Aspirations:

1. Choose Facebook Canvas or Standalone and write-up:

We chose to make Scribl a standalone app due to the greater flexibility it provides. First, despite the fact that our app's initial purpose is to to allow users to share drawings and to create interaction between Facebook users, this app needs not be limited to Facebook. On one hand, we have the option of allowing users to save their drawings locally, so users can actually use the app as a personal quick-drawing tool. From another perspective, if we wish to develop this product further, allowing sharing on other platforms is one way to grow. Besides, we also felt that Canvas applications were more geared towards Facebook games that required extensive permissions and deep usage of the Facebook API.

6. Write out some SQL queries

We use Eloquent ORM provided by Laravel PHP Framework as an ActiveRecord implementation to build our Models and interact with the database through them.

Some queries are:

Eloquent - PHP	Raw SQL
`Post::where('tag','=',$tag)->get();`	select * from "posts" where "tag" = ?
`User::where([ 'username' => $userData->nickname, 'email' => $userData->email, 'avatar' => $userData->avatar, 'fbid' => $userData->id ])`	select * from "users" where ("username" = ? and "email" = ? and "avatar" = ? and "fbid" = ?)
`Post::where([ 'user_id' => $userId, 'fb_id' => null, 'url' => $url, 'tag' => $tag, 'png_url' => $savedFile['pngURL'], 'gif_url' => $savedFile['gifURL'], 'json_url' => $savedFile['jsonURL'] ])`	select * from "posts" where ("user_id" = ? and "fb_id" = ? and "url" = ? and "tag" = ? and "png_url" = ? and "gif_url" = ? and "json_url" = ?)

7. Graph API Queries:

1) Retrieve friends' list for people to share their drawings with; check whether someone coming back to one of drawings through link is friend of the original owner

2) GET request: Query for permissions from /me/permissions to check if user has approved specific Facebook permission to ask for authorization.

3) POST request: Post to /me/photos to upload new photo to users' photos.

8. Feeds (Feed Dialog)

We use a custom feed dialog to publish feed stories via the Graph API. This feed dialog allows the user to choose to share their drawing as an image, or as a GIF. It will only pop up if the user clicks the "Share" button at the bottom right of the drawing area and then choose to post. We also did not implement any automatic posting, so we can be sure that when something from our app appears on Facebook, it was done by the user and not a nuisance feed.

9. Like button:

We placed our Like button at bottom left of the page. This may not seem like a very prominent spot, but we believe that users will subconsciously notice it as they will be focusing a lot on the toolbar on the left edge of the page. We chose to make the Like button more subtle because our app's focus is more on the sharing function. Although the Like button is a good way to promote apps, we feel that our app's use of content sharing will allow it to inherently promote itself as people use it and share their creations. Moreover, forcefully emphasising the Like button may actually interrupt our users' workflow. We believe that our current arrangement will be lead to a better overall user experience.

10. Data handling after removing the app

After a user removes our app, we continue storing their name and drawings on our server for future use, in the case maybe someone wants to access an old drawing. The links between users, their friends and a their drawings can also allow us to prevent unauthorised users from accessing a user's drawing. These are in line with the

functionality of our app. Also, although we keep the data they have given us, we do not continue collecting data.

Facebook has not posed any specific requirement on old data retrieved before the user de-authorizes app. Of course it has consistently urged every app to retrieve information only to improve users' experience and protection of data from misuse, but we believe the data that we keep are very minimal and not misused.

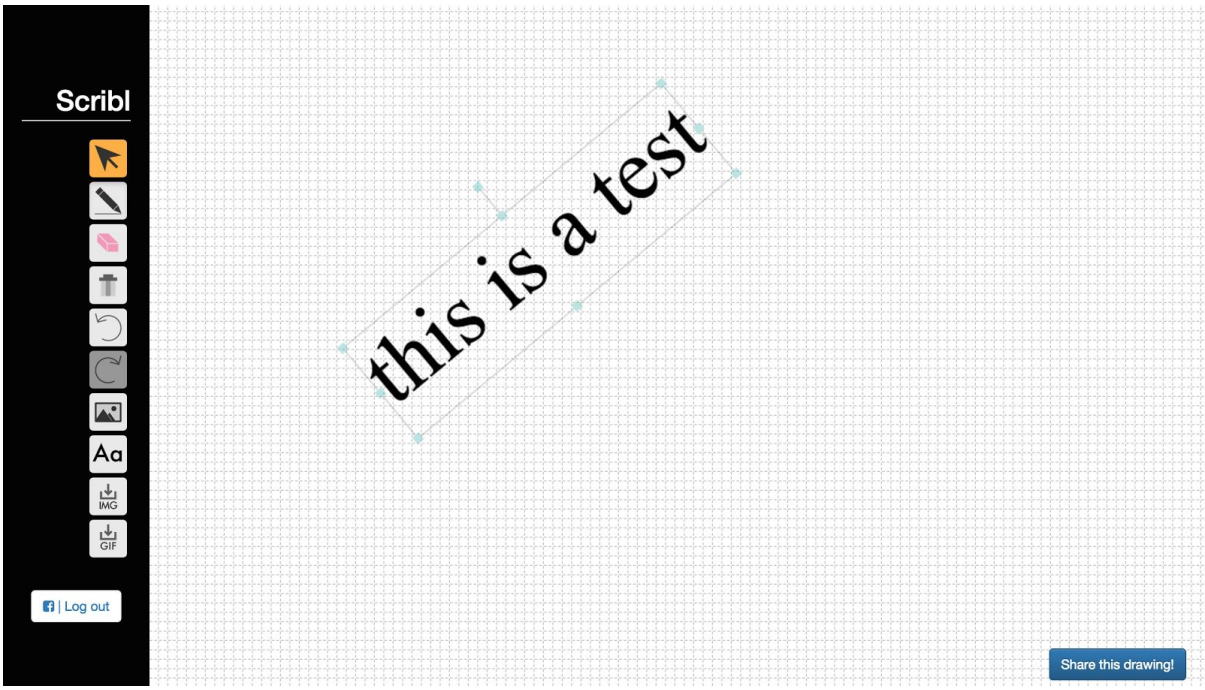
12. User experience

Disabled backspace

When manipulating images or text, it's common for users to instinctively press the delete key to remove a selected item from the canvas. We attempted to allow this keyboard shortcut, and added a listener to do so. However, we found out that on some laptops, the delete key is also the backspace key, which is a shortcut for the back button. This leads the user to the last visited web page, causing unsaved data to be lost and causing the him a ton of frustration afterwards. We resolved this issue by disabling the default functionality of the backspace key in situations when users are likely to hit the delete key, allowing for users to safely use the key to delete selected items in the canvas.

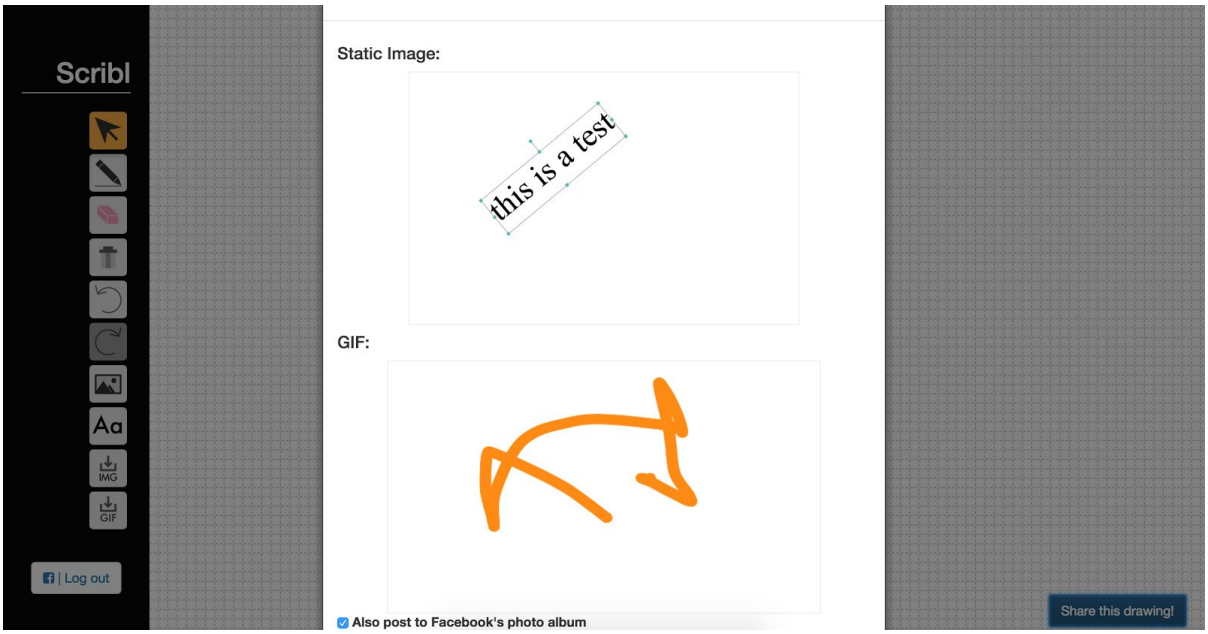
An alternative we considered was to use an autosave function to prevent data from being lost when the user leaves the page by accident. While this also resolves the problem, we felt that the implementation would have been excessively complicated, and would use up a lot of resources, be it locally or on our server.

Selection tool



Our app has 3 main modes: Draw, Erase, and Select. The Draw tool is a mode in which users can draw over anything in the canvas. For this reason, users are not allowed to select any objects in this mode. The same goes for the Erase mode. The Selection tool is hence necessary for allowing users to interact with items already placed on the canvas. After a user creates an image or text via the "Insert image" and "Insert text" buttons, they will be automatically be switched to the Select mode, allowing them to modify their new item as they wish. When they switch to another mode, the app will disable selection, but users can always select the Selection tool again to make changes to items they have previously "committed" to the canvas. We also added in keyboard shortcuts (listed in the button tooltip) for quick switches between modes.

GIF recording



By recording the user's interactions with the app and putting together snapshots of the canvas while the user is using it, we are able to generate GIFs of the creation process. The first time users create a drawing, they will most likely be unaware of this function. However, once they see the GIF in the preview segment of our custom share dialog, they can intuitively understand how the GIF is created by seeing their own process recorded, and can have this in mind when using the app in future. This allows us to pleasantly surprise the users, while also generating interest in using our app again, with a greater focus on creating a cool GIF.

13. DOM Manipulation using jQuery

Our undo and redo buttons are disabled when not applicable by using jQuery to add and remove a class:

```
if (histIndex == 0){
        $("#undo").prop('disabled', true);
} else{
        $("#undo").prop('disabled', false);
}
```

14. Open Graph

We make use of Share Dialog to facilitate user posting to their timeline. The customizable contents for posting include image, gif which records the drawing process and comments. User can choose to tag their friends when they post. Only through this sharing process will the experience we seek to provide to our users be complete. We are encouraging people to draw for sharing and communicating experience and this sharing activity is key to Scribl.

15. jQuery Animation (opt)

On the Share modal, after clicking on the post button, the browser will automatically scroll to the top so that the user can see the success/failure alerts. This is achieved by using jQuery Animation `$("#beforePostModal").animate({ scrollTop: 0 }, "slow");`

16. AJAX (opt)

We implemented AJAX for the form inside share modal. When clicking one of the post buttons, the client will make an AJAX call to our server and post the POST request. On completion, we will handle the response (success or failure) using the callback function. AJAX is helpful here because we want to continue staying at the page after posting instead of navigating away to do the POST request.

17. jQuery Plugins

Bootstrap is an important plugin we have utilized. Use of Bootstrap has kept our style clean and consistent, and made implementing some features much easier. In particular, the tooltips for our buttons use Bootstrap popover, which allows us to create useful descriptions of our buttons over the other elements on the page without needing to handle the HTML.

We also used colpick to implement colour selection for the Draw tool. This plugin handles the creation of the colour selection window, and allows for a visual interface for users while enabling us to obtain the selected colour easily.