6.170 Assignment 3: Fritter Concepts

Objectives

Designing concepts. In the previous assignment, you implemented the basic functionality of Fritter following a design that we gave you. Essentially, you implemented two concepts, *Freet* and *Authentication*, which we designed for you by telling you how they should behave. In this assignment, you will extend Fritter with a collection of new concepts. You will become comfortable thinking about design in terms of concepts and expressing the design of each concept in an idiomatic way (in terms of purpose, structure, actions and operational principles). Even though the concepts will be familiar to you, you will have the freedom to design their detailed behavior as you please.

Data modeling. The design of a concept includes the design of the abstract data model underlying its behavior. If a concept is viewed as a state machine whose actions transition the machine from one state to another, the concept's structure is the *state* of the machine. The key challenge in data modeling is learning how to express the structure succinctly and abstractly, without including any data representation details (and thus leaving open not only the details of the representation but even what kind of database or data structure the data is stored in).

Data representation. In this assignment you will both design your concepts and implement them in your web server. This will include translating the abstract data model of the concept's structure to a concrete data representation in the code. In a practical, large-scale web server, the data is usually stored in a relational or value-key database. But for your assignment, you can store the data in local variables of the web server, creating a data structure using the techniques you've learned in previous programming classes. Of course, this means that the data will not be persistent, so if the server crashes, the state will be lost. **You may also choose to use SQL / SQLite for your data representation.**

Learning node/express. This assignment will give you practice coding with node/express so that you become more confident handling requests, dealing with sessions, and so on.

Specification

Pairs. You and your partner should discuss each others' implementations of the previous assignment and decide on which parts of them you will draw from. You do not need to use the code of only one partner or the other **(though you may choose to do this)**; in fact, ideally you will select the best parts of each to build on.

Concepts. You will augment your Fritter app with the following concepts: *Upvoting* (in which users can choose to "like" certain freets); *Following* (in which one user can choose to follow another); and *Refreeting* (in which a user can repost another user's freet).

Design Freedom. It is up to you to design these concepts: that is, to decide how they should behave. For example, for *Upvoting*, you will have to decide what actions are available to users, how votes are counted, what the consequences of upvoting are, and so on.

Implementation. As with the previous assignment, the focus of your design and implementation efforts should be the server-side functionality. In other words, you are building a web service rather than a full app. In the next two assignments, you will design and implement a client-side interface. Nevertheless, a rudimentary client-side interface is necessary to make it easy to test your web service. To build this, you should extend the client-side code from the previous assignment. It is not necessary for this client-side interface to be natural or easy to use, but it should be straightforward to issue requests and view responses, and it should be organized neatly.

Bad requests. As in the last assignment, your web service should handle requests that are ill-formed or illegal in a reasonable way.

Transient state. As in the last assignment, your web service is *not* required to make its state persistent.

Deliverables

Design. You will record your design in a succinct but comprehensive design notebook. This notebook **should be either a single *.pdf* or a single *.md*** file. It should contain:

- **Concept design models** for each concept using the standard template elements (purpose, structure, actions, operational principles). The structure (that is, the data model) can be given in diagrammatic or textual form, and should include any additional constraints. The actions can be written informally or as assignment statements using the set and relation operators. The operational principles should be written informally. The entire description should include explanatory comments when necessary, and should serve as a rudimentary user manual.
- **A design reflection** that lists each significant design decision you made, and for each one, the alternatives you rejected. Each of these decisions should be titled with a short and compelling title phrase that summarizes the design question being addressed; producing these titles will encourage you to focus effectively. In addition to explaining the different design options, you should succinctly explain the rationale behind your decision: why you chose the option you did, what was wrong with the other options, and what limitations your chosen option still has. Remember that design decisions concern only the observable behavior of the web service and *not* the internal structuring of the code.
- **A social/ethical reflection** that lists a series of design decisions that have social or ethical implications. These will likely be the same design decisions that you discussed in your design reflection, but they need not be. For each decision, you should explain why the decision discussed has particular implications, and, if those are negative, how they might be mitigated. Note that, in contrast to functional aspects of the design, you are *not* expected to resolve these aspects even if they are egregious; the reason for this exercise is to become aware of them. See hints below too.

Code. All of your code should be available in your GitHub repo. It will not be graded, but we may check your code to make sure that there is no evidence of copying from the work of other students.

Deployment. You should deploy your code so that your web service and the provided GUI can be accessed at a public URL. See the deployment guide for instructions. Make sure to use a new name so that the deployed instance of your previous assignment is still accessible in case it has not yet been graded.

Grading

See the accompanying rubric.

Hints

- **Help with concept design.** For background and practice with concept design, make sure to attend class, come to recitation, and make good use of TA office hours.
- **Incremental design notebook.** In our experience, it’s often hard to look back on a design process and “un-see” the solution you’ve crafted. Once you’ve hit a stopping point in your design process, your artifact will feel like the only or most obvious way to address the design problem, and chances are you will have a difficult time remembering the alternatives you had considered, or why you had considered them. Consider keeping a scrappy design notebook to document your process (e.g., a rough set of bulleted points that describe what you’re currently working on, what issues are you hoping to address, why do you believe this idea might work, and what limitations did you observe once you implemented it). You can then synthesize your design reflection from your notebook, and doing so will feel like less of a chore (or even a writeup you invent from whole cloth!).
- **Social/ethical reflection**. To help you think about the social and ethical implications of your design, here are some prompts for what you might think about. Who are the *stakeholders* in your design? These are not just the immediate users; for example, Uber's stakeholders include not just drivers and passengers, but also others on the road, taxi drivers, and the environment. What *time scales* are relevant? Does your design have different implications for its immediate and future use? What *cultural* considerations are at play? What *malicious* uses might your application be put too? What different *usage* scenarios might your app have? Consider for example how it might be used by children, or within a company. Are there places where your interests as a developer are *misaligned* with the interests of your users or other stakeholders? How might Fritter affect *power dynamics*? For example, are powerful people or institutions likely to exploit Fritter to gain even more power?
- **Tagging in Git.** It might be useful to tag the last commit for your previous assignment before beginning this one, so that you can quickly reference past code while working in the same code base.