Portfolio 3
PizzaSeller 2.0
Michael Bonpua and Rushabh Shah

Description: This web application orders pizza saving it to a cart asks for user information on where to send the order and then finalizes the order for delivery or carryout while also outputting information on the specific order including user data.

Table of Contents

Overview……...3
What We Did (New and Complex) 4
Blooms Taxonomy.....6

Overview

This portfolio was an extension of the previous portfolio. We decided this would be the best route because the frameworks learned after the second portfolio didn't give us enough room or ideas for a portfolio that would have enough to outline what we learned throughout this course. What the pizza application now does is include alot more user data. It saves the user data in terms of signup, login, and orders taken. This is all done on the main screen now and modals helped us out alot in javascript and html in terms of not requiring another window to link to and allowing storing data to be much simpler.

The new framework used for this portfolio was backbone.js. This framework allowed easy to track data for users and that was important since orders and sign up data needed to be saved for the entire process of ordering. Also this data storage and modals made the html part of the project be able to update without reloading the page through the javascript code. THis is important in a shopping cart application because it makes data easier to store since no ajax calls need to be made to other files and also it is more user friendly. The cart updating dynamically also helps with the user and also for a better checkout for someone ordering a pizza quickly.

Outside of these new issues alot of the work consisted of learning how to use javascript efficiently and effectively. The html section of the portfolio was primarily formatting and linking to javascript code through the id's of each section. Also the modals needed to be setup through the html code. Bootstrap helped alot with accomplishing this and made it much easier and cleaner to have the pop up show up from an onClick() event.

What We Did (New and Complex)

In terms of new and complex material in this portfolio, we are integrating a new Javascript framework and a library that the framework depends upon. The new framework used is Backbone.JS. Backbone, similar few to other frameworks, focuses on the modularity of the code by following the MVC pattern (or arguably MV* pattern for Backbone). The following will make it clear on how we utilized this framework. Constructing 'Models', the M in MVC, is very convenient in Backbone. Backbone provides two ways to track data in the application. (1) Model, (2) Collection. Model is similar to javascript's object. Backbone's model can contain multiple fields. We took advantage of this feature to implement separate models for each of the category in our menu, including Pizzas, Sides, Drinks and Desserts. We also have a model for User cart. This model is mapped to specific view components of application. Thus every time user adds an item or attempts to checkout, the model for User's cart is updated. Collection was another aspect we looked into but weren't able to utilize it within our application. It could have been a good solution to cart rendering, however Backbone's collection feature only allows one *type* of model to be stored in a collection. Given that we have multiple models this approach was not appropriate. Implementing 'V' of the MVC pattern was the most complex part of Backbone.JS. After doing some research We found out that Backbone itself does not provide a good templating feature that developers can take advantage of. Instead we have to utilize another library, in our case We used Underscore.JS. Using Underscore.Js's template feature, We implemented the templet for our core content. Upon doing some research, We realized that Underscore is essentially using jquery to insert html content in the dom. However we found this structure of templating a bit misleading and troublesome as opposed to AngularJS which, I thought, has more straightforward syntax for implementing views as in Lab 11. Anyhow, the templet is used only in the initialization process of this application. Once it is initialized, we render the content of template based on user interaction. Moving on to the last part, Controller.

Since Backbone is considered a framework, one would expect Backbone to provide controller's functionality. However we weren't able to find any distinct "Controller" object. Rather what we found was a collection of events. Based on such events, content in Model and View gets updated. Another good feature I discovered while exploring Backbone js was routing. Upon trying few of the examples on Backbone's website, I found it a quite similar to Angular's routing. And since most of the logic of our application is handled by user-triggered events, I decided not to include the routings as it is not necessary for our application. We realize that mastering a framework in a short amount of time is not feasible. Given the resources and time constraint, I think we explored both Backbone.JS and Underscore.JS as much as possible by following their rules and utilizing the few important features that they offer. We gained experience with building modular code. Now that we have our logic separated from data and view, it will be easy to extend this application further if desired. For example, if the owner of the Pizzeria wanted to add a new item to any of the category, all we need to do is add it to the model. Other implementation for controller and view will remain the same. It can easily be tested. Just remove / add and item to any of the category. The view and controller will then adopt to the new data.

Another feature added to this version of application is user log-in. Users are now able to log in. This will ensure that they don't have to go through typing name, address, zip, phone, etc. information every time they make an order. As long as a user is logged in, he/she can add items to the cart and checkout, just within few clicks.

Blooms Taxonomy

This project because of the time constraints required more thought into how efficient the framework we pick needs to be. From the previous portfolios we used frameworks and

languages discussed in class in order to help expand our knowledge in the certain area we were learning in class. In this project instead of restarting and finding a new framework or idea to start from we decided going through the process of making our past application better with a new framework. This framework called backbone helped with development in user data, collections, models, and underscore.

The process of analyzing different parts of the project was simple after we decided to use backbone. These parts were considered to accomplish more in terms of a user friendly website application for ordering pizza. Analyzing what we needed to add to our past portfolio involved going over what went well and what we couldn't accomplish. Those things were checked and then completed for this portfolio including the proper signup login and checkout information. Backbone helped develop the remaining new tasks accomplished in the portfolio. These tasks were assigned after the idea to add onto the past portfolio. What was also changed was the javascript for the categories to be more efficient and run from the new framework. Some experimentation that went on for this was alot of testing since this shopping app was dedicated to one page and one javascript file plus the backbone framework files. This both helped and hurt us because it was easier to save data but once more and more tasks were assigned and implemented the files got bigger and bigger. Some solutions here was proper ID’s for the html and javascript links in order to make everything easier to understand. Also the methods for each section were categorized like the buying, user signup/login, and checkout were done individually in their own sections. The backbone implementation is commented well and there is an overview of the changes given before each section of the application. The primary portion of the javascript is in the home.js file that handles the javascript for the actual webpage and also saves all of the user data.

In terms of evaluating the portfolio work it went through smoothly and efficiently. Due to the nature of dead week, finals, and other labs from class a lot of the ideas that could have been implemented just were not possible with the amount of time left in the semester. Also both of our

knowledge of the new framework only allowed us to implement basic webpage changes to the website but this did not eliminate any old features or any new features we wanted to add to the project. Some of the framework and bootstrap functions and design could have been cleaner but there were different problems with porting javascript over to html correctly or bootstrap would only allow certain methods of taking in data dynamically. This is the reason behind taking out ajax calls and moving to saving the data in javascript to use in different models or orders. JSON files were also an afterthought except for saving sign in data and checking for existing users. The last implementation to evaluate was the underscoreJS portion. This allowed us to change how the pictures were dynamically shown based on the category chosen. It moved most of the html coding towards javascript where we could use the data being stored and set collections to output the pizza menu variables.

Creation for this portfolio was more implement and test as we go simply because of the new framework and the fact we already had a working application that needed fixing and additions to the code. The process for the user data side involved scrapping all the outside web pages and storing all the user account data into the model.customer array for use in the checkout. This is all seen in the drop down for login. The checkout already had dynamic shopping cart totals and order information but what was added was better handling for carryout and delivery. The delivery orders required a logged in user to work and this was the case since a delivery order needs to know an address and phone number of the person being delivered to. In terms of backbone the Backbone.Model.extend method was used to store all of the shopping item information with an object array similar to the associative arrays. This was then implemented into the checkout shopping cart to show totals and order information. In terms of viewing content the html code used in the javascript code along with backbone made for easier formatting. The design of this application was improved greatly from this and made for more of a user friendly application for returning users that know what they want to order and users that order quickly. The bigger portion of this portfolio that changed was construction and formatting

of code in order to make a more efficient and easy to understand version of the application. This allows any additions that would need to be made to this app access to all user data, easy to change design, and shopping cart stability for adding new items.