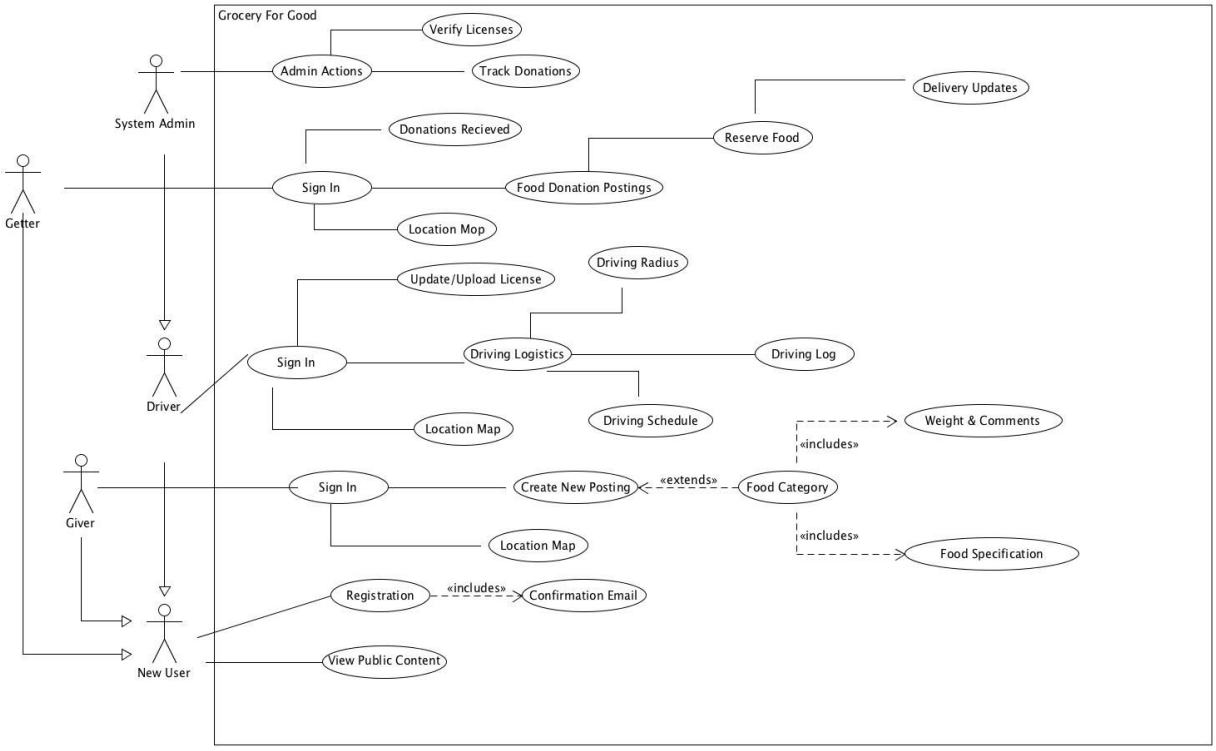
Rebel Scrum: Design Document addressing **Groceries for Good**

1. **Introduction**
    a. Date of Issue
        i. March 14th, 2016
    b. Context
        i. 48 million Americans live in food insecure households, while 1/3 of the food the United States produces is thrown out, 10% is from grocery stores and supermarkets. This is mainly due to consumers skipping over the cosmetically-challenged perishables as well as overstocking to produce a more desirable section of fruits and vegetables. 20% of perishables are rejected by grocery stores and more are discarded after sitting on the shelf for some time. There are no effective delivery systems in place to pull these items for those willing to eat them on a regular bases.
    c. Scope
        i. The project's scope is to provide a detailed understanding of what's to be implemented on the client and server end of the product. As this is a real time application, clients will be notified as soon as other clients send updates to the server. Similar to the networking concerns of a realtime multiplayer game, Groceries For Good needs to act fast to get drivers on the road participating at their convenience.
    d. Authorship
        i. Jeran Ulrich
        ii. Matthew Helms
        iii. Alfonso Juarez
        iv. Kevin Hewitt
        v. Jason Malabed
    e. Change History
        i. March 14th, 2016
            1. Conception
        ii. March 20th, 2016
            1. Added detailed designs
            2. Fixed issues with original diagrams and writing
    f. Summary
        i. Groceries for Good partners with local grocery stores willing to donate goods they would otherwise throw away, to put it into the hands of the less fortunate through a volunteer based delivery system. Volunteer drivers are notified through a mobile app when shelters reserve food from local grocers' in need of delivery. Grocery stores, individuals, and volunteer organization all get to participate in the act of sharing a meal with Groceries for Good.

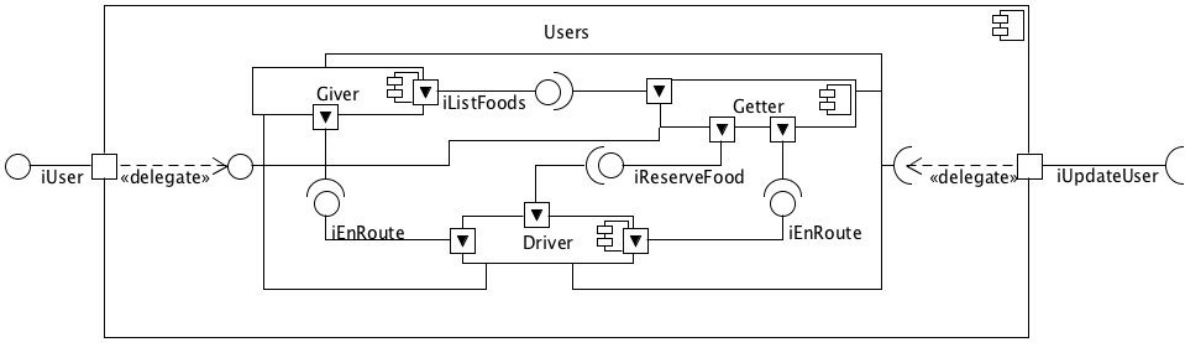
2. **Software Architecture**

a. Overview
    i. Groceries for Good implements an event driven, asynchronous architecture. Since we can imagine many users using the product at once, requests to the server need to be given their own thread, while TCP packet transmission will ensure the order and reliability of packet delivery. This architecture is similar to that of a multiplayer video game, except TCP was chosen over UDP since packet transmission reliability was a primary concern with this product. Since three distinct roles are used in our product, we figured each role must inherit a base user role. Also, when events are triggered by one user, all other users in their transaction are notified as well. This two way server-client transmission channel will be facilitated with web-sockets on our application server.
b. Stakeholders
    i. The primary stakeholders of the system include our customers, partners, and volunteers. The stakeholders apart of the development include the developers, analysts, and managers involved in designing and implementing the software. Other stakeholders include our advisors, partner non-profits, investors, cloud-hosting systems, and DNS lookup providers.
c. System Design Concerns
    i. Real time networking concerns - everyone needs to be aware of each other's status. Packet order is important while return packets are also necessary to ensure packet delivery.
    ii. Integrity of the driver and the confidence we have for them to deliver the goods
    iii. Integrity of the grocery store to accurately log the donated goods
d. Viewpoint 1: Use Case - User View



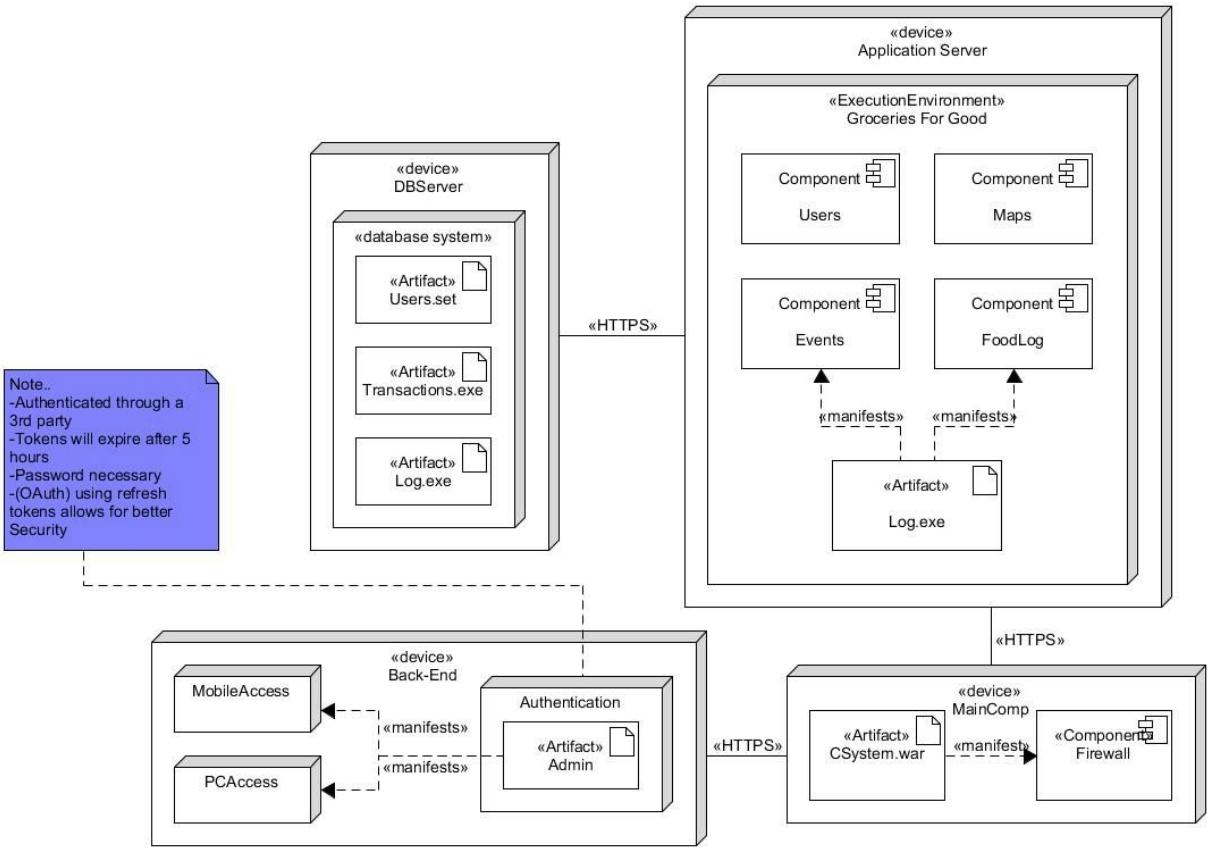
I. Above is our proposed use-case requirements diagram, broken down into 5 different levels of permission and access controls. Access levels are defined by our user types, System Administrator, Guest, Giver, Getter, and Driver. Systems Admin has total operational abilities, enabling them to oversee all pages. Guest capabilities are what a first time user/unlogged in user can view; a home page where they can register for one of the three following roles. Giver, Getter, and Driver are the three major components of our programs model, and each solely have access to their prospective, required pages. All three of these user types have access to a map, of the location of the other user types. The Givers page is optimized for ease of donation, a donation listing page where they can enter the kind and quantity of their donations. The Drivers page has a location for them to upload their valid driver license to be verified, and once they are an approved driver, they can then set the radius of their preferred driving parameter, driving schedule availability, and the total numbers of meals they have delivered. The Getters page highlights the listings of food available, real time order information, and the number of donations their location has received.

E. Viewpoint 2: Component Diagram - Package View



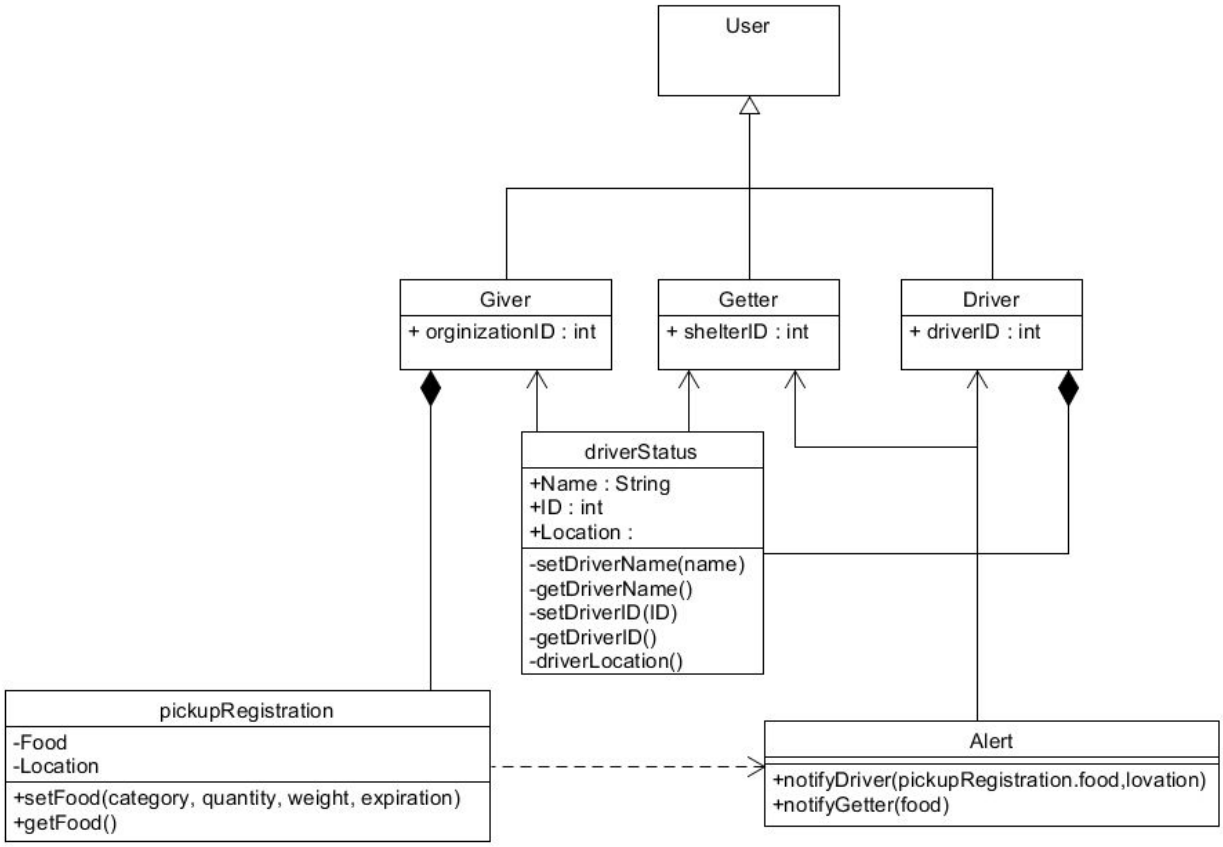
I. The users component provides a simple interface to update (iUpdateUser) and select users (iUser). Our three core user components are divided into separate pools and are highly dependent on one another. Givers will list their donation (iListFoods), which will notify a Getter who can reserve the food (iReserveFood), which will then alert an available Driver in range of the donation for pick up. Once a Driver has agreed to pick up the donation, the Giver and Getter will be notified that a Driver is enroute (iEnRoute) delivering the goods.

F. Viewpoint 3: Deployment Diagram - Physical View



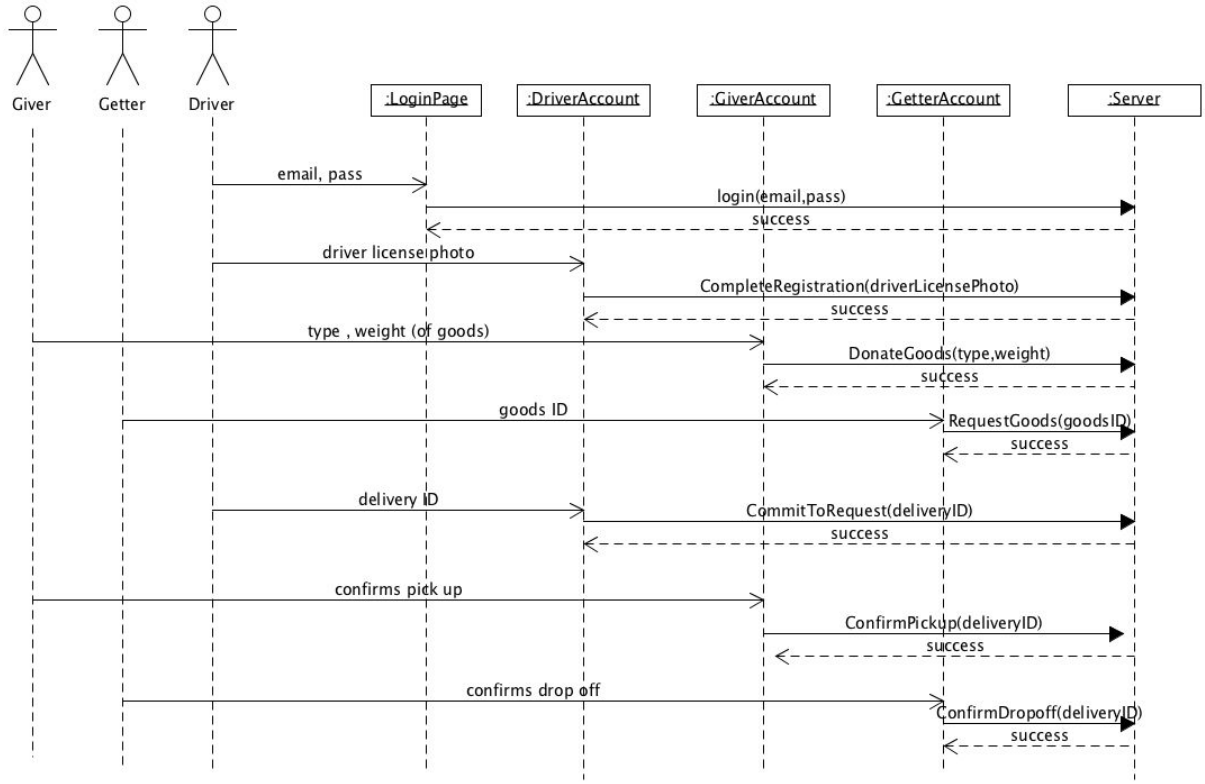
I. The most important aspect to the deployment diagram is the security of the server. As stated within the note, the Authentication device handles most of the security. It prompts the system so that the user must always provide a email and password, the tokens will expire after 5 hours, etc. Within the Back-End block, the admin artifact manifests MobileAccess and PCAccess to allow admission into the system, then it’s connected to the MainComp.The final part of the defense system is the through the MainComp which holds the firewall. As for the other aspects of the diagram, there's a HTTPS connection between the Database System and the Application Server so that the components within the system can have access to and alter any data as need be.

G. Viewpoint 4: Class Diagram - Logical View



I. The actions carried out by our program are relatively simple. Its core systems will compose of a database, but the active components will work as follows. The biggest effort is executed by coordinating the grocery stores and drivers. A class handles the input of info by the grocery store, and then alerts a driver of the pickup needed. The info from the store is also stored into a food log. After the registration class and alert class have activated, a “driverStatus” portion of the program will update the grocery store and the shelter as to the basic information of the driver until the delivery is completed.

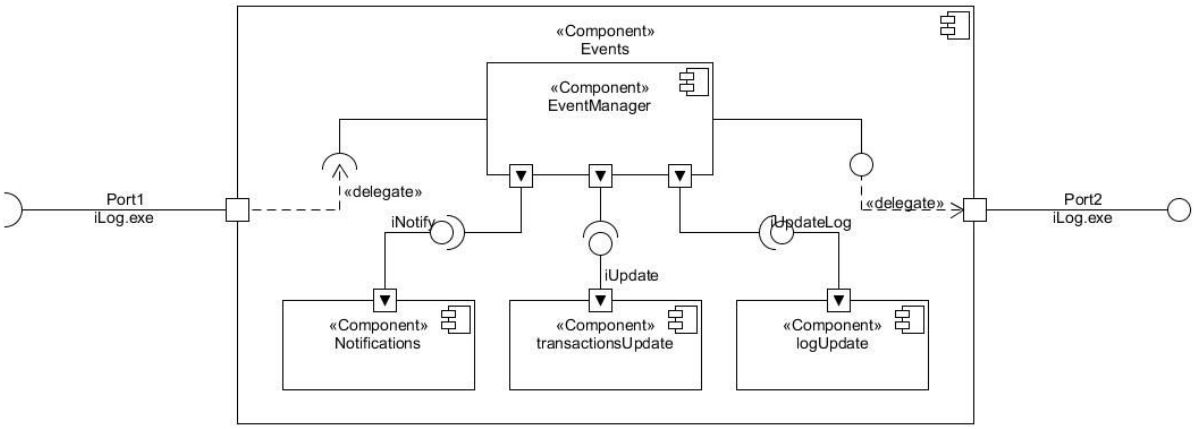
H. Viewpoint 5: Sequence Diagram - Process View



I. The sequence diagram is from the perspective of the driver and contains the methods involved in the front-end application of their product. From logging in to committing to a delivery, each role will be able to fully interact with the application on their account page alone. Since each role has a unique function in the system, the account page will look different for each user. Our system is implementing an event driven structure, as users are primarily posting to the server, the server will then push events to the users in need of notification, as seen in the detailed activity diagram below.

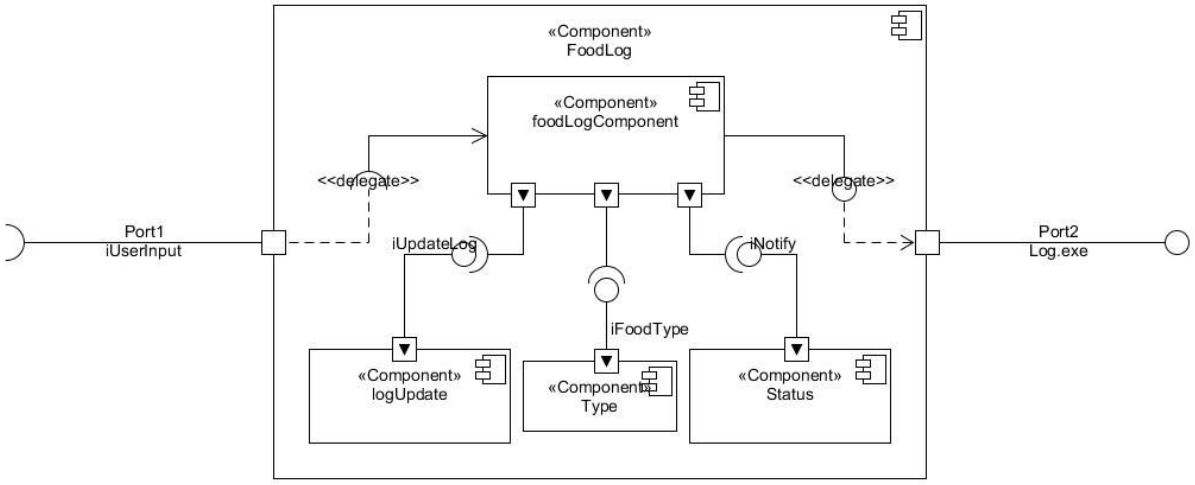
Detailed Designs

I. Events Component (Detail of 2F)



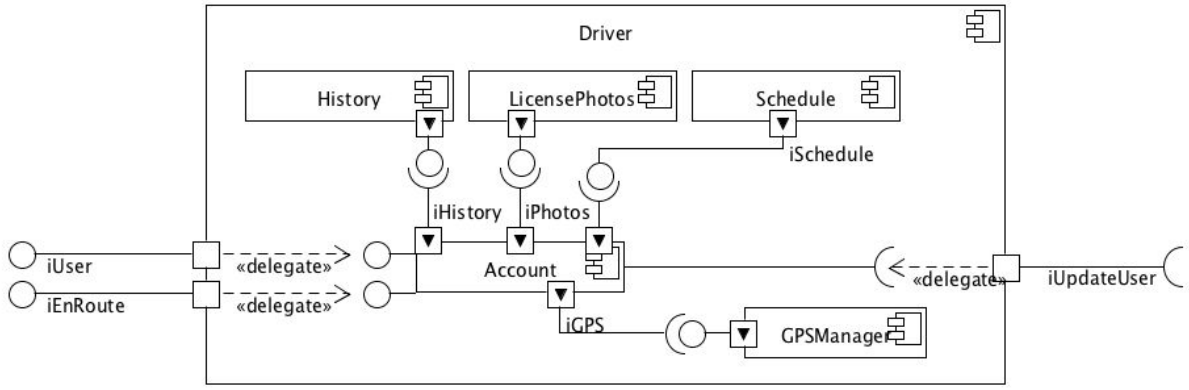
Above is a more-in-depth version, detailed design, of the Events Component that is shown in the Deployment Diagram. For this particular component, the input port draw from and adds to the same location, the Log.exe shown in the Deployment Diagram. Within the EventManager, Notifications and the updating of the Transactions.exe and Log.exe are handled. After the events are successfully processed and sorted and the users are notified, a new updated version of the Log.exe is output into the system.

J. FoodLog Component (Detail of 2F)



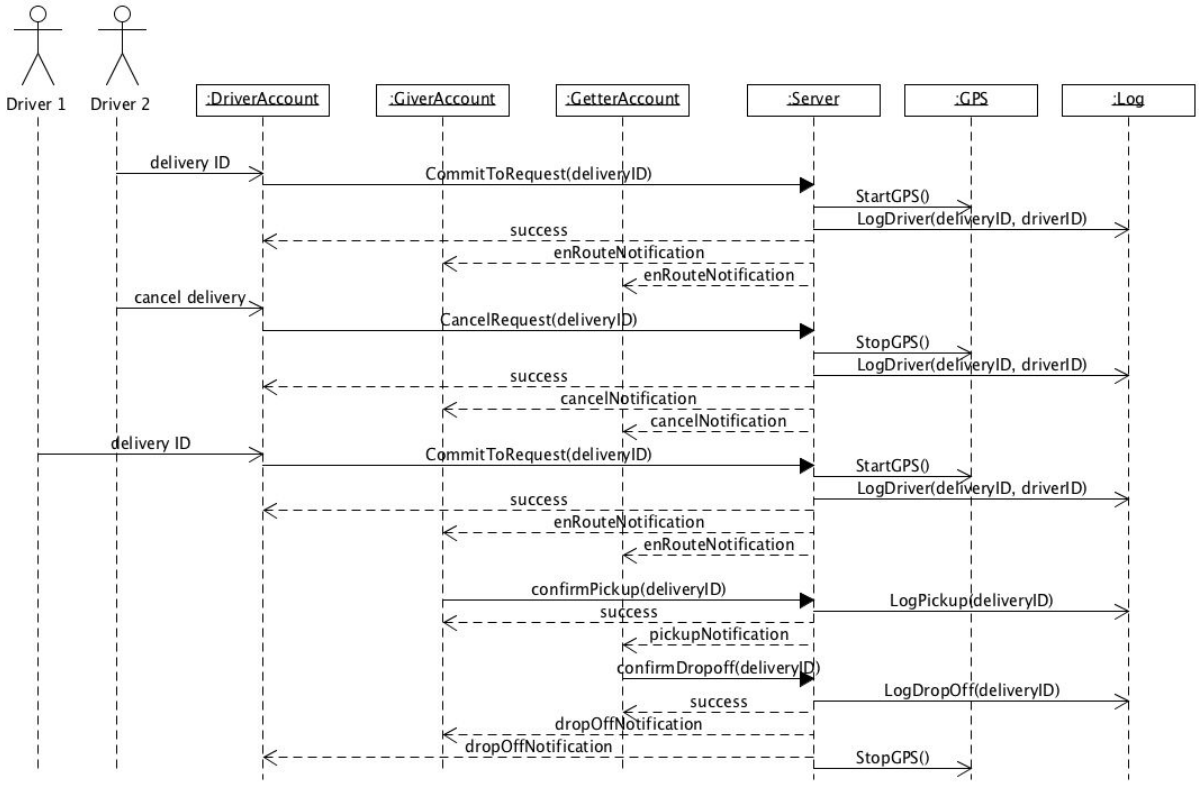
Above is a detailed design of the FoodLog Component that is shown previously in the Deployment Diagram. First off, the component itself has 2 ports to it, an input and an output. The input is given by User Input. The user(Giver, Getter, or Driver) will update the FoodLog quite oftenly, allowing the status(Status Component) to change accordingly to the location and process the food is in(whether in transit, available for pickup, or taken) and the type(Type Component: Fish, Meat, Dairy, Vegetables, Fruit, ets.) of the food it is. After everything is updated and ready, the component outputs an updated version of the Log.exe artifact, through the output port, shown in the Deployment Diagram.

K. Driver Component (Detail of 2E)



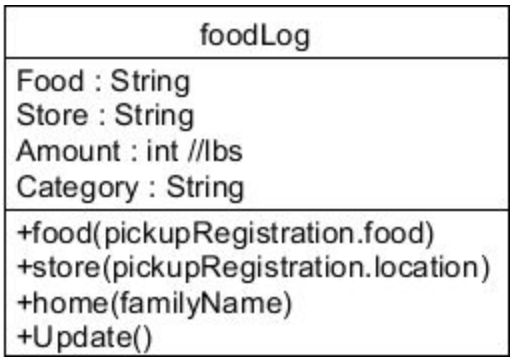
The driver component contains a history, schedule, gps, and license documentation components. The driver component is the only user component that contains a GPSManager, since it's the only user role that requires location information. The driver component provides its own public information as well as providing status updates to the other components in the parent Users component.

L. Driver Sequence (Detail of 2H)



This sequence diagram illustrates a scenario if a driver cancels on a delivery before picking up their requested goods. Driver 2 cancels the delivery and the others are notified, and driver 1 see's it has become available again and commits to the request. In this detailed diagram, you can see how the server acts with the GPS and logging (database) system located within the server.

L. foodLog class (Detail of 4G)



The detailed diagram of the class "foodLog" illustrates the functions and variables used to store information into a database containing the statistics of the food being donated. The class collects data taken from the "pickupRegistration" class, then adds it to the database through the "Update()" method.

Resources Used

Sommerville, Ian. Software Engineering. Boston: Pearson, 2011. Print.

Braude, Eric J. Software Design: From Programming to Architecture. Hoboken, NJ: J. Wiley, 2004. Print.

"Design Patterns and Refactoring." Design Patterns. SourceMaking, n.d. Web. 21 Mar. 2016.