



Team 2 Milestones Submission

A0141138N

A0136070R

A0126172M

A0148038A

Apoorva Ullas

Lim Jia Yee

Won Jun Ru Daphne

Li Zihan

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.

We believe that creating a standalone application will give us greater flexibility in developing our application. We want to keep the option of expanding our user base to other social media users, or those without Facebook accounts in the future. The feeds of our web-application will be hosted on Facebook and what our application offers is the anonymity of the user for both posting and replying.

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.

First, we used a full-stack web framework, because frameworks simplify the development of the app and let us focus on the high-level architecture rather than the low-level, basic, repetitive tasks like socket programming.

SERVER

We used the Laravel PHP framework for our server-side development. We do not want to choose microframeworks such as Flask and Sinatra as they are intended for smaller projects with specialized purposes. For example, for a static website, if we need a simple backend service for one or two forms, we can send the forms from client-side to a Flask backend.

In addition, we can expect more unnecessary labour in configuring microframework apps with the amount of non-trivial features we intend to expand towards. On the other hand, frameworks such as Laravel and Ruby on Rails support the philosophy of "convention over configuration". We can then focus our time on actually developing the app and leave the standards and specifications to the framework. This saves our time, and also helps with debugging and future handover. The community of both Ruby on Rails and Laravel are large, and with most of them sticking with the conventions, we can expect to debug our problems faster, without the distraction of configuration, setup, and variables.

As for the other frameworks like Django, NodeJS, and so on, we will get more flexibility from them compared to the aforementioned two MVC frameworks above. However, we foresee a steeper learning with these due to lack of familiarity. The MVC architecture, on the other hand, is understood well by the team.

In the end, we chose Laravel over Ruby on Rails and other PHP frameworks. PHP is easier to pick up for us as all of us are proficient with HTML, CSS, and JavaScript, but only one of us is familiar with Ruby. And as for Laravel, we appreciate that it emphasises "convention over configuration". Such philosophy are not

emphasised in other PHP frameworks, which could lead to us having problems to debug (less helpful answers on the internet).

CLIENT

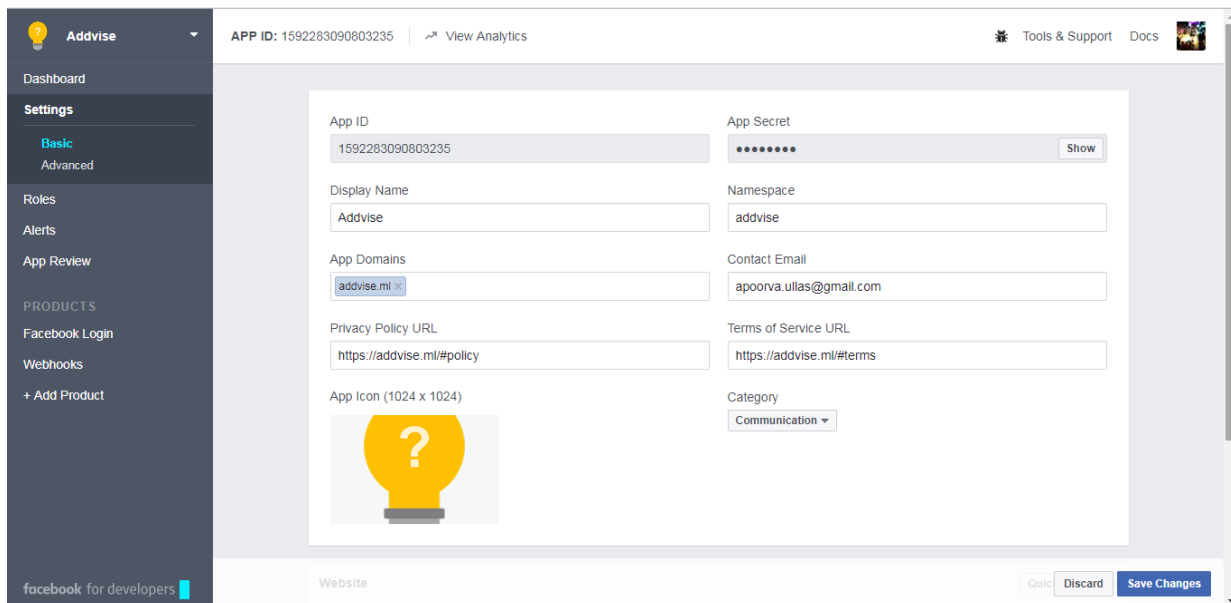
We picked bootstrap as it fits well with Laravel framework. It contains some templates based on HTML and CSS, which makes it efficient to design user interface and implement front-end development. The interface is really pretty and easy to pickup that we have already implemented.

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.

Icon: 

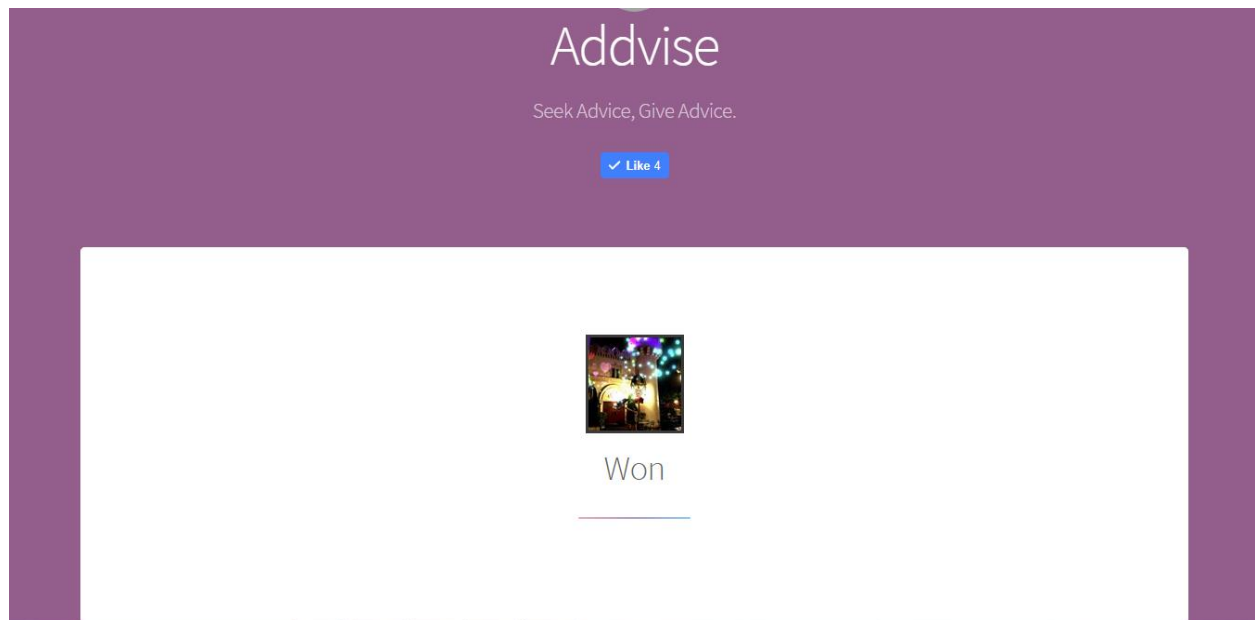
The icon in simplicity is actually a light bulb with a question mark inside. The question mark implies that users can feel free to ask questions and seek for advice in this application. The light bulb signifies the advice and tips given by other users, which could guide the life of those who request for advice.

Screenshot of latest Dashboard:

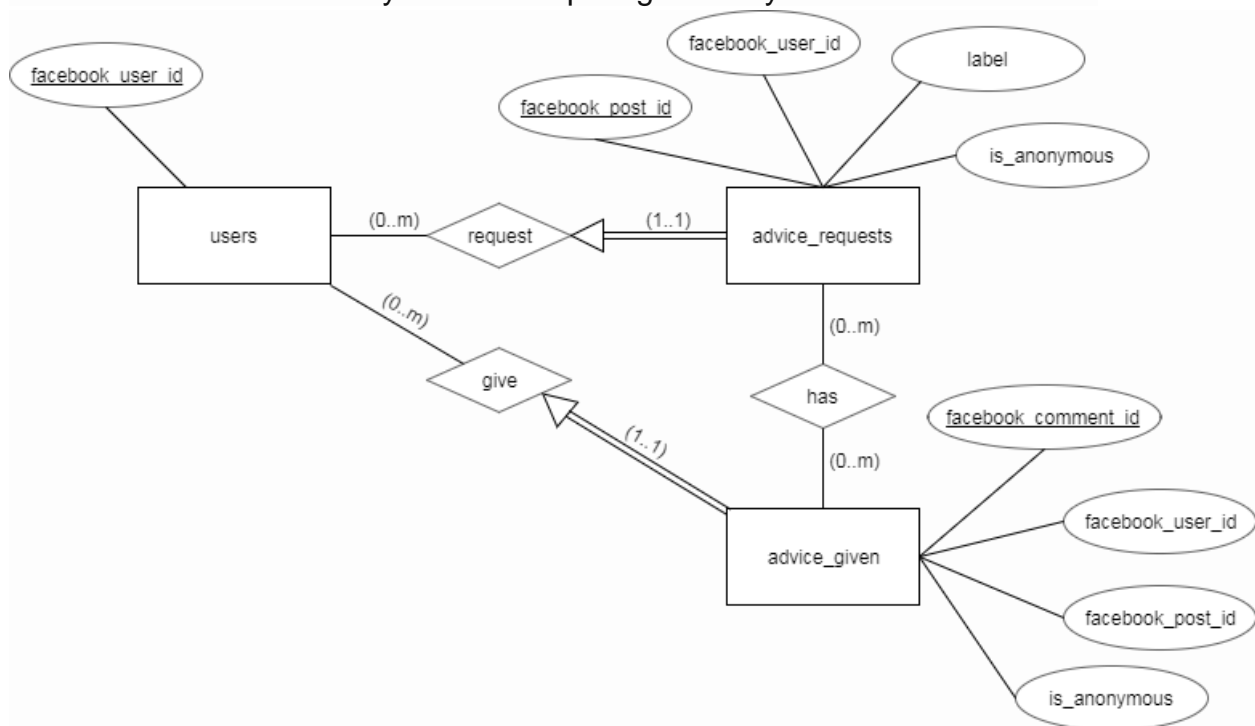


The screenshot shows the Facebook Developer Dashboard for an application named 'Advise'. The left sidebar contains navigation links: Dashboard, Settings (Basic, Advanced), Roles, Alerts, App Review, and PRODUCTS (Facebook Login, Webhooks, + Add Product). The main content area displays the 'App Details' page for 'APP ID: 1592283090803235'. The form includes fields for App ID, App Secret (with a 'Show' button), Display Name (Advise), Namespace (advise), App Domains (advise.ml), Contact Email (apoorva.ullas@gmail.com), Privacy Policy URL (https://advise.ml/#policy), Terms of Service URL (https://advise.ml/#terms), App Icon (1024 x 1024) (a yellow lightbulb with a question mark), and Category (Communication). At the bottom, there are 'Quit', 'Discard', and 'Save Changes' buttons.

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.



Milestone 5: Draw an Entity-Relationship diagram for your database schema.



Breakdown of tables

Users	
Description	To store user settings
PRIMARY KEY	<i>facebook_user_id</i>

Advice_Requests (equivalent to Facebook posts on our App page)	
Description	To store the advice posts
PRIMARY KEY	<i>facebook_post_id</i>
FOREIGN KEY	<i>facebook_user_id</i> reference to table Users <i>facebook_user_id</i>
Remarks	We store the ID so that we can fetch the relevant and updated posts from Facebook quickly.

Advice_Given (equivalent to Facebook comments on our App page)	
Description	To store the facebook comments given to the advice
PRIMARY KEY	<i>facebook_comment_id</i>
FOREIGN KEY	<i>facebook_user_id</i> reference to table Users <i>facebook_user_id</i>
FOREIGN KEY	<i>facebook_post_id</i> reference to table Advice_Requests <i>facebook_post_id</i>
Remarks	Comments can only be owned by one user and each comment can be only under one Advice.

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.

1. Inserting advice request form values into advice_requests table:

```
INSERT INTO advice_requests (fb_post_id, fb_user_id, label,  
is_anonymous)  
  
VALUES ($fb_post_id, $fb_user_id, $label, $is_anonymous)
```

2. Retrieve all details about requests from advice_requests table to show in “Give Advice” section order by reverse-insertion order (because ID is incrementing)

```
SELECT * FROM advice_requests ORDER BY id DESC;
```

3. Inserting given advice form values into advice_given table:

```
INSERT INTO advice_given (fb_comment_id, fb_user_id, fb_post_id,  
is_anonymous)  
  
VALUES ($fb_comment_id, $fb_user_id, $fb_post_id, $is_anonymous)
```

4. Retrieve advice from advice_given table to show in “Give Advice” comments section written by anyone whose fb_user_id is in \$fb_user_id_arr. We do batch queries like this so that we do not need to do X queries for X users.

```
SELECT fb_comment_id FROM advice_given WHERE fb_user_id in  
$fb_user_id_arr;
```

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

Example 01: Posting requests for advice on our Facebook Advise page

Example 02: Retrieving requests and comments from Facebook Advise page

Example 03: Posting comments for advice on our Facebook Advise page

Example 04: Displaying username and profile picture of user in “My Profile”:

```
Query for username: FB.api('/me', function(response) {  
  callback(response.name);  
});  
  
Query for profile picture:  
  
FB.api('/me/picture?type=normal', function(response) {  
  callback(response.data.url);  
});
```

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.

Our feeds are restricted to posts made on our Facebook Advise page by advice givers and advice seekers. We choose not to have any feeds published on our user’s timeline as we provide a personalized “My Requests” and “Advice I Gave” section for users to track their personal requests and advises on our application. We also ensure there will not be spam posts by restricting a single user to maximum post 1 time every 5 minutes.

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 placed our Like button below our logo. It follows the symmetrical concept of centering the button and this works well in both web and mobile version. In addition, we have to reiterate that the fundamental concept of our application is to post. Users will be able to see the like button the first thing when they visit our website.



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.

We had a few plans for offboarding, but we were unable to implement them on time. We only managed to set up a webhook endpoint which subscribes to “permissions”, but we had trouble validating the SHA1 signature (They sign with escaped unicode and we could not get the correct PHP hmac(‘sha1’, ...) option (several bitmasks) to sign and verify).

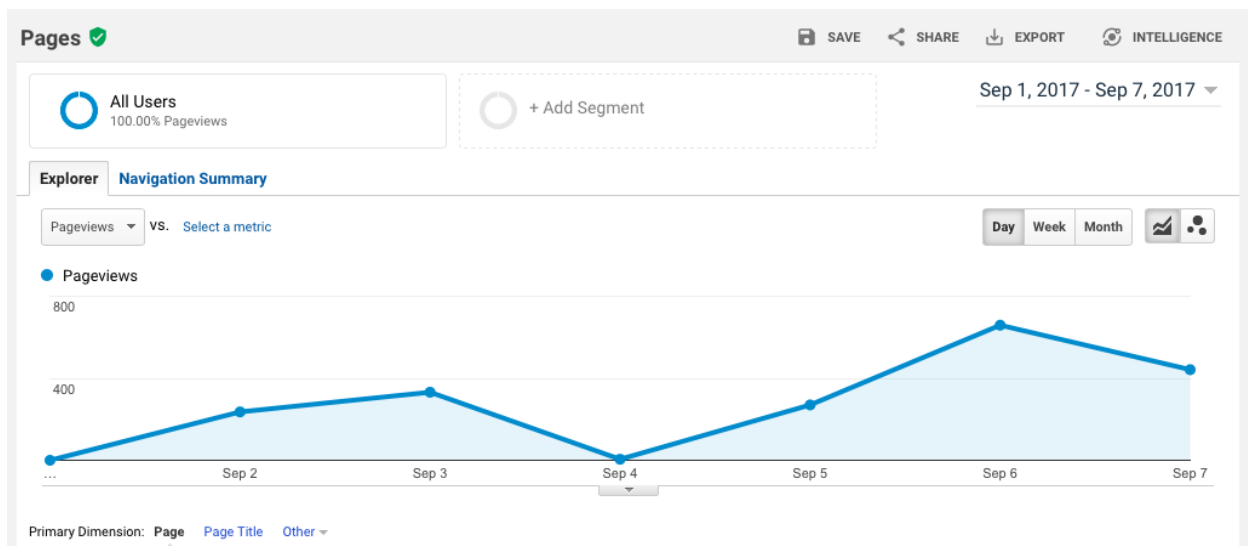
Plan #1: Use Webhooks and detect when a user de-authorizes our application.

Plan #2: Always set expiring keys for users with each activity (This plan is computationally more intensive.)

And then evict the user out of our database after 3 months or so. Eviction entails:

- 1. Delete from the “users” table**
- 2. Need to conduct cascade deletion on “advice_requests” and “advice_given”**
- 3. From (1) and (2), we have deleted everything we know about the user**

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!



Plot Rows

Secondary dimension

Sort Type:

Default

advanced

<input type="checkbox"/>	Page	Pageviews	Unique Pageviews	Avg. Time on Page	Entrances	Bounce Rate	% Exit	Page Value
		1,944 % of Total: 100.00% (1,944)	347 % of Total: 100.00% (347)	00:01:19 Avg for View: 00:01:19 (0.00%)	144 % of Total: 100.00% (144)	15.97% Avg for View: 15.97% (0.00%)	7.41% Avg for View: 7.41% (0.00%)	\$0.00 % of Total: 0.00% (\$0.00)
<input type="checkbox"/>	1. /	1,416 (72.84%)	140 (40.35%)	00:01:09	130 (90.28%)	11.54%	5.23%	\$0.00 (0.00%)
<input type="checkbox"/>	2. /home	180 (9.26%)	87 (25.07%)	00:01:18	1 (0.69%)	100.00%	16.67%	\$0.00 (0.00%)
<input type="checkbox"/>	3. /ask	98 (5.04%)	28 (8.07%)	00:02:14	1 (0.69%)	0.00%	9.18%	\$0.00 (0.00%)
<input type="checkbox"/>	4. /needAdvise	77 (3.96%)	24 (6.92%)	00:01:43	0 (0.00%)	0.00%	12.99%	\$0.00 (0.00%)
<input type="checkbox"/>	5. /needAdvise/advice/me	38 (1.95%)	11 (3.17%)	00:01:54	0 (0.00%)	0.00%	5.26%	\$0.00 (0.00%)
<input type="checkbox"/>	6. /policy	34 (1.75%)	10 (2.88%)	00:01:41	0 (0.00%)	0.00%	17.65%	\$0.00 (0.00%)
<input type="checkbox"/>	7. /trending	30 (1.54%)	15 (4.32%)	00:04:25	11 (7.64%)	63.64%	33.33%	\$0.00 (0.00%)
<input type="checkbox"/>	8. /nojs	21 (1.08%)	7 (2.02%)	00:01:59	1 (0.69%)	0.00%	0.00%	\$0.00 (0.00%)
<input type="checkbox"/>	9. /needAdvise/me	15 (0.77%)	4 (1.15%)	00:01:59	0 (0.00%)	0.00%	0.00%	\$0.00 (0.00%)
<input type="checkbox"/>	10. /me	13 (0.67%)	9 (2.59%)	00:02:34	0 (0.00%)	0.00%	15.38%	\$0.00 (0.00%)

Show rows:

10

Go to:

1

1 - 10 of 17

This report was generated on 9/8/17 at 9:30:46 PM - [Refresh Report](#)

Screenshot above shows the analytics for all our different page views.

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. More input control provided to users requesting for advice. Dropdown has a few broad categories with further sub-categories to make it easier for users to select the appropriate category their request falls into.

Ask for Advice

Type of advice needed

✓ Uncategorized

Academics

Academics

Internships

Overseas

Career

Certification

Companies

Interviews

Personal

Finances

Health

Lifestyle

Social

Bullying

Family

Friendships

Relationships

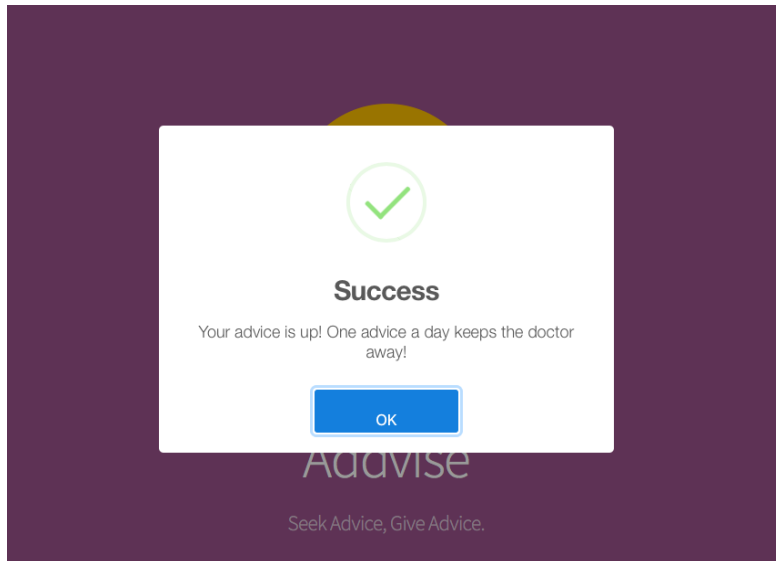
2. Hamburger navigation bar

Video showing responsive hamburger navigation bar:

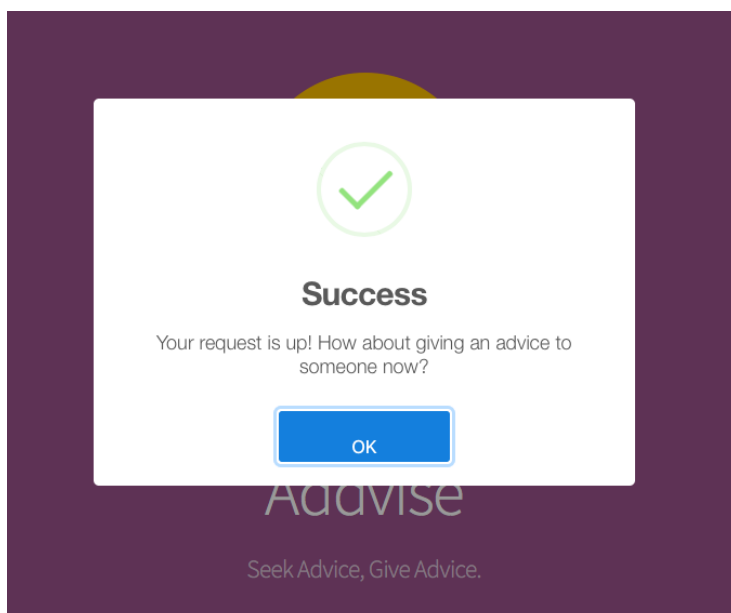
<https://drive.google.com/open?id=0BxGjlsC7FaauRkF4dXFFdFlfUEk>

3. Provide popup alerts to guide users on their actions:

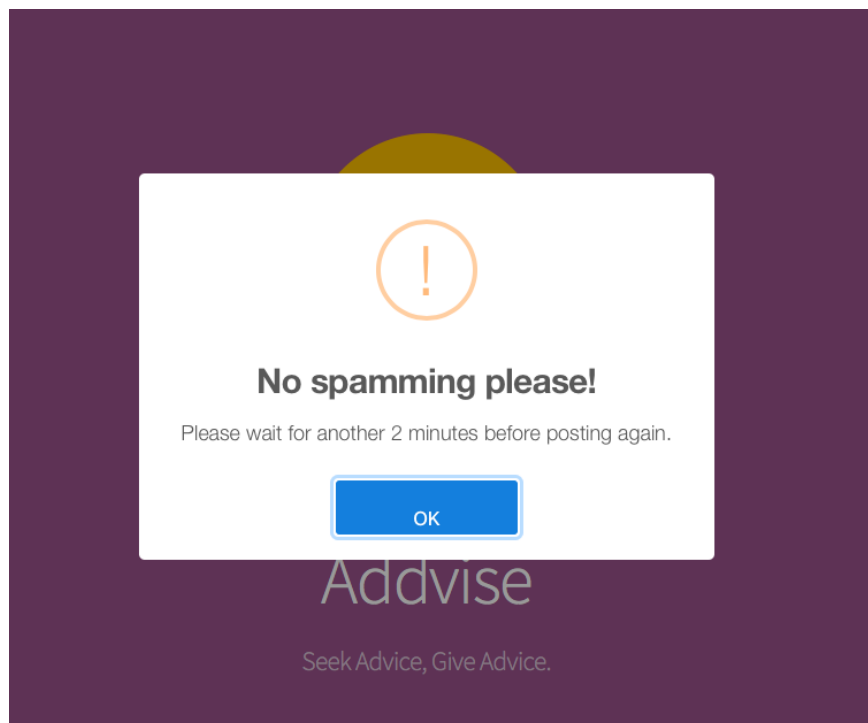
1. For people giving advice:



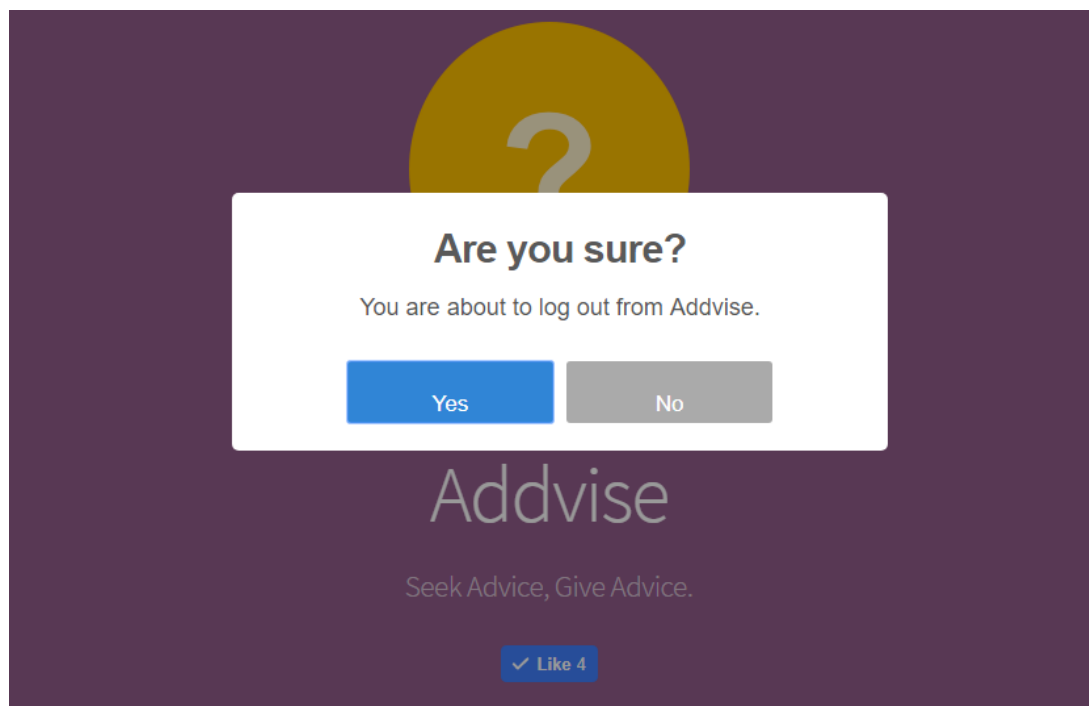
2. For people seeking for advice:



2. For spammers:



3. For people contemplating whether to log out (please try both options for a cute surprise)



Milestone 13 alternative: In replacement of this milestone, we have bonus features:

1. **Automatic redirecting of all HTTP requests to HTTPS** This is done by having two server blocks for 80 and 443 separately. We used LetsEncrypt to enable SSL on our website.

```
ec2-user@ip-172-31-45-33:~
1 server {
2     listen 80;
3     listen [::]:80;
4     server_name advise.ml www.advise.ml;
5     return 301 https://$server_name$request_uri;
6 }
7
8
9
10 server {
11     listen 443 ssl;
12
13     charset utf-8;
14
15     server_name advise.ml www.advise.ml;
16     root /usr/share/nginx/advise/public;
17     index index.php;
18
19     ssl_certificate /etc/letsencrypt/live/advise.ml/fullchain.pem;
20     ssl_certificate_key /etc/letsencrypt/live/advise.ml/privkey.pem;
21
22     location = /favicon.ico { access_log off; log_not_found off; }
23     location = /robots.txt { access_log off; log_not_found off; }
24
25     access_log off;
26     error_log /var/log/nginx/advise.log error;
27
28     # Remove index.php
29     if ($request_uri ~* "(.*)/index\.php$") {
30         return 301 $1;
31     }
32
33     # Remove trailing slash
34     if (!-d $request_filename) {
35         rewrite ^/(.*)/$ /$1 permanent;
36     }
37
38     # Clean Double Slashes
39     if ($request_uri ~* "\/\//") {
40         rewrite ^/(.*) /$1 permanent;
41     }
42
43     location / {
44
45         # First try to serve files
46         try_files $uri $uri/ =404;
47     }
48 }
49
50 # /etc/nginx/conf.d/advise.conf" [readonly] 71L, 1694C
1,1 Top
```

2. **Failover for (1)** If redirection fails somehow (slim chance), we have created a pretty error page to inform users where to go. We absolutely don't want to end up here, but we are always prepared for corner cases.



Say no to port eighty

Please head over to <https://advise.ml> instead.

Thanks!

— Advise



3. **Mobile Friendly.** Our hamburger navigation bar is auto-activated at a reduced screen size. Our text, screen size also auto resize to fit mobile devices.
4. **Personalized Posts data.** We created a “My requests” and “Advice I gave” page to give a more personalized page for the users. User can keep track of their posts without scrolling through hundreds of posts.

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.

Laravel's Eloquent ORM uses PDO parameter binding to avoid SQL injection. Parameter binding ensures that malicious users can't pass in query data which could modify the query's intent. Without PDO parameter, a SQL query would look like this: `SELECT * FROM users WHERE id = 'jane' or 1=1`. Hence, a malicious user can input 'jane'; drop table users; which would result in the following query: `SELECT * FROM users WHERE id = 'jane'; drop table users;` causing the whole database table user to be deleted. However, with PDO parameter, the resulting query will be: `SELECT * FROM users WHERE id = 'jane or 1=1'`, preventing any possible SQL injection attacks by escaping the quotes.

CSRF tokens are used to ensure that attackers cannot initiate a request impersonating the user. This is done by generating a token that must be passed along with the form contents. This token will then be compared with a value additionally saved to the user session. If it matches, the request is deemed valid, otherwise it is deemed invalid. Since we use the laravelcollective package to construct our forms, the CSRF token is automatically added to the forms.

Laravel's `{!!}` syntax will automatically escape any HTML entities passed along via a view variable, preventing XSS attacks. Hence, any scripts passed in through the form fields will not be executed, as the `<script>` tag will be replaced with `<script>`;

We also redirect all HTTP requests to switch to HTTPS so as to protect users data in transport (:

OPTIONAL

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)

Animation 1: The logo

Link to view animation:

<https://drive.google.com/open?id=0BxGjlsC7FaauNWhMMEhxbW5kVHc>

Animation 2: Hamburger Icon

Link to view animation:

<https://drive.google.com/open?id=0BxGjlsC7FaauRkF4dXFFdFIfUEk>

Animation 3: Pop-up alerts:

Link to view animation:

<https://drive.google.com/open?id=0BxGjlsC7FaauVDgyRGN3M19KUEk>