Proj2

- Full Name: Oleksandr Stubailo (sashko@mit.edu)
- Heroku Url: http://powerful-river-9277.herokuapp.com/

Feature Descriptions

You must create an account to use Lists; once you sign in you can create as many lists as you want and then add notes to those lists.

If you are the owner of a list, you can manage permissions by going to a list page and clicking "permissions" at the bottom. This will let you add and remove users that are allowed to read, edit, and manage the notes in your list.

In each list, notes are displayed with the most recently edited one first, also showing which user was last to edit it.

The fancy additional JavaScript feature is that a warning pops up if you are looking at a list or updating a note and someone has edited it in the meanwhile. To test this, open a list in two tabs; add a new note in one tab and then a warning should pop up in the other tab.

Design Challenges

Permissions

Permissions systems come with many challenges. One of the main ones is how to have enough options to let users do what they want while also maintaining a reasonable level of complexity so people aren't scared away.

Options

1. Per-note sharing. This would allow each user to create a note and then share that note with other users. Essentially this becomes Google Docs because if one wants to share lots of content with others they must either keep it all in one note or share a large number of notes which would be tedious. If notes become too big, the app will need to have first-class collaboration features which are difficult to build and maintain.
2. Sharing sets of notes. This allows users to create folders or lists of notes that are shared between users. Then, if the user wants to share a new note with the same group they simply post to that list or folder. The downside to this is if the user wants more granular control over each note they need to create many lists.
3. Posting notes to other users. This would be very close to a social network model, like Google+ has. When creating a note one would also specify a set of friends that would be able to see it by going to the poster's profile. The downside to this option is that it is

already well done by many social network sites and so building it again wouldn't really help anyone.

Choice

I chose option number 2 because it is the one I would most like to use if I were working on a project or collection of notes with someone. This model could be used to keep track of a group todo list or a list of ideas. There are also potential extensions into importing lists from your email (your inbox is a list!), Google Tasks, contacts lists, etc.

More Options

Now that I had decided on using lists I had a few other choices to make about how one would assign collaborators to a list.

1. Assign permissions individually (read, write, edit, etc) to each user. This option would require a lot of work for the list owner every time they want to add a collaborator.
2. Have a predefined list of permission sets that the owner can pick from when adding users. This is a middle ground between 1 and 3.
3. Any user that is added to a list automatically has all permissions. This is nice because it is the simplest model by far, but then the owner of a list has to have complete trust that none of the collaborators or readers will mess up their carefully maintained list.

Choice

I decided on 2 because it was a good compromise. I may also add the ability to make a list publicly readable.

Notes on Code Design

Most of the code is pretty straightforward. There's one part in particular I'm proud of which is using separate scopes/relations for each level of permissions for lists/users. If you look in app/models/user.rb and app/models/list.rb, you'll see that there are the following relations:

- `user.lists`
- `user.owned_lists`
- `user.editable_lists`
- `list.owners`
- `list.editors`
- `list.readers`

The way they are coded lets us use beautiful rails magic to set permissions:

`list.owners << user` is the only code required to create the appropriate relation in the ListPermissions table. You can also do `list.owners.delete user` to remove someone's permissions.

I throught this was really neat and made the several hours it took to figure out Rails relation syntax totally worth it.

Model

Here is the diagram for my data model.
One thing that needs explaining is that a User can only be one of the relations to list - it can't be a reader, editor, and owner at the same time.