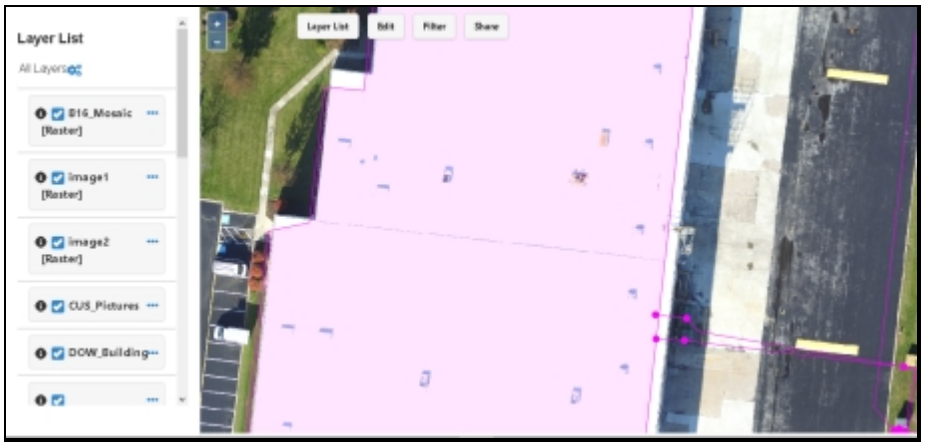
Opensource GIS Program (Prototype)

Table of contents

Introduction

This is a documentation handbook created for the purpose of helping future developers on this project. Before you begin, it is strongly recommended that you read the Welcome page first so that you have an overall idea on the direction on where the project is heading.

If you have any further questions, you can ask for help at #OpenSource channel in Slack.



Welcome

Welcome to the OpenSource project of Data Deploy's ESRI GIS Program
The goal of the project is to eventually develop a similar working program to ESRI's GIS deployed using OpenSource technologies.

In order to make this possible, the following technologies where used in the project.

WEB APPLICATION SIDE

1. Geoserver 2.12.1
2. Openlayers 3 *(Openlayers 4 is already out as of writing the Prototype 1, however the developer of Prototype 1 doesn't guarantee that the piled up codes he has works with v4 that's why he opted for v3 instead but it can certainly be changed)*
3. WAMP 5.7 Stack
    * PHP - Used for the server side fetching and insertion of data. Although it is possible to use Geoserver directly using the REST API but I found it better that PHP is used as the middleware between the Client and the Geoserver that way we can customize what data goes in and out in the Geoserver
    * MYSQL - Frankly, I would have used PostGIS but decided instead to bet on MYSQL's geospatial data support. It works nevertheless you need to configure the geoserver to connect to MYSQL using the mysql-connector plugin.
      NOTE: THE MYSQL DATABASE plays an important role for features that you want to be editable. Although it is possible to directly store features into the server by publishing a ShapeFile(.shp) or GeoPackage but its really a hassle to work on making it successfully in a web program.
4. GIS Plugins - Inorder for the MYSQL Geoserver features to work, you may need to install

GIS plugins such as FWTools. I also found out that installing the PostGIS along with its extras works easier, but again the PostGIS installation is **NOT NECESSARY**.

5. CURL - CURL is heavily by the PHP scripts used for fetching data in the Geoserver. Most of the demos for fetching data for the Geoserver uses CURL so in our case it isn't really a big problem since CURL is usable inside PHP.

DATA PUBLISHING

1. QGIS (Latest Version) - This is our bread and butter for publishing data. However data it publishes is converted into a ShapeFile(.shp) and it can be a problem IF we want the data to be modifiable inside the app. So we use this for rendering layers such as Drone Imageries or informational layers that should never be changed by the end-user

2. ogr2ogr (Part of FWTools) - This converts the ShapeFile(.shp) into a MySQL Database table. Data along with it will also be imported as well. You can use this command as a reference on how to import and convert ShapeFiles

Format:
Hint: Spacing matters here, so only replace content that is highlighted in red and leave everything as is

ogr2ogr -f MySQL MySQL:<database>,host=<host address>,user=<mysql username>,<mysql password>= "<full directory of shape file(enclose it with double quote)>" -nln <name of the table it will create for the import> -update -overwrite -lco engine=MYISAM

Working Example
ogr2ogr -f MySQL MySQL:pictures,host=localhost,user=root,password= "C:\Users\Azure01_Dev\Documents\bbb.shp" -nln cus_layer_pictures -update -overwrite -lco engine=MYISAM

DEVELOPMENT TOOLS

There isn't any in particular. In the developer of Prototype 1's case, he used Netbeans 8.2 with PHP support but then again you can even use a text editor if you wish (such as Sublime or Notepad++)

Getting Started

The application is designed to be easy to catch on in mind that's why frameworks are barely used in this setup.

Rather the project relies mostly on Jquery in terms of layout and functionalities while OpenLayers is the one responsible displaying the data sent out by the server.

This way developers will only have to read the following materials during the course of the development.

1. Openlayers (Version 3 | Version 4) *Only use Version 4 if we are planning to change Openlayers Version*

2. Geoserver User Manual (2.12.1) (MySQL integration on GeoServer)

3. Using MySQL GeoSpatial Data using (WKT)

4. Of course the typical stuff, MySQL DQL, DML, DDL and DCL, Jquery and Jquery Mobile

That's pretty much it. Awesome right? You don't need to learn too many. If you are familiar

with #4, then you only need to study 3 things. Plus there are working codes in the project so it wouldn't be too hard

Please continue to the Application Design page

Application design

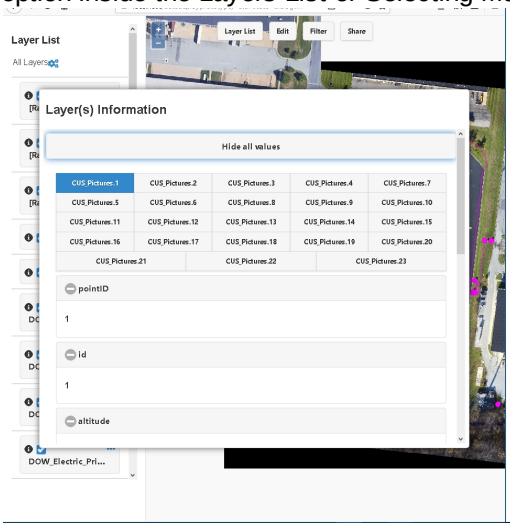
Please read this for first timers to get the overall idea of the project

GUI Interface

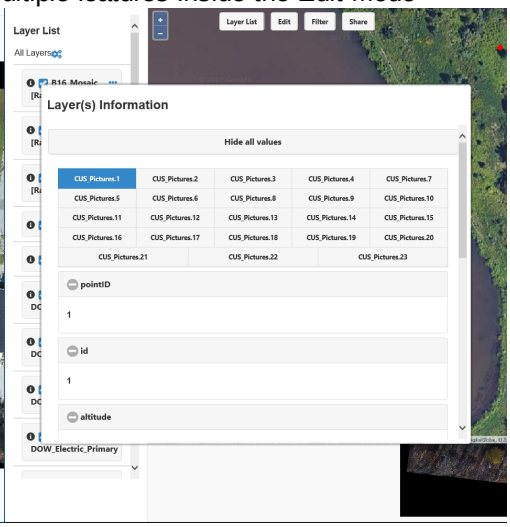
As said in the Getting Started portion, the technologies used in the project are primitive because we want the developers to as much as possible catch up with the project without any special training or whatsoever.

With this the Jquery Mobile is used because it provides majority of the stuff needed while being mobile friendly as well. Bootstrap wasn't consider in the design as well since majority of the time, we only use widgets from Jquery Mobile which has near perfect layout control across all browsers.

Tab View of Jquey Mobile once user selects multiple features using the Show Information option inside the Layers List or Selecting Multiple features inside the Edit Mode



Mozilla Firefox 57



Internet Explorer 12

Mapping Library

Another aspect in the design was the flexibility and customizability of the solution. In the OpenSource area, there are 2 popular mapping libraries used, that is Leaflet or OpenLayers. Certainly Leaflet is nice since it is dead simple but the problem relies on the customizability that we need. There are discussions that at the later phase of the prototype to integrate 3d view into the geoserver and this will be a problem for Leaflet.

Database

For the SQL Server, the MySQL is chosen instead of PostGIS because it is bundled in the WAMP Stack. Anyways there really isn't a feature that we need from MySQL or PostGIS so if in further phase, a decision has to be made in changing database engines, then it's not really a big problem (Just make sure to update the SQL codes in the php files)

The SQL server is necessary since layers that are produced using the SQL are easier to edit in contrast to shapefiles where you have to edit and publish them. Furthermore shapefiles are

NOT concurrent in nature so multiple users cannot add and edit at the same time.

Server Side

For PHP, we use this for manipulating server side actions such as inserting database data making sure that data is compatible with the javascript calls so on and so forth

Conclusion

All of the libraries are hand picked with a purpose

List of Geoserver REST APIs used

These contains the list of REST API used in the geoserver without the need of replication a server side action.

1. GetFeatureInfoUrl (WMS) => used for listing down all the features available on a layer as the user clicks
2. GetFeature (/WFS) => used for drawing out all the layers available in a certain workspace

API Documentation

Project structure

[assets] => This is for storing marker images, is presents because the template the prototype is based on has the assets folder
[css] => The css files used for modifying CSS by Jquery Mobile as well as used for loading bars
[js] => The javascript functionalities
[php] => The php functionalitie
[webfonts] => This is where the fonts are located, mainly we used font-awesome for the icons of certain images, the rest are default configurations of Jquery Mobile
[webinterface] => This is where the openlayers library is stored, i made it like this so that we can make multiple revisions of openlayers WITHOUT modifying the core functionalities
----------[target]
----------[layers] => This section is responsible for deploying and displaying layers into the map
----------[resources] => This section is responsibles for everything openlayers related

Server Side (PHP)

For the server side, majority of the files are self explanatory but for documentation purposes will be listed in here. The server side scripts DO NOT follow any MVC model since it's not really necessary. Also not all actions are made with their PHP equivalent since if the REST API does the job, then there is no point in making a new one (See List of Geoserver REST APIs used for more information)

With that, these are the following files present in the system.

Objects

DataStore.php => represents the datastore object that is returned by the REST API of the Geoserver

but with less data. It contains two classes and they are:
Data => Is the actual data that represents the layer. Since the layer could be a raster image, shape file or MYSQL object depending on the configuration of the Geoserver. Again as stated in the Welcome and Application Design page, only MYSQL layers have editable features which explains why we keep in track what type the layer is in.
DataStore => Is the container class that contains an array of Data

InfoList.php => represents the features that each layer contains. Just like the datastore, this one also uses the REST API if possible.
The file contains two classes as well and these are:
Data => for the actual feature including the data's bundled within the feature.
Informations => for storing the arrays of features contain in a layer.

These objects are designed to be used along with the json_encode. In a way we can consider that these PHP files are APIs as well

Operations

These 2 files are called first by layers.js
coverageStoreList.php => Used for listing all raster layers(or coverages) within the geoserver workspace. This is called first before the second one which is the dataStoreList.php
dataStoreList.php => Used for listing all shape layers within the geoserver workspace.

ShowMySQLInfo.php => Since GetFeature is a WFS only service and does not list down datas in a feature that was projected in MySQL, this file is made to fill that gap.
addMySQLData => Inserts data to the MySQL Table Layer
editMySQLData => Edits the data of a feature in the MYSQL Table Layer

Client Side (Javascript)

The client side is seperated to 6 seperate javascript files with their own specific purposes.
The rest of the javascript files that you see are the library files and should not be editted

Below is the list of files with their specific purposes

[Inside the WebInterface folder]
layers.js => Contains the codes for fetching layers in the geoserver
qgis2web.js => Contains the codes for interacting with the mapping library (This file also loads the layers.js)

[Inside the JS folder.....]
preload.js => Contains the variables that will be used by other files (Mainly, layers.js, qgis2web.js and Autolinker.min.js)
overrides.js => Contains overrides of actions of different libraries from Jquery to OpenLayers
functionalities.js => Contains all the UI functionalities of the web app.
marker.js => Contains all the OpenLayers added functionalities such as adding a marker. It also contains potential

codes for different purposes such as finding the shortest path (Using OpenStreetMaps)

Layers.js

==========

Contains the variables and methods used for loading layers. Variables in here are used by other JS files as well (mostly by functionalities.js)
here are the list of variables used by the Web App

layers

The actual openlayers layer
Do not touch this (such as swapping areas)
As a lot of functionalities (GUI related) rely on this one

layersWMS

The layers that are all using WMS
I need this for fetching information to the geoserver

layersConfig

This will be the one responsible for identifying if the layer is
of MySQL or not

layerExtents

The extent (or position) of a particular layer
The same as above, this should not be touched
since a lot of functionalities rely on this one

layerGeometry

The geometry shape files (converted and loaded after loading)
are stored in here

layerCalls

The URL of the REST API (If exists) is stored into this variable

layerNames

The names of the layers as fetched in the GeoServer
unlike the other two, this one is interchangeable

layerInfoCallStack

This one obtains the layer info url. It resets everytime the user clicks

group_newmap

The layer group array to be feeded to qgis2web.js

group_layer_buffer

The layer group array buffer, this one holds the current groups together

layersList

The layerList to be feeded to qgis2web

imageLastIndex

This keeps track of the last image index loaded by coverageStoreList.php this is important so that we can ignore displaying information when user clicks on an image. Since it doesn't contain important data at all.