Feature Flags

The following guide will explain what feature flags currently exist and how they work.

Current Feature Flags

- `subscriptions`
  - Controls whether or not a user needs to have an active subscription to use the app.
  - For each service, it controls the following things:
    - Backend
      - Whether or not the `authenticate` allows or blocks users without a subscription.
      - Whether or not the `authentication` service tacks on the subscription properties during login.
    - Frontend
      - All of the subscription selectors depend on the flag to determine whether a use can use the app without a subscription.

How Feature Flags Work

The feature flags are all defined in the Backend service as part of the Feathers config -- `config/default.json`.

Feature flags are currently not scoped differently per environment; whatever's defined in `default.json` is the flag value.

The Backend service reads these flags into the `featureFlags` service. The `featureFlags` service exposes a single method -- `find` -- which just returns a map of all the feature flags and whether or not they are on.

This service is then used to construct a hook -- `featureFlagEnabled` -- that is then used to toggle other functionality (primarily, other hooks) on or off.

The `featureFlags` service is also exposed externally (i.e., only the `find` method is). This allows the Frontend to use the flags.

The Frontend fetches the feature flag values at every page load and stores them in Redux. Selectors can then be used to query the value of the feature flags. Note that there are

default values for the feature flags specified on the Frontend, in case the Backend is unavailable.

I chose to have the Frontend always fetch the latest flag values to get around users who use the app with a service worker -- this way, all users will always have the most up-to-date flag values.