ACE · GROEP T

ANTICIPATIVE CONTINUING EDUCATION

2010 Bastien Adriaan

2011 Bavin Michael

Mobile client-server applicatie met Android en Grails: DreamCar Spotter

Ondernemingsproject voorgedragen tot het behalen van het diploma van Gegradueerde in de Informatica

Promotor:
De heer Mathy Paesen

Promotor:
Dr. Ignaz Wanders *Faros nv*

2010 Bastien Adriaan
2011 Bavin Michael

Mobile client-server applicatie met Android en Grails: DreamCar Spotter

Ondernemingsproject voorgedragen tot het behalen van het diploma van Gegradueerde in de Informatica

Promotor:
De heer Mathy Paesen

Promotor:
Dr. Ignaz Wanders *Faros nv*

Dankwoord

Graag bedanken we onze promotoren:

Mathy Paesen, promotor Groep T, heeft ons project gedurende dit schooljaar gecoacht en onze planning goed opgevolgd waardoor we ons project tot een goed einde hebben kunnen brengen.

Ignaz Wanders, copromotor Faros NV, gaf ons de kans om ons project voor Faros NV te maken en gaf duidelijke instructies en inspiratie voor ons project.

Inhoudstabel

1. Inleiding

1.1. Algemeen

Het aandeel van de traditionele gsm neemt af en het die van de smartphone neemt toe, recente studies toonden aan dan 1 Belg op 5 in het bezit is van een smartphone. Op deze ICT trend wilden wij graag inspelen en daarom leek het ons een toffe uitdaging om een smartphone applicatie te ontwikkelen.

Op de smartphonemarkt zijn er 3 grote besturingssystemen te onderscheiden, namelijk:

- Android (Google)
- iOs (Apple)
- Windows Mobile (Microsoft)

Het Android besturingssysteem is Java gebaseerd en wint samen met smartphones constant aan populariteit. Aangezien ontwikkeling in Java voor ons een voor de hand liggende keuze was besloten we ons op Android te focussen en een mobiele client-server applicatie in Java met behulp van Grails als web-framework te ontwikkelen.

Het einddoel van het project is een gebruiksklare mobiele applicatie, voor Android telefoons, te ontwikkelen en deze gratis beschikbaar stellen via de **Android Market**[1].Deze mobiele applicatie moet optimaal gebruik maken van alle voornaamste functionaliteiten van de moderne smartphone, denk maar aan:

- Touchscreen
- Locatievoorziening (GPS)
- Ingebouwde Camera
- Mobiel Internet

[1] Voor meer informatie, zie punt **2.3.2 Android Market** op pagina 9.

1.2. Timing

Ons project bestaat uit fases:

1. **Initiatiefase**
   - Scope
   - Planning
2. **Analyse & Ontwerpfase**
   - Business Analyse
   - Database analyse
   - Requirement analyse
   - UML Diagram
3. **Implementatiefase**
4. **Integratie, test & verificatiefase**
5. **Werking & onderhoudsfase**

De gedetailleerde timing is leesbaar op onze GANTT-chart. Zie appendix 7.1 Planning.

1.3. Aanpak

Projectmatig hebben we dit aangepakt door vergaderingen te plannen op regelmatige tijdstippen en optimaal te profiteren van onze 2 wekelijkse vergaderingen op GroepT.

We gebruikten www.dropbox.com om al onze documentatie bij te houden en up te daten.

Onze planning werd steeds strak opgevolgd door onszelf en onze promotor Mathy Paesen.

2. Toepassingsgebied (Scope)

2.1. Project scope

2.1.1. Concept

Een smartphone applicatie voor Android toestellen bedoeld voor autoliefhebbers, genaamd **DreamCar Spotter**.

2.1.2. Hoe?

Gebruikers zullen gegeolocaliseerde foto's kunnen maken van exclusieve auto's die iedereen wel eens op het straatbeeld tegenkomt. Vervolgens zullen gebruikers via de geolocatie op een interactieve kaart kunnen zien of er in de buurt ook exclusieve wagens staan, die getagged worden door andere gebruikers van de applicatie. Onder andere is het mogelijk om te kunnen reageren evenals een beoordeling te geven op iemand zijn 'Spot'. Vervolgens moet het ook mogelijk zijn een ingediende 'Spot' te delen via sociale media zoals Facebook.

2.2. Faros NV

2.2.1. Het bedrijf

Faros is in 2004 opgericht als onderdeel van de Cronos Groep en is vooral actief op het vlak van Java EE consultancy.

Meer informatie: www.faros.be

2.2.2. Situering Project

Ons project is een try-out binnen Faros NV. Op gebied van Android ontwikkeling heeft Faros NV relatief weinig interne kennis.
Faros ondersteunt het project enerzijds door de grote interesse ernaar en anderzijds als training -en kennisopbouw instrument voor een werknemer.
Het software model dat ontwikkeld werd kan bruikbaar zijn voor andere mobiele Android applicaties.

2.3. Google AndroidBesturingssysteem

2.3.1. Algemeen

Google Android is het mobiele besturingssysteem platform van Google gebaseerd op de Linux en Java technologie. Het op 5 november 2007 geïntroduceerde Google Android is ontwikkeld door Google in samenwerking met de Open Handset Alliance, De OHA.
In het eerste kwartaal van 2011 werden er 350.000 Android-telefoons per dag verkocht en Android was daarmee het meest verkochte besturingssysteem op mobiele telefoons.

2.3.2. Android Market

Een online winkel waar zowel gratis al betalende software kan aangeschaft worden voor de Android smartphone.
Zie ook: http://market.android.com

2.4. Probleemanalyse & Systeemeisen

2.4.1. Gewenst eindresultaat

Een afgewerkte en gebruiksvriendelijke smartphone applicatie die we gratis beschikbaar stellen via de Android Market.

2.4.2. Commerciële beperkingen

De applicatie wordt interessant voor eindgebruikers als er veel interactie is. Vanaf de applicatie in de Android Market verschijnt, zal deze dus best snel aan populariteit winnen, anders wordt het een flop.

2.4.3. Technische beperkingen

De interactie tussen de 2 modules willen we graag zo gemakkelijk mogelijk houden. We maken gebruik van **RESTful Web Services**[2], maar hier is de beveiliging een niet te onderschatten risico.

[2]**Representational State Transfer (REST)** is een software-architectuur voor gedistribueerde mediasystemen zoals het World wide web. Meer uitleg over de implementatie van RESTful web services in ons project vindt u op pagina 28.

2.5. Sterkte-Zwakte Analyse (SWOT)

2.5.1. Schema:

	HULPVOL (t.o.v. het doel)	**SCHADELIJK (t.o.v. het doel)**
INTERNE HERKOMST	**STERKTES** Grote toegankelijkheid: - Engels - Gratis - Gebruiksvriendelijk - Marktbeschikbaarheid Groot interessegebied binnen doelgroep (mannen>21j) Sociale factor	**ZWAKTES** Grote afhankelijkheid: - Platform (OS) - Internet - Medium (Smartphone) Beperkt interessegebied (algemeen) Marketing budgettering
EXTERNE HERKOMST	**KANSEN** Groeiende populariteit van Smartphones Toegankelijkheid van Mobiel internet Winst d.m.v. advertenties Externe interesse	**BEDREIGINGEN** Geen eigen data: Afhankelijk van gebruikersinhoud Niet meer uniek Strakke deadline Schaalbaarheid

2.5.2. Sterktes

Toegankelijkheid:

- De applicatie is Engels
- De applicatie is gratis
- De applicatie is gebruiksvriendelijk
- Via de Android Market is het gemakkelijk om de applicatie te vinden & installeren

Interessegebied:

Smartphone gebruikers zijn voornamelijk mannen met dikwijls een passie voor mooie auto's.

Sociaal:

Vanuit de applicatie kan men informatie publiceren op sociale netwerksite **Facebook**[3]en op deze manier kan de applicatie aan populariteit winnen.

2.5.3. Zwaktes

- Onze applicatie is platform afhankelijk
- Onze applicatie vereist een actieve internetverbinding
- Onze applicatie vereist een moderne smartphone (Android OS)
- Het interessegebied is beperkt
- Er wordt aan marketing gedaan zonder gereserveerd budget

2.5.4. Kansen

- **Technologisch klimaat:**
  - Smartphones, zoals Android toestellen, worden populairder wereldwijd
  - Mobiel Internet is toegankelijker dan ooit
- **Economisch voordeel:**
  - **Google AdMob**[4]: Door onze applicatie te voorzien van advertenties kunnen we een bepaalde omzet genereren die kan uitdraaien op winst

[3]**Facebook** (www.facebook.com) is een populaire sociale netwerksite met wereldwijd al meer dan 680 miljoen gebruikers. Via facebook kunnen gebruikers informatie delen met elkaar, zoals bijvoorbeeld bepaalde gegevens via externe (mobiele) applicaties.

[4]**Google AdMob**: Programma dat advertenties aanpast naar de inhoud van mobiele applicaties en de ontwikkelaars beloont per klik.

- Een goed eindresultaat kan potentiële vraag opwekken bij geïnteresseerde bedrijven die een gelijkaardige applicatie willen laten ontwikkelen.

2.5.5. Bedreigingen

- De applicatie is 'web 2.0' gebaseerd wat er onder andere op neerkomt dat we geen eigen data hebben en afhankelijk zijn van alle gebruiker ingezonden inhoud.
- Tijdens de ontwikkelingsfase van de applicatie aan kan er een gelijkaardige applicatie op de Android Market gepubliceerd worden.
- De applicatie geraakt niet tijdig klaar wegens onverwachte technische hindernissen. (deadline eind mei)
- Indien de applicatie heel populair wordt moet de applicatie-infrastructuur en implementatie hierop voorzien zijn = software – schaalbaarheid.

3. Analyse &Ontwerp

3.1. Requirements Analyse

3.1.1. Identificatie Actoren

- Geregistreerde gebruiker
- Ongeregistreerde gebruiker

3.1.2. Use Case Prioriteiten

Naam	Prioriteit
Authenticatie	Hoog
Register	Hoog
Manage own Entry	Gemiddeld
Like other Entry	Laag
Comment Entry	Laag
Submit new Entry	Hoog
Share submitted Entry	Laag
Explore existing Entries	Gemiddeld

3.1.3. Use Case Beschrijvingen

1. **Authenticate**
   Een geregistreerde gebruiker maakt aan de applicatie bekend wie hij is met behulp van zijn inloggegevens (gebruikersnaam & wachtwoord)
2. **Register**
   Een ongeregistreerde gebruiker dient eerst te registreren alvorens hij kan inloggen om de applicatie te kunnen gebruiken
3. **Manage own Entry**
   De gebruiker kan zijn eigen **Entry**[5] gaan bewerken of verwijderen
4. **Like other Entry**
   De gebruiker kan entries van andere gebruikers gaan **liken**[6]en dit meegeven. Zo kan er een onderscheid gemaakt worden tussen

[5]**Entry**: Dit omvat het geheel dat een gebruiker verstuurd wanneer hij een nieuwe foto doorstuurd met daarbij informatie zoals merk, model, locatie, commentaar.

verschillende entries die de gebruikers al dan niet leuker vinden door middel van het aantal 'likes'.

5. **Comment Entry**
   De gebruiker kan commentaren plaatsen in de vorm reguliere tekst.
6. **Submit new Entry**
   De gebruiker kan een nieuwe Entry aanmaken in, hiervoor zal er een foto in zijn camera-album aanwezig moeten zijn of zelf een directe momentopname moeten maken.
7. **Share submitted Entry**
   De gebruiker kan een Entry delen door middel op verschillende manieren. Zowel in de vorm van een notificatie op Facebook als in de vorm van een e-mail via de ingebouwde mailapplicatie.
8. **Explore existing Entries**
   De gebruiker zal kunnen bladeren tussen de bestaande entries, dit enerzijds door een overzicht te maken van de meest bekeken, de meeste likes, de recentste, of door middel van een kaart waarop de entries zijn getagged.

[6]**Liken**: 'To like something' 'Iets leuk vinden' Op sociale netwerksites wordt deze term vaak gebruikt om als gebruiker aan te geven dat je een bepaald online item leuk vindt.

3.1.4. Gedetailleerde Use Case beschrijvingen

UC-0: Algemene Exception

➔ De actor heeft geen internetverbinding (3G of wifi)

UC-1: Authenticatie

Een geregistreerde gebruiker maakt aan de applicatie bekend wie hij is met behulp van zijn inloggegevens (gebruikersnaam & wachtwoord)

Triggering event	De actor opent de applicatie om een nieuwe sessie te starten
Actors	Geregistreerde gebruiker
Pre-condition	De actor heeft een actief account
Post-condition	De actor heeft nu toegang tot de applicatie en kan gegevens wijzigen
Flow of events	**1.** De actor opent de applicatie **2.** De actor krijgt het scherm te zien om in te loggen **3.** De actor vult zijn gebruikersnaam en wachtwoord in en eventueel een vinkje om zijn inloggegevens te onthouden **4.** De actor verzendt zijn gegevens met de knop "log in" **5.** De actor wordt doorverwezen naar het hoofdscherm
Exception	• De actor gebruikt verkeerde inloggegevens waardoor hij een foutmelding krijgt dat de authenticatie niet gelukt is.
Alternative flows	A. Na stap 1 kan het zijn dat de inloggegevens al bewaard zijn, waardoor er automatisch ingelogd wordt en naar het hoofdscherm kan worden genavigeerd

UC-2: Registratie

Een ongeregistreerde gebruiker dient eerst te registreren alvorens hij kan inloggen om de applicatie te kunnen gebruiken

Triggering event	De actor opent de applicatie om een nieuwe sessie te starten, maar kan niet inloggen omdat hij nog geen inloggegevens heeft.
Actors	Ongeregistreerde gebruiker
Pre-condition	De actor heeft nog geen actief account

Post-condition	De actor kan nu inloggen volgens UC-1
Flow of events	**1.** De actor opent de applicatie **2.** De actor krijgt het scherm te zien om in te loggen **3.** De actor gebruikt de knop "register" **4.** De actor vult gebruikersnaam, wachtwoord en email adres in **5.** De actor verzendt zijn gegevens met de knop "OK" en krijgt een boodschap dat hij geregistreerd is **6.** De actor kan nu inloggen zoals beschreven in UC-1
Exception	• De actor geeft ongeldige gegevens zoals lege gebruikersnaam, ongeldig email adres waardoor hij een foutboodschap te zien krijgt
Alternative flows	A. De gebruiker heeft wel een account, dus wil hij navigeren van het registreerscherm, naar het inlogscherm door middel van de knop "back"

UC-3: Manage own Entry

De gebruiker kan zijn eigen Entry gaan bewerken of verwijderen

Triggering event	De actor opent de applicatie en kan via een overzicht van zijn entries gaan kiezen om een specifieke Entry te kunnen aanpassen of verwijderen
Actors	Geregistreerde gebruiker
Pre-condition	De gebruiker heeft ten minste 1 entry aangemaakt in het verleden
Post-condition	De actor heeft zijn specifieke Entry gewijzigd of verwijderd.
Flow of events	**1.** De actor opent de applicatie **2.** De actor logt in zoals beschreven in UC-1 **3.** De actor kan een overzicht opvragen van zijn reeds bestaande entries **4.** De actor kan zijn gekozen entry aanpassen via de 'Modify' knop. **5.** De actor maakt de nodige aanpassingen aan zijn entry of verwijderd zijn entry door de 'Delete' knop te gebruiken **6.** De actor bevestigt zijn aanpassingen door middel van de knop 'Save modifications' te gebruiken.

Exception	• Indien er verplichte velden zijn zoals 'titelveld' en eventueel andere
Alternative flows	A. De actor wilt niets wijzigen of verwijderen en gebruikt de knop 'cancel' en navigeert dan automatisch terug naar het overzicht.

UC-4: Like other Entry

De gebruiker kan entries van andere gebruikers gaan 'liken' en dit meegeven. Zo kan er een onderscheid gemaakt worden tussen verschillende entries die de gebruikers al dan niet leuker vinden d.m.v. het aantal 'likes'.

Triggering event	De actor bekijkt een bestaande entry van een andere gebruiker en kan deze beoordelen door te 'liken' en daarna eventueel deze actie ongedaan maken door te 'unliken'
Actors	Geregistreerde gebruiker
Pre-condition	Andere gebruikers hebben entries ingegeven
Post-condition	De actor heeft een specifieke entry geliked of geunliked
Flow of events	**1.** De actor opent de applicatie **2.** De actor logt in zoals beschreven in UC-1 **3.** De actor kan een overzicht opvragen van de bestaande entries **4.** De actor kan een entry liken via de 'like' knop of 'disliken' nadat hij 'geliked' heeft.
Exception	• Er zijn geen entries van andere gebruikers
Alternative flows	A. Geen

UC-5: Comment Entry

De gebruiker kan commentaren plaatsen in de vorm reguliere tekst.

Triggering event	De actor bekijkt een bestaande entry en kan een commentaar invullen door de comment knop te gebruiken en vervolgens in het tekst-invoerveld commentaar te plaatsen en deze te verzenden.

Actors	Geregistreerde gebruiker
Pre-condition	Er bestaat minstens één entry De actor is ingelogd en bevindt zich op de home pagina
Post-condition	De actor heeft een commentaar geplaatst op een entry
Flow of events	**1.** De actor navigeert naar een bepaalde entry **2.** De actor klikt op de comment knop **3.** De actor vult commentaar in **4.** De actor verzendt commentaar
Exception	• De actor heeft niets ingevuld in het invoerveld en op de 'send' knop gebruikt.
Alternative flows	A. Bij het invoeren van commentaar beslist de actor om te annuleren door gebruik te maken van de 'cancel' knop. Hierdoor navigeert de actor terug naar de overzicht op het vorige scherm B. Wanneer de actor een commentaar van zichzelf op een entry bekijkt kan hij deze verwijderen door middel van een 'delete' knop

UC-6: Submit new Entry

De gebruiker kan een nieuwe entry aanmaken, hiervoor zal hij een foto in zijn camera-album moeten hebben staan of zelf een momentopname moeten maken

Triggering event	De actor opent de applicatie en kan via de menuoptie 'spot car' kiezen om een bestaande foto te uploaden of te kiezen om een nieuwe foto te nemen met de camera
Actors	Geregistreerde gebruiker
Pre-condition	De actor is ingelogd en bevindt zich op de home pagina De geüploade foto is voorzien met een geotag
Post-condition	De actor heeft een nieuwe entry aangemaakt en deze komt logischerwijs in het overzicht van 'my entries'
Flow of events	**1.** De actor klikt op de 'spot car' knop **2.** De actor kiest/neemt een foto **3.** De actor vult bijkomende gegevens in a. Model b. Type c. Commentaar **4.** De actor klikt op submit

Exception	• Als de foto geen geotag bevat • Als er geen type wordt geselecteerd
Alternative flows	A. Bij het submitten van een entry beslist de actor om te annuleren door gebruik te maken van de 'cancel' knop. Hierdoor navigeert de actor terug naar de homepagina

UC-7: Share submitted Entry

De gebruiker kan een Entry delen door middel op verschillende manieren. Zowel in de vorm van een notificatie op Facebook als in de vorm van een e-mail via de ingebouwde mailapplicatie.

Triggering event	De actor opent de applicatie en navigeert naar 'my entries' en kan vervolgens een bestaande entry bekijken en deze delen via de optie 'share this entry' waarop de vraag wordt gesteld via welk medium hij deze wil delen.
Actors	Geregistreerde gebruiker
Pre-condition	De actor is ingelogd en heeft entries. De actor heeft een account op een van de externe services (facebook, email account)
Post-condition	De actor heeft zijn entry gedeeld via de gekozen service
Flow of events	**1.** De actor navigeert naar 'my entries' **2.** De actor kiest een specifieke entry **3.** De actor klikt op 'share this entry' **4.** De actor kiest het medium **5.** De actor vult extra gegevens in die het gekozen medium vereist (commentaar, emailbericht, bestemmeling, ...) **6.** De actor verzendt
Exception	• De externe service is niet beschikbaar
Alternative flows	A. Bij het sharen van een enty beslist de actor om te annuleren door gebruik te maken van de 'cancel' knop. Hierdoor navigeert de actor terug naar zijn gekozen entry

UC-7: Logout

De gebruiker maakt de applicatie bekend dat hij wil uitloggen zodat andere gebruikers de mogelijkheid krijgen om in te loggen.

Triggering event	De actor opent de applicatie en navigeert naar 'Logout' vervolgens zal de applicatie een bevestigingsboodschap terugsturen en terug op het beginscherm (zie usecase 1) terugkomen
Actors	Geregistreerde gebruiker
Pre-condition	De actor is ingelogd.
Post-condition	De actor is uitgelogd.
Flow of events	**1.** De actor navigeert naar hoofdmenu (optioneel) **2.** De actor klikt op'logout'
Exception	• De externe service is niet beschikbaar
Alternative flows	A. Geen

3.2. Extra functionaliteiten

Tijdens de ontwikkelingsfase van ons project kwamen er, mede dankzij de input van Mathy Paesen ,Groep T promotor en Ignaz Wanders, Faros NV en het groeien van de applicatie in het algemeen, extra innovatieve ideeën boven water drijven. We hebben dus nog wat extra functionaliteit voorzien dan in eerste instantie hierboven beschreven werd in de analyse & ontwerpfase.

3.2.1. Flagging

Flagging is een concept dat een gebruiker kan aantonen dat een entry van een andere gebruiker onjuiste informatie bevat of spam, aanstootgevend of ongepast is. Hierdoor kunnen we de community van gebruikers gebruiken om te modereren.

Deze entries worden zichtbaar in de admin-sectie en kunnen vervolgens verwijderd worden.

Voorbeelden van slechte posts:

- Reclame & spam
- Pornografisch materiaal
- Onherkenbare foto's
- Nissan Micra's

3.2.2. Leaderboard

Om gebruikers te motiveren om actiever onze applicatie te gebruiken besloten we een puntensysteem in te bouwen.
Gebruikers kunnen een 'leaderboard' consulteren waarin de trouwste gebruikers vanboven gerangschikt staan.

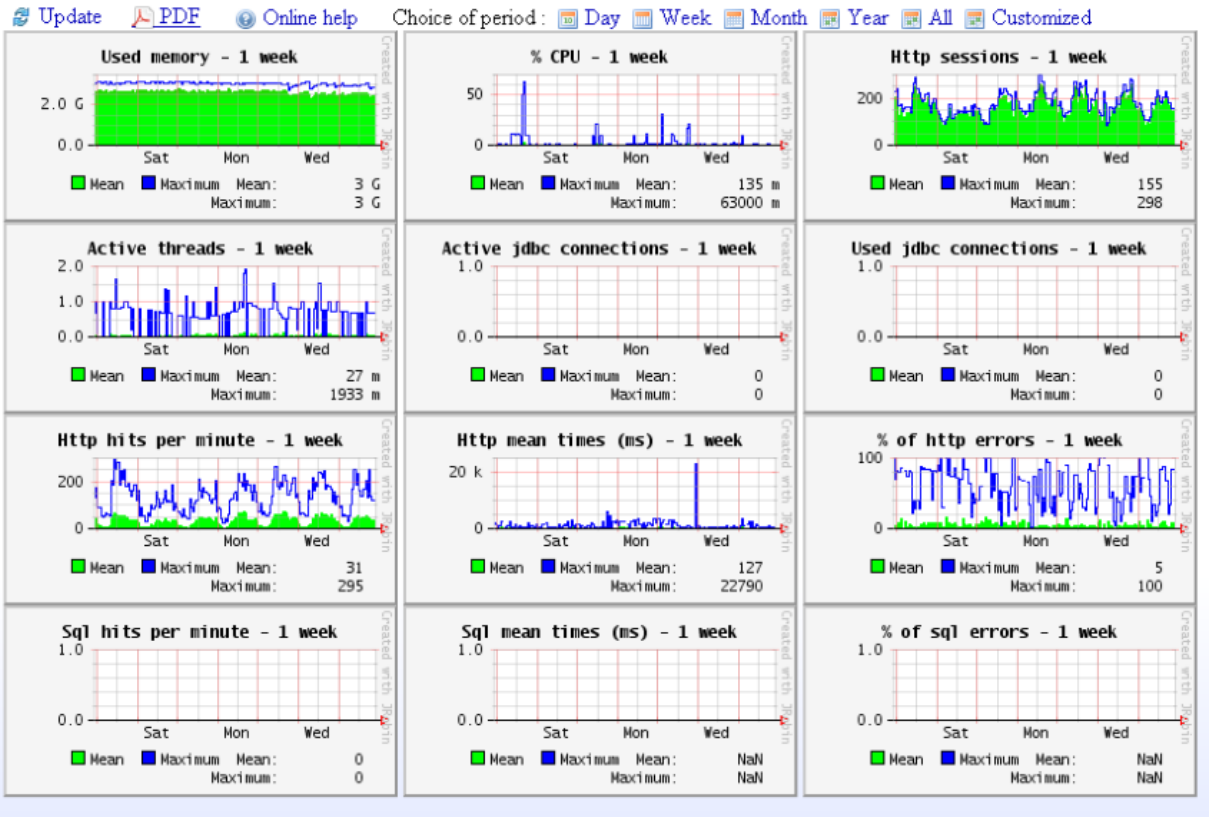
Men kan punten verdienen met volgende acties:

- Nieuwe post = +5 punten
- Post verwijderen -5 punten
- Andere post liken = +1 punt
- Eigen post wordt geliked = +2 punten

3.2.3. Admin sectie

Er bestaat nu ook een beheersgedeelte voor gebruikers die de rol 'administrator' hebben, daar kunnen zij entries kunnen verwijderen of zoals hierboven vermeld de lijst van onjuiste posts raadplegen (flagging).
Statistieken kunnen via **JavaMelody**[7] kunnen geraadpleegd worden.

[7] **JavaMelody** is een framework om monitoring toe te passen op de JVM. (Java Virtual Machine)



De afbeelding hierboven illustreert de grafieken die JavaMeldoy genereerd.

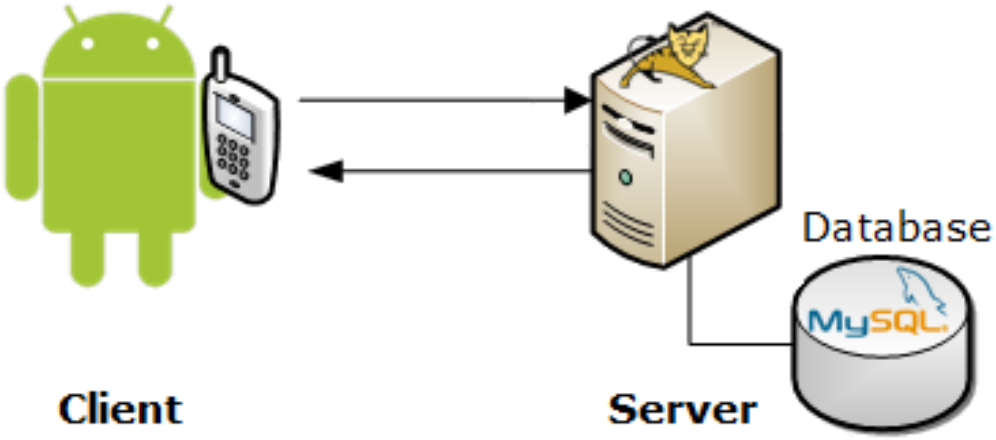
3.3. Client Server Applicatie

3.3.1. Algemeen

Het client-servermodel is een model voor de samenwerking tussen twee of meer programma's, die zich op verschillende computers kunnen bevinden. Kenmerkend voor de asymmetrie in het model is:

- Een **client** is bij gelegenheid actief en neemt het initiatief tot communicatie met de server
- De **server** is permanent beschikbaar en is reactief

3.3.2. Toepassing



- **Client**: De 'DreamCar Spotter' applicatie geïnstalleerd op een Android-smartphone.
- **Server**: De Apache Tomcat webserver met een MySQL database

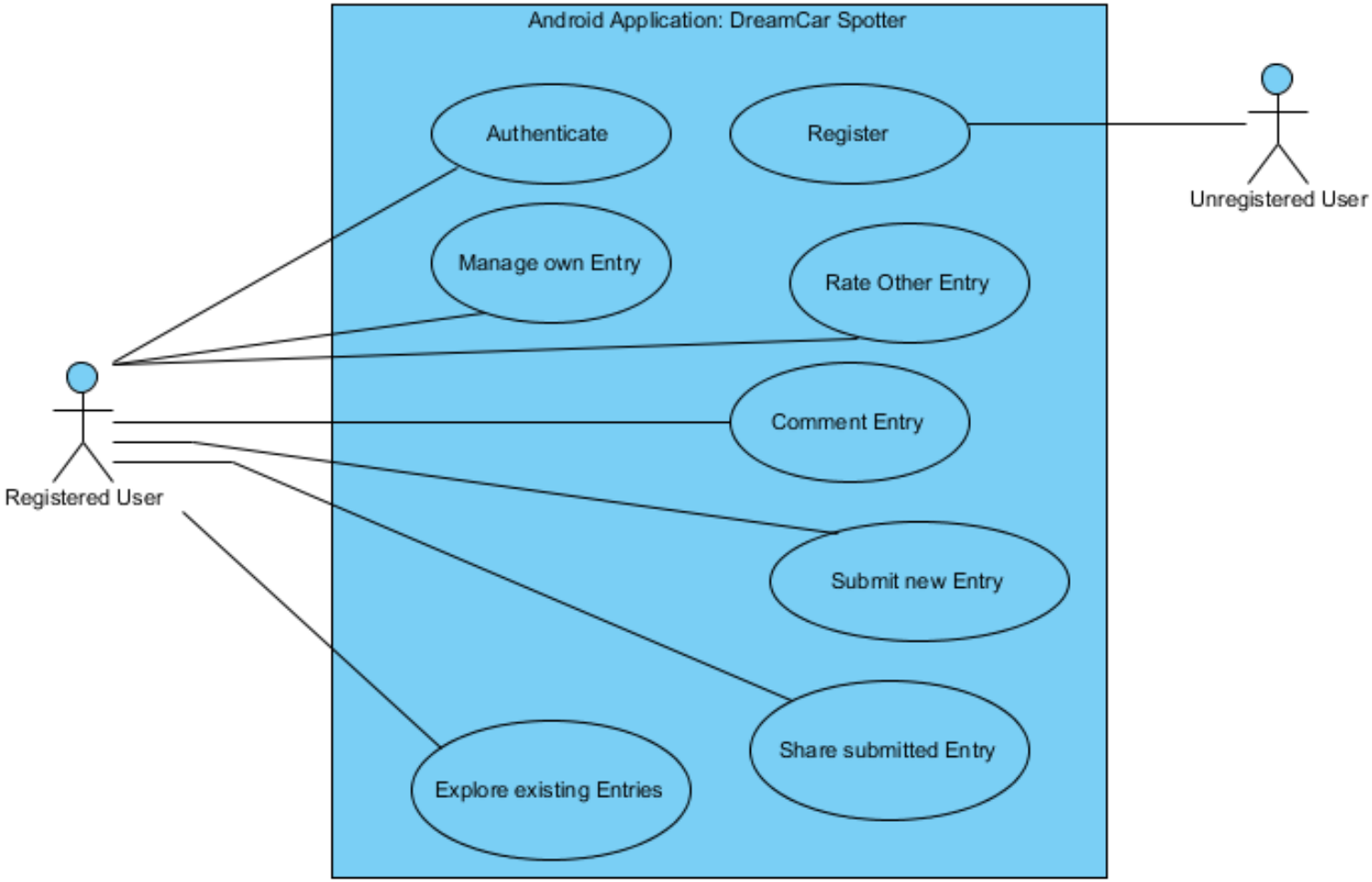
Communicatie tussen Client & server is mogelijk via een actieve netwerkverbinding.

3.4. UML Diagrammen

3.4.1. Use Case Diagram

Use case-diagrammen worden voornamelijk gebruikt in software engineering om de requirements van een systeem in kaart te brengen.

Het doel van het use case-diagram is om een grafisch overzicht te geven van de functionaliteiten van een systeem in termen van actoren.



3.5. Data-model

3.5.1. Aanpak

We hebben een domeinmodel opgezet via **Groovy**[8] klassen, elk object heeft zijn eigen klasse met daarin de properties.
Op de volgende pagina vindt men het klassendiagram waarin al deze klassen zichtbaar zijn samen met hun relaties.
Hibernate[9]wordt gebruikt om van ons datamodel een database-schema te genereren.

3.5.2. Datamodel aanpassingen

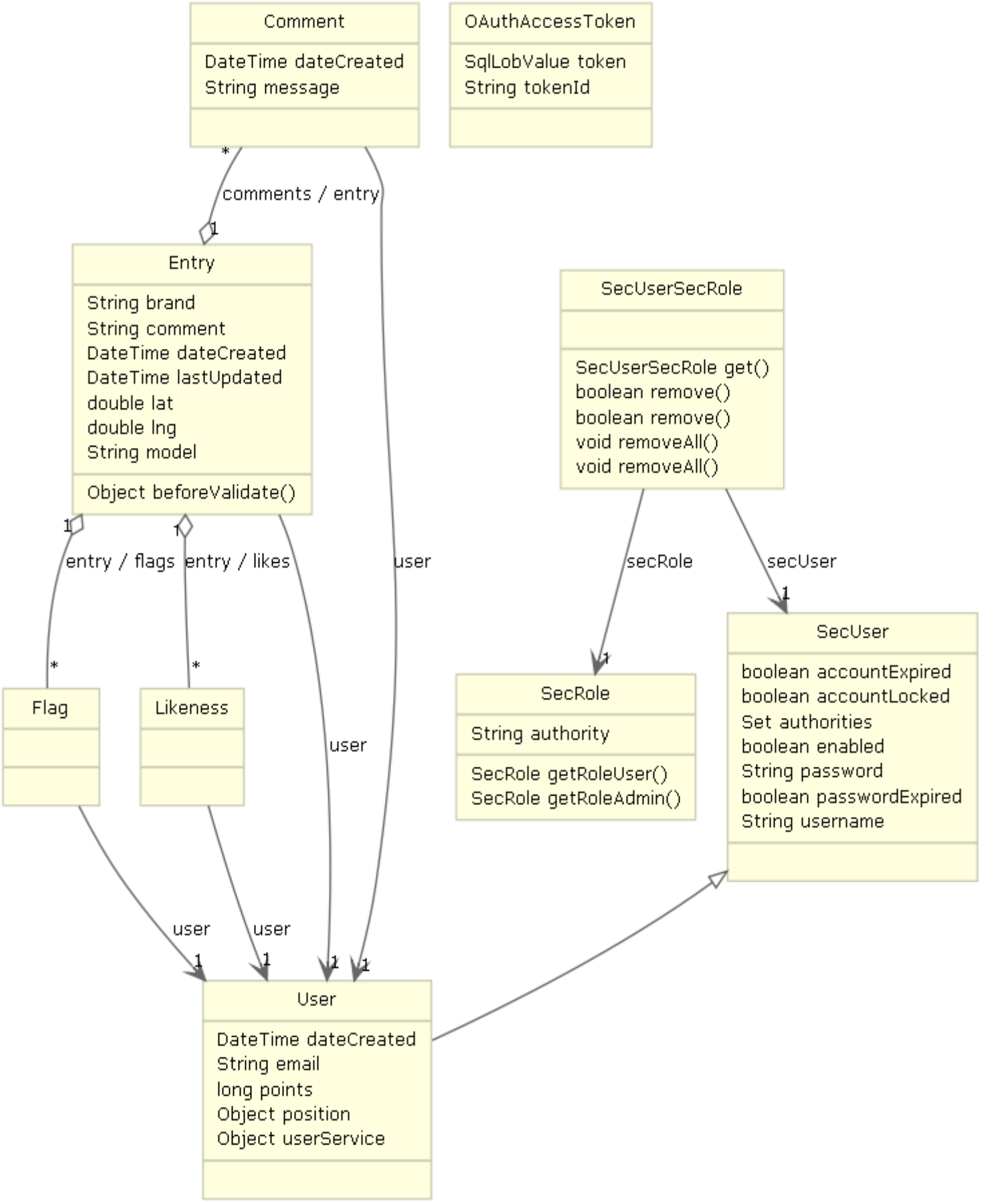
Wanneer we ons datamodel aanpassen zal Hibernate ons database-schema updaten.
Grails[10] kan complexe database migraties uitvoeren via plug-ins gebaseerd op Liquibase.

Meer informatie over deze plugin is terug te vinden op:
http://www.liquibase.org/home

[8]**Groovy** is een objectgeoriënteerde programmeertaal voor het Java-platform als alternatief voor de programmeertaal Java. Het kan worden beschouwd als een scripttaal voor het Java-platform aangezien de taal te vergelijken is met Ruby, Python, Perl en Smalltalk.
[9]**Hibernate** is een ORM-oplossing (Object/Relational Mapping) voor de programmeertaal Java. Het is een framework voor het koppelen van een objectgeoriënteerd domeinmodel aan een traditionele relationele database.
[10]**Grails** is een open source web applicatie framework dat de Groovy programmeertaalgebruikt. Het voorziet een hoog-productiviteits framework voor het Java platform, vertrekt ontwikkelaars een stand-alone-ontwikkelomgeving en verbergt een groot deel van de configuratie details.

3.5.3. Klassendiagram

Het klassendiagram beschrijft de klassen die een systeem vormen en het statische verband tussen hen. De klassen worden gedefinieerd in termen van hun naam, attributen (of gegevens), en gedrag (of methodes). De statische relaties zijn associatie, aggregatie, en overerving.



3.6. REST webservice tussen Grails & Android

3.6.1. Algemeen

Representational State Transfer (**REST**) is een software architectuur om webservices te creëren op basis van bestaande en eenvoudige protocollen van het internet. Elke applicatie die toegang heeft om via **HTTP** te communiceren kan gebruik maken van een dergelijke REST webservice.

Het grootste verschil tussen REST en **SOAP** webservices is dat REST resources zal opvragen, terwijl SOAP aan **R**emote **M**ethod **I**nvocation doet. SOAP zal dus externe methodes oproepen zoals berekeningen, terwijl REST resources zal opvragen of manipuleren die overeenkomen met een unieke URI.

Om resources te bereiken wordt er gebruik gemaakt van HTTP methodes zoals **GET**, **PUT**, **DELETE**, **POST**. Wanneer we bijvoorbeeld resource A met address */resources/A* willen opvragen, zullen we op deze URL een GET methode afvuren. Willen we deze resource verwijderen gebruiken we de DELETE methode. PUT wordt gebruikt om een resource te updaten met nieuwere data. Een nieuwe resource aanmaken doen we via de POST.

3.6.2. Platform onafhankelijkheid

De client applicatie mag in eender welke programmeertaal ontwikkeld zijn, dankzij REST kan deze, eventueel na autorisatie, makkelijk communiceren met de serverapplicatie.

3.7. Grafisch Ontwerp

De standaard Android applicatie interface bestaat uit een niet visueel aantrekkelijke zwarte achtergrond met grijze knoppen.

Het creëren van een uniforme look en feel in een user interface voegt waarde toe aan een product. Het stroomlijnen van de grafische stijl zal de gebruikersinterface professioneler doen uitschijnen.

3.7.1. Aanpak

Om een idee te hebben van welke grafische componenten en iconen we nodig hadden tekenden we een **wireframe**[11] uit.

3.7.2. Richtlijnen

Op de Android Developer website staan richtlijnen om ontwikkelaars te helpen om de iconen voor de verschillende delen van de gebruikersinterface aan te passen binnen het kader van de algemene Android 2.x stijl.

Link Android Developer website richtlijnen:
http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

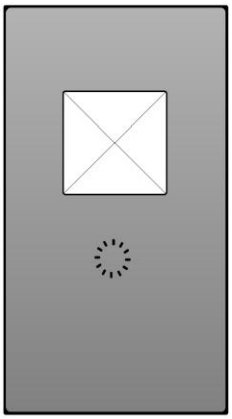
3.7.3. Grafische componenten

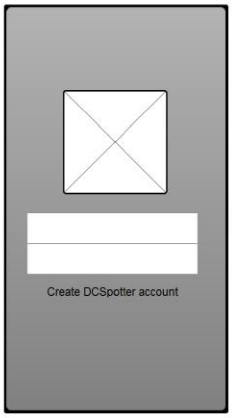
Om dit even in kaart te brengen, dit komt neer op de ontwikkeling van **40** verschillende Grafische componenten zoals:
- Menu icoontjes
- Detail icoontjes
- Achtergrondafbeeldingen
- Logo's

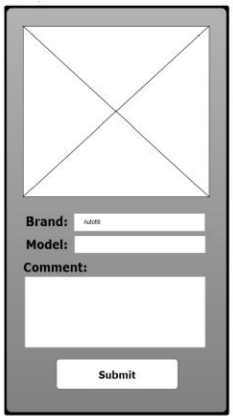
Dit geeft een totaal van 120 componenten omdat we 3 verschillende resoluties ondersteunen. (zie ook punt 3.6.3 Grafische Implementatie)

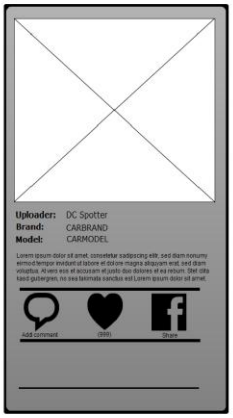
[11]**Wireframe**: meestal gebruikt in een website development omgeving, is een pagina of schematische scherm-blauwdruk, een visuele gids dat het 'skelet' van een website of applicatie voorstelt.

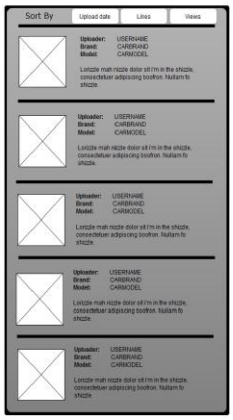
3.7.4. Wireframe











3.7.5. Grafische Implementatie

Onze applicatie moet op elk type Anroid toestel kunnen werken, we moeten dus rekening houden met verschillende schermresoluties.

Volgende resoluties zijn standaard op de verschillende type toestellen:

- QVGA (240x320)
- HVGA (320x480)
- WVGA800 (480x800)
- WVGA854 (480x854)
- WQVGA400 (240x400)
- WQVGA432 (240x432)

Android heeft een mechanisme voorzien om de schermresoluties te detecteren en hiervoor de bestpassende grafische componenten in te laden. Deze dienen in verschillende mappen te staan, opgedeeld per resolutie. De mapnamen zijn respectievelijk:

- drawable-hdpi
- drawable-mdpi
- drawable-ldpi

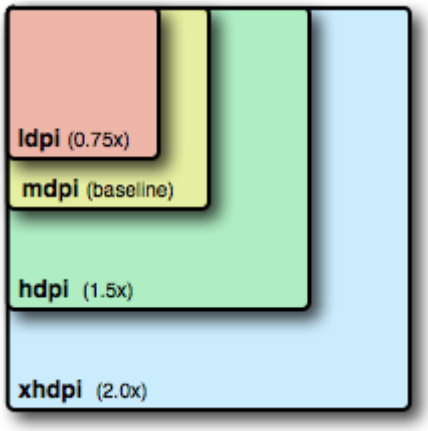
Onderstaande tabel toont de verschillende 'Configuratie kwalificaties' die gedefinieerd werden door Android:

Screen characteristic	**Qualifier**	**Description**
Size	`small`	Resources for *small* size screens.
	`normal`	Resources for *normal* size screens. (This is the baseline size.)
	`large`	Resources for *large* size screens.
	`xlarge`	Resources for *extra large* size screens.
Density	`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
	`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
	`hdpi`	Resources for high-density (*hdpi*) screens (~240dpi).
	`xhdpi`	Resources for extra high-density (*xhdpi*) screens (~320dpi).

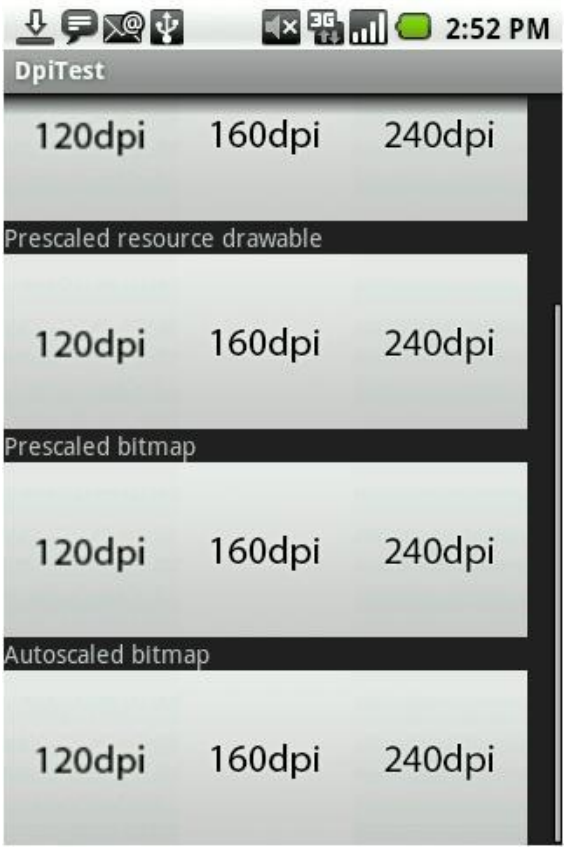
	`nodpi`	Resources for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density.
Orientation	`land`	Resources for screens in the landscape orientation (wide aspect ratio).
	`port`	Resources for screens in the portrait orientation (tall aspect ratio).
Aspect ratio	`long`	Resources for screens that have a significantly taller or wider aspect ratio (when in portrait or landscape orientation, respectively) than the baseline screen configuration.
	`notlong`	Resources for use screens that have an aspect ratio that is similar to the baseline screen configuration.

Wij ondersteunen zoals hierboven vermeld **small**, **normal** en **large** resolutie met respectievelijk lpdi, mdpi en hdpi als densiteit (dots per inch).

Onderstaande afbeelding links illustreert het verschil tussen de resoluties die evenredig zijn met de dichtheid (densiteit) van een afbeelding.



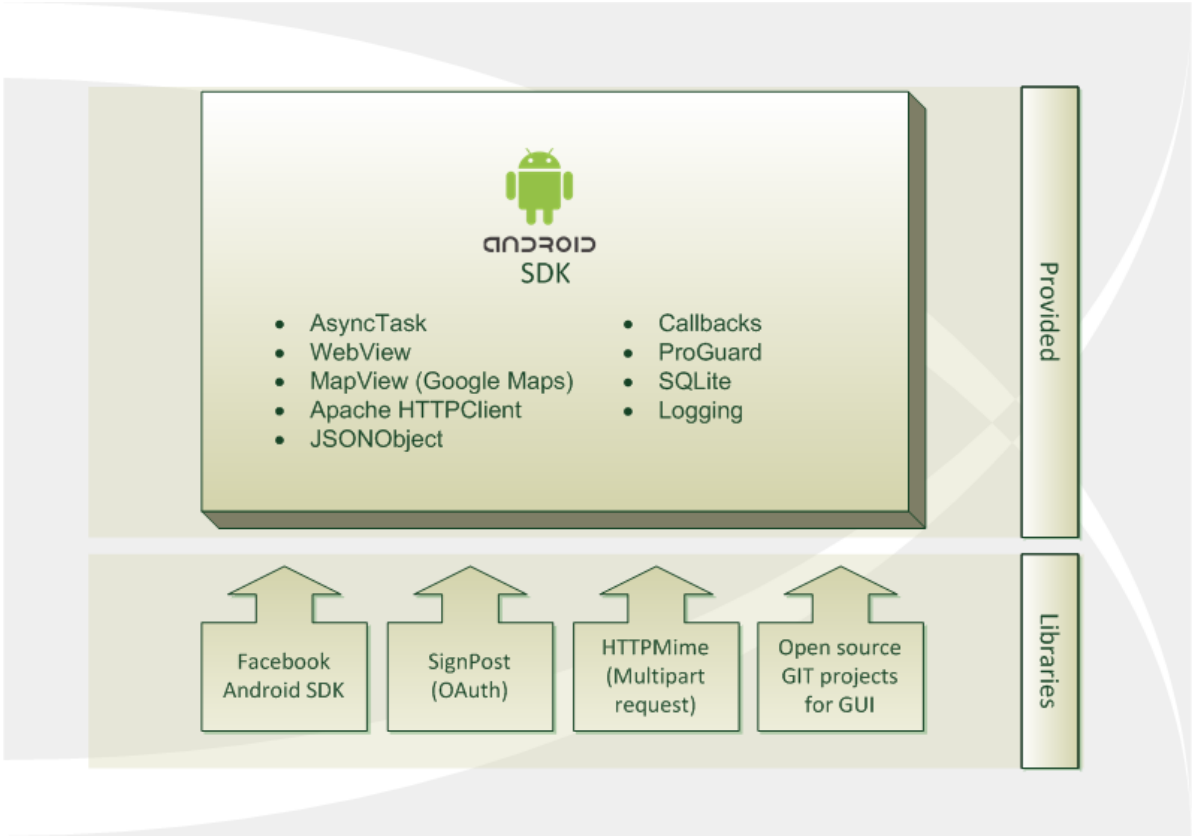
Afbeelding rechts illustreert het verschil in dpi 'dots per inch' op een Android toestel.



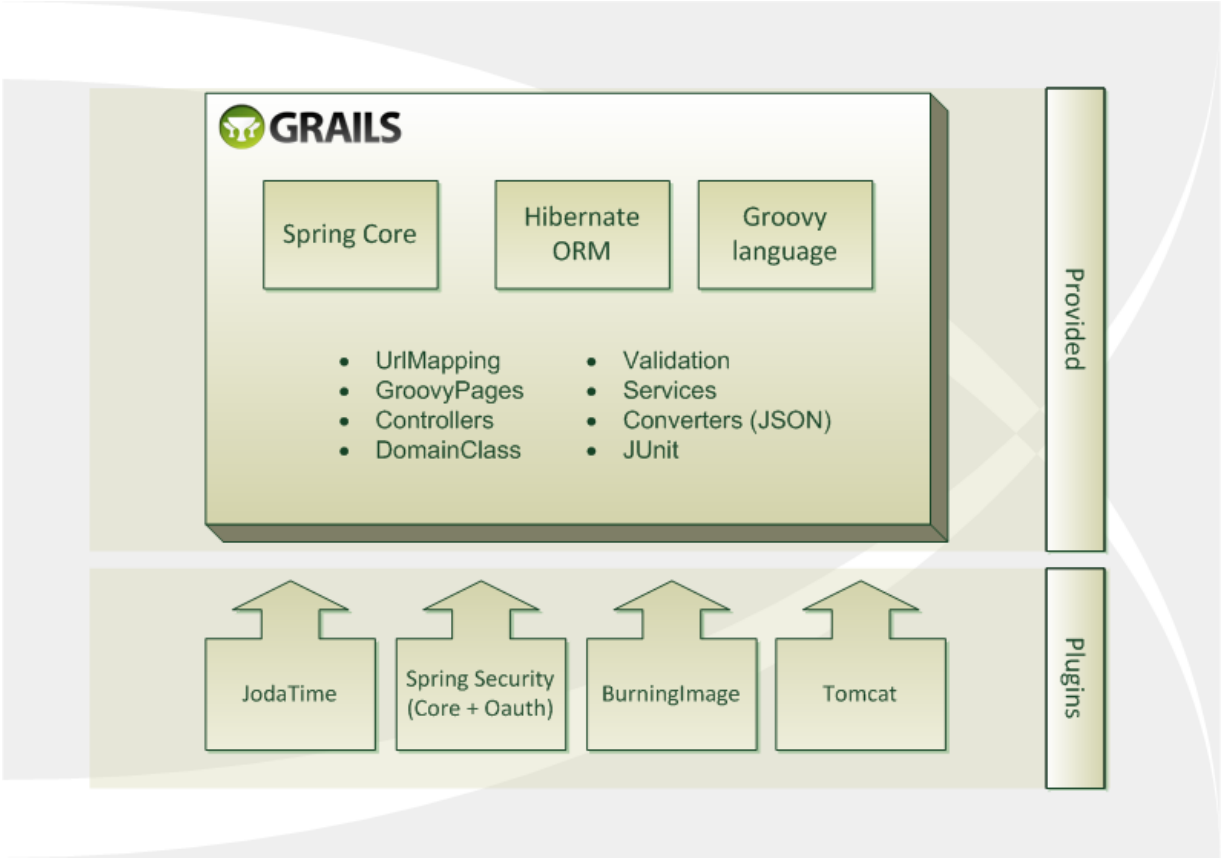
4. Ontwikkeling & Implementatie

4.1. Technology Stacks

4.1.1. Client



4.1.2. Server



4.2. Beveiligen van de REST interface

We gebruiken een interface om te communiceren met onze serverapplicatie, Daarom moet er genoeg controle zijn voor authenticatie en autorisatie.
Er zijn 2 dingen waar rekening mee gehouden moet worden:

- Wanneer iemand een nieuwe entry submit, moet de server-zijde weten wie deze post heeft gesubmit = **Authenticatie**.
- Wanneer iemand niet ingelogd is mag het niet mogelijk zijn om een nieuwe entry te submitten = **Autorisatie**.

Om hiervoor een oplossing te bieden gebruiken we een Usermodel aan de serverzijde. We gaan gebruikers laten registreren en laten inloggen.
Springsource biedt voor traditionele Java applicaties een framework aan dat leeft onder de naam "**Spring sercurity**". Deze biedt een quasi out-of-the-box oplossing om resources te beveiligen, het inlog mechanisme te koppelen aan een handige *securityService* en nog veel meer. Spring security is hedendaags zowat de defacto standaard om enterprise Java applicaties te voorzien van authenticatie en authorisatie.

Zoals eerder vermeld heeft Springsource het Grails framework ontwikkeld waardoor ze ook een Spring security plugin voor Grails beschikbaar maken. We gebruiken deze plugin dus om onze applicatie te beveiligen.
Aangezien deze security flow ook moet kunnen toegepast worden wanneer we met een android applicatie werken, dus niet enkel binnen onze server-applicatie, zijn er een aantal kenmerken aan onze "infrastructuur" waardoor we nog extra functionaliteiten nodig hebben.

We bieden een **REST interface** waarbij elke request onafhankelijk van elkaar moeten functioneren, dus moeten we bij elke request meegeven wie we zijn, tenminste voor de beveiligde resources.

We maken geen gebruik van 'Basic Authentication' waarbij gebruikersnaam en wachtwoord bij elke request worden meegeven.
Dit 'vervuilt' elke request, is onderschepbaar, vraagt om encryptie (HTTPS) en moet gebruikersnaam en wachtwoord steeds gekend zijn aan de cliëntzijde. De credentials zouden dus opgeslagen moeten worden op de telefoon. Wat als een gebruiker bijvoorbeeld zijn telefoon verliest? Aan Basic Authentication zijn heel wat risico's aan verbonden.

4.2.1. OAuth 1.0

OAuth is een protocol om een API te beveiligen die een oplossing biedt voor de zonet beschreven valkuilen. Het is een soort stramien dat je volgt om uiteindelijk een "**token**" of een sleutel als identificatie te hebben. Een token is een gegenereerde tekenreeks met letters en nummers dat uniek is, en gekoppeld is aan een 'secret'. Hiervoor wordt het **SHA-1 hashing algoritme** gebruikt in samenwerking met een gedeelde'secret'. Wij gebruiken de HMAC om die secret te kunnen genereren. Ons doel is om enkel deze token en secret mee te moeten geven aan de request die identificatie vereist.
Wij zullen gebruik maken van OAuth 1.0 volgens het 3-legged model. Er bestaat ook 2-legged, maar dan wordt er niet met aparte profielen per user gewerkt.

1. **Leg 1: Service provider:** De Grails server applicatie die protected resources beschikbaar maakt. Van deze service maken we gebruik om in ruil voor gebruikersnaam en wachtwoord een access token te verkrijgen.

2. **Leg 2: Service consumer:** Onze android applicatie die gebruik maakt van de service provider. Indien we later andere client-applicaties maken zullen we deze moeten registreren bij de service provider. Deze is momenteel statisch omdat we maar 1 consumer hebben. Bij service providers zoals Google Maps is het mogelijk om een API-key aan te vragen voor je eigen applicatie. Hierdoor registreer je een consumer applicatie. Onze Android consumer heeft dus ook een 'API key' die aan beide zijden gekend is. Enkel applicaties die een geldige API key hebben, zullen gebruik kunnen maken van onze OAuth login functionaliteit.

3. **Leg 3: User:** De gebruikers van de android applicatie en dus interacties kunnen uitvoeren die gekoppeld zijn aan hun profiel.

Andere grote API's zoals die van Twitter, Facebook maken ook gebruik van OAuth. Indien men zelf een applicatie zou ontwikkelen die gebruik maakt van deze API's om specifieke user interacties uitvoeren, zal men moeten inloggen op deze API's door middel van OAuth.

4.2.2. OAuth autorisatie stroom

Of beter, 'authorisation flow', hoe kan een user die geregistreerd is op onze applicatie een access token en secret verkrijgen? In onze Android applicatie wordt bij het opstarten nagegaan of er op de telefoon reeds een access token en secret aanwezig zijn, deze worden vervolgens geverifieerd op onze server. Als deze niet (meer) geldig zijn, of de gebruiker start voor de eerste keer de applicatie op zal hij doorgestuurd worden naar het login/registreer scherm dat onze server via een webpagina beschikbaar stelt.

Volgende stappen worden uitgevoerd om onze user een access token te geven:

1. **Een request token aanvragen.**

   Vooraleer de gebruiker wordt doorgestuurd naar de login/registreer pagina wordt er een request token aangevraagd. Die is van structuur gelijkaardig aan de access token. Er zijn een aantal request parameters die worden meegegeven zoals de consumerkey en timestamps:
   Token aanvraag:
   http://62.182.62.173:8080/spotterserver/oauth_request_token

   Antwoord Server:
   Token: d3267938-61cc-4261-823d-17e796132abe
   Secret:
   NgXe8aVd+R9CIUsyhFiZDC1riunJSWu4tzfjmc9n2CwotYEwcn
   VEIZ+L62Irh2g3ujO7lJrBeVE+wjyS76Pz812RfIJnjoLlp9GzZJyj0gE=

   Na deze token aanvragen verkrijgen we een request token die we tijdelijk nodig hebben om in te loggen en vervolgens toegang te verlenen aan onze applicatie.

2. **Autorisatie geven aan onze Android applicatie**

   Op dit moment logt de gebruiker in en wordt er gevraagd of hij de Android applicatie wil laten toestaan om zijn gegevens te kunnen raadplegen.

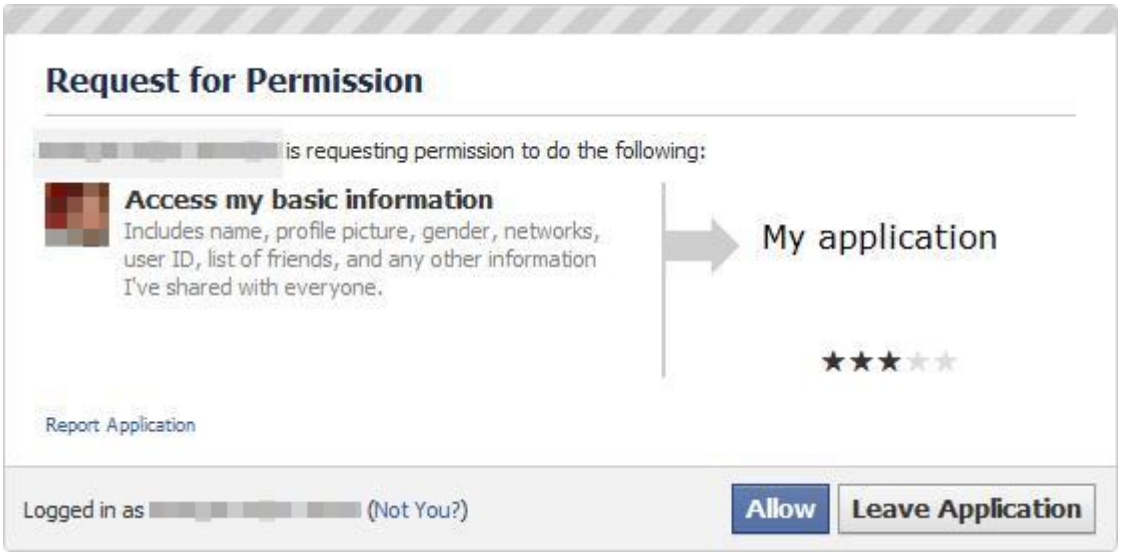
   http://62.182.62.173:8080/spotterserver/OAuth/authenticate_token?oauth_token=d3267938-61cc-4261-823d-17e796132abe

   Bij deze request wordt de request token van stap 1 als parameter meegeven. Als response krijgen de gebruiker de loginpagina te zien waar

de user zijn username en passwoord kan ingeven. Mocht de gebruiker nog geen account hebben, kan hij er ook eerst een aanmaken door te registreren.

Voor het registreren en het inloggen wordt er vanuit onze Android applicatie een interne browser opgestart om de login –of registratiegegevens te laten invullen. Dit is een deel van de filosofie van OAuth. We geven de gegevens rechtstreeks aan de Service provider, dus niet via een eigen login form.
Verder slaan we de stap over om na het inloggen ook expliciet toegang vragen om de android applicatie te vertrouwen. We willen immers zo weinig mogelijk tijd vergen van de gebruiker om aan de slag te gaan met de applicatie. We hebben immers maar 1 service consumer.

API's zoals Facebook hebben honderden of duizenden consumers, dus daarom voeren zij volgende stap uit:



Als response krijgt de gebruiker vervolgens zijn request token terug, die we door gans het autorisatie proces nodig hebben, en een 'verifier'. De verifier is een korte tekenreeks die opnieuw naar de server wordt doorgespeeld als er een access token wordt opgevraagd.

http://62.182.62.173:8080/spotterserver/?oauth_token=null&oauth_verifier=3PKSBR

3. Vraag access token aan in ruil voor request token en verifier

Vanaf nu heeft de gebruiker alle gegevens om zijn access token aan te vragen. De server heeft op dit moment de gebruiker geïdentificeerd aan

de hand van de access token hangt, en met behulp van de verifier weten we dat de gebruiker toestemming heeft gegeven.

http://62.182.62.173:8080/spotterserver/oauth_access_token

Nu is het voldoende om de verifier mee te geven. Als resultaat geeft de server ons het volgende terug:
Access token: be01d1be-bd37-4307-90b7-1dac81418df7
Token secret:
8Q1SbeycTJdQI8VtbLwL01xAVgp1WgbGzXUAhk9ir5+dp/FBMp5TjQimzHsS
95DTVvc6IBudTdJ4Tm9Yk44rispdiIQuGLVKw5gvz8ME6X8=

Deze gegevens worden uiteindelijk op 2 plaatsen bewaard. Op de Android telefoon zelf en in de database bij onze users. De **access tokens** zijn ingesteld voor **een levensduur van 1 jaar**. Na 1 jaar gebruik van onze applicatie op eenzelfde toestel, zonder uit te loggen, zal een gebruiker dus opnieuw moeten inloggen om een nieuwe access token te verkrijgen.
Dit lijkt lang, maar is het niet, dit is juist een deel van de veiligheid van OAuth, de gebruiker moet in principe maar 1 per jaar gebruikersnaam en wachtwoord opnieuw naar onze server verzenden.
Bijvoorbeeld voor Twitter, een bekende applicatie die ook OAuth gebruikt, zijn de access tokens oneindig geldig.

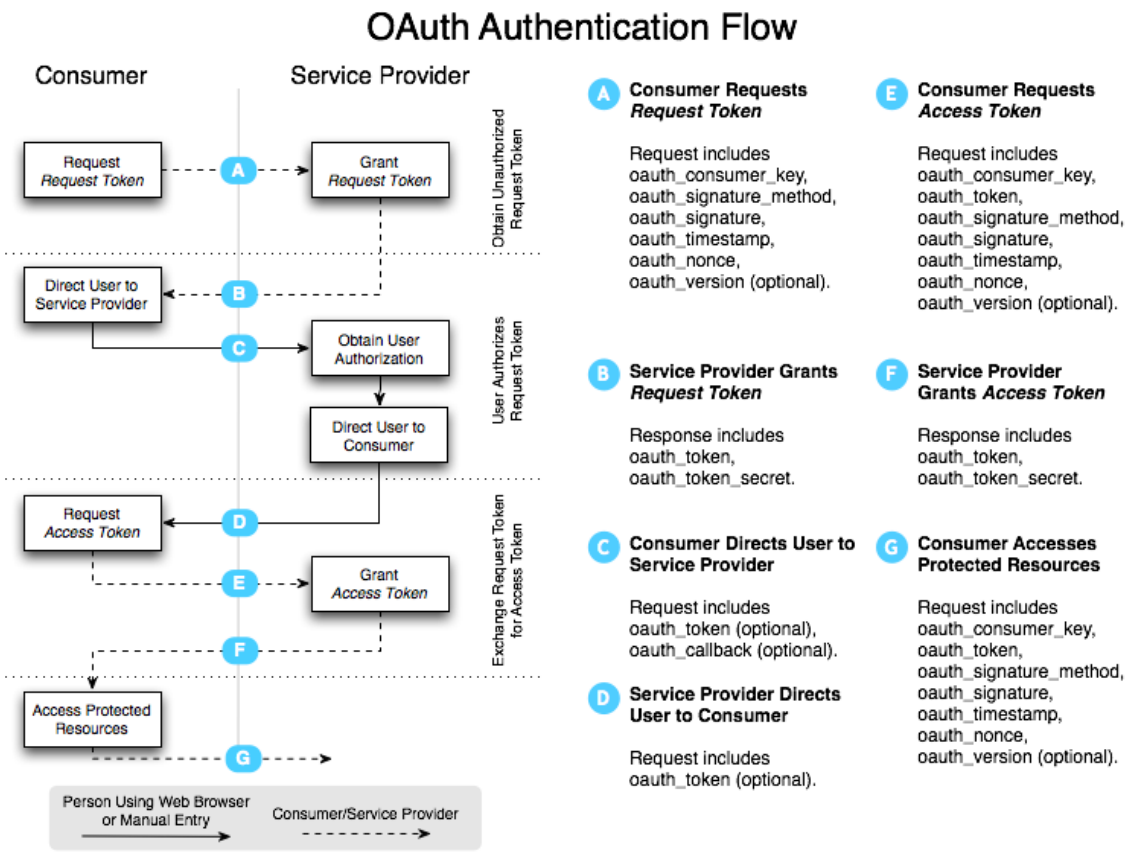
4. Requests maken met de authentication key

Om een beschermde resource te raadplegen zal de user altijd zijn authentication key moeten gebruiken.
Bijvoorbeeld:
Een user probeert de volgende resource op te vragen van onze server: "/entries/me". Door deze resource op te vragen zal men alle geposte entries terugkrijgen van de ingelogde user. Zoals eerder vermeld zijn, REST calls onafhankelijk van mekaar, dus zal de Authorization Header van de request onze access token en secret moeten toegevoegd worden.

GET http://62.182.62.173:8080/spotterserver/entries/me
Authorization: OAuth
realm="spotterserver",
oauth_consumer_key="dj0yJmk9nM9Y29uc3VtZXJzZWNyZXQmeD1lMg--",
oauth_nonce="24829.2331",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="1219450170",

oauth_token="A%3DuqkiebGpiTJl7ThQxU.jDXXaETYyfEy3xAKPyoavokwOOcZcz8Xs_l1Nvnl._KmCEVCeLkxxT1Y6BgRqf5f98sQWHklBM_anetveR7okK_M_5XEmQ1_1reo3UgKQULT_dQT8Gao3.Rrgz5rJxgmnYrhdWWdfgTdMQVzpbJT2aGkz59NTK1O8yXVE1EvZUCqju7WiFYu.WHNEw.9TWq3g--",
oauth_version="1.0",
oauth_signature="O2AQipLITO0aYHKZc9266RzC94%3D"

Als we de resource ophalen zonder een Authentication header, zouden we als response een HTTP Error code 401 Unauthorized krijgen.



Bron: http://dret.net/lectures/mobapp-spring10/img/oauth-diagram.png

4.2.3. OAuth in Java en dus in Android

Bovenstaande theorie moet ook omgezet worden in praktijk. Gelukkig kunnen we gebruik maken van de modulariteit van Java waardoor we bestaande libraries kunnen gaan gebruiken.

Springsource is momenteel vrij intensief bezig met het ontwikkelen van Spring Android http://www.springsource.org/spring-android. Dit is een library die een aantal toevoegingen doet aan de **Android SDK**[12]. Hierbij beloven ze betere logging systemen en een REST-template die ze uit het Spring Web Services project willen migreren. De REST-template klasse maakt het ons als developers makkelijker om REST interfaces aan te spreken. Doch ondersteunt deze momenteel nog geen OAuth dus konden we hier spijtig genoeg geen gebruik van maken. Er is trouwens nog geen final release van uitgekomen, dus er is nog wat werk aan de winkel!

Toen zijn we uitgekomen bij de Signpost library http://code.google.com/p/oauth-signpost/ Deze maakt intern dan weer gebruik van de apache commons http libraries. Signpost bied ons bijvoorbeeld methodes aan om request tokens aan te vragen, authorization headers te setten op onze request etc.

Om bijvoorbeeld een access token op te vragen zoals in stap 3 uitgelegd kunnen we gebruik maken van:

```
CommonsHttpOAuthProvider.retrieveAccessToken(consumer,
oauth_verifier);
```

Of om een GET request te authenticeren kunnen we eenvoudig de sign methode oproepen

```
CommonsHttpOAuthConsumer.sign(p);
```

Hierdoor wordt er voor ons de authorizatie header opgevuld met access token en secret die opgeslagen zijn in de database.

4.2.4. OAuth provider

We hebben ervoor gekozen om onze eigen OAuth provider te maken. We willen namelijk een eigen userdatabase waardoor onze gebruikers zich eerst moeten registreren op onze applicatie. Een alternatief is om via OpenId gebruikers te laten inloggen via Facebook of Google login.

Een eigen Oauth provider maken lijkt opzich makkelijk. Spring security Oauth heeft hiervoor een aantal klasses beschikbaar, dus lijkt de klus snel geklaard door enkel wat services aan elkaar te hangen. Spijtig genoeg was hier niet veel documentatie over te vinden.

[12]**SDK**: **S**oftware **D**evelopment **K**it. Geheel van tools en klasses die developers kunnen gebruiken om software te maken voor een bepaald platform of framework.

Gelukkig heeft Burt Beckwith, een van de lead developers van het Grails framework, enige tijd geleden een plugin gemaakt voor een OAuth provider. Deze maakte wel nog gebruik van een (hele) oude versie van de OAuth library. We konden deze plugin dan als basis nemen en aanpassen zodat die met de nieuwe libraries konden werken.
Laatste puntje was dat er enkel een "In-memory" versie was om de access tokens op te slaan. Elke herstart van de server, of re-deployment zou als gevolg hebben dat alle clients weer zouden moeten inloggen. Hun tokens waren immers niet meer gekend. Oplossing was dus om een eigen implementatie te maken die deze tokens in de database persisteert.
Verder hebben we nog de levensduur van de tokens op 1 jaar gezet, zodat gebruikers in het beste geval maar 1 keer moeten inloggen.

4.2.5. HTTP Risico

Dataverkeer via klassieke HTTP is niet geëncrypteerd. Iedereen kan dus het dataverkeer onderscheppen en de gegevens letterlijk 'aflezen'.
Hierdoor kunnen ze ook onze unieke access token en andere data onderscheppen en bijvoorbeeld van deze access token gebruik maken om zich als iemand anders voor te doen.
'Man in the middle attack'
Oplossing: Gebruik maken van HTTPS verbinding:

> Het protocol **HTTPS** staat voor HyperText Transfer Protocol Secure.

HTTPS is een uitbreiding op het HTTP-protocol met als doel een veilige uitwisseling van gegevens. Bij gebruik van HTTPS worden de gegevens versleuteld, waardoor het voor een buitenstaander, bijvoorbeeld iemand die afluistert, onmogelijk zou moeten zijn om te weten welke gegevens verstuurd worden.

➔ Dit doen we nu niet omdat we momenteel nog afhankelijk zijn van de niet-definitieve webserver (Faros NV). De HTTPS server-configuratie is server- en platformafhankelijk. Dit vereist een SSL-verbinding (Secure Sockets Layer) met een SSL-certifcaat. De keuze zal bij ons liggen bij het Self signed SLL certificaat (kosten).

➔ Grails is compatibel met HTTPS, daarom onze keuze.

4.3. Java Afbeeldingverwerker

4.3.1. Probleemstelling

Android toestellen hebben allemaal x megapixel camera's met een bepaalde resolutie. De foto's die genomen worden zijn vervolgens (te) groot om allemaal te stockeren op onze webserver. We willen dus optimaal gebruik maken van de beschikbare schijfruimte.

4.3.2. Oplossing

4.3.2.1. Android

Wanneer de gebruiker een afbeelding selecteert om te uploaden, wordt de afbeelding automatisch door Android geoptimaliseerd naar een lagere kwaliteit, namelijk JPEG formaat 80%.

4.3.2.2. Java

We maken gebruik van de Grails-plugin **Burning Image**[13]. Deze plugin biedt ons een Spring-service aan die we via dependency injection[14] gebruiken.

De klasse die we hiervoor aangemaakt hebben is onze **PictureService klasse**.

Deze zal ervoor zorgen dat de geüploade afbeelding, zodra deze op de server staat, gecomprimeerd wordt naar volgende 2 formaten:

1. Groot formaat: JPEG **800*600** (maximum)
2. Klein (thumbnail) formaat: JPEG **96*96** (vierkant)

[13] **Burning Image**: http://code.google.com/p/burningimage/

[14] **Dependency Injection** of inversion of control is een methode om de verantwoordelijkheid van de relaties tussen instanties te centraliseren naar één systeem. Dit wordt gecoördineerd door het Spring Core framework.

4.4. Caching

4.4.1. Probleemstelling

Het moment dat een afbeelding ingeladen in de **GUI**[15] uit het scherm verdwijnt, dan wordt deze uit het geheugen gewist. Het gevolg hiervan is dat bij het scrollen door (bvb onze **listview**[16]) de applicatie moet elke afbeelding opnieuw worden gedownload van de webserver via een http request.

→ de Adroid SDK bevat deze functionaliteit niet.

4.4.2. Oplossing: CACHE

4.4.2.1. Wat is Cache?

Een cache is een opslagplaats waarin gegevens tijdelijk worden opgeslagen om sneller toegang tot deze data mogelijk te maken.

Het opslaan van gegevens op een sneller medium om sneller toegang tot deze data te hebben wordt **caching** genoemd. De term *cache* wordt meestal gebruikt voor zowel de data die gecached worden als voor de opslagplaats waar deze data gecached worden

4.4.2.2. Capaciteitsprobleem

Elk Android toestel heeft een bepaalde geheugen capaciteit (vluchtig geheugen of 'RAM geheugen')

Met als gevolg dat de Android applicatie maar een klein toegewezen deeltje van dit vluchtig geheugen in beslag neemt. De grootte hiervan is dus afhankelijk van het merk/type toestel in gebruik. Meestal gaat dit om 16 of 24 Mb.

Aangezien de cache niet oneindig groot is zal de applicatie na bijvoorbeeld 10 minuten in gebruik al het vluchtige geheugen innemen en resuleert dit in een overvol geheugen en zal bij gevolg de applicatie crashen.

[15]**GUI**: **G**raphical **U**ser **I**nterface, soms ook UI genoemd. Dit is het geheel van schermen, knoppen en invoerelementen dat de gebruiker op het scherm te zien krijgt

[16]**Listview**: Een component in Android die we kunnen uitbreiden om een scrollable lijst van rijen te laten zien.

4.4.2.3. Java HashMap Klasse

Oplossing hiervoor is de Java klasse HashMap<String, SoftReference<Bitmap>>

De bitmap in de SoftReference zetten (= zogenaamde 'Wrapper klasse'.)
Op die manier vertellen we aan Java dat onze bitmaps vatbaar zijn voor Garbage Collection.
Als het gebruikte geheugen nu te groot wordt, zal Java de objecten die niet meer gebruikt worden uit het geheugen verwijderen. Deze JDK-functionaliteit noemt: 'Garbage collection'.

4.4.3. Java Implementatie Caching

Onze applicatie maakt gebruik van een 'key-value-pair'. Dit bekomt men door de Map interface te implementeren.

➔ Als 'key' wordt de image-url bijgehouden (http).

➔ De value is de opgeslagen byte array in cache.

Als er naar onder wordt gescrolled in de listview gaat onze applicatie eerst kijken of de 'key' value reeds aanwezig is, zoja wordt de image uit het geheugen gehaald. (geen vertraging of laadtijd)
Indien het niet is wordt de afbeelding gedownload en wordt er een nieuwe **key-value-pair** aangemaakt om vervolgens in de cache te zetten.

4.5. AsyncTask: Synchroon Takenbeheer

4.5.1. Wat?

AsyncTask Is vergelijkbaar met een klassieke Java applicatie die multi-threaded is.
AsyncTask gaat het mogelijk maken om de processen parallel te laten lopen zonder de applicatie te laten 'hangen' of wachten.

Bijvoorbeeld tijdens het upload image proces wordt:

1. Progress bar verhoogd door de main thread
2. Gelijktijdig een http-request uitgevoerd om de effectieve foto-upload af te ronden in een andere thread.

Met andere woorden, zonder deze AsyncTask zou de pogressbar niet gelijktijdig kunnen verhogen samen met de foto-upload.

4.5.2. Java Implementatie

Onze klasse *DonwloadImageTask* erft over van ***AsyncTask*** die vereist dat je 3 methodes implementeert namelijk:
onPreExecute: GUI aanpassingen voordat het process gestart wordt
(bijvoorbeeld image: een standaard image hierin meegeven. Mocht er iets misgaan bij het ophalen kunnen we deze default image tonen)
doInBackground: De uiteindelijke asynchroon proces dat wordt uitgevoerd.
(voorbeeld: http request om image te downloaden)
onPostExecute: Hierin wordt het resultaat op het scherm getoond)
(voorbeeld: image in cache zetten indien nodig en de image in de gui tonen)

Dankzij AsyncTask gaat Android de threads zelf managen wat bij klassieke Java-applicatie niet het geval is.

4.6. Application Programming Interface (API)

Hieronder een overzicht van de API die de server beschikbaar maakt. Response wordt in JSON formaat teruggegeven.

Resource	HTTP Method	Secured	Statuscodes	filters	Omschrijving
/entries	GET	Nee	200	page lng1,lat1,lng2,lat2 (zie hoofdstuk X.Y)	Overzicht van alle entries, gesorteerd op creationDate
/entries	POST	Ja	401 400 200	sort	Submit een nieuwe entry op basis van parameters
/entries/{id}	GET	Nee	200 404		Specifiek entry op basis van id
/entries/{id}	DELETE	Ja	200 403 404		Verwijder een entry waarbij de eigenaar de ingelogde gebruiker is
/entries/{id}/liked	GET	Ja	200 401 404		Is de entry met meegegeven id geliked door de ingelogde gebruiker?
/entries/{id}/likes	POST	Ja	401 400 (reeds geliked) 404 (entry niet		“Like” een entry

			gevonden) 200		
/entries/{id}/likes	DELETE	Ja	401 200 400 (niet geliked) 404 (like niet gevonden voor entry)		Ongedaan maken van het “liken” van een entry
/entries/{id}/liked	GET	Ja	401 200		Heeft de ingelogde user de entry geliked?
/entries/{id}/flags	POST	Ja	401 400 (reeds geflagged) 404 (entry niet gevonden) 200		“Flag” een entry
/entries/{id}/flags	DELETE	Ja	401 200 400 (niet geliked) 404 (flag niet gevonden voor entry)		Ongedaan maken van het “flaggen” van een entry
/entries/{id}/flagged	GET	Ja	401 200		Heeft de ingelogde user de entry geliked?
/entries/me	GET	Ja	401 200	page sort	Overzicht van alle entries die gepost zijn door de ingelogde gebruiker
/entries/{id}/comments	GET	Nee	200	page	Overzicht van comments voor een

			404		bepaalde entry
/entries/{id}/comments	POST	Ja	200 400 401 404		Plaats een nieuwe comment
/pictures/thumb/{id}.jpg	GET	Nee	200 404		Toont afbeelding in klein formaat (96x96) voor bepaalde entry
/pictures/full/{id}.jpg	GET	Nee	200 404		Toont afbeelding in groot formaat (maximum 800x600) voor bepaalde entry

4.6.1. HTTP Status codes:

200 OK, aanvraag is verwerkt

400 Bad request, onjuiste request (bvb geen foto meegegeven bij het posten van een nieuwe entry, validatie errors)

401 user is niet ingelogd en voor de operatie is een ingelogde gebruiker nodig

403 forbidden: user probeert een entry te deleten die niet tot hem behoort

404 de resource is niet gevonden, bvb entry met id 999 bestaat niet

500 internal server error (exception)

4.6.2. Paginering:

Natuurlijk willen we niet alle resultaten inladen wanneer we de lijst van alle entries opvragen. Daarom hebben we paginering ingebouwd waardoor er makkelijk via een "page" parameter de juiste pagina kan worden opgevraagd. Eén pagina bevat 10 entries.

Voorbeeld: /entries?page=3

4.6.3. Sortering:

Ook willen we de data op verschillende manieren kunnen sorteren: we hebben 3 opties:

- **Date**: Sorteren volgens aanmaakdatum met de meest recente bovenaan
- **Comments**: Entries met de meeste comments bovenaan
- **Likes**: Entries met de meeste likes bovenaan.

  Voorbeeld: /entries?sort=likes

4.6.4. MapView

Wanneer de MapView wordt geconsulteerd, zullen niet alle **spots**[17] worden ingeladen maar enkel diegene die binnen het bereik van het scherm vallen. Als de map bijvoorbeeld ingezoomd staat op de stad Leuven, willen we enkel bullets tonen van spots die daar in de buurt staan. In praktijk zullen we onze applicatie altijd 2 coordinaten moeten meegeven om de nabijliggende spots te kunnen identificiëren.

- Punt linksboven MapView scherm
- Punt rechtsonder Mapview scherm

[17] **Spot**: zie 'Entry'

Deze parameters worden doorgegeven via de lat (latitude) en lng (longitude) filters (lengte-en breedtegraad), vervolgens worden de nabijliggende spot geïdentificeerd.

Voorbeeld:/entries?lat1=40.656&lng1=6.9465&lat2=41.369&lng2=7.649

4.6.5. MultipartRequest om foto's te uploaden

Als er een nieuwe spot wordt aangemaakt, de gebruiker heeft alle gegevens ingevuld, een foto geselecteerd met locatie en vervolgens op de submit knop gedrukt, zullen er meerdere types van data worden doorgestuurd via één request. Zowel de informatie van de auto, als een foto.

Om dit te verwezelijken maken we gebruik van een MultipartRequest, waarbij we eenvoudig de verschillende delen van de request kunnen opvullen. Gelukkig werkt deze request ook met ons OAuth systeem en kunnen we deze request gaan signen. Deze klasse erft namelijk ook over van HTTPEntity.

We hebben de library HTTPMime 4.1.1 toegevoegd aan ons android project om multipartrequests te maken.

Wat je vaak nog ziet bij fileuploaders is dat je een progressiebalk te zien krijgt. Stel dat de gebruiker via een trage mobiele internetverbinding werkt, kan dit proces wel wat tijd in beslag nemen. Aangezien mobiele applicaties klein, simpel, makkelijk, maar ook aantrekkelijk zouden moeten zijn, hebben we besloten om dit te implementeren.

Simpelweg hebben we tijdens het uploaden het aantal reeds verzonden bytes nodig, en delen we dit door het totale aantal bytes van de request. Zo bekomen we het percentage dat reeds geuploaded is.

- De totale grootte van de foto kunnen we achterhalen door de *size()* methode op te roepen van de ByteArrayInputStream waarin we de gecompresseerde foto opslaan.
- Om tijdens de upload de bytes te kennen die op dat moment zijn verzonden moeten we een wrapperclass maken rond onze MultipartEntity. Deze zal een listener meekrijgen via de constructor zodat we callback kunnen meegeven wanneer er wordt geupload. Verder overriden we de methode *write* om zelf het aantal bytes te gaan bijhouden in een lokale variable, en voeren we onze callback uit met die lokale variabele als parameter. Nu kunnen we onze *progressDialog* gaan vermeerderen vanuit de *main-thread*.

Een "simpele" *progressDialog* tonen, is dus niet zo vanzelfsprekend dan u op het eerste zicht zou verwachten.

4.7. Publiceren op Facebook

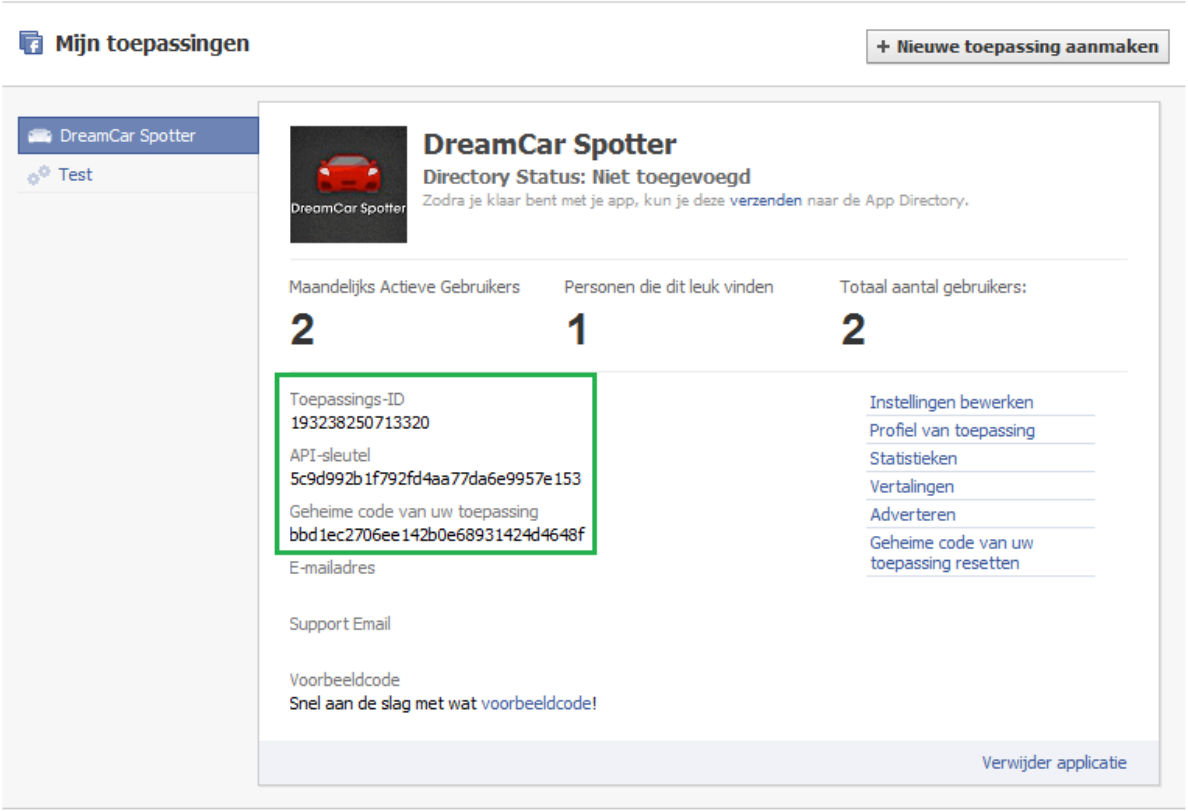
4.7.1. Algemeen

Facebook werkt, zoals eerder in dit document vermeld, ook met OAuth. Een gebruiker die geregistreerd is bij Facebook zal dus individueel toestemming moeten geven aan de dreamcar spotter applicatie om te kunnen fungeren als consumer van zijn Facebook-profiel. Op die manier kan onze applicatie een post plaatsen op de gebruiker zijn Facebook-profiel.

4.7.2. Registratie van applicatie

Om extern op Facebook te publiceren moet onze applicatie eerst bekend zijn voor Facebook, dit kan door een Facebook-app(lication) pagina aan te maken op www.developers.facebook.com.

DreamCar Spotter wordt dus een consumer van Facebook en zal dus ook een access token & secret toegekend krijgen om juist geïdentificeerd te kunnen worden. Hieronder een voorbeeld van de toepassingsbeheer pagina op developers.facebook.com, groen aangeduid de Facebook OAuth gegevens.



4.7.3. Applicatie pagina

Onze applicatie heeft, vanaf registratie, ook een aparte applicatiepagina van Facebook toegewezen gekregen die Facebook gebruikers leuk kunnen vinden (fan worden, volgers) of het mogelijk maakt om, als beheerders, volgers op de hoogte te houden van bijvoorbeeld verdere ontwikkelingen van onze applicatie.
Deze zal, indien officieel in de Facebook app directory opgenomen, beschikbaar zijn op: http://apps.facebook.com/dreamcarspotter

Dit is een handige manier om gebuikers-feedback te krijgen, plus als beheerder kan je heel wat gebruikersstatistieken opvragen.

Voorbeeld van onze DreamCar Spotter Facebook-pagina:

4.8. Testen

Om goede software te schrijven zou je testen moeten schrijven voor de logica die je hebt geïmplementeerd. Het is onvermijdelijk dat al de code die je maakt, in eerste instantie perfect werken, en zonder bugs. Iedereen maakt fouten! Hiervoor gaan we dus testen maken. Ook kunnen deze testen dan later als controlemechanisme werken om na te gaan of bestaande functionaliteiten nog steeds blijven werken. Dit noemen we regressie.

Testmethodes maken is simpelweg een reeks klassen aanmaken (liefst met suffix "Test") en hierin methodes voorzien die je concrete klasses gaan oproepen. Je voedt deze methodes met testdata, en gaat na of het resultaat wat je terugkrijgt wel overeenkomt met wat je verwacht.
Er bestaan verschillende soorten testen, namelijk:

- **Unit testen:**
  Zullen kleine "units" van code testen. En dit binnen 1 laag (n-lagen model).
  Bijvoorbeeld: berekenen van aantal dagen tussen 2 datum.

- **Integratie testen:**
  Deze testen gaan over meerdere lagen. Bijvoorbeeld van controller, tot service, tot database. Je test dus meer de integratie van lagen, en de hele "ketting" van methodes die worden aangeroepen.

In ons project hebben we zo goed als geen "bereken" logica. Hiervoor gebruiken we immers zoveel mogelijk bestaande code uit libraries.
Aangezien onze Grails applicatie vooral een CRUD (Create-Read-Update-Delete) applicatie is die via een REST interface wordt aangeboden, gaan we integratietesten maken. Hiermee testen we

- URL mappings van URI naar controller en methodes
- Security
- CRUD
- Responses

We hebben momenteel 22 integratietesten die vooral als regressietesten zullen gebruikt worden.
Om in Grails onze testen te laten runnen, voeren we het commando **grails test-app** uit.

```
-------------------------------------------------------
Running 22 integration tests...
Running test be.faros.android.controllers.EntryControllerTests...PASSED
Running test be.faros.android.controllers.UserControllerTests...PASSED
Running test be.faros.android.services.PointServiceTests...PASSED
Tests Completed in 6255ms ...
-------------------------------------------------------
Tests passed: 22
Tests failed: 0
-------------------------------------------------------
```

Verder krijgen we een rapportje in html-vorm met een overzicht van alle testen:

Unit Test Results.

Designed for use with JUnit and Ant.

Class be.faros.android.controllers.EntryControllerTests

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EntryControllerTests	12	0	0	4.439	2011-05-05T11:33:59	BZ12XNVISTA

Tests

Name	Status	Type	Time(s)
testGETEntries	Success		2.741
testGETEntriesEmpty	Success		0.039
testGETEntriesID	Success		0.080
testPOSTLike	Success		0.745
testPOSTFlag	Success		0.188
testGETEntriesUSER	Success		0.191
testPOSTMultipart	Success		0.114
testPOSTEntryUnAuth	Success		0.033
testDELETEGoodAuth	Success		0.077
testDELETENoAuth	Success		0.045
testDELETEBadId	Success		0.071
testDELETEBadAuth	Success		0.078

Properties »
System.out »
System.err »

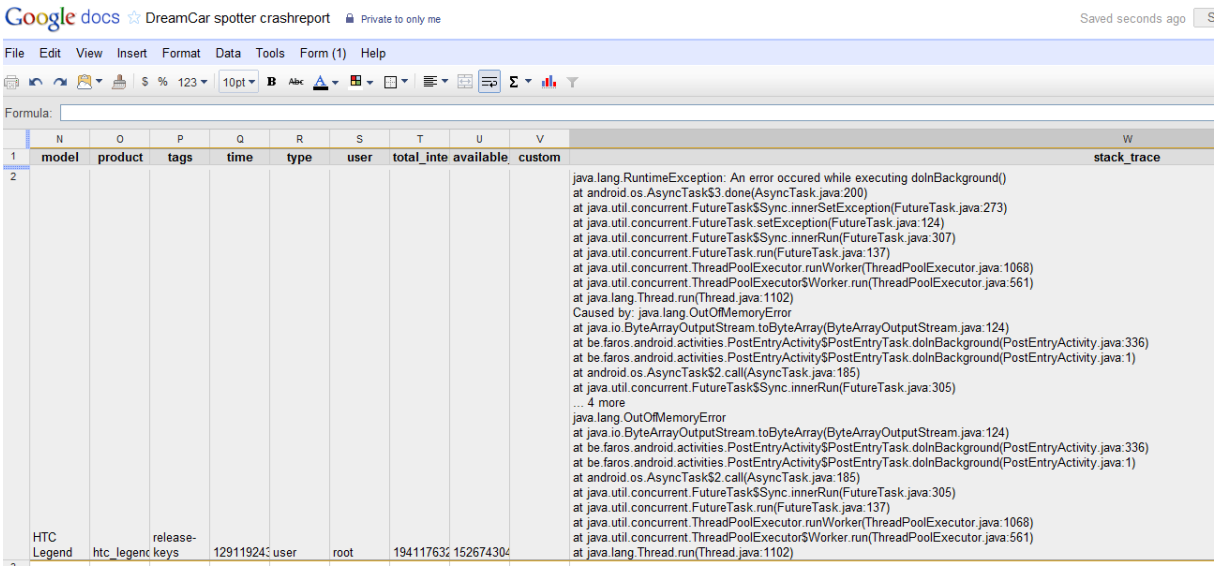
4.9. Crash reports

4.9.1. Algemeen

Het is natuurlijk altijd mogelijk dat onze applicatie crasht omdat het quasi onmogelijk is om met elk type Android-toestel te testen. Dit omdat elk toestel specifieke hardware configuraties heeft.
We zouden graag melding krijgen wanneer dit voorvalt, daarom besloten we Crash reports in te voeren. Hierdoor kunnen we heel gedetailleerde informatie verkrijgen over het type toestel (hardware & software) met daarbij de **Stacktrace**[18] als omschrijving van het probleem.

4.9.2. Application Crash Report for Android

'ACRA' is een opensource framework dat deze functionaliteit beschikbaar maakt voor Android applicaties.
ACRA zorgt ervoor dat de stacktrace en metadata automatisch in een rapport zet en dit beschikbaar stelt op Google-docs.



Dit is een voorbeeldje van een stacktrace die veroorzaakt was door een OutOfMemoryError door een HTC-legend toestel.

[18] **Stacktrace** is de tekstuele weergave van een momentopname tijdens het uitvoeringsproces van een Java applicatie. Wanneer er een RunTimeException of een niet opgevangen Checked Exception optreedt, wordt er automatisch een stacktrace gegenereerd.

5. Conclusie

5.1. Eindconclusie

De ontwikkeling van deze applicatie heeft onze verwachtingen overtroffen.
Het was heel tof om nieuwe technologieën te ontdekken en toe te passen.

We vinden dat we toch wel fier mogen zijn op het resultaat, om op een relatief korte tijdspanne, deze gebruiksklare mobiele applicatie te kunnen ontwikkelen.
Ook onze goede samenwerking en verstandshouding heeft hier toe bijgedragen.

We zijn helemaal niet uit op een nieuw facebook-imperium op te richten maar we geloven wel dat er genoeg doelpubliek zodat onze applicatie populair zou kunnen worden.

"Keep on spotting those cars, spotters!"

5.2. Toekomstplannen

- Betaalbare hosting, eventueel onder de vorm van **Cloud hosting**[19]
- Publicatie Android market in 2 versies:
  - Gratis versie met reclame banner
  - Betalende versie
- Website ook via internet toegankelijk maken (www.dreamcarspotter.com)
- SSL implementatie
- Onze eigen auto kunnen uploaden als DreamCar spotter waardig object

[19]**Cloud hosting**: een nieuwe vorm van hosting op geclusterde (aan elkaar gekoppelde) servers waardoor een grote schaalbaarheid ontstaat. De voordelen zijn betere beschikbaarheid, meer betrouwbaar en hogere snelheid.

6. Bronnen

6.1. Grails & REST:

http://www.grails.org/

http://en.wikipedia.org/wiki/Representational_State_Transfer

http://static.springsource.org/spring-security/site/

http://bmocanu.ro/coding/303/oauth-2legged-model-with-spring-security/

http://cscarioni.blogspot.com/2010/08/simple-introduction-to-oauth-protocol.html

http://blogs.bytecode.com.au/glen/2010/01/15/hacking-custom-authentication-providers-with-grails-spring-security.html

http://code.google.com/p/oauth-signpost/

http://awalkingcity.com/blog/2008/03/13/android-and-fire-eagle-oauth-and-java/

http://www.peej.co.uk/articles/restfully-delicious.html

http://oauth.net/documentation/getting-started/

http://static.springsource.org/spring-security/oauth/tutorial.html

https://github.com/grails-plugins/grails-spring-security-oauth-provider

http://www.software-innovators.nl/2009/05/19/twitter-veilig-met-oauth/

http://lukasz.szmit.eu/2010/08/working-with-google-docs-and-oauth-on_20.html

http://code.google.com/p/burningimage/

http://www.intelligrape.com/blog/2010/05/13/create-json-object-using-grails-converter-only-selective-fields-from-lists-of-objects/

http://developer.yahoo.com/oauth/guide/oauth-auth-flow.html

http://www.dzone.com/links/r/implementing_error_handling_in_restful_webservices.html

http://blog.springsource.com/2011/04/12/one-step-deployment-with-grails-and-cloud-foundry/

http://manbuildswebsite.com/2010/02/15/rendering-json-in-grails-part-3-customise-your-json-with-object-marshallers/

http://tools.ietf.org/html/rfc5849

http://joda-time.sourceforge.net/

http://code.google.com/intl/nl/apis/maps/documentation/javascript/

http://www.hibernate.org/docs

http://www.junit.org/

http://docs.jboss.org/hibernate/core/3.3/reference/en/html/querycriteria.html

http://www.cloudfoundry.com/

http://blog.springsource.com/2011/04/12/one-step-deployment-with-grails-and-cloud-foundry/

http://groovy.codehaus.org

http://hc.apache.org/httpcomponents-client-ga/httpmime/index.html

6.2. Android:

http://developer.android.com/index.html

http://www.dzone.com/links/r/android_ui_design_pattern_dashboard.html

http://www.dzone.com/links/r/springandroid_spring_for_android.html

http://stackoverflow.com/questions/785973/what-is-the-most-appropriate-way-to-store-user-settings-in-android-application

http://stackoverflow.com/questions/4387631/callback-after-twitter-authentication

http://stackoverflow.com/questions/4177305/get-coordinates-on-tapping-map-in-android

https://github.com/commonsguy/cwac-endless

https://github.com/jgilfelt/android-mapviewballoons

http://www.softwarepassion.com/android-series-taking-photos-with-andorid-built-in-camera/

http://stackoverflow.com/questions/2227292/how-to-get-latitude-and-longitude-of-the-mobiledevice-in-android

http://mobiforge.com/developing/story/using-google-maps-android

http://code.google.com/p/androidannotations/

http://code.google.com/p/roboguice/

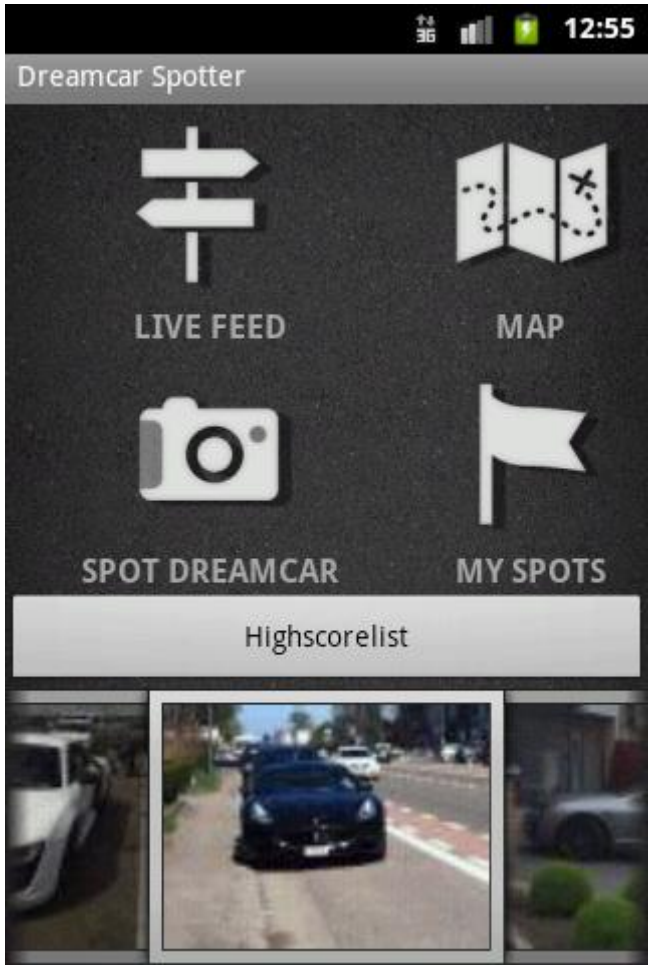
http://developers.facebook.com/

6.3. Andere:

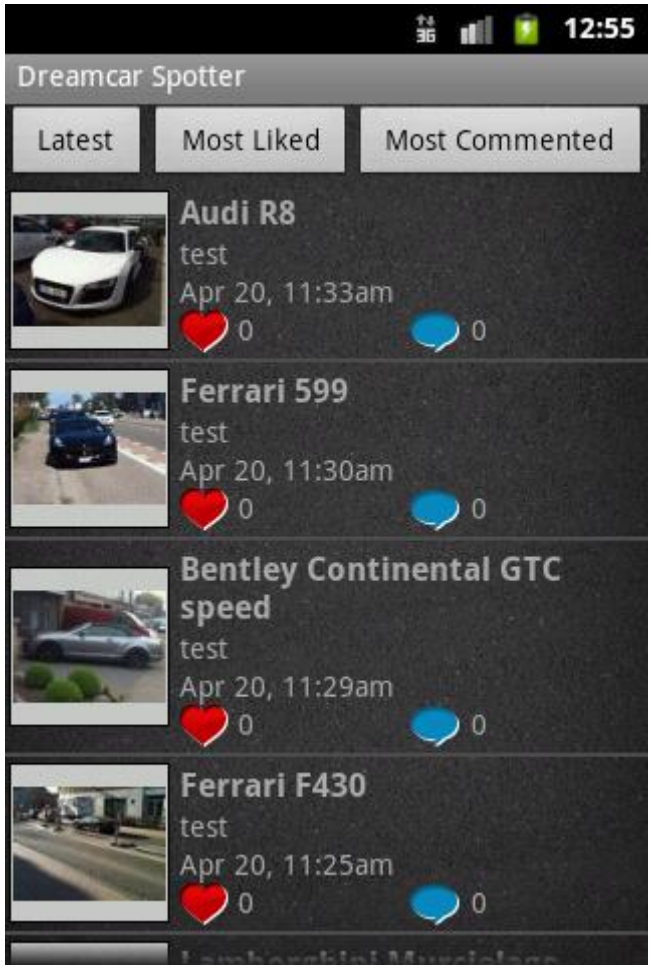
http://nl.wikipedia.org

7. Appendix

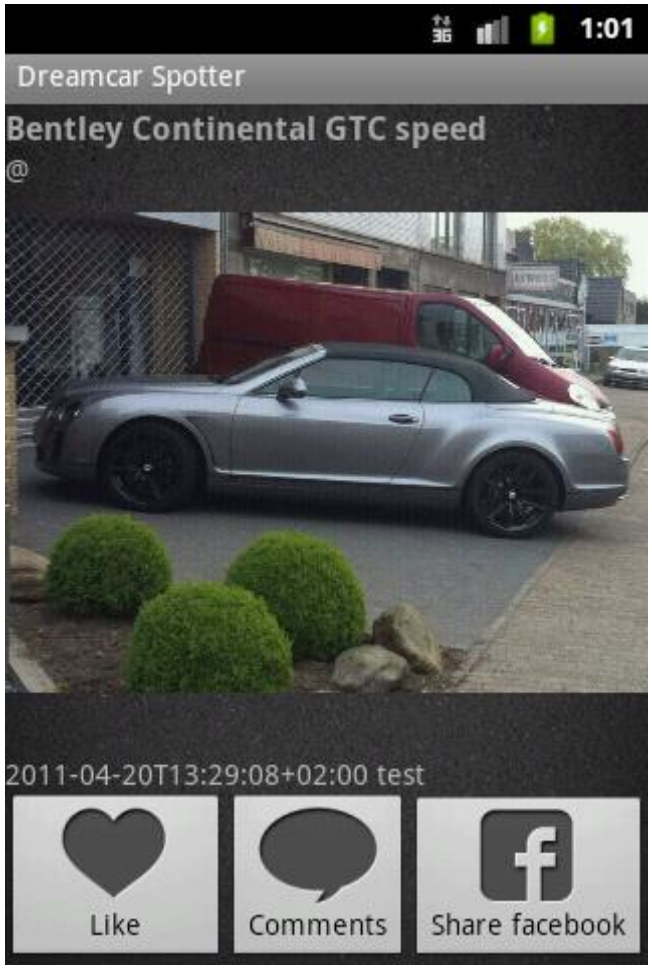
7.1. Referentieschermen



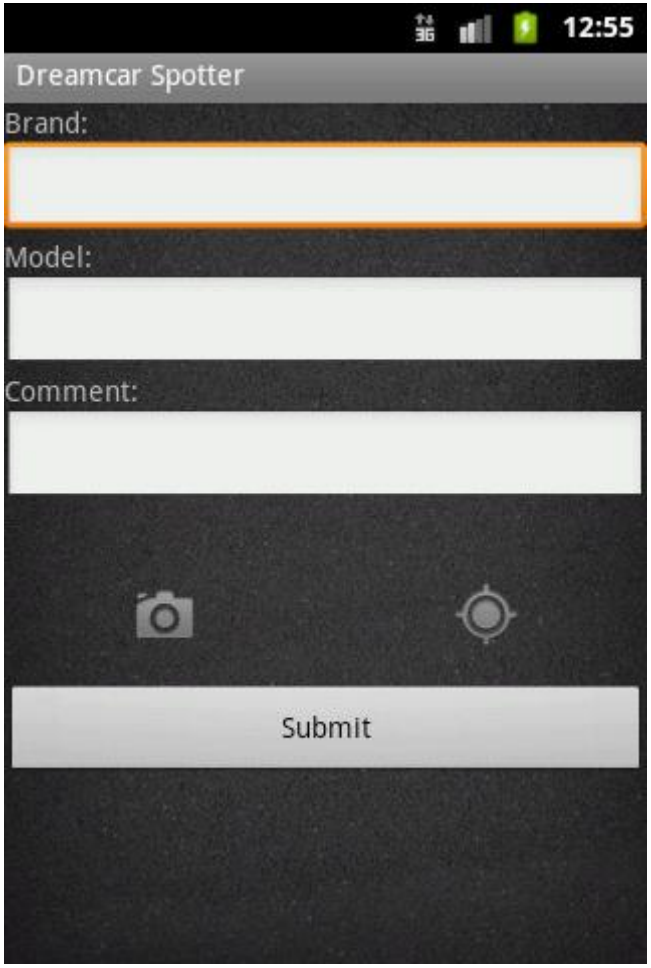
Home



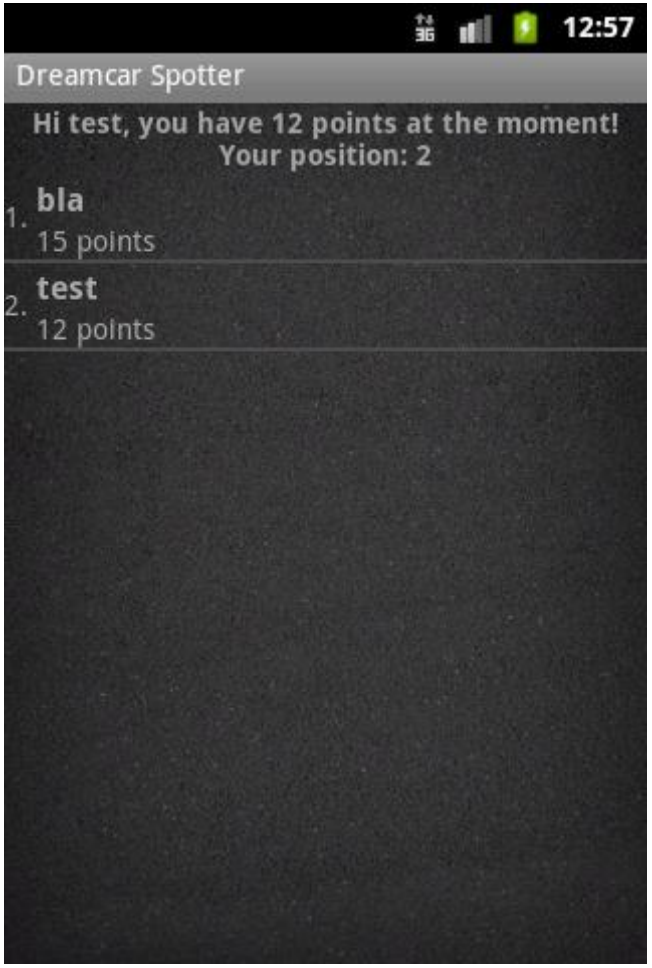
Live Feed



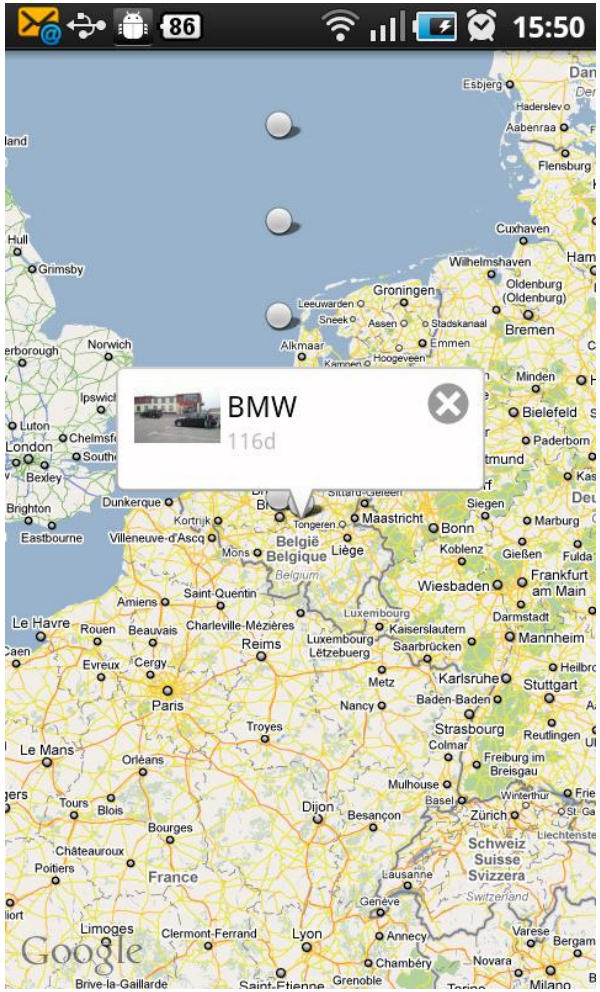
Detail



New spot



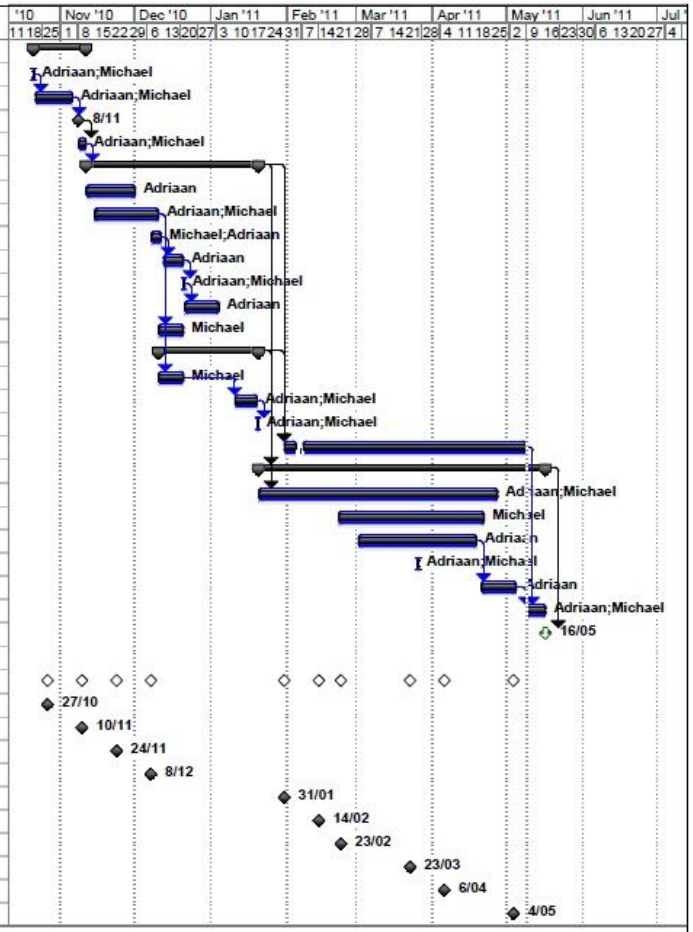
High scores



Mapview

7.2. Planning (GANTT)

ID	Task Name	Duration	Start	Finish	Predec	Resource Names
1	**Inleiding**	**15,5 days?**	**Thu 21/10/10**	**Thu 11/11/10**		
2	Interview met Klant	0,5 days?	Thu 21/10/10	Thu 21/10/10		Adriaan;Michael
3	Keuze project bepalen	11 days?	Fri 22/10/10	Fri 5/11/10	2	Adriaan;Michael
4	Deadline Keuze	0,5 days?	Mon 8/11/10	Mon 8/11/10	3	Adriaan;Michael
5	Planning maken	3 days?	Mon 8/11/10	Thu 11/11/10	4	Adriaan;Michael
6	**Analyse**	**51 days?**	**Thu 11/11/10**	**Thu 20/01/11**	5	
7	Inleiding Project document (Scope)	14 days	Thu 11/11/10	Wed 1/12/10		Adriaan
8	Requirement Analyse (Use Case)	20 days	Mon 15/11/10	Fri 10/12/10		Adriaan;Michael
9	UML Use Case Diagram opstellen	4 days	Wed 8/12/10	Sat 11/12/10		Michael;Adriaan
10	SWOT-Analyse opstellen	6 days	Mon 13/12/10	Mon 20/12/10	9	Adriaan
11	Interview met Klant	0,5 days?	Tue 21/12/10	Tue 21/12/10	10	Adriaan;Michael
12	Business Anasyle opstellen	10 days	Tue 21/12/10	Tue 4/01/11	11	Adriaan
13	Database Analyse opstellen	7 days	Sat 11/12/10	Mon 20/12/10	8	Michael
14	**UML Diagramma**	**29,5 days?**	**Sat 11/12/10**	**Thu 20/01/11**		
15	Activity Diagram opstellen	7 days	Sat 11/12/10	Mon 20/12/10	8	Michael
16	Klassendiagramma opstellen	7 days	Tue 11/01/11	Wed 19/01/11	15	Adriaan;Michael
17	Interview met Klant	0,5 days?	Thu 20/01/11	Thu 20/01/11	16	Adriaan;Michael
18	Eindwerk Document	70 days?	Mon 31/01/11	Sun 8/05/11	14;6	
19	**Implementatie**	**83,5 days**	**Thu 20/01/11**	**Mon 16/05/11**	6	
20	Implementatie	70 days	Thu 20/01/11	Wed 27/04/11	14	Adriaan;Michael
21	Unit Testing	44 days	Tue 22/02/11	Thu 21/04/11		Michael
22	Grafisch design	35 days	Wed 2/03/11	Mon 18/04/11		Adriaan
23	Interview met Klant	0,5 days	Sat 26/03/11	Sat 26/03/11		Adriaan;Michael
24	UA Testing	10 days	Thu 21/04/11	Wed 4/05/11	22	Adriaan
25	Afwerking	6 days	Mon 9/05/11	Mon 16/05/11	24;18	Adriaan;Michael
26	**Deadline Ondernemingsproject**	0 days	Mon 16/05/11	Mon 16/05/11	19	
27						
28	**Promotorvergadering Groep T**	**137 days**	**Wed 27/10/10**	**Wed 4/05/11**		**Adriaan;Michael**
29	Promotorvergadering Groep T 1	0 days	Wed 27/10/10	Wed 27/10/10		
30	Promotorvergadering Groep T 2	0 days	Wed 10/11/10	Wed 10/11/10		
31	Promotorvergadering Groep T 3	0 days	Wed 24/11/10	Wed 24/11/10		
32	Promotorvergadering Groep T 4	0 days	Wed 8/12/10	Wed 8/12/10		
33	Promotorvergadering Groep T 5	0 days	Mon 31/01/11	Mon 31/01/11		
34	Promotorvergadering Groep T 6	0 days	Mon 14/02/11	Mon 14/02/11		
35	Promotorvergadering Groep T 7	0 days	Wed 23/02/11	Wed 23/02/11		
36	Promotorvergadering Groep T 8	0 days	Wed 23/03/11	Wed 23/03/11		
37	Promotorvergadering Groep T 9	0 days	Wed 6/04/11	Wed 6/04/11		
38	Promotorvergadering Groep T 10	0 days	Wed 4/05/11	Wed 4/05/11		



Project: Gantt_MSP
Date: Mon 9/05/11

Task — Milestone — External Tasks
Split — Summary — External Milestone
Progress — Project Summary — Deadline

Page 1

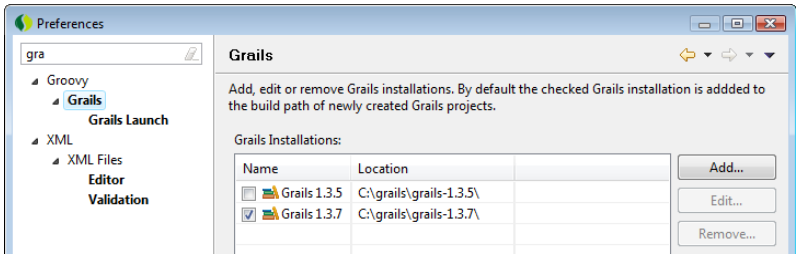
7.3. Installatie handleiding (Engelse versie)

Grails project

1. Install Springsource Toolsuite (Eclipse)
   http://www.springsource.com/developer/sts

2. Download & unpack latest Grails framework on local drive
   http://www.grails.org/

3. Bind Grails framework to eclipse in preferences->Groovy->Grails



4. Install Subclipse svn plugin for Eclipse (Help->Install new software) by entering update site (see http://subclipse.tigris.org/servlets/ProjectProcess;?pageID=p4wYuA)

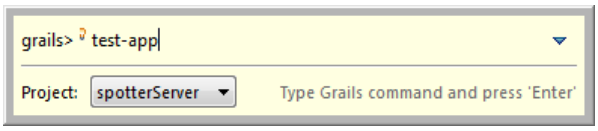
5. Checkout code from svn
   a. svn://62.182.62.173/dcspotter/source/trunk/spotterServer

6. If there are project dependency errors, right-click on project->Grails Tools->Refresh dependencies

7. Data source in development environment uses in-memory local DB, so no configuration needed here (unless you want to specify a person MySQL, do config DataSource.groovy)

8. If you use a newer version of the Grails framework than configured in the project (at this time 1.3.7) Grails will ask you to update the project. (or force it by running grails update)

9. Run all tests by opening the Grails command window (Ctrl+Alt+Shift+G) and enter “test-app”



10. Run the application with “run-app” and browse to http://localhost:8080/spotterserver

Other useful Grails commands:

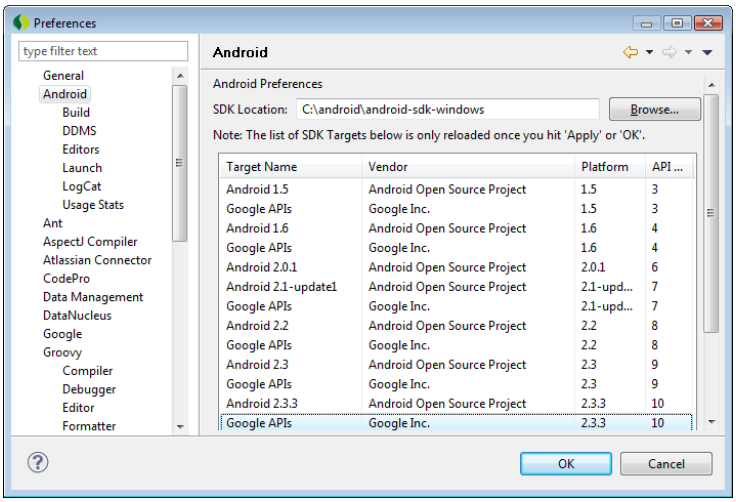
- **prod tomcat deploy:** Deploy to remote tomcat (Faros’ Priorweb hosting, see Config.groovy -> environments)

- **prod cf-update:** Update deployed app on cloudfoundry (http://spotterserver.cloudfoundry.com)
- **prod schema-export stdout:** Prints the database scheme to console

Android project

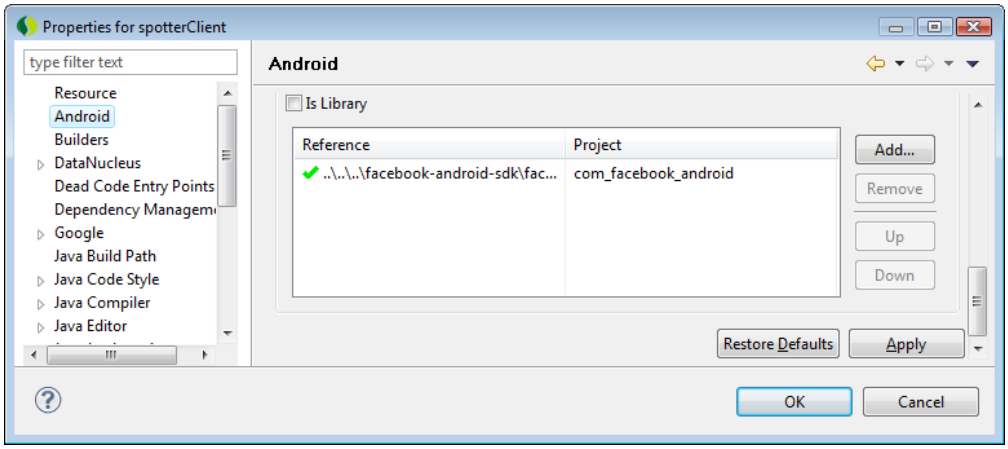
1. Install the Android SDK and ADT eclipse plugin
   http://developer.android.com/sdk/installing.html
   The project is tested on version 1.6 and above, so you can choose a SDK API version starting from 4 (but I guess you want to use the latest of course!) At this time that's API level 10 (2.3.3) Make sure to install the Google API also (needed for Google Maps)

2. Make sure eclipse is configured to use the right Android SDK in Preferences->Android



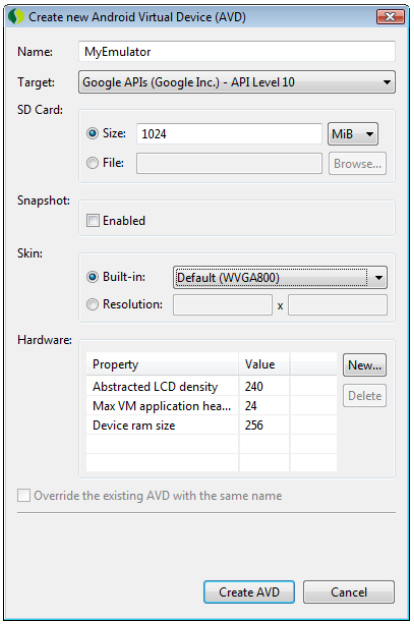
3. Checkout the source code!
   a. svn://62.182.62.173/dcspotter/source/trunk/spotterClient

4. Checkout the android facebook sdk and bind it!
   a. Download git plugin for eclipse http://www.eclipse.org/egit/
   b. Checkout git://github.com/facebook/facebook-android-sdk.git
   c. Right click spotterClient project-> Properties -> Android-> Add Library



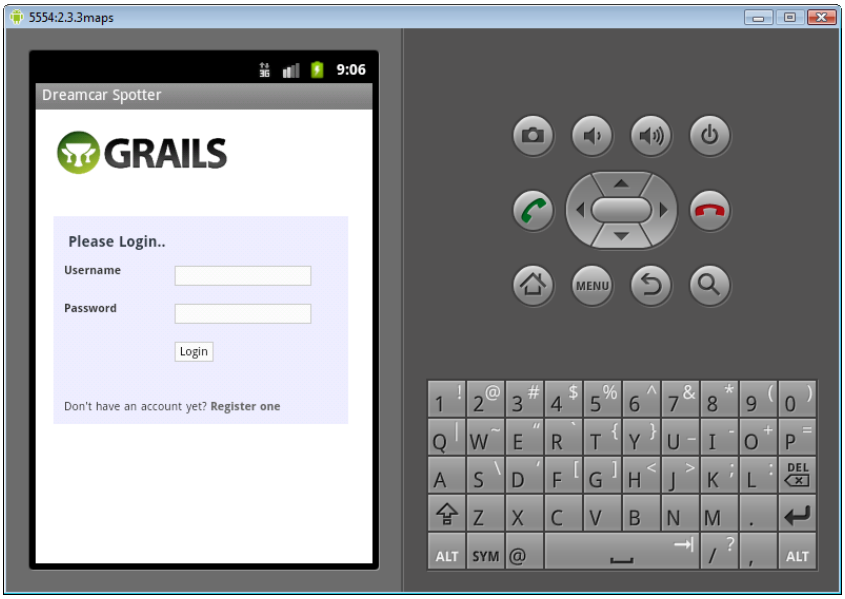
5. Open Config.java and define the server URL (That's where the Grails application runs). Use 10.0.2.2 for localhost. (The emulator has its own IP)
6. Create a new Emulator instance by opening Window -> Android SDK and AVD Manager and choose New…

   Choose a GoogleAPI as target and include a virtual SD-card



7. In the manager screen launch the emulator by selecting it and hitting "Start"

8. If booted up, right click the android project and Run-As -> Android application.



You can also hookup an actual Android device for testing. The only thing you need to do is enable USB debugging on the device and hook it up. Your device will be visible in the Devices panel on the DDMS perspective.