Milestones

Describe the problem that your application solves.

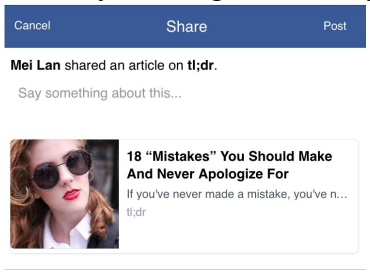
It is a hassle when one having to read through (or the very least, scan through. or even tl;dr) the entire article just to read a particular section that he is interested in. Furthermore, there are so many title-baits these days, so more often than not, we click in and to only realise that article is nothing more than just an attention seeking title. This results in a lot of time wasted reading inefficiently. The problem with Flipboard is that it only provides the first line of the article and when clicked, it will launch another browser tab to bring reader to the original article which is a hassle to switch between tabs.

1. **Describe the application and why it makes more sense on mobile and cloud platform?**

   TL:DR is a mobile application that functions like a content page for articles. It condense articles into their key subheadings and if the reader wants to read more about the particular point, the subheading can be expanded to reveal the content under it. The motivation for this app is to provide more context to readers about the article through showing the subheadings, so that they can decide better if they want to spend time scrolling the entire thing or just read one portion of the full article that they are interested in. People read within our app, and only if they want the original article, they can click on the title to launch the original article in the browser. Our 4 key categories of articles are Money, Tech, Lifestyle, Self-help. It makes sense for TL;DR to be on the mobile platform because people nowadays spend a lot of time reading on their mobile phone, especially when they are on the go. The themes of articles are also suitable for casual reading on the phone, and categorization into subheadings. As a reading app, the UI can be designed to be user-friendly for mobile readers, making it a convenient mobile reading app. They will be able to bookmark the articles if they want to read later.

2. **Who's the target users? How to attract them?**

   Target at people who do casual reading of articles on the go. People who are on the go tend to read on their mobile, and they do not have time and a suitable environment to read everything in detail and with full attention, so what we can provide is the gist of the things. To gain user-base, one method would be to actively share popular articles through Facebook as it will show that the article is shared through our app while the link is still the original article's link. This provides free publicity for our app whenever an article is shared. For a start, we might to growth hack by sharing articles by ourselves to make people notice about our app.



Mei Lan shared an article on **tl;dr**.

Say something about this...

18 "Mistakes" You Should Make And Never Apologize For

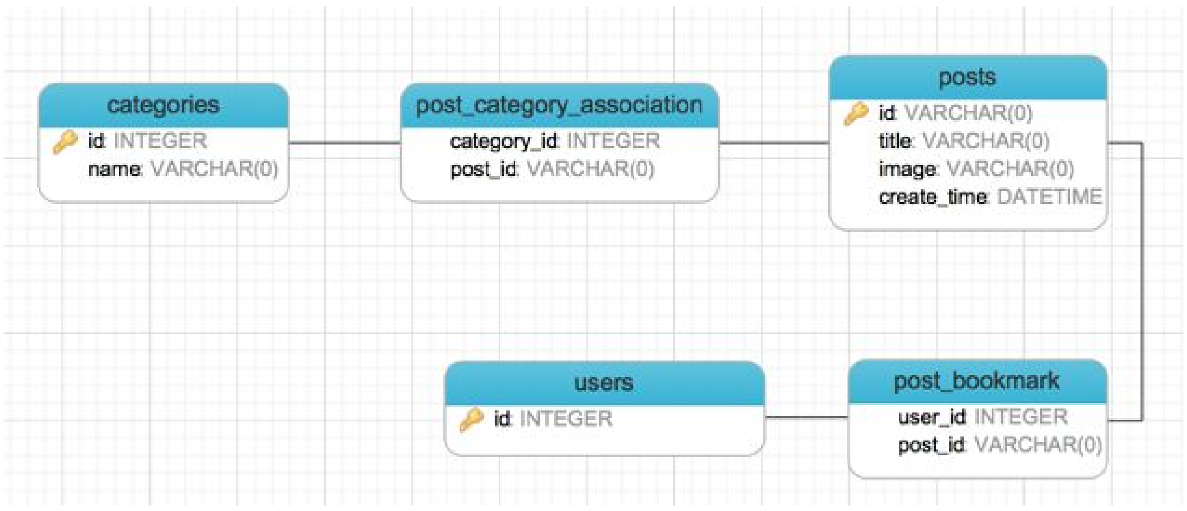
If you've never made a mistake, you've n...

tl;dr

3. **Name the application. (Not graded)**

   TL;DR

4. **Draw database schema.**



5. **Design and document at least 3 RESTful APIs. How do they conform to REST principles?**

1) Get detailed information of one article
URL: `/api/v1/article/:source_id/:article_id`
Method: GET
Returns: {
```
                article_id: String,
                bullets: [{
                     details: String,
                     title: String
                }],
                category: String,
                headlines: Array<String>,
                image: String,
                source: String,
                time: String,
                title: String,
                url: String
          }
```
Throws: HTTP 404 if article not found.

2) Get a page of the article list of a category.
URL: `/api/v1/feed/:category_id?page=:page_number`
Method: GET
Returns: `Array<Object>`
Refer to the Article resource for definition of Article object.
Throws: HTTP 404 if category not found.

3) Users' bookmarks

3.1) Get users' bookmarks within one category

URL: `/api/v1/bookmark?category=:category_id`

Method: GET

Returns:
```
{
        data: Array<Object>,
        total: Integer
    }
```
Refer to the Article resource for definition of Article object.

Throws: HTTP 403 if not logged in or request not sent from our app.

3.2) Create new bookmark for a given article

URL: `/api/v1/bookmark`

Method: POST

Payload:
```
{
        source_id: String,
        article_id: String
    }
```
Returns (echo):
```
{
        source_id: String,
        article_id: String
    }
```
Throws: HTTP 403 if not logged in or request not sent from our app; HTTP 404 if article not found; HTTP 409 if the bookmark already exists.

3.3) Delete a bookmark for a given article

URL: `/api/v1/bookmark`

Method: DELETE

Payload:
```
{
        source_id: String,
        article_id: String
    }
```
Returns (echo):
```
{
        source_id: String,
        article_id: String
    }
```
Throws: HTTP 403 if not logged in or request not sent from our app; HTTP 404 if bookmark not found.

6. **Provide at least 3 interesting SQL queries.**

1) Query trending posts (posts within several days, ordered by number of bookmarks)

```
SELECT posts.id, COUNT(post_bookmark.user_id) AS bookmarks
FROM posts, post_bookmark
WHERE posts.id = post_bookmark.post_id AND posts.create_time > 0
GROUP BY post_bookmark.post_id
ORDER BY bookmarks DESC
```

2) Query user's bookmarks within a particular category

```
SELECT posts.id FROM posts, post_category_association
WHERE post_category_association.post_id = posts.id AND
      post_category_association.category_id = ? AND
      EXISTS (SELECT * FROM post_bookmark
              WHERE post_id = posts.id AND user_id = ?)
```

3) Query articles within a particular category

```
SELECT posts.id, posts.title, posts.image
FROM post_category_association
JOIN posts ON post_category_association.post_id = posts.id
WHERE post_category_association.category_id = ?
ORDER BY posts.create_time DESC
```

7. **What does [QSA, L] mean? What are the most interesting mod_rewrite rules used?**

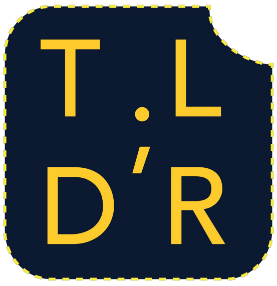
QSA means Query String Append. With this flag, if the target rewrite rule is invoked, the query string(the part after "?") is appended to the URI string generated by the rewrite rule, instead of being discarded.

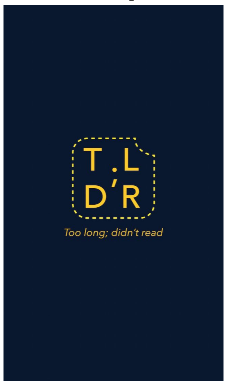
L stands for Last. With this flag, if the target rewrite rule is triggered, subsequent rewrite rules will not be processed, even if the regular expression strings match the URI string.

As Python's Flask framework provides clean URL functionality, we do not need to manually do our own mod_rewrite rules. The following are a few rules if we were to implement our app on a LAMP stack without using any framework.

Mod_rewrite rules:
RewriteBase /
RewriteRule ^/api/v1/feed/([0-9]+)   api.php?action=getfeed&category=$1   [QSA, L]
RewriteRule ^/api/v1/article/([a-zA-Z0-9]+)/([0-9]+)$
api.php?action=getarticle&source=$1&article=$2 [L]

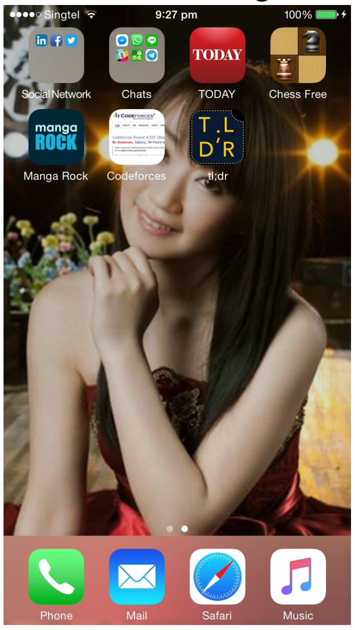
8. **Create attractive icon and splash screen (when users launch from home screen).**

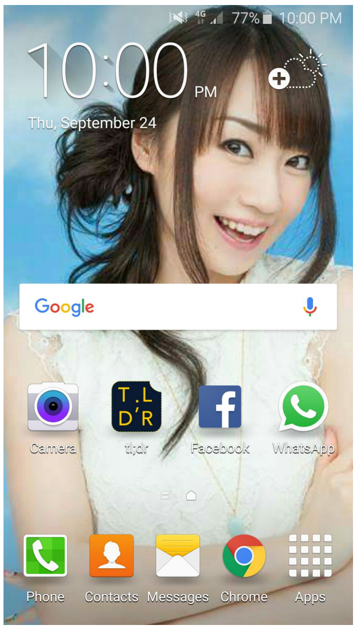




Currently, the splash screen works only on some iOS devices (All iPhones and some iPads) running iOS 8 (iOS 9 seems to have a bug that breaks web app's splash screens) as we have yet to figure out the dimensions and also how to add splash screens for Android devices. Also, there are many various sizes that we need to cater for for Android devices.

9. **Make sure adding to home screen works on iOS Safari and Android Chrome.**





10. **Style UI components using a structured CSS. Why is it the best UI design?**

Most of the styling is taken care of by MaterializeCSS and Angular Material frameworks by using their UI components as much as possible. Additional CSS is used when we need to do minor design tweaks to the UI components, such as adjusting the position and changing the color of the buttons.

The structuring of the CSS files is as such: each of our own custom UI component and angular view partials have their own CSS files that applies to themselves. For the UI components, as it is possible that several instances of the component are used in the same bigger template, embedding the CSS file in the component template may lead to duplicates in the overall HTML page, causing it to be unnecessarily bulky. Partials on the other hand, are guaranteed to have only 1 instance per page due to Angular's ng-view. Thus, to keep things lightweight, CSS files of UI components are included in the main HTML while that of partials are included in the partials themselves.

Our CSS is typically written in LESS files, which is then compiled and minified into CSS files for use in HTML.

11. **Implement and describe offline functionalities. Why do they fit users' expectations?**

    When offline, users are able to view the articles that were last loaded by the app when the app was online. This makes sense because when offline, the app is not able to get newer articles. The next best option is to give users the latest cached articles so that they have something to read.

    If the app goes offline while the user is still using the app, the user can still bookmark/unbookmark articles. The app assumes the requests will be successful(they will be almost all the time) and updates the UI accordingly to let the user know that his/her actions have been received. Synchronisation will be done as described in the next section.

    If the user launches the app offline, the app considers the user as logged out. This is because the app cannot be sure whether the user is logged in to Facebook. This prevents the user from bookmarking/unbookmarking so as to preserve data integrity as bookmark is a relation between a valid logged in user and an article.

12. **How does the client synchronise with the server? What are the possible cases and how each case is handled?**

    The synchronisation scenarios our app needs to cover are: bookmarking/unbookmarking articles while offline, and updating the articles/bookmarks on the current page when the app transits from offline to online.

    When the app first loads, requests to fetch the first page of articles of every category are dispatched in the background using a job queue.When online, the articles fetched, being more recent, are then saved to the web storage. When offline, the articles are being served from the web storage while the requests to update the web storage articles are being queued in the job queue. The job queue continues dispatching the requests when the app goes back online, thus keeping the articles stored locally as up-to-date as possible.

    For bookmarking/unbookmarking of articles, these requests are also sent to the job queue for dispatching. The job queue waits for the app to go online before dispatching these requests to keep the server synchronised in terms of bookmarks.

    Also, when the app returns online, the current page's controller fetches the articles of the currently viewed category and automatically updates the page with the newly-fetched data, giving the user the up-to-date articles at the first opportunity.

13. **Advantages/Disadvantages of basic authentication against digest access authentication/cookies/OAuth. Why is the chosen method the best for the application?**

Basic Authentication is easier to implement and it requires no extra authentication request before the actual API request.
However, it is less safe since basic authentication can be decoded directly if the request is captured by an attacker.

Since our app uses Facebook Javascript SDK to implement FB login feature, we directly use its cookies for authentication. By doing so, no extra work needs to be done in front-end to provide authentication support. Similar to basic authentication method, no extra login requests are required between the front-end and the back-end.

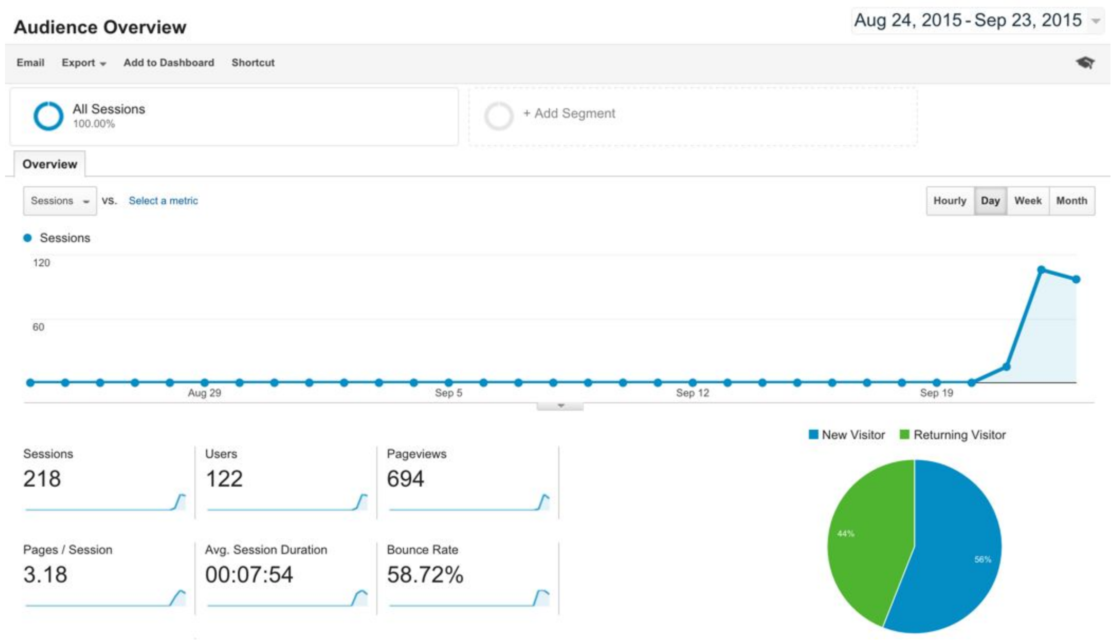
14. Describe 3 workflows. Why are they better than alternatives in enhancing UX?

The use of accordion to display the main points of each article is chosen because when browsing through the list of articles, it can take many scrolls to get to the next article if we do not hide the bulk of the content when viewed on a small mobile screen. This design allows the user to expand the points that he/she is interested in in order to find out more. This is the main essence of the app.

The bookmark and Facebook share button are placed at the bottom of each article. This is so that as the user finishes skimming through the article, he/she may want to save the article for reading at a later time or find it worthy to share with friends. At this point, he/she is at the bottom of the article. Putting the buttons at the top of the article makes it troublesome to have to go back up to bookmark/share.

The bookmarks page provides a personalized point of access for managing the user's bookmarked articles. Over time, the articles that the user likes or wants to read in the future gets pushed down the list such that the user has to scroll down several pages to find back the same article. With the bookmark page, which is accessible from the side menu, the user can focus on only articles that he/she want to read.

15. Embed Google Analytics and screenshot analytics report.



Sessions	Users	Pageviews
218	122	694

Pages / Session	Avg. Session Duration	Bounce Rate
3.18	00:07:54	58.72%

16. **Integrate with social network. Explain choice of social network. (Optional)**

Our app integrates with Facebook as social network. We chose Facebook because it is one of the social media with a large user base. Secondly, many users tend to be lazy to bother with creating and remembering new accounts, so allowing users to log in via their social media account makes things convenient, encouraging them to log in to find out what they can do when they are logged in. Thirdly, we initially wanted to be able to integrate with Twitter as well. However, we felt that we might not make it in time if we take time to learn Twitter's API and try to implement its integration. We are currently more familiar with Facebook's API as we have done it in assignment 1.

Another reason we decided to integrate with social media like Facebook is for the 'Share' social plugin. As a reading app, it would be good for the users if they can just share the article they like on their social media.

17. **Use Geolocation API to plot out position/route on Google Maps. (Optional)**

We are not using any Geolocation API as geolocation does not make sense in a reading app.

18. **Justify choice of UI design framework/library. (Optional)**

We have chosen MaterializeCSS and Angular Material as our UI design frameworks. This is because both provide commonly used UI components with material design which are functional right out of the box, saving us from unnecessary time wasted on figuring out CSS and animations. Also, both frameworks provide components which are responsive, making them suitable for HTML5 application that renders nicely on mobile browsers as well.