CS496 Final Assignment  
Eric Rouse

## URL to video:

## <http://youtu.be/jXB_biCTawc>

## Platform Used

I used the WebIDE from Mozilla to create a mobile application for FirefoxOS. This framework uses standard web technologies to create an application for a real mobile device. So, HTML, CSS and JavaScript used to create a mobile app. I also added the use of jQuery mobile, although not required by Firefox OS. I did this for a cleaner and more adaptive interface.

## The Functionality

This application can add and retrieve local notes, view current position and save user-specific remote data, as well as add, update, delete and view the players and bowls from the rothbowl api (detail below). It also has impliments login/logout account interface.

All the posts are handled by data.js, player.js and bowl.js, the javascript files that back the buttons on those pages. All source included in zip file. Example POST listing below:

[player/bowl]ID = $("#[player/bowl]ID").val();

var url = "https://rothbowl.herokuapp.com/api";

var params = "delete=True&member-type=[player/bowl]&member-id="+[player/bowl]ID;

// If you don't set the mozSystem option, you'll get CORS errors (Cross Origin Resource Sharing)

// You can read more about CORS here: https://developer.mozilla.org/docs/HTTP/Access\_control\_CORS

request = new XMLHttpRequest({ mozSystem: true });

request.open('POST', url, true);

request.responseType = 'text';

request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

request.setRequestHeader("Content-length", params.length);

request.setRequestHeader("Connection", "close");

request.onreadystatechange = function() {//Call a function when the state changes.

if(request.readyState == 4 && request.status == 200) {

//alert(request.responseText);

alert("Your [player/bowl] has been updated on rothbowl via the API");

}

}

request.send(params);

## Rothbowl API

The url structure of this site is simple and is based on <http://rothbowl.herokuapp.com/api>. A call to this base folder is interpreted as a GET (by browser default) and lists all elements (and their properties) in all databases in order. Further refinement is possible, by appending “/scan”, “/player”, “/bowl”, “/count” or “/path” to this base URL. Those will:

* Scan – list all the bowl objects in the bowl\_list set
  + Ex: <http://rothbowl.herokuapp.com/api/scan>
* Player – list all the players in the player hash
  + Appending a player id lists only that players attributes
  + Ex: <http://rothbowl.herokuapp.com/api/player>
  + Ex: <http://rothbowl.herokuapp.com/api/player/0>
* Bowl – list all the bowls in the bowl hash
  + Appending a bowl id lists only that bowls attributes
  + Ex: <http://rothbowl.herokuapp.com/api/bowl>
  + Ex: <http://rothbowl.herokuapp.com/api/bowl/0>
* Count – shows the current count of bowls and players
  + Ex: <http://rothbowl.herokuapp.com/api/count>
* Path –shows the current path, this was used for debug
  + Appending any value will include these values in the returned path
  + Ex: <http://rothbowl.herokuapp.com/api/path>
  + Ex: <http://rothbowl.herokuapp.com/api/path/count/ex/12345>

That concludes the GET commands, the rest of the commands are based on POST; they are post, put, patch and delete. To use these one must submit a POST request to the base url of <http://rothbowl.herokuapp.com/api/> with an identification in the POST data that specifies the type of POST (Ex: put=True, patch=True, or delete=True, any other type will be interpreted as post). For example: curl --data "put=True&player-put=True&put-id=12&fname=put&lname=test" rothbowl.herokuapp.com/api will put a new player at id 12 with the name “put test”. This player would then be located at <http://rothbowl.herokuapp.com/api/player/12>.

Full post, put, patch and delete listing:

* player add via POST
  + player-add=True&fname=***<inputparam>***&lname=***<inputparam>***&winners=***<inputparam>***&natl-champ=***<inputparam>***&scores=***<inputparam>***&points=***<inputparam>***
* bowl add via POST
  + bowl-add=True&name=***<inputparam>***&favorite=***<inputparam>***&underdog=***<inputparam>***&predicted-spread=***<inputparam>***&fav-score=***<inputparam>***&und-score=***<inputparam>***&tot-score=***<inputparam>***&winner=***<inputparam>***&loser=***<inputparam>***&actual-spread=***<inputparam>***&game-is-played=***<inputparam>***
* player put via POST
  + put=True&player-put=***<inputparam>***&put-id=***<inputparam>***&fname=***<inputparam>***&lname=***<inputparam>***
* bowl put via POST
  + put=True&bowl-put=True&put-id=***<inputparam>***&name=***<inputparam>***%20***<inputparam>***t
* patch bowl via POST
  + patch=True&member-type=bowl&member-id=***<inputparam>***&key=name&val=***<inputparam>***
* patch player via POST
  + patch=True&member-type=player&member-id=***<inputparam>***&key=name&val=***<inputparam>***
* delete player via POST
  + delete=True&member-type=player&member-id=***<inputparam>***
* delete bowl via POST
  + delete=True&member-type=bowl&member-id=***<inputparam>***

Logged in users have a few more options:

* data put via POST
  + user-data=True&username=***<inputparam>***&request\_type=put&data1==***<inputparam>***&data2==***<inputparam>***&data3==***<inputparam>***
  + HTTP response is a JSON string of the form: {'data1': ***<outputdata>***, 'data2': ***<outputdata>***, 'data3': ***<outputdata>***, 'username': ***<outputdata>*** }
* data delete via POST
  + user-data=True&username=***<inputparam>***&request\_type=delete
  + HTTP response is plain text: Deleted: ***<outputdata>***
* data get via POST
  + user-data=True&username=***<inputparam>***&request\_type=get
  + HTTP response is a JSON string of the form: {'data1': ***<outputdata>***, 'data2': ***<outputdata>***, 'data3': ***<outputdata>***, 'username': ***<outputdata>*** }

See the table below for all properties in a tabulated format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TYPE | Bowl | Player | | Championship Game | | User |
| PROPERTIES | Bowl Game Name | First Name | | Bowl Game Name | | Username |
| Favorite Team Name | Last Name | | Favorite Team Name | | Data1 |
| Underdog Team Name | List of Spread Winners | | Underdog Team Name | | Data2 |
| Predicted Point Spread | List of High Stakes Bets | | Favorite Team Actual Score | | Data3 |
| Favorite Team Actual Score | Predicted Championship Team | | Underdog Team Actual Score | |  |
| Underdog Team Actual Score | Championship Game Score | | Total Score | |  |
| Total Score | Number of Points | | Winning Team Name | |  |
| Winning Team Name |  | | Losing Team Name | |  |
| Losing Team Name |  | | Has Game Been Played? | |  |
| Actual Point Spread |  | |  | |  |
| Has Game Been Played? | |  | |  |  |

## Rothbowl User Accounts

I made the user account system by hand. It sends the user name and unhashed password as plain text via a POST command. This is handled in login.js.

The create\_request() function of login.js takes the entered username and password and sends them via POST to a PHP page (<http://web.engr.oregonstate.edu/~rousee/CS496/ajaxlogin.php>) that queries the “users” database using a prepared statement. The login PHP script responds with a 0, 1, 2 or 3. A response of zero means that login or new user creation was successful. The 1, 2 and 3 errors are: duplicate username, incorrect username and incorrect password respectively.

The create\_request() uses this response data to determine what data to show the user. Errors are displayed for the error codes, or the main page is launched for a successful log in. Also, upon successful log in, the current username is written to local storage for use by other parts of the application.