Teamker

http://ec2-54-169-114-178.ap-southeast-1.compute.amazonaws.com/

Group 1
Oh Han Gyeol
Fan Weiguang
Ng Jun Wei
Bai Chuan

Milestone 1:

Choice: Standalone application

A standalone application can appeal to the wider demographics, and Facebook Web Games market is already quite saturated. If we make a Facebook Web Game, we will be in a direct competition with the rest of the web games. On the other hand, if we make a standalone application, we will only be in competition with the apps in the same field(e.g. productivity, dating, communication, etc), which has a better chance of survival.

A standalone application also gives us greater control, and more flexibility for the future. As compared to a “Facebook Web Games” application, a standalone application can be integrated into other website and work with other social accounts. In this way, our target user is not limited to Facebook user only. This will be useful especially in some markets where Facebook is not the dominant social network.

Milestone 2:

Language: TypeScript(Front-end), JavaScript(Back-end)

- The most commonly familiar language in team
- As compared to JavaScript, TypeScript provides optional static typing, interfaces and classes. This gives IDEs the ability to spot errors out when user is typing, while JavaScript is dynamically typed, the type of a variable is only known when it is instantiated at run-time.

Server-side: Node.js + Express

- Lots of resources & documents available in internet
- As compared to PHP, our group members are more experienced in Node.js. Another advantage that Node.js has over PHP is that there are many useful packages from npm(node package manager) that we can use. Also, since common PHP calls are I/O blocking, while Node.js calls are asynchronous, this leads to better performance with Node.js when dealing with database.

Client-side: React

- Since react is based on component, and all the react components have a well-defined lifecycle, it makes everything easier to extend and maintain, as compared to the traditional templated or HTML directives.
- With VirtualDOM, react solves the problem that rendering to DOM is too slow and too expensive. As long as we are careful about the state updates, react has a much better performance.
- As compared to Angular, which itself is huge framework, it requires more time for us to learn. Although Angular is more featured than react, which is not even a framework but just a library, it does not give us the freedom to choose the libraries we love.

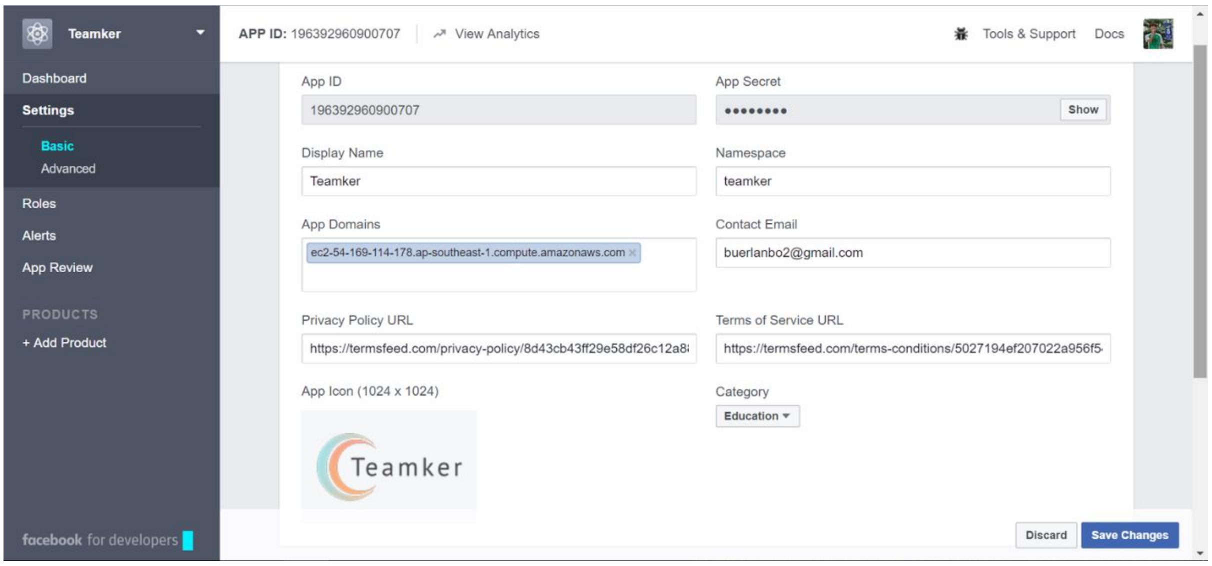
Cloud Hosting: AWS EC2 + NGINX

- AWS EC2 has a generous free-tier limit for the cloud service, with easier interface to check the billing status.
- AWS is one of the few cloud services that has a data center in Singapore. By choosing the service location to Singapore, I can expect better latency for the cloud service.
- AWS has the most straightforward set of APIs which now became the industrial standard. It is powerful and gives lots of freedom to the developer compared to, for example Heorku whose server is ephemeral in nature.

Database: AWS RDS + MySQL

- Nicely compatible with AWS EC2MySQL is more scalable than other options, like postgres

Milestone 3:



APP ID: 196392960900707

App ID: 196392960900707

App Secret: ●●●●●●●●

Display Name: Teamker

Namespace: teamker

App Domains: ec2-54-169-114-178.ap-southeast-1.compute.amazonaws.com

Contact Email: buerlanbo2@gmail.com

Privacy Policy URL: https://termsfeed.com/privacy-policy/8d43cb43ff29e58df26c12a8

Terms of Service URL: https://termsfeed.com/terms-conditions/5027194ef207022a956f5

App Icon (1024 x 1024)

Category: Education

Bonus Milestones (Between 4 and 5):

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 server-side method?

Client-side

Pros:

- Works when internet connection is lost
- Responsive, no need to wait for the response from the server
- Info is easily cacheable, can alleviate some performance issue.

Cons:

- Browser has extra works to do. If there are too many computation assigned to client-side, the application might lag.
- Exposing the logic code to the outside world, there could be a room of exploitation by hackers.
- Browser compatibility issue (a.k.a Internet Explorer)

Server-side

Pros:

- Better separation of concern. By right, the core logic should be assigned to the server, and the client-side should be sole responsible for display of data (MVC model).
- Less code for the browser, so it could lead to better performance at the client side.
- Better security, because data is stored in server instead of the browser.

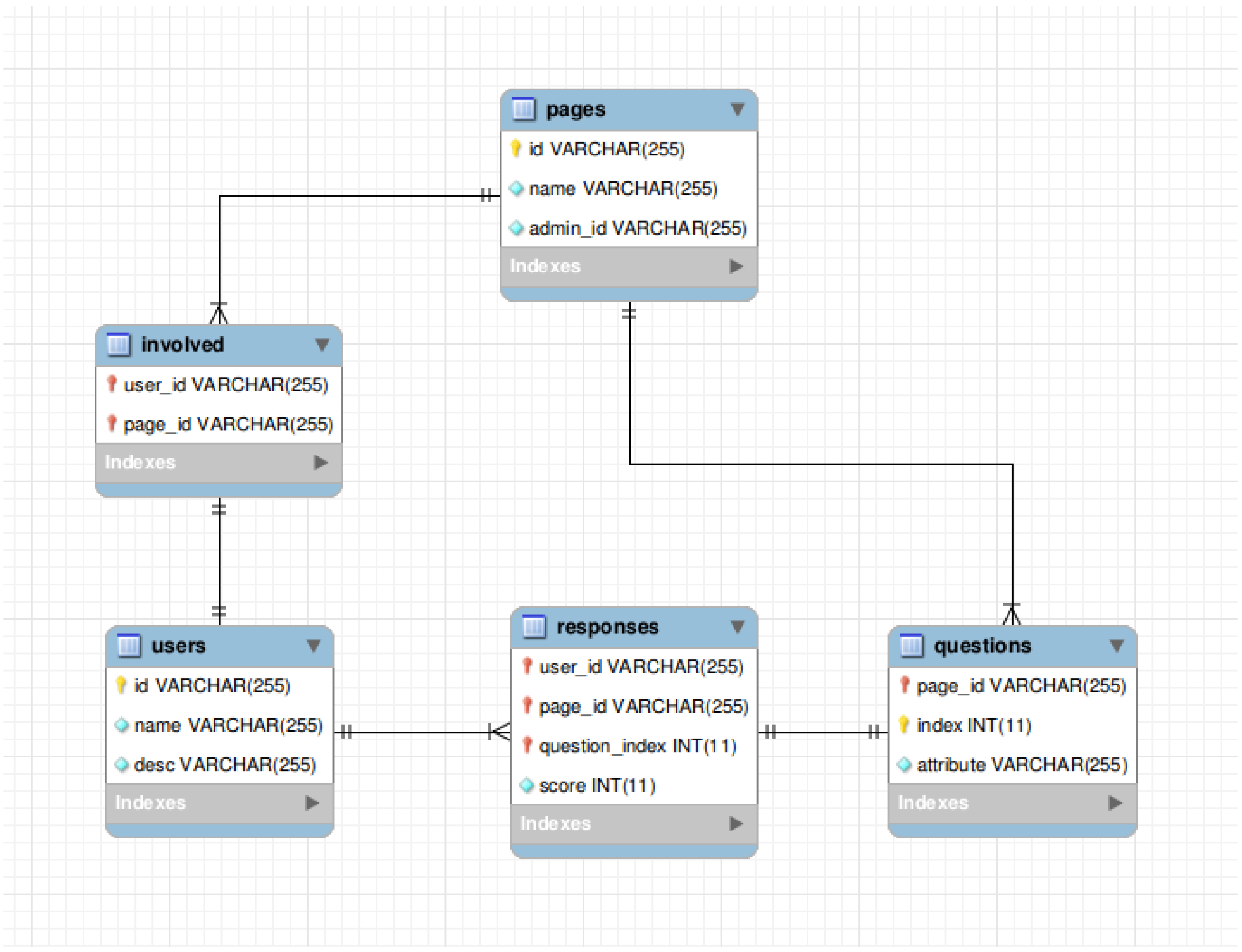
- Information hiding. Server-side implementations make it possible to show only the necessary objects to the user. For example, if there are different pages for a normal user and an administrator, the client should not derive another page from the DOM.

Cons:

- Slow compared to client-side handle. There are communication overheads between the client and the server. Especially so nowadays where browser engines (e.g. V8) are getting powerful.
- Useless in no internet connection

2. What is the primary key of the home_faculties table?
   - Tuple/Pair of (matric_no, faculty)

Milestone 5:



Milestone 6:
We are using the mysql module for node.js that allows us to implement the mysql protocol. After importing the mysql module, we initialize a connection pool which takes in our database configs and caches the connections so that it can be reused for future connections. This will enhance performance of executing commands on the database.

Add a new user:
```
INSERT IGNORE INTO Teamker.users
VALUES(user_id, user_name);
```
Insert a new user to the database with given id and name. Ignore any duplicate entry.

Get registered user list:
```
SELECT DISTINCT users.id, users.name, involved.user_desc,
GROUP_CONCAT(responses.score) AS attributes
FROM Teamker.users AS users
INNER JOIN Teamker.involved AS involved ON
users.id=involved.user_id
INNER JOIN Teamker.responses AS responses ON
users.id=responses.user_id
INNER JOIN Teamker.questions AS questions ON
involved.page_id=questions.page_id
AND questions.page_id=responses.page_id
WHERE involved.page_id='page_id'
GROUP BY users.id, involved.user_desc;
```
Get a list of registered users with given page_id. Conduct inner join between users, involved, questions, and responses tables and select user_id, user_name, user_desc, .

Get unregistered user list:
```
SELECT DISTINCT users.id, users.name, involved.user_desc,
questions.page_id
FROM Teamker.users AS users
INNER JOIN Teamker.involved AS involved ON
users.id=involved.user_id
LEFT JOIN Teamker.responses AS responses ON
users.id=responses.user_id
AND responses.page_id=involved.page_id
LEFT JOIN Teamker.questions AS questions ON
responses.page_id=questions.page_id
AND involved.page_id=questions.page_id
WHERE involved.page_id='page_id'
```

```
AND questions.page_id is NULL;
```
Get a list of unregistered users with given page_id. Conduct inner join with involved table, and left join with questions and responses tables. Unregistered users will have null in their page_id value. Select user_id, user_name and user_desc.

Get user data and responses:

```
SELECT users.id, users.name, involved.user_desc,
GROUP_CONCAT(responses.score) AS attributes
FROM Teamker.users
INNER JOIN Teamker.responses ON users.id=responses.user_id
INNER JOIN Teamker.involved ON users.id=involved.user_id
WHERE responses.page_id='page_id'
GROUP BY users.id, involved.user_desc;
```

Get a list of users and their name and responses to the questions with given page_id. Response scores are concatenated into an array, using GROUP_CONCAT command.

Milestone 7:

Example 1:

```
FB.api(
          "/me/groups",
          function (response: any) {
                             let groupList: Group[] = [];
              if (response && !response.error) {
                  /* handle the result */
              }
          }
      );
```

This graph query is to get all the groups that the user manages (the user is an admin of that group), and those groups will be shown in the entry page which the user can choose which group to manage as an admin. This group list will also be shown in the side panel in admin page. This query requires ‘user_manage_group’ permission.

Example 2:

```
FB.api(
          "/" + groupId + "/members",
          function (response: any) {
              let memberList: User[] = [];
              if (response && !response.error) {
                  /* handle the result */
              }
          }
      );
```

This graph query is to get all the members (include the admin himself) of a certain group. After an admin registers a group (create a question form), the members of the group will be extracted from this query and persisted to database together with the group id and the question form. So next time when a user log in, the group name will be shown to him if he is a member of the group.

Example 3:

```
var photolink = ("http://graph.facebook.com/" + this.props.userId +
"/picture?type=square")
```

This query is to get the profile picture of the current user and display that in the top bar of our website.

Milestone 8:

In our website, we have implemented a feed which is a share dialog that pops up when an admin successful create the question form for a group. We think that after the admin create the form, It is likely that he wants to announce in the facebook group that the form is available for everyone in the group, therefore we think it is valuable and intuitive to have a share dialog pop up for the admin. If he does not want to share the group on his facebook, he can just close the dialog.

As for the member side, initially we want to pop up a share dialog after the user has completed the questions. However, it might be annoying when the user finishes the long list of questions and really wants to see the result of matching but he sees the share dialog blocking his results.

Also it is more natural to ask the user to share the website after he actually gets what he wants: the match result.
Therefore we put a share button on top to provide the share option for the user, so he can click on it whenever he feels satisfied about this website.

Apart from that, we did not find any other places that are very suitable to inserts some feeds.

Milestone 9:

Our application has a like page on the very top of the page beside our App logo. We feel that this is the best place to place the button because it is the most visible. Users can also see their friends who have like the application and may be compelled to do so as well!

Milestone 10:

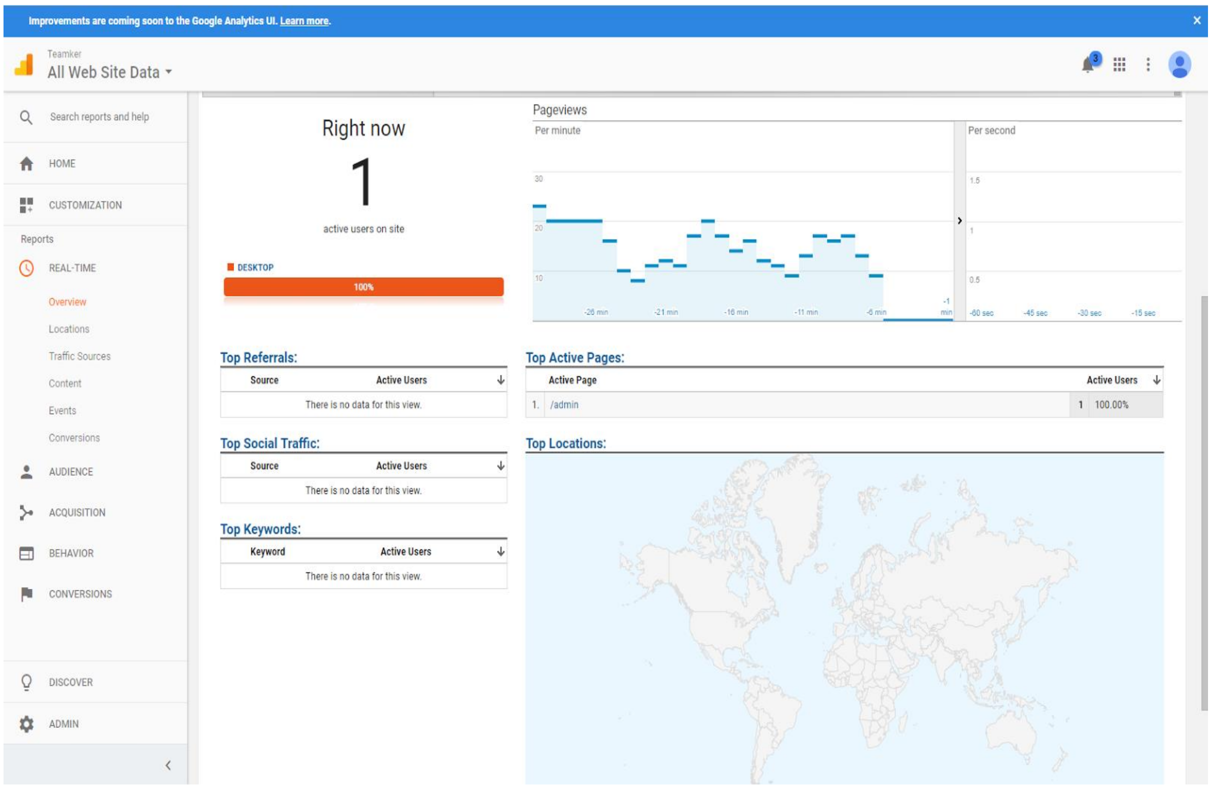
For our application, only the admin can delete his page from teamker, upon doing so, the links between the users and the application will be terminated. As such, even though the users are still in the database, there will not be a link to them and as such others will not be able to access them through conventional means.

As such, it is similar to one of the subclauses of Facebook:

When you delete IP content, it is deleted in a manner similar to emptying the recycle bin on a computer. However, you understand that removed content may persist in backup copies for a reasonable period of time (but will not be available to others).

Furthermore, we are taking it a step further as we intend to run a cron job everyday to clear users who are inactive for more than a set amount of time. This will remove user data for those who have deleted the app or are no longer using it.

Milestone 11:



Milestone 12:

User interaction 1: Signing up a team for Teamker

For an administrator, signing a team up for Teamker is really easy. We wanted the on boarding process to be as easy as possible. Therefore, we directly pulled the admin's group onto the view so that the admin only has to select his group. We also wanted the admin to do as little work as possible, so for each question, we only expect the admin to type a single word or phrase as the attribute. Lastly, the match type is also a toggle so that the admin can switch between most different and most similar with just a click.

User interaction 2: Answering the survey

Besides the admin, we also want the users to have an easy time using the app. As a result, we made the survey so that the user only has to slide a slider to select his preference instead of typing out his answers. Also, the slider is intentionally initialized at a value of 5 so that people who are not sure of their abilities have an easier time answering

User interaction 3: Viewing the matching result

After answering the survey, the users are able to view their best matches in order of how closely they match them as well as they names and profile pictures. This helps to facilitate group forming because users may not know each other’s names and faces upon first entering the module and as such this will help them find the ideal teammate with much greater ease.

Milestone 13:

For the open graph story, since the custom action and object are deprecated in the recent version of graphic api, only the common action and object are available for use.

We went through all the actions and objects, and found out that there was not any pair that was suitable for our application.

Even the most common one which every app can use -- “og,likes” does not fit in very well because of two reasons: firstly we have an like button on top, it is quite repetitive if we ask users to like again somewhere else.

Secondly due to the nature of our app, a user usually is able to repeat the same routine again because he can have multiple groups. For example, he can click in to a group, fill up the question form, check the match result, and he can do exactly the same again by starting with another group. Therefore it’s likely the open graph story will be invoked multiple times for the same user. However the user can only like the website once, so ‘og.likes’ is not suitable.

Milestone 14:

There are two recommended ways to stop CSRF, which is to:

1. Check standard headers to verify the request is same origin
2. AND Check CSRF token

In order to prevent XSS, the URI has to be sanitized by the client-side code to prevent malicious code from running through “get” requests

For our application, we implemented the various measures to protect ourselves from CSRF, XSS and SQLi:

CSRF -> We do not allow cross origin requests on production mode. This means that our server will only respond to requests from within the same origin, as such, it will protect to a certain extent against malicious code aiming from an outside source attempting to use a user’s authentication token.

XSS ->
By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks.

SQLi -> All our inputs that are meant to interact with the server are escaped and placeholders are used in the actual SQL queries. This protects from SQL injections as the inputs are treated very carefully so that they are never given the chance to run or tamper with any state.

Milestone 15:

Highlight on hover
We decided to add a highlight when a user hovers over a user response. This allows the user to have a clearer focus on what he is about to click and will prevent confusion as the user will realise that the element is clickable.

Loading spinner
While the app is loading, we utilise a loading spinner to tell the user that the application is still working fine and that his computer has not hanged

and is merely waiting for a response. This allows UX to improve as the user is not confused and know exactly what is happening in the app.

Milestone 16:

AJAX requests are used for facebook login, submit response and fetch questions

When the questions are fetched, it is done through an AJAX request. Since AJAX requests are asynchronous and thus non blocking, this allows the app to load all the other different components such as the assets without needing to wait for the response from the server. As a result, most of the template is rendered before the response from the server arrives. After which, the response will trigger the callback function to render the response as part of the template so that the user experience appears very smooth even with the slower response time of the AJAX call.