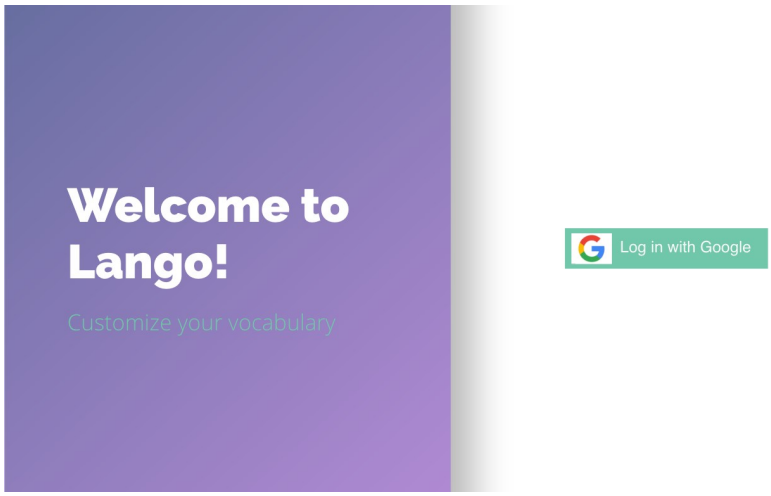
Flashcard App, Part 2
Spring 2019

Please submit by 10PM, Thursday 6/6, using Canvas.
We will accept submissions by teams of at most three. Everyone in the team should submit the same files, and will get the same grade.



We will now complete the flashcards application. The main new feature will be the login screen. Here is a demo, loginServer.zip, which allows users to login using Google, and then see a page ("hello.html") which is not publicly available. To make it work, you'll need to get Oauth2 credentials for the Google project you made for your app (where you got your API key). Paste those in where indicted in the code, and try it out. What happens when you go to "hello.html" when you are not logged in? When you go to "login.html" when you are?

You'll need to integrate this functionality into your existing pipeline, and extend it in several ways.

- Add a new table to your database, to hold user information.

It should include columns for first name, last name, and Google ID. You can use the Google ID as the unique primary key identifying each row of the table, and also as the userID for each flashcard in the Flashcards table (feel free to do this some other way, if you prefer).

- In the login code, find the "gotProfile" function which Passport calls once it successfully gets the Google profile information for a user. Here, check if the user is already in the database table. If so, you can just put the Google ID into the "dbRowID" variable for processing in the later stages. Otherwise you need to insert the new user in the table first.
- Next, find the "deserializeUser" function, which should get the Google ID passed in (from "gotProfile"). There, look up any information a later pipeline stage might need about the user from the database, and put it into the "userData" object. This gets stored as "req.userData" in the request object which all pipeline stages take as input. If you don't want to get data out of the table at this point, just use the Google ID as the user data.
- Alter your pipeline so that the create cards and review cards views are only visible to users after login.
- After login, redirect to a url called "auth/accept". From there, check to see if the user has any flashcards. If so, redirect to the review cards view. Otherwise redirect to the create cards view.

To finish the app, you'll need to implement the review cards view. To make the card flip, you can start with this demo, flipcard.zip (put it in /public, start your server and navigate to the page to see it go). It's a React component. The demo card flips when the user hovers over the card. Instead, our card should flip when the user clicks on the card, or when he hits return in the text entry box (whether or not there is text in the box). The "back of the card" should show the English text, if the user did not get it correctly. If he did, it should show "Correct!", as in the mock-ups.

Add an AJAX query so that your front-end Javascript can get the the user's first name, and add it to the footer as in the mock-ups.

The order of the flashcards shown is up to you. You need some algorithm that takes the number of times a card has been seen, and the number of times the user got it right, into account. I recommend this strategy, which is an example of a more general strategy for producing a non-uniform random distribution:

1. Define a score for each card, which indicates how much we want it to be seen. For instance, it might be:

   ```
   score = ( max(1,5-correct) + max(1,5-seen) +
   5*( (seen-correct)/seen) )
   ```

   Notice all three terms are between 0 and 5, since `seen >= correct`, that the total score is at least two, and that a card will get a score of 15 when it has not been seen at all.
2. Pick a random card belonging to the user, and compute it's score. Then, pick a random number between 0 and 15. If the random number is less than or equal to the score show the card. Otherwise, pick a new card and continue until you show a card.

For this to work, of course, you need to increment "times seen" and "times correct" in the database whenever the user reviews a flashcard. You are welcome to come up with other strategies!