

Cardhub

CS3216 (AY2016/17) Assignment 3 Milestones

GROUP 8

Jiang Sheng

Si Jun Ke

Piyush Varanjani

Wang Yanhao

Application Live at <https://cardhub.tk>

Milestone 0: Describe the problem that your application solves. (*Not graded*)

Some notes and content are best curated in the format of flashcards where the front and the back of the cards are meaningfully utilised. Also, the cards should be readily accessible and easy to share. However, currently there is no such App available with great UX.

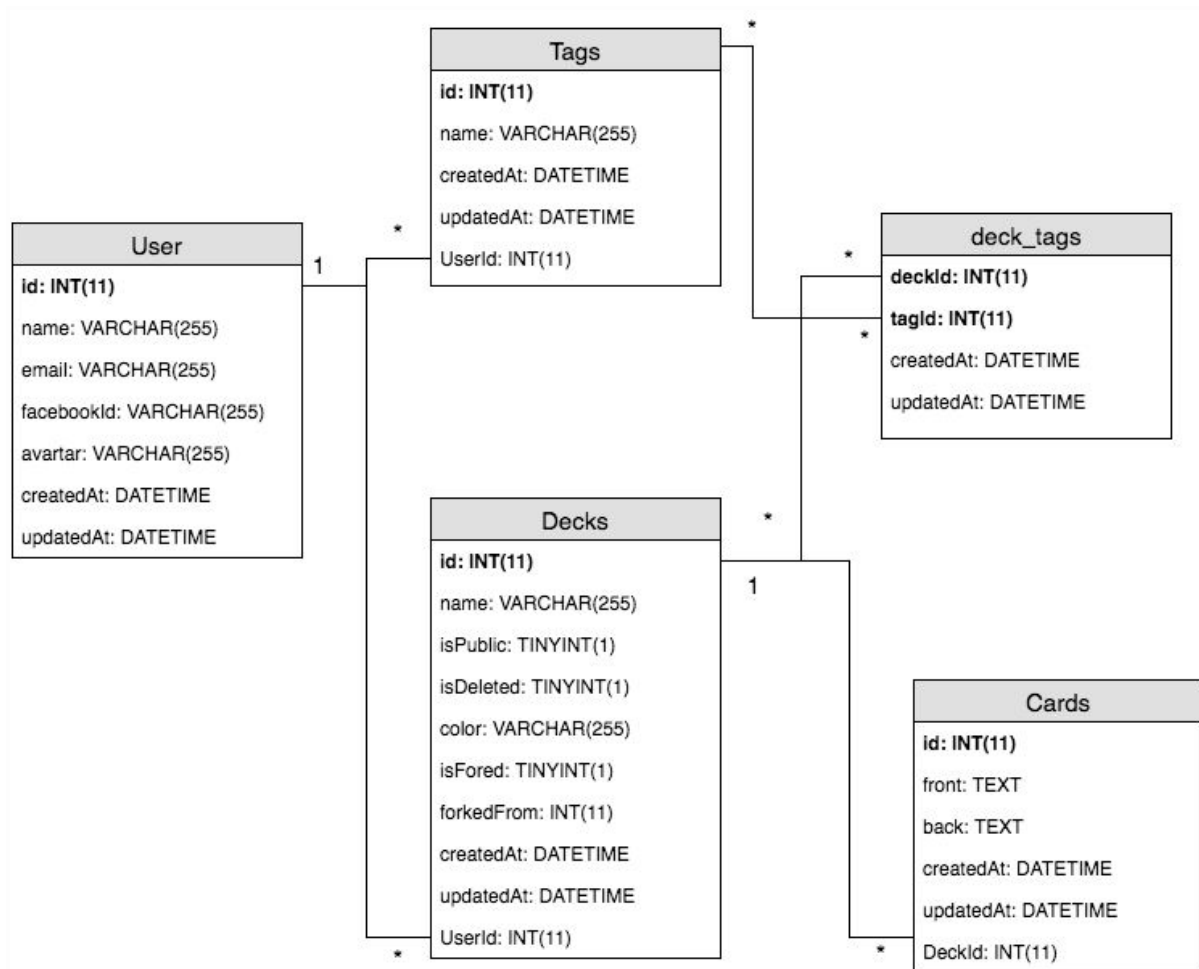
Milestone 1: Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

Cardhub allows users to create and store decks of cards, organise their decks by tags and colors, share with others their decks, and search and clone other people's decks. As a utility App, we want to make the user-created content readily accessible whenever and wherever to them as much as possible. Mobile cloud computing ensures that users can access their content across various devices since user data are stored remotely. Caching user data also allows users to read the content when or where internet connection is not available. Hence, mobile cloud computing helps us to allow our users to access their cards.

Milestone 2: Describe your target users. Explain how you plan to promote your application to attract your target users.

Our target users are people who like to use cards as the medium to take notes or to keep contents, and want to share their cards with their friends. We intend to attract our target users by interesting contents as well as the UX that resembles real double-sided cards. We allow users to share their public decks of cards on Facebook, so that their friends can see the interesting contents available and may want to use our App to access the contents.

Milestone 3: Draw the database schema of your application.



Decks and Tags are many-to-many relation with a join table deck_tags.

Milestone 4: Design and document all your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any.)

The REST API doc could be found at: <http://docs.cardhub.apiary.io/#reference> (noted the apiary console may not work)

We use a middleware to extract requestor's user information from the Authorization header, and hence achieve authentication and protection for the api.

Most of API purposes should be quite straightforward. Here are some exception to the principles:

GET /forkDeck/:id

We use this api to fork a deck to be owned by the current user. This is a special api as it does not fall into any of CRUD categories, and it has several operations underneath such as find, create and save. We think it is not suitable to apply REST design principles on it, and what it does doesn't match any HTTP request method as well. That's why we create a separate endpoint for it.

GET /search

When search decks by keyword, we felt the search API doesn't really make sense to be applied to a specific resource endpoint. Therefore we adopt a standalone path for it, and use query string to build the query and pagination.

Milestone 5: 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.

Select all cards, tags from a specific deck for the current user..

```
SELECT `Deck`.`id`
      ,`Deck`.`name`
      ,`Deck`.`isPublic`
      ,`Deck`.`color`
      ,`Deck`.`isForked`
      ,`Deck`.`forkedFrom`
      ,`Deck`.`isDeleted`
      ,`Deck`.`UserId`
      ,`Tags`.`id` AS `Tags.id`
      ,`Tags`.`name` AS `Tags.name`
      ,`Tags.deck_tags`.`createdAt` AS `Tags.deck_tags.createdAt`
      ,`Tags.deck_tags`.`updatedAt` AS `Tags.deck_tags.updatedAt`
      ,`Tags.deck_tags`.`deckId` AS `Tags.deck_tags.deckId`
```

```

        , `Tags.deck_tags`.`tagId` AS `Tags.deck_tags.tagId`
        , `Cards`.`id` AS `Cards.id`
        , `Cards`.`front` AS `Cards.front`
        , `Cards`.`back` AS `Cards.back`
        , `User`.`id` AS `User.id`
        , `User`.`name` AS `User.name`
        , `User`.`facebookId` AS `User.facebookId`
FROM `Decks` AS `Deck`
LEFT JOIN (
    `deck_tags` AS `Tags.deck_tags` INNER JOIN `Tags` AS `Tags` ON
    `Tags`.`id` = `Tags.deck_tags`.`tagId`
    ) ON `Deck`.`id` = `Tags.deck_tags`.`deckId`
LEFT JOIN `Cards` AS `Cards` ON `Deck`.`id` = `Cards`.`DeckId`
LEFT JOIN `Users` AS `User` ON `Deck`.`UserId` = `User`.`id`
WHERE `Deck`.`id` = '39';

```

Search by keyword and page.

```

SELECT `Deck`.`id`
    , `Deck`.`name`
    , `Deck`.`color`
    , `Deck`.`isForked`
    , `Deck`.`forkedFrom`
    , `Deck`.`UserId`
    , (
        SELECT COUNT(*)
        FROM Decks
        WHERE Decks.forkedFrom = Deck.id
    ) AS `ForkCount`
    , `User`.`id` AS `User.id`
    , `User`.`name` AS `User.name`
    , `User`.`facebookId` AS `User.facebookId`
FROM `Decks` AS `Deck`
LEFT JOIN `Users` AS `User` ON `Deck`.`UserId` = `User`.`id`
WHERE `Deck`.`name` LIKE '%New%'
    AND `Deck`.`isPublic` = true
    AND `Deck`.`isDeleted` = false
ORDER BY ForkCount DESC LIMIT 0
    ,10;

```

Update Tag Name

```

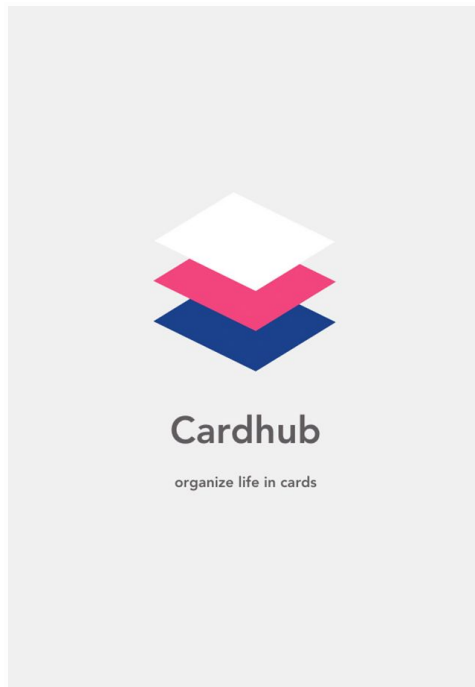
UPDATE `Tags`
SET `name` = 'Update tag name'
    , `updatedAt` = '2016-09-22 18:49:56'
WHERE `id` = '12'
    AND `UserId` = 1

```

Milestone 6: Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly. Include an image of the icon and a screenshot of the splash screen in your writeup. If you did not implement a splash screen, justify your decision with a short paragraph. Add your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.



logo



Splash screen

“Add to Home” for both iOS Safari and Android Chrome are working properly.

Milestone 7: Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies (or others if you know of them) and implement it in your application. Justify your choice of methodology.

We make use of Angular Material for the UI design. The reason is our team believe material design on mobile is elegant and well-designed. They also have a full set of icons and design guides to help us get started.

For the main UI, we use a drawer menu as a centralised place to access the different features and the drawer helps to save screen space so that more data can be displayed. The breadcrumb in the header allows the user to navigate back to the previous states. The floating action button at the right bottom corner is where the user can perform the create, update, delete actions on decks and cards. All these components help to keep the UI clean and organised.

For the design of the decks and cards, we make them resemble real decks and cards without being overly done so that the UI looks simple and clear. We also provide feedback to user's actions. For example, toast messages are shown if the user has successfully created or failed to create a deck.

We have a few colors in our App, as we provide a color palette for user to style their cards. However, the colors are mellow and don't look messy when they are put together. We use a bright pink color as our action color for consistency.

Overall, our UI is clean, organised and elegant, and suited for deck and card display.

We used OOCSS (Object Oriented CSS) in our implementation. The main reason is we divide our application UI into different component, and many components are reusable, such as cards and decks. While the structure is repetitive, they often have different style, such as different color for different deck. By adopting OOCSS, we could write reusable structure codes to improve reusability, but style them differently (such as different color). This is correspond to one of the OOCSS principle: separation of structure from skin.

In this way, our CSS codes are more manageable and highly reusable, and also it has good extendability if we want to introduce more styles or structures.

Milestone 8: Set up HTTPS for your application, and also redirect users to the `https://` version if the user tries to access your site via `http://`. Name 3 best practices for adopting HTTPS for your application. Explain the term "certificate pinning" and discuss the pros and cons of adopting it.

1. Use strong private key and certificate and obtain certificates from a Reliable CA. We used Let's Encrypt to assist us to obtain the SSL certificate.
2. Use Secure Protocols. There are five protocols in the SSL/TLS family: SSL v2, SSL v3, TLS v1.0, TLS v1.1, and TLS v1.2. We should prefer using TLS v1.1 and v1.2 as they are both without known security issues now. We can do the setup in Nginx's configuration.
3. Use Secure Cipher Suites. In SSL and TLS, cipher suites define how secure communication takes place. We should use the more secure cipher than those weak ones.

SSL Pinning is making sure the client checks the server's certificate against a known copy of that certificate. By associating the host with their certificate, we immediately know if the server we are visiting is an known authentic server. We can bundle the server's SSL certificate inside the application, and make sure any SSL request first validates that the server's certificate exactly matches the bundle's certificate.

Pros: We can guarantee that the client is indeed talking directly to the server and without a man in the middle eavesdropping or some fake servers. This enhance the security of the server-client communication.

Cons: The certificate may expire after some time. We have to implement a way to update the certificate contained in the app. Also, if we allow user to interact with some website that is not in our certificate, they will fail to do that.

Milestone 9: Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users' expectations. State if you have used service workers, Web Storage, or any other technology. Explain your choice. Make sure that you are able to run and use the a reasonable subset of features of your application from the home screen without any internet connection.

We have used Service Worker, AppCache and LocalStorage in our application. We use service worker and AppCache to precache all static resources such as styles, images and javascripts so the user could view the application offline. We store the user profile in localStorage for offline access. We use service worker to cache dynamic request such as api call and facebook avatar request. If a user has visited some decks or cards, they will be able to visit them offline as those data are cached by service worker already. When offline, a banner will be show to inform users that they are in offline mode.

Most users are expected to view cards offline. For service worker enabled browser, our users could first view some decks and cards when online, and later view them offline. User also can view their tags and filter decks by tag when offline.

Milestone 10: Implement and explain how you will keep your client synchronised with the server if your application is being used offline. Elaborate on the cases you have taken into consideration and how they will be handled.

Basically, on client side we only allow “view” operation. User can view their decks, cards, tags, etc. But they could not create, modify or delete anything. We make use of the online/offline event. If user is offline when using the application, the UI will be changed to prevent user from modifying the data offline. If there are any update happening when user is offline, once the client is back to online, we will synchronize the data by retrieving user data from server again to make sure they see the updated content.

Milestone 11: Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application.

Token-based authentication:

- Stateless: no session is persisted server-side
- Advantages: decreasing memory usage, easy scale-ability and total flexibility (tokens can be exchanged with another client); mobile application ready

Session-based authentication:

- Stateful: associates the session id with a user's account
- Advantages: can restrict or limit this session to certain operations or a certain time period and can invalidate it if there are security concern; can log users' every moves on the website(s)
- Disadvantages: bad scale-ability and extensive memory usage.

We chose to use token-based authentication in our application because we are developing a progressive web application and we don't want to stress the server too much. Moreover, we want to enable the user to share their information with friends from other sites in the future.

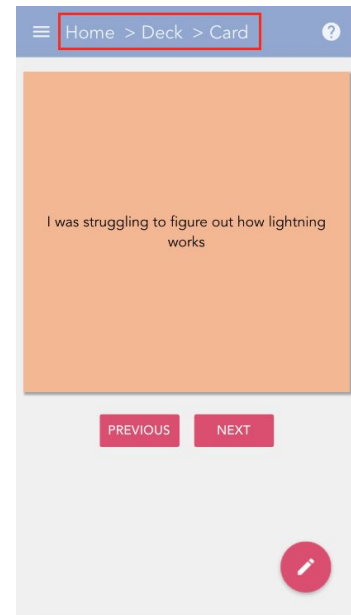
Milestone 12: Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

We use Angular Material and Angular.js because we are familiar with the framework and we thought it would be better use of time if we stick to Angular. Angular Material has good support for building a mobile-friendly webapp. Its material design makes the UI look like native-app. It also has support for mobile gesture such as swipe, which we used in card navigation. We feel that

Angular Material is not any less adequate in terms of fulfilling our needs compared to the recommended framework/library, and we are already quite familiar with it, so we decided to use it.

Principle 3: Make it easy to get back to the home page

The breadcrumb in the header (highlighted in the red rectangle) allows the user to go back to the previous pages visited, including the home page. When the user goes back by tapping on the breadcrumb, data such as tag filters and search result in the 'Explore page' are preserved.



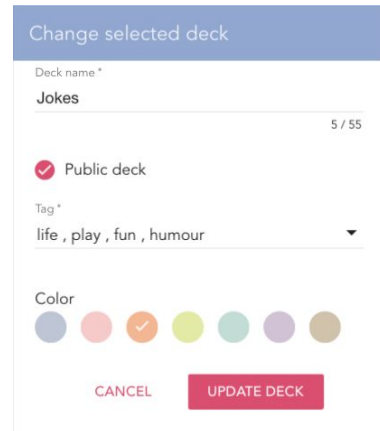
Principle 17: Minimize form errors with labeling and real-time validation

In all the form inputs where there is a limit on the number of characters, such as deck title and tag name, the number of characters is updated while the user is keying in. The counter turns red if the number of characters exceeds to warn the user.

A mobile app interface for creating a new deck. The title is 'Create new deck'. The first input field is 'Deck name *' with a red underline and a red counter '73 / 55'. Below it is a radio button for 'Public deck'. The second input field is 'Tag *' with a dropdown arrow. Below it are six color options: blue (selected), pink, orange, green, teal, and purple. At the bottom are two buttons: 'CANCEL' and 'ADD DECK'.

Principle 18: Design efficient forms

When user edit deck/card in our App, we pre-fill the form with the deck/card's existing information so it is easier to edit. When the user is in the 'Home' page and has chosen a particular tag from the sidenav, the create deck form will have this tag pre-filled. The screenshot is the form shown after select a deck to edit.

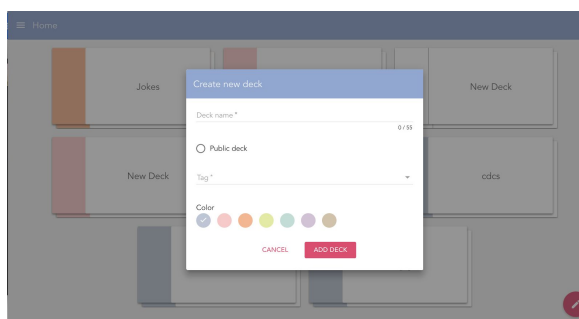


Principle 19: Optimize your entire site for mobile

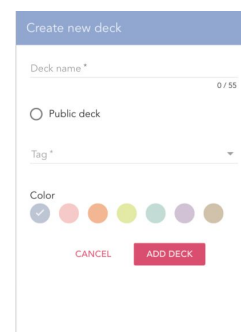
We used Angular Material library to make the layout mobile responsive so that the decks and cards fit the mobile screens.

Principle 20: Don't make users pinch-to-zoom

There is no horizontal scroll on mobile. We use dialogs to create/edit deck/card. While on desktop the dialogs' width is half the screen width, on mobile the dialogs are full-screen so that users can fill in easily. Below is a comparison of the add a deck dialog on desktop and on mobile.



Desktop



Mobile

Milestone 13: Describe 3 common workflows within your application. Explain why those workflows were chosen over alternatives with regards to improving the user's overall experience with your application.

1. Create a deck

Tap on the pencil floating action button (fab) → tap on '+' and enter the create deck dialog → enter deck name (compulsory), select if the deck is public, select tags(multi-select), select deck color and tap on 'ADD DECK' → dialog closes, a green toast that confirms creation success shows, the newly created deck shows.

We choose to group the editing actions all into the fab to avoid cluttering the UI. The deck creation is a dialog rather than a page so that when user goes to other pages and goes back the user won't be greeted with a deck creation page again.

2. Read cards

Tap on the deck → enter the deck overview page where the attributes of the deck and all the cards in the deck are displayed → tap on a card to go to the card page → tap on the card to flip the card → swipe left and right on the card to go to the next/previous card, or tap on the "PREVIOUS" and "NEXT" button

We provide a deck overview page so that the user can choose which card to view without swiping the card one by one. The deck overview page also shows details of the deck such as the original author if the deck was forked. This is better than going to the card straight from the deck. Tapping the card to flip and swiping the card to navigate is a more intuitive way on mobile to interact with the cards, compared to buttons.

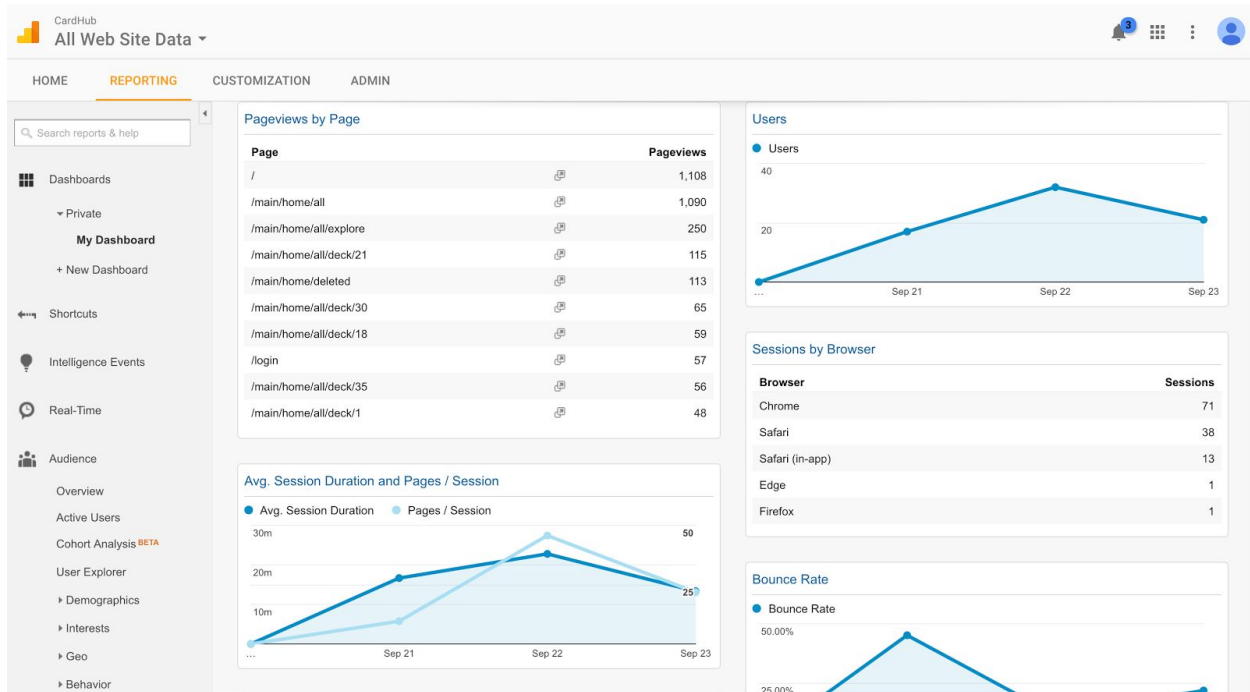
3. Fork a deck

Go to the 'Explore' page by selecting it from the sidenav → enter keywords in the search box and tap 'Search' → see a list of relevant decks → tap on the deck to see the cards inside → tap on the 'Fork' button to have a personal copy of the deck → green toast shows up if success, red toast if failure

Alternatively, user can see a list of recommended decks on the explore page and can fork them. Currently, the recommended decks are hand-picked by us, but in the future we intend to recommend decks that have the most number of forks if we have more user data.

This explore page can satisfies the needs of the users who have ideas in mind about what decks they want, and also those who just want to took a look at some random interesting decks. The fork button is in the deck overview page. This is more intuitive compared to putting the button in other pages such as on the deck itself in the searched result, because users would want to look at the content before deciding to fork the deck or not.

Milestone 14: Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.



Milestone 15: Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

We have integrated Facebook login and posting in our App. Facebook login makes it convenient for the users to sign up for our App. Posting allows the user to share his or her Cardhub profile page which contains the user's public decks. This helps to attract more users to our page.