**Supplier Configuration Utility**

**TraceGains, Inc.**

**Orion Kostival**

**March 8, 2013**

1. **Introduction**

**TraceGains, Inc. is an emerging document management company that aims to attach actionable intelligence to the data extracted from their customer’s documentation. Focused heavily on the food manufacturing industry, the main goal of TraceGains is to provide their customers with a way to digitize their documents and attach real-time monitoring to the data present in those documents. A majority of food manufacturing companies currently have filing cabinets full of regulatory documents which will often satisfy the federal health and safety requirements, but does not allow the companies to extract and act upon the meaningful data within those documents. By digitizing these documents, extracting data from them and analyzing the data as it is received, TraceGains can provide instant feedback based on the customer defined specifications. This helps reduce costs for the customer as well as the customer’s suppliers all while ensuring that the suppliers are meeting the outline quality and safety standards. The ultimate goal for TraceGains is to enable continuous improvement of the food manufacturing process across the industry.**

**TraceGains has a great solution for their customers once the system is fully configured and they are able to begin utilizing the software as intended, however the initial configuration is currently a very cumbersome process. For each TraceGains customer, the initial configuration phase can take anywhere from two weeks to six months due to the fact that everything about the software is fully configurable and can be completely tailored to fit the customer’s business needs. There is currently no defined user interface to assist with the initial configuration process of a customer’s suppliers and ingredients, which often takes about 33 percent of the time for an initial configuration. The vision for the project is that a new user interface can be designed and implemented to satisfy these needs and ultimately give the customer a way to configure their own suppliers and ingredients. If a user friendly interface can be designed to assist in the completion of the project, this will likely free up several weeks of time for the internal staff to allow them to assist with other configuration tasks or customer training. This particular configuration utility is a small piece of the long term goal that will ultimately allow each customer to completely configure their own site.**

1. **Requirements**
   1. **Functional Requirements**
      1. **The utility must provide an interface for configuring suppliers, ingredients**
      2. **The utility must provide an interface for associating suppliers and ingredients**
      3. **The utility must provide a way to upload suppliers, ingredients and supplier/ingredient associations from an Excel spreadsheet**
      4. **The utility should provide the ability to associate multiple suppliers and ingredients at the same time**
      5. **The utility must provide a way to delete suppliers, ingredients and supplier/ingredient associations**
      6. **The utility should allow for inline editing of supplier and ingredient information on their respective pages**
      7. **The utility should not commit any data to the database (including data from an Excel spreadsheet) until the user performs a commit**
      8. **The utility should provide a method for exporting suppliers, ingredients and supplier/ingredient associations to an Excel spreadsheet**
      9. **The utility must not allow for duplicate suppliers or ingredients to be created (based on hID)**
      10. **The utility must not allow suppliers and ingredients to be associated more than once**
      11. **The utility must not allow for suppliers or ingredients to be deleted if they are part of any supplier/ingredient association**
      12. **The utility should provide sorting, filtering and paging capabilities**
   2. **Non-Functional Requirements**
      1. **The utility must interface with the existing security and login framework**
      2. **The utility should utilize the standard page template present in the existing web application**
      3. **The utility must run under IIS 7.0**
      4. **The utility should be written in C#**
      5. **The utility must store data in the existing database with no schema changes**
      6. **The utility should be fully supported on all IE7 – IE10 browsers**
      7. **The utility should be fully supported on a 1024x768 resolution screen**
2. **System Architecture**

**The architecture, which was largely determined by the existing internal architecture, is outlined in Figure 2.1 of the Appendix. TraceGains customers can access the application login page from any web enabled device that supports an Internet Explorer browser and has a resolution of at least 1024x768. As the company has realized over the past couple of years the food manufacturing industry is often very slow to adopt new technologies and as such, many customers access the application on Internet Explorer 6.0.**

1. **Technical Design**
2. **Design & Implementation Decisions**

**Throughout the course of the project, many of the design decisions were made knowing that the new utility was required to interface seamlessly with the existing internal infrastructure. In order to facilitate ease of use and rapid integration with the existing implementation process, no changes were allowed to the existing database schema or existing code base. There were also a number of existing software support agreements between TraceGains and their customers which specified supported browsers and screen sizes. Ultimately this meant that the new utility was allowed to utilize any existing functionality and was required to act as a standalone module to be interfaced with the existing software in order to enhance the user experience. These restrictions significantly limited a number of design decisions and also added a number of interesting implementation issues along the way.**

**When deciding what languages to use for the project, TraceGains explicitly stated that everything in the user interface and front end database access layer was to be written in C# using either ASP.NET controls or Telerik controls as necessary. This requirement was primarily due to the fact that all software solutions were required to run under IIS on a Windows Server instance and in part because all existing software was written in C# using either native Microsoft components or third party Telerik components. In order to maintain 100 percent compatibility between the applications and allow the rest of the development team to maintain the utility moving forward, it was best to maintain a common set of languages across both. For the same reason, the requirements for all database development were that all database queries were to be written in T-SQL and interface with the existing SQL Server instance.**

**Based on the requirement to use C# as the primary language, I still found that there was a significant amount of flexibility in the design decisions relating to the new utility. One of the first major decisions was whether or not to use an object oriented approach for the project. Most of the existing application was written in a non-object oriented manner where each page was a class and encapsulated all of the necessary code. In general, there were no objects being passed between pages and raw data was generally manipulated rather than performing manipulation against objects. Ultimately this was due to the comfort level of many of the developers with a non-object oriented approach. While there was a case to be made for remaining consistent, I felt that it was best to adopt an object oriented approach for the project in order to make the code cleaner, easier to maintain and more portable to other platforms and future configuration utilities. Initially I started with a non-object oriented approach and found that it was very difficult to interface the Suppliers and Ingredients particularly when attempting to associate the two. It was must easier and much cleaner to adopt a Supplier object and an Ingredient object and simply provide a mechanism for interfacing the two.**

**Another one of the other major changes that I adopted from the existing application was the presence of a database access layer. In the existing application nearly all of the database calls and database queries were coded directly in to the web page code behind files. While this approach maintains all of the code for a single page in one place, I felt that it was messy, inconsistent and often resulted in a significant amount of duplicated code. When implementing the database access layer I found that it was very natural to build up the necessary T-SQL queries inside of the supplier or ingredient objects and then pass the queries off to a database access layer. Within the SQLClient libraries in C# there are a number of different methods for executing database queries, so I initially found it very challenging to accommodate all of the different scenarios that I would need for the project, but once the functions were implemented, this approach made the database access easy and consistent across all of the new pages.**

**Once a majority of the back end was developed and ready to go, the rest of the development effort was spent meeting the criteria outlined for the user interface. The first design decision relating to the user interface was which set of controls to adopt for maintaining the various web forms. Initially I had decided to use the ASP.NET controls due to the fact that they are native to Microsoft and are significantly faster within a web browser in comparison to Telerik controls. Implementing the data grids was very easy using the standard ASP.NET controls; however a number of the other requirements became too complicated in the time frame allotted without using a set of third-party controls. The number one reason for switching over to Telerik controls was the ability for the user to perform inline editing. The ASP.NET data grid provides a mechanism for editing one row at a time, but requires user interaction in order enable editing on each row. Telerik on the other hand provides an editable grid that is much like an Excel spreadsheet where the user can click in to any editable cell and immediately edit the contents of the cell. After discussing the two options with TraceGains, the management team unanimously agreed that the spreadsheet approach provided by Telerik far outweighed any performance differences in the two controls.**

**By far the biggest challenge that the project presented was maintaining all of the user’s data and updates while meeting the specification that nothing was to be committed to the database until the user performed a commit. This meant that hundreds of existing records could be updated and hundreds of new records could be created in the UI, but never committed. In particular I found that maintaining the data modified by the user in the front end was most difficult across a post-back to the server. Based on feedback from other developers and some research about the recommended approach by Microsoft, I found that there were two viable solutions. The first was to develop a web service that could be called by JavaScript or AJAX which would allow server side processing without a full post-back. The second approach was to rely heavily on session state and ensure that all of the objects were maintained in session state during a post-back. Ultimately I found that maintaining and debugging all of the JavaScript functions necessary to support the web service calls quickly became cumbersome and cluttered, so I decided to rely more heavily on session state to meet a majority of this requirement. The biggest hurdle was ensuring that all of the data in the user interface was committed to session state immediately on a post-back to the server in order to maintain the current state. Once I finalized a method for saving everything in to session state, this problem became quite trivial even during the validation process and the Excel import process as the collection of objects could be loaded from session state and manipulated or added to with ease.**

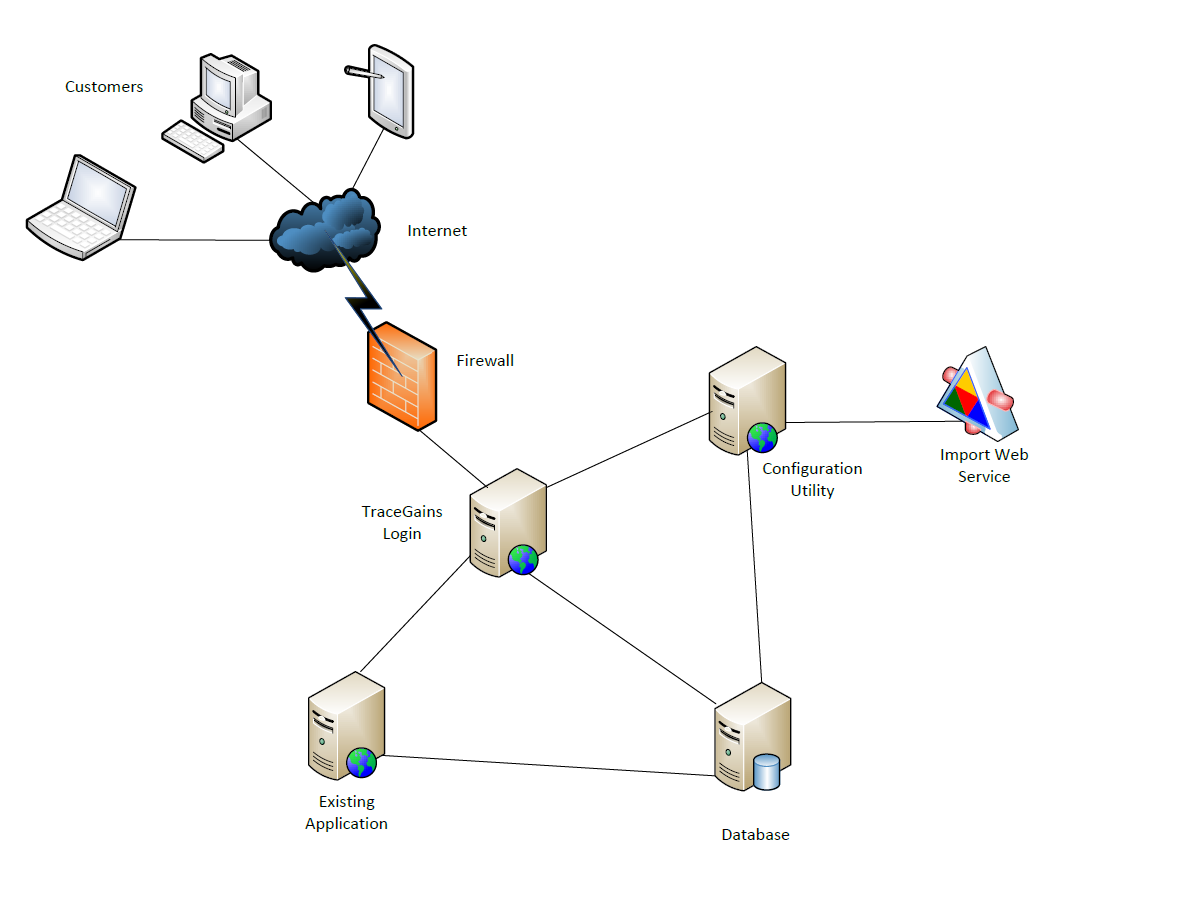
1. **Results**

**At the end of the project all of the functional and non-functional requirements had been met. Combined with other internal TraceGains staff, I spent the last week of the project testing the requirements and functionality of the new configuration utility. The company was in the process of implementing several new customers at the time and utility immediately became critical to the new customer provisioning and implementation process. Due to the fact that this utility was initially designed to assist internal staff the utility was adopted immediately once the QA department had signed off and the utility was released to a production environment. Aside from the basic ability to manage suppliers and ingredients, the feature of the utility that was deemed the most important by the implementation staff was the ability to mass upload suppliers, ingredients and their associations. Over the course of many customer implementations TraceGains realized that their customers, almost exclusively, kept records of their suppliers, ingredients and their associations in a digital format such as an Excel spreadsheet. Ultimately the implementation staff felt that the single largest waste of time during the configuration of suppliers and ingredients was extracting the data from the spreadsheets and manually performing the configuration steps using the customer facing interface which requires that someone manually enter the data for each supplier and each ingredient one at a time. Even though there were often a number of changes that had to be made to the lists of supplier and ingredients provided by the customer in order to match the database schema, the general consensus was that the new utility would eliminate nearly 30 percent of the typical configuration time for a new customer.**

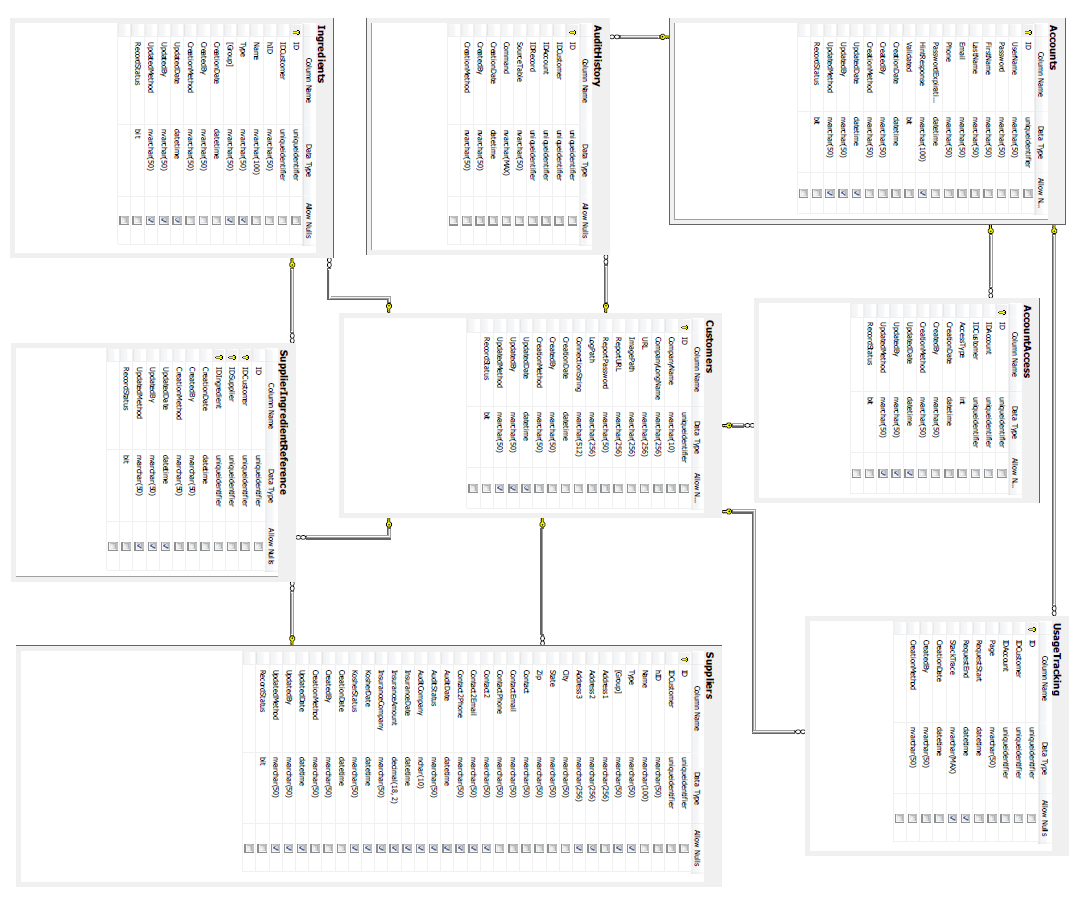
**Due to the immediate impact that the utility had on the internal implementation staff and the perceived ease of use that the utility presented, the goal for TraceGains is to begin providing customers with access to the utility within the next couple of months. Nearly one third of the total development time for the utility was spent on error handling and error checking to ensure that the user was not configuring any of the suppliers or ingredients in error. The software which is maintained and provided by TraceGains is entirely configurable to the customer’s needs and as such one of the main goals of any configuration utility provided by TraceGains is to build in enough of the business intelligence so that the software can prevent any configuration errors.**

**The supplier and ingredient configuration utility was the first of a series of planned utilities aimed at minimizing the overhead associated with new customer implementations for TraceGains. Using the knowledge obtained over the first two years in dealing with food manufacturing companies, TraceGains has acquired a good understanding of the format in which their customers currently store their data and how that data will be provided in order to perform the initial configuration. Based on feedback from the customers and internal implementation staff, the long term goal of the project is to use the framework outlined by this utility to expand support for customer self-configuration utilies. Future plans include support for the configuration of business logic, electronic notifications, user groups, users and customer documentation.**

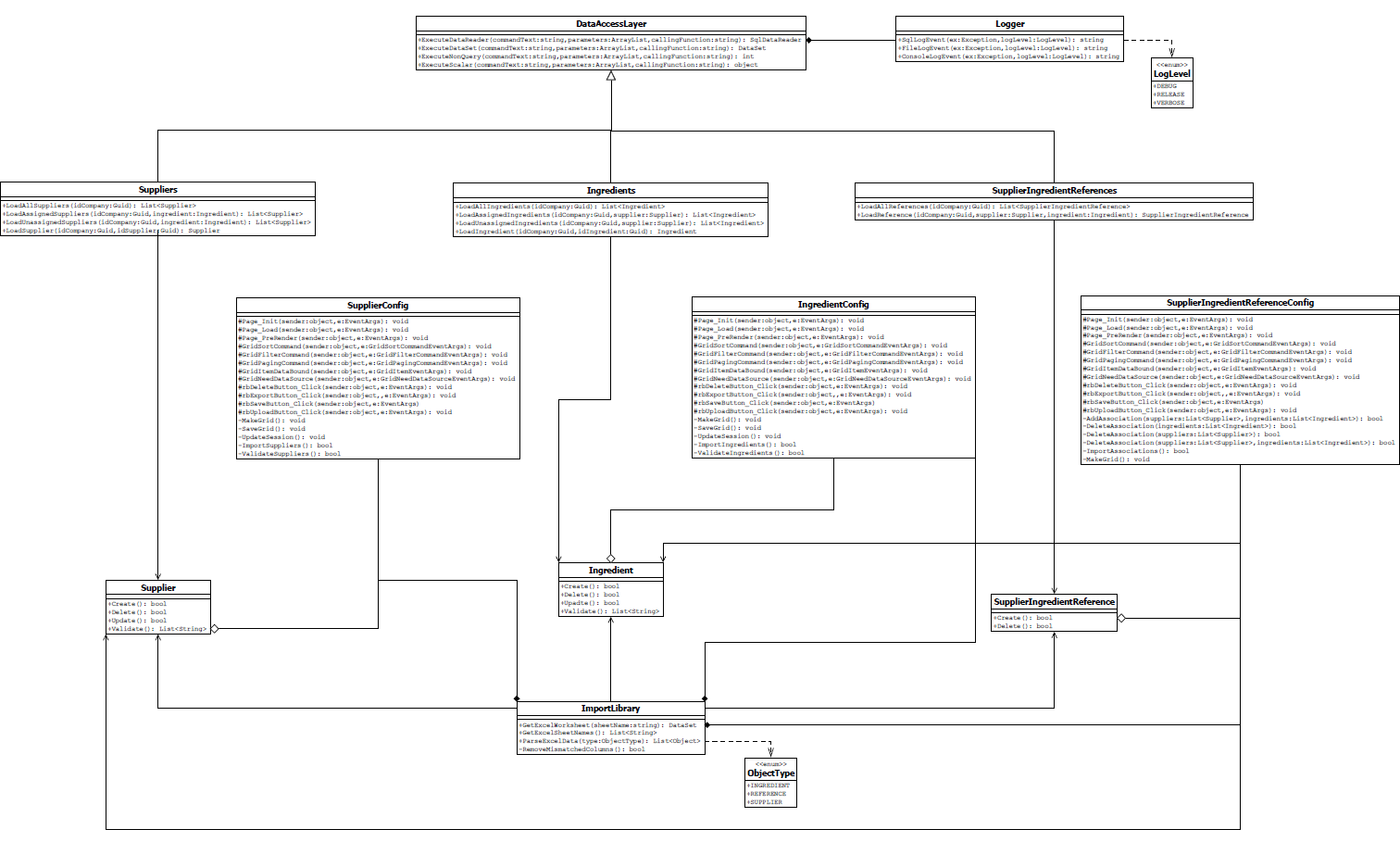
**Appendix**



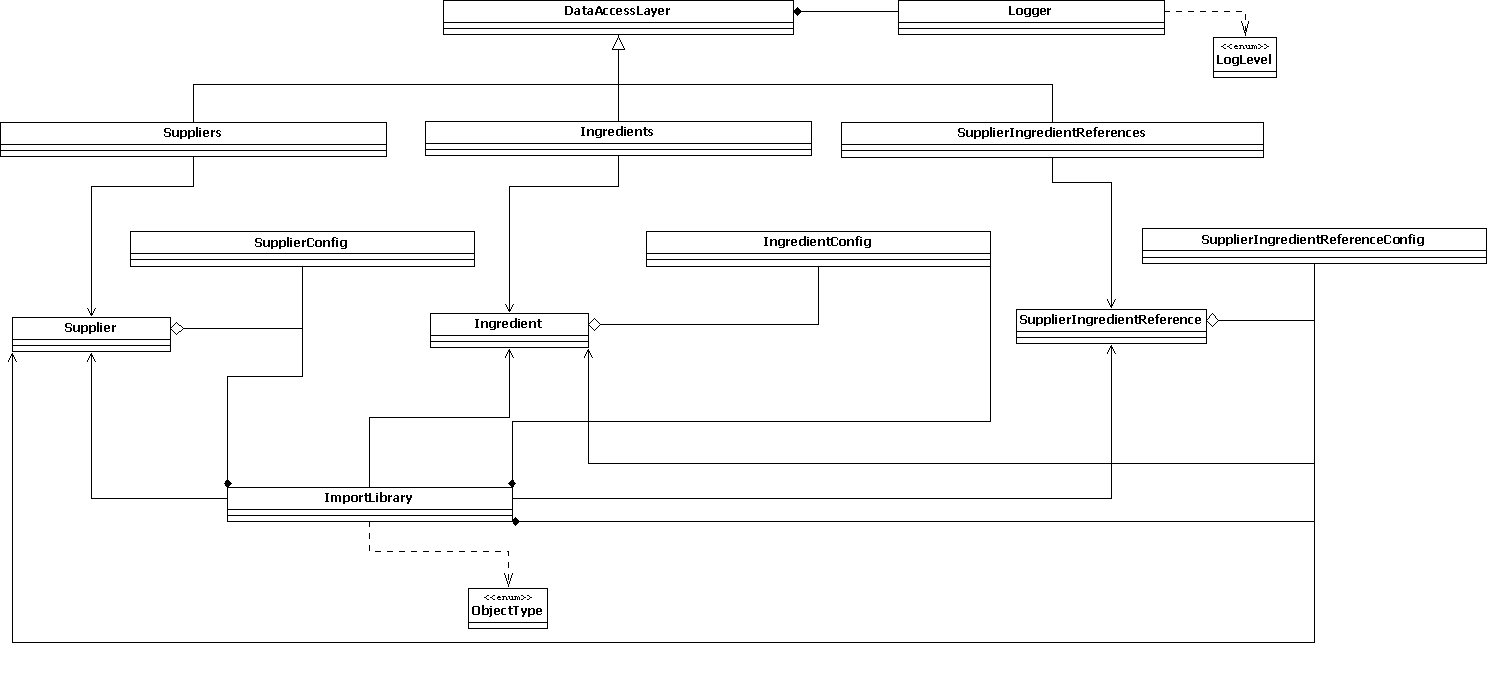
**Figure 2.1: System Architecture**



**Figure 3.1: Database Schema**



**Figure 3.2: Application UML With Methods**



**Figure 3.3: Redacted Application UML**