Eatsafe DC: American University Capstone Project

Giancarlo M. Guerra

02 May 2014

Advisor: Prof. Guy-Alain Amoussou

**Table of Contents**

Introduction 3

Project Definition and Accomplishments 4

Project Description and Implementation 7

Lessons learned 10

Conclusion 11

Bibliography 12

**I. Introduction**

Genetics can sometimes severely hinder the ability of a human being to function and take part of normal life activities. Genetics can render our bodies useless by impeding our physical abilities such as walking or moving our hands. It can also impede our mental abilities by inflicting upon us conditions such as Down Syndrome or Cerebral Palsy. Genetics can also directly affect our ability to consume certain food items: allergies. When people hear the term “allergies”, most think about the seasonal allergies that causes repeated sneezing and runny nose. Most don't recognize that allergies can be something much worse: a food allergy. Although food allergies can be acquired by the repeated exposure or consumption of a particular food item, food allergies are usually inherited from parents, meaning that there is nothing you can do to “cure” it. Allergies can also vary by severity: a symptom can range from a mild stomachache, all the way to ataxia and death. In the case of a severe food allergy, the most effective strategy to cope with the condition is to avoid the allergen altogether. This means avoiding places where the allergen is present in order to reduce as much as possible the trigger to an allergy.

I know how this feels, because I personally suffer from a severe food allergy to shellfish. In other words, anything with a shell that comes from the sea can kill me. For years, I have had to turn down invitations to go to restaurants, or insist that my food is prepared and delivered separate from those of my friends. Also, I have had to internalize the fact that I may not be able to visit some regions of the world, where shellfish consumption is predominant. Living with this allergy has obligated me to compromise between reducing my quality of life and dying. What is worse, I did not ask to be born with this condition, and there exist no treatments or medicines to cure this allergy. What I can do is to not let this allergy determine what I can and can't do. I chose not to let this allergy hinder my life. I chose to enjoy life. That's the mission of this capstone: To allow allergy sufferers to stop living in fear that, if they are not careful of the places they go to eat, they may die.

To that end, the objective of this capstone is to have users be aware of the many restaurant types in DC that can accommodate people with particular food allergies. What helped me overcome my fears associated with my allergy was information. If I have information about restaurants that I can go to and not suffer a reaction, I immediately feel more comfortable. This information includes the restaurant type and the menu and, if necessary, a contact number to inquire about the particular accommodations the venue may have for people with allergens.

This report will first describe the project's inputs and outputs. It will then display test cases and will proceed with a discussion on the design goals met and not met, in addition to discussing work that remains to be done. The report will then proceed to describe the project in technical detail, providing information on the frameworks and programming languages used, in addition to API's and other software. The report will conclude with a personal evaluation of what was accomplished.

**II. Project Definition and Accomplishments**

EatSafe DC is a tool that can be used to find restaurants that are accommodating to a particular allergen or group of allergens. Specifically, the user is asked to input what food group they are allergic to, and EatSafe DC will randomly generate a restaurant that does not serve the selected item, or serves very little of it. The way the application takes input is by way of a check-box. In the current implementation, the user is asked to check off allergies from these three categories: Shellfish, Nuts, and Dairy. These groups were selected because they are the most common and severe food allergies[[1]](#footnote-2). The user makes the selections by way of a check-box. Upon pressing the search button, the application will output several things. First, it will output the name of the restaurant and, if available, a phone number an a website. Eatsafe DC will also output the coordinate location of the restaurant, which allows the Google Maps API to display a map of this restaurant's location. All of this is presented in an attractive user interface which not only makes it obvious the input requested by the user, but also the steps to take after the input has been provided.

The way I decided to test this application was to query it repeated times, and also have users use the application and get their feedback. The following were some of the test cases:

- Checking off the *shellfish* checkbox and hitting the search button, returns *Jalapeño Mexican Grill*

*-* Checking off the *nuts* checkbox and hitting the search button, returns *Tara Thai Restaurant*

*-* Checking off the *dairy* checkbox and hitting the search button, returns *McDonald's Tenleytown*

*-* Checking off both the *shellfish* and the *dairy* checkboxes and hitting the search button returns *Jalapeño Mexican Grill*

- Checking off both the *nuts* and *dairy* checkboxes and hitting the search button returns *Yo! Sushi Bar*.

Although the basic functionality of the project was achieved, there is still work to be done. The following features were achieved: Easy user interface, information gathering, basic database functionality, and successful API calls. Successful applications make it easy not only for the user to gather the information that they desire, they also make the interaction with the web-site easy and obvious. This was achieved by placing all the elements into a single *index.html* page with clearly legible instructions and buttons. When the user first accesses the application, he or she is prompted with a slogan of the application. Immediately the user hovers over that description, he or she is prompted to click on that place. After clicking, the user is taken to the main content area, where all interactions will take place. On the left, the choices for users to check off are clearly labeled, along with instructions on what the user is doing. Once the user finishes checking off the input data, in this case what he or she is allergic to, the user is clearly compelled to hit the search button. Finally, the user is provided with a randomly generated venue which contains all information necessary about the venue, including a telephone number, a website, location and directions.

All the necessary information about the venues was gathered successfully. This information includes names of places, telephone numbers, websites, restaurant types and coordinate locations. This information was made possible to gather thanks to the Factual API[[2]](#footnote-3). The Factual API is a system that provides information about restaurants all over the world. Information about a restaurant's location, name, and contact information is readily available. Because of the API, it was possible to gather restaurant data easily and construct a small data set for testing purposes. Using the Google Maps API, it was very easy to determine the location of a restaurant. Usage of the API immediately increased the usability and attractiveness of the application because it added a new dimension of user interaction: location.

The following features still need to be added to the application: expansion of the database of venues, front-end bug fixes, and user-generated comments. For testing purposes, I decided to test out a small data set of five restaurants in order to determine functionality. For this, I created five records one restaurant each. Each record contains the name of the venue, the telephone number, the website, and the URL that the Google Maps API will use in order to determine the location. In the future, I hope to add to this database to include many of the restaurants in the DC area. In addition to the database, there is a significant bug that I have yet to fix. When the user queries the database or, in this case, presses the *Search* button, the application goes back to the Index page. This forces the user to once more click on the starting div. Despite the fact that the database will update with the information requested, this bug makes the application unattractive and gives it a sloppy character. Finally, I hope to add a comments feature. As usage of the application expands, I would like users to comment on their particular experience at the venue. This will best be done by having comments available. These comments will be indexed in a new table in the database which will be linked uniquely by user and by restaurant. With these details fixed, the application will have extended functionality and attractiveness which will make it a useful tool for people.

**III. Project Description and Implementation**

EatSafe DC is a web-based application which performs a few basic operations. The key back-end operation the application performs is HTTP POST requests to a database which is stored in the server script. Each POST request is, in turn, initialized when the user checks off a particular food item and hits the *Search* button in the front-end. As of the writing of this report, the application will return five different restaurants based on what the user checks off. As described above. When the user hits the search bar, the back-end script will go through a series of if statements to determine which option was chosen(*Shrimp, Nuts, Dairy, or a combination).* After the option has been identified, the script will randomly access one of the database records nested within the if statement. After a record has been selected, each value of the record(*name, telephone number, website, etc.)* is then displayed in the front-end for the user to see.

How is the information actually able to be displayed on the front-end page? In order to achieve this, I used templates. Templates are very similar to traditional html pages, but they differ in the sense that the programmer may designate specific areas of the html that will change given the back-end script. To demonstrate this concept, I will display the contents of my *info* div, which contains the name of the restaurant, the telephone number, and the website:

<div id = "info">

<p> {{rest.name}} </p>

<p> Phone Number: {{rest.Phone}} </p>

<p> Website: {{rest.website}} </p>

</div>

The areas marked in yellow highlighting represent the script that will update based on the information supplied by the back-end. This is what is called the template *placeholder.* For further clarity, I will demonstrate an example of how a record looks:

*{'name': "Jalapeno Mexican Bar and Grill", 'Phone': '704-555-5555', 'website' : 'www.Jalapeno.com', 'url':"*[*https://www.google.com/maps/embed/v1/place&key=AIzaSyA3Ck-0wERwXyfu-MU7MslIXMEThrYEi7A&q=jalapeno+mexican+grill+DC&zoom=14*](https://www.google.com/maps/embed/v1/place&key=AIzaSyA3Ck-0wERwXyfu-MU7MslIXMEThrYEi7A&q=jalapeno+mexican+grill+DC&zoom=14)*"}*

After seeing the record, it becomes clear how the front-end can provide the information necessary to the user. In this particular case, if the random restaurant chosen by the back-end is *Jalapeno,* this record is then stored into a variable called *rest.* After this process concludes, the back-end script calls the *render\_template* function in order to update the front-end. The *render\_template* function looks like this:

*return render\_template("index.html", rest = rest)*

This will generate the index.html page, where all our front-end is located, and will update the highlighted values in the excerpts above by applying the .*name, .Phone,* and *.website* methods to the *rest* variable. The user will recall that in the *rest* variable the chosen record is stored. After this process, the user will see the information posted on the screen.

To implement all of this, I used the Flask[[3]](#footnote-4) micro-framework, which uses the python programming language. Because of the versatility of Python, Flask allows for the fast implementation of small yet scalable applications. That is, Flask is well suited for applications that perform a limited amount of back-end operations. Because Python is an interpreted language, it is inherently slower than other compiled languages such as C, C++ or Java. Thus, using Flask for bigger-sized applications would not be suitable. For our purposes, though, the Flask micro-framework fits perfectly. The following is a directory tree of the typical Flask application:

*Application*

*/run.py*

*/ app*

*/static*

*/templates*

*/views.py*

*/\_\_init\_\_.py*

*/ flask*

*/ tmp*

The *run.py* script will start the server which will host our application. In the *app* directory, you will find subdirectories *static* and *templates,* which contain all *html, css* and *js* which will define our front-end along with any placeholders necessary(see above). The *views.py* script contains all redirections and handles all database calls that forms the backbone of the server side commands such as HTTP POST requests. The *\_\_init\_\_.py* script is called by the *run.py* script to start the server along with the integration of all necessary files such as front-end pages and the *views.py* script. Exiting the *app* directory, the *flask* directory contains all the packages that Flask needs in order to run. The *tmp* directory is there in case a particular function call needs to store any temporary data information. EatSafe DC follows this same structure, but also contains several scripts that hold the database.

**IV. Lessons Learned**

By doing this project, I learned about server-side scripting, API calls, and basic database concepts. I also learned the importance of building an attractive front-end design and how crucial the efficient organization of program code actually is. Prior to this project, I had absolutely no experience building back-end scripts. My experience with coding only limited itself to concepts learned in introductory level Computer Science courses, and front-end development(*html, css*). Doing this project, I learned about HTTP requests. I learned how to emit a POST request in order to query a database to return necessary data. In addition, I had to build scripts that would request data from a specific API and clean it up for use in the application. This was perhaps the most valuable lesson. A lot of applications today are built upon making requests for information from an API. I learned about the Factual API and how it could provide me the restaurant data necessary for my application, and I also learned how to integrate that data in order to request location information from the Google Maps API. This knowledge, along with the newly-acquired knowledge of how to create a database, will serve me well in the future.

Building an attractive front-end is crucial to the success of any application. If the front-end looks unattractive, or if the information presented looks complicated, the amount of users that will utilize your application will be drastically reduced. During the process of building this project, I found myself thinking more and more about how the end-user would feel about using the applications. These thoughs, in turn, forced me to reconsider certain aspects of how the elements of my *html* should be arranged. In the end, this thought process made for an attractive presentation page where instructions are clearly stated, and at the same time provides information in an easy and understandable way.

**V. Conclusion**

If a person has the misfortune of being born with a genetic food allergy, options are severely limited. Since there exists no actual cure to this affliciton, the individual must make extemely difficult life choices that may severely hinder quality of life. This includes restrictions on what venues to visit and even what places to travel to. EatSafe DC's mission is to alleviate as much as possible the degree to which a person must make these life choices. EatSafe DC presents the user a restaurant that is accommodating to a particular food allergy. This, in turn, encourages users to feel more confident about going out and, in turn, improves quality of life. The application was implemented using the Flask micro-framework, which in turn uses the python programming languages. By making POST requests to a select amount of database records, the application generates a randomly selected restaurant that fits the particular allergy of the user. The information is translated to the *html* document by way of template placeholders, which update every time the database is queried.

The process of building this application was not only an extremely pedagogical experience, but it was also a fulfilling one. I learned a lot about back-end scripting and databases. I now feel more confident in my abilities as a programmer. In addition, it gives me great pride that I was able to implement this idea not only for myself as a person allergic to shellfish, but for others like me out there that are afflicted by the same or a similar allergy.

**VI. Bibliography**

1. Allergy Statistics. (2014, January 1). . Retrieved April 28, 2014, from <http://www.aaaai.org/about-the-aaaai/newsroom/allergy-statistics.aspx>

2. Factual Developers Page. (2014, January 1). *Factual API Documentation*. Retrieved March 3, 2014, from http://developer.factual.com/

3. Ronacher, A. (2014, January 1). Flask Developer's Page. *Welcome*. Retrieved March 3, 2014, from <http://flask.pocoo.org/>

4. Grinberg, M. (2012, May 7). The Flask Mega-Tutorial . *- miguelgrinberg.com*. Retrieved March 3, 2014, from http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

1. http://www.aaaai.org/about-the-aaaai/newsroom/allergy-statistics.aspx [↑](#footnote-ref-2)
2. http://developer.factual.com/ [↑](#footnote-ref-3)
3. http://flask.pocoo.org/ [↑](#footnote-ref-4)