**Team Fuel**

**Adam Schenkelberg, Lauren Allpress, Michael Brandon, Huong Truong, Aaron Anderson**

**Software Design Document**

1. **Introduction** This document explains the thinking behind the way the implementation was designed. The intended purpose of this document is to give the team overall guidance to the architecture of the software project. Fuel is strongly rooted in database management and is a tool that will allow its users to track the flow of items in and out of the inventory system. The following shows how this is a possibility.
2. **Architecture Description**
   1. Architecture  
       Fuel Donation Management System uses the Model View Controller (MVC) design pattern for the architecture of the application. Team Fuel chose this architecture because it is easy to understand and makes it easy to connect a web-based front end with a model and database on the back end. The popularity of MVC designs in modern web development also contributed to this decision.
   2. Framework  
       Fuel implements MVC using the Ruby on Rails framework so that the developers on the team could reinforce skills from class. In Ruby on Rails, models are created with ActiveRecord and interface between a database and the rest of the application. Fuel uses PostgreSQL for the relational database. Views present data in Rails using ActionView and HTML abstraction markup language (Haml). Presentation of a view is triggered by the controller, which also queries the models and returns data as needed for a particular view. Rails uses ActionController, and interfaces between the database interface, ActiveRecord, and the presentation engine, ActionView.
3. **Decomposition Description**
   1. Class/Module Decomposition  
       Fuel is primarily concerned with three entities: 1) donated items, 2) organizations associated with the food bank where donated items are distributed, and 3) the people and organizations who have previously donated in the past. Based on this, the application can be broken down into corresponding classes for each: Items, Organizations, and Donors with the basic on CRUD functions and sorting function.

This decomposition allows the user to view all items of a particular type regardless of which organization it is stored at or to only view items at that location. It also preserves the history of donors who have contributed even if there are currently no items associated with the donor in inventory. This will allow a user to contact previous donors based on their donation records for a specific item or type of item.

Inventories class will hold information for each kind of items be donated and information concerned that items such as category, expiration date, quantity, organization be delivered and the date be donated (created\_at attribute).

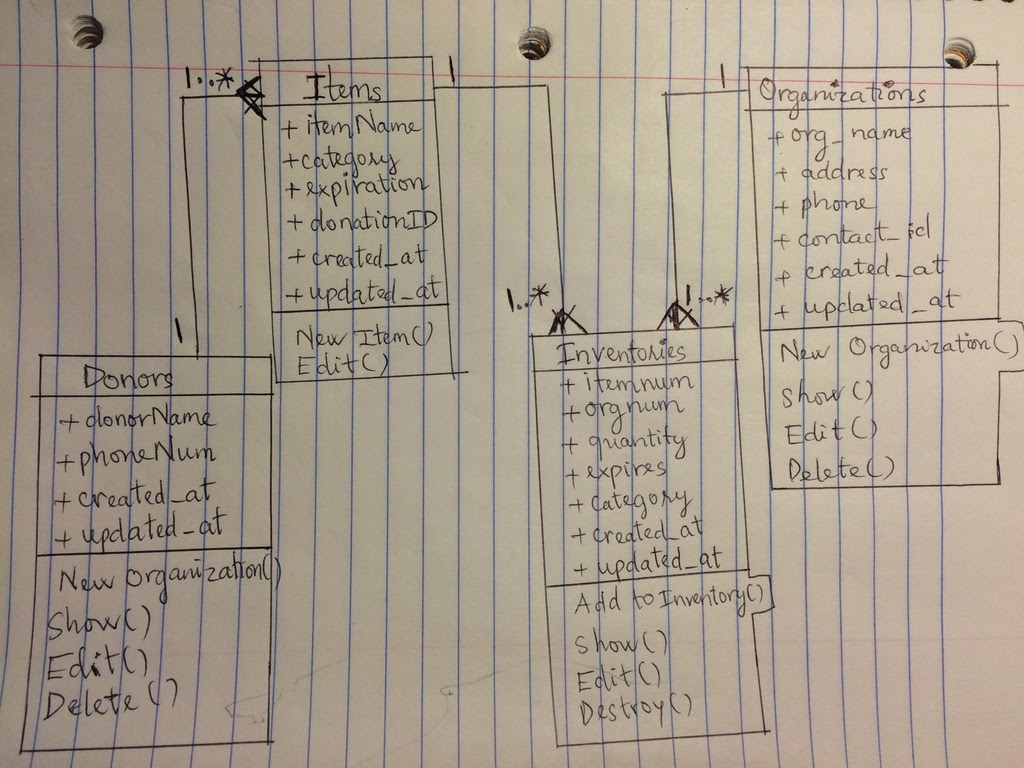
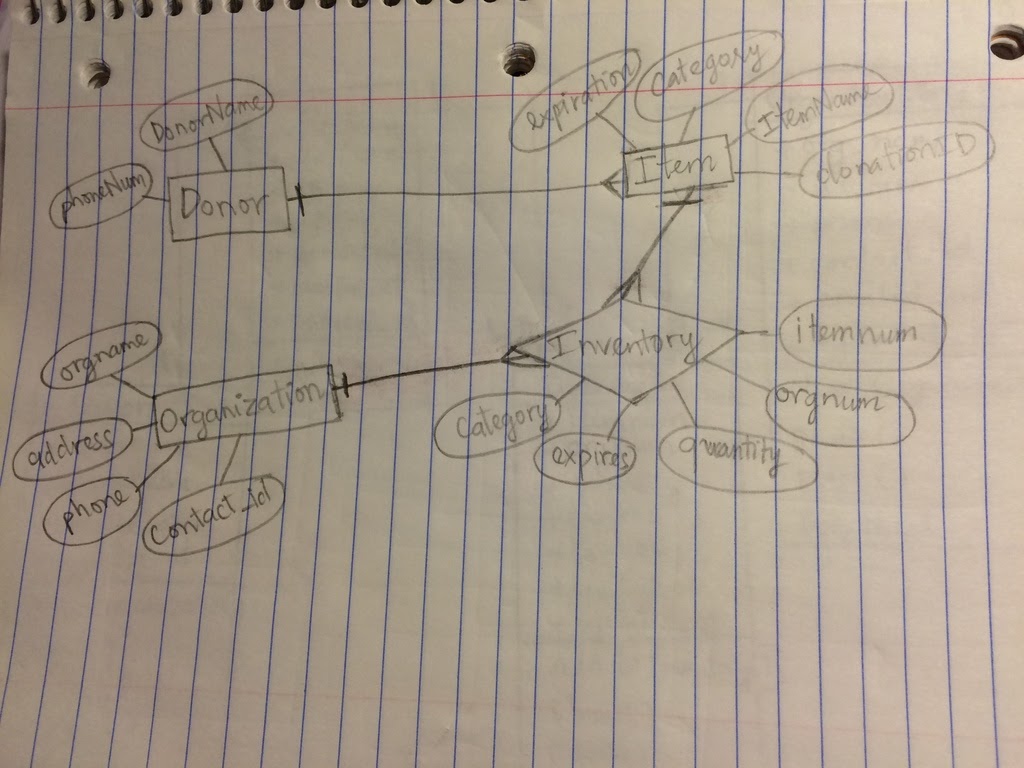


Figure: Class Diagram for Application

* 1. Data Design

****

The data design shown above shows the three entities involved in the software and their relationships. This is a way of tracking what donations have been submitted and who donated them, as well as where would they be stored and delivered during time they were donated till they would be delivered to needed people. To control this system, we created the Donors entities to control Donors’ information and save them in the system as a history records, which Donation Center would needed in future to contact in case of asking for help or send appreciated cards in Holiday.  
 The next, a donor can submit many kinds of items such as canned food, chicken soup, milk, etc. Each kind of food would be defined in Items entity, where the employee can input many kinds of items, specify each name of them and their category with the donor for who donated them.

The Organization entity controls information for who and where the donated items would be delivered. From those Donation Organization’s locations, the items would be delivered to the needed people.

The Inventories entity hold the information for items, their category, their expiration date, and the organization name, location where stores them.

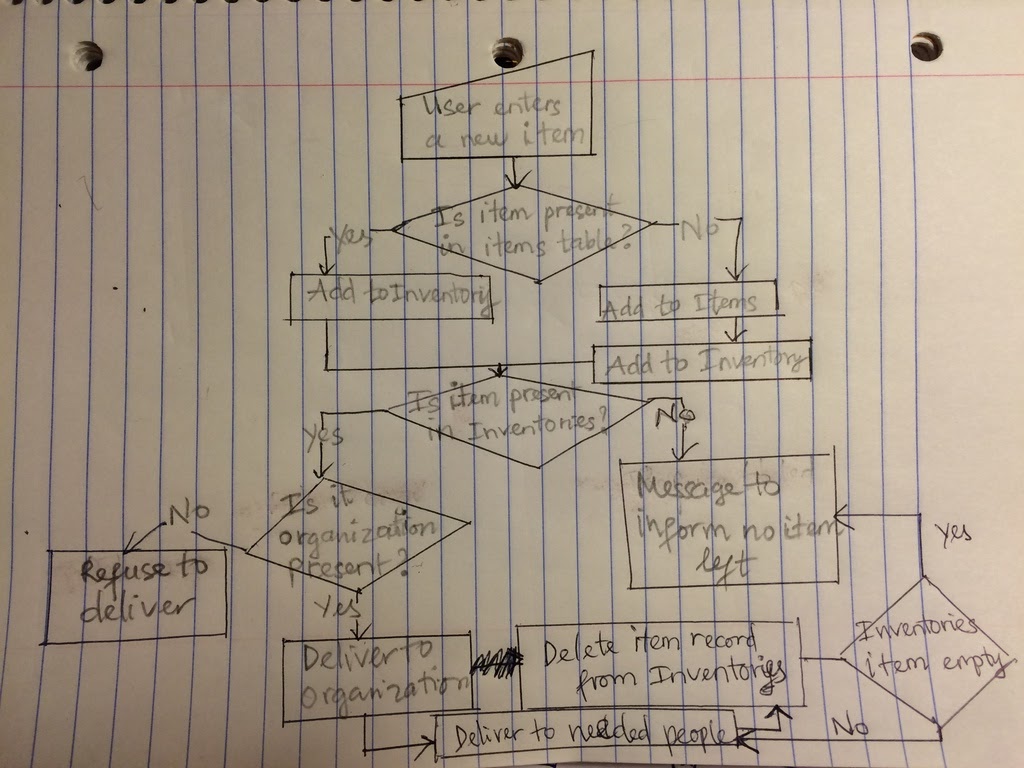
On the theory, if we could success to create the associated tables, the Primary key and Foreign key between tables, then the application would be affected on each time someone deleted the Donors table or Organizations table. There are two ways to saying it. The table would be refused to be deleted by the constrained relationship. Then people need to deactivate the foreign key, then can delete the table. After deleting table, the cascade in database engine updates or deletes the corresponding rows in the child data if you update or delete that data in the parent table.However, our application couldn’t create the association relationship between tables. So, deleting the Organizations and Donors would not affect to Inventories or other tables.

We could store information for Donors by Donors table without using DonorHistory table . The updated model has shown as above. It is important to us a group that we are able to see what locations contain what items. The organizations being able to contact each other or automate a process where an order is put in for items at another location would be a in a future version of this software.

The cardinality of the entities shows that the Item entity and the Donor entity is a one-to-many relationship. One single item can only have one donor but there are many possible donors. The Item to Organization relationship is a little different because the Inventory relationship is a shared inventory. Therefore although one item may only be in one physical location, it can be viewed from many locations and is available to each of those locations. Many Item entities available to many Organization entities.

1. **Dependency Description**
   1. View Navigation

In the view navigation, we start from Homepage, with the three buttons or links which connect to Organization Page, Donors Page, or Inventory Page. From Organization page and Donors Page. we set up a link or a Fuel logo to connect back to Homepage. We also have the button for each function on each page, such as ‘Add’ will connect to a Create New Organization or Create New Donors page. On this new page, we have the button of link back to Organization page and Donors page. With the Inventory page, we have function ‘Add’ working the same way as above, the ‘Delete’, ‘Edit’, ‘Search’ and ‘Save’ work on the the same page. We will set routes for each connect page to direct to the page we link to.

* 1. Process Dependencies  
     Fuel is primarily concerned with allowing users to add, modify, and delete items from an inventory. The action of adding a new item ultimately touches each of the classes in the application if information is provided for both organization and donor. Since the entities in the Inventory table use the Item ID generated by the Items table, the system first checks if the item is present in the Items table. When a user adds an item, the organization and donor are both pre-populated dropdown menus so there is no need to check their respective tables before adding to inventory. After the item is added to.
  2. Data Mapping  
      The use of Object Relational Mapping (ORM) simplifies the use of databases in applications. It does this by using objects to hold database records and making one class represent one table. The objects of these classes correspond to the rows of these tables. The attributes of the object correspond to the columns of the row. ActiveRecord is the base class for models in rails. ActiveRecord implements the ORM and examines the database schema for a table and makes appropriate attributes and methods available in the class automatically. The CRUD method is Creating a new record in the table, Reading in the attributes, Updating the table, and Deleting any necessary rows.  
      Migrations in Rails provide a framework that makes the schema evolution much easier to manage. If we assume from the start that the schema will go through a series of versions, a migration provides code that will update the database schema from one version to the next, and back again. Rails keeps track of the database’s current version and will apply appropriate migrations to switch to any other version. Use migration to create database schema not SQL.

1. **Interface Description**
   1. Please refer to generated RDocs located in the “rdoc” directory in the Submission folder or at this link: [RDoc](https://drive.google.com/drive/folders/0B51sAMIZWATyLXlXOF96eGxFTVU?usp=sharing)