journey

Group 2 | Chua Yao Hui | Lee Kai Yi (pm) | Tan Jia Min Michelle | Ten Zhi-Yang

Milestone 1

We chose to develop our application as a standalone one because we felt the need for more control over our technology stack. We wanted to leverage on existing web frameworks to create a seamless experience for those using our application, regardless of their platform.

We considered using Facebook's Canvas to build our application[1] and noted that doing so could reduce load times. Canvas has been largely used to develop Facebook games, and we discovered that the very same games were usually launched as standalone applications (as opposed to Canvas ones) when we tried to access them on mobile platforms.

We finally decided against building our application on Canvas because it would have compromised the user experience. Having the Facebook interface wrapped around our application would have created a jarring, distracting aesthetic.

Milestone 2

Now we will describe our technology stack. Our choice of frontend, backend and database toolsets are shown in Figure 1.

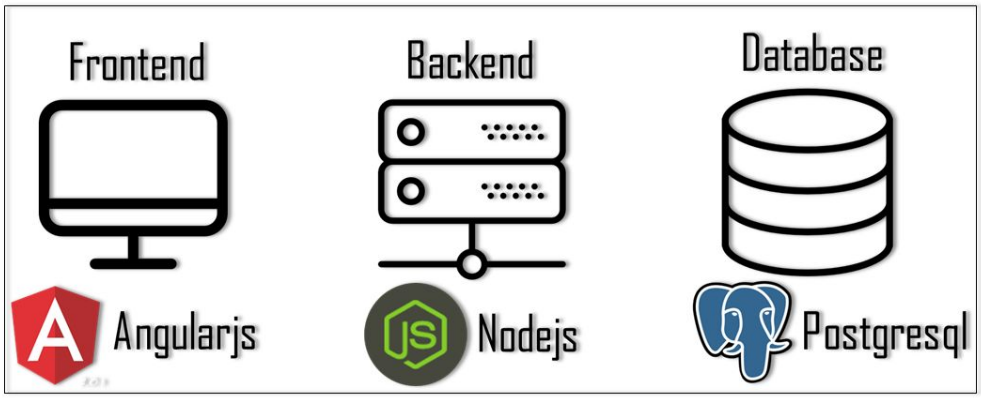


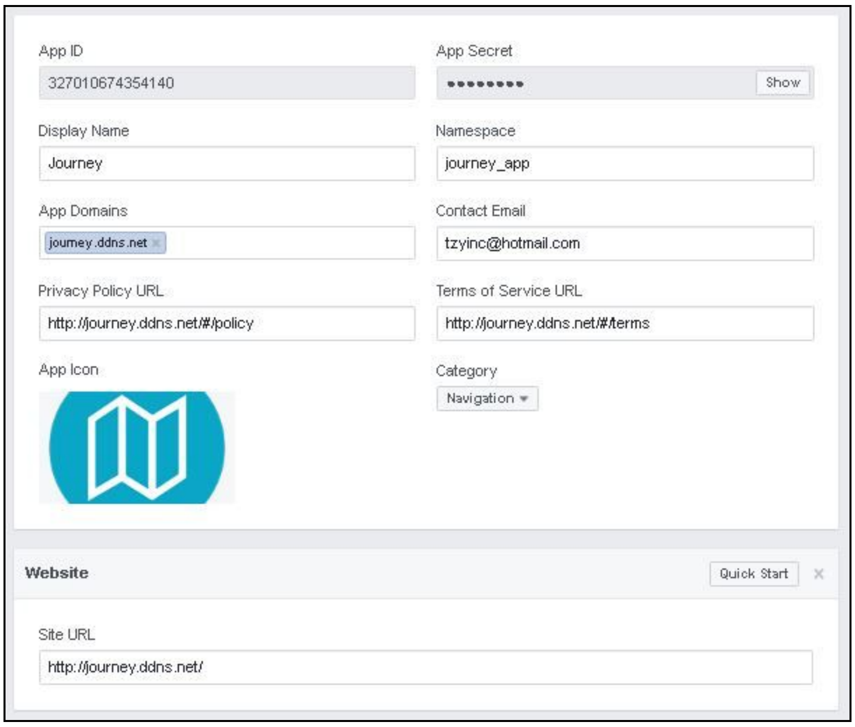
*Figure 1: Chosen toolsets for Journey**

We considered both Laravel and NodeJS as potential candidates for server-side frameworks. Both frameworks make it simple for us to create our server, build routes, and use ORM tools. However, NodeJS allows us to work purely in JavaScript, facilitating code-sharing between both the front and back end. In particular, the Express library provides us with a framework to handle routes easily. The node package manager also offers a powerful range of tools for development, such as *morgan* (for logging client requests) and *bluebird* (for handling asynchronous operations). Given that all of us on the team were familiar with JavaScript, it became the obvious choice for our backend.

Our preference for JavaScript naturally carried itself to the frontend. We chose to work with AngularJS. Angular's use of *directives*, *controllers* and *factories* made it very simple for us to manage the view-scopes of the various variables we wanted to display. Since we set out to build a single-page application (to avoid burdening the client browser with full page reloads), Angular's ui-router routing framework was the clear choice as it allowed us to bind the aforementioned directives, controllers and factories together as *states*. We considered adopting React, but none of us were too familiar with the framework. We felt that Angular was a perfectly reasonable option that suited the tree-like structure of our data (Users > Journeys > Posts), especially with its tight scoping and templating power.

<u>Milestone 3</u>

The details of our application can be seen below (Figure 2).



App ID: 327010674354140	App Secret: ••••••••
Display Name: Journey	Namespace: journey_app
App Domains: journey.ddns.net	Contact Email: tzyinc@hotmail.com
Privacy Policy URL: http://journey.ddns.net/#/policy	Terms of Service URL: http://journey.ddns.net/#/terms
App Icon	Category: Navigation

Website

Site URL: http://journey.ddns.net/

Figure 2: Facebook application details page for Journey

<u>Milestone 4</u>

When users log in through Facebook to the Journey app, they are able to see their name, and their profile picture at the top of the (collapsable) navbar.

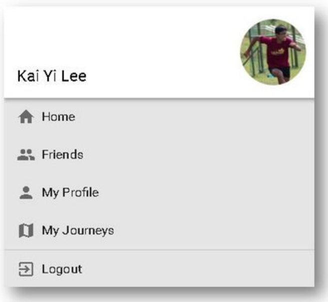


Figure 3: Screenshot of side navbar after integrated with Facebook

Our application uses the JS SDK to authenticate users to Facebook. We chose to work the the JS SDK because of its compatibility with browser code, and simplicity of usage. We did not use the PHP SDK as our server had little need to communicate with Facebook directly. We decided to only run API calls from the JS SDK to optimize loading speeds for the users, so that there will not be server propagation lag in our API calls.

On top of allowing us to make efficient Graph API calls, we further integrated Facebook in our application by allowing the user to generate dialogs to directly share their journeys to the Facebook news feed (as outlined in later milestones).

Milestone 5

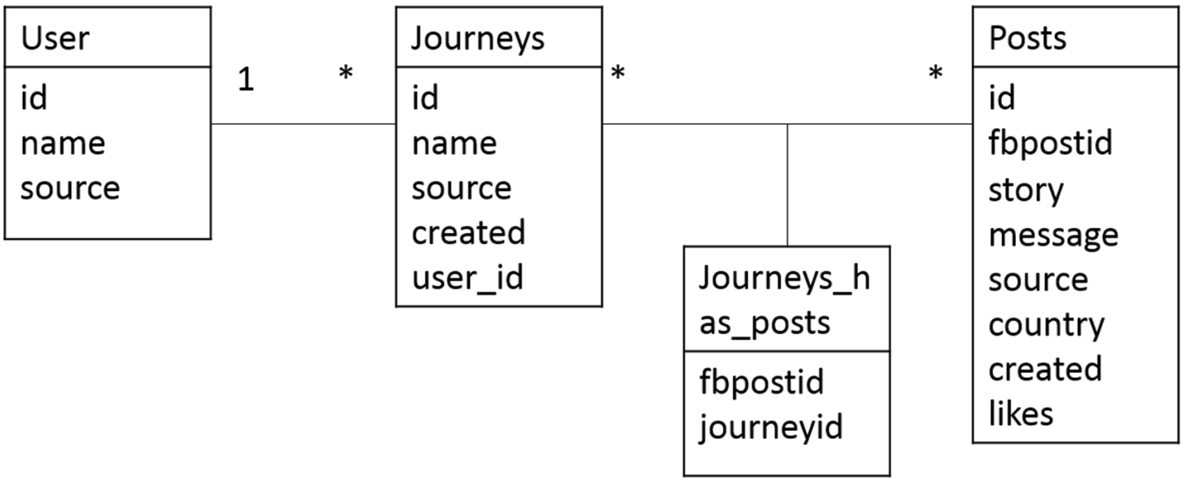


Figure 4: Database schema of Journey

Our schema was designed to be straightforward and efficient. Most of the information that we care about is carried on the Posts table, so one can think of Users and Journeys as "wrappers" which carry very little information by themselves. Indeed, we would care about a user's (or journey's) name and image because we need some information to package journeys (or posts), but that is about all we need for those models. With only three tables in our database, information can be easily accessed through ORMs such as Sequelize.

Milestone 6

Aim	**Sequelize Query**	**Actual SQL**

Get all of a single user's data. This data is used on the page that displays a user's name profile picture, and all of a his/her journeys.	`User.findOne({ where: { id: “123” }, include: [{ model: Journey, include: [Post]}]`	`SELECT * FROM "users" AS "user" LEFT OUTER JOIN "journeys" AS "journeys" ON "user"."id" = "journeys"."user_id" LEFT OUTER JOIN ("journeypost" AS "journeys.posts.journeypost" INNER JOIN "posts" AS "journeys.posts" ON "journeys.posts"."id" = "journeys.posts.journeypost"."post_id") ON "journeys"."id" = "journeys.posts.journeypost"."journey_id" WHERE "user"."id" = '123';`
Get one journey. This data is used to display all of the posts belonging to a journey.	`Journey.findOne({ where: { id: “123” }, include: [Post]})`	`SELECT * FROM "journeys" AS "journey" LEFT OUTER JOIN ("journeypost" AS "posts.journeypost" INNER JOIN "posts" AS "posts" ON "posts"."id" = "posts.journeypost"."post_id") ON "journey"."id" = "posts.journeypost"."journey_id" WHERE "journey"."id" = '1';`
Create a journey. This data is persisted onto the database when the user confirms a new journey instance.	`Journey.create({ name: "Ten Zhi Yang", source: "http://123.com/ten.jpg" , created: 2016-09-01 })`	`INSERT INTO "journeys" ("id","name","source","created","user_id") VALUES (DEFAULT,'Singapore','https://pixabay.com/get/e834b20d2bf1093ed95c4518b74f4794ea7ee7d404b015469 7f7c27ca0eeb5_640.jpg','2016-09-01 11:13:53.000 +00:00','1135531139835623') RETURNING *;`

Milestone 7

When a user logs into our application for the first time, we make a Graph API call to pull the user's feed from the past year. This API call is shown below (Figure 5). We do this in order to automate the generation of Journeys, and we do this *only once* so that users have something to share right after they start to use our site.

```
var query='me/feed?fields=id,created_time,story,message,likes.limit(0)
        .summary(true),place,full_picture&since=';
query+= FacebookFactory.getLastYear();
query+='&limit=1000';
FB.api(query, function(response) {
        //application logic
});
```
Figure 5: Facebook graph query to pull a user's feed

Once we obtain the user's feed, we transform it into an array of Journeys and store that array into our database for future use. This array is populated on the browser as shown below (Figure 6).

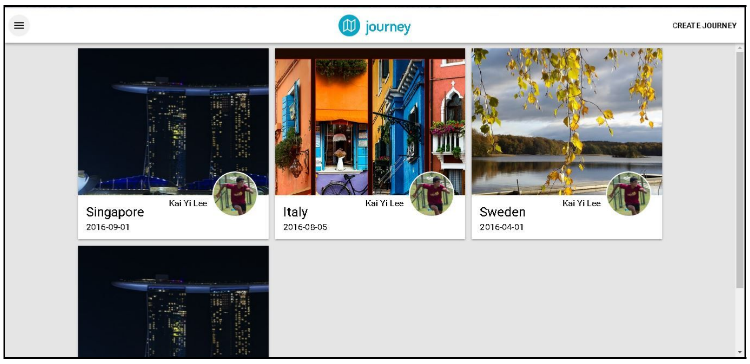


Figure 6: A user's own Journeys

Having your own Journeys is nice, but one would expect the average user to care more about their friends' Journeys! We use the following graph API query (Figure 7) to obtain the basic details of all of a user's friends. These details (in particular, their Facebook id) allows users to access their friends' Journeys via the Friends page.

```
var query ='me/friends?fields=id,name,picture.type(large)';
FB.api(query, function(response) {
});
```

Figure 7: Facebook graph query to retrieve all friends

Milestone 8

Feeds are implemented in our application to allow users to share the journeys with others. We expect that users would not only be interested in looking at their own journeys, as they would also communicate their experiences to their friends. Thus, in the context of our application, utilising the Facebook news feed will surely enhance the user experience.

Our personal experiences convinced us that users would find applications which share directly to Facebook (without any prompting) to be a nuisance. Hence, we implemented feeds using a share dialog to ensure that users are able to customize their feeds before publishing it on Facebook, as seen in Figure 8.

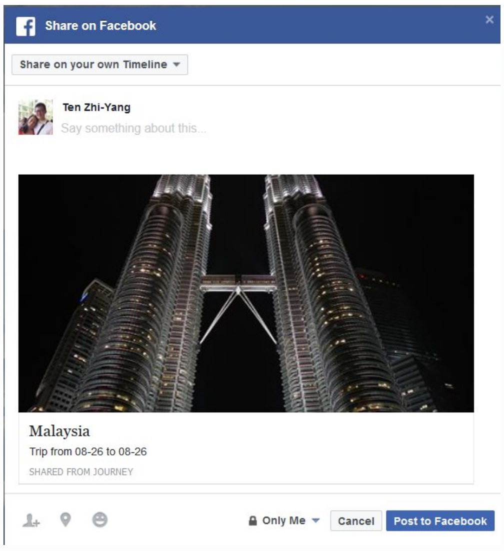


Figure 8: Sharing a Journey on Facebook

Milestone 9

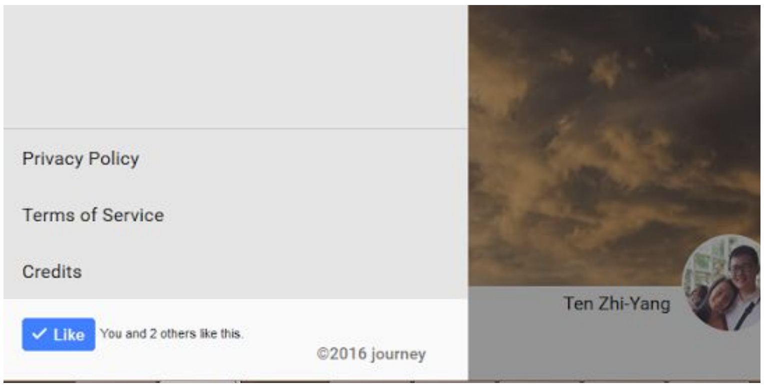


Figure 9: Like button in the navigational sidebar

With regards to the ‘like’ button, we decided that we did not want it to clutter our user interface. At the same time, we also wanted it to be prominent enough from the user’s point of view.

With these two considerations in mind, we decided to place it at the bottom of our navigation drawer, as shown in Figure 9. The like button is generated once the user logs into our application. The user only sees it when he/she tries to navigate to other pages through the navbar, thus the button will not ruin the user experience by standing out too prominently.

Milestone 10

We store data of a user’s posts when he first registers or logs in to our application, as aligned with the terms of Facebook[2]. The terms give our application permission to make use of a user’s IP content to generate his/her journeys.

When a user removes our application, in line with acceptable Facebook data use policies[3], we will delete all IP content and personal information from the user that we use in our application.

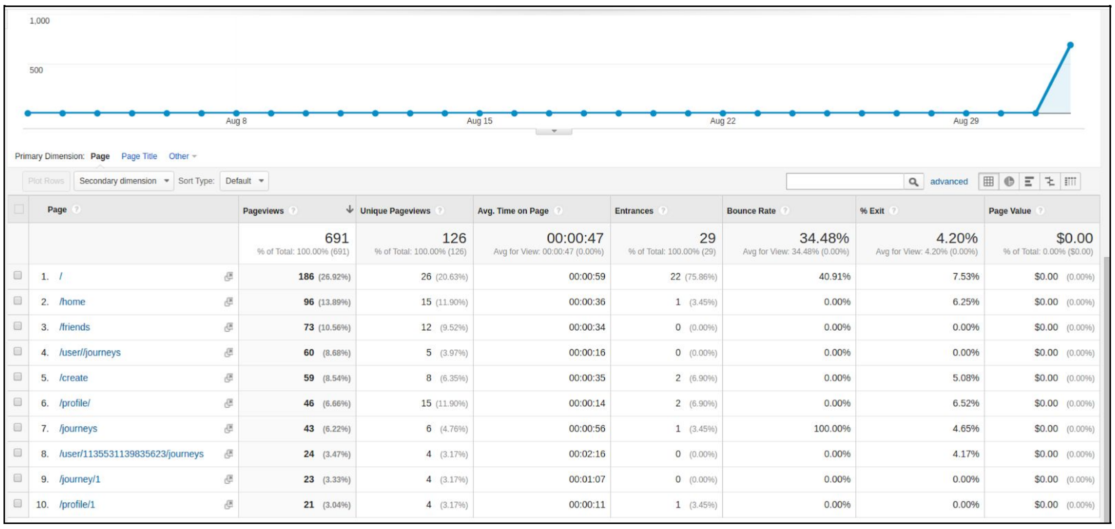
The above behavior adheres to the Facebook’s data use policies, as well as the terms & conditions which it sets out for Facebook users. Users will expect similar behavior for applications leveraging on the Facebook platform, thus handling user’s data in this manner is the respectful and appropriate.

We were unable to implement the removal of user data through the deauthorise callback method, and hence have worked around this by removing their data the next time they enter the landing page after revoking data.

Milestone 11

Initially, we were worried that Google Analytics (GA) would not be able to track multiple views within a Single Page Application (SPA) and thought it could only handle websites with multiple pages. Fortunately, we figured out how to integrate GA with Angular's ui-router, such that every time a Angular *state* change occurred, GA would update its own tracker and note the view that the SPA had switched to.

In Figure 10, you may observe the number of visits to each view, as well as other details, such as the average amount of time spent on a view. Functions provided by GA in the other tabs (not seen below) include graphical representations of user demographics (IE vs Chrome vs Firefox, Returning Users vs New Visitors, etc).



	Page	Pageviews	Unique Pageviews	Avg. Time on Page	Entrances	Bounce Rate	% Exit	Page Value
		691 % of Total: 100.00% (691)	126 % of Total: 100.00% (126)	00:00:47 Avg for View: 00:00:47 (0.00%)	29 % of Total: 100.00% (29)	34.48% Avg for View: 34.48% (0.00%)	4.20% Avg for View: 4.20% (0.00%)	$0.00 % of Total: 0.00% ($0.00)
1.	/	186 (26.92%)	26 (20.63%)	00:00:59	22 (75.86%)	40.91%	7.53%	$0.00 (0.00%)
2.	/home	96 (13.89%)	15 (11.90%)	00:00:36	1 (3.45%)	0.00%	6.25%	$0.00 (0.00%)
3.	/friends	73 (10.56%)	12 (9.52%)	00:00:34	0 (0.00%)	0.00%	0.00%	$0.00 (0.00%)
4.	/user//journeys	60 (8.68%)	5 (3.97%)	00:00:16	0 (0.00%)	0.00%	0.00%	$0.00 (0.00%)
5.	/create	59 (8.54%)	8 (6.35%)	00:00:35	2 (6.90%)	0.00%	5.08%	$0.00 (0.00%)
6.	/profile/	46 (6.66%)	15 (11.90%)	00:00:14	2 (6.90%)	0.00%	6.52%	$0.00 (0.00%)
7.	/journeys	43 (6.22%)	6 (4.76%)	00:00:56	1 (3.45%)	100.00%	4.65%	$0.00 (0.00%)
8.	/user/1135531139835623/journeys	24 (3.47%)	4 (3.17%)	00:02:16	0 (0.00%)	0.00%	4.17%	$0.00 (0.00%)
9.	/journey/1	23 (3.33%)	4 (3.17%)	00:01:07	0 (0.00%)	0.00%	0.00%	$0.00 (0.00%)
10.	/profile/1	21 (3.04%)	4 (3.17%)	00:00:11	1 (3.45%)	0.00%	0.00%	$0.00 (0.00%)

Figure 10: Google Analytics displaying information on our application

<u>Milestone 12</u>

Links to all gifs that we have recorded for this milestone are in the /gif folder of our root directory.

#1 Finding friends through the search bar on the friends page <journey-interaction1.jpg>.
- When he lands on the friend page, the user is able to see at one glance all the friends on Journeys
- To make it easier for a user to find the friend that he is searching for, the search bar narrows down the friends of the user on the screen leaving only the people whose name matches the string, as can be seen in our gif.
- We decided on implementing the friends page in this manner as it ensures that a user's friends are displayed cleanly to him following our application design guidelines

#2 A user wants to navigate through the application from one screen to another <journey-interaction2.jpg>
- A user wants to find a specific friend's journeys. To do this, he accesses the "friend" tab on the side navbar.
- The side navbar slides out smoothly, and populate all the different views that a user would want to access, including application settings and other misc functions like terms, privacy, and credits.
- We eventually decided on this method for the navigation drawer as opposed to a navigation bar on the top of our application, because it makes the application look

less cluttered, and more user friendly, likely encouraging users to focus more on the journeys that they are looking at

#3 Customizing your own journey <journey-interaction3.jpg>

- All of a user's Journeys will be automatically generated for the user when he first logs into our application, and this behavior is customized to be convenient for users.
- On the create journey page, a user is able to view all the journeys that he can create, and filter it by countries. This is designed to make the experience of the end user of creating new journeys to be as painless as possible

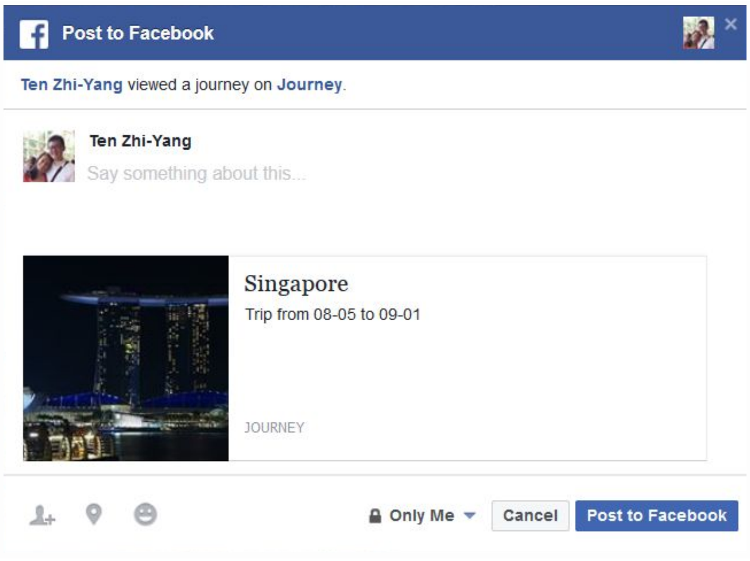
Milestone 12

Our application is mainly built to allow users to view their friend's journeys. As such, we implemented stories to allow users to tell other people on Facebook that they have viewed their friend's journeys, so that other users may be intrigued to try out looking at their other friend's journeys. This dialog will be available when users click the share button on other application user's journeys, where they can tell others that they have viewed said journey.

In this case, we have used the following format to post our story, to help the users of our application publish the above mentioned story on Facebook:

<User> viewed a journey on Journey

Users will also be able to see rich open graph marked up Facebook link posts. These link users straight to the shared journey. An example of the open graph sharing dialog is shown in Figure 11.


Ten Zhi-Yang viewed a journey on Journey.

Ten Zhi-Yang

Say something about this...

Singapore

Trip from 08-05 to 09-01

JOURNEY

Figure 11: Publishing the story of "viewing a journey"

Milestone 14

1. Cross Site Request Forgery (CSRF)

CSRF is caused when a web server does not verify that a request is coming from an authorized user, but only verifies that a request comes from the *browser* of an authorized user[4]. Hence, other sites can run requests like POST requests on a web server that maliciously modifies user information.

The best way to stop CSRF attacks is to make use of pseudo-random CSRF tokens which the server will send to the client. The client will store this CSRF token as a cookie, and also send it in every form that it submits to do a POST request to the server.

- If the client sends the form while on the same origin (correct behavior), the form will be sent with the correct CSRF token which will match the session cookie token stored on the client (which will be verified in the server), and the request will go through
- However, if the client sends the form while in a different origin (etc: on a different web page, which is an incorrect behavior which might suggest CSRF), the form will be sent with a different token, which will not match the session cookie token stored in the client. When the server receives such a request with 2 tokens that do not match, it is a sign that CSRF attacks might be occurring, and it will reject the request.

CSRF tokens are the best way to stop CSRF attacks as they are not only easy and fast to implement, but they are also secure as it is hard for hackers to guess the pseudo-random tokens that we check for on the server side between submitted forms and browser cookies. As such, for the purposes of our application, this is the best technique to stop CSRF.

We have implemented CSRF protection using the csurf[5] library in nodejs, which gives us an easy way to append our CSRF token request to the server that could mutate the state of user's data, and automatically help us to check that the token agrees with the current user's cookies stored in the browser.

2. SQL Injections (SQLi)

SQLi involves attacks to the server by hackers who attempt to execute malicious SQL statements to the server. Hackers could potentially use inputs within web applications that are included in SQL statements to maliciously modify database records of the web application[6].

To protect against SQLi, we are using the Sequelize library to make our database queries, so we do not explicitly make SQL queries in our backend, but instead use object relational mapping (ORM). However, Sequelize had faced problems with SQLi vulnerabilities in the past, but we are using a version of Sequelize (3.24.1) that has been protected from these vulnerabilities [7], so the queries we use are safe from this.

We also make use positive input validation to only allow specific characters into our SQL queries, further restricting the chance of SQLi. Once again, we are fortunate that Sequelize takes this into account -- using specific type definitions (e.g. Sequelize.STRING) and input validations (e.g. notContains: "<"), we can customise our own restrictions to persisted data. Sequelize makes it a point to fire *rejection errors* whenever inserted values do not meet our defined requirements.

We address SQLi vulnerabilities by both sanitizing strings and reducing reliance on direct SQL queries with ORM.

3. Cross-site Scripting (XSS)

XSS occurs when hackers are able to execute malicious code in real web applications. This occurs when hacker is able to inject scripts that are run by the browser, due to lack of proper validation or encoding before including user inputs in it's html page. A hacker could do this through several means, which can lead to unwanted consequences like gaining access to the session cookie[9].

We mainly take user inputs from 2 sources in Journey, and the two ways by which we take user input are secure against XSS. The first way we take user input is the which data we parse from Facebook in the form of a user's news feed, and this method is secure from XSS as even if a user keys in script tags as his post message, Facebook apis would already have their inbuilt mechanism to sanitize their inputs from XSS before saving data into their database. Furthemore, this data is served in our code through Angular's ng-bind directive and double braces {{}}, which does not run html scripts by default, and only serves data.

The second way we take in user input is in our search bars. However, these search bars are automatically configured in our application to sanitize user input, hence are safe from XSS vectors.

Application URL

Thank you for taking the time to read our write-up. Our application, which we are proud of, can be accessed via:

http://journey.ddns.net/

Made with love, by Michelle, Yao Hui, Ten, and Kai Yi.

Citations

[1] Canvas Hosting by Facebook, online: Facebook for Developers Canvas Hosting by Facebook <https://developers.facebook.com/docs/games/services/contenthosting>.

[2] Statement of Rights and Responsibilities, 30 January 2015, 1 UNTS 1, online: Facebook Statement of Rights and Responsibilities <https://www.facebook.com/terms>.

[3] Full Data Use Policy, 30 January 2015, IV , online: Facebook Data Policy <https://www.facebook.com/full_data_use_policy>.

[4] Preventing CSRF and XSRF Attacks, 14 October 2008, online: Coding Horror <https://blog.codinghorror.com/preventing-csrf-and-xsrf-attacks/>.

[5] CSURF token middleware, 28 May 2016, online: CSURF token middleware <https://github.com/expressjs/csurf>.

[6] SQL Injection, 2016, online: Acunetix SQL Injection <http://www.acunetix.com/websitesecurity/sql-injection/>.

[7] SQL Injection, 19 January, 2015, online: Node Security Platform SQL Injection <https://nodesecurity.io/advisories/sequelize-sql-injection-order>

[8] String validation and sanitization, online: validator.js <https://github.com/chriso/validator.js>.

[9] Cross site scripting, 2016, online: Acunetix Cross site scripting <http://www.acunetix.com/websitesecurity/cross-site-scripting/>.

[10] UNWTO Anuual Report 2015, 2016, online: UNWTO <http://cf.cdn.unwto.org/sites/all/files/pdf/annual_report_2015_lr.pdf>.

[11] Singaporeans made more than 8 million overseas trips last year, 16 August, 2014, online: Today Online <http://www.todayonline.com/singapore/singaporeans-made-more-8-million-overseas-trips-last-year-natas>.

Credits

Flaticon, online: Flaticon <http://flaticon.com>.