Andrew M. Calhoun

CS496-Final Project Post Mortem

Due June 5, 2016

My final project is the prototype for a sommelier app I am developing, called Vin.ly, which I was inspired to create as my parents have recently bought a vineyard in Sonoma County, and I was rather uninspired by many of the food and drink pairing apps on the market. Although this is in an early state alpha, it has some functionality that lets users create an inventory of wines they have or might have enjoyed and find foods that might match with them in a limited food pairing library (at the moment). The program is also planned to grow to include beer and spirits as well, as well as one day maybe allowing users to crowdsource drink and food combinations.

For the moment; however, it lets users keep track of their wines and even search their inventories by type, region, or year of vintage. Users can also add, delete, and edit wines in their collections and the system cross-references user names to wines in their collection, so multiple users who have the same named, typed, regioned, or yeared wine do not overlap or see each other's data. There is also a GPS functionality to help lost users find out where they are and get some direction.

The primary user-wine functionality has been thoroughly debugged, but there is a lingering issue with the food-wine pairing that does not necessarily respect user inventories with certain types of wine. However, the two compare each other in the API URI call, and respect each other when used in a RESTful API interface.

I used all four of the major RESTful API functions (GET, POST, PUT, and DELETE) to give interactivity to the system. A user can get their wine list, wines they have of a particular region/type/year, edit those wines, or get rid of wines through deletion.

The native functionality was built in C# through Xamarin for Android, and can be easily ported to iOS and Windows Phone as well. The cloud backend was done in node.js and JavaScript and is hosted on Amazon Web Services.

The API can be accessed at: http://52.27.116.225:8000/api/.

A demonstration video can be found at: https://youtu.be/MSF3YMdkcfA

Give us the list of URIs, what calls to those URIs do using the 4 major HTTP verbs (GET, POST, PUT and DELETE).

Within the program, there are several URIs used, both from the old API that I did earlier in the term, as well as account based adjustments:

URIS:

router.get('/user/:userName/wine/ - get all wines assigned to a particular user by the username parameters.

router.get('/user/:userName/wine/region/:region – used to find wines for the user by region.

router.get('/user/:userName/wine/year/:year – used to find wines specified by year assigned to the user..

router.get('/user/:userName/food/wine/:type – used to find wines specified by types associated with the username.

router.post('/user', - used to create new users or check if a name is already taken.

router.post('/user/:userName/wine/ - allows the user to post wines with their names, regional origin, year, and type.

router.put('/user/:userName/wine/:name – allows the user to make edits through the native interface, based on the content on the edit page.

router.delete('/user/:userName/wine/:name – allows an individual user, set by the :username parameters to erase the wine that is named in the parameters.

router.get('/food' – used to contact the API and get the food listings and find matches to wines. Uses arrays for each food with compatible wine types.

Not available from program, but available through RESTful Interface for admin purposes

router.delete('/user/:id' – erase user account by Id.

router.delete('/user/name/:name – allows the administrator or accessor to delete user accounts, particularly by name if duplicates appear and the system needs to be purged.

Finally, discuss your account system. If you created it by hand the standards are lower than if you use an 'off the shelf' implementation. For example, if you create your own implementation it would be fine to simply send a username and password in the clear as authorization and authentication. This would obviously be a very poor option in terms of security, but it will be a fairly complex problem without worrying about security. If you use something like an implementation of OAuth then it should generally work as intended.

I created my implementation by hand, using a separate collection in my database to store the users. While it is not as secure as other off-the-shelf options, it allows me to consolidate much of the information and create relations between wines, foods, and users, as well as easier further development for information checking. The system stores the user name and password in the user collection of my database. Eventually I plan to secure and hash this collection, but that was beyond the scope of this assignment.

The user name when signed in is stored in a global variable until log out and uses this to associate searches with a particular user. This way, a user can see their data and only their data. It also allows

users to have wines of the same name, year, and type without there being any confusion or error in the data gathering or misattribution of a wine from one user to another.

As the data is stored in a NoSQL database, it persists from session to session, so when a user logs out, logs in under another account, and then back into their account, their information will be exactly as they left it.

When a user creates an account, it does check to see a username is not already in the system. If the user's name is already taken, the system notifies the user. Likewise, it also alerts the user if there is a name or password mismatch or if a name is not in the system.