A Social HungryGoWhere

https://www.nuseats.club

Done By:

Jovin Liew	A0154833A
Charlton Lim	A0135788M
Ho Yi Hang	A0144915A
Wang Riwu	A0135766W

Milestone 1:

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

About NUSEats

Finding friends to join you for a meal can be a difficult in NUS due to the differing schedules. NUSEats aims to solve this problem by allowing you to broadcast your intended eating location and time on your Facebook. Friends can then indicate their interest by joining the meeting.

NUSEats also helps you make more informed decisions by providing real-time information like the approximate walking time to every canteen. The app also supports user ratings and reviews so that you know which stalls are worth the money.

The stall reviews also provides an avenue to give (potentially anonymous) feedback to the stall owners who read the reviews, enabling them to further improve their services.

Milestone 2:

Explain your choice of libraries 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.

Client Libraries/Frameworks

The frontend is written using React and Redux (and various other component libraries for React).

We chose client-side rendering over the more traditional server-side rendering so that we can offer a more fluid user experience (e.g., there is no need to do full-page reload when navigating between links).

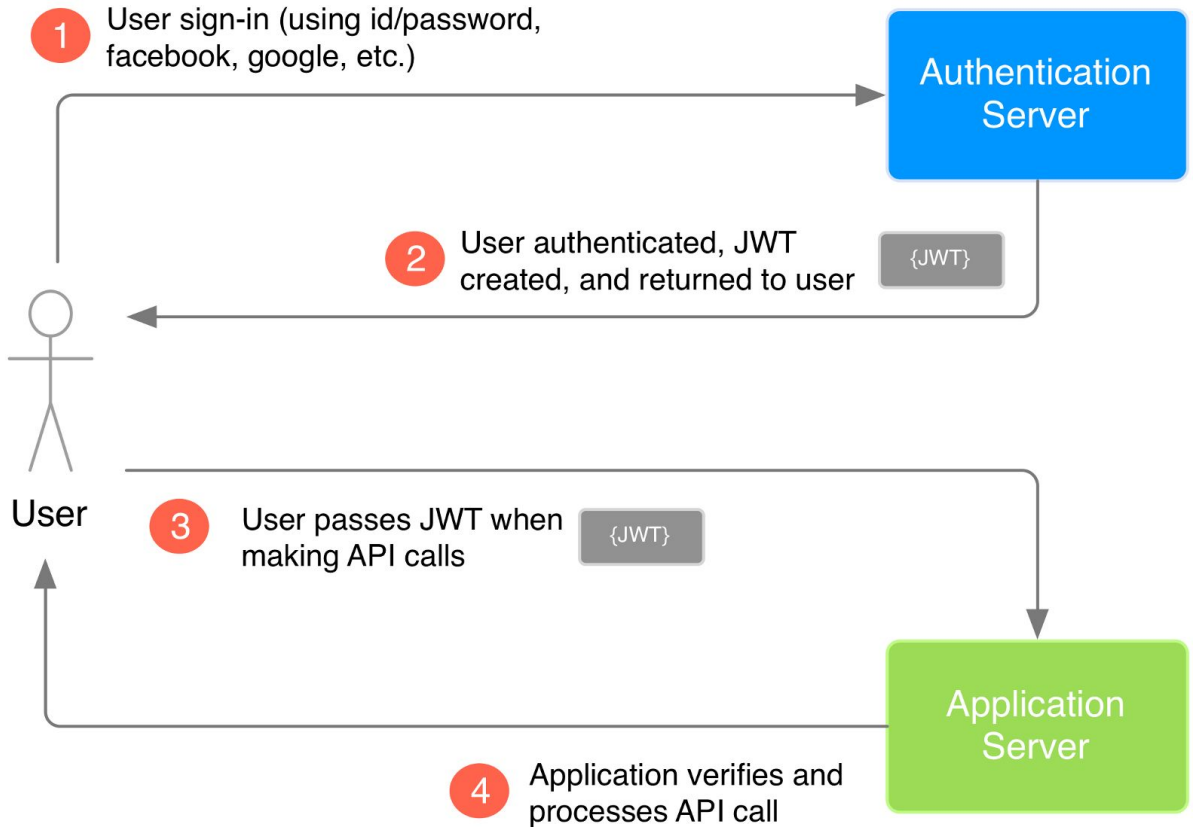
We chose React over other client-side libraries such as Vue and Ember as React has large community support and numerous up-to-date packages available. React's declarative style, one-way data flow and minimalistic API also makes it simpler to learn and develop in compare to Angular. Furthermore, we do not

find any significant advantage that any of these client-side library can offer over the others.

Redux is chosen for state management as our app deals frequently with state mutation and asynchronous actions (eg. fetch), hence the restrictions imposed by the Redux framework would make state mutations much more predictable.

Server Libraries/Frameworks

The backend server is written using the Express framework. We chose Express over other Node.js frameworks (like Hapi and Koa) because it is very mature and stable. It also has less boilerplate code than Hapi, which is desired in a simple application like this. However, Express lacks proper error handling mechanisms and thus we have to use an external library *Boom* to handle this.
As we want our backend to be stateless, we chose to use JSON Web Tokens (JWT) over sessions. This means that we do not require additional storage and makes the design scalable. Furthermore, sessions expires and needs to be garbage collected, while JWT carries expiry date along with the user data. The image below shows the how authentication is done using JWT. In our application, we use the *Passport* middleware to keep the code simple and unobtrusive.

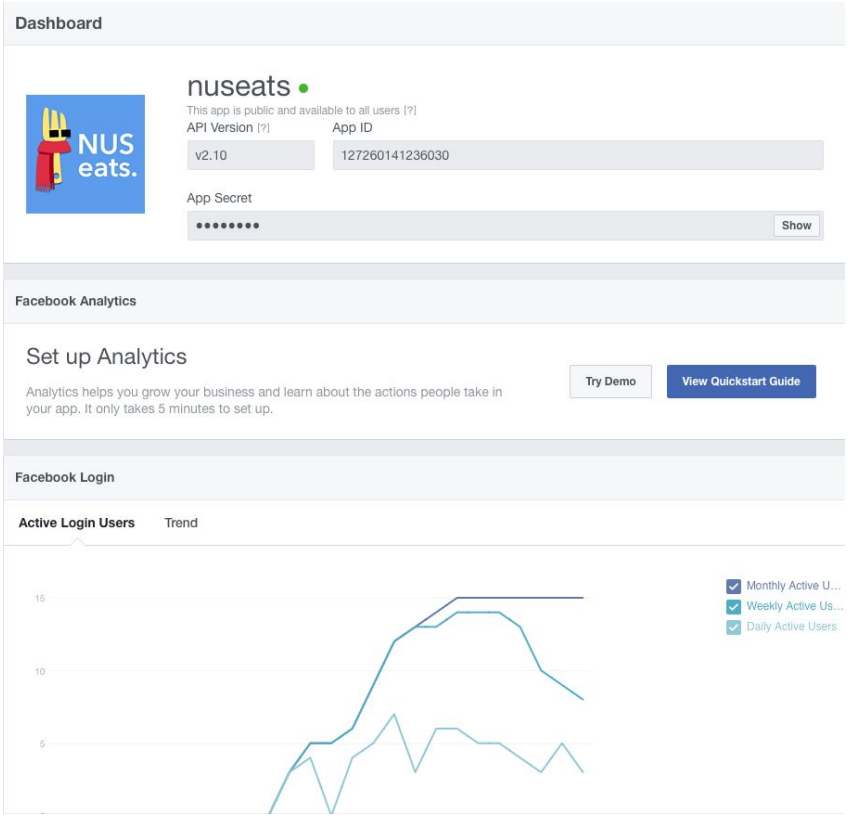


Instead of writing raw SQL queries, we use the *Sequelize* library, an Object-Relational Mapping (ORM) tool, to interact with the database. This gives us several advantages such as having the flexibility to switch between different Relational Database Management Systems (RDBMS). It is also more secure and robust from SQL injection attacks as there are preventive measures in place. Finally, it can save a lot of development time by simplifying the migration process.

For deployment, the Node server is running on AWS EC2. Behind the EC2 are a PostgreSQL database running on AWS RDS and AWS S3 to store photos.

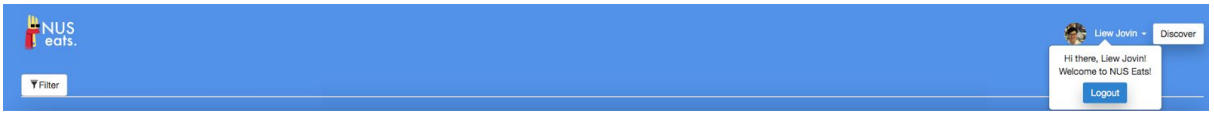
Milestone 3:

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! Screenshot the dashboard fields for your midterm submission.



Milestone 4:

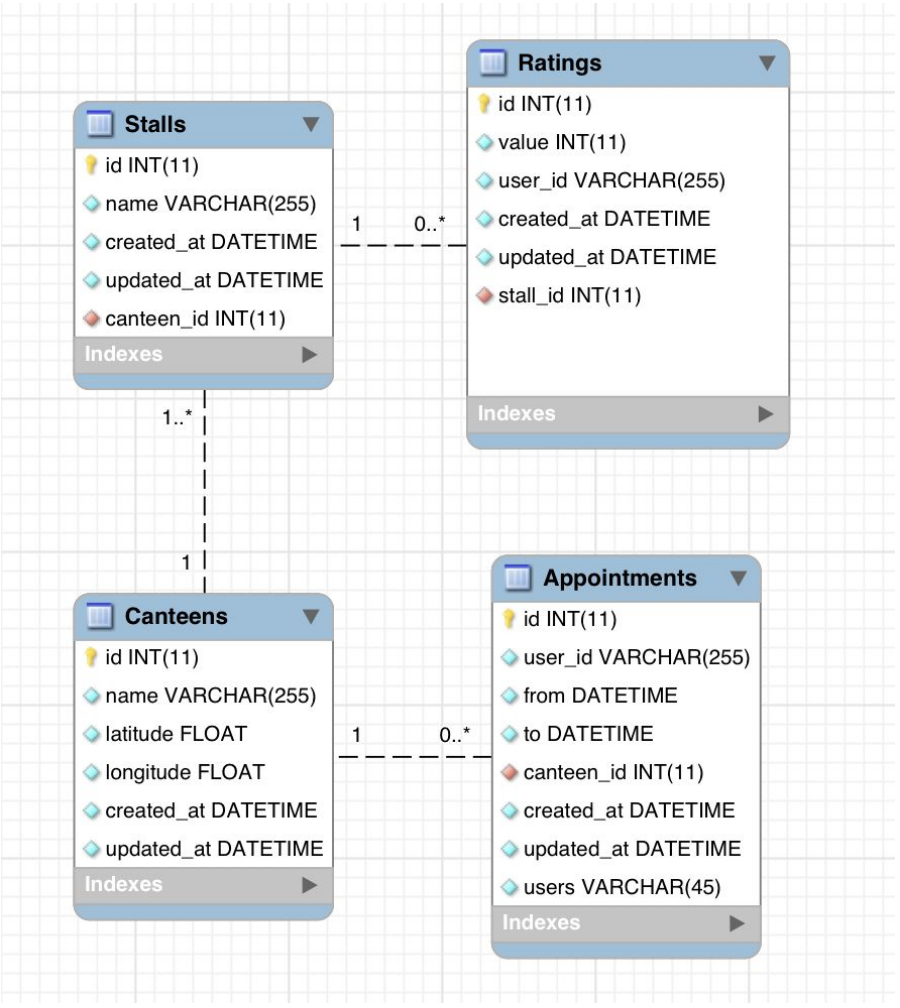
Integrate your application with Facebook. If you are developing a Facebook Facebook Web Games app, then users should be able to visit your app and at least see their name (retrieved using the API) on the page. Similarly, if you are developing a standalone app, users should be able to login to your app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



This can be seen in our navbar - users can login and logout from there. We also use the Facebook API to retrieve their friends so that the user can see meetings created by his friends. We also make use of Facebook comments plugin for stalls' reviews and like button for images.

Milestone 5:

Draw an Entity-Relationship diagram for your database schema.



As mentioned earlier, we chose PostgreSQL for our database.

1. Notice that we don't have a user table. This is because we do not need one. We are using JSON web token (JWT) as the access token to our API endpoints. A JWT is a cryptographically-signed JSON string. Typically, this JSON string describes the authenticated user. In our case, this JSON string contains the Facebook ID and access token of the authenticated user. Using these information, we can retrieve everything else that we need (name, friends etc) once the user logs in. We also plan to use Facebook comments for the stall reviews, hence eliminating the need for handling user comments as Facebook stores the comments on their own servers, which can be retrieved through the Facebook ID and access token.

2. In the appointment table, the users property is actually a JSON array (Postgres supports this), but is indicated as VARCHAR because we use MySQL workbench to generate the EER diagram, which doesn't support JSON. This is the array of user IDs that are joining the appointment.

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](https://www.wikiwand.com/en/Object-relational_mapping), find out the underlying query) and explain how it works.

Computing Average Rating

When we are querying for the list of stalls, we want to compute what is the average rating of each stall and return it as a field for each stall. Note that there is a one to many relationship between the Stalls and Ratings table and we need to do a join query on the Stalls table in order to achieve what we want. The image below shows the raw SQL query. We make use of the built in aggregate functions in PostgreSQL to compute the average rating. This query is a left outer join because we want all the properties in Stalls but only one in Ratings.

```
SELECT "stall"."id",
       "stall"."name",
       "stall"."uuid",
       "stall"."createdAt",
       "stall"."updatedAt",
       "stall"."canteenId",
       AVG("ratings"."value") AS "averageRating"
FROM "stalls" AS "stall"
LEFT OUTER JOIN "ratings" AS "ratings" ON "stall"."id" =
"ratings"."stallId"
GROUP BY "stall"."id"
ORDER BY "stall"."name" ASC;
```

Inserting Photos

To insert into the database, we use the INSERT INTO command and state the respective values according to the column sequence.

```
INSERT INTO "photos" ("uuid",
                      "userId",
                      "createdAt",
                      "updatedAt",
                      "stallId")
VALUES ('c4427bdd-5d71-44a0-a962-8773c7fa10cf',
        '1524106284336095',
        '2017-09-08 15:45:48.414 +00:00',
        '2017-09-08 15:45:48.414 +00:00',
        '6') RETURNING *;
```

Finding Appointments

To find all appointments created by the user, we specify it with the WHERE command.

```
SELECT "id",
       "userId",
       "startTime",
       "endTime",
       "attendees",
       "title",
       "description",
       "createdAt",
       "updatedAt",
       "deletedAt",
       "canteenId"
FROM "appointments" AS "appointment"
WHERE "appointment"."userId" IN ('111816802885088');
```

Milestone 7:

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

Me - Returns information about the current user. We take information such as profile picture and name of the users.

Me/friends - Returns friends of the current user.

/:user-id - This gives us the name of the arbitrary users, we use this in the feed feature so that users can discover different groups and meetings.

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! Explain what feeds you implemented and your thought process for your feeds.

We implement feed by showing a share dialog for the users to share a scheduled meeting. We use a share dialog over public_actions as users are typically more wary of applications that want to publish things automatically. Also, a share dialog allows the user add a caption to the meeting that the user is sharing.

We feel that the concept of jio-ing friends or meeting new people through our app is augmented by the sharing function. This also helps aid discovery of our web application by Facebook users that don't already know about NUSeats.

Milestone 9:

Your application should include a 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 feature whereby users can upload photos of food for each stall. We feel that a like button is appropriate here as it helps to incentivise users to upload content. At the same time, it helps users to discover which food is the most popular and probably the best tasting for each individual stall.

Milestone 10:

Explain how you handle a user's data when he removes your application and implement it. 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.

When the user de-authorises us on Facebook, we perform the following actions.

1) We delete the ratings that they have submitted for the canteens
2) Delete all photos that they have uploaded to our website
3) Mark all the meetings they created as cancelled
    a) Note that we do not delete the meeting as this might affect the other users - they may be confused from the sudden disappearance of the meeting

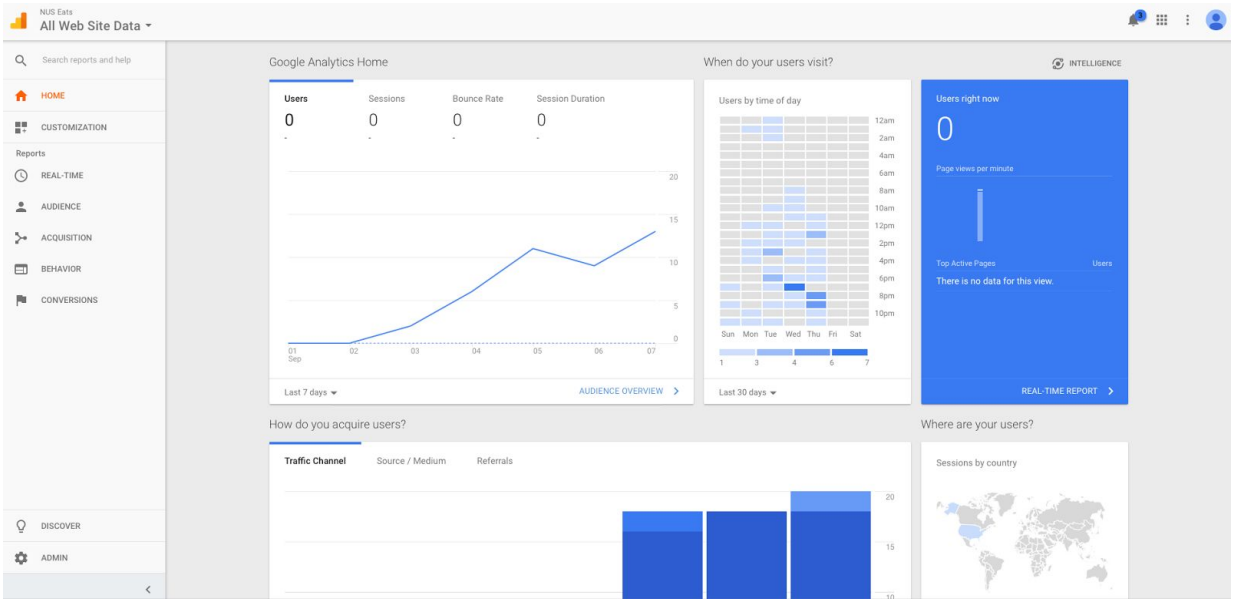
We made sure that we followed Facebook's terms and condition for developers.

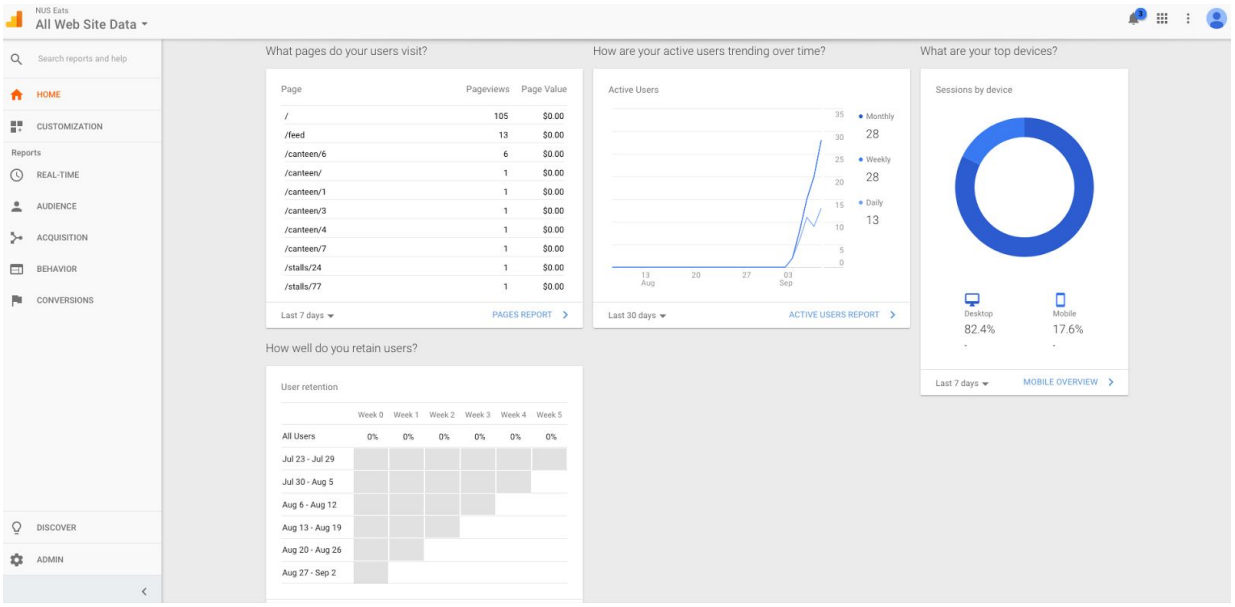
For sharing of meetings - we made sure to obtain consent from people before publishing content on their behalf.

We also delete the person's data once they deauthorise us on Facebook

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.

1. Reviewing, commenting and uploading photos of food for a stall in a specific canteen.

2. Creating meetings on a specific day for a specific time.

3. Joining and unjoining meetings as well as sharing these meetings.

Milestone 13:

Implement at least one Story involving an Action and Object in your application. Describe what actions the user has to take to trigger the Story (we will test it!) and why you think it will create an engaging experience for the users of your application.

We are not implementing an Open Graph Story because we think that the available actions and objects do not fit our application.

Instead, we make use of the geolocation API to compute the distance between the user's location and each canteen. Notice that this distance is not shown by default as it requires the user to grant us permission to access its physical location. To have a good user experience, the user has to click on the "How far away?" link to enable this feature (instead of prompting for permission out of nowhere).

Because there is no way (on most browsers) to query whether the geolocation permission has been granted (without asking for the permission, which will show the permission prompt), we are unable to automatically enable this feature on subsequent visits.

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.

SQLi - We use an ORM on the backend (Sequelize). This ORM automatically handles escaping of parameters. We are also careful in not using any raw SQL queries.

XSS - All the UI is rendered by React. React performs escaping by default (opting out requires a function call that we did not use).

CSRF - The backend server reads the authorization tokens from the Authorization header field (but not cookies). Hence, we are not vulnerable to CSRF.

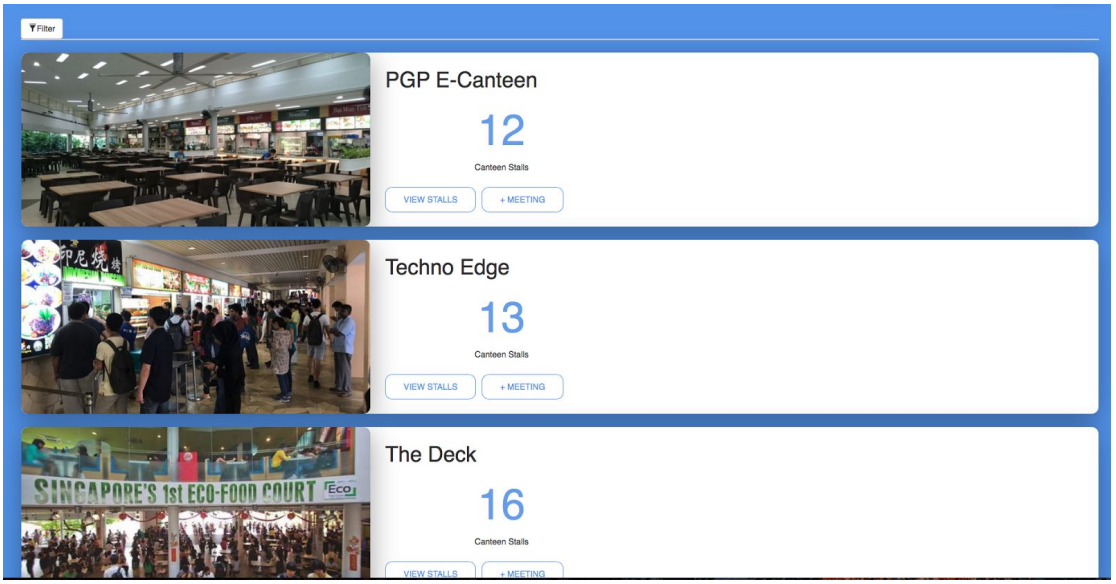
Milestone 15:

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

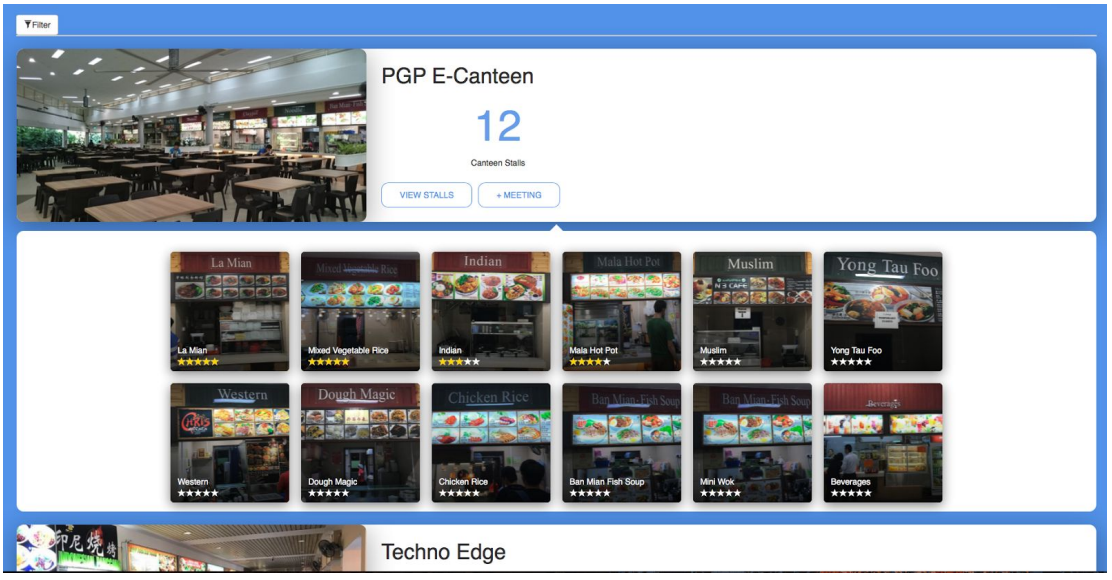
Stalls panel expansion/contraction

We chose to add this as we want to minimise the actions users have to take, for example, if they enter a canteen page, they have to press the back button in order to browse another canteen page. By adding this, we help users discover new stalls and canteens in a friendly and efficient manner.

Before Expansion



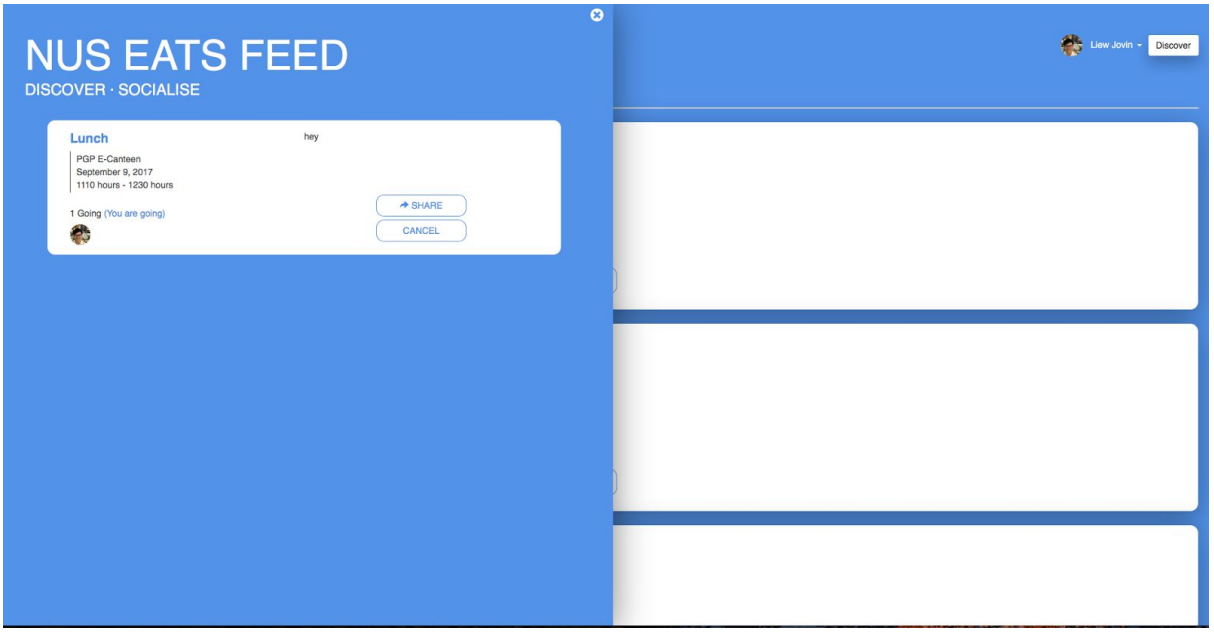
After Expansion



Feed slider

We chose to add a slider to the feed as it helps to reduce UI clutter. Instead of putting the feeds on a separate page, it allows the users to stay at the current page, and not have to switch around different pages to check out different canteens and stores.

Feed Slider



Milestone 16:

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

We make use of AJAX almost everywhere. In most cases, we are able to offer a more reactive user experience. For example, when a user clicks Join Meeting or Unjoin Meeting, we can immediately update the UI while the server request is still pending.