Final Report

Geospatial Mapping w/ Coverage Overlays

Group May1610:

Luke Milius

Brenda Lopez

Jacob Caithamer

Sarah Ulmer

Franklin Nelson

Advisors:

Swamy Ponpandi

Akhilesh Tyagi

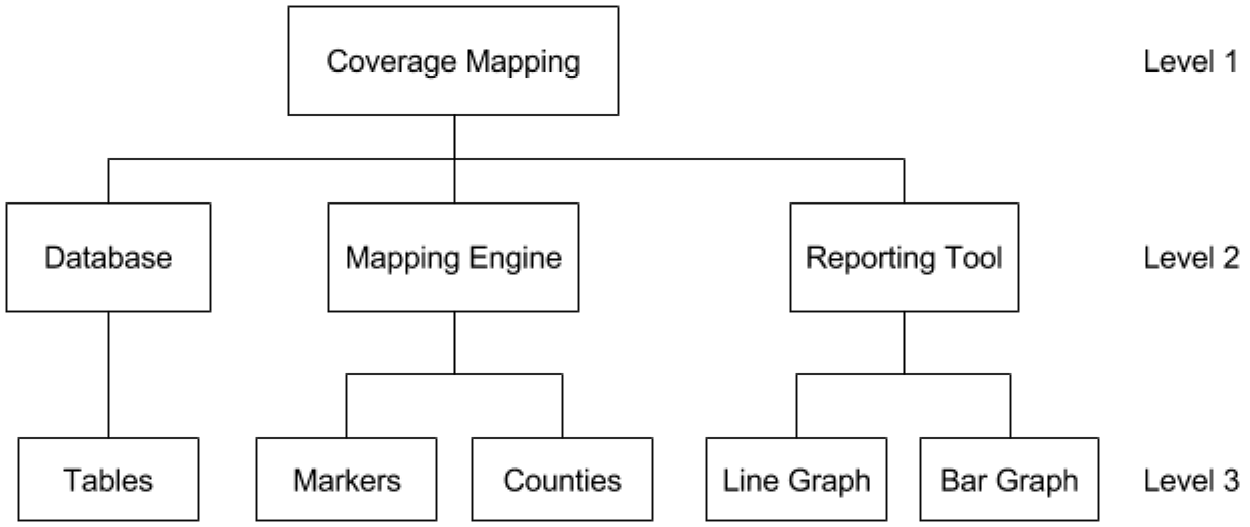
Client:

Benazir Fateh

Nicholas Sitter

Table of Contents

1. Project Design



Level 1

Coverage Mapping

The structure of the entire project, containing the database, mapping engine, and reporting tool along with their subcategories.

Level 2

Database

The database is the backbone of the entire project, both the mapping engine and reporting tool rely upon the database in order to function properly. The database uses PostgreSQL with PostGIS.

Mapping Engine

The first half of the analysis tools provided in this project. The mapping engine plots points based on data retrieved from the database. The user will then be able to see a visual representation of variations in a selected attribute based on the location the point is located in.

Reporting Tool

The second half of the analysis tools provided in this project. The reporting tool takes data from a drive test in the database, then creates either a line graph comparing two different attributes, or a bar graph of one attribute.

Level 3

Tables

The database contains 4 tables arranged in a hierarchical manner. The top level table, *driveTests* stores all the drive tests that exist in the database, each with an id and name. Two mid level tables are created for each drive test, *logFiles* stores log files, and *averages* stores averages for every county. Finally, the bottom table, *dataPoints* stores data retrieved from a csv file and is created for each log file.

Markers

Points are placed on the map corresponding to the latitude and longitude columns from the *dataPoints* table. The color of each point is dependant on an attribute which is chosen by the user. Points can be rendered individually, clustered together, or as a heat map.

Counties

Provides an average of the corresponding attribute drive test values and is displayed by county. Each Iowa county is displayed accordingly.

Line Graph

Creates a line graph comparing two different attributes from the *dataPoints* table for selected drive tests and counties.

Bar Graph

Creates a bar graph which shows a comparison of one attribute per county in Iowa for selected drive tests.

2. Implementation Details

Repository

Project is stored and version-controlled on Git repository at Iowa State University that John Deere also has access to.

Database

A PostgreSQL database is used along with the PostGIS extension to store test data from csv files generated by John Deere's telematic devices. The database contains the following tables:

dataPoints
dataid
time
fixtime
…
logfileid
geom

logFiles
logfileid
…
drivetestid

driveTests
drivetestid
name

averages
averageid
drivetestid
…

File Upload

In order to populate the database, a user may upload any number of CSV files into different drive tests. These are uploaded via a POST request and processed by the server to ensure that they are correctly formatted and contain the expected types of data. Once verified, the data contained in the file is added to the database, and a response is sent to the client indicating which files were successfully uploaded and which failed.

Mapping Engine

Mapbox is used as the mapping engine for the project. In addition, two third party plugins, leaflet-heat and leaflet-marker, are implemented to add extra rendering styles for markers place on the map. A TIGER/Line® Shapefile is used to generate an outline of every county in Iowa.

Reporting Tool

The reporting tool uses Scalable Vector Graphics, or SVG, to render line and bar graphs in browser. We chose to use SVG because it is widely supported and performs well with large amounts of data.

When a user makes a request on the reports page, that request is sent to the server via an Ajax call, which processes it and extracts the relevant data from the database. The server then orders this data, trims it if it is too large, and uses it to construct SVG markup of the requested graph. Finally, this markup is sent back to the client and inserted into the DOM for display.

3. Testing

Manual user testing was the primary method of testing for bugs in this project. Because the mapping engine and reporting tool did not interact with each other in any way; and the database needed to function correctly for either the map or reports to work, user testing was sufficient for making sure each module functioned properly.

4. Operation Manual

Deployment Notes

The setup configuration listed here is for deploying to any Ubuntu 14.04 desktop or server. This tool should run fine on any other version of linux or Windows/OSX. Although the configuration options would be different, the setup should follow the same concept however.

Firewall Configuration

The firewall on the host can be set to allow only ports 22 and 80 as incoming, and everything as outgoing. The rules could be set up more strict than this, but for sure, ssh access is necessary to view log files while 80 is required to view the web application.

Server Configuration

For Ubuntu 14.04, use apt-get to install the necessary packages from the repository:

- sudo apt-get update
- sudo apt-get upgrade
- sudo apt-get install default-jre postgresql postgis apache2 tomcat7
- sudo apt-get install postgresql-9.3 postgresql-9.3-postgis-2.1 postgresql-client-9.3

The above packages should be everything needed to properly run the tool.

Database Configuration

The database must be configured with the correct tables and columns in order to persist the uploaded data and precomputed averages. The .sql files used in the following commands can be found at the root of the repository (under the may1610 folder). These .sql files will need to be transferred to the remote server (if running remotely) in order to run them through the psql commands. If a separate server is being used as the database server, these commands will need to be altered to be run from your local machine to communicate with the remote server.

- sudo vi /etc/postgresql/9.3/main/pg_hba.conf
    - towards the bottom of the file, change the first line for the postgres user on the very right side from peer to md5
- sudo -u postgres psql
    - \password postgres

    - (Enter password for postgres user twice)
    - \q
- sudo service postgresql restart
- psql -U postgres -f setupDatabase.sql
- This is only needed if you need to re-create the countyShapeFile
    - shp2pgsql -s 4269 -I tl_2015_us_county/tl_2015_us_county.shp public.tl_2015_us_county > countyShapeFile.sql
- psql -U postgres -d coverageOverlays -f countyShapeFile.sql
- psql -U postgres -d coverageOverlays -f createAverageTable.sql
- Copy database.properties file into the coverage-overlays/src/main/resources/properties directory.
    - Contents of database.properties file:
        - jdbc.driverClassName=org.postgresql.Driver
        - jdbc.url=jdbc:postgresql://localhost:5432/coverageOverlays
        - jdbc.username=postgres
        - jdbc.password=(password for database goes here)

Apache Configuration

The default apache configuration should work. It would be good to check that in the /etc/apache2/sites-enabled/000-default.conf file, that everything is commented out between the two xml tags: <VirtualHost *:80> and </VirtualHost> except for the two log file lines:

- ErrorLog ${APACHE_LOG_DIR}/error.log
- CustomLog ${APACHE_LOG_DIR}/access.log combined

This configuration should force the port 8080 that tomcat is providing on to be automatically forwarded to port 80.

Tomcat Configuration

The first configuration of tomcat is to set the Java Virtual Machine max heap size in order to accommodate the large amounts of data during large drive/log tests.

- vi /etc/default/tomcat7
    - Edit the JAVA_OPTS variable by changing the -Xmx parameter to at least 512m (so it should look like: -Xmx512m).
- sudo service tomcat7 restart

The tomcat admin tool must be configured for maven to connect to the manager-script URL API in order to deploy the application through maven. This can be run manually from a local development machine or even from a build server. The server side configuration is as follows:

- To enable management scripts:
    - Add user with user roles : “manager-script” and “admin-script” for the manager and host manager URL’s respectively using:
    - sudo vi /etc/tomcat7/tomcat-users.xml
        - <role rolename="manager-script"/>
        - <role rolename="admin-script"/>
        - <user username="tomcatmgr" password="(password goes here in quotes)" roles="manager-script,admin-script"/>
- To add the connection info to your local machine or the machine that will run and deploy using maven:
    - Add the following server options to %MAVEN_PATH%/conf/settings.xml on the maven configuration (not on the server):

        -
          ```
          <?xml version="1.0" encoding="UTF-8"?>
          <settings ...>
                  <servers>
                          <server>
                                  <id>TomcatServer</id>
                                  <username>tomcatmgr</username>
                                  <password>(password goes here)</password>
                          </server>
                  </servers>
          </settings>
          ```

- The deployment configuration is already included in the project's pom.xml, so there are three options that can be used to take advantage of this scripted deployment:
    - Deploy works for deploying the web application for the first time, after that, if the application is already deployed, this option will result in an error stating that there is already an application deployed at this endpoint
        - mvn tomcat7:deploy
    - Undeploy works if you would like to undeploy the application before deploying a new one. This can also be useful if you would like to undeploy the web application altogether and take it offline.
        - mvn tomcat7:undeploy
    - Redeploy is probably the most used option for this command. Redeploy runs undeploy, then deploy in the same command, right after one another. This can be much quicker to use since it does everything for you in one command.
        - mvn tomcat7:redeploy

Authentication Configuration

Currently, the application authentication is configured to use authentication that is hard coded into the Spring Framework configuration files. In the may1610/coverage-overlays/src/main/resources/contexts/spring-security.xml file. Towards the bottom of that file, there is a section that looks like the xml group below. The authentication-provider is the id or class name of the Spring Bean that will be handling the authentication. This can either be a bean defined in the same file or it can be a java class file that authenticates via the database or an authentication server like LDAP or Active Directory.

```
<security:authentication-manager>
```

```
                        <security:authentication-provider
                        user-service-ref="userDetailsService">
                        <security:password-encoder ref="passwordEncoder" />
                        </security:authentication-provider>
            </security:authentication-manager>
```

The following bean configures to use Spring Security's password encoder to hash all passwords:

```
    <bean
    class="org.springframework.security.authentication.encoding.ShaPasswordEncoder"
        id="passwordEncoder">
        <constructor-arg value="256"/>
    </bean>
```

Finally, the last portion is the list of in-memory users with their hashed password value:

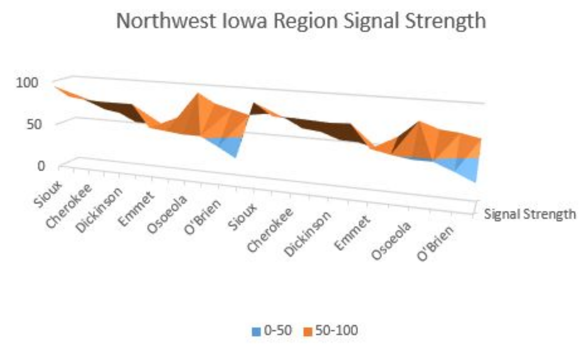
```
    <security:user-service id="userDetailsService">
        <security:user name="admin" password="PasswordHashHere"
        authorities="ROLE_USER, ROLE_ADMIN" />
        <security:user name="user" password="PasswordHashHere"
        authorities="ROLE_USER" />
    </security:user-service>
```

Accessing the Application

After the above configuration is complete, the web application should be up and running correctly. The web application should be accessible by either navigating to the IP address of the machine it is running on or the DNS name that is assigned to that machine.

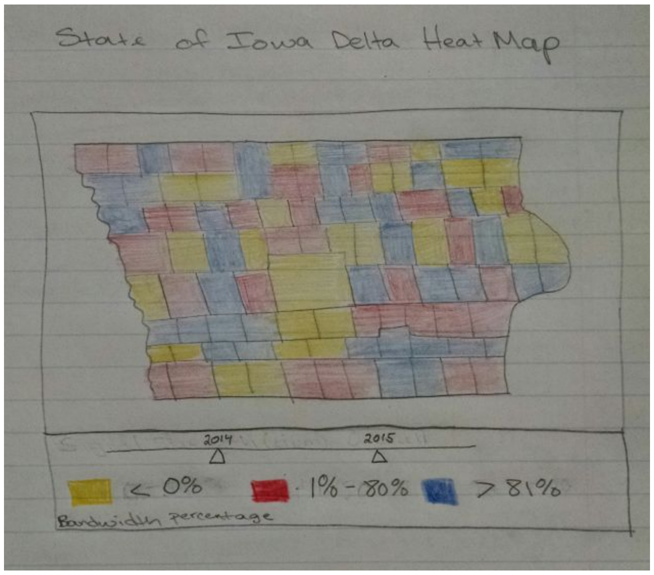
5. Previous Designs

Reporting Tool



Our original design for the reporting tool portion of the project included 3-D graphs such as the one pictured above. After receiving feedback from our client, this type of tool was determined to be not very useful and we did not implement this feature.

Mapping Tool



Our original design was to display a heat map of all the Iowa counties by a single attribute filtered by year. The original design did not include the individual markers as points, clusters, and heat. To make our project modular, we decided to have a drop down menus to select the drive test(s), date, attribute, and to decide how to display the data.

We also considered integrating both the mapping and reporting tools into one larger page, which would allow users to more easily compare different types of data. However, we scrapped this because of page space limitations, as well as for ease of development.

6. Other Considerations

Deploying the project into the Amazon Web Server for the first time was a huge challenge because it was deployed far into the development. It took quite a while to figure out all of the dependencies to install for the environment and to get permissions set correctly to be both secure but non-intrusive to the workings of the web application.

A second major challenge included implementing spring security into an existing spring web application. We found it was much easier to integrate spring security into a new application for future work. For this project, there were several attempts made but each one either broke some functionality of the application or prevented it from working at all. Eventually, we found a conflicting dependency between two versions of spring and jetty and were able to get those resolved with a different version of jetty.