1 Working with the JavascriptMVC backend.

Working with the javascript MVC backend involves making AJAX calls to the dynamic fixtures created for the various models. Each such call will receive a response of JSON data, either a single object or a list of objects. These objects describe the different model's data, and it's through these objects that information is grabbed.

If that makes no sense, don't worry, let's explain it a bit, but first a quick glossary for Fixtures and JSON.

1.1 Fixtures

Fixtures are a way to simulate HTTP requests in order to grab information. That means that, for example, if we want to get all the beers, we simulate a `GET` request to `/beers`. JavascriptMVC intercepts this request and executes the relevant code in models/fixtures/fixtures.js.

An example of a fixture, as it exists in fixtures.js, looks like this:

```
"GET /beers" : function(request, response)
{
  // Do stuff with the request, then call response with the
      relevant information.
}
```

This means that when we use AJAX to perform a get request to `/beers`, we will do the code inside that function. The function will then return with a response call, which is how information will be handled when the request is completed, as will be shown below.

1.2 JSON

JSON is a way to structure data in javascript, essentially it's a more human-readable alternative to XML used by Java and Javascript. For our purposes, the chief advantage of it is how easy it is to get information out of JSON objects in javascript.

For a JSON object `json`, any value can be retreived by name. For example, if we want to find the `description` field of `json`, we just refer to it by its name as such: `json.description`.

1.3 AJAX

With that out of the way, here's a more detailed example of how to make use of the backend, through AJAX calls.

An AJAX call is a call to the function `$.ajax()` with a JSON object whose values decide what operation to perform. If we want to fetch the inventory from the database using the username *tehoi* and password *tehoi*, we would make the following call:

```
$.ajax(
        {
          type: "GET",
          url: "/beers",
          data:
          {
            username: "tehoi",
            password: "tehoi"
          },
          success: function(json)
          {
            // json.data here is a list of all beers.
          },
          error: function(json)
          {
            // Error information is in json.
          }
        }
      );
```

and that's pretty much it. That call will make a GET (see the type: parameter) request to `/beers` (the url: parameter). The username and password are set inside another JSON object named data:, and that's how you pass along any additional information necessary by the function, such as id numbers and so on. success is a function, with a JSON object as parameter, that is executed if the request succeeds. It's in this function any code using the requested information goes and the information is accessed through `json.data`. In the above example, `json.data` contains the list of beers as returned by the database.

There is an exception to the above format, and that is when you wish to perform operations on single-data, such as logging in. When performing that kind of request, the fixture is of the format `/logins/{username}`, which means that instead of defining the username in the data section of the ajax request, it's defined in the url as follows:

```
$.ajax(
        {
          type: "GET",
          url: "/logins/tehoi",
          data:
          {
            password: "tehoi"
          }
          ...
        }
      );
```

1.4 List of fixtures

This section is not finished yet; for a detailed if a little complicated to read overview of all fixtures, checkout models/fixtures/fixtures.js. The variables assigned to `request.data. ...` are the parameters that need to be assigned in the ajax datablock, with exceptions for fixtures of the above format, where one of the parameters is defined in the url (the parameter is defined in the fixture URL in fixtures.js inside { and })