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

1.4.1 Beers

GET /beers

Using an AJAX get request to /beers retrieves the full inventory of available beers. The list is not cleaned up, so result may contain garbage data.

The request requires the following data:
Username

Password

The response is a list of JSON objects with each element containing the following information: beer_id
count
namn
namn2
price
pub_price
sbl_price

Example:

```
$.ajax(
        {
          type: "GET",
          url: "/beers",
          data:
          {
            username: "tehoi",
            password: "tehoi"
          },
          success: function(json)
          {
            console.log(json.data[0].namn); // Print the
                name of the
            // first beer to the console.
          },
          error: function(json)
          {
            // Error information is in json.
          }
        }
      );
```

GET /beers/{id}

This request returns detailed information about a single beer.

The required data, beyond id, are:
username
password

The request returns a JSON containing the following:
alkoholhalt
argang
artikelid
ekologisk
forpackning
forslutning
koscher
leverantor

modul
namn
namn2
nr
prisinklmoms
prisperliter
producent
provadargang
saljstart
slutlev
sortiment
ursprung
ursprunglandnamn
varnummer
varugrupp
volymiml

Example:

```
var id = 148803;
$.ajax({
  type:"GET",
  url:"/beers/"+id,
  data:
  {
    username:"jorass",
    password:"jorass"
  },
  ...
  });
```

PUT /beers/{id}

The PUT request updates an item that already exists in the database, that is to say it is how you change the price or amount of a beer, such as when one is sold.

The put request requires the following data beyond id:
username
password
amount
price

The request does not return anything beyond success/failure.

Example:

```
var id = 148803;
$.ajax({
  type:"PUT",
  url:"/beers/"+id,
  data:
  {
    username:"jorass",
    password:"jorass",
    amount: "0",
    price: "10.90"
  },
  ...
});
```

1.4.2 Accounts

GET /accounts/{username}

This request retrieves information about a user.

The data requiered, beyond a username, is a password.

The request returns the following information:
user_id
first_name
last_name
assets

Example:

```
var user = "jorass";
$.ajax({
  type:"GET",
  url:"/accounts/"+user,
  data:
  {
    password:"jorass",
  },
...
  });
```

PUT /accounts/{user_id}

Identical to **PUT /payments/{user_id}**.

1.4.3 Purchases

GET /purchases

This request grabs a list of all purchases registered on the server. It requires the username and password of an administrator as data.

The request returns a JSON containing:
beer_id *The beer purchased*
first_name

last_name *The name of the user who placed the purchase*
namn *The name of the beer*
namn2
price
timestamp
transaction_id
user_id
username

Example:

```
$.ajax({
  type:"GET",
  url:"/purchases",
  data:
  {
    username:"ervtod",
    password:"ervtod",
  },
  ...
  });
```

GET /purchases/username

Similar to above, except it returns only the purchases made by the user in question. Does not require admin privileges. The data required, beyond the username, is a password.

The returned data is the same as above, except the following has been **removed**:
first_name
last_name
username

Example:

```
$.ajax({
  type:"GET",
  url:"/purchases",
  data:
  {
    username:"ervtod",
    password:"ervtod",
  },
  ...
  });
```

POST /purchases/' This adds a single purchase to the database.

This does not require admin privileges and the data required are as follows:
username
password
beer_id

The request returns nothing beyond success/error.

Example:

```
$.ajax({
  type:"POST",
  url:"/purchases/",
  data:
  {
    username:"jorass",
    password:"jorass",
    beer_id: 14882
  },
  ...
  });
```

1.4.4 Payments

GET /payments

This request retrieves all payments made by users into their account-assets. The request requires an admin's username and password.

The request returns the following data:
admin_id *The ID of the administrator who logged the payment*
admin_username
amount
first_name *The name of the owner of the account.*
last_name
timestamp
username *The username of the account being paid into.*

Example:

```
$.ajax({
  type:"GET",
  url:"/payments",
  data:
  {
    username:"ervtod",
    password:"ervtod"
  },
  ...
});
```

GET /payments/{username}

Like above, but returns payment information only about a single user. Does not require admin privileges but does require username and password.

The information returned is as follows:
admin_id

amount
timestamp
transaction_id
user_id
Example:

```
$.ajax({
  type:"GET",
  url:"/payments/ervtod",
  data:
  {
    username:"ervtod",
    password:"ervtod"
  },
  ...
});
```

PUT /payments/{user_id}

This request adds a new payment onto the account of the user with id user_id. This action requires administrator privileges.

The request takes the following data:
user_id *The id of the user whos account is to be paid into.*
username *The username of the admin approving the payment.*
password *The password of the admin approving the payment.*
amount

This request returns nothing but success/failure. Example:

```
$.ajax({
  type:"PUT",
  url:"/payments/2",
  data:
  {
    username:"ervtod",
    password:"ervtod",
    amount: 50
  },
  ...
});
```

1.4.5 Logins

GET /logins/{username}

This request performs a login operation and returns some basic user information. The request requires the username and password of a non-admin user.

The request returns the following data:
user_id
username
password

first_name
last_naem assets

Example:

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

GET /logins/admin/{username} Same as above but for admins.

The request returns the same information as above as well as a list of all purchases stored in purchases_list.

Example:

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