6.170 Assignment 2: Fritter Service

Overview

This is the first in a series of four assignments in which you will build Fritter, a clone of a well-known social app whose primary purpose sometimes seems to be to fritter your time away.

In this first assignment, you will implement the server-side functionality required to make a very basic version of Fritter work. You will create a web service to handle user accounts and allow users to create, read, update, and delete short messages called Freets. For this assignment, we are providing you with a basic client-side user interface to test your work; you will need to write a small amount of client-side code to connect it to your web service.

In the upcoming assignments, you will extend the functionality to include richer behavior, and you will design and implement a graphical user interface for Fritter. Because those assignments build on this one, you will want to design and code this initial version carefully so that you save time in the long run. This is especially important if you are working in a pair, as you and your partner will need to understand each others' code to select the best parts of each and build on them.

Objectives

Learn essential web service structure. In this assignment, you will become familiar with the key concepts of a web service including routes, requests, verbs, and status codes. These are the fundamentals of how communication over the web works.

Practice designing a web service. You will make design decisions regarding what information is required by your routes and what the responses should be. Particularly, you will need to think about how to notify users if, for some reason, the operation they requested cannot be completed.

Understand the power of REST. You will gain an understanding of how to design a web service with routes that are RESTful -- that is, conforming to the Representational State Transfer (REST) architectural style. RESTful services are easier to maintain, and they make it easier for users to guess how they work.

Learn how to work with sessions. Sessions allow you to keep track of the current user and are used by many web services. You will learn how to manage sessions using cookies, and how to use them to control access to operations.

Gain experience with Node and Express. These are two very popular tools used in combination to create web services. With Node, you can use the same language (JS) on both the front-end and the back-end of your web application, and Express's lightweight design allows you to quickly get web services running.

Specification

Functionality. A user can create, edit and delete freets, which are messages of 140 characters or fewer that are displayed along with the user's name, and can view the freets of other users. A user can create an account that is accessed with a username and password, both of which can be changed, and can also delete the account.

Access. Freets can be viewed without logging in and even without an account, but all other actions on freets require the user to be logged in. Users can only edit and delete their own freets, and modify or delete their own accounts.

Routes. Your web service should be accessible through routes that follow the principles of REST as discussed in class (i.e. route structure and HTTP verbs).

Robustness. Your service should fail safely and gracefully with appropriate status codes and error messages when bad requests are presented. These include requests for non-existent operations, and access violations.

Persistent storage. You are *not* required to store account and freet data persistently. So you can store the application state in JavaScript variables in the server code. This means that data will not survive a server restart.

Basic front-end. We are providing you with a basic front-end to test your work. However, you will need to write a small amount of client-side code for calls to your API.

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.

Design Reflection. Note that the specification given here is not complete, and leaves you various opportunities to make your own design decisions. You will document these in your design reflection. Your reflection should address the following questions:

- What ambiguities or omissions did you discover in the specification that needed to be resolved?
- What design decisions did you make in regard to these, and other aspects of the behavior, and what was your rationale for them? What alternatives did you consider and why did you reject them?
- For each of your design decisions: do they have ethical or social implications, and if so, what?

Note that your design reflection should *not* include any engineering decisions about the structure or internals of the code. We have now moved to design as explained in the opening lecture, which concerns the shaping of the observable behavior and not the engineering of the implementation.

Deliverables

- The entire code of this project, including the front-end user interface, is provided in the starter code.
- A file called *reflection.md,* in the top-level directory of the repo, that contains the design reflection.
- Filled out *critiques.md.*
- Submit the deployment URL so that the TAs can run your application along with your final commit hash and the URL to your GitHub repo for this assignment.

Collaboration Policy for Partner Assignments

Please review and follow the collaboration policy for partner assignments on the course syllabus.

Grading

See the accompanying rubric.

Hints

- **Attend recitation.** This week's recitation will let you practice playing with the code of a simple web service with a similar GUI, sending requests, and extending an API with routes that are session-aware.
- **Platform.** You must build your app in the Node and Express frameworks; we don't have the teaching resources to support alternative frameworks. In line with the course's policy on code reuse, you are free to use any modules in the Node ecosystem. In particular, aside from Express, you may want to look at the following packages: Cookie Parser, Express Session, UUID. You may *not* use code that covers a large part of the application-level functionality; for example, you have to implement authentication yourself. Feel free to ask on Piazza if you have questions about this.
- **Testing.** While unit tests are not a part of your grade, they can be useful for making sure your code works (and doesn't break when you change it). Jest is our recommended test library, and Supertest Session lets you send session-aware requests to your API without a browser.
- **Code Quality and Documentation.** We expect you to organize your code thoughtfully, with appropriate structure, names, comments, etc. You don't have to follow any particular documentation style (although if you want to, JSDoc is a popular one) — just be consistent throughout your code. We will not be grading your code, but if you write sloppy code, you will likely find it very hard (especially in later assignments that build on this one) to achieve reliable functionality (which *will* be graded).
- **Authentication of users.** Simple username and password checking suffices. You can store your passwords in plain text.
- **Ethical considerations.** Examples of ethical considerations include: What impact might the order in which freets appear have? Will a user know if a freet has been edited? Could one user masquerade as another?

RES.TLL-008 Social and Ethical Responsibilities of Computing (SERC)
Fall 2021