Abstract

For my senior design project, I have created a primitive form of a restaurant guide for the University of Pennsylvania surrounding area. I have put together a SQL database of restaurant information, and have succeeded in linking it to a map, taken from Google Maps. Note that this system is up and running, and can be accessed at http://wren.cis.upenn.edu/deg.

In February of this year, Google released a Google Maps related API documentation and Javascript source code to the general population for private use. Commercial use is permitted pending approval by Google, Inc. This allows any user to add code to his or her own webpage that will generate a customized street or satellite map.

My finished product includes a very limited homepage, allowing the user to click one of two links. The first link is to the Restaurants Map, a page that has the Google map with customized icons I will explain in detail later. The Map page also has a filters section at the top. Playing with the filters allows one to see only those restaurants that satisfy certain conditions. Later, I will detail my methods for implementing this feature and the way I designed the maps page.

The second link takes you to the Add New Record page. On this page, you can type in all the known information about an establishment, and the site will automatically post the new restaurant on the map. Ideally, down the line, this page will send the restaurant information to me rather than putting it up in the database right away so that I can approve it first. However, I have yet to implement this functionality. There are a few other security features I want to add as well. As of now, for instance, the files I use for the customized map icons are saved in a highly insecure location. I will get into this in my technical approach. Additionally, I have a comments page for each restaurant where customers can post their comments. My idea in the end is to force people to log in with their Penn user names in order to access this page. That way, the comments will all come from Penn students. As you can see, one of the things I feel makes this project unique is that it is directed at UPenn students and staff. Anyway, read on for the rest.

Related Work

Much work has been done just by outsiders since the time that Google released the API documentation, but in addition, Google itself has a very similar product already available on the web. Google Local allows someone to search for a type of retailer within a certain distance of some known location. Note Figure 1 below for an example of a restaurant-related search on Google Local. In this picture, I have gone to the Google
Local website and searched for restaurants in Philadelphia—in particular, I have pulled up an information window for Ruth’s Chris Steak House to see what it looks like. My system will resemble that of Google Local, but I explain later what makes mine unique.

In addition to this, there are a number of web developers who have already taken advantage of Google’s map API documentation to create their own customized maps. I have included one such application in Figure 2. This shows a customized map with places to buy cars. The user can provide a city name and a desired radius about the city in which he or she would like the locations to be. Here, I have selected Ford Explorers available within 10 miles of Philadelphia. I have clicked on one particular point on the map, representing a car that is being sold right in Philadelphia.

There are a large number of similar sites on the web that utilize Google Maps and its recently published API. The way I see it, Google Local focuses on providing the largest volume of information possible, but the information and functionality on the site is not always the most relevant. For instance, when searching for restaurants on Google Local, there is no option for filtering these restaurants in any way. One simply has to view all of the points, and go through them one by one to find out the aspects of each.

My page is different from Google Local in the relevance of its functionality and different from other customized maps in its ability to cater to Penn students. Suppose it is midnight, and you are stuck in the Towne lab working on a programming assignment.
You want to find restaurants near you that are open late at night. Using Google Local, you have determined restaurants that are near you, but you want a quick way to filter out all of the points that are not open late. Additionally, using customized maps others have made, you cannot find an available system that focuses on the University City area. Thus, you are out of luck. That is where my system comes in. My map page has a quick and easy way to filter out all of the restaurants that do not meet specific criteria, such as being open on a specific day at a specific time. Finally, my system is unique in that it includes a comments section for visitors to post their own input about restaurants. Google Local does not have such a feature (at least not that I have been able to locate).

Figure 2: A currently existing Google map mashup linking automobile data to a map of the US.

**Technical Approach**

This is the section of my paper that has changed drastically since the last time you read it. In my original project, I was implementing my site with an XML file for the data and javascript for the rest. Since then, I have switched over to a SQL Server database with a combination of ASP.NET and C-sharp code for the actively updating pages. Additionally, I retain some HTML and Javascript. In this section, I will explain in detail how each of my pages is implemented and why I chose the particular methods that I did.

*Part I: The Map Page*
I will first explain a few things about the map itself, but the important part of this page is the map overlays (the icons plus their HTML windows) and the implementation of the filters section. Essentially, the map is placed inside its own HTML <div> element at the bottom of the page, and it is created using the following Javascript code:

```javascript
var map = new GMap(document.getElementById("map"));
map.centerAndZoom(new GPoint(-75.19505, 39.952032), 2);
map.setMapType(G_MAP_TYPE);
```

There are couple things of note about this code. First of all, it uses functions that are defined in Google’s Javascript file, which is publicly accessible provided you sign up for a Google Maps Key (that is easy to do at their website). Then, you simply have to include a section of code given to you by Google in the header of the page—this section includes your Maps Key. There is not much to say about the code itself. It simply initializes a new map, centers it appropriately on Penn’s campus with the desired zoom level, and sets the type. The three Google map types are Normal, Satellite, and Hybrid. I use Normal in my page.

I use several steps to set up the map overlays. First and foremost, when the page loads, I open a connection to the Restaurants table in my database, grab all of the records, and store them in a table. Then, I go through the table row by row in a for loop. For each individual row, I test whether the row satisfies whichever filter values are set. If it does (that is, it is supposed to be displayed on the map), then I append the relevant Javascript code to place that restaurant on the map using an icon that is grabbed from the database. I also add an HTML window to each restaurant, which simply means that when a user clicks on the restaurant location, a window pops up with certain HTML code. Note that all information needed to display the restaurant item on the map such as latitude and longitude is stored in my database. I will get into how this happens in the section later about the Add New Record page.

In Figure 3, I show the maps page the way it initially loads. When it first loads, note that all filters are set to their default values. If the user selects different filter values, and then clicks the Submit All Filters button, the page posts back the information and reloads, causing each restaurant to be checked against the new filters. In figure 4 below, I have clicked on one of the overlays to demonstrate the information window generated on this event.

The only information I use on this page that is not directly grabbed from the database is regarding the icons. In my database, I have a field that lets me know whether the restaurant uploaded a custom icon or not. If there is a custom icon, I will have saved it in a folder to which I have write access. Thus, I simply let the map know that it is to grab the icon for that restaurant from the folder. If there is no custom icon, I use a default icon that I have saved in my web folder. In the HTML window that comes up on a user click, there is a link to a details page about the restaurant. In my next section, I will explain how I implemented the details page.

Part II: The Restaurant Details Page

For the restaurant details page, once again the first thing I do is connect to the database. Here, though, instead of simply grabbing information from my Restaurants
Figure 3. An image of the way the map initially loads. The restaurants are on the map viewable by icon.

table, I also need to grab information from my time/integer table. The time/integer table is something I came up with to help me keep track of opening and closing times for the restaurants. In my Restaurants table, I have fourteen total time fields (two for each day of the week, one for the time the restaurant opens and one for the time it closes). Each of these time fields is an integer between 0 and 95 that lets me know an actual time of day. I use the time/integer table as a decoder to go from integers to times and vice versa where needed. For instance, on the restaurant details page, I want to grab the opening and closing times for this restaurant and turn them into readable times, so I use the time/integer table for that. However, on the maps page, I want to take the time the user inputs in the filter section and turn that back into an integer for comparison purposes against the opening and closing times in the database. Figure 5 shows a screenshot of a details page for one of the restaurants.

On the details page are several links. One of these is to a comments page; this, I will discuss later. Next, however, I would like to describe the link I have to a menu if available.

Part III: The Handler Page

When a user clicks on the link to attempt to view a restaurant’s menu, I do not actually redirect the user to a unique page for each restaurant. Rather, I direct the user to a handler page that takes in as parameters the menu file type and the restaurant ID. The
restaurant ID is simply the primary key of the Restaurants table—it is a unique identifier that each restaurant has. I use this as a parameter in the details page as well to determine which information to post. In the handler page, I also take in the menu file type. This has a chance of either being a PDF or an image. In either case, what I have done when adding this restaurant is saved the bytes of the menu file right into the database. Then, in the handler, I grab the bytes from the database, check the parameter to see if it is in PDF or image format, and appropriately display the file. This is not too difficult to implement once I know the file type.

Something to note here is that initially, I was using this implementation as well for the map icons. That is, I would save the bytes of the icon file right into the database, and whenever I needed to access the icon, I would grab it from the handler page. However, this proved too monumental of a task for the Google maps to have to reload the handler page every single time it posts a restaurant on the map (dozens of times per map load). It resulted in the restaurants map page requiring upwards of 45 seconds to load each time. I decided this was far too much time and switched to simply saving the icon files in a folder as I described above. The reason I do not use this implementation for menus as well is that the folder to which I save the map icons is very insecure. In fact, it is readable and writeable by anybody—if I am to seriously put this website into use, I will gain access to a different folder that is more secure. For this reason, though, I
wanted to do as much as I could without using this folder, so I used the other method for menus.

**Part III: The Comments Page**

In Figure 6, I show a screenshot of my restaurant comments page. In the restaurant details page discussed above, I have a link that allows a user to post comments regarding his or her own personal experience at the given restaurant. Following this link takes one to the comments page for that restaurant. Like the details and handler pages, the comments page uses a parameter that signifies the restaurant ID. I have a comments table in my database separate from the Restaurants table. The comments table has a foreign key restaurant ID, which is the primary key in the Restaurants table. It also has the comment represented as a string as well as the date and time the comment was posted. To load this page, I simply connect to the comments database, grab all comments with the appropriate restaurant ID, and post them on the page. When the user posts back a comment, I connect to the database and add that comment in and then reload the page. Below is a screenshot of the comments page for a sample restaurant. At the time of
Figure 6. A comments page for one of the restaurants. There is not much to it; it simply displays all comments with their date and time, and posts back and comments that are written.

publication of this paper, the comments page does not require a Pennkey login, but I expect it to once I get in touch with the right people.

Part IV: The Add New Record Page

This was by far the most in-depth and difficult to implement page of all the pages I created. Figure 7 shows a screenshot of the page. When a user clicks the Submit button, the first thing I do is validate the form. I created my own validate function to suit my personal requirements. Thus, if the name or address fields are left blank I instruct the restaurant to go back and fill these in. Also, there are other things that will cause an invalid form warning such as failure to select any methods of payment or selecting Closed for every day of the week.

Once I am satisfied that the form is valid, I connect to the database and begin adding values. I insert the restaurant name, address, phone number, and description as is right away. The same goes for cuisine type. I use a geocoder web service to grab the address and convert it into latitude and longitude values. Then, I insert the latitude and longitude into the database. This will be used when adding the restaurant to the map. I put together the selected methods of payment into one long string, and insert that into the database. I also grab all of the time values for opening and closing for database insertion. If a restaurant is closed on a particular day, I signify that by setting a bit field in my
Restaurants table to 1 and setting open and close values for that day to 1. Finally, if the restaurant has selected to upload a customized icon, I save the icon to my appropriate folder. If the establishment has uploaded a menu, I run a function to test whether the menu is an image or a pdf, set this information in the database, and save the file bytes directly into the database. I have two additional fields to set in the database: one for whether or not the restaurant accepts cash, and one for whether it accepts any type of credit card. These fields are useful to me later in my map filters implementation.

Final Thoughts on Technical Approach

All of the database interaction I did was done using a combination of SQL text and C-sharp code, all of which is a bit unwieldy but any of which I am happy to discuss with anyone who is interested. I used HTML to format the pages and some ASP.NET, however most of the ASP.NET code was actually generated for me using Visual Studio.

I tried to handle for every possible exception I could think of, and also for things a user might do that could mess up the functionality of the website. For instance, if someone adding their restaurant to the database fails to add an icon, I will default to an icon I have saved. If they do not add a menu, then I cause the handler page to indicate that the establishment has yet to upload a menu. When they decide to add the menu, I can add it to the database fairly easily. Also, if a user is at the handler, comments, or details page and tries to change the restaurant ID parameter to something invalid, I catch that and send a message that the restaurant ID was not found in the database. I accounted for almost every possible error I could think of.

Finally, figure 8 is a block diagram of the interaction of each of my pages with the database. Note that the comments page is the only page that both sends information to the database and grabs information from it. The Add New Record page sends its information to the database but never actually queries the database, while the restaurant details and map pages grab a lot of information from the database but never send anything to it.

The only potentially counter-intuitive implementation in my system is the open and closing hours (using a separate table for these), but I think it is helpful if not necessary when I try to implement the filters on the map. The rest of the ways I did things are probably as expected.

Conclusion

To this point, I have accomplished what I set out to do for the second semester. At the end of the first half of this project, I had a working XML/ Javascript version of the site, but it did not involve a database nor did it have many of the other features I wanted to add such as menus and customized icons. This semester, I learned C sharp and strengthened my knowledge of HTML. I also picked up SQL along the way and learned all the aspects of database programming. Using Microsoft Visual Studio, I have taken advantage of the use of active server pages to make my pages change based on various details.

The system is fully functioning now and one can add new restaurants, comment on existing restaurants, and view details for restaurants. The map works the way I had
envisioned and I have icon and menu capabilities. I very much enjoyed working on this assignment and I am glad I had the opportunity to learn these topics, because I think they will be very useful for me in the future.

I felt that the most difficult part of the project was initially learning the set of skills I would need for database programming. Once I got past that initial learning curve, the rest seemed to fall into place because there is so much you can do with this stuff once you get the hang of it. The most challenging part of my system was far and away the adding new records page. That page has to send the most information over to the database, and there was the potential for many bugs. However, it is arguably the most important page to my system in that it gives me a user-friendly form for updating the database. Updating by hand would be difficult mostly because I would not quite know how to manually add a menu, as it needs to be in bytes format.

In all, I believe I accomplished just about everything I set out to in this assignment and I felt it was an excellent experience. Hopefully, I can continue to add features to the system after I graduate and I eventually intend to market it to the campus foods website or to Penn itself.

Figure 7. The Add New Record Page. This is used for adding a new restaurant to the restaurants table and in turn putting it on the map.
Figure 8. Block diagram of database interaction with each of the parts of my system.
Works Cited


