Bachelor Informatica
Universiteit van Amsterdam

FNWI Rooster App

Sven Meyer

19 juli 2012

Begeleider(s): Robert Belleman (UvA)

Ondertekend:

Samenvatting

Dit document beschrijft de opzet, voortgang en afsluiting van het bachelor afstudeerproject rondom de ontwikkeling van een mobiele roosterapplicatie voor de FNWI. Een analyse van de praktische situatie, waaronder marktaandeel van smartphoneplatforms en de mogelijkheden van de digitale infrastructuur van de UvA wordt beschreven. Hieruit wordt een set functionele eisen opgesteld. De applicatie dient het persoonlijke rooster te kunnen weergeven en als catalogus te worden gebruikt. De ontwikkeling van de applicatie geschied met behulp van de Titanium SDK, dat de ontwikkelaar in staat stelt de applicatie voor Android en iOS te ontwikkelen. De implementatie van deze applicatie wordt in detail beschreven. Aan de functionele eisen wordt niet voldaan: de iPhone applicatie is nagenoeg functioneel, maar de Android versie niet. Oorzaken zijn onderschatting van het werk, teveel focus op uiterlijk ten koste van functie en misschatting van de Titanium SDK.

Inhoudsopgave

1 Inleiding **4**

2 Probleemstelling **5**

2.1 Uitwerking 5

2.1.1 DataNose 5

2.1.2 Gebruikers en platforms 6

2.2 Functionele eisen 7

3 Gerelateerd Onderzoek **9**

4 Ontwerp **10**

4.1 Functioneel Ontwerp 10

4.1.1 Persoonlijk rooster 10

4.1.2 Zoeken naar roosters 11

4.1.3 Indeling en navigatie 12

4.2 Technisch ontwerp 13

4.2.1 Ontwikkeltools 13

4.2.2 Architectuur 16

4.2.3 Componenten 16

5 Implementatie **19**

5.1 Titanium 19

5.2 Interne Representatie 20

5.3 OData Adapter 20

5.3.1 DataNose webservice 20

5.3.2 Adapter 21

5.4 Model 22

5.4.1 Communicatie . . . 22
5.4.2 Activity Model . . . 22
5.5 Cache . . . 23
5.5.1 Synchronisatie . . . 23
5.6 Controller . . . 24
5.7 View Controllers . . . 24
5.8 Tests . . . 24

6 Resultaten **26**
6.1 Functionaliteit . . . 26
6.1.1 iOS . . . 26
6.1.2 Android . . . 27
6.1.3 Gebruikersvriendelijkheid . . . 27
6.2 Prestaties . . . 27
6.2.1 Interpretatie . . . 27
6.3 Titanium . . . 28

7 Conclusie **30**
7.1 Reflectie . . . 30
7.2 Aanbevelingen . . . 31

8 Bibliografie **32**

A Data Modellen **33**
A.1 Datanose EDM . . . 33
A.2 Cache Data Model . . . 34

B Marktaandeel **35**

HOOFDSTUK 1

Inleiding

In de ontwikkelingen binnen de consumentenelektronica hebben de computers in het recente verleden en nu nog steeds een sterke boventoon gevoerd. Sinds een aantal jaren is er een nieuw soort apparaat bijgekomen: de smartphone. Bellen en sms'en, spelletjes spelen en internetten kunnen nu allemaal met 1 apparaat. Smartphones zijn de nieuwe computers, de taken die alleen toevertrouwd kunnen worden aan een computer, worden naar mate de tijd verstrijkt meer uitzondering dan regel.

De Universiteit van Amsterdam heeft een website, een online studiegids en rooster, studenten kunnen zich online inschrijven voor vakken, lesmateriaal wordt via het web beschikbaar gemaakt en de digitale bibliotheek stelt de student in staat binnen enkele muisklikken door de enorme hoeveelheid aan (digitale) bronnen te zoeken. De digitale dienstverlening van de UvA staat ook zeker niet meer in de kinderschoenen, alles is afgestemd op de computergebruikende en met Internet verbonden student.

De alomtegenwoordigheid van smartphones onder de UvA-studenten staat echter in schril contrast met de schaarste van het voor deze apparaten geschikte aanbod. Via de browser op de smartphone zijn de diensten van de UvA wel beschikbaar, maar niet op een wijze die voor de touchscreengebruiker praktisch is te noemen.

Dit project is erop gericht deze situatie te veranderen. Een mobiele applicatie die snel en gebruiksvriendelijk toegang geeft tot de digitale dienstverlening van de UvA zal een uitkomst kunnen bieden. In het kader van het afstudeerproject Informatica wordt er gekeken naar de invulling van de mobiele informatievoorziening van de UvA met behulp van de ontwikkeling van een mobiele applicatie.

De realisatie hiervan kan vele vormen aannemen, maar er zijn ook beperkende factoren. Een mobiele applicatie is van externe bronnen afhankelijk om de gebruikers van de gevraagde gegevens en/of diensten te voorzien. De aanwezigheid en hoedanigheid van deze bronnen bepalen voor een groot deel de mogelijkheden van een mobiele applicatie. Daarnaast hebben de studenten van de universiteit een reeks aan soorten smartphones, waarmee, zo mogelijk, allen een dergelijke mobiele applicatie moeten kunnen gebruiken. De verschillende mobiele platforms brengen elk ontwikkelsoftware en programmeertalen met zich mee en verschillende manieren om een applicatie te implementeren. Er is maar een beperkte hoeveelheid aan tijd en er zullen dus keuzes gemaakt moeten worden.

Dit document beschrijft de opzet, voortgang en afsluiting van dit project. Daarnaast zal er vooruitgeblikt worden op de mogelijkheden voor toekomstige projecten.

HOOFDSTUK 2

Probleemstelling

Het project heeft als doel de digitale dienstverlening van de UvA aan te vullen met een mobiele applicatie. Er is nog alle kanten op te gaan met deze probleemstelling: er zijn legio manieren om invulling te geven aan de wens naar "een" mobiele applicatie en de technische uitwerking ervan. Om het speelveld te verkleinen en een realistische set eisen voor een mobiele applicatie op te kunnen stellen dient er gekeken te worden naar de beperkende factoren van de praktische situatie.

2.1 Uitwerking

Aan de hand van de volgende vragen kan het aantal mogelijkheden beperkt worden:

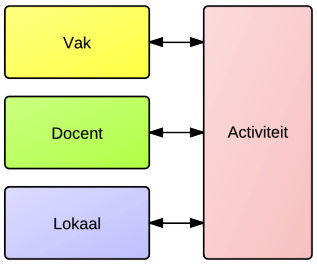
- Wat voor mogelijkheden biedt de bestaande digitale infrastructuur?
- Wie zijn de gebruikers van de digitale dienstverlening?
- Voor welk(e) platform(s) moet de applicatie ontwikkeld worden?

Wat betreft de meeste soorten diensten bestaat de digitale infrastructuur over het algemeen uit websites en webapplicaties waar de gebruiker direct mee werkt (bijv. online studiegids en SIS). Ze vormen geen gestructureerde informatievoorziening voor applicaties en bieden geen API aan voor het uitvoeren van administratieve taken.

2.1.1 DataNose

Een uitzondering in het digitale landschap van de UvA is DataNose [13]. DataNose is een systeem dat de enorme hoeveelheid informatie over de studieactiviteiten en -voortgang beschikbaar maakt aan studenten en docenten via een webapplicatie. Deze informatie behelst bijvoorbeeld het rooster, dat op vele verschillende manieren te doorzoeken is, verschaft studenten met overzichten van hun behaalde resultaten en hun persoonlijke rooster. De informatie uit DataNose heeft slechts betrekking op de studieactiviteiten binnen de FNWI. De webfrontend zelf biedt geen gestructureerde informatie, maar DataNose biedt ook informatie aan via een web*service*. Deze webservice is een relatief nieuwe ontwikkeling die in 2011 in het leven is geroepen ten behoeve van een bachelorproject vergelijkbaar met dit project [11]. De DataNose webservice biedt echter alleen informatie met betrekking tot de roosters.

In essentie gaat het om een catalogus van vakken, docenten, lokalen en activiteiten (colleges) die aangeboden wordt met behulp van het OData protocol. De details van dit protocol worden nader toegelicht in hoofdstuk 5: Implementatie. Er kan naar vakken, docenten en lokalen gezocht worden op basis van attributen zoals bijvoorbeeld de naam (van een vak, docent of lokaal) of type (van een activiteit). Vakken zijn eveneens op te vragen op basis van inschrijving (met behulp van een collegekaartnummer). Alle soorten items zijn daarnaast ook op basis van de relaties ertussen op te vragen, deze relaties zijn geïllustreerd in figuur 2.1.



Figuur 2.1: De relaties tussen de soorten items

We zijn door de bestaande infrastructuur dus beperkt tot het presenteren van roosterinformatie aan de studenten, en de gebruikers van de te ontwikkelen mobiele applicatie zullen dus studenten van deze faculteit zijn.

2.1.2 Gebruikers en platforms

De derde vraag, keuze van platform, is afhankelijk van het marktaandeel van de verschillende smartphoneplatforms onder de doelgroep. De gebruikers van de DataNose website zullen een redelijke afspiegeling zijn voor de algemene studentenpopulatie van de FNWI. De server waarop de DataNose website draait houdt bij wat de "User-Agent string" is van de client, elke keer als er een HTTP-request binnenkomt. Hierdoor is het mogelijk een beeld te krijgen van het browsermarktaandeel onder de bezoekers van DataNose. Browsers van smartphones hebben een andere User-Agent string dan desktopbrowsers, op deze manier kan er achterhaald worden welke mobiele besturingsystemen er gebruikt worden door de bezoekers van DataNose.

Tabel 2.1: Marktaandeel OS'en onder de gebruikers van DataNose

OS	Bezoekers	Bezoekers met handheld
Windows/Linux/Mac	61,43 %	
iOS	18,98 %	56,24 %
Android	13,94 %	41,30 %
Onbekend	0,81 %	
BlackBerry	0,66 %	1,96 %
Bada	0,17 %	0,50 %
Symbian	0,12 %	0,12 %
Totaal handheld	33,87 %	100,00%
Totaal	96,11%	100,00%

Tabel 2.1 is een samenvatting van de analyse van de User-Agent strings ranglijst, opgesteld met behulp van webstatistieksoftware op de DataNose server (het origineel met korte uitleg is te vinden in de appendix, bijlage B: Marktaandeel). Bijna alle bezoekers (96,11 %) zijn

vertegenwoordigd in dit overzicht. Er is te zien dat bijna 35 % van alle bezoekers de website met hun handheld apparaten bekijken en dat iOS en Android onder de handheld apparaten de overgrote meerderheid vormen (samen 97,54 %). Kiezen voor één platform zal als resultaat hebben dat maar de ongeveer helft van de doelgroep gebruik kan maken van de applicatie, in het "beste" geval 56,24 % (iOS). De platforms iOS en Android zullen dus zeker ondersteund moeten worden. Ondersteuning van andere platforms stelt slechts een zeer klein deel van de doelgroep tevreden, dus dit heeft een veel lagere prioriteit.

2.2 Functionele eisen

Met de beantwoording van de in de vorige sectie gestelde vragen is er een redelijk goed beeld geschept van de mogelijkheden en is het mogelijk een set eisen op te stellen waaraan de mobiele applicatie dient te voldoen. Vier algemene eisen kunnen gesteld worden aan de applicatie.

Ten eerste dient de applicatie het rooster van de gebruikende student te kunnen tonen. Het rooster kan:

- opgehaald worden zodra de gebruiker een correct collegekaartnummer invoert;
- als een lijst activiteiten worden weergegeven, chronologisch;
- per activiteit in detail worden weergegeven:
  - hierbij wordt het vak, de docent en het lokaal bij elke activiteit getoond;
- gecached worden, zodat het ook werkt zonder internetverbinding.

Daarnaast hoort de applicatie de gebruiker te laten zoeken door de informatie van de webservice.

- Er kan gezocht worden op basis van:
  - vak;
  - docent;
  - lokaal.
- Zoeken gebeurt op basis van relevante attributen van de gezochte items (later te bepalen).
- De gevonden items kunnen in een lijst worden weergegeven.
- Individuele items kunnen in detail worden weergegeven.
- De roosters van de individuele items kunnen op dezelfde manier als het persoonlijke rooster weergegeven worden.
- Items en de roosters ervan kunnen worden gecached zodat herhaalde zoekopdrachten sneller resultaat opleveren.

Ten derde dient de applicatie gebruiksvriendelijk te zijn en snel te werken. Deze eis volgt niet uit de uitwerking van de probleemstelling, maar geldt in principe voor elke goed functionerende mobiele applicatie. De applicatie:

- dient zo snel mogelijk de gebruiker zijn/haar persoonlijke rooster te kunnen tonen;
- dient de gebruiker gemakkelijk te laten navigeren tussen verschillende functies van de applicatie.

- dient de lijsten en andere weergaven in een zo overzichtelijk mogelijke manier aan de gebruiker te weergeven.

Tenslote dient de applicatie ontwikkeld te worden voor de platforms iOS en Android. Andere platforms worden niet ondersteund.

Deze gestelde eisen zullen een concrete invulling krijgen in hoofdstuk 4: Ontwerp.

HOOFDSTUK 3

Gerelateerd Onderzoek

Dit hoofdstuk geeft enkele voorbeelden van onderzoeken die een raakvlak hebben met dit project.

Een artikel uit IEEE Software, getiteld "Development Platforms for Mobile Applications: Status and Trends" [10], beschrijft het onderzoek naar mobiele ontwikkelingsplatforms vanuit het opzicht van 4 runtime environments, namelijk die van Java ME, .NET CF, Flash Lite en Android. De bruikbaarheid van de vier runtimes wordt aan de hand van functionaliteit, ontwikkelsnelheid, (executie- en geheugen-) performance, en portability (ook vanuit de desktopvariant) beoordeeld. De conclusie van het onderzoek brengt geen duidelijke winnaar naar voren, en elke runtime is wel ergens goed voor te gebruiken. Zaken zoals runtime support over meerdere soorten devices, inclusief de desktopvariant (portability), volwassen documentatie en grootte van de developer community base, beginnersgemak met de taal (ontwikkelsnelheid), ingebouwde support voor compressieformaten (performance), geschiktheid voor games (functionaliteit) komen langs de revue.

In de bachelorscriptie "Smartphone Cross-Platform Frameworks" [15] wordt een diepgaande analyse van een aantal cross-platform oplossingen uitgevoerd. De algemene karakteristieken van de verschillende mobiele platforms worden geanalyseerd, waarna verschillende cross-platform oplossingen grondig met elkaar worden vergeleken en worden getoetst op bepaalde vooraf gestelde eisen. Het doel is het bepalen van het optimale ontwikkelingsplatform voor de client, een mobiele applicatie- en webdevelopmentbedrijf. Het onderzoek leidt de auteur uiteindelijk naar het platform Titanium van Appcelerator.

Als laatste is er de bachelorscriptie van R. Gopal [11]. Deze scriptie behelst de ontwikkeling van een roosterapplicatie voor de UvA voor Android. Er is ook een informele enquête afgenomen naar de interesse van een nieuwe mobiele applicatie überhaupt en enkele specifieke functies van zo'n applicatie in het bijzonder. Het resultaat toont interesse onder de studenten naar een mobiele applicatie en roosterfunctionaliteit.

Het huidige project gaat minder diep in op alle platforms dan de eerste twee werken, maar richt zich voor een groot deel op het gebruik van één ervan en de ontwikkeling van een mobiele applicatie. Dit project overlapt vooral met de twee bachelorscripties en minder met het artikel, aangezien dit zich vooral richt op soorten runtimes, daarnaast worden iOS en de in dit project behandelde XMT's (Cross-platform Mobile Development Tools) volledig buiten beschouwing gelaten.

Onderzoek naar de kwaliteit van verschillende XMT's is dus al eerder gedaan. Dit project richt zich er vooral op snel een afweging te maken met de huidige praktische situatie in gedachten (de vraag naar een mobiele roosterapplicatie voor de studenten van de FNWI) en daarmee zo snel mogelijk met de ontwikkeling van de mobiele applicatie te kunnen beginnen.

HOOFDSTUK 4

Ontwerp

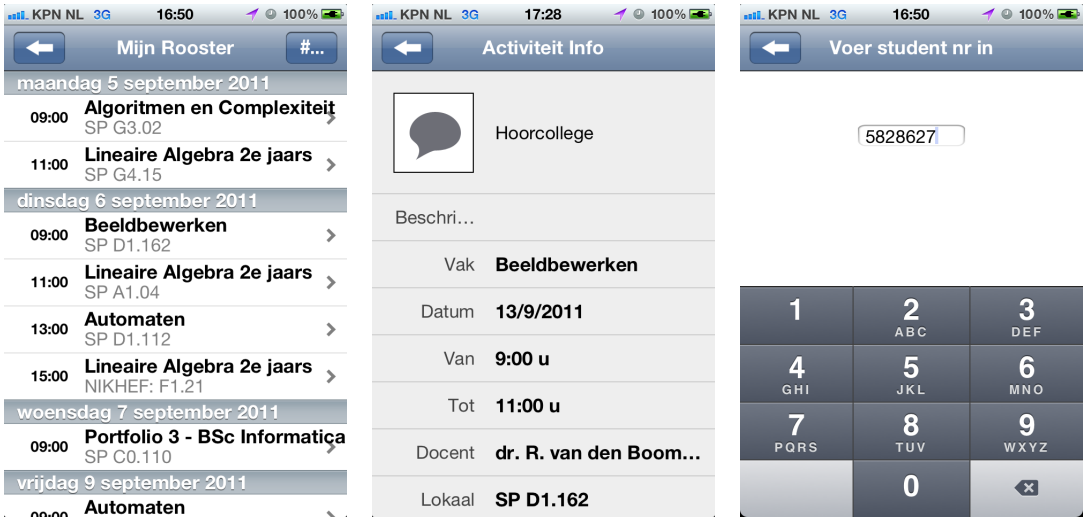
In dit hoofdstuk worden het functionele en technische ontwerp van de applicatie beschreven.

4.1 Functioneel Ontwerp

Het ontwerp baseert zich op de functionele eisen die gesteld zijn in sectie 2.2.

4.1.1 Persoonlijk rooster

De primaire functionaliteit van de applicatie, het weergeven van het persoonlijke rooster, wordt vervuld met behulp van drie schermen.



(a) Het rooster (b) Activiteitinfo (c) Collegekaartnummer invoeren

Figuur 4.1: Persoonlijk rooster

Een roosterweergave (zie figuur 4.1(a)) wordt gebruikt om een overzicht te weergeven van de activiteiten. Een roosterweergave is een lijstweergave waarbij de items in de lijst de tijd laten zien waarop de activiteit plaatsvindt (links), het vak waarvan het deel uitmaakt (titel) en de

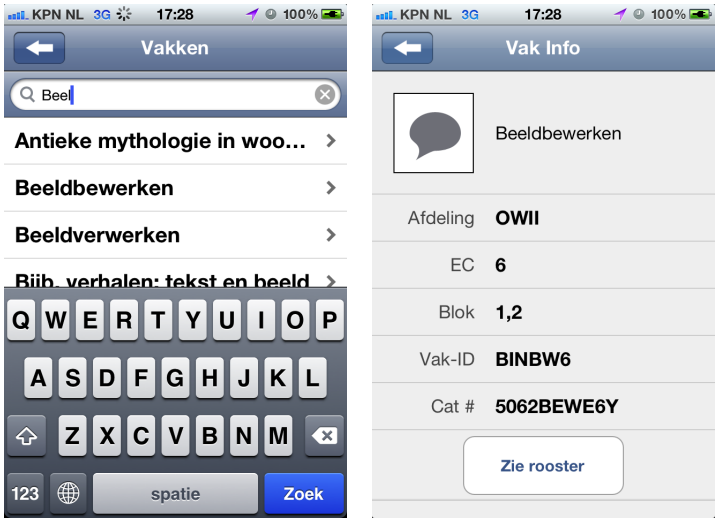
locatie waarop de activiteit plaatsvindt (subtitel). Activiteiten zijn chronologisch weergegeven en om het overzicht te bevorderen per dag gegroepeerd. Tikken op een activiteit opent de detailweergave (4.1(b)).

Bovenaan elk scherm bevindt zich de titelbalk met links een navigatieknop en, in het geval van de lijstweergave, rechts een knop die leidt naar het collegekaartnummer invoerscherm. De functie van de navigatieknop (met de pijl naar links erop) wordt uitgelegd in sectie 4.1.3: Indeling en navigatie.

Figuur 4.1(c) geeft het collegekaartnummer invoerscherm weer waarmee de student het collegekaartnummer kan invoeren. Wanneer de applicatie voor het eerst start zal dit tevens het scherm zijn dat het eerst getoond wordt, zodat de gebruiker direct het nummer kan invoeren om zo zijn persoonlijke rooster te kunnen zien. Zodra de gebruiker een nummer invoert zal de applicatie het rooster automatisch gaan proberen te downloaden, daarna bij elke start van de applicatie. Als het nummer ongeldig is, gebeurt er niets en zal de applicatie bij het openen van het roosterscherm een fout rapporteren aan de gebruiker. Klopt het nummer, dan zal de applicatie het roosterscherm openen en zal de gebruiker in het roosterscherm een lijst activiteiten aantreffen. Volgende starts van de applicatie zullen naar het roosterscherm leiden in plaats van het collegekaartnummer invoerscherm. Als de gebruiker het collegekaartnummerveld leeglaat, of leegmaakt (als de gebruiker zich bedenkt), gebeurt er niets en zal het roosterscherm leeg aangetroffen worden en bij volgende starts van de applicatie zal het "Vakken"-zoekscherm automatisch worden geopend. Voorlopig volstaat het om te zeggen dat in iOS de navigatieknop vanuit het collegekaart invoerscherm naar het roosterscherm zal leiden en bij Androidtelefoons zal hiervoor de hardware "terug"-knop gebruikt dienen te worden. Gecachede activiteiten zullen altijd direct weergegeven worden, en zullen vervangen worden met activiteiten van het internet zodra deze binnenkomen.

4.1.2 Zoeken naar roosters

Het zoeken naar roosters gaat op basis van de categorie vakken, docenten of lokalen. Elke zoekcategorie heeft zijn eigen zoekscherm dat in het geval van de categorie vakken eruit ziet zoals weergegeven in figuur 4.2(a).



(a) Lijstweergave (b) Detailweergave

Figuur 4.2: Zoeken naar rooster

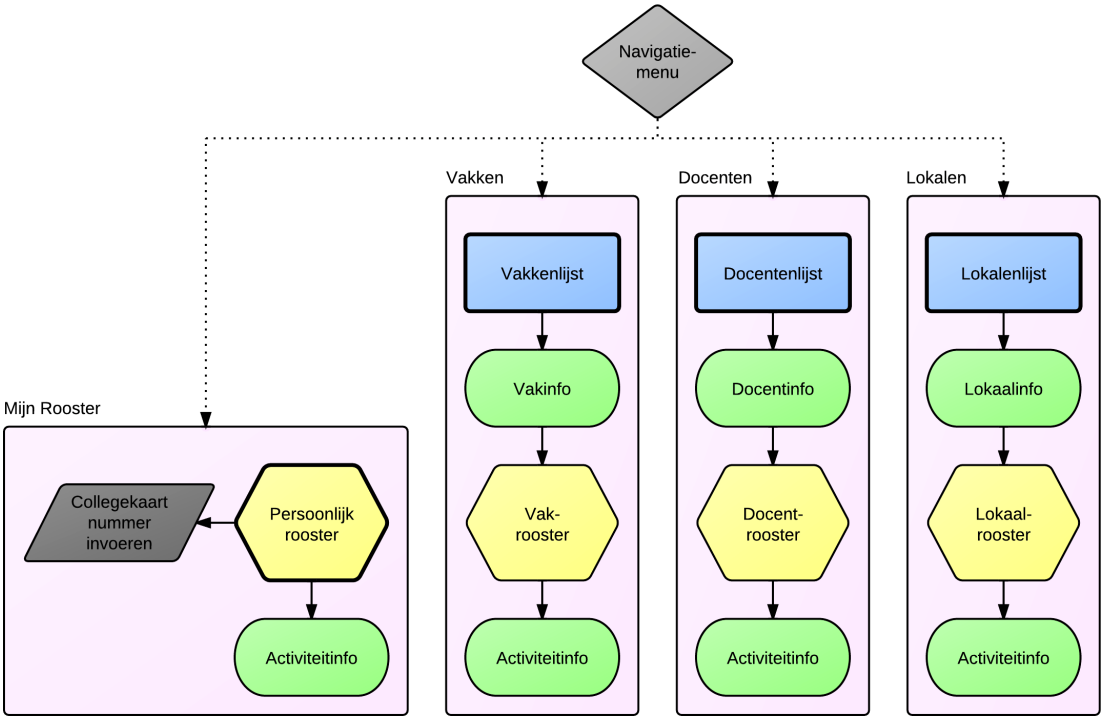
Bovenaan het scherm staat een titelbalk met daaronder een zoekbalk. Wanneer erin getypt wordt, worden de overeenkomende items (bijvoorbeeld vakken) in de lijst eronder weergegeven. Tikken

op items in de lijst opent de detailweergave van dat item (figuur 4.2(b)). De detailweergave toont alle attributen van het item en bevat een knop waarmee het rooster van dat item weergegeven kan worden (zie figuren 4.1(a) en 4.1(b) voor voorbeelden van roosterweergave). Gecachede items zullen eerst getoond worden, daarna zullen deze vervangen worden door items van het web, zodra deze binnenkomen.

4.1.3 Indeling en navigatie

Om elke functie een duidelijke plaats te geven is de applicatie ingedeeld in 4 secties: “Mijn Rooster”, “Vakken”, “Docenten” en “Lokalen”. De sectie “Mijn Rooster” geeft toegang tot de functionaliteit van het persoonlijke rooster (zie sectie 4.1.1: Persoonlijk rooster) en de andere drie secties geven toegang tot het zoeken naar respectievelijk de vakken, docenten en lokalen en de roosters daarvan (zie sectie 4.1.2: Zoeken naar roosters).

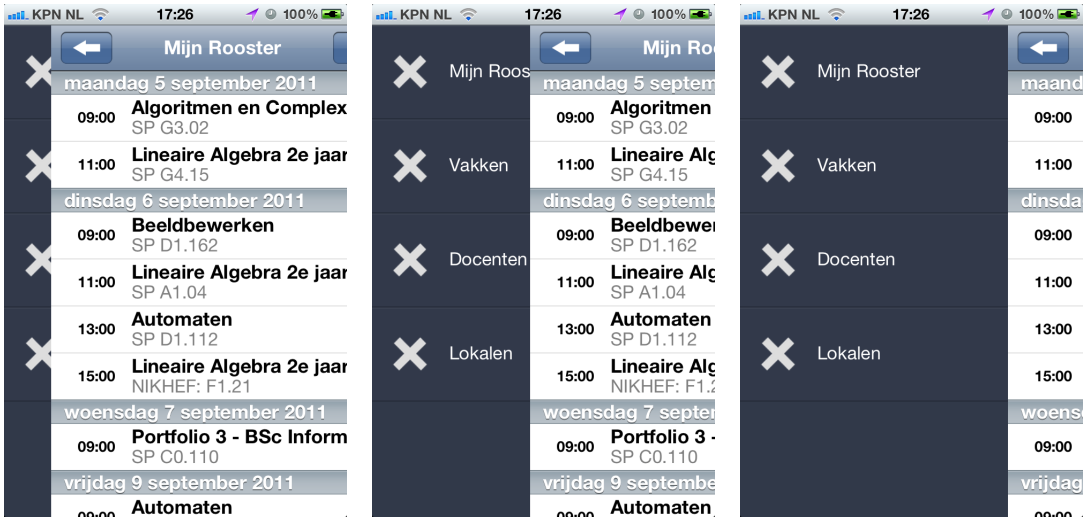
In figuur 4.3 staat de navigatiestructuur geïllustreerd. De secties van de applicatie zijn aangegeven met de 4 grote kaders. De stippellijnpijlen lopen van het navigatiemenu naar de secties die in het navigatiemenu vertegenwoordigd zijn. De elementen in de secties met een dikke rand stellen de startschermen van die secties voor. Doorgetrokken pijlen tussen elementen in de secties lopen in de richting van de voorwaartse navigatie tussen de schermen die door deze elementen vertegenwoordigd worden.



Figuur 4.3: Schematisch overzicht van de navigatiestructuur

Om te kunnen navigeren tussen de verschillende secties bevat de applicatie een navigatiemenu. Het gedrag en uiterlijk van het navigatiemenu is gemodelleerd naar dat van de Facebook applicatie voor iOS en Android [3, 2, 4], vanwege het gebruiksgemak van dit type menu, en de mogelijkheid het menu vanuit elk scherm direct op te roepen. Het is een menu dat vanuit de linkerkant van het scherm verschijnt, doordat het dan zichtbare scherm bij oproepen van het menu naar rechts schuift (gedeeltelijk het beeld uit). Zie figuur 4.4 voor een illustratie van de werking van het menu. Het oproepen van het menu doet men door de navigatieknop aan te raken (te vinden linksbovenaan elk scherm in de navigatiebalk) of de navigatieknop met een vinger op

het touchscreen naar rechts te slepen. Het tot op dat moment zichtbare scherm verdwijnt bijna uit beeld en is weer terug te krijgen door het nog zichtbare gedeelte aan te raken of terug naar links te slepen.



(a) Begin van de animatie (b) Midden van de animatie (c) Eind van de animatie

Figuur 4.4: Werking van het navigatiemenu

De navigatie heen is op Android en iOS identiek, er dient op items of (rooster)knoppen getikt te worden. De navigatie terug werkt verschillend op beide platfoms. Androidapparaten hebben een hardware “terug”-knop en iOS apparaten niet. Onder iOS doet de navigatieknop dubbel dienst als een “terug”-knop, en is het navigatiemenu alleen op te roepen door de knop naar rechts te slepen.

4.2 Technisch ontwerp

In deze sectie wordt het technisch ontwerp beschreven. Het eerste punt van aandacht is de keuze van ontwikkeltools. De architectuur wordt vervolgens in grote lijnen uitgezet en de taken en verantwoordelijkheden van de interne onderdelen worden beschreven. De implementatiedetails per onderdeel worden uitgebreid besproken in hoofdstuk 5: Implementatie.

4.2.1 Ontwikkeltools

De eerste implementatiekeuze is de keuze van development tools. Een inventarisatie van enkele beschikbare ontwikkeltools zal een licht schijnen op de mogelijkheden.

Algemeen

Er zijn in principe drie soorten applicaties te schrijven voor een platform zoals iOS en Android:

- native applicatie;
- webapplicatie;
- hybride applicatie.

Native applicaties worden ontwikkeld met de SDK behorende bij het doelplatform zelf en dienen geïnstalleerd te worden op de apparaten om gebruikt te kunnen worden. In principe is het de "standaard"-manier om applicaties te ontwikkelen. Met native applicaties kunnen de beste systeemprestaties behaald worden en alle device functies (zoals bijvoorbeeld GPS, de camera, multitouch) kunnen ten volste benut worden. Er moet wel voor elk platform apart een applicatie ontwikkeld te worden, wat de ontwikkeltijd enorm vergroot.

Webapplicaties kunnen aangepast zijn voor mobiel gebruik, zodat ze enigszins op native applicaties gaan lijken. Ze dienen echter altijd via de browser gebruikt te worden, en men dient altijd online te zijn om ze te gebruiken. De "native" user-interface is nooit zo snel als bij native applicaties en device functies (al worden sommige zaken zoals locatiediensten tegenwoordig via HTML5 APIs via de browser blootgelegd) zijn in principe ook niet beschikbaar tot de applicatie. Ontwikkeling gaat wel sneller dan native, aangezien het ondersteunen van meerdere browsers voor een groot deel mogelijk is door webstandaarden aan te houden.

Hybride applicaties zijn ontwikkeld met behulp van XMT's (Cross-platform Mobile Development Platoforms). Deze applicaties zijn native in de zin dat ze op het device geïnstalleerd worden en (voor het platform) native code bevatten, maar het algemene programma is niet geschreven met alleen de SDKs horende bij het doelplatform, maar met een intermediate SDK die een vertalingsbrug plaatst tussen de originele SDK's en de ontwikkelaar. Zodoende kan men in één keer meerdere native-achtige applicaties maken, voor elk platform een, vaak (maar niet altijd) met behulp van webtechnologie. Een aantal devicefuncties zijn vaak op een universele manier aan te spreken en in een enkel geval is er ook gebruik te maken van de native user-interface van de apparaten. De ondersteuning van verschillende platformen varieert tussen de XMT's.

De native en hybride oplossingen worden in acht genomen bij dit project, aangezien de aard van een webapplicatie voorschrijft dat de gebruiker altijd online dient te zijn, en dit is incompatibel met functionele eis dat de applicatie ook dient te werken zonder internetverbinding.

Native

Aangezien iOS en Android de enige ondersteunde platforms zijn zullen we, wat native development betreft, alleen de SDK's voor deze platforms belicht worden.

Android SDK

Hiermee kunnen er native applicaties ontwikkeld worden voor het Android platform [16]. Applicaties voor Android worden in Java geprogrammeerd, en de officieel ondersteunde IDE is Eclipse in combinatie met de "Android Development Tools" plugin. Applicaties worden naar Dalvik executable bytecode gecompileerd [1]; libraries gecompileerd naar machinecode kunnen geïntegreerd worden met behulp van de Native Development Kit (ten behoeve van codehergebruik en/of performance) [17]. De Android SDK kan men gebruiken op Windows, Linux en Mac.

iOS SDK

De iOS SDK stelt men in staat native applicaties te ontwikkelen voor het iOS platform. Applicaties maken gebruik van het Cocoa Touch framework, dat is geïmplementeerd in C, Objective-C en C++. Dit zijn dan ook de talen waarin geprogrammeerd wordt. Applicaties worden naar machinecode compileerd en verder wordt met de SDK de IDE "Xcode" meegeleverd. Apple ondersteunt slechts Mac OS X als ontwikkelingsplatform om applicaties mee te ontwikkelen voor iOS [8].

Cross Platform

Een paar populaire XMT's zullen hier de revue passeren. PhoneGap, Titanium en RhoMobile zijn op het Internet de meest aangedragen XMT's.

PhoneGap
Hiermee kunnen hybride applicaties gemaakt worden. PhoneGap ondersteunt de platforms: iOS, Android, Blackberry, WebOS, Windows Phone 7, Symbian en Bada. Hybride applicaties gebruiken de web-engine van het platform waarop ze uitgevoerd worden om het programma, ontwikkeld in webtalen zoals HTML, CSS en JavaScript uit te voeren. Een subset van de native API wordt door de JavaScript interpreter blootgesteld aan de ontwikkelaar. Met PhoneGap wordt het gebruik van de originele SDK's niet ontweken, er wordt een projecttemplate klaargezet waarmee verder gewerkt kan worden in de IDE behorende bij het platform waarvoor ontwikkeld gaat worden.

Titanium SDK
Hiermee kunnen "pseudo"-native applicaties ontwikkeld worden voor de platforms iOS en Android [7]. Pseudo-native wijst hier op het feit dat er native UI componenten kunnen worden aangeroepen vanuit de non-native programmacode. Applicaties worden geschreven in JavaScript. Bij iOS wordt de JavaScriptcode geïntegreerd in de application binary, en "at runtime" geïnterpreteerd. Bij Android wordt de code gecompileerd naar Java bytecode [14]. De Titanium SDK levert Titanium Studio als IDE, een Eclipse kloon. Titanium Studio compileert applicaties met behulp van de Android of iOS SDK, deze zijn dus nog wel nodig. Voor iPhone ontwikkeling is dus ook steeds een Mac nodig.

RhoMobile
Vergelijkbaar met PhoneGap, in zoverre dat er hybride applicaties mee ontwikkeld kunnen worden. RhoMobile gebruikt het Rhodes framework, eigendom van Motorola onder de MIT licentie. Met Rhodes framework gebruikt een MVC design pattern. De applicatielogica wordt in Ruby geschreven en de presentatie van data gebeurt met behulp van HTML in webviews. De IDE die erbij geleverd wordt is RhoStudio, onderdeel van de "RhoMobile Suite". Ontwikkeling met RhoMobile wordt ondersteund op Windows, Linux en Mac OS X en iPhone applicaties zijn met dit systeem wel te ontwikkelen onder Windows [12].

Keuze

De keuze van ontwikkeltools hangt af van een aantal factoren. Ten eerste is er de mate van maintainability van de codebase die gecreëerd wordt met een ontwikkelomgeving. Ontwikkelen van meerdere native applicaties zal meerdere codebases opleveren. Voor elke aanpassing zal extra werk verricht moeten worden. Vervolgens is het marktaandeel van de platforms waarvoor ontwikkeld kan worden met de SDK van belang. Kan ermee voor Android én iPhone ontwikkeld worden? In hoeverre worden de overige platforms ondersteund? De systeemprestaties van applicaties ontwikkeld met verschillende tools zullen ook onderling verschillen, voor een data-centrische applicatie niet onbelangrijk. De door de SDK aangeboden API moet de ontwikkelaar in staat stellen de functies redelijkerwijs te kunnen implementeren. Verder zal een applicatie met native UI componenten beter aanspreken bij de gebruiker, wat de gebruiksvriendelijkheid van de applicatie ten goede komt.

In tabel 4.1 staat de afweging tussen de verschillende platforms uitgewerkt. Het komt uiteindelijk neer op een gelijkspel. RhoMobile stelt de ontwikkelaar in staat iPhone applicaties te ontwikkelen op Windows, iets dat de andere platforms niet kunnen. Het is echter niet gelukt om applicaties te testen op een device, de oorzaak hiervoor is niet achterhaald. Hierdoor is het werken met RhoMobile direct gestaakt. De twee tools die overblijven zijn PhoneGap en Titanium. PhoneGap ondersteunt de meeste platforms, maar er is al besloten om alleen voor iOS en Android te ontwikkelen, het extra marktaandeel heeft geen prioriteit meer. De native UI mogelijkheid van Titanium is een aantrekkelijke keuze, vanwege de mogelijkheid om een solide native-ogende applicatie te ontwikkelen maar toch op beide platforms tegelijk. De keuze is hierom uiteindelijk gevallen op Titanium.

Tabel 4.1: Bruikbaarheidsmatrix van de verschillende SDK's

Factor	iPhone SDK	Android SDK	PhoneGap	Titanium	RhoMobile
iOS	✓		✓	✓	✓
Android OS		✓	✓	✓	✓
Overig OS			✓		✓
Snelheid	✓	✓			
Native UI	✓	✓		✓	
API	✓	✓	✓	✓	✓
Maintainability			✓	✓	✓
Totaal	4	4	5	5	5

4.2.2 Architectuur

De architectuur van de applicatie is gebaseerd op het Model-View-Controller design pattern. Er zijn een aantal verfijningen dooregevoerd. De architectuur is geïllustreerd in figuur 4.5. Links in de figuur staan de groepen waarin de verschillende componenten verdeeld zijn. De "Start"-cirkel geeft aan waar de loop van het programma begint.

4.2.3 Componenten

Database en webservice

De webservice en database zijn de "resources", componenten die gebruikt worden als primaire bron en als opslagmedium voor de data. Ze worden aangeroepen door de resource models die precies kennis hebben van de API van deze resources.

OData adapter

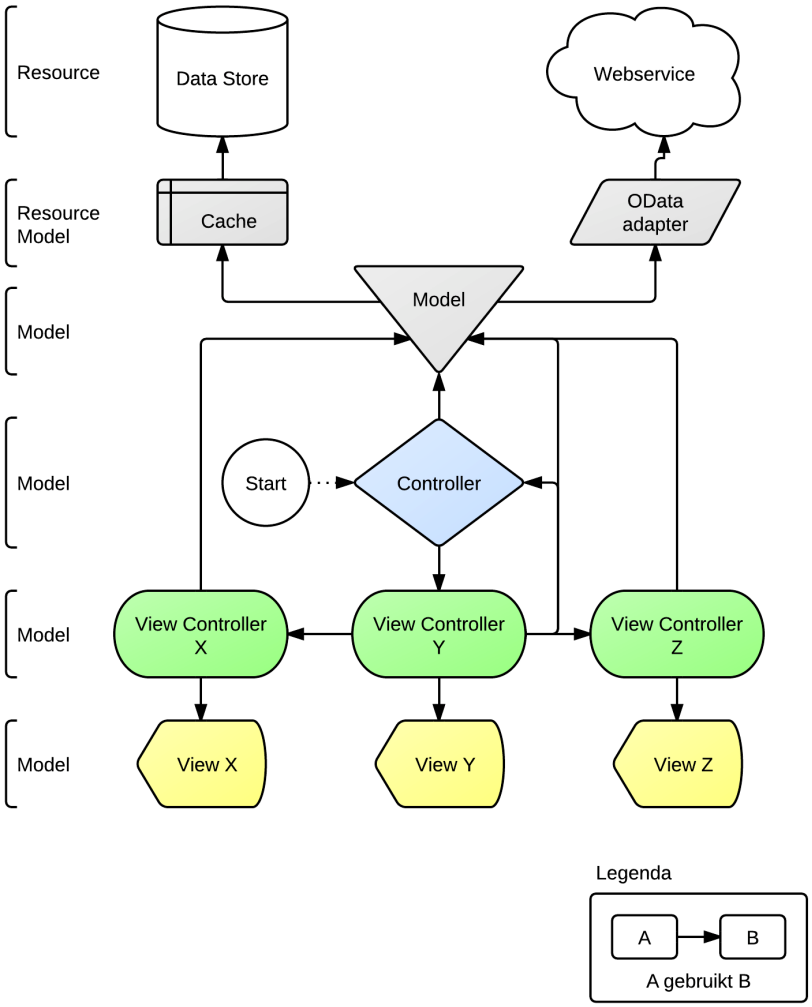
De OData adapter ("resource model") fungeert als een abstractielaag voor de rest van het programma voor de communicatie met de OData webservice van DataNose. Het is verantwoordelijk voor het uitvoeren van zoekopdrachten naar het web, en het verwerken van binnenkomende data naar een interne representatie waar de andere componenten mee om kunnen gaan.

Cache

De cache ("resource model") fungeert als een abstractielaag voor de rest van de applicatie voor het opslaan en ophalen van vakken, docenten, lokalen en activiteiten. Het model probeert de gevraagde data altijd eerst uit de cache te halen.

Model

Het model levert de data aan de view controllers die daarom vragen. Hierbij hoeven ze geen rekening houden met de daadwerkelijke bron van de informatie (of de data van de cache moet



Figuur 4.5: Schematisch overzicht van de architectuur van de applicatie

komen of van het web). Het model zorgt er ook voor dat wanneer er gegevens van het web gedownload worden, dat de cache geüpdate wordt met deze nieuwe gegevens en in samenwerking met de cache wordt ervoor gezorgd dat de data in de cache synchroon blijft lopen met de data op het web.

Controller

De intialisatie van de verschillende onderdelen, waar nodig, zal door de controller afgehandeld worden. Ook het automatisch ophalen van het persoonlijke rooster, aangezien dit proces op de achtergrond uitgevoerd dient te worden, zal gedaan worden door de controller.

View Controllers en Views

De view controllers vragen de gewenste gegevens op uit de models en gebruiken de views om deze aan de gebruiker te tonen. De views verzorgen de juiste rendering van onderdelen van de schermpjes, de layout van de schermpjes wordt bepaald door de view controllers. Er zit redelijk wat logica in de view controllers en deze is direct toegespitst op hoe de data gepresenteerd dient te worden. Bij fouten (uitgevallen internetverbinding) is het de verantwoordelijkheid van de view

controllers om een foutmelding op het scherm te tonen. Het komt voor dat een view controller de controller “gebruikt” om het nieuwe rooster op te halen, bijvoorbeeld zodra de gebruiker een collegekaartnummer invult.

HOOFDSTUK 5

Implementatie

In dit hoofdstuk wordt de implementatie van de applicatie gedetailleerd beschreven. Ten eerste wordt de Titanium SDK besproken en de implicaties voor de implementatiewijzse. Daarna wordt besproken hoe de interne representatie van de data is geïmplementeerd. Deze is nodig om de componenten gemakkelijk met elkaar te laten communiceren met en over de data. Vervolgens worden de implementaties van de individuele onderdelen van de applicatie beproken. Tenslotte wordt de implementatie van de prestatietests beschreven.

5.1 Titanium

De Titanium SDK levert een framework dat een subset van de APIs van de frameworks van de Android en iOS SDK's vertaalt naar een eigen API. Gebruik van de API gebeurt met behulp van de taal JavaScript. De API van het framework is beschikbaar gemaakt via een global variabele `Titanium` (alias `Ti`). Het framework is modulair opgebouwd en de modules zijn vertegenwoordigd als properties van het `Ti` global object (bijvoorbeeld `Ti.UI` en `Ti.Network`). Conceptuele zaken zoals lijstjes en vensters (views of windows) zijn in beide (iOS en Android) SDK's aanwezig en kunnen daarom op een universele manier aangeroepen worden (`Tititanium.UI.Window`). Andere zaken die alleen op iOS of Android bestaan zijn vertegenwoordigd in submodules (`Ti.UI.iPhone.appBadge()` of `Ti.UI.Android.hideSoftKeyboard()`). Titanium maakt gebruik van de CommonJS specificatie om modulaire opbouw van programmacode mogelijk te maken. Om cross-platform development gemakkelijker te maken is in de Titanium SDK de `require()` functie uitgebreid [6]. Stel, men roept de `require()` functie als volgt aan:

```
require("path/to/module")
```

De standaardfunctionaliteit van CommonJS schrijft voor dat het bestand op de locatie `path/to/module.js` zoekt. In de Titanium omgeving wordt er eerst echter gekeken op de locatie `<platform>/path/to/module.js`, waar op Androidapparaten de waarde van `<platform>` "android" zal zijn en in iOS zal het "iphone" zijn. Op deze manier kunnen er zonder `if`-statements te gebruiken platformafhankelijke modules gemaakt worden. Deze modules kunnen bijvoorbeeld klassen bevatten die overerven van een klasse die code bevat die voor beide platforms geschikt is.

5.2 Interne Representatie

De naamgeving van entities van de DataNose webservice zijn niet gegarandeerd compatibel met de regels van de programmeertaal waarin gewerkt wordt (in dit geval JavaScript) en hetzelfde geldt voor de namen van de attributen. Daarom is er een nieuwe interne representatie nodig voor de data die gebruikt wordt door alle componenten, dit vooral voor programmeergemak op de korte termijn en maintainability op de lange termijn. De interne representatie is geïmplementeerd met behulp van de klasse `Entity` en een `EntityDescription`-module.

De `Entity` klasse werkt op de volgende manier: een instantie ervan wordt gecreëerd met behulp van een `EntityDescription` object dat de namen van de properties van een bepaalde entity beschrijft. Het `Entity` object zal aangevuld worden met lege properties op basis van het gegeven `EntityDescription` object en accessormethoden.

5.3 OData Adapter

De implementatie van de OData adapter is in grote mate afhankelijk van de interface van de DataNose webservice. Voordat de adapter zelf besproken wordt, worden de belangrijke details van de webservice uiteengezet.

5.3.1 DataNose webservice

De DataNose webservice is geïmplementeerd als OData webservice. OData (Open Data Procotol [5]) is een protocol voor het opvragen en updaten van gegevens. Het procotol werkt bovenop HTTP en data ophalen gebeurt via HTTP-requests naar bepaalde URL's. Het formaat van de geretourneerde gegevens kan Atom of JSON zijn. De OData webservice beschrijft de aangeboden data in een *Entity Data Model*, waarin de typen entities (typenamen en attributnamen en -typen), de collecties aan entities en de speciale operaties worden beschreven in een zogeheten *entitycontainer*. Tabel 5.1 beschrijft beknopt de inhoud van de entitycontainer van de gebruikte webservice. In de appendix staat het EDM in detail beschreven, sectie A.1: Datanose EDM.

Tabel 5.1: Entity sets en functions in het EDM van de DataNose webservice

	Abstract type	Naam / Functiedefinitie	Entity (return) type
Entity sets	Vakken	Courses	TTCourse
	Docenten	Staff	TTStaff
	Lokalen	Locations	TTLocation
	Activiteiten	Activities	TActivity
Functions	Vakken	GetCoursesByStudent(*id*)	TTCourse
	Docenten	GetStaffByActivity(*id*)	TTStaff
	Lokalen	GetLocationsByActivity(*id*)	TTLocation
	Activiteiten	GetActivitiesByCourse(*id*)	TTActivity

URL's

Data wordt beschikbaar gemaakt via URL's. De basisstructuur van een URL voor een onderdeel uit de entity container is als volgt:

$$\underbrace{\text{http://content.datanose.nl}}_{A} / \underbrace{\text{Timetable.svc}}_{B} / \underbrace{\text{Courses}}_{C} \qquad \begin{cases} A & \text{root-URL} \\ B & \text{root document} \\ C & \text{resource pad} \end{cases} \tag{5.1}$$

De root-URL identificeert de server. Het root document identificeert de service (er kunnen meerdere OData webservices naast elkaar aanwezig zijn op een server); het kan elke naam hebben, maar over het algemeen bevat het de extensie "svc". Het resource pad identificeert een onderdeel uit de entity container, in dit geval een function of entity set. Een HTTP-request naar bovenstaande URL zal de hele entity set *Courses* opleveren (strict genomen slechts een deel, zie *paginering*). De OData specificatie beschrijft een aantal manieren om naar de data te queryen, onder andere met behulp van query string-parameters. De in de applicatie gebruikte manieren zullen belicht worden. Wat betreft de notatie in de voorbeelden: variabelen zijn schuingedrukt en dienen vervangen te worden met waarden.

Functions lijken erg op andere resources, met een verschil: ze verwachten parameters met bepaalde namen, argumenten voor de functie. In het geval van `GetCoursesByStudent(id)`, gaat het om één parameter, genaamd `id`, en deze dient als query string-parameter toegevoegd te worden aan de url:

$$\ldots/\text{Timetable.svc/GetCoursesByStudent?id=}studentId \tag{5.2}$$

In plaats van *studentId* dient het collegekaartnummer van de betreffende student meegegeven te worden. Een ongeldig nummer of een nummer van een student van een andere faculteit resulteert in een fout-respons.

Zoeken op basis van bepaalde attributen is ook mogelijk. Hiervoor wordt de query string-parameter `$filter` gebruikt. Deze geeft de mogelijkheid logische predicaten (en bijvoorbeeld stringfuncties) te gebruiken om te bepalen aan welke condities de resultaten moeten voldoen. Voorbeeld:

$$\ldots/\text{Timetable.svc/Courses?\$filter=substringof('architectuur', tolower(Name))} \tag{5.3}$$

Verder ondersteunt het OData protocol nog een aantal querymogelijkheden, onder andere het ophalen van items via de koppelingen ertussen (navigation properties), het beperken van de attributensets van entities die terugkomen met een response (vergeljkbaar met SQL `SELECT`) het sorteren van de items in de respons (SQL `ORDER BY`)

5.3.2 Adapter

De adapter verbindt met de webservice om de data aan het model door te geven. Voor de communicatie met de webservice wordt er de datajs library gebruikt, geleverd door OData.org [9]. De binnenkomende data wordt door deze library omgezet in een objectstructuur die gemakkelijk te behandelen is. De adapter loopt door alle binnengekomen items heen en kopieert alle attributen die genoemd worden in de Entity Description module behorend bij dat type entity naar een object en stopt al deze objecten in een array. De array wordt teruggegeven aan het Model met behulp van callback methodes die meegegeven moeten worden aan aanroepen van de OData adapter.

5.4 Model

Het model maakt gebruik van de OData adapter en de Cache om de data van het Internet te halen en op te slaan voor later gebruik. Het bestaat uit 4 klassen: `Course`, `Staff`, `Classroom` en `Activity`. De naamgeving van `Classroom` rust op het feit dat `location` een gereserveerde term is in JavaScript. Om problemen te voorkomen heeft dit model een andere naam gekregen (voor consistentie de naam van de bijbehorende cache ook). Veel hebben ze met elkaar gemeen, ze zijn dan ook geïmplementeerd als klassen die overerven van een basisklasse `AbstractModel`.

5.4.1 Communicatie

Het model communiceert de data aanvraag door naar de OData adapter en de Cache. Als een view controller data van een model opvraagt, gebeuren er de volgende dingen:

1. View Controller (VC) roept Model aan;
2. Model roept Cache aan;
3. Gecached data komt terug bij Model (cache hit/miss);
4. In geval van een cache hit, stuurt het Model de gecachete data terug naar VC;
5. Model roept OData adapter aan;
6. Webdata komt binnen bij model;
7. Model update cache met nieuwe webdata;
8. Model stuurt webdata terug naar VC.

In figuur 5.1 is een voorbeeld weergegeven dat de werking van het model illustreert.

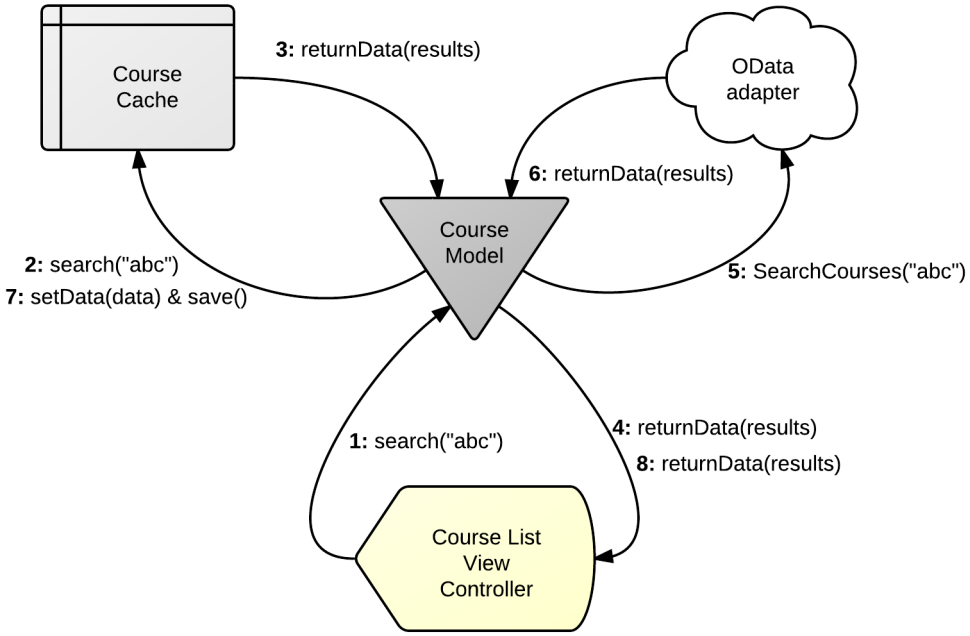
De figuur toont de afhandeling van een zoekopdracht aan het Course Model (Vakken). De 8 bovenstaande stappen komen overeen met de cijferlabels in de figuur. Bijzonderheden zijn: `returnData()` (3) wordt voor elke 100 cache resultaten een keer uitgevoerd, en daarmee `returnData()` (4) ook. `returnData()` (6) wordt voor elke 100 webresultaten een keer uitgevoerd en daarmee (8) ook. De aanroepen `save()` en `setData()` (7) worden een keer uitgevoerd voor elk resultaat dat terugkomt (vaak meer dan 1) met een aanroep van `returnData()` (6). Een mogelijk patroon van aanroepen zou kunnen zijn (* betekent "vaker uitgevoerd"):

$$1, 2, (3, 4)^*, 5, (6, 7^*, 8)^* \tag{5.4}$$

De functieaanroepen kloppen niet exact met de API, dit om de illustratie overzichtelijk te houden. Aan de `search...()` functies moet bijvoorbeeld ook enkele callbackfuncties meegegeven worden.

5.4.2 Activity Model

De Course, Staff en Classroom models houden zich alleen bezig met hun eigen soort data. Het Activity model is hierin anders. Aangezien een `Activity`, zoals deze vanaf DataNose binnenkomt, slechts een deel van de essentiële informatie bij zich draagt voor een activiteit, de informatie over lokatie, het vak en de docent zitten in gekoppelde items, worden activiteiten tegelijk deze items opgehaald. Eerst wordt het lokaal en de docent opgeslagen in de cache, waarna de activiteit wordt opgeslagen. Het Activity model geeft dan door aan de cache dat er een koppeling gemaakt moet worden tussen deze items. Verder beelden de Activity entities niet 1:1 af op werkelijke activiteiten (zie onderdel **??** in subsectie 5.3.1). Het formaat moet omgezet worden naar een voor een view beter handelbaar formaat omgezet worden. Deze omzetting naar activiteiten met een enkel datum veld wordt elke keer als er data van het web of van de cache gehaald wordt uitgevoerd.



Figuur 5.1: Afhandeling van een zoekopdracht

5.5 Cache

De Cache handelt de opslag van gegevens af. De *resource* die de cache hiervoor gebruikt is een SQLite database. De Titanium SDK bevat een SQLite database engine, dus hiervoor is geen externe library nodig. De cache voert de nodige SQL-queries en updates uit om de data op te halen en op te slaan. Het datamodel van de cache wordt beschreven in de appendix, sectie A.2: Cache Data Model.

De cache bestaat, net als het model, uit 4 onderdelen ("cache models"): CourseCache, StaffCache, ClassroomCache en ActivityCache. Elk cache model is verantwoordelijk voor zijn eigen tabel in de database. Er wordt in de database gebruik gemaakt van integrity constraints, om zo de koppelingen tussen de verschillende tabellen consistent te houden. Als een record uit een tabel verwijderd wordt, worden de gekoppelde items automatische ook verwijderd. Verder worden er indexes op de tabellen gebruikt om zoekopdrachten sneller te laten verlopen.

5.5.1 Synchronisatie

De cache models moeten gesynchroniseerd worden met de online collectie en daarbij moet op 2 zaken gelet worden: de waarden van de attributen van de entities in de cache moeten overeenkomen met die van de entities van de DataNose webservice en entities die uit de collectie van DataNose verwijderd zijn moeten ook worden verwijderd uit de cache. Bij het verwerken van zoekresultaten kunnen alle items in de cache die voldoen aan de zoekcriteria gemarkeerd worden voor verwijdering. Items die terugkomen van de zoekopdracht naar de webservice worden gebruikt om de cache te updaten, waarbij de betreffende items in de cache hun markering kwijtraken, items in de cache die hun markering aan het eind nog hebben zijn verwijderd uit de online collectie en kunnen dan worden verwijderd uit de cache. Items uit de datanose webservice hebben een permanent ID waarmee de koppeling tussen de cache items en de online collectie gewaarborgd kan worden.

5.6 Controller

De Controller is het startpunt van de applicatie. De primaire taak die de controller heeft, is het initialiseren van de Main View Controller. Daarnaast start het de applicatie in testmodus, waarbij er prestatietests uitgevoerd worden en de frontend van de applicatie niet ingeladen wordt.

5.7 View Controllers

De implementatie van de view controllers is redelijk triviaal. Over het algemeen laden de view controllers allemaal (behalve die voor het collegekaartinvoerscherm) een TableView in en roepen het model aan om de TableView in te laden met data. De view controllers voor de zoekvensters gebruiken uiteraard ook een SearchBar.

5.8 Tests

Om te testen hoe de applicatie presteert in het verwerken van data, is er een component ontwikkeld dat dit proces beheert: de Profiler. Daarnaast zijn er ook aanpassingen gemaakt in de models op de plekken waar ze getest worden. Sommige data opvraagdmethoden van de models accepteren een callback functie als extra argument, deze functie wordt op bepaalde plekken in de code uitgevoerd, waardoor de Profiler kan meten hoe lang bepaalde taken duren.

De Profiler meet de prestaties van de volgende taken:

- Ophalen van rooster uit de cache;
- Vernieuwen van het rooster via het web en opslaan in de cache;
- Ophalen van alle eenvoudige collecties (vakken, docenten en lokalen) uit de cache, het web, en het weer opslaan ervan in de cache;

Voor de drie soorten taken meet de Profiler onderstaande onderdelen:

- Rooster uit cache:
  - Cache items gekoppeld aan ingeschreven vakken ophalen (`cache_timetable_results`);
  - DB-resultaten omzetten naar IR (`cache_timetable_process`);
- Rooster via web:
  - HTTP-request voor ingeschreven vakken (`odata_read`);
  - HTTP-request voor activiteiten van opgehaalde vakken (`odata_read`);
  - HTTP-response omzetten naar IR (`adapter_process`);
  - Activiteiten uitvouwen naar 1:1 formaat (`activity_expand`);
  - Alle activiteiten opslaan in cache en koppelen (`major_update`)
- Eenvoudige collecties:
  - Aantal cache items tellen met SQL-query (`cache_count`);
  - Cache items ophalen met SQL-query (`cache_result`);
  - DB-resultaten omzetten naar IR (`cache_process`);
  - HTTP-request uitvoeren (`odata_read`);

- HTTP-response omzetten naar IR (`adapter_process`);
- SQL updates/inserts (`cache_update`).

De benamingen tussen haakjes zijn de namen die intern worden gegeven aan de tijdmetingen van die onderdelen. In hoofdstuk 6: Resultaten zullen deze namen ook gebruikt worden.

In totaal test de profiler de componenten 15 keer achter elkaar. Alle gemeten systemen zijn geprogrammeerd slechts 500 resultaten te verwerken en dan te stoppen, aangezien niet alle collecties even groot zijn, maar wel groter dan 500. Na elke test stuurt het de resultaten met behulp van een HTTP GET request met querystring parameters voor alle resultaten naar een server op een hardcoded adres. Deze server is in dit geval een VPS met PHP erop geïnstalleerd die MySQL gebruikt om de resultaten op te slaan. De kwantitatieve resultaten in dit documenet zijn gemiddelden van alle tests berekend met behulp van SQL-queries van de volgende soort:

```
SELECT type, stddev(duration / size * 500), avg(duration / size * 500)
FROM results
```

HOOFDSTUK 6

Resultaten

In dit hoofdstuk worden de resultaten van het project besproken. De ontwikkelde mobiele applicatie wordt getoetst aan de in de probleemstelling gestelde eisen (zie hoofdstuk 2). Vervolgens worden de dataverwerkingsprestaties van de applicatie belicht. Tenslotte worden de ervaringen met de Titanium SDK kort besproken in het licht van de bruikbaarheid in de huidige praktische situatie.

6.1 Functionaliteit

De functionaliteitsvereisten voor de applicatie zijn:

- Rooster bekijken op basis van vak;
- Persoonlijk rooster bekijken per student;
- Functioneel op iOS en Android.

6.1.1 iOS

De ontwikkelde iOS-applicatie voldoet in principe aan twee minimale eisen die gesteld zijn aan de applicatie. De gebruiker is met de applicatie in staat zijn/haar persoonlijke rooster te bekijken na het invullen van het collegekaartnummer. Daarnaast kan met de de zoekopties: “Vakken”, “Docenten” en “Lokalen” de gebruiker op basis van deze 3 uitgangspunten een rooster opgevraagd worden. Wanneer de gebruiker zijn/haar collegekaartnummer invult, duurt het 3 seconden voordat downloaden van het rooster begint, en pas als het rooster in zijn geheel is gedownload ($\approx$ 20 seconden), wordt het weergegeven.

Bug

De applicatie haalt ook nog niet automatisch bij een nieuwe start het rooster opnieuw op. De cache wordt wel ingeladen zodra de roosterweergave geopend wordt, maar er dient naar het collegekaartnummer invoerscherm heen en weer gegaan te worden om een nieuwe versie van het rooster ingeladen te laten worden.

Andere afwijkingen

De bevestigingsknop op het scherm “Collegekaartnummer” invullen is afwezig, dit is niet volgens de specificatie.

6.1.2 Android

De applicatie voor Android voldoet niet aan de functionele eisen. De applicatie is in staat een menu weer te geven met 4 opties: “Mijn Rooster”, “Vakken”, “Docenten” en “Lokalen”. De 4 opties leiden in slechts drie gevallen (alle behalve “Mijn Rooster”) naar een functioneel volgend scherm, alwaar gezocht kan worden naar vakken, docenten of lokalen. Tikt men op een item, dan stopt de applicatie met functioneren (crash). De backend (models en dergelijke) functioneren wel op Android in de simulator. Profiler resultaten zijn niet beschikbaar vanwege een probleem met het installeren op het Android device. Het lukt in de laatste week van ontwikkeling niet om de applicatie op de Androidtelefoon te installeren, wat testen op het apparaat onmogelijk maakt. Profilertesten zijn alleen nuttig op het apparaat, en zijn dus niet uitgevoerd voor het Androidplatform. Het is niet geheel duidelijk of het probleem bij de Titaniumsoftware ligt of bij een verkeerde configuratie van de Android SDK’s.

6.1.3 Gebruikersvriendelijkheid

De iOS applicatie is momenteel ook nog verre van gebruikersvriendelijk. Tijdens het ophalen van het rooster krijgt de gebruiker wel een draaiend wieltje te zien waarmee aangegeven wordt dat de gebruiker even moet wachten. Het wachten duurt alleen wel 20 seconden en dat is voor smartphonegebruikers veel.

6.2 Prestaties

De responsiviteit van de applicatie van hangt voor een groot deel af van de snelheid waarmee de binnenkomende data verwerkt wordt. De prestaties van de verschillende delen van de ontwikkelde software zijn getest door middel van *method profiling*. De uitvoering van een aantal belangrijke subroutines in de software zijn in de tijd gemeten om zo een beeld te krijgen van de bottlenecks en de algemene effciëntie van de dataverwerking. Er is gemeten hoe lang bepaalde delen van het programma erover doen om een bepaalde hoeveelheid data te verwerken. Om deze resultaten te verkrijgen is de ontwikkelde applicatie aangevuld met subroutines die de metingen uitvoeren (zie sectie 5.8: Tests).

In tabel 6.1 staan de resultaten voor het ophalen van lijsten items en het ophalen van het rooster van de cache. De resultaten zijn genormaliseerd naar 500 items voor alle verwerkingstijden. De eerste drie kolommen bevatten metingen die ook daadwerkelijk verricht zijn op operaties op 500 items. Er is te zien dat het ophalen van vakken langer duurt per stuk dan docenten en lokalen. De grootste tijd in het updaten van de cache, daarna is het ophalen via HTTP het meest langzaam.

Het ophalen van de items van het web gaat gemiddeld in 22,5 seconden met een standaarddeviatie van ongeveer 1,6 seconden (zie tabel 6.2).

6.2.1 Interpretatie

Er is te zien dat tegen verwachtingen in het updaten van de cache langer duurt dan het opvragen van gegevens bij de webservice. Hier is nog veel verbetering qua efficiëntie te behalen. De

Tabel 6.1: Prestaties van de operaties in verwerkingstijd (ms) gemiddeld over 500 items

Meting	Ophalen vakken		Ophalen docenten		Ophalen lokalen		Ophalen rooster (cache)	
	Vak		Doc		Lok		Act	
	μ	σ	μ	σ	μ	σ	μ	σ
`cache_result`	39	2.21	37	2.96	37	1.15	26	4.44
`cache_process`	912	10.99	727	13.48	666	14.18	1209	22.96
`odata_read`	6789	3347.08	4106	2250.68	4668	4668		
`adapter_process`	147	7.67	134	14.30	125	7.83		
`cache_update`	19638	480.31	18064	501.46	18345	216.04		
`major_update`								
`activity_expand`								
`model_loadAll`	27540		23083	2359.56	23855	2542.88	1608	43.93

Tabel 6.2: Ophalen rooster van het web

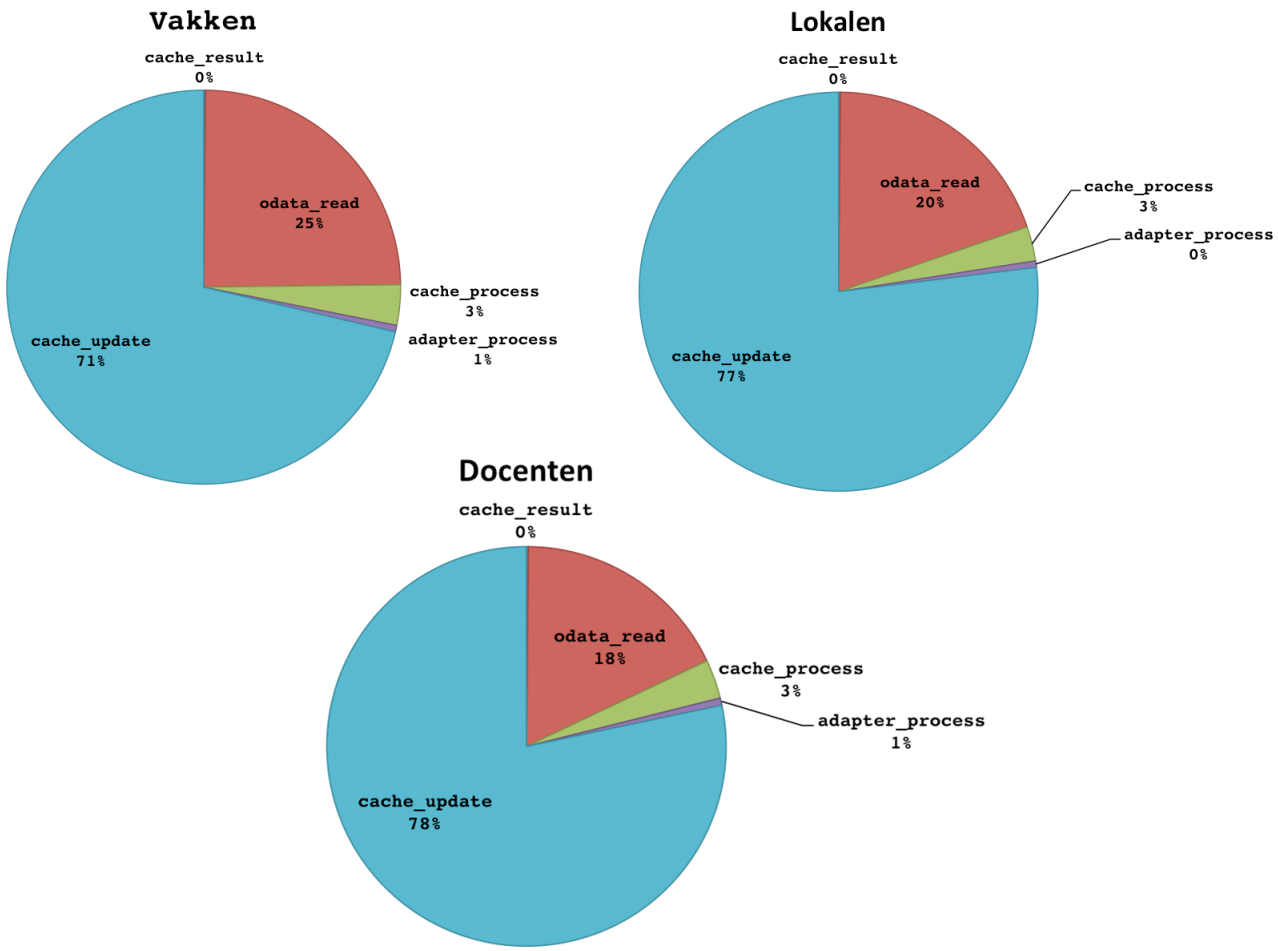
Meting	Verwerkingstijd (ms)		Aantal items
	μ	σ	
`odata_read_course`	557.4667	153.81	7
`adapter_process_course`	2.73	1.52	7
`odata_read_activity`	9935.20	1462.17	61
`adapter_process_activity`	95.60	2.41	61
`major_update_activity`	11076.40	925.14	185
`activity_expand_activity`	78.28	2.05	61
`model_loadAll_activity`	22250.80	1645.63	11

cache update maakt gebruik van een SQL-query per item, dit kan het verschil verklaren met het ophalen van gegegevens van de database, aangezien dit met één enkele query gebeurt.

6.3 Titanium

Titanium is gekozen voor dit project vanwege de mogelijkheid om voor beide groot vertegenwoordigde platforms (iOS en Android) tegelijk een een "native" applicatie te ontwikkelen. De ervaringen met Titanium worden hier besproken.

De taal JavaScript maakt de Titanium SDK toegankelijk voor beginnende programmeurs, en laat de ervaren programmeur compleet vrij in de stijl van programmeren. Heel veel programmacode werkt voor beide platforms. Daarnaast maakt het gebruik van CommonJS het mogelijk platformspecifieke code te schrijven in aparte directorystructuren, waardoor grote `if`-statements voor het controleren van het uitvoerende platform vermeden kunnen worden.



Figuur 6.1: Grafische representatie van de meetresultaten

Naarmate er meer gewerkt wordt met dit systeeem wordt ook steeds meer duidelijk dat dit geen sine cure is om een applicatie te ontwikkelen voor twee platforms tegelijk.

De API van Titanium is onregelmatig, de documentatie is vaak niet accuraat en in “debug”-modus worden exceptions regelmatig niet afgevangen. Vaak ontstaan er willekeurige problemen waardoor de applicatie crasht. Deze problemen zijn via het verwijderen van de build-directory vaak te verhelpen, maar dit is slechts symptoombestrijding.

HOOFDSTUK 7

Conclusie

De probleemstelling die ten grondslag ligt aan dit project luidt: de digitale dienstverlening van de UvA loopt op het mobiele platform achter, er zijn nog geen websites/webapplicaties speciaal geschikt voor studenten met smartphones. De ontwikkeling van een mobiele applicatie zou een uitkomst moeten bieden. Er is gekeken naar de huidige praktische situatie en hoe deze vastere vorm kan geven aan het idee "mobiele applicatie".

De enige bron van gestructureerde data is de DataNose webservice en deze levert voorlopig alleen roosterdata. De eisen voor de mobiele applicatie vinden hier hun basis: het leveren van roosterinformatie op basis van het vak en op basis van het studentnummer. De ontwikkeling van beide applicaties is vooralsnog niet gereed. De iPhone-applicatie voldoet aan de minimale functionele eisen, men kan er het eigen rooster mee bekijken en eveneens via vak, docent of lokaal, het rooster van dat item bekijken. De snelheid waarmee de data verwerkt wordt kan hoogstwaarschijnlijk nog erg verbeterd worden, het duurt nu 20 seconden voordat de gebruiker zijn/haar rooster kan zien.

De uiterlijke kenmerken maken de applicatie nog niet geschikt voor de markt. Er zitten nog twee visuele bugs in de applicatie, en er is nog niet uitvoerig getest door gebruikers. De Android-applicatie functioneert nagenoeg niet vanuit het perspectief van de gebruiker, aan de functionele eisen is niet voldaan.

De Titanium SDK stelt men in staat tegelijk voor iPhone en Android een applicatie te ontwikkelen. De voordelen waarmee het wordt aangeprezen, snelle ontwikkeling, gemakkelijke programmeertaal, uniforme programmeeromgeving voor iOS en Android, staan in sterk contrast met de ervaringen met het systeem. De API is onregelmatig (bepaalde functionaliteit die men ergens redelijkerwijs zou verwachten op basis van eerdere ervaring ontbreekt, of werkt anders), de documentatie is zeer gebrekkig, en de combinatie van de twee kan het programmeren ten zeerste bemoeilijken. De conclusie is dat Titanium nog een onvolwassen ontwikkelplatform is met meer haken en ogen dan men op het eerste gezicht zou verwachten.

7.1 Reflectie

De ontwikkeling van de mobiele applicaties is niet met succes verlopen. Er is tijdens het project in het begin veel aandacht gegaan naar het uiterlijk van de applicatie waardoor er te weinig tijd overbleef voor de invulling van de functionele eisen. Door te weinig te testen op het Androidplatform zijn heel veel problemen te laat ontdekt en kon de Android versie niet op tijd afgemaakt worden. De eerdere ervaring met programmeren voor iOS heeft me afgeleid van het feit dat mobiele applicatieontwikkeling voor een nieuw platform nog steeds een leercurve kan hebben,

zelfs wanneer men programmeert in JavaScript. De vele problemen in de Titanium SKD en ontwikkelomgeving hebben voor extra vertraging gezorgd. Er is vooraf geen uitgebreid onderzoek gedaan naar de API en de documentatie van Titanium, slechts oppervlakkige vergelijkingen, enigszins vanwege de tijdsdruk, maar ook omdat het niet nodig geacht werd. Het vroeg herkennen van deze patronen kan een volgend traject in betere banen doen lopen en het inschatten van de hoeveelheid werk verbeteren. Het opzetten van de architectuur van de backend van de applicatie is een redelijk succes. De klassenstructuur laat de backend open voor uitbreiding, iets dat aan de andere kant ook weer nodig is, want in de loop van het project zijn er veel op elkaar lijkende SQL-query's in de code terechtgekomen.

7.2 Aanbevelingen

In geval van verdere ontwikkeling applicaties voor de UvA zijn er een aantal aanbevelingen. Titanium kan zeker wel gebruikt worden voor cross-platform ontwikkeling, maar men dient een aantal zaken in acht te nemen. Ten eerste is Titanium een systeem dat meerdere API's (namelijk die van iOS en van Android) abstraheert en hier gaat onherroepelijk vrijheid verloren in ruil voor het gemak dat het oplevert. Ontwikkeling van visuele elementen en effecten die niet ingebouwd zitten in de Titanium SDK (bepaald soort animaties, of structuren) zal veel meer tijd kosten met Titanium dan wanneer men met webtechnologie werkt of native. Ten tweede moet men ervan bewust zijn dat Titanium JavaScript code *niet* compileert maar verwerkt in de application binaries, met als gevolg dat de code alsnog wordt geïnterpreteerd. Data-zware applicaties kunnen hier last van hebben en het is dan eventueel een beter idee om een native applicatie te ontwikkelen. Tenslotte zijn de verschillen tussen de API op de twee platforms iOS en Android niet te onderschatten. Veel visuele componenten die op dezelfde manier in het leven worden geroepen gedragen zich anders op het ene platform dan op het andere.

Het zelf ontwikkelen van een laag tussen de database en de rest van de applicatiecode kost veel moeite. Het kan erg praktisch zijn te zoeken naar eerdere gebruikte, geteste data-oplossingen zoals een Object Relational Mapping- of een Entity Attribute Value Model-implementatie zullen de cache betrouwbaarder en beter geschikt voor uitbreiding maken.

Wat betreft de verdere ontwikkeling van deze roosterapplicatie zijn er een aantal functies die de applicatie mist en nog goed kan gebruiken. Synchroniseren met de interne agenda-applicatie is er een van. Dit is een functie die veel mensen op prijs zullen stellen, maar de implementatie kan complex worden. De DataNose webapplicatie beidt een `.ics` file aan bij het bekijken van het persoonlijk rooster, men kan de agenda applicatie van de smartphone laten abonneren op deze "feed". Dit proces kan eventueel geautomatiseerd worden. Een kaartje op het lokaalinfoscherm dat aangeeft waar het lokaal is eventueel samen met een route-knop (leidend naar de interne Kaarten-applicatie van een smartphone) zal voor beginnende studenten praktisch kunnen zijn.

HOOFDSTUK 8

Bibliografie

[1] APK (file format). http://en.wikipedia.org/wiki/APK_(file_format).

[2] Facebook App voor Android. https://play.google.com/store/apps/details?id=com.facebook.katana&hl=nl.

[3] Facebook App voor iOS. http://itunes.apple.com/nl/app/facebook/id284882215.

[4] Facebook voor Android update: Snel & Smooth. http://www.myandroidphone.nl/nieuws/115/facebook_voor_android_update:_snel_&_smooth.

[5] Open data protocol. http://www.odata.org/.

[6] Appcelerator. CommonJS Modules in Titanium. https://wiki.appcelerator.org/display/guides/CommonJS+Modules+in+Titanium.

[7] Appcelerator. Titanium SDK. http://www.appcelerator.com/platform/titanium-sdk.

[8] Apple. iOS Dev Center. https://developer.apple.com/devcenter/ios/index.action.

[9] Codeplex. datajs - JavaScript Library for data-centric web applications. http://datajs.codeplex.com/.

[10] D. Gavalas and D. Economou. Development platforms for mobile applications: Status and trends. *Software, IEEE*, 28(1):77 –86, jan.-feb. 2011.

[11] R. Gopal. De UvA App, apr 2011. Universiteit van Amsterdam (bachelor's thesis).

[12] Motorola. Rhodes. http://www.motorola.com/Business/US-EN/RhoMobile+Suite/Rhodes.

[13] Gerrit Oomens. Datanose.nl. http://datanose.nl.

[14] Stack Overflow. What happens to javascript code after app is compiled using titanium mobile. http://stackoverflow.com/questions/4217551/what-happens-to-javascript-code-after-app-is-compiled-using-titanium-mobile/47985474798547.

[15] T. Paananen. Smartphone cross-platform frameworks (bachelor's thesis), 2011. Jamk University of Applied Sciences.

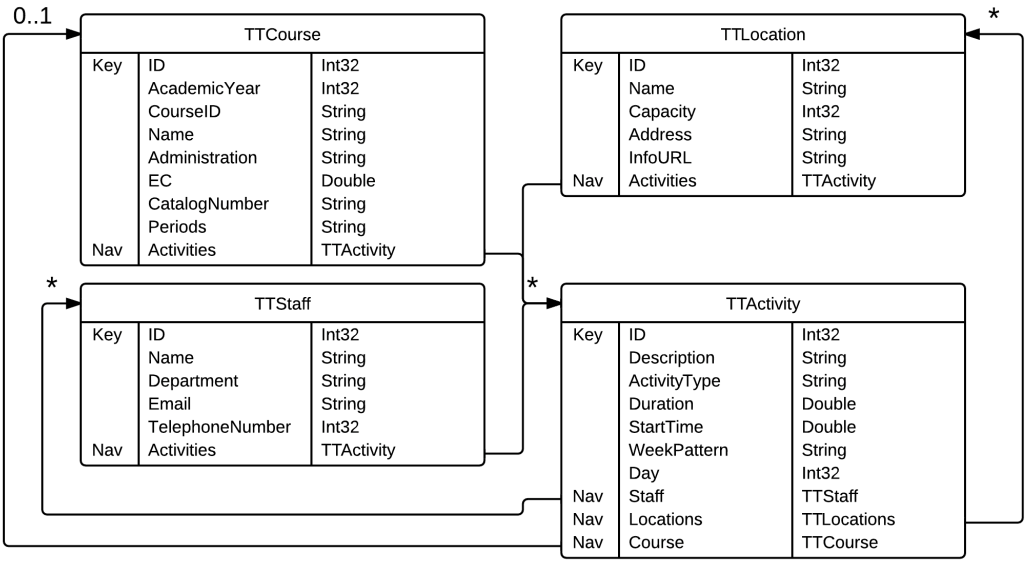
[16] Android Developers website. Exploring the Android SDK. http://developer.android.com/sdk/exploring.html.

[17] Android Developers website. What is the NDK? http://developer.android.com/tools/sdk/ndk/overview.html.

BIJLAGE A

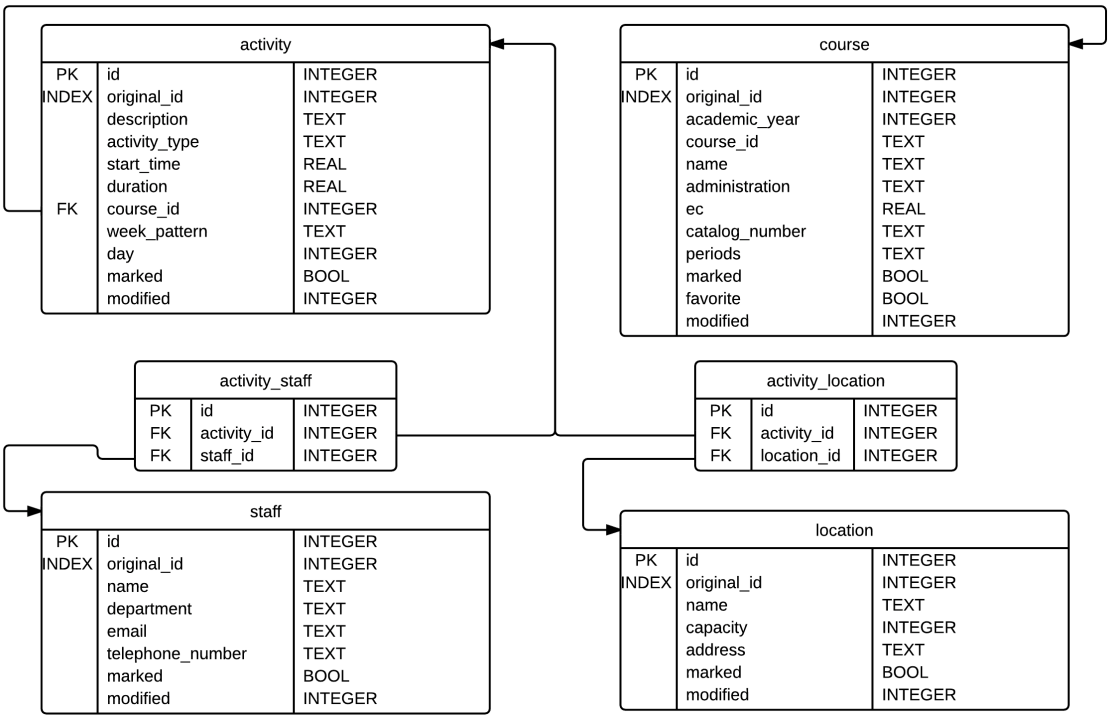
Data Modellen

A.1 Datanose EDM



Figuur A.1: Definities van de entity types

A.2 Cache Data Model



Figuur A.2: Het data model van de SQLite database

BIJLAGE B

Marktaandeel

In tabellen B.1, B.2 B.3 staan de 50 meest gebruikte user agent strings. De eerste 50 nemen 96,12% van het publiek voor hun rekening. De software waar dit overzicht mee is gemaakt kan enkele user agent strings vertalen naar browsernamen, maar meeste dient handmatig gedaan te worden. Het resultaat van deze interpretatie staat in de kolom “Smartphone OS”. Bij niet-handheld systemen staat er: “n.v.t”.

	Browser	Smartphone OS	Hits	Visitors	%ofTotalVisitors
1	Firefox	n.v.t	14.843.567	38.712	19,39%
2	Google Chrome	n.v.t	10.906.332	37.712	18,89%
3	Android Browser	Android	330.487	27.837	13,94%
4	Internet Explorer	n.v.t	4.883.078	22.413	11,22%
5	Mobile Safari	iOS	300.071	20.567	10,30%
6	Safari	n.v.t	2.170.473	20.342	10,19%
7	iOS/5.0.1 (9A405) dataaccessd/1.0	iOS	73.945	10.002	5,01%
8	Opera	n.v.t	320.113	3.003	1,50%
9	DataAccess/1.0 (8C148)	iOS	6.838	2.343	1,17%
10	iOS/5.0 (9A334) dataaccessd/1.0	iOS	23.273	1.366	0,68%
11	DataAccess/1.0 (8J2)	iOS	13.283	1.189	0,60%
12	Others	Unknown	165.349	931	0,47%
13	DataAccess/1.0 (8B117)	iOS	12.051	844	0,42%
14	MobileSafari/7534.48.3 CFNetwork/548.0.4 Darwin/11.0.0	iOS	3.677	667	0,33%
15	MobileSafari/7534.48.3 CFNetwork/548.0.3 Darwin/11.0.0	iOS	2.861	367	0,18%
16	Mozilla/5.0 (SAMSUNG; SAMSUNG-GT-S8500/S8500XXKF3; U; Bada/1.0; nl-nl) AppleWebKit/533.1 (KHTML, like Gecko) Dolfin/2.0 Mobile WVGA SMM-MMS/1.2.0 OPN-B	Bada	975	200	0,10%
17	BlackBerry9700/5.0.0.743 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/150	BlackBerry	997	199	0,10%
18	CalendarStore/5.0.1 (1139.14); iCal/5.0.1 (1547.4); Mac OS X/10.7.2 (11C74)	n.v.t	3.055	184	0,09%
19	BlackBerry8520/5.0.0.1036 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/120	BlackBerry	1.599	178	0,09%
20	SAMSUNG-GT-S5620/S5620CEJB4 SHP/VPP/R5 Dolfin/1.5 Nextreaming SMM-MMS/1.2.0 profile/MIDP-2.1 configuration/CLDC-1.1	BlackBerry	497	174	0,09%
	Subtotal		37.414.383	191.928	96,12%
	Total		37.482.251	199.677	100,00%

Tabel B.1: Top 20 van de meestgebruikte User Agent strings

	Browser	Smartphone OS	Hits	Visitors	%ofTotalVisitors
21	BlackBerry8520/5.0.0.681 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/134	BlackBerry	1.346	165	0,08%
22	BlackBerry9700/5.0.0.344 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/150	BlackBerry	1.458	156	0,08%
23	DataAccess/1.0 (8L1)	iOS	675	146	0,07%
24	DataAccess/1.0 (8A293)	iOS	178	144	0,07%
25	Mozilla/5.0 SF/2.03b	Unknown	3.312.735	135	0,07%
26	BlackBerry9700/5.0.0.593 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/603	BlackBerry	875	128	0,06%
27	Test Certificate Info	Unknown	131	128	0,06%
28	MobileSafari/6531.22.7 CFNetwork/485.2 Darwin/10.3.1	iOS	463	125	0,06%
29	Gecko/20080917 Sunbird/0.9	Unknown	388	124	0,06%
30	Apache-HttpClient/UNAVAILABLE (java 1.4)	Unknown	122	122	0,06%
31	BlackBerry9700/5.0.0.862 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/120	BlackBerry	832	108	0,05%
32	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534 (KHTML, like Gecko)	n.v.t	670	104	0,05%
33	Mozilla/5.0 (SymbianOS/9.3; Series60/3.2 NokiaE72-1/022.007; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/525 (KHTML, like Gecko) Version/3.0 BrowserNG/7.1.19974	Symbian	901	99	0,05%
34	BlackBerry9700/5.0.0.593 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/134	BlackBerry	1.194	90	0,05%
35	check_http/v1.4.15 (nagios-plugins 1.4.15)	Unknown	24.313	90	0,05%
36	Mozilla/5.0 (SAMSUNG; SAMSUNG-GT-S8500/S8500XXJI2; U; Bada/1.0; nl-nl) AppleWebKit/533.1 (KHTML, like Gecko) Dolfin/2.0 Mobile WVGA SMM-MMS/1.2.0 OPN-B	Bada	181	85	0,04%
	Subtotal		37.414.383	191.928	96,12%
	Total		37.482.251	199.677	100,00%

Tabel B.2: Top 21-36 van de meestgebruikte User Agent strings

	Browser	Smartphone OS	Hits	Visitors	%ofTotalVisitors
37	Mozilla/5.0 (SymbianOS/9.4; Series60/5.0 NokiaN97-4/11.0.102; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/525 (KHTML, like Gecko) BrowserNG/7.1.4	Symbian	380	84	0,04%
38	MobileSafari/6533.18.5 CFNetwork/485.13.9 Darwin/11.0.0	iOS	428	78	0,04%
39	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534 (KHTML, like Gecko) BingPreview/1.0b	n.v.t	482	72	0,04%
40	MobileSafari/6533.18.5 CFNetwork/485.12.7 Darwin/10.4.0	iOS	382	65	0,03%
41	BlackBerry8520/4.6.1.314 Profile/MIDP-2.0 Configuration/CLDC-1.1 VendorID/603	BlackBerry	1.326	56	0,03%
42	Mozilla/5.0 (SymbianOS/9.3; Series60/3.2 Nokia6700s/033.014; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/525 (KHTML, like Gecko) Version/3.0 BrowserNG/7.2.6	Symbian	193	52	0,03%
43	Mozilla/5.0 (SAMSUNG; SAMSUNG-GT-S8500/S8500XXJF8; U; Bada/1.0; nl-nl) AppleWebKit/533.1 (KHTML, like Gecko) Dolfin/2.0 Mobile WVGA SMM-MMS/1.2.0 OPN-B	Bada	629	51	0,03%
44	BlackBerry9300/5.0.0.832 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/120	BlackBerry	303	50	0,03%
45	Gecko/20110929 Thunderbird/7.0.1 Lightning/1.0b7	Unknown	111	48	0,02%
46	Microsoft Office Protocol Discovery	n.v.t	85	44	0,02%
47	Microsoft Office Outlook 2010	n.v.t	420	43	0,02%
48	Gecko/20111229 Thunderbird/9.0 Lightning/1.1.1	Unknown	109	38	0,02%
49	Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9A405	iOS	514	34	0,02%
50	Microsoft Office/14.0 (Windows NT 5.1; Microsoft Outlook 14.0.6023; Pro)	n.v.t	38	34	0,02%
	Subtotal		37.414.383	191.928	96,12%
	Total		37.482.251	199.677	100,00%

Tabel B.3: Top 37-50 van de meestgebruikte User Agent strings