Fakesbook

Changes needed

- open source the project
- admin view
  - add friends between existing users
  - add/delete/update users
  - bulk-edit users
  - import/export database
    - randomize names of users?
      - possibly get a list of valid names for this
      - ensure inclusivity if using a name list
    - use a tar.gz and some predictable filenames
      - e.g. saved.db, thumbnail.png
      - don’t save with .tar.gz extension to prevent accidental opening
    - snapshot the graph svg, use as a thumbnail for saved db
      - maybe export as tiff / png?
    - import selector in admin view with preview thumbnails
    - customizable filename to export (“save as”)
- shred(1) automatically

Programming Interactive Education

- have students reconstruct 'damaged' database?
- reverse engineer/steganography on a database?
- “hack” the site?
- interactive graph theory?
  - BFS/DFS/Dijkstra's
  - Hamilton paths/circuits
- basic data science?
  - “Who would you share this post with to make the most people see it?”
  - “Are people with X characteristic more likely to Y?”

Admin view

The application needs additional configurability beyond manually creating accounts and customizing them. Additionally, it needs the ability for a professor/moderator to be logged in and view the live graph without showing up as a user of the system.

The solution to this is that it needs an admin interface. With username “admin” and some customizable (non-hardcoded!) password, a user authenticating to the system should see what the regular users see, except with no hidden profile data (this will require modification of the profile viewing logic).

Additionally the admin should be able to modify any profile data for any user, ideally even select groups of users and make batch changes.

The admin should be able to add and remove users.

The admin should be able to add and delete friendships between users.

The admin should be able to click a button to randomize the names in the live graph. A diverse selection of names should be imported to the project that can be randomly selected from without replacement.

The admin should be able to export the user database to a save file. This save file should capture all state of the live graph – this might be a privacy risk, but we put that liability on the professor/moderator running the lab.

When saving the graph database to a file, a predictable and extensible format should be used. I think the best would be to render the graph SVG to an image (JPEG or PNG) that can be used as a thumbnail – a reminder of what that graph looks like – and the SQLite database file. Saving these together in a Tar archive, then GZIPing the archive, would be simple, effective, and future-proof. The admin should be able to save this file with a custom name.

The saved file should then add an intuitive and unique filename extension for the app to look for when importing databases, such as .fbsave.

The system should then also be able to import saved databases. In doing this, a custom import selector will be needed. The import selector should be able to choose a filesystem path, then unpack all the files with the correct extension in that directory into memory (this will be tricky, but possible. The import selector shouldn’t leave a mess on the filesystem). From these, the selector should display the thumbnails of the available save files. When importing the database, copy the database file from it in, and when saving back out with the same name, replace the old thumbnail and database file. Do not delete anything prematurely.

This will be a challenging and complex change to the existing system, and so development should be done in a git branch and *very* thoroughly tested. Check in with Dr. Wood often to ensure the UI for these features is usable and elegant.

Shred automatically

Shred is a utility which securely deletes files stored in RAM/Hard Disk by writing over them with random data in multiple passes. Using this will ensure that student info can’t be leaked accidentally.

I’ve added a function to the Python code which will shell out a command to the system to securely delete a given filename. Use with *extreme* caution as this can’t be undone.

The admin user will need their own database for this to work properly. The system should therefore use two databases, one for the admin and other general configuration/runtime data, and the other strictly only for user data. This way the admin is not lost when the user database is shredded.

Finally, a button should be added to the admin interface to shred the user database. It should confirm this action with some kind of dialogue.