CS3216 Assignment 1 - ExchangeBuddy

Group 5
Irvin Lim Wei Quan
Leon Mak An Sheng
Lam Chi Thanh
Ng Chuen eng Eugene

Application URL: https://app.exchangebuddy.com/

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 chose to do a standalone application for the following reasons:

1. Our website only makes use of basic Facebook Graph features such as login, like and share. These features works perfectly well and can be implemented easily in a standalone web application.
2. A standalone app offer more freedom in UI. While a Facebook canvas app may need to adopt certain UI features from Facebook, our standalone app can be designed more freely.
3. A standalone app can scale better in the future. As our application grows, we may want to be independent of Facebook (users can sign up independently or log in with Facebook, or even other OAuth services).

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.

We used Meteor as our framework, with React as our rendering engine as well as Material-UI for the the design templates. We also used MySQL for most of the database data, along with MongoDB solely to host the chat messages.

The main reason is because two of our members, Irvin and Leon, have had experience working with Meteor and React, and it would help to speed up development process by having at least some people in the team who are familiar with the toolset. We also considered using Express + Socket.io as our backend framework, but we had encountered too much difficulties, and decided to stick with something more familiar.

We also used a relational database for all information (with the exception of chat messages), using MySQL with Sequelize as our ORM. Since Meteor originally comes packaged with MongoDB, for the purposes of this assignment we had to use MySQL which meant that we could not take advantage of Meteor's built-in Optimistic UI and Pubs/Subs features. However, it was also advantageous to use a relational database considering that many tables are so interlinked.

We had to use MongoDB for chat messages simply because there was no easy solution to poll a MySQL database in real-time efficiently for changes, and since chat needs to be real-time, we decided to use MongoDB to fully make use of Meteor's built-in functionality for such a use case. Furthermore, since we are making use of a NoSQL database, we are able to store different types of chat messages with totally different structure in the same database. For example, a chat message can either be plaintext, or an image, or in our case, we allow users to post a Facebook event onto the chat, and hence we would require the chat message database row to store additional data easily.

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!

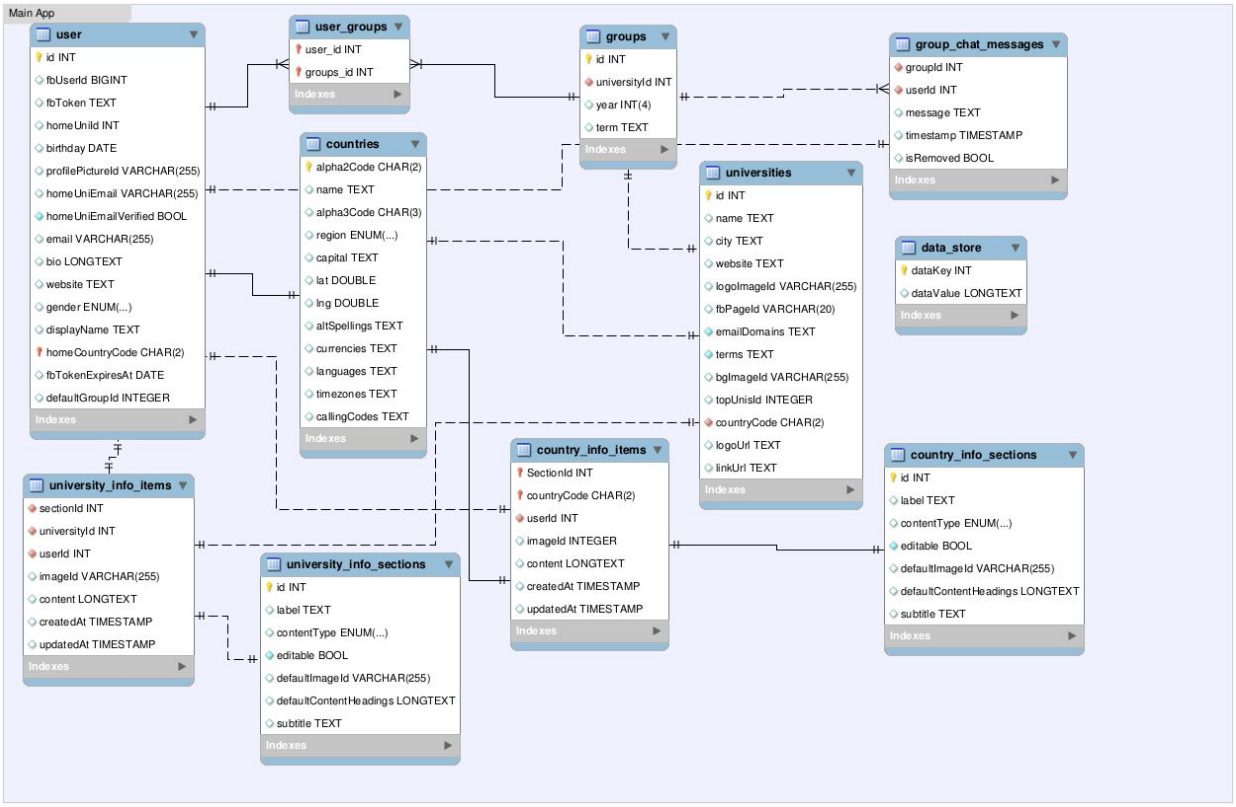
The app is available at https://developers.facebook.com/apps/580995375434079/dashboard/.

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.

Implemented. Upon logging in with Facebook on the landing page, users will be redirected to a page to complete their profile, indicate their exchange university, as well as to send out a verification email.

User information is also visible at the top right of a Group page, when the popover menu is activated.

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 (if you are using an ORM, find out the underlying query) and explain how it works.

<u>Querying for a specific user (by ID):</u>

Sequelize query:

```
User.findOne({
    where: { id: userId },
    include: [
        { model: University, as: 'homeUniversity' },
        { model: Country, as: 'homeCountry' }
    ]});
```

Raw query:

```
SELECT `users`.*, `homeUniversity`.*, `homeCountry`.* FROM `users` AS `users`
LEFT OUTER JOIN `universities` AS `homeUniversity` ON `users`.`homeUniId` =
`homeUniversity`.`id` LEFT OUTER JOIN `countries` AS `homeCountry` ON
`users`.`homeCountryCode` = `homeCountry`.`alpha2Code` WHERE `users`.`id` = 1;
```

This query simply gets all information about a specific user (given a user ID), and performs a left outer join on the universities and countries table, and passes the user object to components which can consume and display certain information about a user and his university/country.

Querying for the latest edit on a wiki section:

Sequelize query:

```
CountryInfoItem.findOne({
    where: { countryCode, sectionId },
    order: [[ 'createdAt', 'DESC' ]],
    include: [
        { model: CountryInfoSection, as: 'section' },
        { model: Country, as: 'country' }
    ]});
```

Raw query:

```
SELECT `country_info_items`.*, `section`.*, `country`.* LEFT OUTER JOIN
`country_info_sections` AS `section` ON `country_info_items`.`sectionId` =
`section`.`id` LEFT OUTER JOIN `countries` AS `country` ON
`country_info_items`.`countryCode` = `country`.`alpha2Code` WHERE
`country_info_items`.`countryCode` = 'SG' AND `country_info_items`.`sectionId`
= 1 ORDER BY `country_info_items`.`createdAt` DESC LIMIT 1;
```

This query gets the latest edit for an article on the information wiki for a particular section (given a section ID) for a given country (given the country code). This query is to display the current content for an article, and hence we would only be interested in latest data, and one row only, which is done by `ORDER BY `country_info_items`.`createdAt` DESC LIMIT 1`.

Querying for a list of users by user IDs:

Sequelize query:

```
User.findAll({ where: { id: { in: ids } } });
```

Raw query:

```
SELECT * FROM `users` AS `users` WHERE `users`.`id` IN (1,2,3);
```

This query gets an array of users by user IDs. The purpose of doing so is because we cannot perform a JOIN across a MySQL table and a MongoDB collection, and so we have to join the users to the corresponding message objects manually in the server-side code. Though performing two queries on two different databases sounds complicated, it is necessary to implement real-time chat message updates on the client (using Meteor's Pubs/Subs and Minimongo).

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

We used graph queries to get a list of facebook events from the lat long data. 2 graph queries were needed since FQL was removed. The npm package 'facebook-events-by-location-core' was used for this.

- 1st to search for places

```
"https://graph.facebook.com/" + self.version + "/search" + "?type=place" + "&q=" + self.query
+ "&center=" + self.latitude + "," + self.longitude + "&distance=" + self.distance +
"&limit=1000" + "&fields=id" + "&access_token=" + self.accessToken;
```

Eg.: `https://graph.facebook.com/2.7/search?type=place&q=Singapore&center=1.3521,103.8198&distance=1000&limit=1000&fields=id`

- 2nd to search for events in the result of each of these places.

```
var eventsFields = [ "id", "type", "name", "cover.fields(id,source)", "picture.type(large)",
"description", "start_time", "end_time", "attending_count", "declined_count", "maybe_count",
"noreply_count" ];
var fields = [ "id", "name", "about", "emails", "cover.fields(id,source)",
"picture.type(large)", "location", "events.fields(" + eventsFields.join(",") + ")" ];
"https://graph.facebook.com/" + self.version + "/" + "?ids=" + idArray.join(",") +
"&access_token=" + self.accessToken + "&fields=" + fields.join(",") + ".since(" + self.since +
")";
```

Eg:`https://graph.facebook.com/2.7/?ids=47372682766,135972916504730,215407548566363,111017992241640&fields=id,name,about,emails,cover.fields(id,source),picture.type(large),location,events.fields(id,type,name,cover.fields(id,source),picture.type(large),description,start_time,end_time,attending_count,declined_count,maybe_count,noreply_count)`

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!

We have allowed users to share articles from the wiki using a Share button. We make use of the Feed dialog as it allows us to define custom title, description and image for the Facebook post that appears on a user's timeline, as opposed to the Share dialog, whereby although it does not require any specific permissions or app ID, it gets the data automatically by crawling the web page at the URL. Hence, by making use of the Feed dialog, we can optimise a post for the Facebook timeline.

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.

We have added the Like button on the wiki article page as well. Nowadays, likes are an important metric which many people use to determine the quality of content, and we feel that this may be a powerful tool to help people to validate and filter good content from noise. Users can see how many people have liked the article in total, as well as which of their Facebook friends have also liked it.

One thing to consider was the fact that the wiki articles are user-contributed. So by looking at the number of likes on a particular articles give others users and us as moderators have a glimpse about the quality of that article.

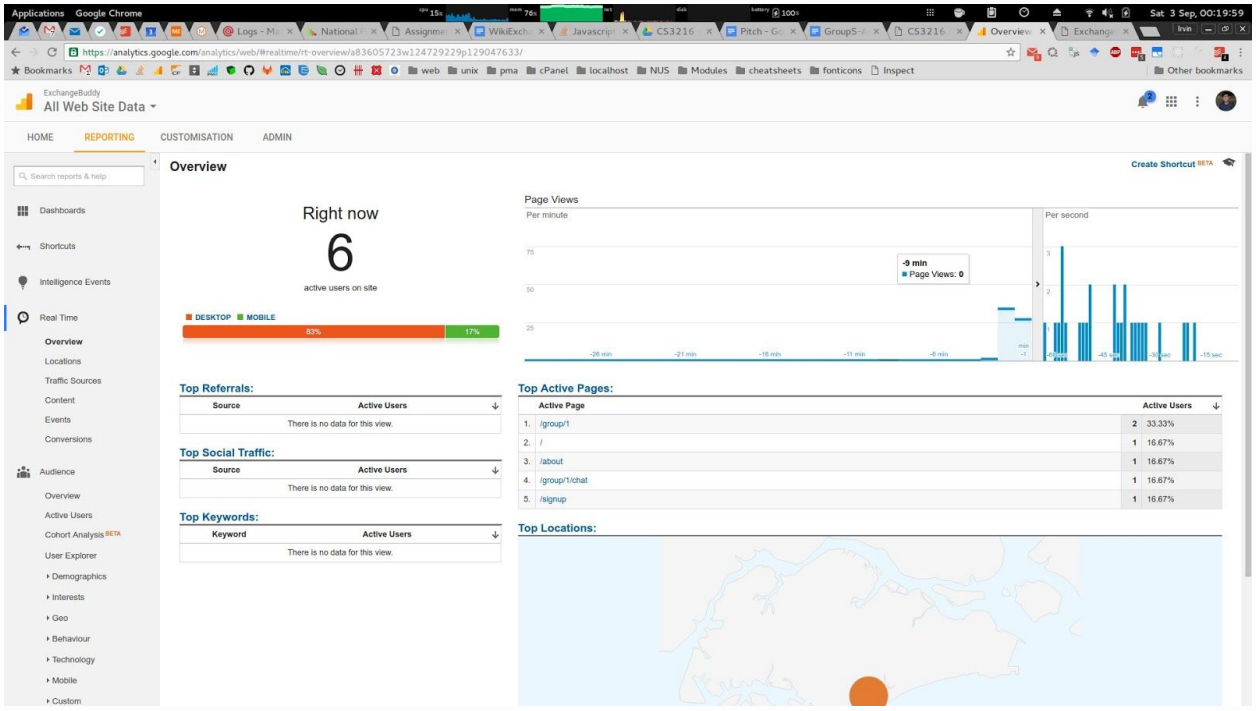
Milestone 10: Explain how you handle a user's data when he removes your application. Convince us that your approach is necessary and that it is the most logical. Do also include an explanation on whether you violate Facebook's terms and conditions.

We make use of Facebook's deauthorize callback URL to receive a notification event when a user removes the application through Facebook's App Settings. Using a server-side route, we intercept the POST request, where Facebook sends a signed request, and when properly decoded, prompts the app to remove the user from the database.

Hence, the deauthorization will cause the user's personal data to be removed from our database, and even though any user-generated content still may remain, no data should be personally identifiable and traceable back to the user. We believe this should be sufficient to fulfil Facebook's TOCs.

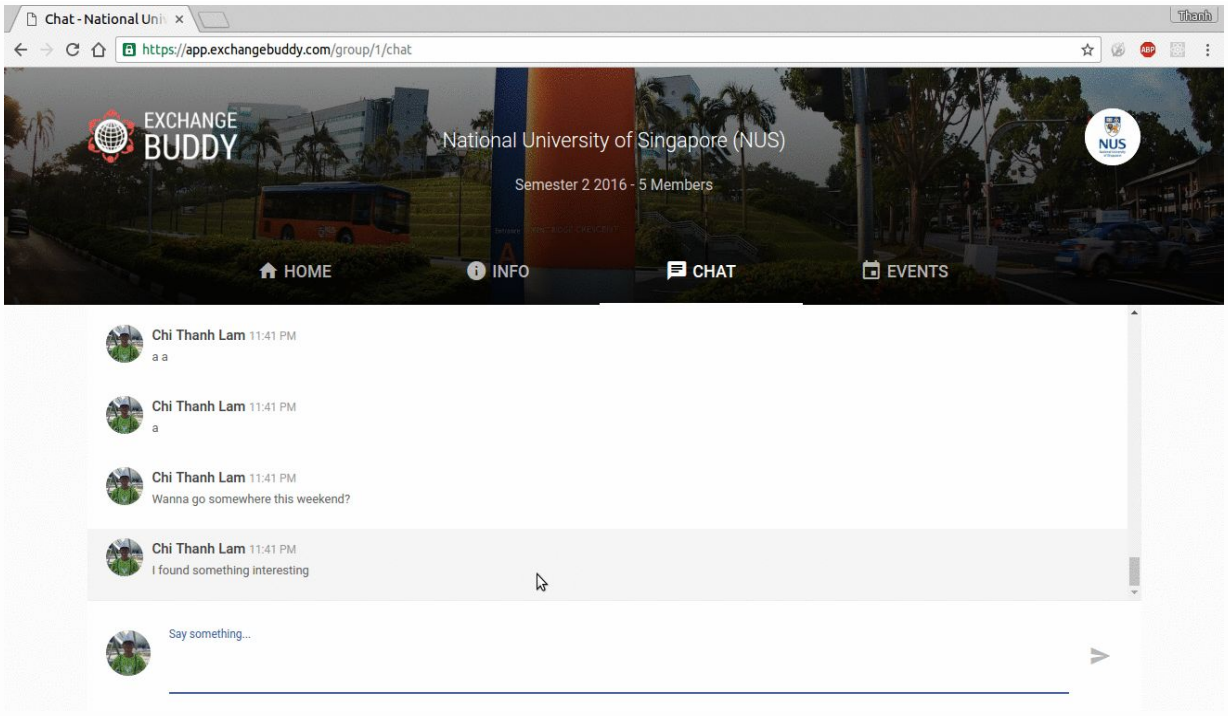
Milestone 11: Embed Google Analytics on all your pages and give us a screenshot of the report. Make sure the different page views are being tracked!

We make use the react-ga npm package to track page views from react-router. Hence, we are able to distinguish page views by URL.

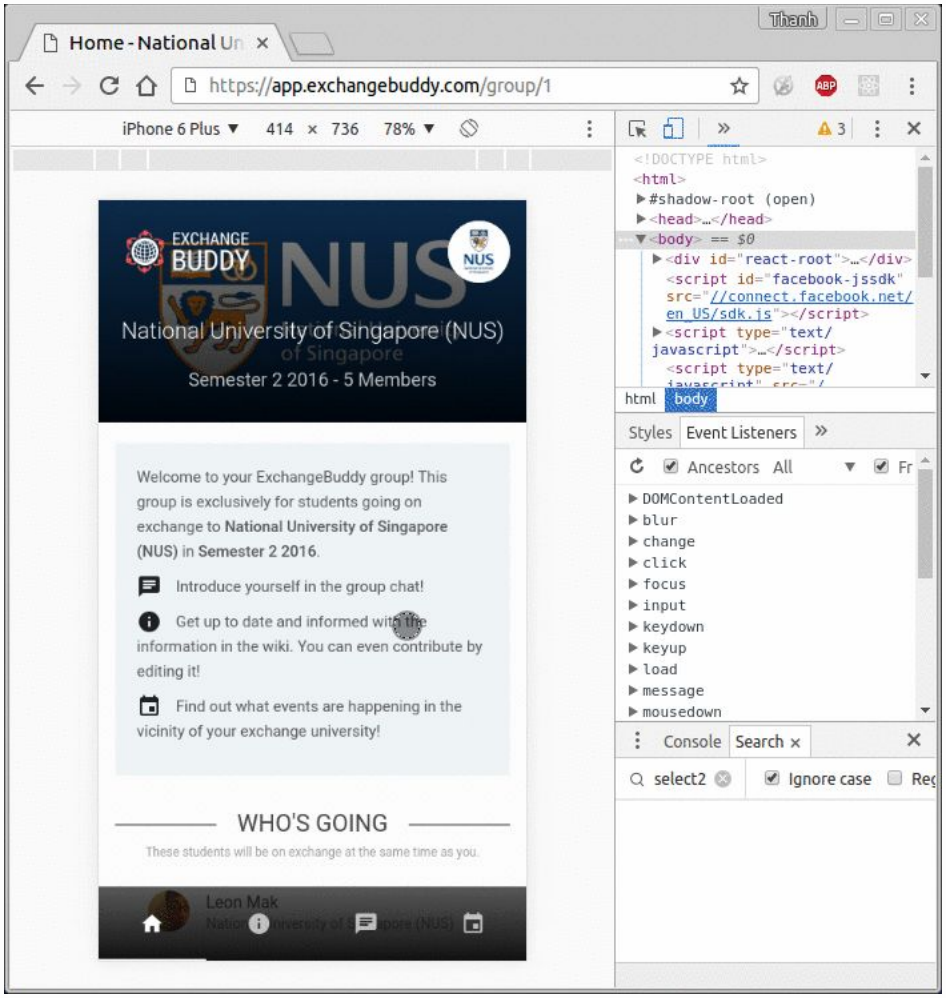


Milestone 12: Describe 3 user interactions in your application and show us that you have thought through those interactions. You can even record gifs to demonstrate that interaction! It would be great if you could also describe other alternatives that you decided to discard, if any.

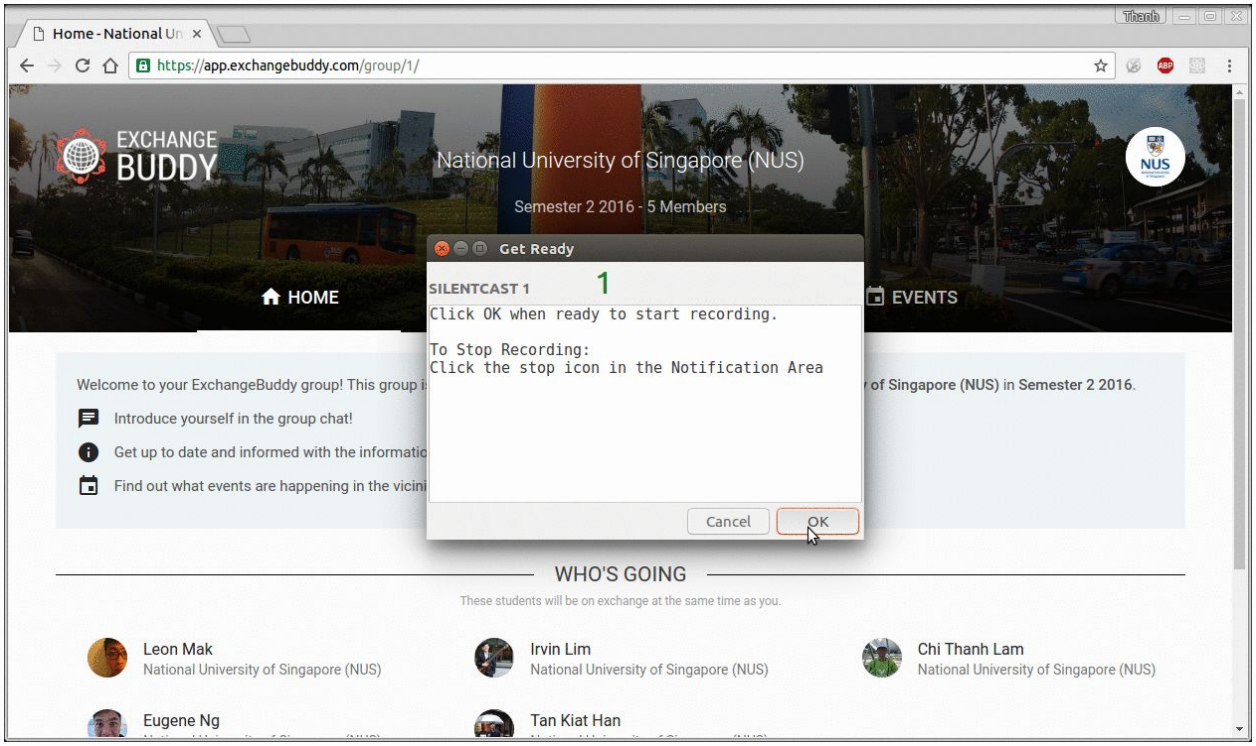
1. Posting MeetUp and Facebook events to the group chat. We have a group chat session for users in a group to communicate. We have an events route that pulls events from Facebook and MeetUp. So if users want to invite their friends to an event or let others know that he is interested in it, he can click a button to post it to the chat group for other users to see.



2. During the design process, we always take into consideration mobile users. Hence we try to make our app as mobile-friendly as possible, all the features our app offers works seamlessly on both desktop and mobile.



3. We encourage user collaboration for example in the information sections and also let the users customize their groups as much as possible. For example, we allow users to update the background header of the group, all group members can see the changes.



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.

We have created a Story where the user ‘Joins’(Action) an ‘Exchange Group’(Object). We feel that when users join the group on Exchange Buddy they will want to let others in their social network, especially other students going on exchange, know that they can link up and chat on on our platform. Also it will be compelling for other exchange users to see what exchange buddy can help them with.

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 vulnerabilities (SQLi, XSS, CSRF), explain what preventive measures you have taken in your application to tackle these issues.

We do not use cookies, so CSRF attacks are not possible. We make use of Meteor' Session variables, which are stored in localStorage, and thus other domains cannot read or intercept other domain's session variables, as opposed to cookies. We make use of JSONWebTokens (JWTs), encrypted with a secret key, in order to create a short token which could be easily transferred to and fro the client and server.

Since we are using React, it normally already prevents XSS, since it does not allow any form of HTML tags to be rendered from strings, and instead all elements are React components/objects. Only through the use of methods such as React's dangerouslySetInnerHTML would then XSS be possible.

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)

Opacity and padding css transitions are used to fade in the 'add photo icon' to indicate to the user that pictures can be uploaded by clicking the area, for example in the group 'cover' photo or on the information edit page.

Besides making transitions in the web ap smoother views, used internally in react components, for example tabs for material ui, show a clear transition of application state. animations in the textfield component, prompting the user that text can be typed in the field. Animations in the sign up page help demonstrate of progress of signing up, making user easier to navigate and follow instruction.

Milestone 16: Describe how AJAX is utilised in your application, and how it improves the user experience. (Optional)

Meteor makes use of WebSockets through Distributed Data Protocol (DDP) and thus does not need any use of AJAX. The whole app is somewhat a Single-Page Application (SPA) since the browser does not refresh the page and instead gets new content through routers components, using technology such as Redux. React's props and state allows components to be updated reactively upon new data, and thus it would create a more seamless experience for the user.