Fragen

Application URL:	fragen.cmq.me
Project Manager:	Yos Riady
Team members:	
Chen Minqi	A0099314Y
Muhammad Muneer	A0101168R
Wang Boyang	A0078695H
Yos Riady	A0099317U

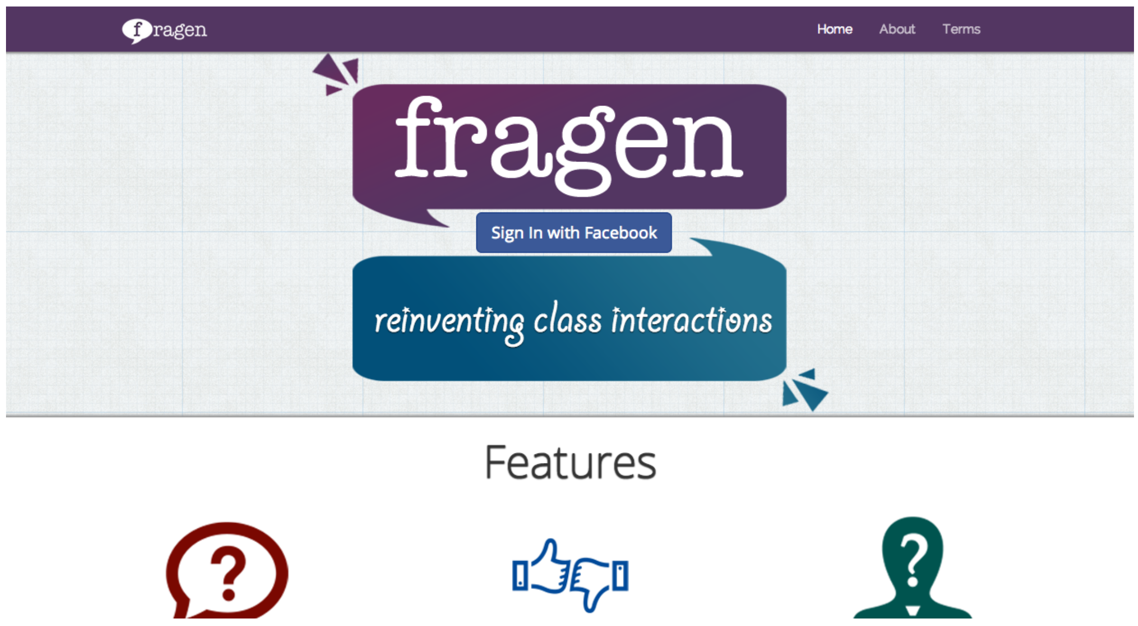
Aspiration 1: Choose to do an IFrame application or a standalone application or both. Choose wisely and justify your choice to us with a short write-up.

We've decided to do a standalone application because of two reasons. Firstly, we realize that having our academic-focused app inside Facebook can distract students and will disrupt the learning process. Instead of paying attention to the questions and answers posted on fragen, there is a high possibility that they will pay attention to Facebook feed instead. Our second reason to do a standalone app is for future extensibility. For example, we may later choose to support authentication methods other than Facebook. OAuth and IVLE API support are two viable options. Being too dependent on Facebook takes away the possibility of extending features such as these.

Aspiration 3: Application Logo

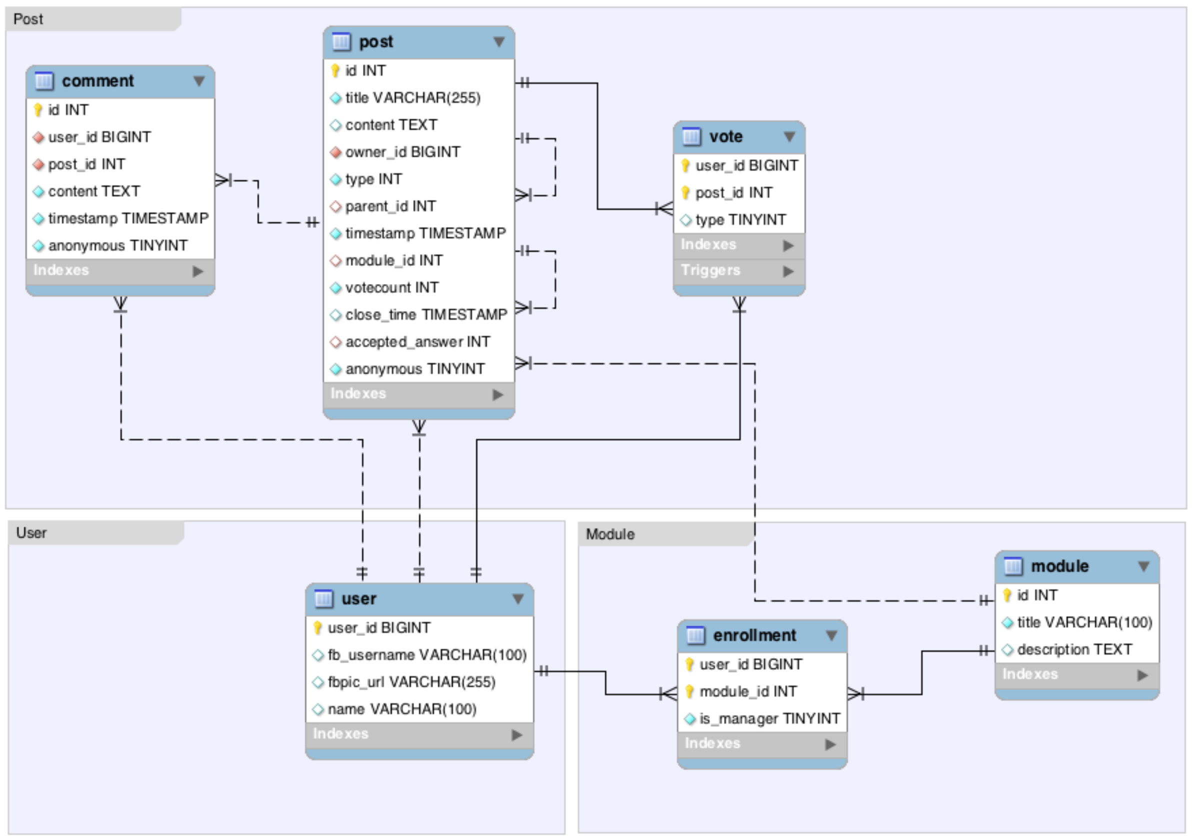
Above is the logo of our application, a speech bubble containing the first letter of our application name. Fragen (fʁaːgən) means 'to ask' in German.

Aspiration 4: Integrate your application with Facebook.



Users of our standalone app can login to Fragen using their Facebook account and see their own name appearing together with their Facebook profile picture. Our application is also closely integrated with Facebook's Open Graph, Graph API, Timeline, and Social Plugins. By using Facebook Login to sign in users with their real identities, calling the Graph API and using Open Graph to tell stories, our users will be more engaged and more likely to return.

Aspiration 5: Draw the database schema of your application.



The questions and answers are generalised as ‘posts’. They are stored in the same table since they have very similar properties. The post table stores the content of the post as well as other metadata such as poster id, date created, whether it is closed, whether it is anonymous and so on. We use a ‘type’ column to distinguish questions from answers. Since there is a one to many relationship between questions and answers, each answer has a ‘parent_id’ column that indicates which question it belongs to. Also, each question may have an ‘accepted_answer’ that indicates which answer is accepted by the question owner.

We then have a comment table to store comments, and a vote table. In the vote table, user_id and post_id forms the primary key, since each user should only cast one vote to one post. We use -1 to flag downvote and +1 to flag upvote. Notice that we also keep track of the vote count in the post table. The consistency between `post`.`votecount` and vote table is ensured by DELETE, UPDATE and INSERT triggers, which will update the votecount column each time a vote is cast.

We use a user table to store each user's Facebook ID as well as some basic information, so we don't need to query Facebook for things like name or profile picture URL every time we need them.

The module table contains a list of all available modules (categories), and each user can be enrolled in multiple modules. This relation is kept in the enrollment table. It also records if a user is manager (perform maintenance actions such as deleting unwanted post) of this module. These can accommodate future feature development that have not been implemented yet.

Answer to Bonus questions

1. What are the pros and cons of each method of visibility control? When should one use the Javascript method and when should one use the PHP method? (1%)

Using JavaScript allows elements to be toggled on the fly without reloading the page, as long as JavaScript is enabled in the browser. However, it can be toggled by the user should they modify the CSS of the hidden element. PHP method doesn't depend on JavaScript and works even if the client browser disables JavaScript, but it would require the page to be reloaded.

JavaScript is the common method for visibility control. It is especially important in situations where we want to avoid page refresh (For example, a user is writing a post. The content of the post will likely be lost if the page is reloaded). However, as highlighted above, JavaScript should not be used for access control, since clients will be able to see what they are not supposed to view. Instead, PHP should be used.

Furthermore, PHP can be used to accompany JavaScript as a fallback, should the client has their browser disable JavaScript.

2. What is the primary key of the home faculties table? (0.5%)

Answer: matric_no

Aspiration 6: Tell us some user queries (at least 3) in your app that needs database access. Provide the actual SQL queries you use and explain how it works.

```
SELECT *
FROM post
WHERE TYPE = 0 -- 0 is the constant value for question
AND module_id = 5
ORDER BY timestamp DESC LIMIT 0,
                              30;
```

This is probably the most frequently used query, since it is used to retrieve a list of questions from a particular module (in this case module ID is 5). It sorts the result in reverse chronological order, with latest question on top. This can be easily changed to `ORDER BY votecount DESC` so we have the highest voted question on top. It also supports paging of results, in this case it fetch the first 30 questions for display.

```
UPDATE post
SET accepted_answer = 30
WHERE id= -- we need to find the parent question ...
    (SELECT parent_id
     FROM
       (SELECT * -- strange MySQL limitation
        FROM post) p
     WHERE id = 30);
```

This is the query we will use to allow the question owner to accept an answer. Provided the ID of the answer we want to accept, in this case answer ID is 30, this query needs to find the parent (the question) it belongs to, and update that entry to reflect the accepted answer.

In this query, we use a subquery to figure out the parent question, but the interesting part is the `(SELECT parent_id FROM (SELECT * FROM post)` part. It is a redundant sub-subquery since it is just equivalent to `(SELECT parent_id FROM post)`. This is a workaround of a strange limitation of MySQL, as documented together with the UPDATE syntax: "currently, you cannot update a table and select from the same table in a subquery".

```
CREATE TRIGGER `vote_AUPD` AFTER UPDATE ON vote
FOR EACH ROW
UPDATE post p
SET p.votecount=p.votecount + NEW.type - OLD.type
WHERE p.id = NEW.post_id
```

This is not really a query, but a post update trigger. In our database, we store each vote cast as a triple: (user_id, post_id, type) where 1 indicates an upvote and -1 indicates an downvote. We also keep track of the net vote count for each post in the post table. This trigger is one of three triggers (the other two being pre-delete and post-insert) we use to maintain consistency between vote records and vote count.

Whenever the vote record is changed, we subtract the old vote type from the vote count, and adds the new one. The constant values chosen for upvote and downvote makes this calculation very convenient. For example when we change a downvote to upvote, we subtract -1 and add 1, resulting in an net increase of 1 - (-1) = 2 votes.

Aspiration 7: Show us some of your most interesting Graph or FQL queries. Explain what they are used for. (2-3 examples)

```
SELECT uid, username, name, pic_square FROM user WHERE uid in(SELECT uid2 FROM friend WHERE uid1 = me() LIMIT 0,100)
```

This FQL query is responsible for retrieving basic profile information such as Facebook display name and profile picture of all the users' friends. You can fetch data from one query and use it in another query within the same call. The WHERE clause is optional in the latter query, since it references data that's already been fetched. We first select all the ids that are paired up with our users' id. We use a friend table that can be used to return a list of a user's friends or to show whether two users are friends. We then find users with those ids, and retrieve uid, username, name, and pic_square information from the Graph API. We can also limit the number of rows returned by specifying the LIMIT parameter.

```
SELECT uid, username, name, pic_square FROM user WHERE uid=me()
```

This FQL query returns the specified columns of a Facebook user's profile information, whose facebook id is equal to uid. In this case, we return the public information of the currently logged in user to retrieve and store her picture URL and other information into the database for later use.

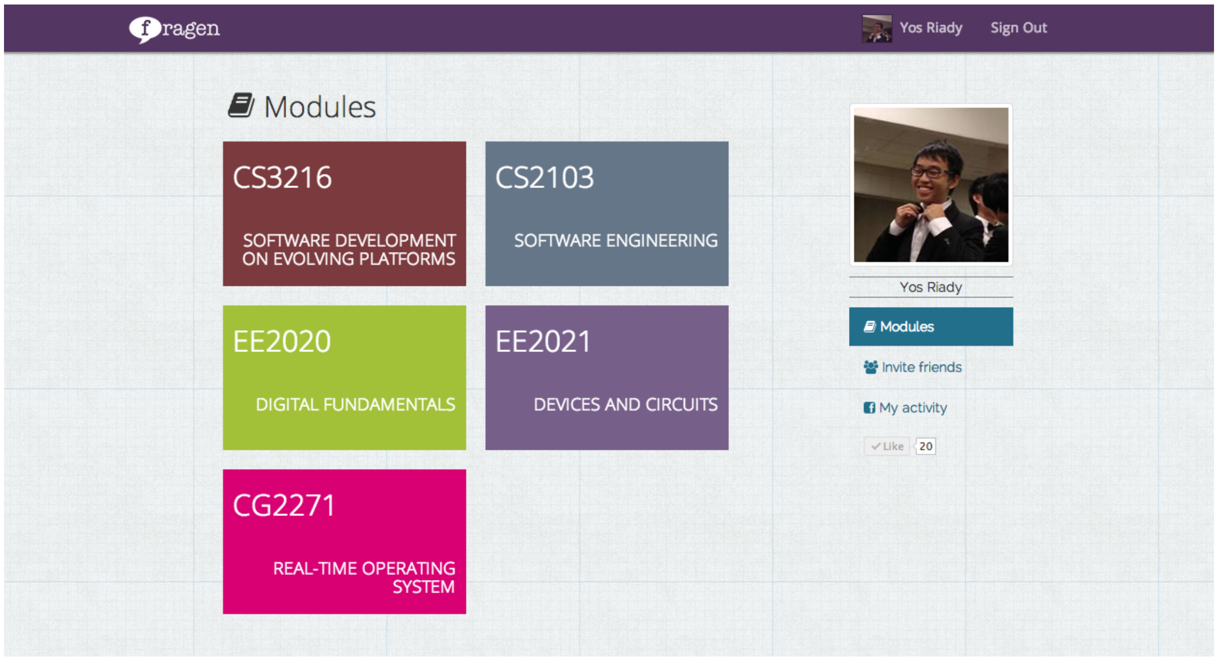
Aspiration 8: We want some feeds! BUT remember to put thought into this. Nuisance feeds will not only earn you no credit but may incur penalization!

We use feeds for the most important system messages such as user invitations because we feel that too much feed integration can be an annoyance to the user. There is a risk that because of too many feed messages, users will choose to filter out messages from Fragen. Thus, for event-related messages such as "John posted a question on Fragen!"

or “Colin answered a question on Fragen!” we have decided to use Open Graph Actions and Objects instead, which are better integrated to the Facebook Timeline and at the less intrusive. By default these feeds can be found in the “recent activity” column, which is generally not as publicly visible. If the user wishes to make her activity feed more prominent, she can do so by adding the Fragen activity feed in the ‘More’ section of their Facebook Timeline.

Aspiration 9: Your application should include the Like button for your users to click on. Convince us why you think that is the best place you should place the button.

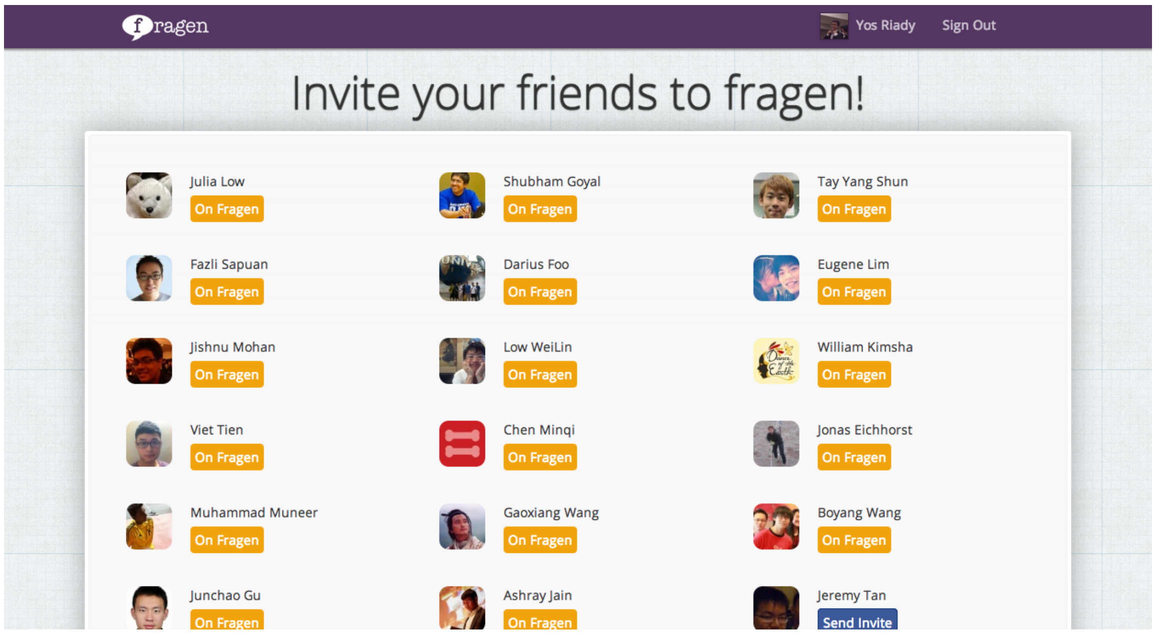
We have a few options of where to place the like button, namely the main page, the dashboard, and individual module pages. First, it made little sense for us to place it on the main page because of its low visibility. Users spend the least amount of time on the welcome page. As for individual module pages, the current compact layout makes it such that it is no room for the like button. We can choose to put it there anyway, but to the detriment of the app’s core functionality and user experience.



Thus, we’ve decided to place the Like button in the dashboard below the invite friends link. The dashboard is our highest traffic page, since all users are redirected to the dashboard upon login. The current position of the like button makes it highly visible because it is above the fold on the page, visible without scrolling. Furthermore, it is conveniently grouped together with other Facebook- integrated features, making it the natural choice.

Aspiration 10: First, include the feature to let users invite their friends to use your application. Next, create a page showing either (a) all friends who are users of your application; or (b) all friends (or some sensible subset) who are not users of your application.

It’s worth nothing that Facebook no longer allows friend invitations from standalone apps without supplying a canvas URL (February 2013 Breaking Changes). Since it does not make sense to supply canvas URLs for a standalone app, we instead use the Facebook Feed Dialog functionality to invite friends.



Invite your friends to fragen!

Julia Low — On Fragen	Shubham Goyal — On Fragen	Tay Yang Shun — On Fragen
Fazli Sapuan — On Fragen	Darius Foo — On Fragen	Eugene Lim — On Fragen
Jishnu Mohan — On Fragen	Low WeiLin — On Fragen	William Kimsha — On Fragen
Viet Tien — On Fragen	Chen Minqi — On Fragen	Jonas Eichhorst — On Fragen
Muhammad Muneer — On Fragen	Gaoxiang Wang — On Fragen	Boyang Wang — On Fragen
Junchao Gu — On Fragen	Ashray Jain — On Fragen	Jeremy Tan — Send Invite

On our application, users can invite their facebook friends to Fragen by going to 'Invite friends' from the dashboard. Here, users can view their friends and post invite messages to their friends' wall through the Feed Dialog. On this invite page, users can see all friends who are users of our application and all friends who are not users of our application.

We visually highlight who amongst the users' facebook friends are already users of Fragen apart from friends who are not users of Fragen. This way, users can more easily invite their friends who have yet to use our application.

Aspiration 11: Explain how you handle a user's data when she removes your application. Convince us that your approach is necessary and that it is the most logical. Do also include an explanation on whether you violate Facebook's terms and condition.

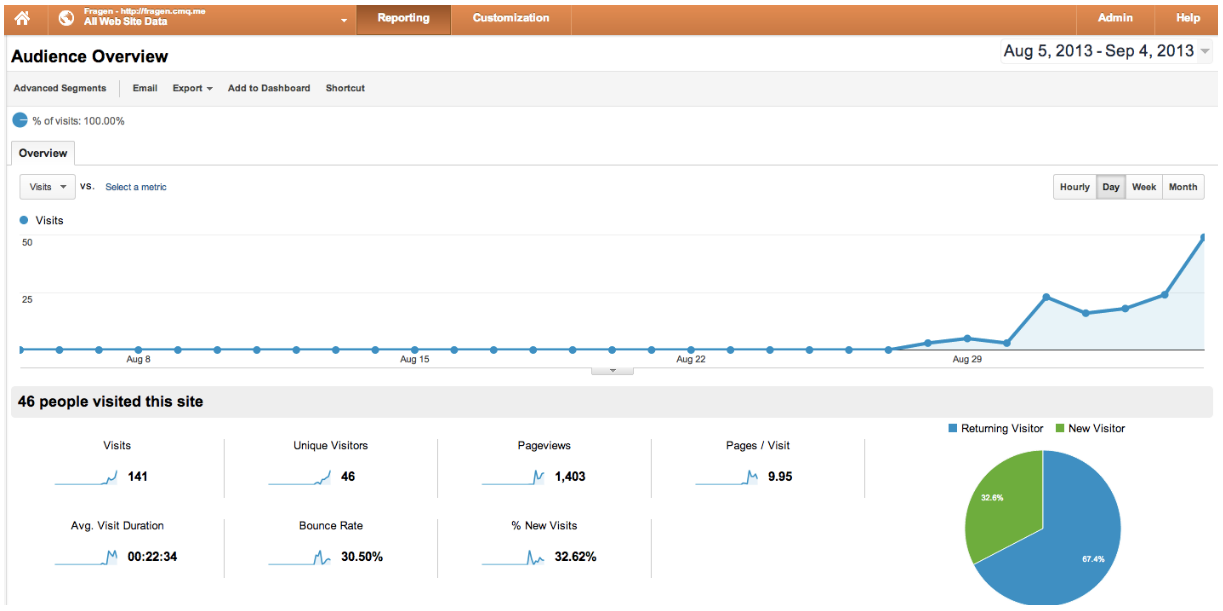
We only store basic information about a user, such as their name and profile picture link. This is for performance concerns. Caching this data allows us to fetch them directly without querying Facebook via API every time, a major performance and user experience improvement: you won't have to wait a few seconds for jQuery to return the name of the users.

According to the Platform Policy (developers.facebook.com/policy/), this is allowed. "You may cache data you receive through use of the Facebook API in order to improve your application's user experience, but you should try to keep the data up to date." To fulfill this requirement, we update the cache on each user login.

When a user removes our application, we do not delete those information from our database. Since we rely on user's basic information for post attribution, removing their names from our database will make all posts made by them become nameless. This is not only undesirable from a user experience point of view, but there could also have legal complications. The users are copyright holders of the content they posted, so we at least should attribute the posts to them.

This does not violate the Terms and Conditions of Facebook, because according to Data Use Policy, “if you've removed an application and want them to delete the information you've already shared with them, you should contact the application and ask them to delete it”. If a user wish to have their data removed, they should contact us (our contact information is available on the front page of our app), and we would delete their data together with all their posts.

Aspiration 12: Embed Google Analytics on all your pages and give us a screenshot of the report. Note that this means you have to install Analytics at least 48 hours before submission deadline as Analytics only updates the report once per day.



Audience Overview

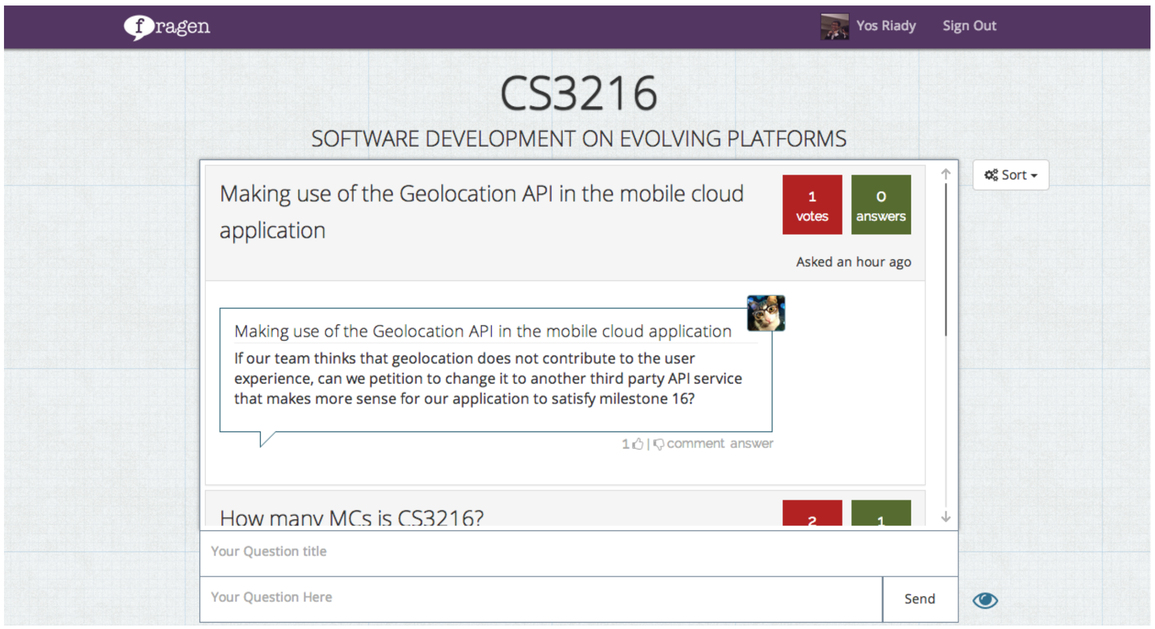
Aug 5, 2013 - Sep 4, 2013

% of visits: 100.00%

46 people visited this site

Visits 141
Unique Visitors 46
Pageviews 1,403
Pages / Visit 9.95
Avg. Visit Duration 00:22:34
Bounce Rate 30.50%
% New Visits 32.62%

Aspiration 13: Describe 2-3 user interactions in your application and show us that you have thought through those interactions. It would be great if you could also describe other alternatives that you decided to discard, if any.



Making use of the Geolocation API in the mobile cloud application

Making use of the Geolocation API in the mobile cloud application

If our team thinks that geolocation does not contribute to the user experience, can we petition to change it to another third party API service that makes more sense for our application to satisfy milestone 16?

How many MCs is CS3216?

Only <u>ONE</u> textbox

One of the main functionalities of our app allows users to take part in the discussion within a module by posting questions, commenting and answering on a post. Initially we achieved this by having different textboxes that served its own purpose. As a result, the site seemed clouded with textboxes and was not aesthetically pleasing. In addition, when we tested with few users, they were confused about the functionality of each textbox and could not distinguish between the textboxes. So we changed to a new layout where there is only one textbox which changes dynamically when user wants to carry out different operations.

By default, the textbox is set for asking questions. But when a user clicks on 'answer' or 'comment', the textbox changes accordingly, with the title area fading out. This new layout has been successful, as seen from positive feedbacks from our users.

Questions

Any new user of the application is able to recognise the similarity between Fragen and other popular Q&A sites such as stackoverflow. In stackoverflow and other popular Q&A sites, to view a question one has to open it in a new tab. It gets frustrating when you want to follow few questions as many tabs will be opened on the browser.

To make sure, our users don't face similar problems. We have styled our questions as collapsible elements. Users can simply click on the question to view it and click again to hide it. During testing, users are delighted that they are not redirected to a new tab when they click a question and they are able to navigate through the questions without us telling them what to do.

Chat-like Interface

Chatting with friends in platforms such as skype and facebook has become a norm and so students are very used to the chat interfaces that we see on these platforms. Hence, we have designed our interface to look a chat. We feel this layout will encourage students to ask more questions.

Aspiration 14: Show us an interesting DOM manipulation through JQuery that occurs in your application.

We make substantial use of JQuery as it helps us "write less, do more". The following are a few examples.

1. Since most of our contents are added dynamically and in real-time from server, we append all these DOM from our script. To store id and relevant info that need to be used by client-side script, we made use of "data-" attributes, for example:

```
var masterPostDiv = $('<div class="masterPostDiv" data-timestamp="' +
data.timestamp + '" data-msgid="' + data.id + '">');
```

Later, when there is a need to get hold of the data, we do something like:

```
var target = $('.masterPostDiv[data-msgid="' + postData.parent_id + '"]
.answersDiv .answerDiv[data-msgid="' + postData.id + '"] .ansVoteDiv
span.votes');
```

2. We sort the questions and answers using this JQuery plugin called tinysort (http://tinysort.sjeiti.com/). It works well and is lightweight. Sample code:

```
$(".messageBoard .masterPostDiv .answersDiv .answerDiv").tsort('.ansVoteDiv
span.votes', {order: 'desc'}, {order: 'desc', attr: 'data-msgid'});
```

3. We achieve the blink effect using JQuery fadeIn() fadeOut() function:

```
function myBlink($obj) {
      $obj.fadeOut(500);
      $obj.fadeIn(1000);
}
```

All data in the above string concatenation do not involve user supplied content, so there is no worry of injection.

Aspiration 15: Describe at least one Action and Object that you have created for your application and why you think it will create an engaging experience for the users of your app

The core Object we have is ‘Question’, as it is the fundamental concept in our our application, and almost all user actions interact with it. Users either ‘Answer’ or ‘Ask’ a question most of the time, so we have defined ‘Answer a Question’ and ‘Ask a Question’ for our Open Graph stories. When a user asks or answers a question, a story would be created in recent activity on his profile. This is very similar to how Quora uses Open Graph stories.

Whenever a ‘ask a question’ is published, all the user’s friends will be able to see it. This drastically increase the coverage as compared to only displaying the question in our application. There is a much higher chance for someone that is able to answer the question to come to help. An ‘answer a question’ story, on the other hand, is not only a ‘bragging’ opportunity for the user, but also sends out an impression that he would be able to answer such questions, should his peers need help.

Aspiration 16: Describe how you have integrated with Timeline using Open Graph Collections.

Open Graph Collection is one step further than the Open Graph stories. We display users’ recent activities on Fragen, including both question asked and answers provided. Users would be able to add this section to their Facebook profile by clicking ‘My Activities’ on the dashboard and show the world (depends on the privacy settings, of course) about their activities.

Aspiration 17: Describe 2 to 3 uses of animations in your application. Explain what kind of value do they add to the context to which they are applied. Highlight your more innovative animations. We would be interested to know! (Optional)

One animation we used to improve user experience is a blink/ fading in of newly created questions, answers, and comments. Whenever a user posts new content to a module's fragen page, the animation draws other user's attention to that element and make it clear that it is new. This creates a nice visual feedback and help users spot new content, especially when new answer or comment are added to a question. This animation has become a standard in the industry, and Facebook, Twitter, Hush are all using it in their user interface.

We've also used an external jQuery animation plugin, Typer, which is used on our main index page to help illustrate what fragen is all about, question and answers. It adds value to our application by visually introducing users to Fragen's core functionality on our welcome page.

Aspiration 18: Describe any cool utilization of AJAX in your application. (Optional)

We've made some careful considerations of when and for what purpose to use AJAX and WebSockets.

Most of the real-time messaging events in Fragen is not done via AJAX calls because it is not a suitable mode of transport. Instead, our application is relying heavily on WebSockets to ensure real time updates of information. All contents are updated in real time: new questions and answers, comments, and vote count are broadcasted to all clients. For example, when a user posts a new question on Fragen, other users with the page open will receive that question in real time. This is true for all content updates. Cool!

Like AJAX, all this is done without the user having to refresh the page. The advantage of using Websockets over AJAX is since the connection between the client and the server stays open, the server can send messages to the client without the client asking for anything which is impossible with Ajax.

Having said that, AJAX gives web browsers the ability to fetch data from the server on demand. For our application, we use AJAX as the transport mechanism for data from our server to display a read-only version of individual questions for public sharing on social networks such as Facebook and the Facebook Timeline.

Aspiration 19: Tell us how you have made use of any existing JQuery plugins to enhance your application functionality. (Optional)

We've used a number of external jQuery plugins to both extend our application's functionality and enhance the overall user experience.

Typer.js
We are using this plugin to add 'typing' animation in the welcome page. This adds value to our application by giving our users an illustration of the purpose of our application.

Metro.js
We styled the modules 'boxes' to look like the Windows 8 Metro tiles. For a further Metro feel, we are using this metro plugin to give the Metro-like animation when a user clicks on the module tiles.

Malihu’s Custom Scroll Bar

We are using this plugin to add styling to the scroll bar that appears within the discussion chatbox. More importantly, we have used this plugin to add slight inertia to the scrolling. Since every collapsible element(questions) is relatively big and the size of its container varies for different devices, scrolling down without any inertia skips some questions. So we have added a slight inertia, to solve this problem and enhance the user experience.

Tiny Sort

TinySort is a small jQuery plugin that will sort any nodetype by it's text-, attribute- or data value, or by that of one of it's children. It’s lightweight and powerful. We use it to provide a flexible sorting functionality where users get to choose from two sorting schemes.

qTip

qTip is the second generation of the advanced tooltip JQuery plugin. We chose to use it instead of JqueryUI due to compatibility issues between bootstrap and JqueryUI, and the fact that qTip is much lighter.