Multimedia: final report

Groep “Nog geen naam, maar wanneer hij er is zal hij episch zijn!”
Blog https://teammume.wordpress.com/
HTML5 URL http://bptoledo.appspot.com/
Brian Burlet, 1Ma Ing.Wet:CW
Pieter Bamps, 2Ma Ing.Wet:CW

Abstract

Dit is het finale verslag voor multimedia van de groep ‘*Groep nog geen naam; maar wanneer hij er is zal hij episch zijn!*’ In dit verslag wordt een mobiele versie van een interactief leerplatform ontwikkeld gebaseerd op informatie uit Twitter en Google Docs.
In deze applicatie staan notificaties en een post-stream centraal. Notificaties zijn hier nieuwe posts, berichten, commentaren op berichten, aanpassingen aan documenten,... De post-stream is een pagina waarop alle studenten berichten kunnen posten die vak-gerelateerd zijn (gebaseerd op de Facebook wall). Ook de verscheidene tweets die bij een vak horen worden hier weergegeven. Deze applicatie is ontwikkeld in HTML5 en voor de handheld iOS in Objective-C. Als back-end werd de Google App Engine gebruikt.

1. Idee

Het hoofddoel van dit project is het ontwikkelen van een mobiele versie van een interactief leerplatform (Toledo). In deze applicatie wordt de focus uitsluitend gelegd op de integratie van tweets en de ontwikkeling van een berichten-muur,waar studenten informatie op kunnen plaatsen die gerelateerd is aan de verschillende cursussen. Hierbij wordt gebruik gemaakt van een rating-systeem om de nuttige berichten te onderscheiden van de rest. Er zal ook een notificatie-systeem ontwikkeld worden die de gebruiker op elk moment laat weten hoeveel ongelezen boodschappen er aanwezig zijn in het systeem. De 2de grote peiler in deze applicatie is het gebruik van zogenaamde ‘workfloors’’. Deze workfloors stellen de student een omgeving ter beschikking waar hij op een makkelijke manier groepswerken kan organiseren. De student kan andere gebruikers uitnodigen tot zijn workfloor en hierin kunnen ze dan gebruik maken van een Doodle-tool/een Google Docs om groepswerken te vergemakkelijken. De laatste peiler is de kalender-tool. Er zal een kalender ontwikkeld worden waarin de studenten ook persoonlijke afspraken kan plannen. In de huidige versie van Toledo is dit onmogelijk en wordt de student verplicht gebruik te maken van een secundaire kalendertool zoals Google Calendar. Dit is echter verre van ideaal en een integratie van deze in een algemene kalender lijkt ons zeer handig.
Het sterkste punt van deze applicatie is volgens ons het concept van de workfloors. Voor elk groepswerk wordt vaak wel een Google Docs en Doodle opgericht om vanop afstand te kunnen samenwerken en het lijkt ons dus zeer handig als het mobiele leerplatform dit reeds ter beschikking zou stellen. We denken hierbij dan vooral aan tablet-gebruikers die een groter scherm ter beschikking hebben. Echter kan dit ook voor smartphone gebruikers nuttig zijn om snel even een Doodle in te vullen of een todo-lijstje te controleren. Het zwakke punt van deze applicatie is dat er op geen enkel moment gebruik wordt gemaakt van typische mobiele

functionaliteit.(gps-sensor etc) Het workfloor idee is voortgevloeid uit Google Wave en het berichten-wall idee is afkomstig van Facebook.

2. Scenario

Het uitschrijven van enkele scenario's is zeer nuttig om de kern functionaliteit van de applicatie duidelijk te maken zonder dat men zich moet bekommeren om implementatie-details. Het is echter zeer nuttig om samen met het scenario een paper prototype of storyboard te ontwikkelen want deze geven extra informatie omtrend de organisatie van de use-cases.

Scenario 1: catching up (english)

- Pieter wants to catch up on one of his courses. He opens our application on his portable device and sees that he has 6 new updates for his MUME course.
- He initially sees the course stream and sees that there are 3 new updates on the stream, he checks them and he gives a positive rating to a video that one of his fellow students had posted. This to indicate that he thinks that the video is useful to his fellow students.
- On his screen he sees that among the other categories: Slides, Group, Information,… There are still 2 updates for the Group section and 1 update for the Slides section. He clicks on the Slides icon and sees that the professor has updated one of the files.

Scenario 2: creating a group (english)

- Brian is commencing a group project for the MUME course. He goes to our webapp via his laptop and goes to the conversation page.
- He clicks on the group icon, which will result in a "create a group"-wizard. (because Brian isn't part of a group yet)
- Brian first selects the other students that are part of his group .
- Once finished the other students get a notification that a group has been created.
- Brian immediately wants to schedule a meeting and clicks on the scheduler icon. This scheduler compiles a list of time slots based on the free moments in the calendar of the students in the group. Brian selects which time slots seem interesting and a doodle gets created.

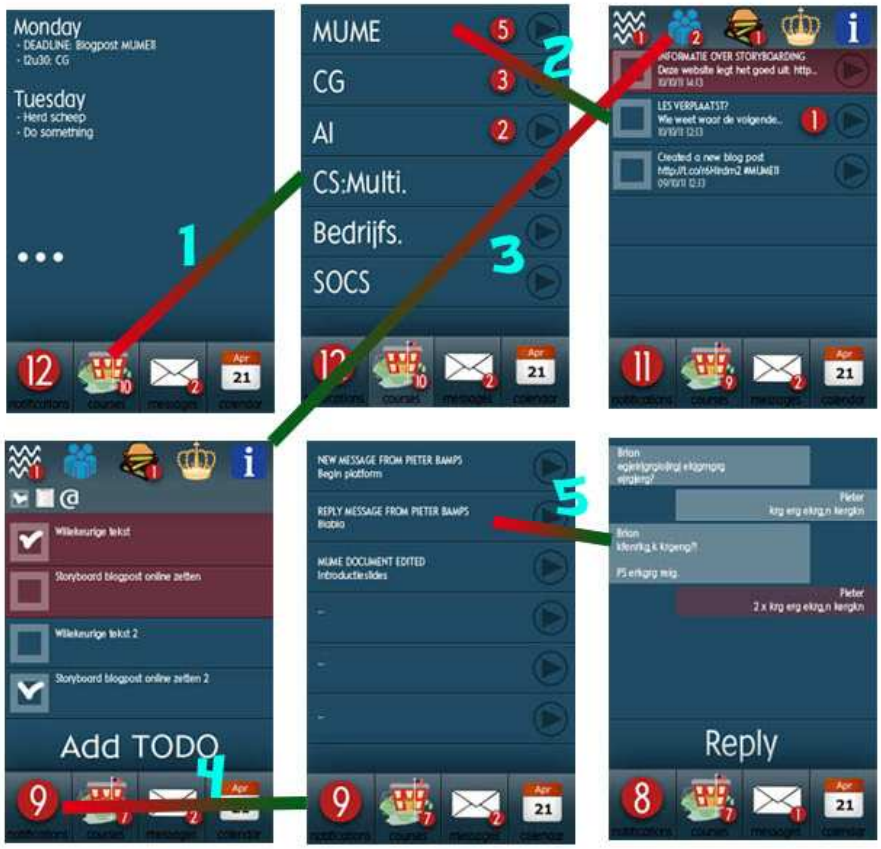
Het eerste scenario focust zich duidelijk op de notificaties. Het is zeer belangrijk dat het systeem de gebruiker duidelijk maakt waar zich nieuwe informatie bevindt en hoe deze te bereiken. Het ratingsysteem heeft echter de finale versie van de applicatie niet bereikt wegens tijdsgebrek. Het idee achter het 2de scenario is om het samenwerken aan groepswerken te vergemakkelijken. Enerzijds is er een algemene pagina voor ieder vak waar alle vak-gerelateerde informatie gebundeld staat(files/tweets/messages) anderzijds is er de groepswerkpagina waar de gebruiker persoonlijk mensen kan toevoegen om hiermee project-gerelateerde conversaties te houden. De doodle en kalendertool om meetings te regelen is niet ontwikkeld in de finale versie van de applicatie. Er is echter wel een Google Docs pagina die de gebruiker kan delen met andere studenten.

3. Storyboard

De hoofdfunctionaliteiten van deze applicatie zijn:

1. Op een duidelijke manier gebruikers op de hoogte brengen van nieuwe beschikbare informatie. (updates, reacties op commentaar, nieuwe afspraken,...)
2. Het voor studenten makkelijker maken om groepswerken te regelen via het mobiele leerplatform. Dit enerzijds door privé-conversaties en anderzijds met een geïntegreerde Google Docs.

Dit is duidelijk zichtbaar in het storyboard:



Our initial screen shows the calendar with all the upcoming deadlines/classes.

1. The user clicks on courses and gets a new screen with all his courses and the number of updates each course has.
2. The user clicks on MUME, a course that has 5 updates. He arrives on the initial screen, being the public stream.
    - Notice that the number of new updates has decreased because new updates have been viewed. In this case “Informatie over storyboarding”.
3. The user clicks on the group icon to indicate that he wants to see how his group work is progressing. The new screen shows a list of TODOs and the group sub-menu, the icons on this sub-menu depend on the choices that were made by the first user. (Check scenario 2)
    - Again the number of new notifications dropped (@group icon, @courses icon and @notifications), because 2 of them are being seen by the user.
4. The user clicks on the notifications icon, he now receives a list of the new notifications.
5. He clicks on one of the items on the list to check what Pieter replyed to his message.

Update: Uiteindelijk hebben we de focus van deze applicatie op de Twitter en Facebook wall functionaliteit gelegd waardoor de kalender-functionaliteit niet is ontwikkeld. Een 2de verandering in het storyboard is het startscherm. Na het inloggen komt de gebruiker onmiddelijk op de pagina met zijn cursussen terecht ipv op de kalenderpagina. Ook het leaderbord is niet ontwikkeld omdat dit voor ons maar secundaire functionaliteit was. Studenten kunnen elkaar uitnodigen voor conversaties en hierbij privé met elkaar communiceren ipv dat de informatie op de vak-stream komt. De screens van onze applicatie staan in de appendix. (HTML5 screens)

4. Globale architectuur

De back-end architectuur van dit project is gebaseerd op de Google App Engine. Deze stelt een databank ter beschikking die het mogelijk maakt op een makkelijke manier informatie te bewaren en op te vragen door middel van een API geschreven in Java. Een tweede grote pijler in de backend is de scraper. Deze scraper, gebaseerd op JSON laat de applicatie toe om tweets en blogposts van websites af te halen. De communicatie met deze services werkt door middel van cron jobs die in Java geprogrammeerd zijn. De verwerking van deze gegevens gebeurt allemaal in de server.De front-end van onze applicatie in beide technologieën zal worden besproken in puntje 5 en 6.

4.1 Login/register

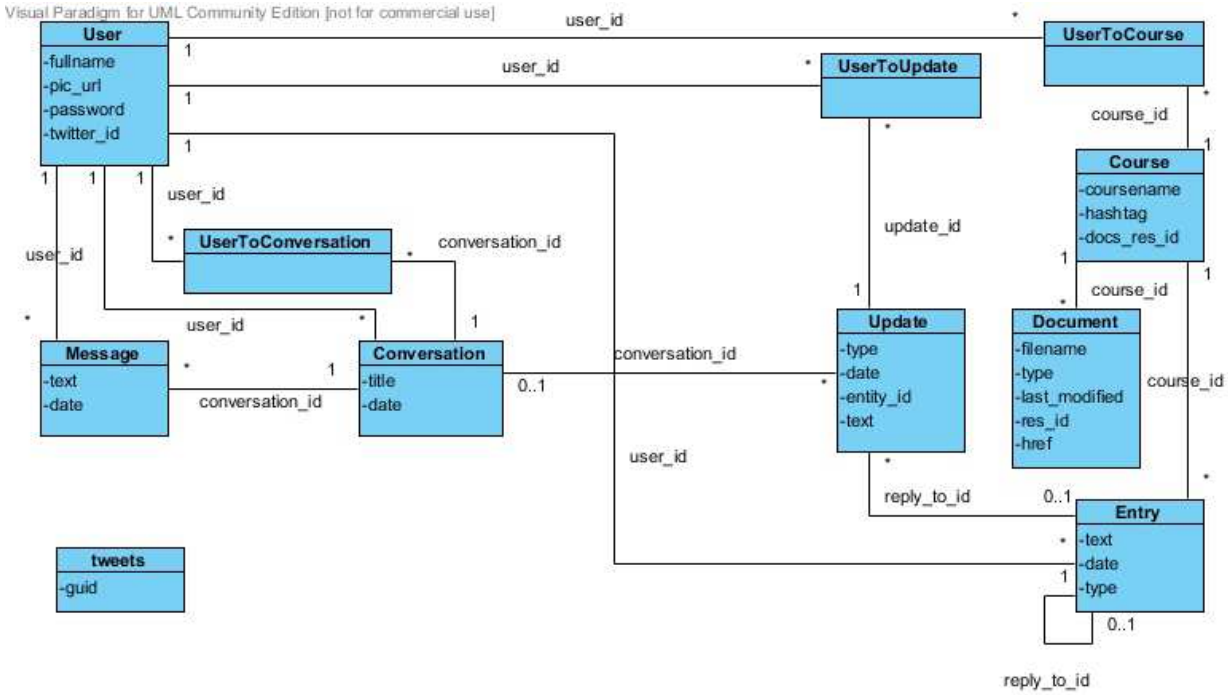
De login hebben we verbonden met de Twitter API. Een gebruiker moet zijn Twitter id en een wachtwoord opgeven en samen met de Twitter API komen we zo meer te weten over de nieuwe gebruiker.

4.2 App Engine

Voor de ontwikkeling van deze applicaties wordt gebruik gemaakt van de Google app engine. Hierin is zoveel mogelijk functionaliteit gestoken zodat de HTML5 en iOS applicatie zich volledig konden focussen op de view.
Deze back-end heeft 4 grote pijlers, namelijk: “Database, Cron jobs, GET JSON, POST”.

4.2.1 Database

Allereerst wordt er gebruik gemaakt van een database die door Google wordt aangeboden samen met de GAE API. De database kan in een nieuwe staat terecht komen door simpele Java opdrachten. Een alternatief is het gebruik van een MySQL database, maar die zijn iets complexer in omgang.
Hieronder een domeinmodel van onze database. De enigste verbinding is de entity_id van Update, omdat deze afhankelijk van de type verbonden kan zijn met een Document, Entry, Conversation of Message.



4.2.2 Cron jobs

4.2.2.1 Twitter

Om de 30 minuten worden van alle vakken aan de hand van hun Twitter hashtag tweets opgeslagen in het systeem. Deze tweets worden aan de hand van de gebruikersnaam van de tweeter gelinked aan de juiste gebruiker.

4.2.2.2 Google Docs

Om het uur worden van alle vakken alle documenten van de persoonlijke Google Docs collection gesynchroniseerd. Deze collections worden gescraped aan de hand van de GoogleService (GS) API. Deze API is verbonden aan een Google Account die we gemaakt hebben voor dit vak. In geval dat je nieuwe bestanden wil toevoegen dan kan dat door in te loggen op de Toledo Google account en een bestand up te loaden naar de juiste collection. (deze collection wordt automatisch aangemaakt bij het aanmaken van een nieuw vak)

4.2.3 GET JSON

Data uit de database wordt opgevraagd aan de hand van zelfgeschreven API's die JSON resultaten leveren. Beide technologieën gebruiken deze om data op te vragen. Voor HTML5 was er ook de mogelijkheid om gebruik te maken van de JSP pagina's, maar de technologie bij de JSP pagina's laat niet toe om asynchroon data op te vragen. Daarom hebben we besloten om te werken met een mix van jQuery en JSON voor HTML5.
De regel bij onze zelfgeschreven API is dat alle informatie die nodig is ook meteen wordt meegegeven. Zodat we niet meerdere calls moeten maken voor 1 pagina. (docs@appendix) Vrijwel elke getter geeft dan ook het aantal updates mee die in elke categorie overblijven. In het geval dat je resultaten meegeeft die zelf nog updates zijn dan worden deze updates verwijderd uit de database.
Bijvoorbeeld indien je de comments opvraagt van een entry en 3 entries daarvan zijn nog niet gelezen, dan worden deze als updates verwijderd en krijgen deze bij het opvragen een extra

flag die op true wordt gezet om aan te tonen dat deze nog niet gelezen zijn. (in dit geval maken we deze cellen rood)

4.2.4 POST

Om data in de database te steken maken we gebruik van de POST methode. Een alternatief is gebruik maken van JSON, maar dat is onveilig en licht onpraktisch. Bij het oproepen van deze POST-methodes worden ook de juiste updates toegevoegd. (namelijk voor alle betrokken partijen, behalve de user die het bericht gepost heeft) (docs@appendix)

5 iOS

Het ontwerp in iOS maakt gebruik van het zogenaamde model-view-controller pattern. Hierbij wordt een strikte scheiding gemaakt tussen enerzijds de User-Interface en anderzijds 'het model'. De communicatie tussen deze 2 wordt onderhouden door controllers. Telkens als de user-interface informatie nodig heeft uit het model vraagt het dit via de gepaste controller die vervolgens de data uit het model haalt. In deze iOS-applicatie staan de verschillende controllerklassen in verbinding met de Google App Engine via *HTTP get* en *HTTP post* requests. Voor deze functionaliteit is gebruik gemaakt van het SBJson pakket. Dit laat het mogelijk om op een zeer eenvoudige manier requests naar de http-pagina te sturen via JSON berichten.

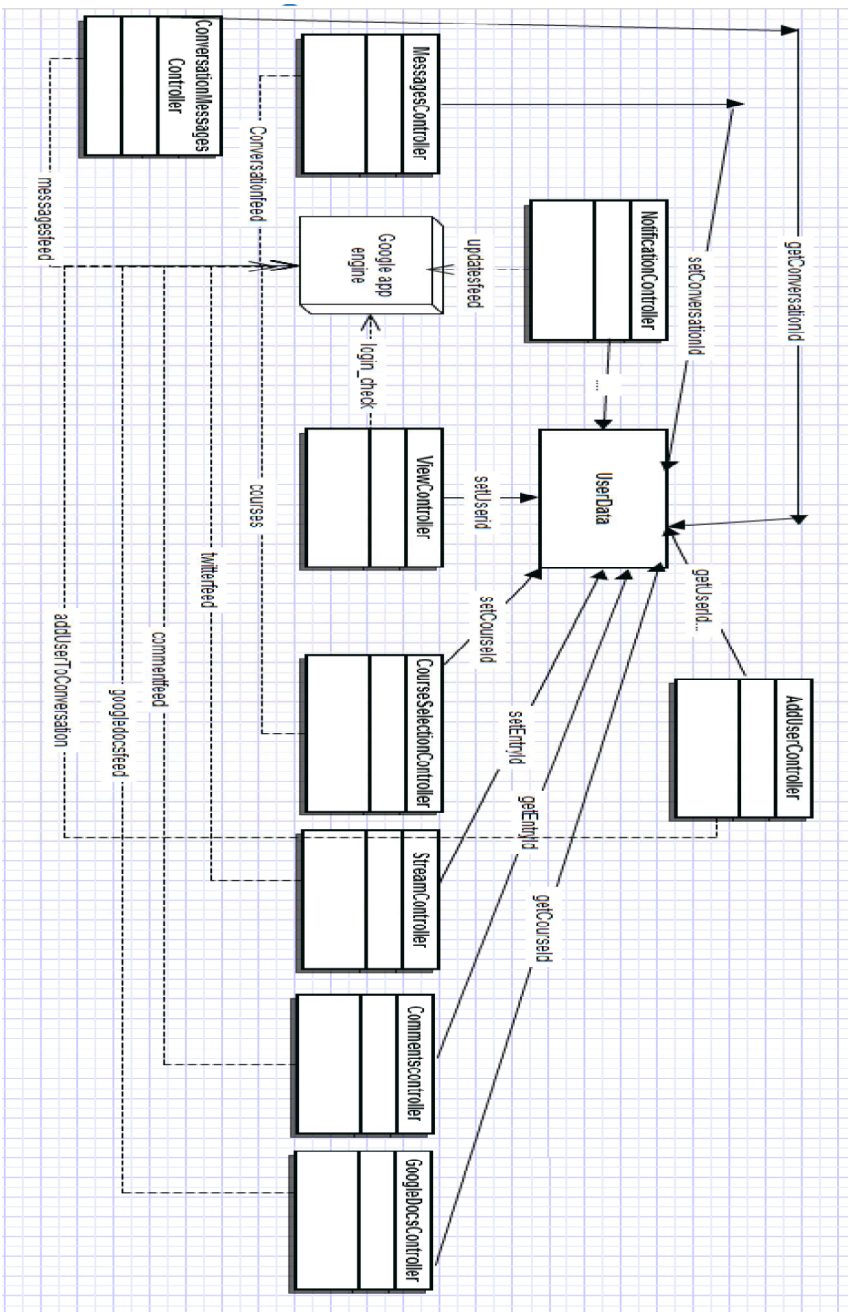
5.1 Storyboard

Voor de ontwikkeling van de user interface werd gebruik gemaakt van de handige functionaliteit van het storyboard. Dit maakt het mogelijk om op een makkelijke manier navigation controllers en tabbar controllers te creëren die instaan voor de navigatie doorheen de applicatie. Voor elke tabel die de applicatie bevat is een custom cell ontwikkeld die de nodige informatie op een mooie manier op het scherm toont. Zo bevat de streamcell een Twitter-avatar terwijl dit voor de notificaties niet nodig is. Voor invoer van de gebruiker op te vangen is gebruik gemaakt van 'alerts' die text input toelaten. Dit lijkt ons een makkelijkere manier dan dat de gebruiker moet zoeken naar een input-field op het scherm. In deze applicatie wordt in de verschillende tabs gebruik gemaakt van badges om het aantal nieuwe notificaties weer te geven. Deze worden dynamisch aangepast indien een nieuw bericht bekeken wordt door de gebruiker. Hierdoor zijn er 2 extra tabbarcontroller-klassen aangemaakt in het project die niet in het klassendiagram (zie 5.2) voorkomen. Deze 2 klassen bevatten een *'checkUpdate'*-methode die dit aanpast. Deze klassen staan ook in verbinding met de Userdata klasse (zie 5.2) want hier wordt namelijk bijgehouden op welke pagina de gebruiker op elk moment in de applicatie zit om zo de correcte navigatie te voorzien.

5.2 Klassendiagram

Het klassendiagram van de iOS-applicatie heeft voor iedere view een aparte controllerklasse. Zo is er de Google*docs-controller,de notification-controller, de stream-controller etc*. Al deze klassen hebben als functie om de nodige informatie uit het model te halen en dit vervolgens weer te geven in een tableview. Iedere klasse is ook verbonden met de klasse 'UserData'. Hierin wordt informatie van de huidige gebruiker gestoken. Bij het inloggen wordt hier het *userId* bijgehouden, indien er op een entry van de stream geklikt wordt zal hier het *entryId* worden

bijgehouden om zo de juiste get-methoden naar de back-end te sturen bij het opvragen van de antwoorden op een vraag. De *AddUser-klasse* bevat een searchbalk. Met deze searchbalk is de gebruiker in staat om gebruikers te zoeken in het systeem en op deze manier toe te voegen aan een conversatie.
Deze searchbalk bevat autocomplete. (deze bevat wel een kleine bug waardoor hij 1 letter te laat de lijst met zoekresultaten aanpast) De Google Docs controller tenslotte bevat een methode om nieuwe email-adressen toe te voegen. Hiermee kan je groepsleden bij een project toevoegen zodat je samen aan een project kan werken via Google Docs. Deze staat dan in verbinding met de browser-client van de iPhone om de Google Docs-webpagina weer te geven.



Figuur: klassendiagram iOS

6 HTML5

6.1 View

Er zijn momenteel 2 pagina's. De login en de applicatie. De login, waar de gebruiker zich aanmeld, en de applicatie pagina, een vrij eenvoudig ontwerp met vanonder een navigation bar, centraal een listview en vanboven een header, waar de werkelijke applicatie zich bevindt, deze pagina wordt asynchroon aangepast.
Aan de navigation bar verandert er niets buiten het aantal updates, indien een update wordt bekeken dan zullen de correcte tellers in deze bar naar beneden gaan.
De listview is het element dat het vaakst verandering zal meemaken. Vrijwel telkens als er op een knop gebruikt wordt wordt deze lijst geledigd en wordt er asynchroon de nieuwe correcte informatie ingestoken.
De header duidt aan waar de gebruiker zit in de applicatie en hierdoor wordt de titel vaak aangepast, ook zijn er meestal 2 knoppen ter beschikking, 1 om een verdieping hoger terug te keren (van een vakpagina naar een lijst met vakken, van een message naar de lijst met conversations,...) en een knop om items toe te voegen (nieuwe conversation, nieuwe post, nieuwe comment,...)

6.2 Async vs sync

Voor onze applicatie hebben we gekozen om Het voordeel van een asynchrone website tegenover een synchrone website is dat er voor een "nieuwe" pagina een minimum van informatie wordt opgevraagd en dat deze pagina's op die manier sneller bij de gebruiker terecht komen. De nadelen zijn dat je enorm goed moet weten hoe je architectuur in elkaar steekt en je te maken kan hebben met licht complexe code. (in de plaats van al je pagina's apart te construeren moet je de veranderingen construeren)
Indien je zelf een functie schrijft die dit voor jouw regelt dan zijn amper problemen meer.
Een tweede nadeel is dat mobile jQuery dynamische updates niet compleet enorm ondersteunt, daarom moeten we soms "hacks" gebruiken om zeker te zijn dat onze layout niet gecompromitteerd wordt.

Voorbeeld van een "hack"

```
<!-- oorspronkelijke button (pijl omlaag, die onclick een pijl omhoog moet worden) -->
<a id="add" data-icon="arrow-d">Add</a>
```

In een synchrone pagina zouden we onze code onclick kunnen schrijven als:

```
<!-- oorspronkelijke button (pijl omlaag, die onclick een pijl omhoog moet worden) -->
<a id="add" data-icon="arrow-d">Add</a>
```

Maar omdat we werken met asynchrone pagina's moeten we de css zelf aanpassen door.

```
<!-- hack, de reden is omdat jQuery aan de hand van attributen zelf een nieuwe structuur voor de pagina's maakt -->
$("#add > .ui-btn-inner > .ui-icon").removeClass("ui-icon-arrow-d");
$("#add > .ui-btn-inner > .ui-icon").addClass("ui-icon-arrow-u");
```

6.3 jQuery

De HTML5 applicatie maakt zwaar gebruik van jQuery een javascript platform dat gebruikt maakt van graceful degradation om zijn layout on demand te genereren. Het voordeel is dat hierdoor is dat een complexe layout maken enorm simpel wordt een klein nadeel is dat de layout enkel tijdens het laden van de pagina gegenereerd wordt, dus wanneer we asynchroon data opvroegen werdt onze layout niet mee geupdated. Een klein stukje code, namelijk listview('refresh'), dat de layout genereerd voor een specifiek stukje code als een listview.

6.4 Pagina async laden

De meeste pagina's worden op dezelfde manier geladen in HTML5.

1. Eerst wordt window.sessionStorage.page op de juiste pagina gezet. (is nodig voor wanneer een pagina herlaad wordt)
2. De juiste JSON wordt asynchroon opgevraagt met de juiste parameters.
3. Eens de respons er is wordt de layout aangepast.
    a. De pagina titel wordt op de correcte waarde gezet.
    b. De lijst wordt geledigd voor de nieuwe data.
    c. De update tellers worden aangepast.
4. Hierna worden de resultaten van de JSON één voor één ge-append aan de lijst.
    a. In geval dat iets een update is wordt de cell rood gemaakt.
    b. De resultaten worden ge-append omdat de backend de resultaten al voor ons in de juiste volgorde hebben geplaatst.
5. Tenslotte wordt listview('refresh') opgeroepen op de nieuwe lijst met data.

7 Vergelijking van de technologieën

Het eerste grote verschil tussen iOS en HTML5 is dat in iOS de developer moet werken volgens het model-view-controller pattern. Daar waar dit meestal altijd een goede programmeertechniek is , wordt dit niet verplicht opgelegd door HTML5. Een 2de verschil is het storyboard. Het is in Xcode zeer gemakkelijk om een User Interface te maken van de applicatie. Links tussen verschillende pagina's worden op een visuele manier voorgesteld en dit is zeer handig om het overzicht van de applicatie te behouden. Een van de moeilijkheden bij de ontwikkeling voor HTML5 is dat de errors pas aangegeven worden at runtime. Dit in tegenstelling tot de iOS waar het debuggen vlotter verloopt met duidelijk error messages. Het grote voordeel aan HTML5 is dat het op ieder device kan runnen daar waar men voor iOS een Apple Device nodig heeft om de applicatie op te testen. Ook het programmeren dat alleen mogelijk is op een Mac was een vrij groot nadeel.
Voor HTML5 wordt de layout on demand pas gemaakt, daarom hoeft men tijdens intens

debuggen niet steeds de server te herstarten. Enkel wanneer men iets aanpast aan de parameters van de backend die wel op voorhand wordt gecompileerd.
Een groot verschil tussen onze iOS en HTML5 applicatie is dat de layout synchroon geladen worden in iOS en in HTML5 is alles asynchroon. Dit omdat iOS werkt met gecompileerde code die lokaal is opgeslagen en dus is er vrijwel geen tijdverlies tijdens het wisselen van pagina, dit tegenover HTML5 waarbij de nieuwe pagina nog ge-download'ed moeten worden en daarna moet de layout nog helemaal gegenereerd moet worden.

8 Peer review

Score van:	Job performance	Attitude	Leadership	Resources	Communication
Pieter aan Brian	4	4	4	4	4
Brian aan Pieter	3	4	3	3	4

Pieter: Brian scoort in elke categorie bovengemiddeld. Hij was een week voor de deadline al klaar met de HTML5 applicatie waardoor hij mij kon helpen met de iOS-app. De attitude was ook altijd goed. Op gebied van leadership scoort hij ook goed. De laatste week heeft hij mij geholpen met de iOS applicatie af te werken. Brian werkt altijd geconcentreerd door en verspilt weinig tijd aan minder noodzakelijke dingen. In deze applicatie hebben we altijd onze tijd eerst in kernfunctionaliteit gestoken en vervolgens als er tijd over was werd deze gespendeerd aan randfunctionaliteit. Voor de communicatie maakten we gebruik van een facebookgroep en dit verliep zeer vlot.

Brian: Tijdens de eerste peer-review stond de iOS applicatie enorm ver achter en voelde ik me genoodzaakt om een vrij lage score te geven. Al wist ik zelf dat dit vooral aan de initiële grote leercurve van iOS lag. Maar sindsdien heeft Pieter enorm hard gewerkt om de applicatie toch op tijd af te krijgen, en zag ik dagelijks grote vooruitgang in het ontwerp tijdens de vele bijeenkomsten die Pieter regelden.

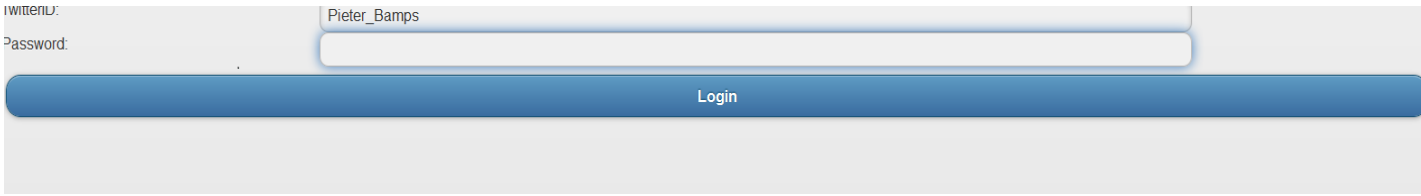
9 Timemanagement tabel met uren

	Brian Burlet	Pieter Bamps
Breinstorm	3	3
Scenario	2	2
Storyboard	2	3
Eigen blog	5	5

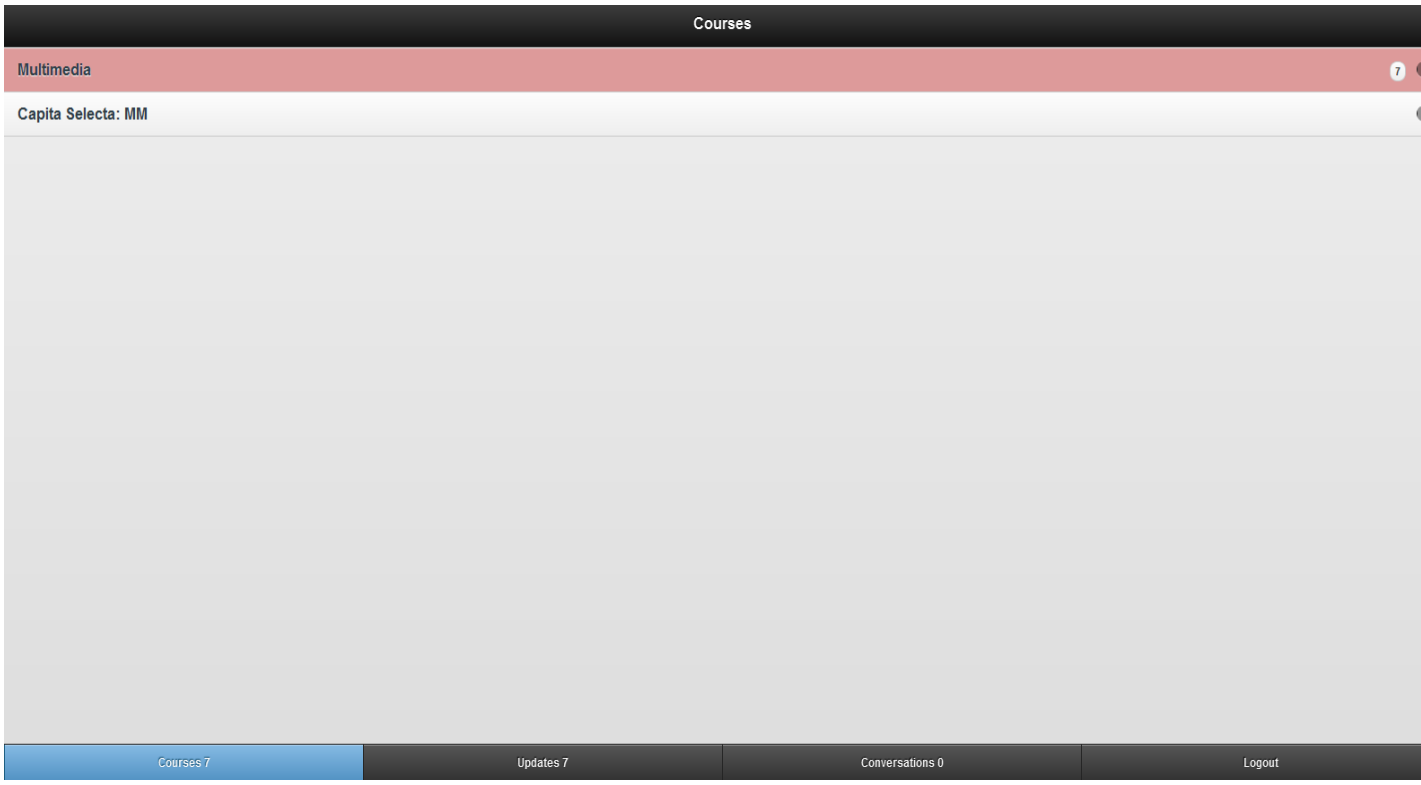
Blog van anderen	4	4
Twitter	1	1
Verslag 1	1	1
Aanleren technologie	22	40
=> Google App Engine	10	7
=> iOS	12	15
=> HTML5	10	18
Programmeren	55	95
=> Google App Engine	37	10
=> iOS	25	65
=> HTML5	30	20
Intern overleg	3	3
Verslag 2	3	3
Verslag 3	4	6
Totaal	150	166

10 Appendix

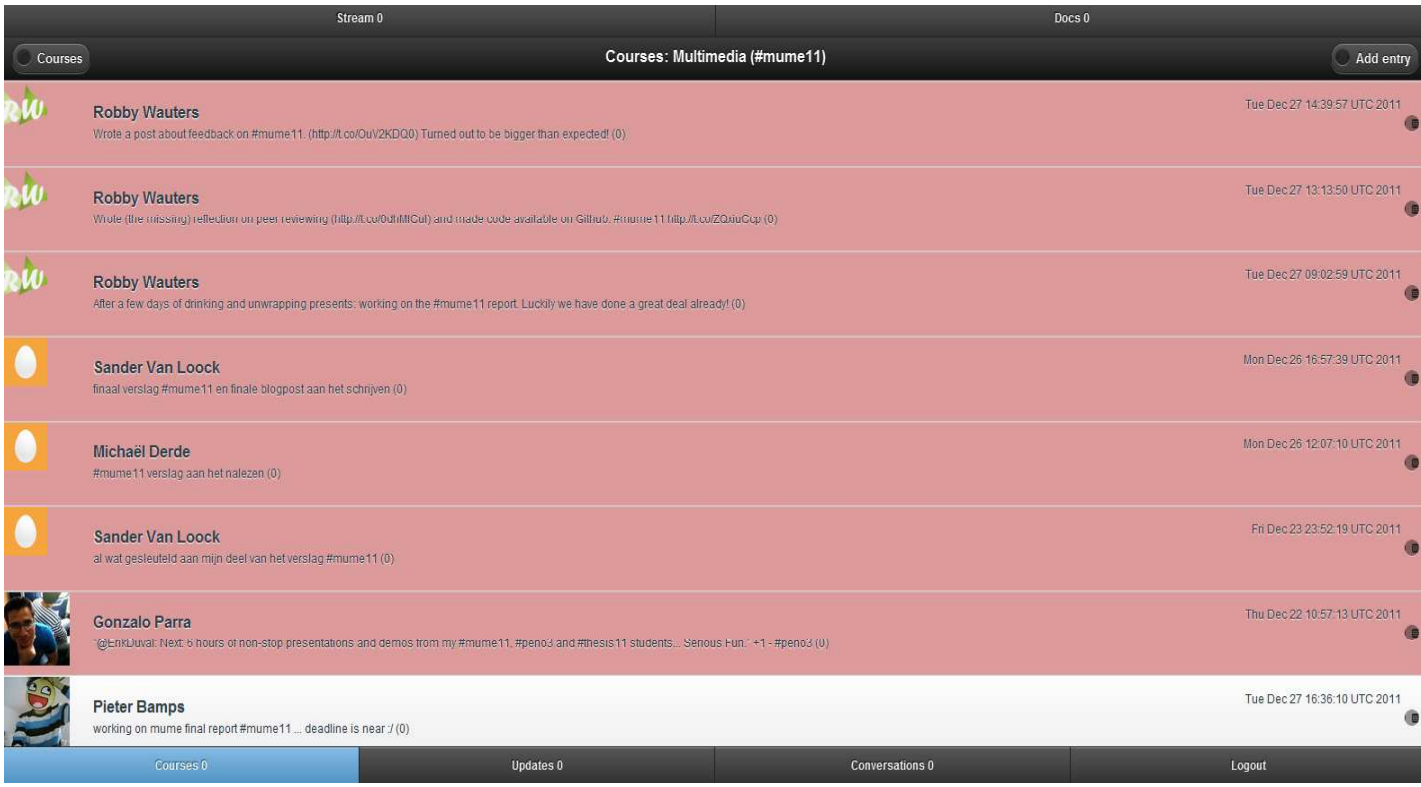
login screen



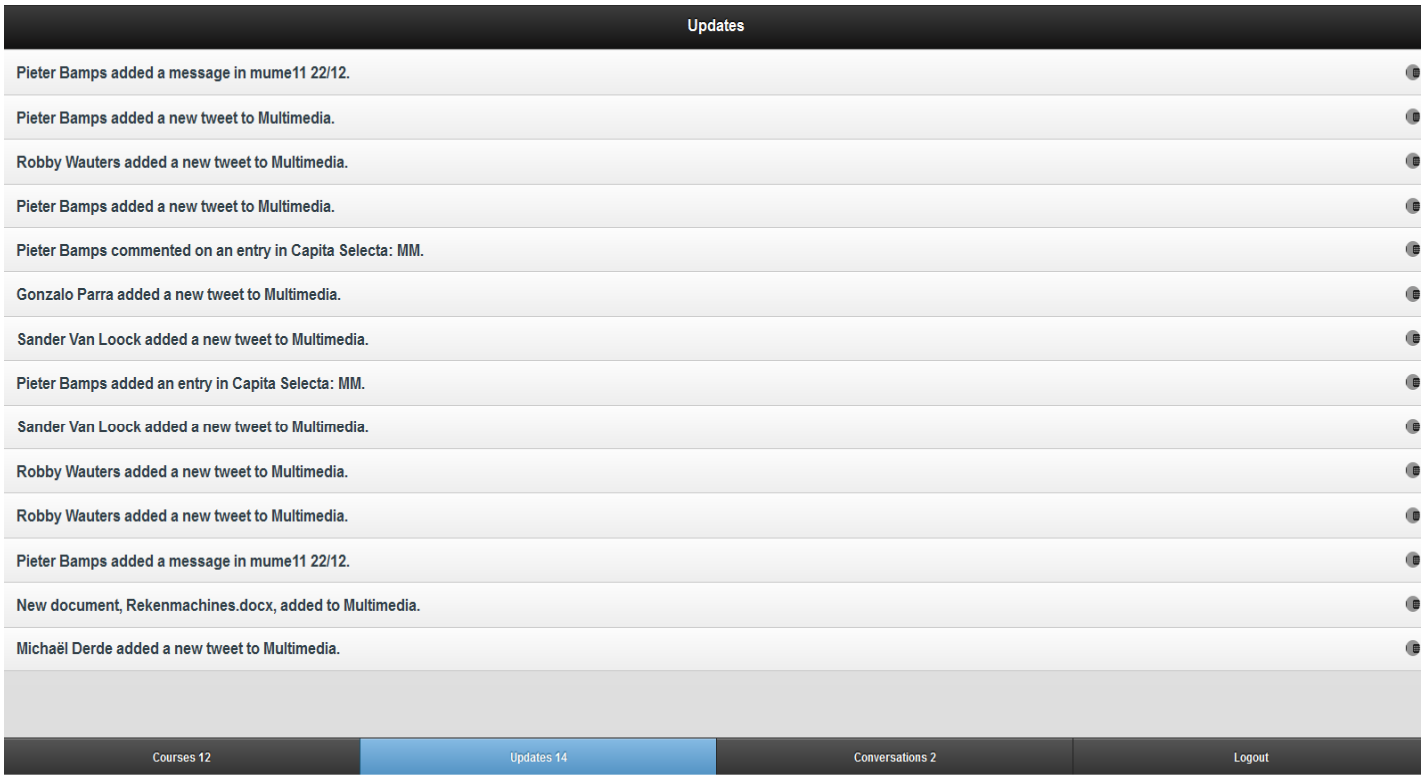
Course selection screen



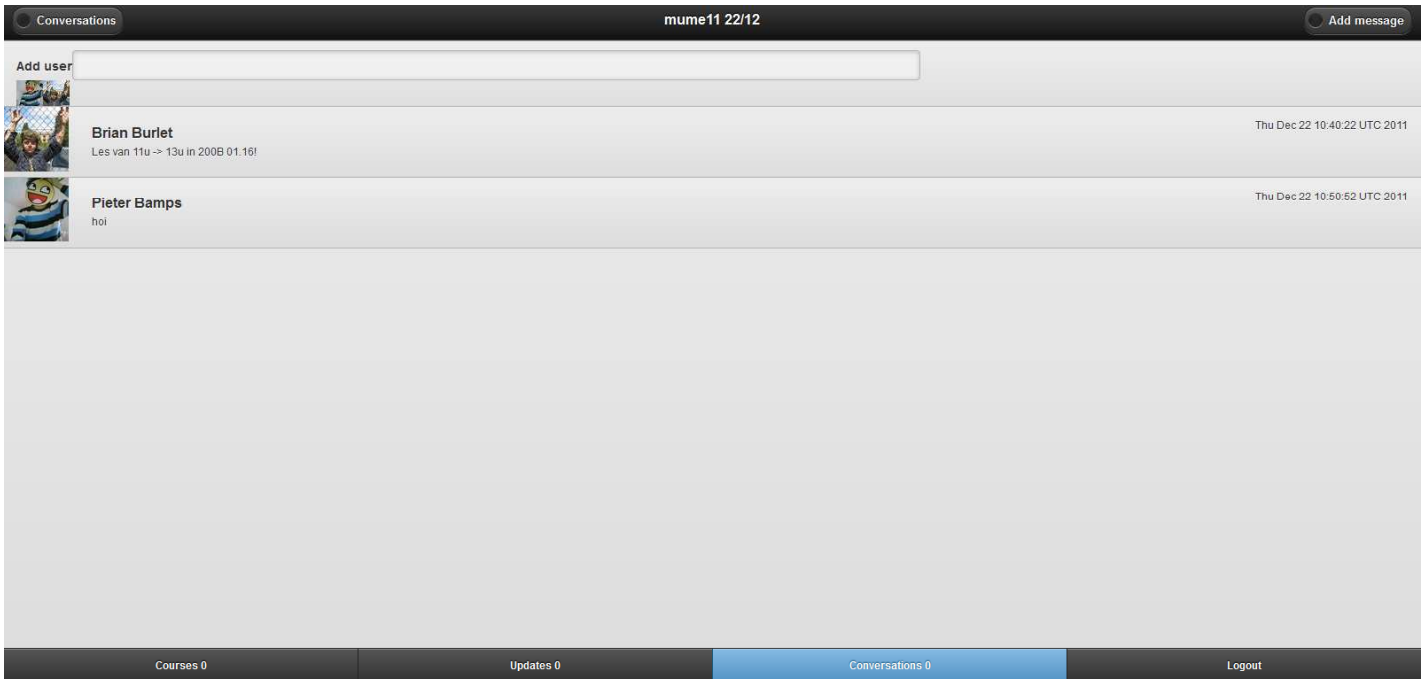
Stream screen



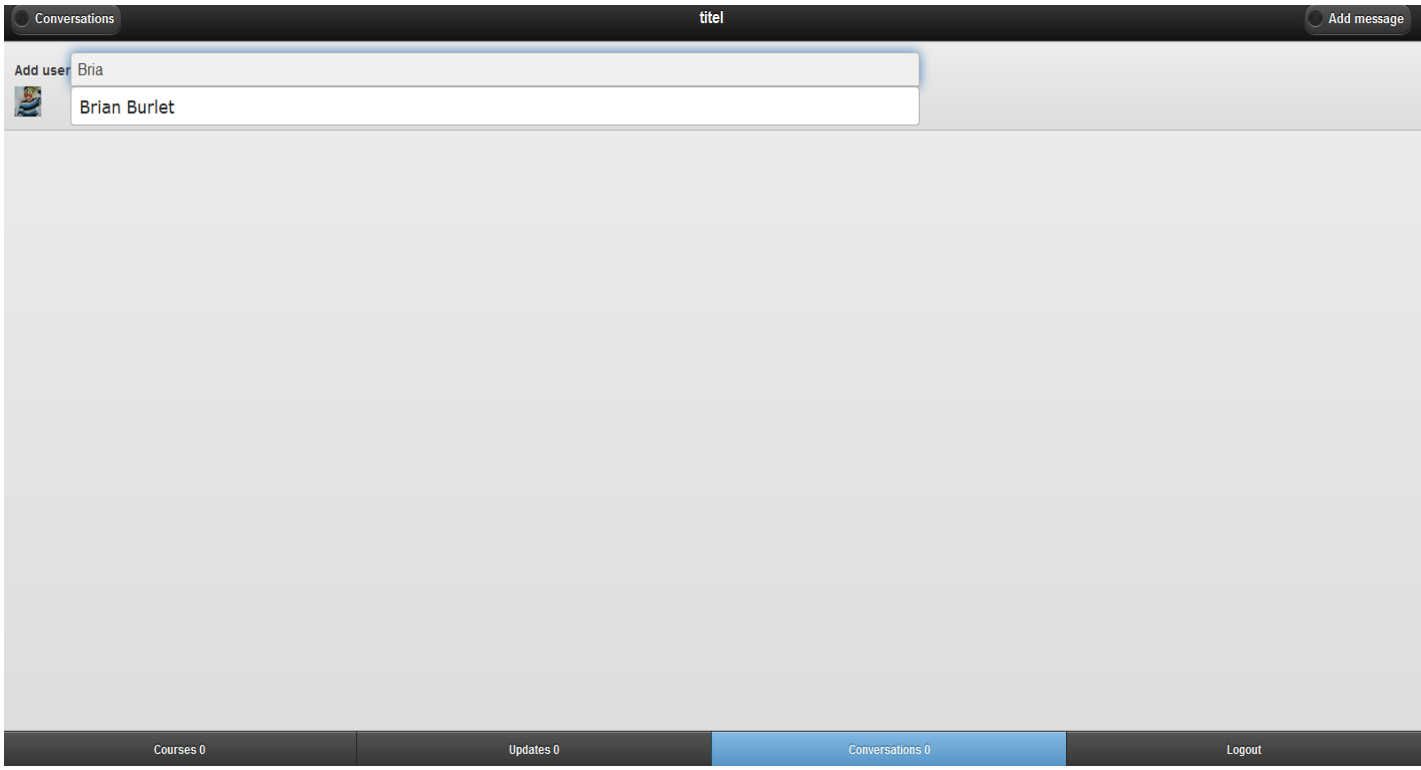
Notification screen



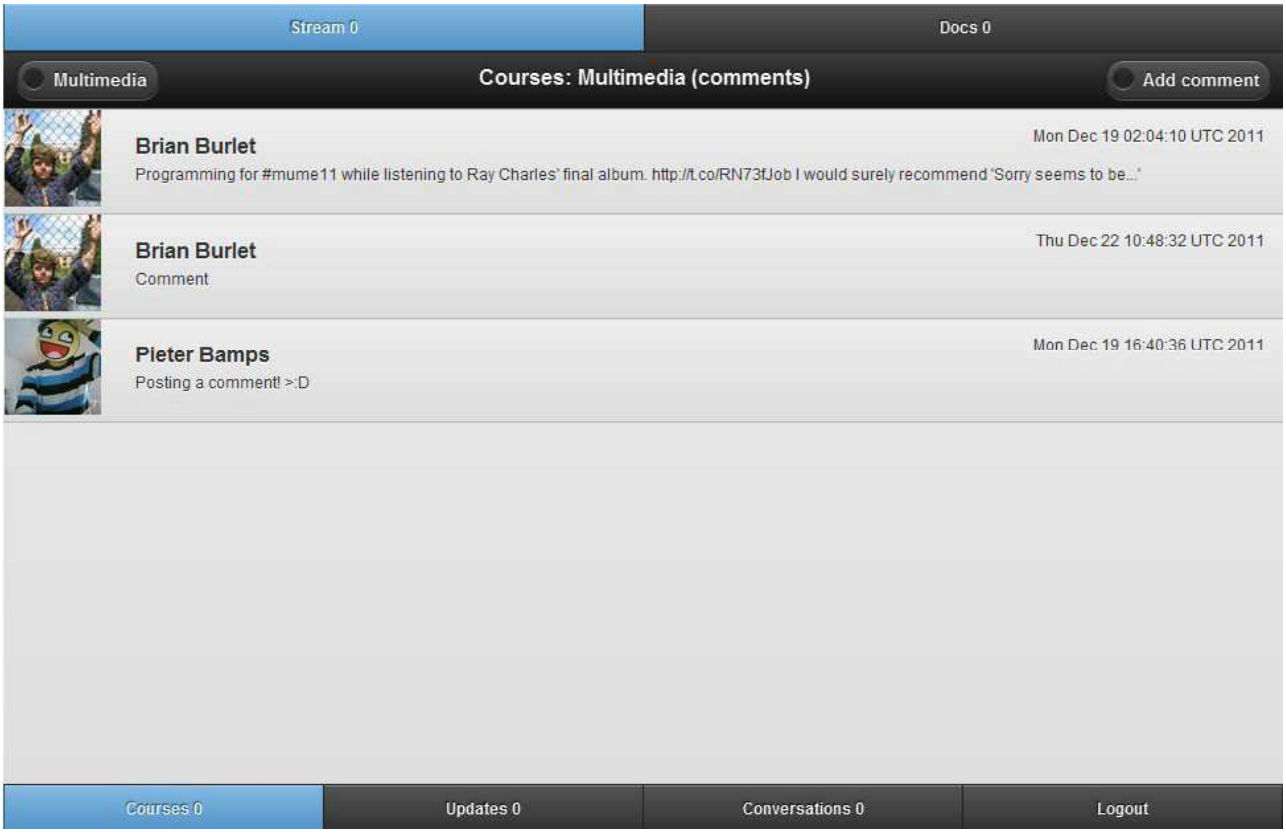
Conversation screen



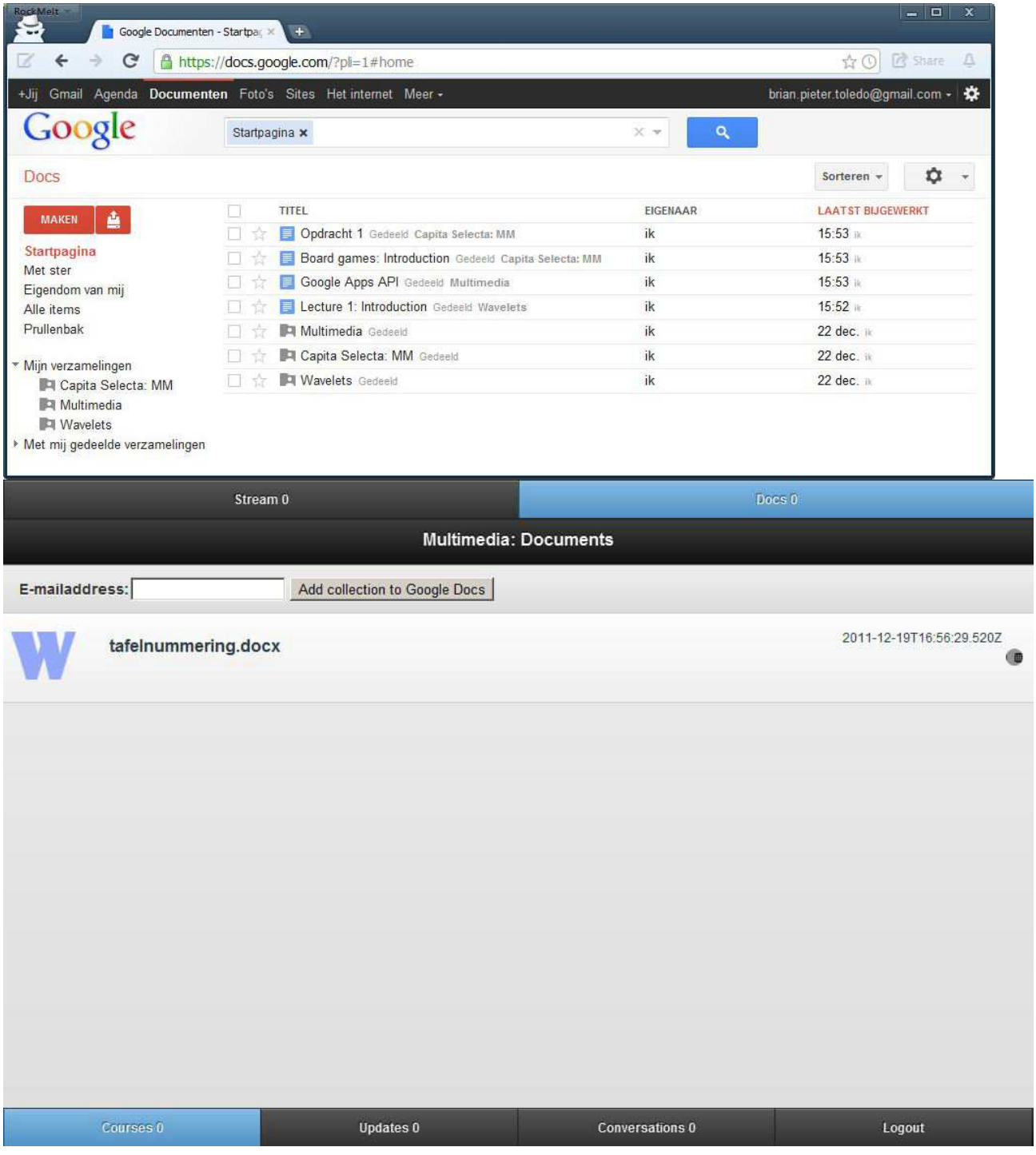
Add user screen:



Comment screen:



Documents screen:



POST (setters)

/add_user
params: password,Twitter_id

/add_course

params: coursename,hashtag

/add_user_to_course
params: user_id,course_id

/add_entry
params: course_id,user_id,text

/add_comment
params: entry_id,user_id,text

/add_conversation
params: user_id,title

/add_user_to_conversation
params: user_id,poster_id,conversation_id

/add_message
params: user_id,text,conversation_id

GET JSON (getters)

get/user/valid_login?username=REQUIRED&password=REQUIRED
respons:

```
{
 "logged_in":1, // kan uiteraard ook 0 zijn in geval dat er iets fout gelopen is
 "user_id":"USER_ID" // store in session
}
```

get/user/courses?user_id=REQUIRED
respons:

```
{
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES"
 ],
 "results":
 [
  "course_id":"COURSE_ID",
  "coursename":"NAME",
  "num_updates":"",
  "hashtag":"HASHTAG"
 ], lijst
}
```

get/user/conversations?user_id=REQUIRED
respons:

```
{
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES"
 ],
 "results":
 [
  "conversation_id":"CNVERSATION_ID",
  "title":"TITLE",
  "fullname":"USER_FULLNAME",
  "numl_updates":"NUM_UPDATES",
  "pic_url":"USER_PIC_URL",
  "hashtag":"HASHTAG"
 ], lijst
}
```

get/user/updates?user_id=REQUIRED

respons:
```
{
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES"
 ],
 "results":
 [
  "course_id":"COURSE_ID",
  "entry_id":"ENTRY_ID",
  "type":"TYPE", // tweet, entry, comment, file, message
  "text":"TEXT", // het bericht
  "reply_to_id":"REPLY_TO_ID", // is leeg in geval van geen comment
  "fullname":"USER_NAME", // gebruiker die het gepost heeft
  "pic_url":"USER_NAME", // gebruiker die het gepost heeft
  "date":"DATE",
  "course_name":"COURSE_NAME"
 ], lijst
}
```

get/course/entries?user_id=REQUIRED&course_id=REQUIRED
respons:
```
{
 "coursename":"COURSENAME", // van de gegeven course_id
 "hashtag":"HASHTAG", // van de gegeven course_id
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES",
  "documents":"NUM_COURSE_DOCUMENT_UPDATES",
  "stream":"NUM_COURSE_STREAM_UPDATES"
 ],
 "results":
 [
  "entry_id":"ENTRY_ID",
  "type":"TYPE", // tweet, entry
  "text":"TEXT", // het bericht
  "fullname":"USER_NAME", // gebruiker die het gepost heeft
  "pic_url":"USER_NAME", // gebruiker die het gepost heeft
  "num_updates":"",
  "date":"DATE",
  "num_replies":"AANTAL_REPLIES"
 ], lijst
}
```

get/entry/comments?user_id=REQUIRED&entry_id=REQUIRED
respons:
```
{
 "text":"TEXT", // van de gegeven entry_id
 "date":"DATE", // van de gegeven entry_id
 "fullname":"FULLNAME", // van de gegeven entry_id
 "pic_url":"PIC_URL", // van de gegeven entry_id
 "coursename":"COURSENAME", // van de gegeven entry_id
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES",
  "documents":"NUM_COURSE_DOCUMENT_UPDATES",
  "stream":"NUM_COURSE_STREAM_UPDATES"
 ],
 "results":
 [
  "text":"TEXT", // het bericht
  "fullname":"USER_NAME", // gebruiker die het gepost heeft
  "pic_url":"USER_NAME", // gebruiker die het gepost heeft
```

```
  "is_update":"", // y in geval van update
  "date":"DATE"
 ], lijst
}
```

get/conversation/messages?user_id=REQUIRED&conversation_id=REQUIRED
respons:

```
{
 "title":"TEXT", // van de gegeven entry_id
 "updates":
 [
  "updates":"NUM_UPDATES",
  "courses":"NUM_COURSE_UPDATES",
  "conversations":"NUM_CONVERSATION_UPDATES"
 ],
 "users":
 [
  "fullname":"USER_NAME", // gebruiker die het gepost heeft
  "pic_url":"USER_NAME" // gebruiker die het gepost heeft
 ], lijst
 "results":
 [
  "text":"TEXT", // het bericht
  "fullname":"USER_NAME", // gebruiker die het gepost heeft
  "pic_url":"USER_NAME", // gebruiker die het gepost heeft
  "is_update":"", // y in geval van update
  "date":"DATE"
 ], lijst
}
```