Design

The project follows the MVC pattern.

Model: The User and Frit model files reside in /models and are based on Mongoose schema models.

View: The view consists of the EJS files in /model, the client-side JavaScript in /public/javascript, and the CSS file in /public/stylesheets.

Controller: The routes in /routes make up the controllers.

Tech Stack: Mongoose for database management, EJS for templating, Express-Session for cookies, Bootstrap for UI, Underscore for input sanitation, Bcrypt for password encryption

Overview: When the user accesses the site, if they are logged in (have userId in cookies), it immediately redirects them to their feed. Otherwise, it directs them to the login screen, where they either login or register, and then get redirected to their feed. The feed displays all of the Frits posted to the site so far. Each Frit can only be modified or deleted by the user that posted it (based on the user's email – which is unique to the system). The user can post new Frits here, view other Frits, modify or delete existing Frits, or logout (deletes cookies and redirects to login page).

Challenges

1. What module to use to connect MongoDB to Node: I was deciding between Node-mongodb-native, Monk, and Mongoose. I choose Mongoose because it is an ORM and abstracts out database queries into more human-understandable object functions. It also helped me create my model, which will come in handy when I start adding more complicated relationships.
2. Schema design of Frit: I could've made each Frit just an entry in my User schema, or created a Frit schema and connected it “relationally” to its User. I chose the latter because my Frit object could have a lot of metadata (right now it just has date) that would make it difficult and clunky to manipulate as just a series of fields in User.
3. Schema design of the User-Frit relationship: I could've had the User have a list of Frits or not hold such information and just query it (a Frit would know its User). I decided on the latter because it would be redundant to have to update in two places if a Frit gets

deleted or changed.

4. User authentication: I had the option of OpenID, username-password, and cookies. I opted to use email-password for user accounts because emails are unique and would result in less frustration (duplicates are still checked for). I also used cookies so users would stay logged in. I hash all passwords using Bcrypt for security.
5. How to prevent modification of a Frit by users that are not the author of the Frit: I could create a separate view with only editable Frits, or I could display all Frits, but only have editing buttons next to those that are editable. I decided to do the latter, matching the Frit's author's email against the logged in user's email (these are unique) to determine if the Frit is editable. This option is the most convenient solution for the user because everything is in one place.

Grading Directions

1. My User model/schema doesn't hold references to its Frits to avoid redundancy (Frits know who they belong to). But it does provide methods that query and sort those Frits, returning a list of Frits with populated information about the User model (using Mongoose's populate) to prevent further queries to retrieve that information.
2. I separated the pure HTML/EJS files from the javascript/jquery manipulating the viewto prevent clutter and to separate concerns: the HTML/EJS dictate what is displayed, the javascript/jquery make the pages pretty and responsive, but are not vital for grasping what is going on in the pages.
3. To avoid callback hell, I separated and named methods that were getting too long, such as the async password validation that happens upon login.
4. I used cookies to automatically redirect already logged in users to their feed to prevent them from having to re-login. Similarly, if they are not logged in, the feed redirects them to the login page. This prevents users from seeing pages they aren't allowed to see yet, and manages the page flow for them so they don't have to memorize URLs to get to different pages.
5. I have checks in both model and view so that the necessary and valid things get added to the database. I also have nice error handling for the login and Frit editing pages. There are placeholders that are filled with error and success messages if there are any.