

A.G. WASSENAAR

**A.G. Wassenaar**  
Geotechnical and Environmental Consultants | **Inc.**

---

Cecilia Smyth and Jimmie Butler

6/20/2012

## Introduction

A.G. Wassenaar is an engineering firm which provides geotechnical and construction-related services for commercial and residential properties in the Denver Metro area. In particular, A.G. Wassenaar field engineers perform “observations” at up and coming property sites. Observations are inspections of particular parts of buildings or the environment in which they are being built and can result in a pass or fail, based on how the work or site compare to known quality standards. A passing observation is good for a limited amount of time; a failing observation requires the client to take corrective measures and set up a re-observation. This process of taking client information, scheduling field engineers, performing observations, and recording results, unfortunately, produces a lot of paperwork. At present, Microsoft Access is in use in the A.G. Wassenaar office and, while this program does a great job of managing client databases, this software is insufficient for managing, generating, and distributing the paperwork necessary for the field engineer’s use in the field. As such, A.G. Wassenaar was looking to have made for them an Android application to be used on tablets carried by the field engineers. This application would eliminate the need for printed assignment sheets (of which one was being printed per scheduled observation, see Figure 1 below), observation review sheets (Figure 2), printed daily schedules, and hard copies of the pass and fail forms (Figures 3 and 4). The end goal is to move toward a paperless process by utilizing email for distribution of paperwork along with database access and easy form-filling through the much more mobile tablet interface.

Company: **Infinity Communities, LLC** Project: **111952**

Region: **Lowry Area** Subdivision: **Stapleton**

Lot: **7** Block: **11** Filing: **32** Blg: Unit:

Tech Engineer: **Retoff, John**

Insp Address: **7959 East 32nd Avenue**

PO:

**Observations:**

Excavation and Footing Form Observation

**Special Conditions:**

\*\*\* NOTE: This customer\subdivision has red flags \*\*\*

Date: **Tuesday, May 15, 2012** Time: **2:30 PM**

Caller: **Amber 303-841-9542 (Penley)** Pour: **3:30 PM**

Figure 1: A typical assignment form for a field engineer

PIER LOG \_\_\_\_\_

CLIENT \_\_\_\_\_ FIELD ENGINEER \_\_\_\_\_

PROJECT \_\_\_\_\_ ADDRESS \_\_\_\_\_

DATE \_\_\_\_\_ LOT \_\_\_\_\_ BLOCK \_\_\_\_\_ BLDG. \_\_\_\_\_ UNIT \_\_\_\_\_ FILING \_\_\_\_\_

ENGINEER/PLAN NUMBER: \_\_\_\_\_ SUBDIVISION \_\_\_\_\_

PIER NO.	PIER DIA.	PIER DEPTH				SPECIFIED PENETRATION	CASED DEPTH	PIER LENGTH	SPECIFIED LENGTH	DEPTH OF WATER (3" MAXIMUM)	CONDITION OF WALLS	CONDITION OF BOTTOM	UPPER LEVEL	LOWER LEVEL	WALKOUT LEVEL	DESCRIPTION OF SUPPORTING STRATUM
		OVERBURDEN IN	IN	BEDROCK IN	IN											

S= STABLE    C= CLEAN    D = DRY    CA= CASED    W= WET    CS= CLAYSTONE    SS= SANDSTONE

**FOUNDATION PLANS**  
 \_\_\_\_\_ MINIMUM LENGTH  
 \_\_\_\_\_ SPECIFIED PENETRATION  
 \_\_\_\_ # \_\_\_\_ STEEL SIZE - CAGES ON SITE    YES    NO  
 4"   6"   8"   VOID SIZE  
 BEARING PRESSURES  
 END/ SIDE / DEAD LOAD  
 \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_  
 ENGINEER: \_\_\_\_\_  
 PLAN NUMBER: \_\_\_\_\_    STRUCT FLR    SOG

**SOILS REPORT**  
 \_\_\_\_\_ MINIMUM LENGTH  
 \_\_\_\_\_ SPECIFIED PENETRATION  
 \_\_\_\_ # \_\_\_\_ STEEL SIZE - CAGED    YES    NO  
 4"   6"   8"   VOID SIZE  
 BEARING PRESSURES  
 END /SIDE / DEAD LOAD  
 \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_  
 ENGINEER: AGW  
 REPORT NUMBER: \_\_\_\_\_

DRILLING CO. \_\_\_\_\_    NUMBER OF RIGS: \_\_\_\_\_  
 STEEL ON SITE:     YES     NO  
 CONCRETE ON SITE:  YES     NO  
 SONATUBE ON SITE:  YES     NO  
 PUMP ON SITE:     YES     NO  
 SHEAR RINGS:     YES     NO

STEEL IN PIERS:     YES     NO  
 CONCRETE IN PIERS:  YES     NO  
 DRILL AND POUR:     YES     NO  
 PUMP FROM BOTTOM UP:  YES     NO  
 ROUGHENING         YES     NO

REMARKS: \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

Figure 2: An observation review form to be filled out by a field engineer

# A.G. Wassenaar

Geotechnical and Environmental Consultants

2180 South Ivanhoe Street, Suite 5  
Denver, Colorado 80222-5710  
303-759-8100 Fax 303-756-2920  
www.agwassenaar.com

# Inc.

CLIENT:  \_\_\_\_\_ DATE OF OBSERVATION: \_\_\_\_\_  
ADDRESS:  \_\_\_\_\_  
LOT: \_\_\_\_\_ BLOCK: \_\_\_\_\_ BLDG: \_\_\_\_\_ UNIT: \_\_\_\_\_ FILING: \_\_\_\_\_  
SUBDIVISION:  \_\_\_\_\_ PROJECT #:  \_\_\_\_\_  
ENGINEER / PLAN #: \_\_\_\_\_

**OBSERVATIONS MARKED BELOW WERE FOUND TO BE IN GENERAL CONFORMANCE WITH OUR RECOMMENDATIONS AND/OR PROJECT PLANS/SPECIFICATIONS AT THE TIME OF OUR VISIT.**

- EXCAVATION
- PT / RAFT FOUNDATION (TRENCHES / CABLE PLACEMENT / STRESSING)
- FOOTING (DECK / PORCH / PATIO)
- PIER / HELICAL (DECK / PORCH / PATIO)
- FOUNDATION WALL REINFORCING STEEL
- CONCRETE ENCASED ELECTRODE
- FOUNDATION VOID 4" 6" 8" 10" 12"
- PERIMETER DRAIN SYSTEM (INTERIOR / EXTERIOR)
- UNDERDRAIN EXTENSION
- DAMPPROOFING
- FOUNDATION INSULATION
- STRUCTURAL FLOOR (STEEL FRAME / CONCRETE REINFORCING)
- SLAB-ON-GRADE (SOG / SF)
- COMPULEVEL SURVEY
- CRAWL SPACE ENVIRONMENT (BASEMENT / MAIN LEVEL)
- VAPOR RETARDER (BASEMENT / MAIN LEVEL)
- BACKFILL (FOUNDATION / UTILITY)
- MOISTURE / DENSITY TEST (GARAGE / DRIVEWAY / SIDEWALK)
- OTHER: \_\_\_\_\_

REMARKS: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

ENGINEERING FIELD TECHNICIAN: \_\_\_\_\_

FORMS/OBS FORM 1\_RES (04-12-05)

Figure 3: A blank pass form for a completed observation

# A.G. Wassenaar

Geotechnical and Environmental Consultants

2180 South Ivanhoe Street, Suite 5  
Denver, Colorado 80222-5710

303-759-8100 Fax 303-756-2920

www.agwassenaar.com

# Inc.

CLIENT: \_\_\_\_\_ DATE OF OBSERVATION: \_\_\_\_\_

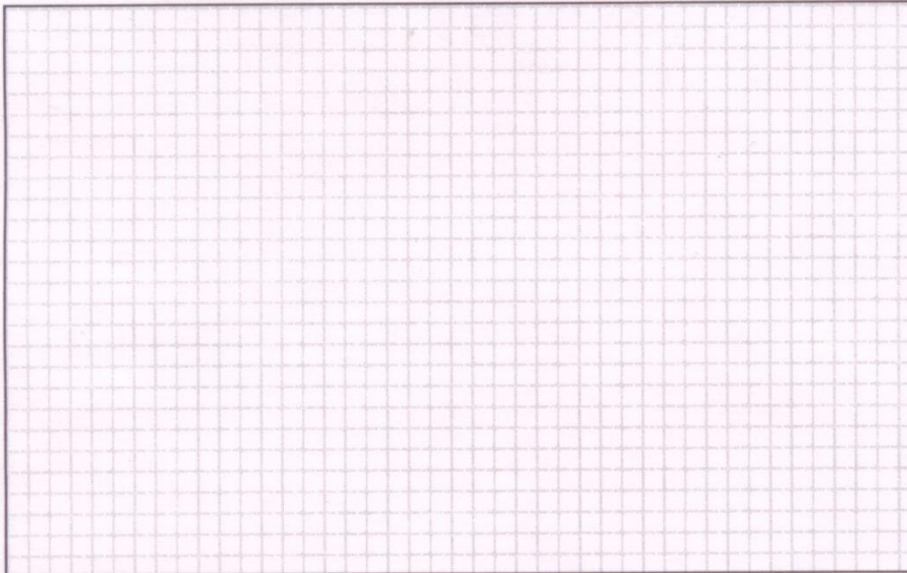
ADDRESS: \_\_\_\_\_

LOT: \_\_\_\_\_ BLOCK: \_\_\_\_\_ BLDG: \_\_\_\_\_ UNIT: \_\_\_\_\_ FILING: \_\_\_\_\_

SUBDIVISION: \_\_\_\_\_ PROJECT #: \_\_\_\_\_

ENGINEER/PLAN #: \_\_\_\_\_

REQUIRED RE-OBSERVATION(s): \_\_\_\_\_



REMARKS: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

CANCELLED ON-SITE TIME: \_\_\_\_\_

ENGINEERING FIELD TECHNICIAN: \_\_\_\_\_

## RE-OBSERVATION REQUIRED

FORMSIOBS FORM 2\_RES (04-12-05)

Figure 4: A blank fail form for a completed observation

## Requirements

### *A. Functional Requirements*

1. Information from existing Access database must be imported into the new application's database
2. Application must have authentication preventing unauthorized use by either outside entities or employees lacking proper clearance
3. Application administrators need be able to modify and establish new roles within application authentication such that item 3 above be properly maintained
4. Application administrators need be able to create, view, update, and archive information stored in the database
5. Data within the database should never be deleted or destroyed, but in some way kept for future reference
6. Field engineers must be able to generate/create pass and fail tickets that are emailed to the client
7. Application should facilitate scheduling of employees
8. Red flags (notes that field engineers must take into account regarding a particular client or project, such as an usual delivery method for pass/fail tickets or previously encountered problems with a particular site that need special notice) should be properly related to the client and/or project and as part of the field engineer display

### *B. Non-Functional Requirements*

1. Application must run or be viewable on an Android tablet
2. Any development environment may used, as appropriate for the project
3. Any programming language may be used, as appropriate for the project
4. Measures must be taken to keep the application out of the hands of competitors of A.G. Wassenaar
5. Application must have security measures to prevent tampering by outside entities
6. Database schema must be established such that the existing Access database components can be related to the functions of the new application
7. Application should have clear and concise user interface
8. Database schema and organizational relationships should not be damaged or at all affected by any actions available to application users
9. Deployment options for A.G. Wassenaar should be discussed with the end result of a deployed application or an application with the means by which to be deployed in the future

## System Architecture

The very basic architectural flow of the program is illustrated below (Figure 5):

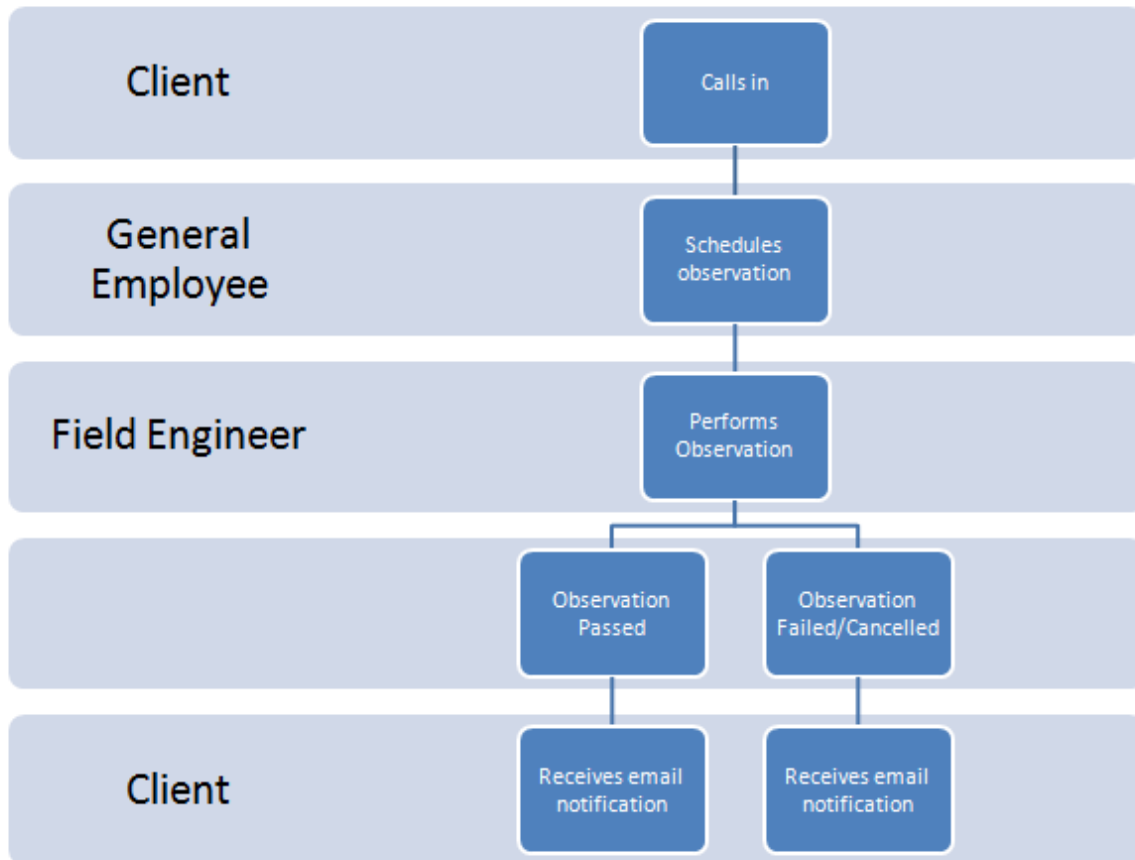


Figure 5: Very basic architectural flow of proposed (and implemented) application

The design architecture used was MVC, meaning Model, View, Controller. This follows the de facto standard of the Ruby on Rails language. The core philosophy of this architecture is to separate the display logic (the view) from the database interface (the model), allowing users to manage the application (through the controller).

Furthermore, the controllers conform to that of a typical RESTful framework, also standard for Rails. A RESTful controller is converted to CRUD, the shorthand term for create, read, update, delete functionality. Many of our controllers contain only the first three methods, leaving out the delete action in favor of a special instance of update that we called “archive”. Using a RESTful controller helps to keep the contained code tidy and neat, while still allowing any possible actions.

The overall database is shown below (Figure 6) and acts a good field of reference for further discussion of application architecture:



### AGWebApp domain model

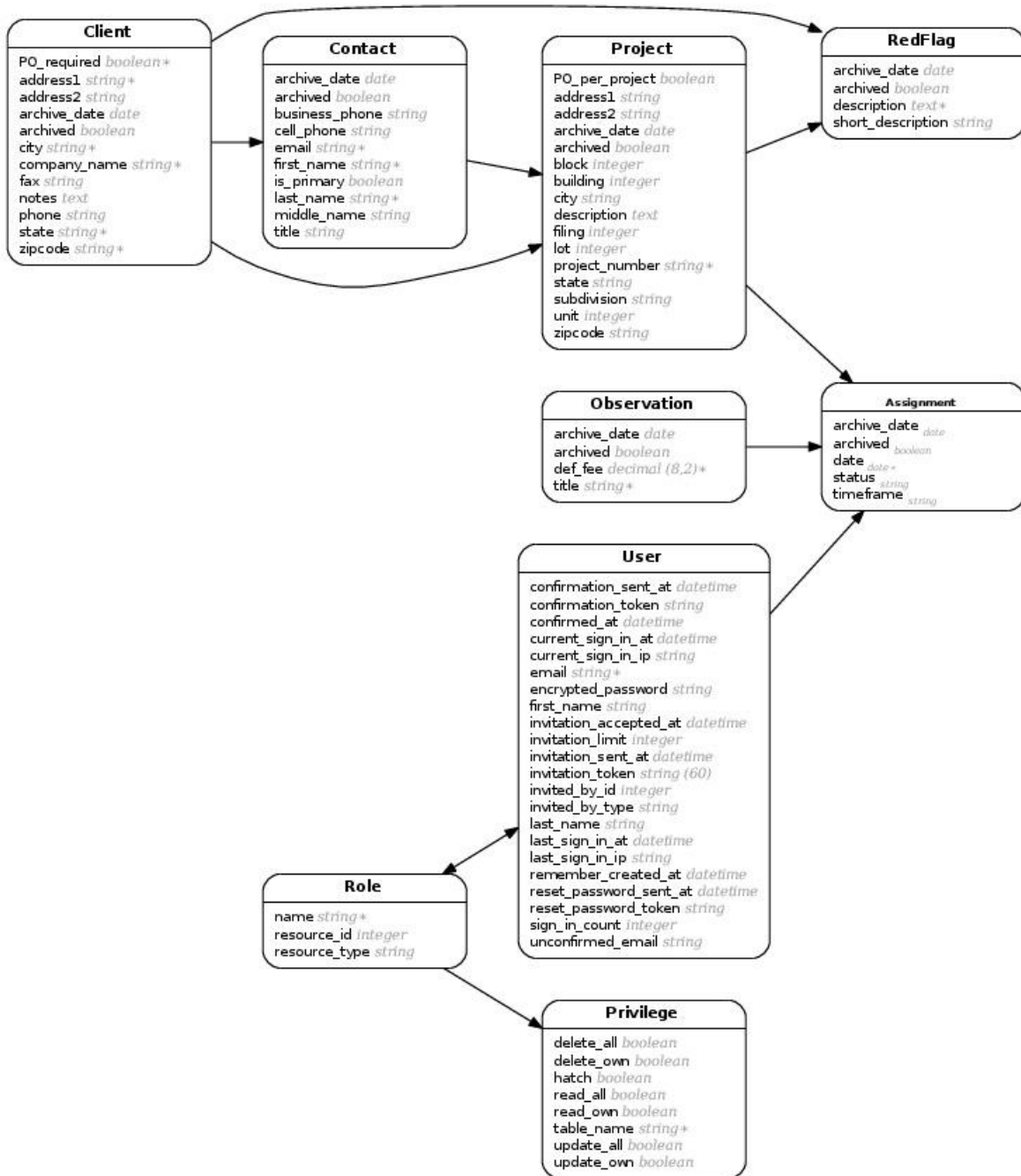


Figure 6: Overview of database schema

An Assignment is associated with an Observation (inspection to be performed), a Project (where to perform it), and may be associated with a User (who to perform it). Assignments also have fields called “timeframes,” which specify when the Assignment is to take place. Each combination of these is

not necessarily unique; an observation can be re-performed by the same field engineer, at the same location, if it had previously failed or been cancelled. The Assignments table is also non-exhaustive.

An Observation has a default fee, which is overridden in the case of price negotiations with a client (unimplemented).

A Project is associated with a Client and three Contacts: the primary Contact, the letter Contact (the person to which all hard copies of correspondence and documentation should be addressed), and the billing Contact (usually the same as the primary contact). A Project must have a primary Contact specified. All location information needed to find a project site is also stored in the Projects table.

A Red Flag is associated with either a Project or a Client. This table gives a field engineer an idea of anything which may need to be looked at in extra detail, such as swelling soils or a client which wishes to be contacted immediately after completing an observation.

The Contacts table is associated only with a Client. It contains information regarding how to contact a particular person.

The Clients table contains any information which may be associated with the office of a given client, such as name of company, business phone number, and company main office location.

A very brief and general overview of the aforementioned relational architecture is illustrated below (Figure7):

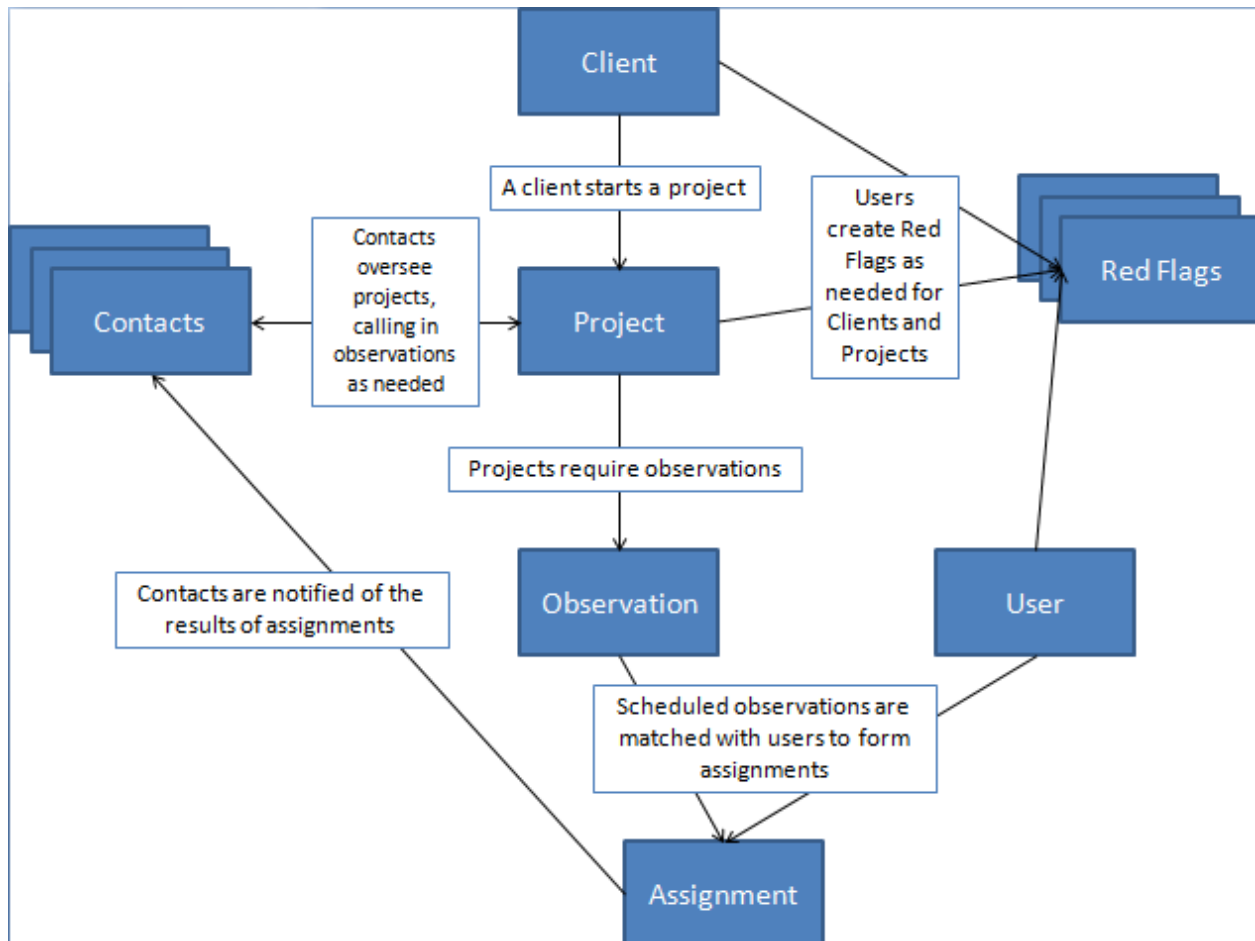


Figure 7: Basic architectural flow diagram

The authentication architecture, however, is a bit more unique (though not overly so). A user has and belongs to many roles, each of which is defined by a collection of privileges (where a privilege is the go-ahead to access certain functionality). The administrator role is hard coded to always have all privileges, whereas privileges can be given to or revoked from all other users. A privilege contains abilities corresponding to the ability to create, read, update, or archive/delete an entry in a given database table. Basically, an ability is mapped to a single CRUD action, and thus is not stored in the database. New users can be registered on an invitation only basis, giving the administrator the ability to restrict outside access to the application. The first measure taken to prevent unauthorized entry, and an extremely weak one, was to declare the robots.txt file to request that this site not be included on a search engine.

## Technical Design

For authentication we used a combination of the Ruby Gems Devise, CanCan, and Rolify. Devise handles basic login and logout, as well as invitations and confirmation of user email addresses. CanCan centralizes the authentication logic in a single file which also allowed us to create a system of privileges derived from the database. Each user may have many roles (or none at all), and the actions he or she can perform is the combination of all the privileges that are associated with each of their roles.

After authentication, we looked toward importation. In order to import A.G. Wassenaar's existing data into our application's production database three steps were necessary. First, we converted the data to a comma-delineated file (file extension .csv). For this step we used either Access itself to export or an open-source tool known as mdbtools. Second, we removed any characters incompatible with utf-8, using a script utilizing "sed" (a command line string editor for \*nix systems). Third, we used Rake tasks to import and clean the data. This last step was by far the messiest, considering the fact that the new application and the previous program (Microsoft Access) had different definitions of required fields. As a result, unknown values were required for some of the fields that the new application specified should not be null.

Mailing was handled through Google's Gmail, resulting in a maximum of 500 emails outgoing and received per day. Gmail allowed for easy use, as opposed to a system such as sendmail, which would potentially require some server-side authentication.

The overall look and feel of the application was managed by yet another tool. Bootstrap, an interface development toolkit from and used by the creators of Twitter, was used extensively throughout the program. This toolkit provided built-in, simplistic, and clean user interface styling and functionality. Specifically, Bootstrap contains HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript (in the form of twelve custom JQuery plugins) so as to encapsulate a very wide range of design needs. The result is a uniformly colored and styled interface that has the flexibility to be viewed properly on a number of different devices, from desktops to tablets to smartphones. For example, an entire table can be stylized simply by adding a class, aptly named 'table', which is predefined by the Bootstrap kit. Components of the Bootstrap toolkit used in this application include typography, tables, forms, the navigation bar, alerts, working close icons for the alerts, and navigation by tabs. Of particular note in this list are the navigation bar and the navigation by tabs. The navigation bar collapses a specified portion of the navigation links into a button on the navigation bar when the application is viewed at 940 pixels or less in width. This is important for use of the application on mobile devices. Similarly, the navigation by tabs creates a clean, concise means of accessing related data in a manner which will keep smaller screens uncluttered without compromising usability. Overall, the Bootstrap toolkit provided an invaluable means of creating a basic, professional user interface that is both easy to maintain and easy to use.

One gem which proved useful in the user interface was called "nested\_form" and it allowed us to create a form to edit associated objects in a more streamlined manner than previously available. This can be observed in both the Role edit view, and the User form view. The Role form view lists all Privileges associated with that Role and allows for the editing of them. The User view is even more unique, displaying all Roles as check boxes, and allowing someone with proper Privileges (probably an administrator), to grant or remove any Role from the User being edited.

## Design and Implementation Decisions

We chose Rails for this application because of our team's previous experience with the framework, as well as its popularity among the web development community. Furthermore, the Rails support community is active and helpful. The framework allows for gems, which are community created extensions for the framework. One example of the flexibility of the Rails framework was mentioned earlier in this report; the database schematic diagram provided is automatically generated by one such gem. There are over 40,000 gems in existence. The Ruby language, which Rails supplements, is a high-level language which allows for rapid development. Ruby does, however, lack somewhat in speed, but Rails compensates for this through caching.

Authentication was handled through Devise, CanCan and Rolify because they provided a clean, streamlined, and easy to use means of creating and maintaining authentication for the application. These pieces of software are not only well-documented, but also very frequently and successfully used together to create just the type of flexible, role and privilege based authentication system we were looking for. Additionally, this combination of software is designed to interface well with the Twitter Bootstrap toolkit. In fact, it was through research on the use of Devise, CanCan, and Rolify that we discovered Bootstrap in the first place.

Deployment was originally intended for GoDaddy, but this was scrapped due to their lack of support for any updates to the Ruby on Rails language issued in the last six years. Heroku was turned to as backup means of deployment, but proved slow and unreliable for even the size of the client's current database. Some more tweaking may be able to remedy this.

As a result of the change in deployment option, from GoDaddy to Heroku, the database management system also had to be changed. To begin the project we had used sqlite3, which comes prepackaged with Ruby on Rails, but the default and strongly recommendation database system used by Heroku is PostgreSQL, prompting us to change our application. The feature set available for PostgreSQL is very large, more than we actually needed for this project, in fact. Actually, sqlite3 would be a better option for the smaller size that this project is. If Heroku is not used as the final means of deployment, we recommend reverting back to sqlite3 for the performance benefits, as well as ease of use. This would have been our next step had we had the time to complete it.

## Results

Much more can be done on this application, such as adding Ajax forms or adding Google assistance to the scheduling page to bring up a map of the Assignment location. A drag and drop style system can be added to the scheduling page, as well as the addition of a more in depth email generator. Billing could also be added, as well as a way in which to override a fee. Improved data cleanup and import from the Access database would be beneficial. A sketching or photo field can be added to the pass/fail side of things, and an observation form generator which can be used in the web interface could be used. This generator would add considerable flexibility to the site. A native Android app can be used for greater speed, and lower network usage. Alternatively, Ajax could be used for the same purpose. Rails handles performance through heavy caching, though the PostgreSQL database side of things is slow and could potentially benefit from being reverted back to sqlite3.

## Appendix

This application is designed to run on Ruby version 1.9.2 with the Rails gem version 3.2.1. Product installation instructions will depend on the production environment deployed to. I would recommend either a cloud based deployment or an in-house server. An in-house server would likely be the more expensive of the two due to utilities, time to configure, maintenance, and reliability. Examples of cloud based solutions include Heroku (free version proved unreliable), EngineYard, DreamHost, and HostGator.