Carnegie Mellon MISM GIS Project for MIWatch.org—Technical Documentation

Technologies Used

PHP

PHP was used on the backend. The only additional code we used was the PEAR::HTTP_Request module. See the Architecture section below for details on this; it is possible to deploy the application without using this module.

MySQL

MySQL was used as the database server.

Google Maps

We used the Google Maps API to both geocode the locations and then render the maps with markers. An API key is required to use the Google Maps API. You can set yours in the application in `/php/init.php`.

jQuery

jQuery is a popular cutting-edge JavaScript abstraction library. This made it easy to write clean code for the additional JavaScript we needed. We load jQuery at the same time Google Maps is loaded as part of the larger Google AJAX Libraries API.

YUI (Yahoo! User Interface Library)

We took advantage of YUI's design pattern library for the grid layout and basic styles that come with it.

File Structure

The root directory is used as the webroot. The main index page and the data-retrieving PHP file are included in this directory. Within that directory there are couple of other directories:

/php

This folder contains two files: `init.php` is the first file loaded and contains configuration information, as well as some helper functions used throughout the other pages/files. `dbconf.inc` contains connection information for the MySQL database.

/style

This folder contains `style.css` with our own styles. Other CSS files are loaded directly from YUI.

/support

This folder contains:

- README as both Textile and PDF
- `database-blank.sql`: A **structure only** dump of the application database, without any data.
- `database-populated.sql`: A full dump of the application database, with data.

It is **highly recommended** to remove this directory before deploying the application on a public web server.

Database Population

Add information here.

Architecture and Data Flow

`index.php`

Everything is loaded by `index.php`:

- `init.php`: this pulls in the database configuration (`dbconf.inc`), makes sure the PEAR::HTTP_Request module is present, sets the proper API key for Google API usage, and includes several helper functions used throughout the application.
- YUI stylesheets directly from Google's content distribution network (so we don't have to maintain them ourselves).
- `/style/style.css`: our own stylesheet, taking care of custom UI elements that the YUI stylesheets don't cover.
- Google AJAX Libraries API directly from Google. This allows us to later load jQuery and the Google Maps API.

Upon load of the `index.php` page, the following things happen in JavaScript:

- The Google Map is initialized, the center set, and normal map controls added.
- An event handler is attached to the category selection checkboxes. When they are changed, this function will be called. It does the following:
  - Collects the category filters and constructs a string of parameters for the HTTP GET request that will occur later.
  - Calls the JavaScript function `populateMap()`, which reloads the map with the new parameters.
- The event handler described above is triggered in order to populate the map initially.

The `populateMap()` function pulls the data for markers from PHP/MySQL, and places them on the map. On a detailed level, it does the following:

- Shows the previously-hidden `#spinner` DIV, which contains the loading message.
- Clears the map of any previously-added markers and the sidebar of any existing entries.
- Calls the Google Maps JavaScript function `DownloadUrl()`, which does a GET request with the given URL (our `data.php` file, with our parameters from earlier added), then upon completion feeds the returned data (which is XML in our case) into a callback function (described next).
- The callback function takes the XML data returned from `data.php` and:
  - Gets all the location markers.
  - For each marker, creates a marker, adds it to the map, creates a sidebar entry, adds it to the sidebar.
- Hides the `#spinner` DIV.

`data.php`

This file essentially gets locations from the database and builds an XML document of the markers.

If the geographic coordinates (latitude and longitude) are not already known, a request will be made to the Google Maps geocoding service to get coordinates based on the address (which we always know). This set of coordinates, along with a status code and accuracy indicator from the geocoder, is then stored in the database for future usage so that we don't have to go to the geocoder every time.

The database query to get the locations will be customized based on the parameters passed in the GET request. This is how filtering by category is accomplished.

A mechanism is built into the file to prevent overuse of the Google Maps geocoder. If the geocoder returns a status code indicating that too many requests have been made, an increasing delay interval will be used between each geocode request.

Use of PEAR::HTTP_Request

The PEAR HTTP Request module is used to make the GET request to the Google Maps geocoder. As described above, these requests are **only** made when the geographic coordinates are not already known. For the initial state of the database, we have already done the geocoding for all existing points. **Therefore, it is possible to deploy and use the application without the PEAR HTTP Request module.** However, if new locations are ever added to the database table, they will need to be geocoded in order to appear on the map, and in order to do that the PEAR HTTP Request module must be available. If you would like to temporarily disable the requirement for the HTTP Request module, the place to do it is `init.php`.