Published 18 Sep 2018

6.170 Assignment 3: Fritter, Part I

Due: 25 Sep 2018 @ 11:59 a.m. — Submission Link

Overview

This is the first challenge in a series of three 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. The actual purposes of Fritter include enabling short and timely communications between users, helping users promote themselves with public postings, and letting users filter their views to focus on other users of interest.

In this first assignment, you will implement the server-side functionality required to make Fritter work. That is, 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 UI 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 next two assignments, you will design a graphical user interface for Fritter and use a database for persistent storage, respectively. The next two assignments may also involve implementing additional features, so thinking carefully about the design of your API now may save you time in the long run.

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 an API with routes that are RESTful -- that is, conforming to the Representational State Transfer (REST) architectural style. RESTful APIs 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 thus used by many web services. While authentication techniques are beyond the scope of the class, you will provide a basic login, and you will define actions whose behavior depends on who the current user is.

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 an account with a username and password and sign in and out of their account. An account owner can change their username and password and delete their account. An account owner can create a Freet, a textual message of 140 characters or fewer. A Freet has an author and a number of votes, which can be changed by upvoting and downvoting. An account owner can edit and delete their Freets. Any user can see the global list of Freets and search for Freets by author.

Robustness. Users should not be able to change each other's usernames and passwords or delete each other's accounts. Users should not be able to create Freets or vote on Freets without being signed in. Users should not be able to edit or delete Freets they did not author. While your API need not be robust to arbitrary request bodies, it should fail safely and gracefully with appropriate status codes and error messages for all requests that could be made from the GUI or by navigating to a URL in the browser.

Design. The details of your web service are up to you. However, your routes should follow the principles of REST as discussed in class (i.e. route structure and HTTP verbs). You should document the key design decisions you make in a markdown file (e.g. README.md).

Code Quality. We expect you to organize your code thoughtfully, with appropriate structure, names, comments, etc. This will ease the modifications you are likely to need to make when you build on this code in the next two assignments.

Technology. You must use the Node and Express frameworks. You are allowed to use any modules in the Node ecosystem provided they do not do a significant portion of the assignment for you. If you think a package you found might do too much, ask on Piazza before using it. You may not use a database or any persistent storage; just store the application state in JavaScript variables in the server code. This means that data will not survive a server restart.

GUI Configuration. 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 must deploy your code so that your API and the provided GUI can be accessed at a public URL. See the deployment guide for instructions.

Grading

We will run your code using Node version **10.10.0** or higher.

Here is the rubric we will use to grade your submission.

Don't forget to fill out the submission form. If you fail to fill out the form before the deadline, your assignment will be marked late.

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.
- **Node packages.** Aside from Express, you may want to look at the following packages: Cookie Parser, Express Session, UUID
- **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.
- **Documentation.** 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.
- **Authentication of users.** We have not covered security, so you do not need to implement any user authentication beyond what was specified: simple username and password checking. You can store your passwords in plain text.
- **Vote management.** We did not specify a limit to the number of times a user may vote on a given Freet. However, it may be wise to consider what changes you might need to make later to add or remove such a limit.