CS3216 Final Assignment Report Submission

Larry Huang Lie Jun | Ng Xu Jie | Ken Oung | Patrick Cho

Milestone 1

Choose to do a Facebook Canvas application, a standalone application, or both. Choose wisely and justify your choice with a short write-up.

We have chosen to do a standalone application for a couple of reasons.

First and foremost, it was rather unnecessary to have a Facebook Canvas application given that our platform can obtain what it requires from the Facebook Graph API. The full Facebook Canvas features while comprehensive, were beyond the scope of what we needed and therefore unnecessary.

Using a standalone application also provides us with greater control and autonomy over our application. For instance, Canvas applications can only use Facebook-approved advertisers. Facebook's own ads are also shown on the sidebar, and can at times clutter the look and feel of the page. In contrast, using a standalone application will provide us with much more flexibility to customize the look and feel of our page.

Interestingly, "unbundling the big blue app" is something that Mark Zuckerberg has spoken about in his interview with New York Times in 2014. Applications like Messenger are separated from the Facebook platform to give users greater control of their notifications. More importantly, Zuckerberg thinks that in mobile, there is a lot of value in "creating single-purpose first-class experience". Likewise, having a standalone application will allow us unbundle it from Facebook. In doing so, it provides us with the full ability to control virtually every aspect of our application and enables our users to be in more control of their experience.

Last but not least, having a standalone application grants us the flexibility to scale in the future. Unlike Facebook Canvas applications, a stand-alone application can be better integrated with other platforms and it is easier to pull data from sites and social networks other than Facebook. Furthermore, Facebook Canvas applications necessitate the user to have a Facebook account. In the event that our target market does not utilize Facebook as its main social network (think China), our standalone application can then include other forms of login.

In sum, we went ahead for the standalone application after a consideration and evaluation of both approaches' strengths and weaknesses. Even though Facebook Canvas applications might integrate more seamlessly with Facebook, having a standalone application provides us with the control and flexibility to scale and plan for future growth and usage.

Milestone 2

Explain your choice of toolset and what alternatives you have considered for your Facebook application on both the client-side and server-side. If you have decided to go with the vanilla approaches (not using libraries/frameworks for the core application), do justify your decisions too.

***Server-side Choices**: Node.js, Express, MySQL*

Node.js vs PHP

Node.js	Criteria	PHP
One of our group members had used Node.js before in his previous projects, allowing us to do more iterations if we chose Node.js.	Familiarity	While our research showed that PHP was relatively easy to pick up, given the tight deadline and Prof Ben Leong's advice of 'CS3216 not being about learning frameworks', we decided that Node.js might be the better option.
Node's Node Package Manager (NPM) consists of many useful packages such as moment for datetime manipulation and express-session for managing sessions.	Libraries	Although we are less familiar with PHP's Compose project, our research showed that PHP has a smaller active repository.
Node.js queries are asynchronous, allowing other commands to be executed while the query continues (e.g. if an update to a db is not immediately used, we can render the response first). Hence, I/O is non-blocking.	Input/Output Blocking	PHP database queries are commonly I/O blocking. This leads to slower responses.

Abundant 3rd Party Libraries (e.g. Passport.js for login) that are used extensively.	Facebook Integration	Integration done by Facebook itself. So documentation for Facebook integration with PHP is the best as compared to other server-side languages.

In the end, we decided to go with Node.js for the simple reason that we were more confident of implementing a web app on Node.js within 3 weeks. Indeed, the large amount of resources on Node.js and our previous experience with Node helped us greatly in the completion of this project.

MySQL vs MongoDB

MySQL	**Criteria**	**MongoDB**
While we used Bookshelf.js for simpler MySQL queries, more complex queries (which required joining tables on two columns) had to be constructed on Knex, which was lengthy and prone to error (this is a limitation of Bookshelf).	Ease of Constructing Queries	Constructing queries (such as for notifications) might be even more complex without denormalization (since cannot join table in NoSQL). Storing all notifications on user documents would, on the other hand, require many update queries, which is also undesirable.
Usage of MySQL with Node requires a complex object-relational mapping (ORM) in order to translate code to relational database queries.	Integration with Node.js	MongoDB documents are BSONs which are very similar to JSONs and therefore integrate better with Node.
Ensuring data integrity is critical on our platform. For instance, every single want needs to be associated with a unique user and a unique item. Data Integrity can be ensured by the MySQL schema.	Ensuring Data Integrity	Harder to ensure data integrity given the lack of rules enforced by schema.
Changing database structure is much more difficult to do in MySQL, as we have found during the course of this project,	Add / Alter Tables	No need to specify a document design or even a collection up-front. Adding new data to documents is simple.

due to the strict rules enforced in relational databases.		
In MySQL, we can normalize to reduce data redundancy. For instance, every 'want' only needs to store its corresponding userID and itemID without specifying the user's or the item's details.	Denormalization Requirements	For fast queries, it is recommended on MongoDB to store all relevant information about a document in the document itself (no such thing as join documents). Hence, if many users want the same item, each user document would contain the same information about that item. Doing so would cause a lot of duplicate information. If the item were to be updated, all these user documents would also have to be updated.

Client-side Choices: *Bootstrap*

Criteria	**Bootstrap**
Universality of use	Responsive Design Bootstrap is an adaptable, highly customisable library that primarily serves to deliver a responsive mobile web experience. This means that by tapping on the Bootstrap library and its wide open-source community online, we can quickly implement a web application that is portable and usable between devices of different screen sizes. This was one of our priorities when we designed Give For Free. We want our users to have minimal resistance in grabbing a photo of their unwanted items and getting them online. Meanwhile, we also want them to be able to casually browse free items on our website anywhere and anytime they want.
Ease of implementation	Bootstrap is relatively easy to use. The built-in components and scripts are sufficient to kickstart a good website without much issues. Considering the time we had for this project, Bootstrap may not necessarily be the most powerful option, but it is the wisest one.
Consistency	Bootstrap is consistent. This is an important factor that affected our choice on the front-end. We believe that a consistent user experience is key to guiding the user along the usage of our application. We made use of repeating design patterns such as panels, thumbnails and navbars to ensure that our pages and

	views are self-explanatory and intuitive to our users on the first look. Besides, Bootstrap offers components that are used by many web applications. The consistency does not only apply internally through our web application, but also externally across different popular sites.

Milestone 3

Give your newborn application some love. Go to the App Details page and fill the page with as much appropriate information as you can. And yes, we expect a nice application icon!

App details are filled in already.

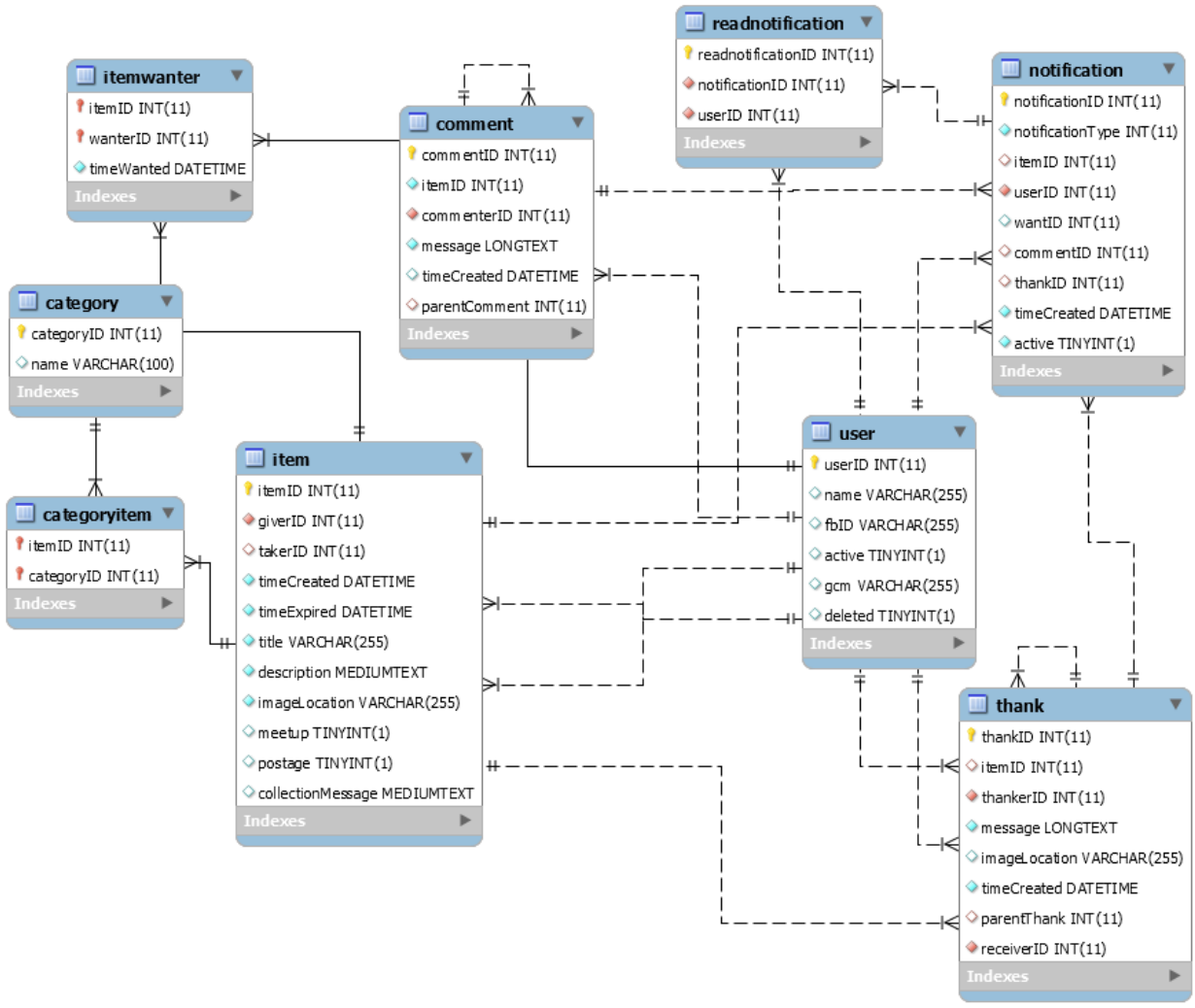
Milestone 4

Integrate your application with Facebook. If you are developing a Facebook canvas app, then users should be able to visit your app and at least see their name (retrieved using the API) on the page. Similarly, if you are developing a standalone app, users should be able to login to your app using their Facebook account and see their own name appearing.

See: https://giveforfree.sg

Milestone 5

Draw the database schema of your application.



Milestone 6

Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use and explain how it works.

SQL Query 1:

```
SELECT i.itemID, i.timeExpired, i.imageLocation, i.title, i.takerID,
i.description, i.giverID, i.meetup, i.postage, i.collectionMessage,
u.name, u.userID, u.fbID, t.name as takerName, t.userID as takerId,
COUNT(iw.wanterID) as numWants,
COUNT(DISTINCT iwu.itemID) as meWant
FROM item AS i
```

```
LEFT JOIN user AS u
ON u.userID = i.giverID
LEFT JOIN user AS t
ON t.userID = i.takerID
LEFT JOIN itemWanter as iw
ON iw.itemID = i.itemID
LEFT JOIN itemWanter as iwu
ON iwu.itemID = i.itemID AND iwu.wanterID = <insert user id>
WHERE i.itemID = <insert item id>
LIMIT 1;
```

The above query is executed on the item page. On the item page, we want to know more details about the item giver as well as the item taker. Hence, we left join the users table twice according to both the 'giverID' and the 'takerID'. At the same time, we want to determine whether the current user of the platform has liked the item. This is done by left joining with the itemWanter table on two conditions: the 'itemID' being the same and the 'wanterID' being that of the current user. We count the number of distinct 'itemIDs' that arise from this left join. If this value is 1, the current user wants the item. If this value is 0, the current user does not want the item. Lastly, we also want to find out how popular the item is. This is done by left joining the itemWanter table on a single condition: the 'itemID' being the same. By counting the number of 'wanterIDs' that arise from this left join, we are able to get the total number of wants for this item.

SQL Query 2:

```
SELECT i.itemID, i.imageLocation, i.title, i.description,
u.name, u.userID, u.fbID,
COUNT(iw.itemID) AS numWants, COUNT(DISTINCT iwu.itemID) as meWant
FROM item AS i
LEFT JOIN user AS u
ON u.userID = i.giverID
LEFT JOIN itemWanter AS iw
ON iw.itemID = i.itemID
LEFT JOIN itemWanter AS iwu
ON iwu.itemID = i.itemID AND iwu.wanterID = <insert user id>
WHERE i.giverID != <insert user id>
AND i.itemID < <insert smallest itemID seen so far>
AND i.timeExpired > NOW()
AND i.takerID IS NULL
GROUP BY i.itemID
ORDER BY i.itemID DESC
LIMIT <insert number of items to retrieve>;
```

The above query is executed on the home page. On the home page, an AJAX POST request is sent everytime the scroll reaches the end. When this happens, this query is run to fetch the next *n* items to be shown. Similar to the previous query, we want to know more information about the giver so we left join with the user table. Similarly, we want to know whether a user wants the item and the total number of users who currently want this item. Hence, we left join the itemWanter table twice with different conditions on each left join. There are also a few other conditions that are stated in this query. Firstly, the item's expiry datetime should be larger than the current datetime since we only want non-expired items to be shown on the home page. Furthermore, the item should not have been given out already. Hence, the takerID should be null. Lastly, since we are fetching n number of queries in reverse chronological order, we are essentially querying from large itemIDs to small itemIDs. Hence, everytime we do a query, we record the smallest itemID seen so far and query only for itemIDs smaller than the smallest itemID seen so far. In this way, we work our way from the end of the item table to the start of the item table, never displaying an item twice on the home page.

SQL Query 3:

```
SELECT u.name, n.notificationID, n.notificationType, n.itemID,
n.userID, n.wantID, n.commentID, n.thankID, n.timeCreated,
iw.timeWanted, n.active, rn.readnotificationID, i.giverID,
i.takerID, i.title, iw.wanterID, t.receiverID, c.commenterID
FROM notification AS n
LEFT JOIN readnotification AS rn
ON n.notificationID = rn.notificationID AND rn.userID = <insert user id>
LEFT JOIN user AS u
ON u.userID = n.userID
LEFT JOIN item AS i
ON n.itemID = i.itemID
LEFT JOIN itemWanter AS iw
ON iw.itemID = n.itemID AND iw.wanterID = <insert user id> AND n.timeCreated >
iw.timeWanted
LEFT JOIN thank AS t
ON n.thankID = t.thankID AND t.receiverID = <insert user id> AND t.thankerID !=
<insert user id>
LEFT JOIN comment AS c
ON n.commentID = c.commentID AND c.commenterID != <insert user id>
WHERE n.active = 1
AND rn.readnotificationID IS NULL
AND n.timeCreated <= NOW()
AND (
```

```
(n.notificationType in (1,2) AND i.giverID = <insert user id> AND i.takerID IS NULL) OR
(n.notificationType = 3 AND i.giverID = <insert user id> AND c.commenterID IS NOT
NULL) OR
(n.notificationType = 3 AND iw.wanterID IS NOT NULL AND i.takerID IS NULL AND
c.commenterID IS NOT NULL) OR
(n.notificationType = 4 AND iw.wanterID IS NOT NULL) OR
(n.notificationType = 5 AND t.receiverID IS NOT NULL)
)
ORDER BY n.timeCreated DESC
LIMIT <insert number of notifications to retrieve>
```

The above query is executed in order to retrieve all relevant notifications for a certain user. There are 5 types of notifications:

```
// NotificationType 1: Someone wants an item that I am currently giving but haven't
given out
// NotificationType 2: My item has just expired and I haven't given out the item
// NotificationType 3A: Someone commented on an item I am giving out OR have
given out
// NotificationType 3B: Someone commented on an item that I am currently wanting
AND hasn't been given out
// NotificationType 4: The item that I am currently snagging has been given out
// NotificationType 5: I receive a thanks on my wall
```

From the above, we see that there are 5 main types of actions that can occur on our platform - wanting, expiring, commenting, giving and thanking. Hence, whenever one of these actions is performed by any user, a row is generated on the notification table. All users will then query this notification table to find the notifications that they will potentially be interested in. When they read the notification, another query is done to update a readnotification table to indicate that they have read the corresponding notificationID and are no longer interested in this notification.

Milestone 7

Show us some of your most interesting Facebook Graph queries. Explain what they are used for. (2-3 examples)

/me/friends:

Find out all your friends who are using this app. This graph query is very useful because your friends are the ones you'd likely give free items to and receive free items from. Hence, being able to explore your friends on the application is very helpful. This query can also help push users to invite their friends to use our application in order for the application to become more useful for them.

/me/objects/fbgiveforfree:freeitem & /me/fbgiveforfree:give:

The first query creates a custom object (a free item) on our platform. The second query creates a custom action (give) with its corresponding object (a free item). These two queries create the story that can be shared on Facebook to allow friends of users to find out about the stuff they are giving away.

/me:

The most basic query to get detail of the user's name which we store on our application.

Milestone 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!

There are two main types of feeds, both implemented through sharing stories. As mentioned below, the two stories are 'giving a free item' and 'receiving a free item'. These are the two main activities that occur on our platform (beyond wanting/unwanting an item).

It is very helpful for a user to share the items he/she is giving out on Facebook since this is most likely the platform on which his/her friends first see the opportunity to get the free item. This would in turn direct a lot of traffic to our site and increase the chances of the preloved good finding a second home.

Receiving a good is the ultimate goal of our platform (merely uploading a free item to be given away would not help). Hence, we believe that it is important to celebrate this story. Moreover, sharing this story on Facebook provides users with a platform to show their appreciation to givers.

Milestone 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.

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.

Upon consideration, we will be placing the like button above-the-fold for 2 reasons: (1) People focus their attention towards content that is found above-the-fold. A similar eye-tracking study performed by Nielsen Norman Group found that respondents spent 80.3% of their viewing time above-the-fold (see Figure 1). In this way, the Like button will not be overlooked. (2) Furthermore, the number of likes displayed beside the Like button, if significant, provides social proof and will encourage other users to explore the page.

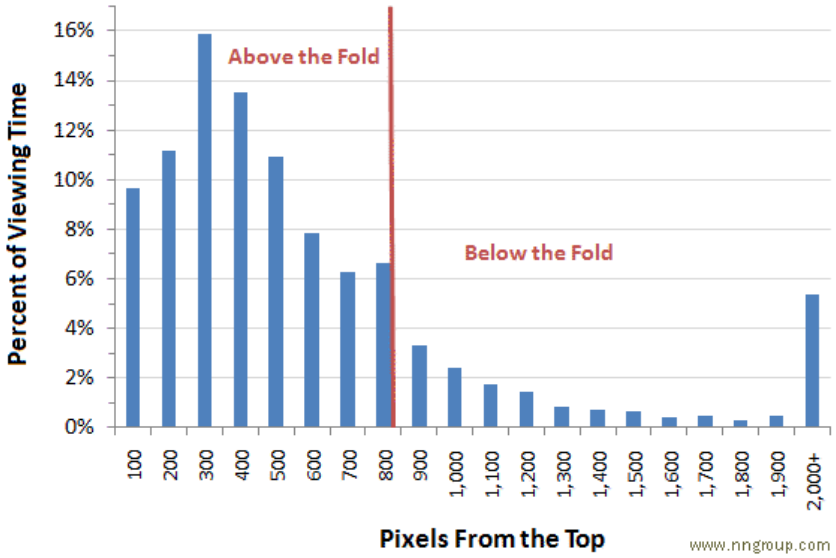


Figure 1: Percentage of viewing time from top to down

As can be seen from Figure 2 below, the like button is placed above the fold and right below our call to action in the Feeds page. With the button placement in the Feeds page, we believe that the free items posted in the Feeds page as well as the positive call-to-action will provide double reinforcement for the user to like the page.

Figure 2: Like button placement

Milestone 10

Explain how you handle a user's data when he 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 conditions.

Facebook's platform policy states that:

> "Delete all of a person's data you have received from us (including friend data) if that person asks you to, unless you are required to keep it by law, regulation, or separate agreement with us. You may keep aggregated data only if no information identifying a specific person could be inferred or created from it."

We receive a user's name, profile and friends list from Facebook. However, we only store their names in our database. Their friends list is never stored, and is obtained from Facebook via an API call when necessary.

Since we only store the name, we only have to remove that information upon account deletion to be fully compliant with Facebook's terms and conditions.

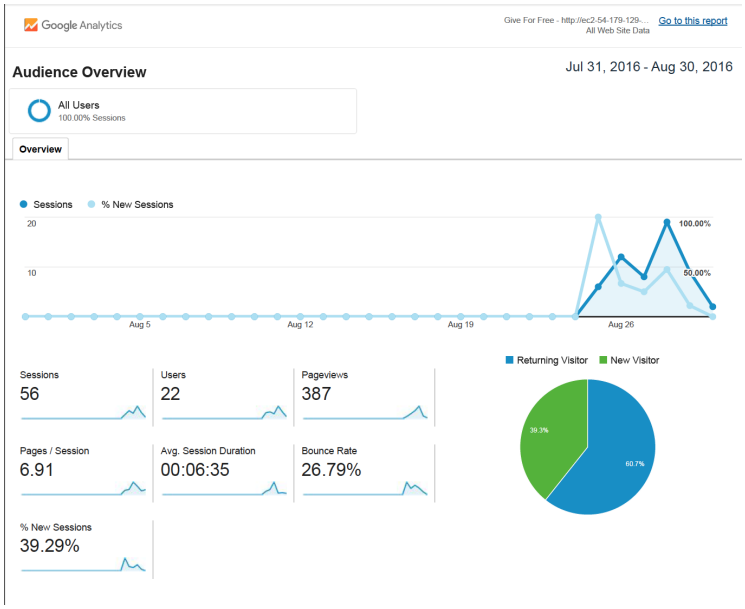
This is what happens when a user deletes his account:

1. The user's name is changed to "Deleted User" in the database.
2. We activate a delete flag for the user.
3. We delete all the user's ungiven items (since he probably won't be giving them out anymore).

All other information that the user accumulates from his interactions with our web application will not be deleted. This allows the user to have a seamless experience if he decides to start using our app again. It'll also allow others to see his past comments, previously given items, and other traces of data he left here and there.

Milestone 11

Embed Google Analytics on all your pages and give us a screenshot of the report. Make sure the different page views are being tracked!



Milestone 12

Describe 3 user interactions in your application and show us that you have thought through those interactions. You can even record gifs to demonstrate that interaction! It would be great if you could also describe other alternatives that you decided to discard, if any.

User interaction 1: The gift button is placed in the navigation bar such that users can see it and use it.

Initially, the gift button was found in a floating action button at the bottom right of page (see Figure 3). However, during our usability testings, we discovered that users could not find the gift button easily. This signified that one of most important intended user action - to gift - could not be completed efficiently.

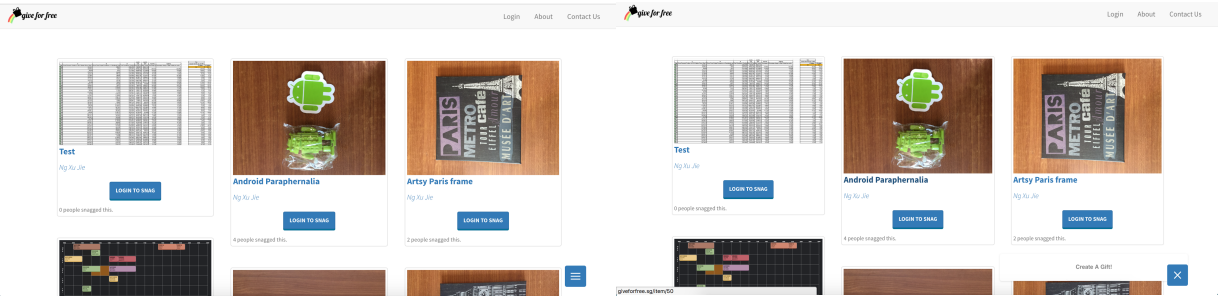


Figure 3: Old gift button placement

Therefore, in the current version, you will see the “Gift” button right up on the navigation panel. This will allow users to effectively complete the user action of gifting.

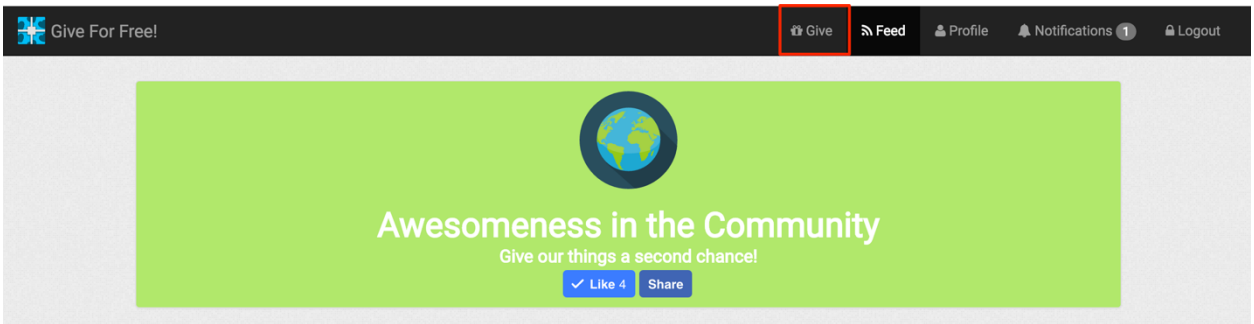


Figure 4: New Gift button placement

User Interaction 2: On the Feeds page, users can view the number of likes per item. This information will help them to decide if they want the item or not. It will also encourage them to click into the item to view further details.

Initially, the “snag” button - an older version of the “want” button was placed in the Feeds page over each item (see Figure 5 below). After some observation however, we realized that users might just click on the snag button without actually finding out more about the item description.

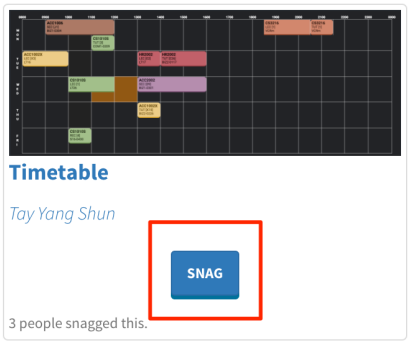


Figure 5: Snag button placement

This was undesirable because there might be users who actually do not want or need the item but yet click on the “snag” button because it is simply too easy to do so. Hence, other users who genuinely need or want the item would stand a lower chance of getting it (if item is allocated by random draw) and be discouraged from using the platform in the long run.

Therefore, in the current version that you see, users can only indicate that they want an item after clicking into the item page, which generates enough friction to prevent unnecessary snagging and yet not deter from genuinely interested parties from indicating their interest.

User Interaction 3: In our application, we built in features that helped item givers give away stuff easily. These features included (1) a comments section for item wanters to ask questions or post their reasons for wanting an item, (2) the option for the item giver to specify the mode of collection as well as (3) the ability to set an expiry date for the item.

We included (1) because we wanted the platform to truly connect old items to people who needed them the most. Only in this way will greatest value be created. Therefore, the comments section would enable the giver to select the recipient based on what each potential recipient writes in the comments section. This idea was actually borrowed from the Neighbourhood Freecycle 2 Facebook group, whereby users wrote in the Facebook comments section why they should deserve the item in question.

(2) and (3) were added based on user input that we have gathered from the Singapore Really Really Free Market (SRRFM). (2) was added because we wanted to align the different expectations that givers and want-ers might have about the method of delivery. For instance, we learnt that some givers were open to providing their house

address. However, some were unwilling to divulge such information and would instead prefer mailing the item to the want-er. In the latter case, the want-er is responsible for paying the postal fees. Therefore, the specific mode of collection should be made known by the giver such that any user who wants the item but is not agreeable to the method of collection is automatically sieved out.

Last but not least, the implementation of the expiry date for an item was predicated on an understanding of user psychology. While speaking to item-givers at the SRRFM, we learnt that one of the reasons they would bring all of their items to a physical location instead of uploading it online on a freecycling platform was due to the uncertainty of getting rid of them. In other words, when a user uploads an item for donation on an online platform (e.g. the Freecycle 2 Facebook group), there is no chance of knowing when will somebody come along and want that item. This creates a problem for the item-giver because he or she would have to hold on to that item for an extended period of time. Furthermore, we learnt for most item-givers, the alternative course of action would have been to simply throw those items away. The uncertainty of item disposal is therefore, something that users have to wrestle with. In view of that, allowing our users the opportunity to specify the expiry date helps to ease the uncertainty as well as provide a strong and urgent call-to-action for the item-wanters to indicate their interest for the item.

Milestone 13

Implement at least one Story involving an Action and Object in your application. Describe why you think it will create an engaging experience for the users of your application.

There are 2 main stories that are implemented on our platform - giving a free item and receiving a free item. In the prior story, ‘give’ is the custom action and ‘free item’ is the custom object. In the latter story, ‘receive’ is the custom action and ‘free item’ is the custom object.

When uploading a free item, a user is given the option of sharing this on Facebook. If this option is selected, a story is automatically created and posted on the user’s Facebook Feed. By specifying the ‘end_time’ on the creation of this story as the expiry date of this free item, we ensure that the correct tense is used on this Facebook story - ‘[User] **is giving** a free item’. This sharing functionality is critical for our platform as it helps users publicize their free item to their friends. Friends of the giver can then see the FREE item while scrolling through Facebook and immediately find the item by

clicking on the link associated with this story. By sharing the story on Facebook, users dramatically increase the chances of their preloved goods finding a new home.

After an item is given by user A to user B, a button is offered to user B on the corresponding item page to share the story of him having '**received** a free item'. This button gives an avenue for user B to thank user A by sharing this story on Facebook. In this case, instead of explicitly sharing the story, user B is invited to type a thank you message and the opportunity to tag the giver. This story is extremely important in emphasizing the social incentive of giving. User A would be able to see that his/her preloved good is in good hands and that he/she has done a good deed. Emphasizing on such social incentive is important in driving users back to our platform as people look to do more good given their positive experience in their previous giveaways.

Milestone 14

What is the best technique to stop CSRF, and why? What is the set of special characters that needs to be escaped in order to prevent XSS? For each of the above vulnerabilties (SQLi, XSS, CSRF), explain what preventive measures you have taken in your application to tackle these issues.

What is CSRF?

Cross-Site Request Forgery (CSRF) occurs when a user has been authenticated, and a malicious third-party takes advantage of the user's logged in status to perform some unwanted action on the site. If the site is not properly secured, the hacker can do anything - from changing the user's password to transferring funds out of the user's account.

To prevent a CSRF attack, the site needs to validate post requests made to the site. This can be done either programmatically, or with the help of the user.

User Re-Authentication

For important transactions such as money transfers, it would be prudent to get the users to re-authenticate by entering their password again, or filling in a CAPTCHA form. This would make it prohibitively difficult to remotely trigger transactions using CSRF.

CSRF Token Validation

However, for things that are less critical, such as posting a new blog post, it would not make sense for users to re-authenticate. In this situation, it would be better to use a

CSRF token for authentication. For each request where the user makes changes to the database, the site generates a new CSRF token and sends it to the client, and this token is inserted as a hidden form input. When the form is submitted, the server will validate the token before performing any operations. If the token is not sent with the form, it raises alarm bells and the site will not process the transaction.

Preventing XSS

You could go on to check the Origin and Referer headers, since it is near impossible to spoof headers in a CSRF attack. That said, the above should be sufficient to secure your site against CSRF attacks, as long as the site has also prepared itself against XSS attacks. Otherwise, XSS can be used to read the page, obtain the CSRF token, and use that token to pass itself off as a real user.

XSS Special Characters: &, <, >, ", ', `, , !, @, $, %, (,), =, +, {, }, [,]
(From http://wonko.com/post/html-escaping)

Preventative Measures

To prevent XSS attacks, we sanitized all our form inputs using *express-validator* and *xss*.

To prevent CSRF attacks, we used *csurf* to generate CSRF tokens and perform validation of the tokens for all pages that required form inputs. We ensured that we did not use GET requests when we needed to change the database, and our API only accepted JSON via *bodyParser.json*.

Milestone 15

Describe 2 or 3 animations used in your application and explain how they add value to the context in which they are applied. (Optional)

1. **Preloader**

We managed to implement a preloader we've found online. The preloader fires up on the start of the page, and disappears when the HTML document is ready. This allows us to engage the user while our pages load, and at the same time shows more transparency with regards to conveying the status of the site, helping to reduce waiting anxiety.

2. **Hover Highlight**

We want users to understand that the thumbnails are clickable. Thus we decided to animate our thumbnails to fade color on mouseover.

Milestone 16

Describe how AJAX is utilised in your application, and how it improves the user experience.

Ajax, which stands for Asynchronous JavaScript and XML, is a method of building interactive applications for the Web that process user requests immediately.

AJAX is implemented in the infinite scrolling on our home page as well as profile page (for gifts and wants tab). Since there are likely to be many items being given out for free on our platform, it would take very long to query for all items currently available. Doing such a query would cause pages to load too slowly. Instead, we implemented infinite scrolling. A few items would be loaded initially (since query has a limited number of items returned, the query is much faster). When the scroll gets to the bottom of the page, we use AJAX to get a few more items from the database. In this manner, we spread out the query time and make a much better user experience in browsing products. Furthermore, users are unlikely to actually view all the items available on our platform. Using AJAX to query our database would ensure that we only return the necessary items to the user.

AJAX is also implemented in the ‘wanting’ and ‘unwanting’ process. The ‘want’ and ‘unwant’ process on our platform is very similar to the ‘like’ button on Facebook. When we ‘like’ something on Facebook, this POST request does not need to interrupt our user experience. For instance, it would be very irritating if this POST request caused a page reload everytime a user ‘wants’ or ‘unwants’ an item.

Bonus Question (a):
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 server-side method?

Reasons	JavaScript Performance	Server Performance	Conclusion
Interruption of User Interaction	A dynamic page is served, on which the JavaScript will determine which layout to render based on the user state. No page reload is typically required.	A static page is sent after concluding the state of the user. A page reload is usually required to swap out the DOM elements.	JavaScript reduces interruption to the user interaction by dynamically changing the page without reloading or fetching a new HTML file.
Latency of dynamic pages	If there are a lot of elements to hide / show, the processing time on the client-side would increase.	Using the server-side approach, the elements and layouts are pre-defined and processed. The load time depends on the network performance alone.	Server-side implementations reduces the processing on the client-side.
Information Hiding	To show or hide different elements using JavaScript, these elements will be included in the DOM even prior to login or authentication.	Loading from the server-side, we pick whichever page that the client is authorised to view and serve.	Server-side implementations allow us to hide the implementations from being revealed to the client. For instance, the client would not be able to derive the administrator's dashboard from the DOM since the elements are on different pages and controlled strictly by the server business logic and the user session login.
Programmer	Elements require	Static pages free	If the number of elements to

Time	programming effort in controlling their differences in layout and appearance based on user sessions.	programmers of the responsibility to rearrange elements on-the-fly.	hide/show is minimal, JavaScript will be effective. Otherwise, the JavaScript approach would take too much programmer effort.
Server Traffic	A JavaScript need not require different HTML documents.	Server-side loading will require multiple fetches of DOM documents.	If server access frequency is expensive, JavaScript is a better choice.
Security	Links, assets or data could be found in the hidden elements.	Data is protected and concealed from the unauthorised user.	Server-controlled loading will reduce the incidence and exposure to hacking or information gathering.

Bonus Question (b):
What is the primary key of the home_faculties table?

Ans: matric_no

END