Project 3: Google Mashup

due Monday, 27 April 2009, noon ET

Goals.

- Implement a mashup!
- Leverage Ajax and RSS.
- Read the news.

Recommended Reading.

- `http://www.w3schools.com/xml/`
- `http://www.w3schools.com/dom/`
- `http://www.w3schools.com/rss/`
- `http://www.w3schools.com/js/`
- `http://www.w3schools.com/ajax/`

- Google Map API Concepts
  http://code.google.com/apis/maps/documentation/

- Google Map API Reference
  http://code.google.com/apis/maps/documentation/reference.html

- RSS 2.0 Specification
  http://cyber.law.harvard.edu/rss/rss.html

- Using PHP/MySQL with Google Maps
  http://code.google.com/support/bin/answer.py?answer=65622&topic=11364

- Using Debugging Tools with the Google Maps API
  http://code.google.com/support/bin/answer.py?answer=87133&topic=11364

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (*e.g.*, by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. If in doubt as to the appropriateness of some discussion, contact the staff.

You may even turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly.

Grades.

Your code (CSS, JavaScript, PHP, SQL, XHTML, *etc*.) will be evaluated along the following axes.

Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

panel.cs75.net.

☐ Surf on over to `http://panel.cs75.net/`, log in, and follow the link to **Subdomain Management** under **Your Account**. On the page that appears, add a subdomain called **project3** to your domain. Not only will the panel automatically create A records for `project3.domain.tld` and `www.project3.domain.tld` for you, where domain.tld is your domain, it will also create a subdirectory called `project3` in your `public_html` directory. All of your work for this project must ultimately reside within that subdirectory.

☐ Return to your panel's home page and select **Password Protected Directories**, also under **Your Account**. On the page that appears, click **Find a Directory to Password Protect**. You should then see the contents of your `public_html` directory. In the row containing your `project3` subdirectory, click **Protect** in the column labeled **Action**. On the page that appears, check the box next to **Protection Enabled**, and then fill out the rest of the form. Any username and password are fine, as is any value for **Protected Directory Prompt**, which is just an aesthetic detail. When done, click **Save**. The panel will proceed to create (or edit) at least two files for you: `~/public_html/project3/.htaccess`, which tells Apache to password-protect the directory, and `~/domains/domain.tld/.htpasswd/public_html/project3/.htpasswd`, which contains your choice of username and password, separated by a colon, with the latter encrypted. Had you not a convenient panel, you could also have just created files like those by hand.

☐ Return to your panel's home page and select **MySQL Management**, also under **Your Account**. On the page that appears, click **Create a new Database**. Provide a name, username, and password for your database, then click **Create**. Make note of all details on the page that appears! You must use that database for this project.

project3.domain.tld.

☐ Your mission for this problem set is to implement a mashup that integrates Google Maps with Google News with a MySQL database containing 42,073 unique zip codes. But first, some inspiration!

If you've a friend who owns a Nintendo Wii (that's connected to the Internet), see if you can invite yourself over there to do some "research." Ask your friend to start up the **News Channel**, then click **National News** with the Wiimote, then click any of the articles, then click **Globe** in the screen's bottom-right corner. Notice how you can spin the Earth by clicking **A** and dragging, thereby revealing stacks of articles from different cities and geographic areas. And by zooming in and out with **+** and **–**, you can reveal more or fewer stacks. By clicking a stack's icon, you can then read local news.

If you haven't said friend, watch this instead, paying close attention between 01:04 and 01:50:

`http://www.youtube.com/watch?v=uO6J8ryTKYk`

The challenge ahead isn't to implement precisely that interface but the spirit thereof in the (largely) two-dimensional world of Google Maps! Specifically, your mashup will present users with

a Google Map, next to which will be a form. Upon submitting an address via that form, users will be whisked away to that location on the map, which will be sprinkled with markers representing news articles pertaining to the area. Clicking a marker will trigger a balloon to appear, inside of which will be links to those articles.

By problem set's end, then, you'll have a tool whose URL you can share with family and friends back home (that they might actually find useful)!

Alright, where to begin?

☐ Surf on over to Google Maps at `http://maps.google.com/`. Input something like **02138** into the page's form and click **Search Maps**. You should find yourself whisked away to a familiar location. Well that was easy. Notice, though, that the page's URL did not change. I wonder how they did that![1]

Now input **28.42, -81.58** instead. Perhaps you'd rather be there?

It looks like Google Maps understands zip codes as well as longitude and latitude. Interesting...

If unfamiliar with longitude and latitude, feel free to read up, albeit in more detail than is necessary for this problem set:

```
http://en.wikipedia.org/wiki/Longitude
http://en.wikipedia.org/wiki/Latitude
```

☐ Now surf on over to Google News at `http://news.google.com/`. Click the link to **Advanced news search** at top-right, and notice how you can **Return only articles about a local area**. Input a zip code like **02138**, and you should see news local to Cambridge. Notice the link to **RSS** on the page's left-hand side. Click it, and you should find yourself at a URL like the below (which, unfortunately, wraps on to two lines):

```
http://news.google.com/news?svnum=10&as_scoring=r&ned=us&as_drrb=q&as_qdr=
&as_mind=22&as_minm=10&as_maxd=21&as_maxm=11&geo=02138&aq=f&nolr=1&output=rss
```

Within that otherwise cryptic string should be a familiar value (*i.e.*, **02138**). Hm, that could be useful...

☐ Alright, clearly there's a way to whisk a user away to a particular point on a map, using zip codes or longitude and latitude. But Google News expects only zip codes, so how to find zip codes that are proximal to another? It'd be nice if we had a big list...

Let me Google that for you:

```
http://letmegooglethatforyou.com/?q=zip+code+database+list
```

[1] Cough cough, Ajax.

Among the results should be a link to `zip-codes.com`. Head to this page in particular:

`http://www.zip-codes.com/zip-code-database.asp`

Look how much data you can get for $79.95.[2] Don't worry, we'll pick up the tab. Notice, though, that the data comes in multiple formats, among them CSV.[3] That's perfect, because we can easily parse that programmatically![4]

To get a sense of what we bought, download this sample:

`http://www.zip-codes.com/files/sample_database/zip-codes-database-DELUXE-SAMPLE.zip`

Inside that archive, you'll find a file called `zip-codes-database-DELUXE-SAMPLE.csv` (among others). Go ahead and open it with Excel (or any old text editor). Notice that the database contains 49 fields (*i.e.*, columns), each of which is defined for you in the CSV file's first row. If you open up `ZipCodeDatabaseSpecifications-Deluxe.pdf` from that same archive, you'll find that pages 2 and 3 elaborate on those fields' types. That same PDF, for reference, is available at the URL below.

`http://www.zip-codes.com/files/documents/ZipCodeDatabaseSpecifications-Deluxe.pdf`

You're welcome to examine the actual CSV file that we bought, but you'll find that its 79,960 rows won't fit in older versions of Excel:[5]

`http://www.cs75.net/projects/3/zip-codes-database-DELUXE.csv`

By default, incidentally, Excel doesn't show leading zeroes in CSV files. And so some of the zip codes in these CSV files' first columns might appear to be fewer than five digits, even though they are not. If you open those same files with any old text editor, you'll see leading zeroes!

☐ So, thanks to its 79,960 rows and 49 columns, the CSV file we bought is 28 MB. Yet most of those fields you won't even need. Interesting though it may be to know how many Hawaiians lived in 02138 as of the 2000 Census, you don't really need to know that for this problem set.[6] You're only going to waste time later on if you have to sift through so many unneeded bytes while searching for zip codes. So let's "pre-process" the data.

[2] Why go Standard when you can have Deluxe? Actually, we wanted population data, so we got upsold.
[3] `http://en.wikipedia.org/wiki/Comma-separated_values`
[4] Keep an eye out for such things in the future!
[5] Someone decided to limit Excel to 2^{16} = 65,536 rows (and 2^8 = 256 columns)!
[6] But the answer is 25!

Write, in a file called `import` in `~/public_html/project3/`, a "script" (*i.e.*, an interpreted program) that imports these fields (and only these fields) into a MySQL database:[7]

i. `ZipCode`
ii. `Population`
iii. `Latitude`
iv. `Longitude`
v. `State`
vi. `City`

The table into which you import these fields must be called `zips`, and it must live in the database that you created for this project.

We leave the design of `zips` up to you, but you might find some good hints in that PDF from `zip-codes.com`. Take care not to allocate more space than you need to for fields. And take care to define, at least, a primary key. Realize, though, that the CSV file actually contains some zip codes multiple times, each of whose rows might have identical values for `Latitude` and `Longitude` but different values for, say, `CityAliasName`. (The file contains 42,073 unique zip codes.) Do not insert any zip code into your table multiple times. We leave it to you to figure out how.

Once you have a schema (*i.e.*, design) in mind, you'll, of course, need to `CREATE` the table, certainly before you can `INSERT` any rows into to with your script. You are welcome to `CREATE` the table "at runtime" via a call to `mysql_query` in your own script or in advance via phpMyAdmin.

You'll probably want to use phpMyAdmin quite a bite while developing your script. Odds are you'll make a mistake at least once and want to `ALTER` or `DELETE` rows from your table so as to start fresh. You'll likely find phpMyAdmin's **Browse**, **Structure**, **SQL**, **Empty**, and/or **Drop** tabs of particular help.

Note, incidentally, that we did not ask you to name your script `import.php`. Whereas webservers generally require that PHP scripts' names end in `.php` for execution, your shell (*i.e.*, your prompt) only requires that they start with a "shebang," `#` followed by `!`, followed by the full path to `php` (PHP's interpreter).

How to find that full path? Execute the below on `cs75.net`.

```
which php
```

[7] Be sure to import the fields with precisely these names, as defined by the CSV file's first row. If you find, while implementing your mashup, that you would like to make use of other fields from the CSV file, you may modify `import` and `zips` to accommodate. For space's sake, though, do not import more fields than you actually plan to use.

Aha! You should see that `php` lives in `/usr/local/bin/php` on `cs75.net`. Go ahead, then, and put precisely this line atop `import`:[8]

```
#!/usr/local/bin/php
```

Take care not to put any characters at all (even whitespace) before this shebang. Even with the shebang present, you still need to tell `php` to interpret what follows as actual code (rather than just text, a la XHTML, that should be outputted raw). The second line of your file should thus be:

```
<?php
```

or

```
<?
```

And your last line should be:

```
?>
```

It's everything in between that we now leave to you. Implement `import`! If implementing this project on your own machine, though, best to download your own copy of that CSV so that you're not downloading 79,960 rows from `cs75.net` every time you try running your script!

Once done, you should be able to populate that table called `zips` in your database by executing, quite simply, the command below after `chmod`'ing `import` `700`:

```
import /path/to/zip-codes-database-DELUXE.csv
```

As this usage implies, you should accept one (and only one) command-line argument: the path to a file to import. You might thus want to read up on, at least, `$argv`:

```
http://us3.php.net/manual/en/reserved.variables.argv.php
```

Incidentally, you might want to try importing a much smaller file than ours first, lest you fill your table with thousands of mistakes. Just fill a text file with some values and commas!

Once you've imported all 42,073 unique zip codes from `zip-codes-database-DELUXE.csv`, leave `zips` alone. You'll need it later.

[8] Alternatively, you could omit the shebang and execute `import` with `` `/usr/local/bin/php import` ``. In fact, if developing `import` on your own Windows machine, you'll need to do just, instead executing your script with, *e.g.*, `` `C:\xampp\php\php.exe import` ``.

☐ Okay, it's now time to turn our (well, your) attention to the Google Maps API (Application Programming Interface), which "lets you embed Google Maps in your own web pages with JavaScript," provided you've signed up for an "API key." Head to

`http://code.google.com/apis/maps/`

and follow the link to **Sign up for a Google Maps API key**. When asked for **My web site URL**, provide

`http://project3.domain.tld/`

as your answer and then click the button labeled **Generate API Key**.[9] On the page that appears next, you should be thanked and informed of your key. Remember that key! Well, don't actually try to memorize it. But copy and paste it somewhere safe.[10]

☐ Next head to

`http://code.google.com/apis/maps/documentation/`

and follow the link at top-left to **Basics** under **Developer Guide**. Go ahead and copy the XHTML for "The 'Hello World' of Google Maps" to your clipboard and paste it into a file called `index.html` in `~/public_html/project3/`, setting its permissions appropriately and replacing the default value of `key` (`abcdefg`) with your own key and the value of `sensor` with `false`. Then visit the URL below.

`http://project3.domain.tld/`

☐ Okay, at this point you should have a little baby map embedded in an otherwise unremarkable page. But, hey, you can at least drag it around. Let's learn how to make it more interesting. Head back to

`http://code.google.com/apis/maps/documentation/introduction.html`

and read the whole page! You'll be introduced to not only to cartography but also to "info windows." Oh and by "read" we mean actually read! And don't simply play with the examples; view each's source!

If you've any questions about what you've read, simply ask via the course's bulletin board!

Before we move on, let's have you apply what you just learned by making your own map more interesting. It's fine, incidentally, to leave all of your JavaScript in `index.html` or relocate it to some external `.js` file.

[9] If implementing the project on your own machine, you may also want to obtain a key for `http://localhost/`. Just be sure that the code you submit contains the key for your actual domain!

[10] If you nonetheless lose track of your key, you can re-generate it via this same process. But it's not nearly as much fun the second time around.

Center your map on your hometown instead of Palo Alto. And feel free to alter your map's default zoom level at the same time. If you don't know any longitudes or latitudes offhand, follow the directions in **Section 3** of this tutorial to find some using Google Maps itself:

```
http://code.google.com/support/bin/answer.py?answer=74725&topic=11364
```

Plant an info window at the place you grew up and fill it with some XHTML that includes both some text (*e.g.*, your name) and at least one clickable link (to, *e.g.*, your hometown's website).

Feel free, too, to make your map bigger or perhaps center it in your page. Perhaps supplement the map with some logo or text.

☐ Head now to

```
http://code.google.com/apis/maps/documentation/events.html
```

to introduce yourself to "events," "listeners," and "closures." As before, play with and read through each and every example!

☐ Now head to

```
http://code.google.com/apis/maps/documentation/controls.html
```

and read up on "controls" and "types." Once you have a sense of what's possible, add one or more controls to your map and feel free to alter your map's type.

☐ Now teach yourself about "overlays" and "markers" by heading to:

```
http://code.google.com/apis/maps/documentation/overlays.html
```

You can stop reading once you hit "polylines," but don't let us stop you if you'd like to read on.

☐ Almost done with the reading! Head to

```
http://code.google.com/apis/maps/documentation/services.html
```

for an introduction to "services" and "geocoding" specifically. You can stop once you hit **Using Street View Objects**, but, again, feel free to read on.

It's now time to enhance your own map once more. Add to your map a text field and button. When the latter is clicked, the user should be whisked away (if possible) to whatever location has been typed in the former. There's no need to plant a marker or info window at that spot, but you're welcome to do so. You might find the example called `geocoding-simple.html` of particular help.

☐ Take note of, but don't (yet) read in its entirety, the API Reference for Google Maps:

`http://code.google.com/apis/maps/documentation/reference.html`

Overwhelming though it may seem at first glance, there's a lot of good stuff there, as that page defines every one of the API's classes, types, and functions. It's not terribly easy to find things there, though, so ctrl-f and the three columns of links atop the page may prove your best friends.

☐ And now it's time to bring (well, mash) all this together. Enhance your map in such a way that when the user searches for a location (via that same text field and button you just added), not only is he or she whisked away to that location (if possible), the resulting map is also sprinkled with up to $n \leq 5$ markers representing the n largest cities within view. Each city's marker, when clicked, should trigger an info window to appear containing clickable links (that open in new windows) to a few news articles pertinent to the city.

Those articles' titles and links, of course, should be culled via RSS from Google News based on the zip code(s) within view.

How to make all of this happen? Well, your map already knows how to whisk users away to some location provided via that text field. It sounds like, once done whisking, your map will need to go GET, wink wink, some articles. Because of "same-origin policies," your JavaScript code cannot contact Google News directly.[11] But it can contact, say, a PHP file that lives on `cs75.net` (where your JavaScript came from). Perhaps you could write a PHP script (whose name ends in `.php`) that accepts some HTTP parameters (say, the coordinates of a map's southwest and northeast corners), figures out which zip codes live within the rectangle defined by those corners, and retrieves RSS news feeds for $n \leq 5$ of those zip codes? Think of this PHP script as your JavaScript's proxy to Google News. Of course, some cities have multiple zip codes, and you need to fetch news from the n largest cities (not zip codes) within view, so you might need to sum a few zip codes' populations. But that should be doable, perhaps via `GROUP BY` and `SUM` in SQL or via simple arithmetic in PHP.

How to contact that PHP file via JavaScript? Well, you might want to re-examine `xhr-requests.html`, which we're certain you looked at while reading up on this API's services:

`http://code.google.com/apis/maps/documentation/services.html#XML_Requests`

Note that the example relies on `data.xml`, whose structure you can see here:

`http://code.google.com/apis/maps/documentation/examples/include/data.xml`

[11] http://en.wikipedia.org/wiki/Same_origin_policy

You won't, of course, be downloading a static XML file (as the example does `data.xml`), but surely your PHP script could spit out some XML that you yourself have designed? Incidentally, so that your PHP code does not spit out a "MIME type" of `text/html`, as it does by default, you'll likely want to include a line like the below atop your PHP file:

```
header("Content-type: text/xml");
```

How to retrieve RSS from Google News and return just some articles and URLs in your own XML format? Well, this article you might like:

```
http://www.ibm.com/developerworks/library/x-simplexml.html
```

Odds are, you'll need to generate XML that has a bit more structure (*i.e.*, nesting) than that example's `data.xml`. After all, your PHP script will need to return multiple articles (*i.e.*, titles and URLs) for multiple cities.

How to navigate your PHP script's XML (or, really, DOM) in JavaScript once returned to the browser? You might find that this example gets you started:

```
http://www.captain.at/howto-ajax-process-xml.php
```

And here's an example that is similar in spirit but also discusses the PHP side:

```
http://www.w3schools.com/php/php_ajax_xml.asp
```

Alternatively, instead of XML, you are welcome to use JSON!

☐ Once you've gotten your map working, go back and ensure that it responds to events like `dragend`, `resize`, and `zoomend` by clearing any and all markers and then re-laying up to 5 representing the new area's news. (Beware responding to `moveend`! Why?) You may want to re-visit the section on event listeners over here:

```
http://code.google.com/apis/maps/documentation/events.html#Event_Listeners
```

☐ Next make sure that your map grows to fill a user's window, while still leaving room for your form, much like Google Maps itself. You'll find that growing and shrinking your map horizontally is easy with XHTML and/or CSS alone; vertically will require some JavaScript! Take care, as does Google Maps itself, to hide any scrollbars.

☐ As always, your website must somehow handle all possible errors (*e.g.*, invalid zip codes, zero articles, *etc.*) gracefully. Under no circumstances should users encounter JavaScript runtime errors.

☐ And now for some technical requirements.

- ☐ Your site must live at `http://project3.domain.tld/`, where `domain.tld` is your own domain.
- ☐ All PHP files must be chmod'd 600.
- ☐ Your XHTML must be valid (or "tentatively" valid), unless some feature of your site requires otherwise (for the sake of some browser); explain in XHTML comments any intentional invalidities. Your XHTML should also be as pretty-printed as possible. Your CSS need not be valid.
- ☐ Your JavaScript and PHP must be extensively commented and be as pretty-printed as possible.
- ☐ You may use a WYSIWYG editor to generate XHTML and/or CSS that you would like to use in your site.
- ☐ If you incorporate or adapt snippets of code from the Web into your project, cite the code's origins with comments.
- ☐ If you incorporate images from the Web into your project, cite the images' origins with XHTML comments.
- ☐ Your website must appear and behave the same on at least two major browsers, namely:
  - ☐ Chrome 0.x
  - ☐ Firefox 3.x
  - ☐ Internet Explorer 7.x
  - ☐ Opera 9.x
  - ☐ Safari 3.x

Submission.

☐ Once done with your site, put together a readme at `http://project3.domain.tld/readme/`. Treat this readme as your opportunity not only to explain but to justify your design decisions. Tell us with which two (or more) browsers we should evaluate your site. And give us an overall sense of how your site works (*e.g.*, tell us which files do what). But still be succinct; keep this readme to just a few paragraphs in length.

☐ By this project's deadline, submit your website by SSHing to your domain and executing the command below in order to submit the contents of your `~/public_html/project3/` directory.

```
submit project 3
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You may re-submit as many times as you'd like, but take care not to re-submit after the project's deadline, lest your work be deemed late.