermen

(a) Favorietenoverzicht (b) Favorieten Toevoegen

Figuur D.4: Favorieten-schermen



(a) Groepenoverzicht (b) Groep Detail (c) Groepsinstellingen

Figuur D.5: Groeps-schermen

E

Testresultaten

Onderstaande tabel bevat de resultaten van de enquête. In totaal zijn er 31 enquête-formulieren ingeleverd. Sommige vragen waren multiple choice en sommige op basis van een schaal naar belangrijkheid, waar 1 onbelangrijk is en 5 erg belangrijk.

	Distributie of	
	Gemiddelde	Standaard deviatie
Algemeen		
Leeftijd	24.1 jaar	9.0
Geslacht	70% man, 30% vrouw	
Sporten	96.7% Schaatsen, 66.7% Skeeleren, 6,7% Baanwielrennen	
Sporten overig	Hardlopen, weg-wielrennen, athletiek	
Telefoon	70.0% Android, 23.3% iPhone 4+ 3.3% iPhone 2G/3G, 3.3% Windows Phone	
Live informatie		
Eigen doorkomsten	4.53 uit 5	0.73
Andermans doorkomsten	3.47 uit 5	1.11
Opgeslagen data		
Inzien doorkomsten	4.57 uit 5	0.63
Inzicht in trainingen	4.30 uit 5	0.84
Sociaal		
Leaderboards	3.23 uit 5	0.94
Groepen maken	3.83 uit 5	0.91
Suggesties	2.73 uit 5	1.14
Facebook login	2.13 uit 5	1.20
Delen resultaten	2.67 uit 5	1.24
Delen via FB	2.00 uit 5	1.20
Delen via RunKeeper	1.63 uit 5	1.00
Gebruiksgemak		
Audio-cue	3.83 uit 5	1.26
Overig		
De app willen gebruiken	96.7% ja, 3.3% nee	
Wat de app mag kosten	€ 1.72	2.13

Verdere functionaliteit opgegeven door respondenten

- Tijdsverschillen tussen de rondjes (mobiel bij trainer en rondetijden gelijk terugkijken na oefening)
- Actuele snelheid zou wel vet zijn. Of een verwachte rondetijd op basis van de huidige snelheid.
- Trainingen en prestaties delen met endomondo
- Hardslagmeter, calorieën-schatter. Suggesties voor trainingschema's: lange termijn (intensievere trainingsweek of extensievere trainingsweek) en korte termijn (schaats-schema's die te laden zijn en aan de prestatie gekoppeld worden).
- Gebruiksvriendelijk, snelheid bv: snel opgestart, etc. De Hockey-WK-app is bijvoorbeeld erg traag en dat is echt niet fijn.
- Vooraf ingesteld trainingsschema (wat nu gaan doen?)
- Tussentijdse feedback op rijden (tijdens het rijden een piep voor elke "100"meter (niet echt honderd meter, maar herkenbare punten: ingaan bocht/rechte eind, passeren van 1000m start, enz als je een bepaald tempo moet rijden bijvoorbeeld.
- Tijdens training liever niet vergelijken met anderem, achteraf wel leuk.
- Misschien een mogelijkheid om je trainingsschema vooraf in te vullen. Dan kun je vervolgens live zien of je dat ook aan het bewerkstelligen bent of dat je niet doet wat je moest doen. Chearleaders die je aanmoedigen als je een beetje achterop raakt op je schema en het boze hoofd van John de Wolf als je gefaald hebt zijn bonus.
- Metingen op basis van gps en niet alleen via transponders (als dat mogelijk is, vooral handig voor skeeleren).
- Misschien een rondeteller (zoals die oude mannetjes altijd hebben) voor als je veel rondjes gaat rijden en het zou ook wel vet zijn als je ook je hartslag erbij kan zien, maar dat is voor de echte pro's!

Overige opmerkingen door respondenten

- Leuk bedacht, alleen jammer dat je er een transponder voor nodig hebt, wordt het gelijk wel duur. Zou dit ook met GPS kunnen?
- Het zou wel geniaal zijn als je via je oortjes je hele trainingsschema te horen krijgt en deze vervolgens rijdt en dat de app weet hoe ver je steeds bent. En het lijkt me een vooruitgang dat je real-time weet hoe je snel je de afstanden aflegt, zodat je bijvoorbeeld een vlak of oplopend schema kan rijden.
- Altijd met telefoon in de hand wordt de ijsbaan niet veiliger van. Denk goed na wanneer je je tel wel/niet in de hand hebt.
- Een van de belangrijkste punten is wat mij betreft de audiofeedback tijdens het sporten. Het terugkijken van gereden tijden is leuk, maar direct feedback tijdens het sporten werkt voor mij (uit ervaring met RunTastic apps voor hardlopen en wielrennen) erg motiverend.
- iPhone 4 en verder zijn in dit onderzoek onder een noemer gebracht. Ik merk nu al dat de RunTastic apps, die inmiddels geoptimaliseerd zijn voor iPhone 5C/S, behoorlijk trager werken op iPhone 4. Het zou geweldig zijn als deze app ook soepel runt op iPhone 4.
- Werkt de app ook zonder transponder, bijvoorbeeld op GPS?

Trainingsapp voor Baansporten

emando TU Delft Delft University of Technology

Voor het Bachelor Eindproject **Trainingsapp voor baansporten** vragen wij u om ons te helpen. Vul alstublieft het onderstaande formulier in, gebaseerd op uw verwachtingen van de applicatie. Bedankt voor uw tijd.

Waar gaat het over?

Wij, Hylke Visser, Patrick van Hesteren en Herman Banken, maken een applicatie die live de data van je transponder (van MyLAPS) toont in een smartphone applicatie. Voor ons project maken we een iPhone versie, maar een Android versie behoort zeker tot de mogelijkheden.

Algemeen

Naam

Leeftijd

Geslacht ○ Man ○ Vrouw

Datum

Welke van de volgende sporten beoefent u?
- ☐ Schaatsen
- ☐ Baanwielrennen
- ☐ Skeeleren

Bij welke andere sporten zou u deze App willen gebruiken?

Wat voor een telefoon heeft u? ○ iPhone 4 of hoger ○ iPhone 2G/3G(S) ○ Android ○ Windows Phone ○ Anders

Live Informatie

Het is de bedoeling dat de app live rondetijden en snelheden zou kunnen tonen. In hoeverre vindt u de onderstaande functionaliteiten belangrijk voor de app?

Live inzien van uw eigen doorkomsten en rondetijden in de app	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Live inzien van andersmans doorkomsten en rondetijden in de app	Niet belangrijk	○	○	○	○	○	Zeer belangrijk

Opgeslagen gegevens

De app zou ook historische gegevens moeten kunnen tonen, zoals bijvoorbeeld eerdere trainings-sessies. In hoeverre vindt u de onderstaande functionaliteiten belangrijk voor de app?

Inzien van eerdere doorkomsten en rondetijden	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Inzicht in trainingen (trainings-sessies, oefeningen/runs, rustperiodes, rondetijden, etc.)	Niet belangrijk	○	○	○	○	○	Zeer belangrijk

Sociaal

De app zou mogelijkheden kunnen bieden om trainingen en prestaties van anderen te zien, bijvoorbeeld in de vorm van leaderboards of door het delen van prestaties op sociale netwerken. In hoeverre vindt u de onderstaande functionaliteiten belangrijk voor de app?

Leaderboards gerangschikt naar prestaties en disciplines, zoals rondetijden, snelheden, afstanden etc.	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Mogelijkheid om groepen te maken waarin u andere sporters kunt volgen	Niet belangrijk	○	○	○	○	○	Zeer belangrijk

Suggesties om met sporters van vergelijkbaar niveau te gaan sporten	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Uw Facebook account koppelen aan uw account	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Trainingen en prestaties delen met anderen	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Trainingen en prestaties delen met Facebook vrienden	Niet belangrijk	○	○	○	○	○	Zeer belangrijk
Trainingen en prestaties delen met op Runkeeper	Niet belangrijk	○	○	○	○	○	Zeer belangrijk

Gebruiksgemak

Audio-cue met informatie over doorkomsten	Niet belangrijk	○	○	○	○	○	Zeer belangrijk

Overig

Welke functionaliteit zou u verder graag terugzien in de app?

Zou u deze app willen gebruiken bij uw trainingen? ○ Ja ○ Nee

Hoe veel geld zou deze app moeten kosten?

Opmerkingen / Toevoegingen:

Als de app bijna af is, zouden we graag een gebruikerstest willen uitvoeren. Vul hier uw email adres in als wij later contact met u mogen opnemen voor deze test.

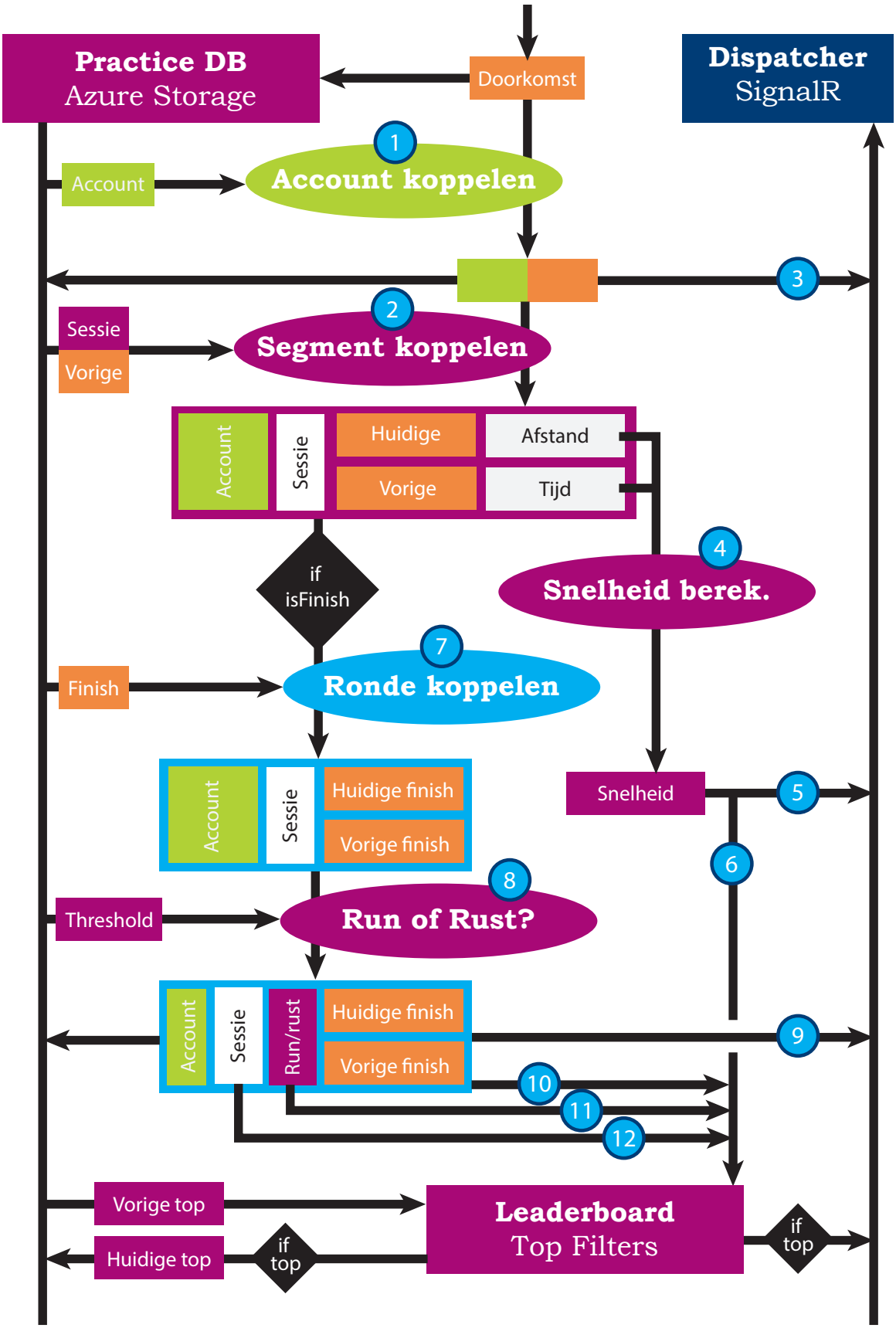
Email adres

Bedankt voor het nemen van de tijd voor het invullen van deze enquête. Uw feedback wordt ten zeerste gewaardeerd!

Verzenden

F

Aggregatie Flow



Figuur F.1: Flow-diagram van het aggregatie proces

Bibliografie

[1] D. Clegg and R. Barker, *Case Method Fast-Track: A Rad Approach* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994).

[2] *Key words for use in rfcs to indicate requirement levels,* http://www.ietf.org/rfc/rfc2119., accessed: 2014-06-20.

[3] E. Ries, *Venture hacks interview: "what is the minimum viable product?",* (2009).

[4] K. Schwaber and M. Beedle, *Agile software development with scrum,* (2002).

[5] M. Fowler and J. Highsmith, *The agile manifesto,* Software Development **9**, 28 (2001).

[6] *MSDN - Entity Framework,* http://msdn.microsoft.com/nl-nl/data/ef.aspx (), accessed: 2014-06-08.

[7] *Setting up Entity Framework for production use,* http://www.dotnetcurry.com/showarticle.aspx?ID=820 (), accessed: 2014-06-08.

[8] R. Likert, *A technique for the measurement of attitudes.* Archives of psychology (1932).

[9] W. L. Hürsch and C. V. Lopes, *Separation of Concerns*, Tech. Rep. (1995).

[10] *Key principles of software architecture,* http://msdn.microsoft.com/en-us/library/ee658124.aspx, note = Accessed: 2014-06-19.

[11] *PaaS Providers List: 2014 Comparison And Guide,* http://www.tomsitpro.com/articles/paas-providers,1-1517.html, accessed: 2014-04-28.

[12] *Using Scala and the Play Framework in Windows Azure,* http://blog.smarx.com/posts/using-scala-and-the-play-framework-in-windows-azure (), accessed: 2014-04-28.

[13] *Amazon Web Services (AWS) - Getting Started,* https://aws.amazon.com/developers/getting-started/, accessed: 2014-04-28.

[14] E. Meijer, *The world according to linq,* Queue **9**, 60 (2011).

[15] *jQuery Mobile,* http://jquerymobile.com/, accessed: 2014-04-23.

[16] *Sencha Touch,* http://www.sencha.com/, accessed: 2014-04-23.

[17] *Meteor,* http://www.meteor.com/, accessed: 2014-04-23, Real-time Javascript Web App Framework.

[18] *AngularJS,* https://angularjs.org/, accessed: 2014-04-23, JavaScript MVVM framework by Google.

[19] *Backbone.js,* http://backbonejs.org/, accessed: 2014-04-23, Lightweight JavaScript Model–view–presenter framework by DocumentCloud.

[20] *Ember.js,* http://emberjs.com/, accessed: 2014-04-23.

[21] *Ionic,* http://ionicframework.com/, accessed: 2014-04-23.

[22] *React,* http://facebook.github.io/react/, accessed: 2014-04-23, JavaScript UI framework by Facebook.

[23] *Apache Cordova,* https://cordova.apache.org/, accessed: 2014-04-23, formerly known as PhoneGap. PhoneGap is now a distribution of Apache Cordova.

[24] *Titanium,* http://www.appcelerator.com/titanium/, accessed: 2014-04-23.

[25] *Xamarin,* http://xamarin.com/, accessed: 2014-04-23.

[26] *Adobe AIR,* http://www.adobe.com/nl/products/air.html, accessed: 2014-04-23.

[27] *XMLVM,* http://xmlvm.org/, accessed: 2014-05-01.

[28] *Apportable,* http://www.apportable.com/, accessed: 2014-05-01.

[29] *Microsoft azure table storage – not your father's database,* http://msdn.microsoft.com/en-us/magazine/ff796231.aspx (), accessed: 2014-05-01.