Food Trucks Fight Hunger - Requirements

Group 24

Antonio Bermudez  
Aaron Berns  
Wenwen Dong  
Steven Nowicki  
Hunter Schallhorn

### Functional requirements and non-functional requirements in terms of the environment

Functional requirements

Hardware

* Internet connection.
* The UI should be accessible from a desktop/laptop, a mobile device, or a smartphone.
  + The connecting device needs an input component, allowing the user to type and fill out forms.
  + The device needs an output, which can display the UI.
* On desktops/laptops the UI can run on all major web browsers: Chrome, Safari, and Internet Explorer.
* Smartphones must download an app to use the system.
* In addition to new messages from other users being displayed in the UI, the connecting device should be able to make a noise or vibrate when a message or notification is received.
* The connecting device must be able to communicate with the distance calculation API by allowing the user to input their location or through use of the device’s GPS location.
* If the connecting device is a desktop/laptop, the user may want a separate device with a phone to make calls.

Software

* A relational database is needed to store information on food vendors and food banks. Tables and fields are TBD. This could be implemented in MySQL, Postgresql, or some other RDMS.
* A UI library is needed to create the user interface. A Javascript based library would make the most sense, as it is the most commonly used language on the web.
* A back end language to communicate between the database and the frontend UI is needed. Something like Java, php, or nodeJS would work.

Non-functional requirements

* The system should not take up too much memory when running in a browser or on the mobile app. Users should be able to maintain normal functioning of their device while using the app.
  + The system should not drain batteries on mobile devices, especially when distance based navigation is being used.
* Smartphones with cameras are not required, but they would allow the user to take photos of food and send the photos to another user. Whether photos are transmitted via Feature 2 or simply emailed is TBD.

### Functional requirements and non-functional requirements in terms of the system's interfaces

**Feature 1. A database of mobile food vendors and food banks/organizations that both sides can access**

Functional Requirements

This will require building a database and a UI that can communicate with the database. The UI should allow the users to do the following.

User can Search

* Food vendor users can search for food banks. Can find out location of food bank, times they accept food, and types of food accepted.
* Food bank users can search for food vendors. Can find vendors to receive food donations.

User Can Post/Update

* Food vendor users can post data showing what type of food they can donate and at what time.
* Food bank users can post their location, donation times, and types of food accepted.

Non-functional requirements

* Search results should reflect the most recent state of the database, in real time.
* Search should allow users to search by criteria. These criteria can match fields in the database tables.
* Posting to the database should require the user to post a minimum field set. This field set is TBD and will differ dependent on user type (vendor or bank).

**Feature 2. Text based messaging system**

Functional Requirements

Parties that are part of the database can message each other, using a text based app. A user should be able to find another user via Feature 1 and then message that user within the UI. This would require building a system that can transmit data from one party to another - or utilizing a system that is already present on the connecting device. This transmission system would also need to be able to interface with the database in Feature 1, in order to look up where messages should be sent and possibly to store messages and contacts for users.

Non-functional Requirements

* Messages should transmit in real time.
* Messages should alert the user through the UI when they are received.
* Messaging should support one-to-one user communication and one-to-many communication. That is, a user can send a message to one user or to many users at a time.

**Feature 3. Automated distance based alerts**

Functional Requirements

The system should notify vendors when nearby food banks need food and notify food banks when nearby vendors have food to donate.

* This will require users to set a “need food” or “have food to donate” flag, in the database. There will need to be an interface between the distance calculation API and the database of vendors and food banks to determine if there is a match between “has” and “needs” that is within the proper distance.
* This system is distance based. The user should be able to set a max distance from their current location in the UI. Current location and max distance should then be updated in the database. The system should then only alert them if there is a “has” or “needs” within distance. The Google Maps API can be used to calculate distance.
* Current location can either be manually set or derived by the connecting device’s GPS.

Non-functional Requirements

* Vendors are mobile food trucks, so the distance calculation API should be dynamic.
* Alerts should only be produced if the food vendor is active or the food bank is open for donation.
* Alerts could also sync with other more specific information in the database. “Has” and “Needs” flags could be more specific. For example, “Has bread” or “Needs fruit”.

**Feature 4. Donation Tracking for taxes**

Functional Requirements  
The system should keep a history of donations made by a vendor. A vendor should be able to access this donation history and generate detailed reports that can be used for tax purposes.

* This will require the system to save information about the donation in its database. The information will have to enter the database via the vendor’s UI. The recipient food bank will also need to ‘approve’ the donation through a UI of their own before it is logged in the database.
* Once a donation is ‘approved’, the vendor can be emailed a receipt containing summary and detail information about their donation. The receipt should be a PDF attachment. Creation of this receipt will require the database to interface with a PDF generation program. Sending of the email will require the system to interface with an email protocol.
* A vendor should be able to use the UI to search through all of their logged donations.
  + Search can use fields related to the donation as filtering criteria, such as date, food bank, or estimated dollar amount.
  + Returned search results should be able to be displayed in the UI at a summary and detail level.
  + Summary and detail level search results can be saved as PDF reports and printed. This will require the system to interface with the connecting device’s file system. Like emailed PDFs, this will also require the system to interface with a software that can generate PDFs.

Non-functional Requirements

* This feature requires the vendor to have a registered email address.
* If PDFs are to be downloaded, the connecting device will need a way to store files.
* If PDFs are to be viewed, the connecting device will need a PDF viewer or web browser.
* The system should be able to retain many years’ worth of donation records, in case they are ever needed for tax purposes.
* Search should return quickly, i.e. within a few seconds or less.

Other functional requirements

* User should be able to register for an account. There are two distinct types of account: food vendor and food bank. The user type selected will determine the user’s UI. I.e., vendors search for food banks and post food they can donate and banks search for vendors who can donate the type of food they need. Registration in the UI should create a record for the user in the database with the appropriate fields.

### Use Case 1

The first use case is a situation where a food truck (vendor) has leftover food and wants to find a food bank that is on their route home and is currently accepting donations. This relationship between vendor and bank in this use case contains an active vendor and a more passive bank. In this situation the bank would have already posted the food needed (see use case #2). Thus, this use case involves the vendor being very interactive with the application but minimal interactivity required between the application and the food bank to schedule a delivery. The vendor would post to the application the food he or she has available along with an optional picture. This information would be stored in the database and kick off a process of locating nearby food banks that meet three requirements:

* Need the type of food available
* Are available to receive the food
* Are on an efficient route home for the vendor

Following this, the application would send a notification to the food banks found that meet the requirements requesting a confirmation from the food banks that they are willing to accept the food. The banks would have a 15 minute window in which to confirm the donation. After the 15 minute window, the application would return to the vendor the food banks that have accepted the food along with an efficient route for the food truck to stop by all the banks and deliver the food. The vendor would then deliver the food on the way home.

App-use flow for the vendor and bank would be as follows:

* End of day is approaching and vendor analyzes food stocks and takes an account of what will be left over.
* Vendor, ideally but not necessarily, takes pictures of examples of leftover food stock.
* Vendor then posts to application each donation type (bread, meat, etc.) available along with picture of each if taken.
* Application stores food in database and determines closest bank that is willing and able to accept to donation.
* Application issues notification to bank requesting confirmation that they can receive food
* Food bank responds to confirmation.
* Application determines most efficient route for vendor to drop off food.
* Application gives the vendor an efficient route to banks willing to take food that have confirmed.
* Vendor delivers food.

### **Use Case 2**

The second use case is the situation where a shelter is currently low on food and needs to broadcast that they are open to taking donations. This use case involves high interactivity between food bank and application and low interactivity between vendor and application. The general idea is that a food bank notices it is low on food. They would then make an account of what is needed and post to the application. The application would then notify vendors in the area of nearby food banks and the foods needed. At this point, the vendors could make decisions about whether or not to conserve certain food items for donation and report their availability at the end of the day (via use case #1). As long as a food bank maintains a “need” for an item, they will be displayed in the app.

The app-flow for vendor and food bank in this use case would be as follows:

* The food bank notices it is low on at least one type of food.
* Food bank posts to the application that they are in need of food.
* After each post or at the start of each day, the application displays banks that need food for the day..
* A vendor can then decide whether or not they want to set up near a needy bank for the day so they can make an efficient donation trip at the end of the day.

This use-case is vital for the operation of the application. If a food bank does not post food needed then vendors cannot be notified of food banks in the area that can accept their end-of-day donations.

### Use Case 3

The third use case would involve a food bank and a vendor who have possibly had a good relationship in the past to communicate and schedule specific dropoffs. In this case, a shelter and a food truck communicate with each other beforehand or during the day to arrange a custom drop off. An example of this would be a case where a vendor has consistently donated to a food bank and the food bank values their donations very highly. The food bank could then personally contact a vendor and discuss what food the vendor may or may not have available and set up a special donation. Or the inverse could occur, where a vendor directly contacts a food bank about possible leftover food and whether or not the vendor would want it. This is a use case a food bank would like to have for a possible situation when they know there will be abnormally high demand for food or if there is a special event (such as Thanksgiving) that the food bank will be giving away food for. Additionally, this could be a case where a food truck perhaps catered for a special event and will have a great deal of a specific type of food left over and would like to contact a shelter about donating it. This use-case thus necessitates a personal messaging system between vendor and food bank where each is notified of a message and can respond as needed. This app-flow for this use case could actually be divided into two sub-cases. The first sub-case would be where a vendor knows he or she will have excess food and contacts the bank. The second sub-case would be where a bank knows they need a specific donation of food and contacts a vendor.

App-flow use for the vendor and bank in the first sub-case would be as follows:

* Vendor knows he or she has excess food and vendor has a good working relationship with a specific food bank.
* Vendor uses application messaging system to contact food bank.
* Food bank either accepts or denies special donations.
* Vendor makes delivery.
* Donation added to database.

App-flow use for the vendor and bank in the second sub-case would be as follows:

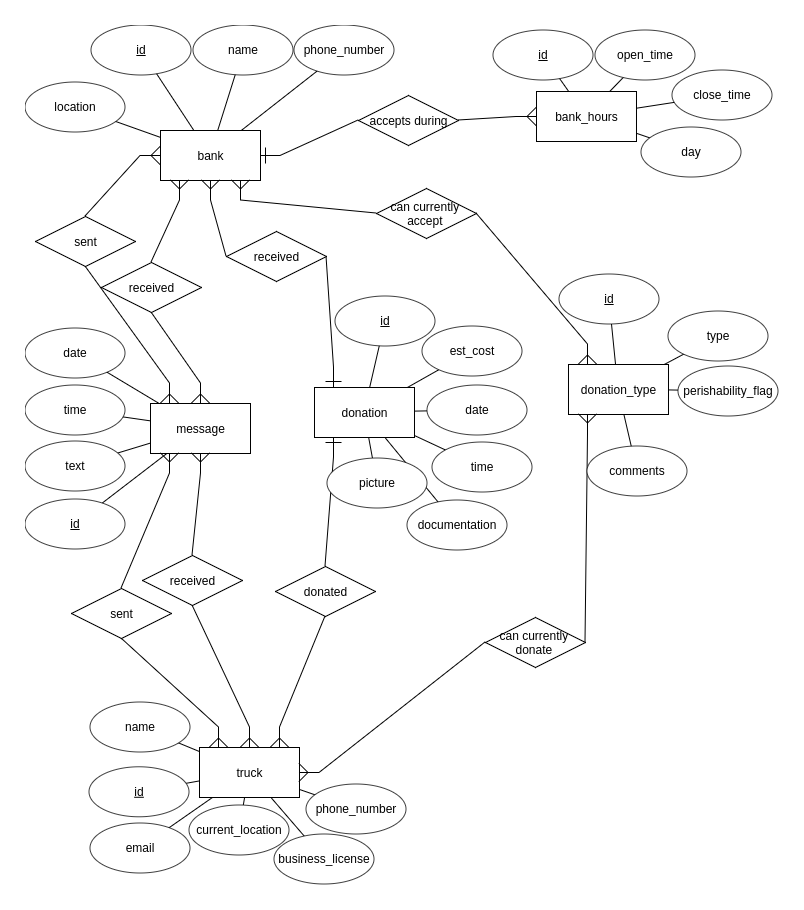
* Food bank knows of special situation in which an excess amount of food or food of a specific type will be required and has good working relationship with vendor.
* Food bank uses application messaging system to contact vendor.
* Vendor either accepts or denies making a special donation.
* Vendor makes delivery if accepted.
* Donation added to database.

### Use Case 4

The fourth use case for the application would involve a history of donations a vendor has made. The vendor would be able to access a complete record of all donations made, where they were made, and what they included with the picture uploads, if they had submitted it. This would allow the vendor to present a record to anyone who needed it, such as the IRS if the vendor wanted a tax break. This use case would require a “donated” relation in the database so that the vendor could easily pull up the information in the application. The application should allow the vendor to specify a date range of donations as well as a specific food bank they donated to. This would allow them to get more specific information. A list would be pulled up based on the search parameters they entered and then a vendor could click on an item in that list. The items in the list would contain basic information about the donation such as date, estimated amount, food bank donated to, and small thumbnail picture if one existed. Once a user clicked on an item, they could see more detailed and specific information such as time and documentation/notes on the donation. A separate option would be available on both the list and specific item view to print out a report with the details shown in each view.

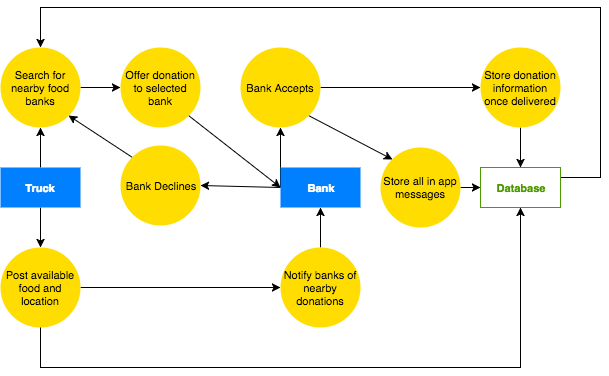
App-flow use for the vendor would be as follows:

* Vendor requests a history of donations made; can specify based on both date and food bank donated to.
* Application returns requested list
* Vendor can print list if desired
* Vendor can select specific donation for more detailed view
* Application returns specific donation with all detailed information
* Vendor can print detailed view as report

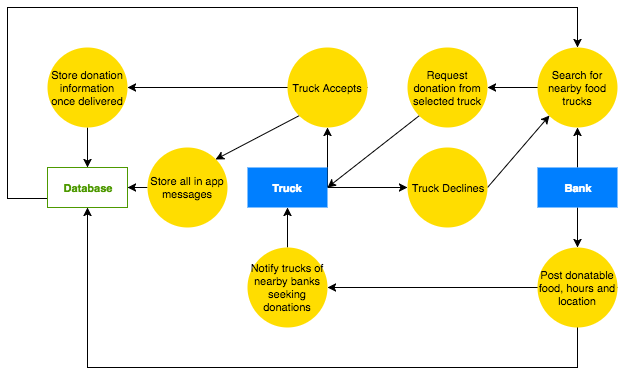
**ER Diagram**

**Data Flow Diagram**

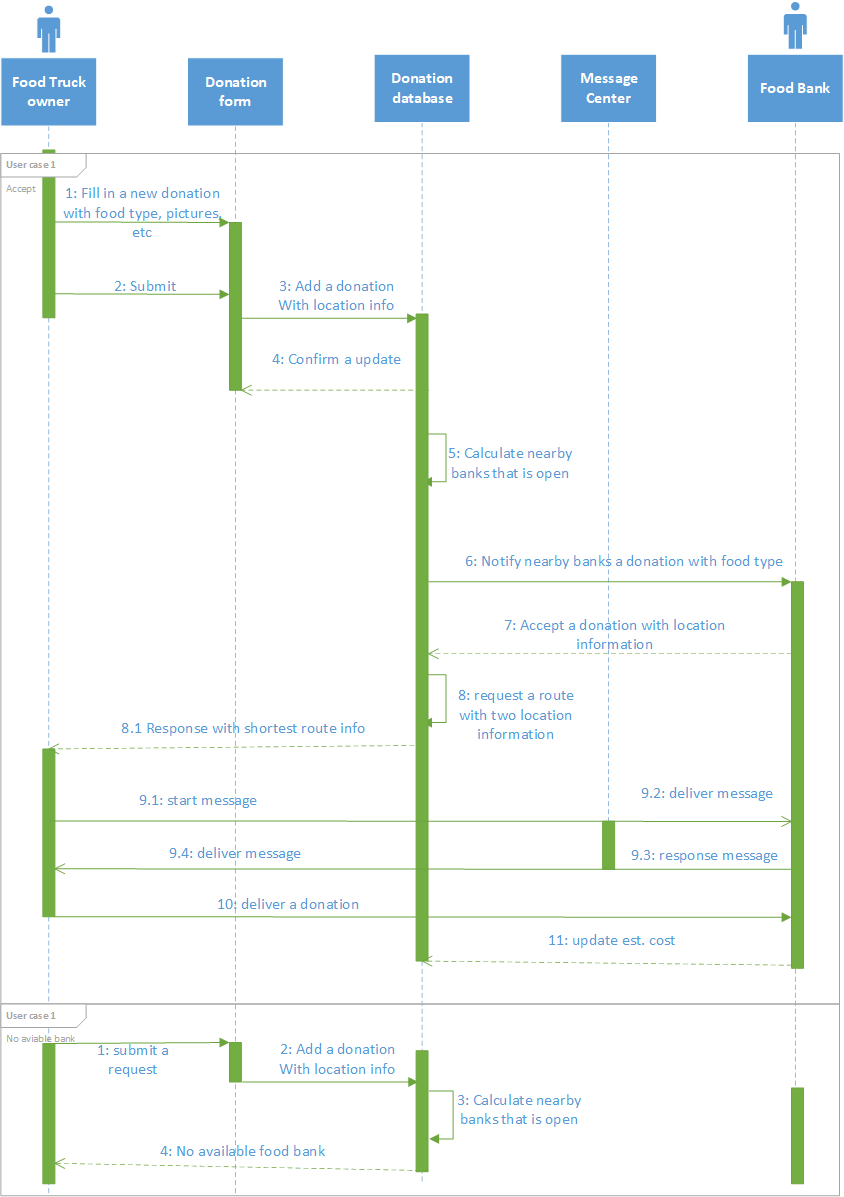
As a Food Truck user:



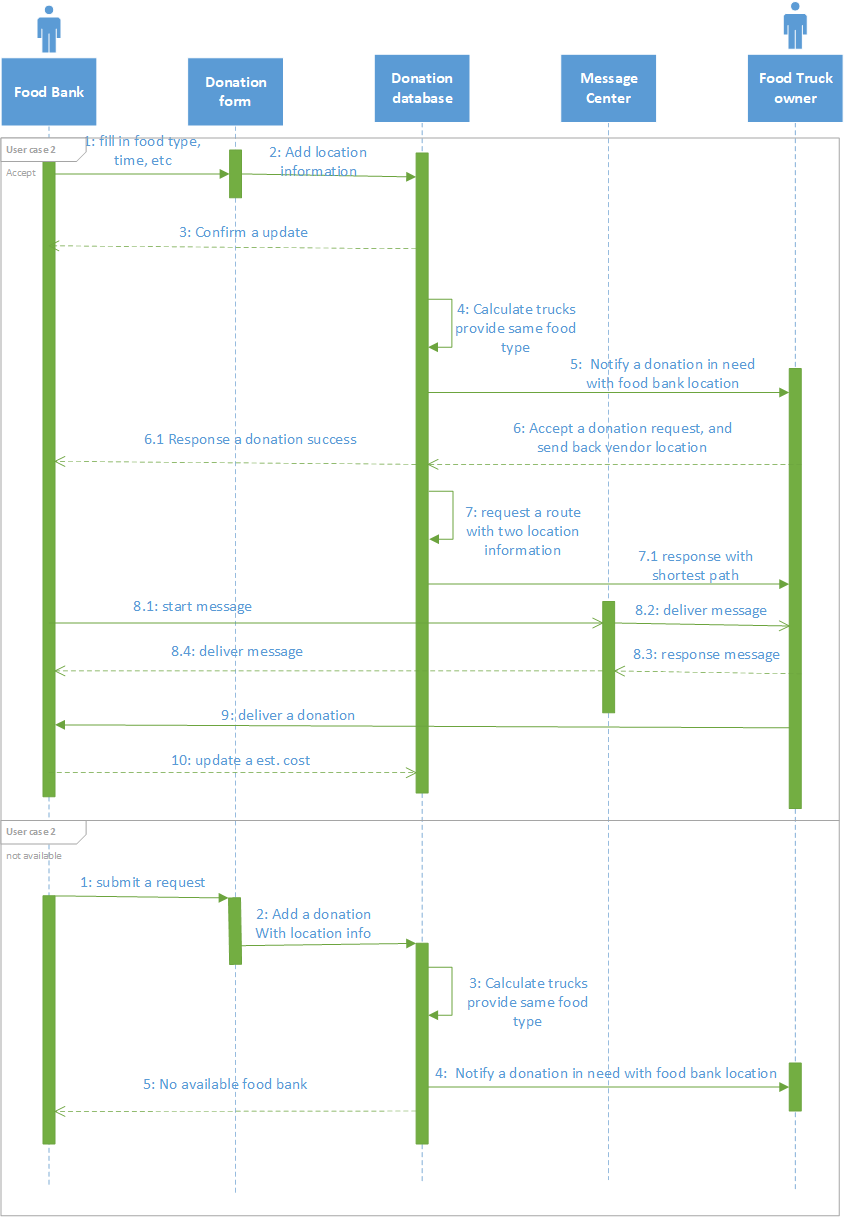
As a Food Bank user:



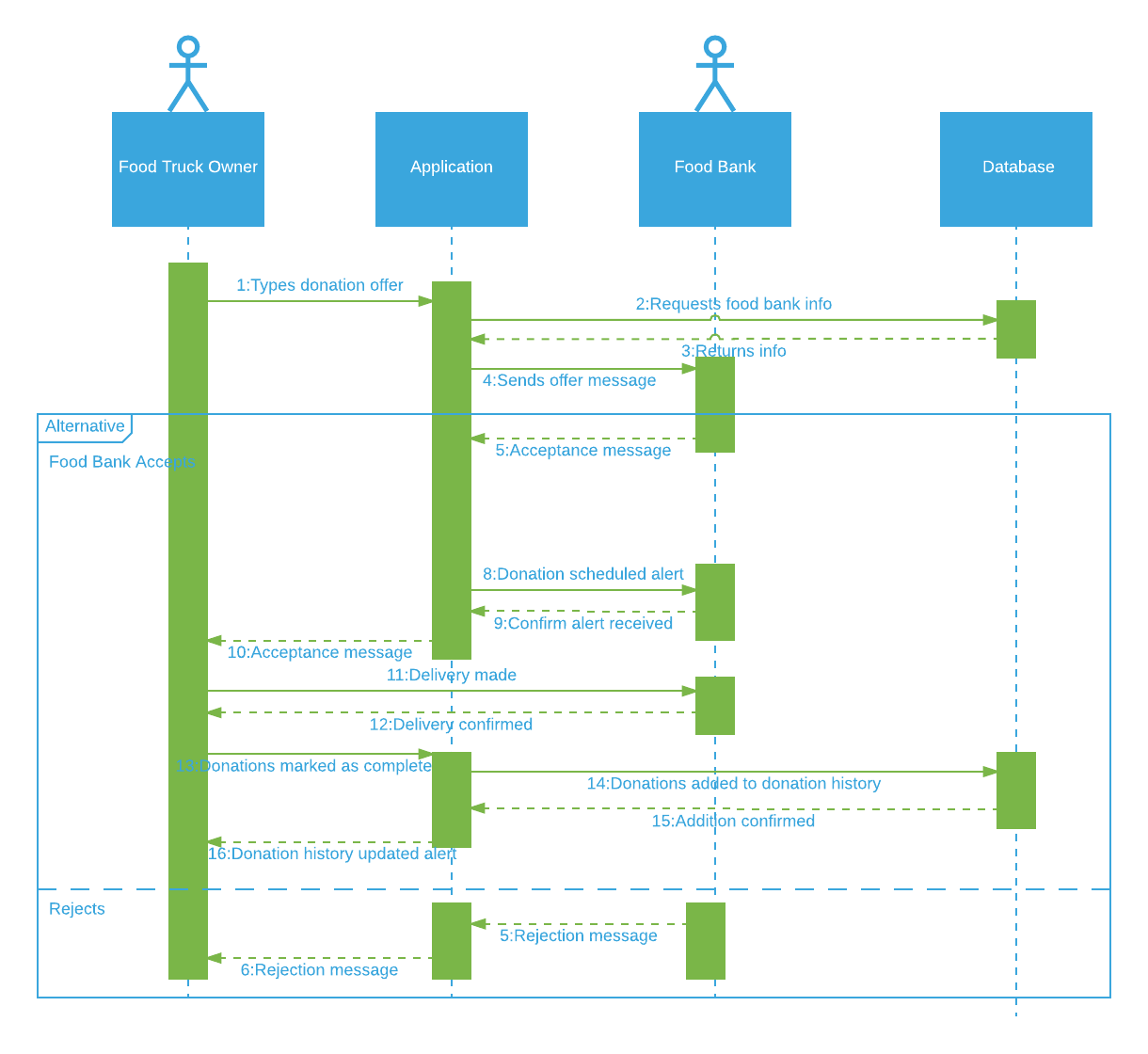
### Use Case 1 Sequence Diagram



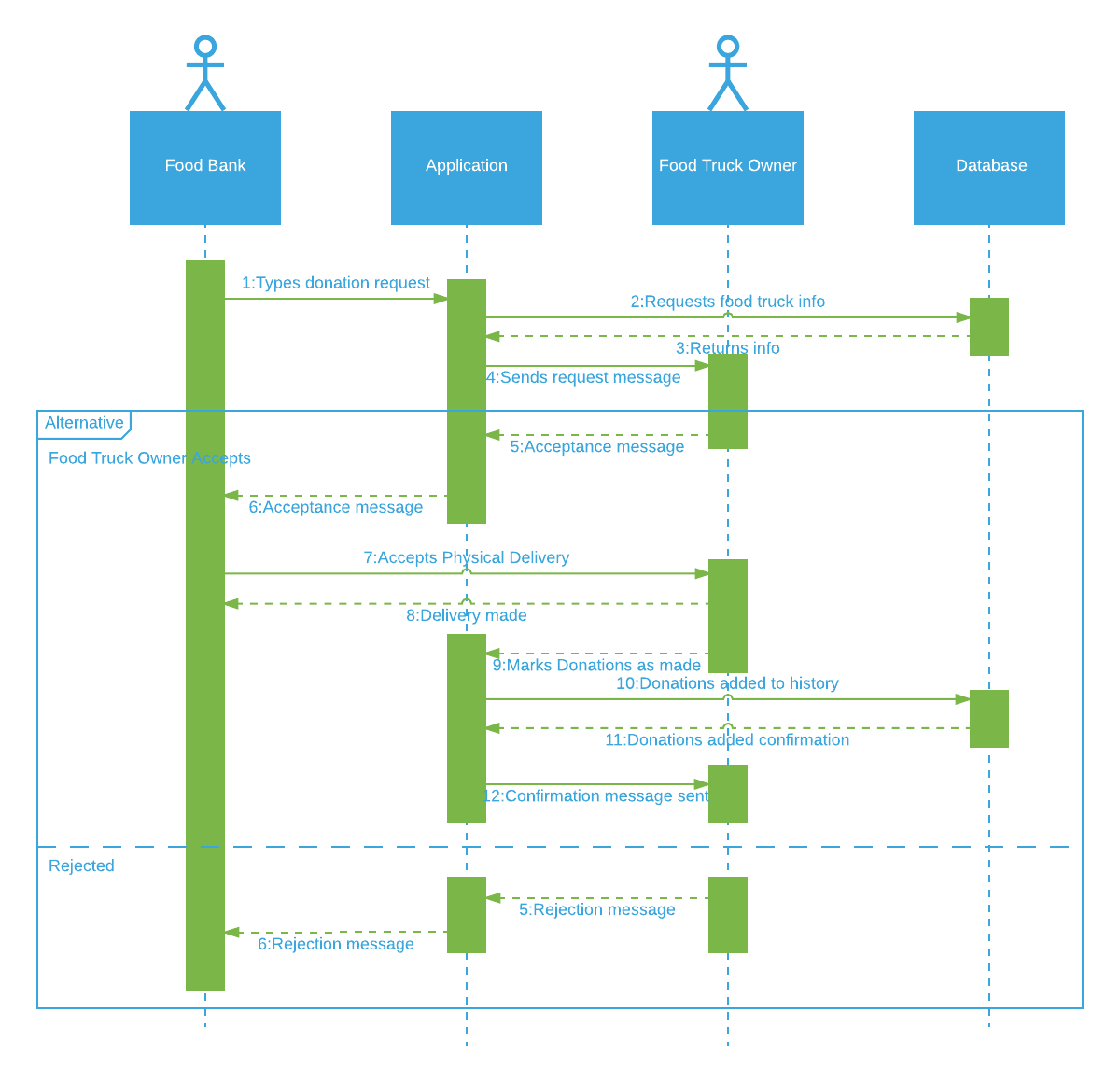
### Use Case 2 Sequence Diagram



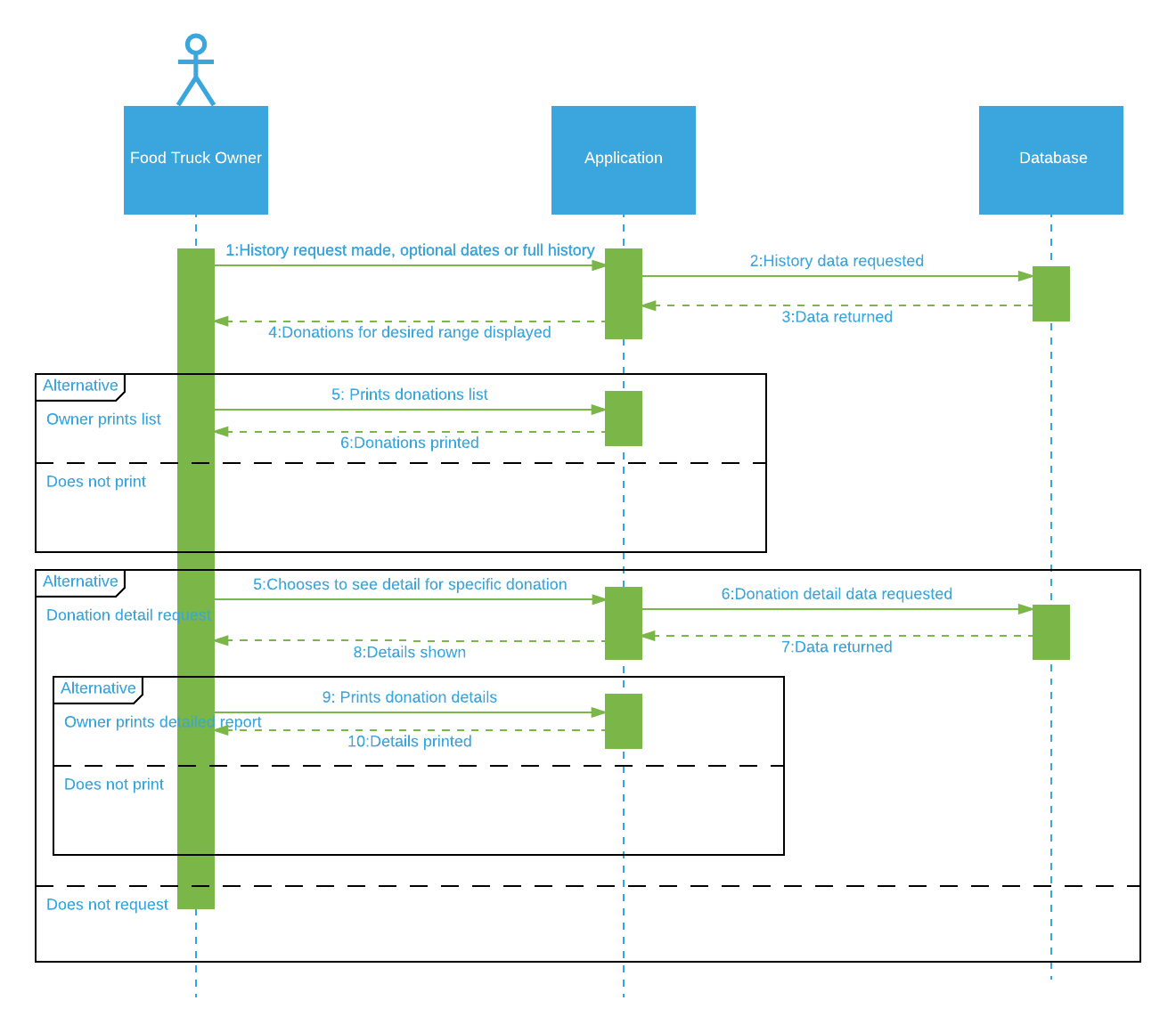
**Use Case 3.1 Sequence Diagram**



**Use Case 3.2 Sequence Diagram**



### Use Case 4 Sequence Diagram



### Summary

Although we attempted to communicate with our customer a few times, she was unable to be reached during the first week. Given the short amount of time given to complete the assignment, the group opted to continue without the customer’s input. This resulted in us completing most of the assignment without the customer’s input.

Fortunately, we did make contact with the customer during the second week and she provided important feedback that resulted in the addition of a fourth main feature: donation tracking for tax purposes. The customer’s feedback also resulted in some small changes to our database related to food type flags.

### Member Contributions

We use Google hangouts as our primary communication tools. After discussion, Steven Nowicki is in charge of the the requirement, technical writing part, as well as customer contact. Antonio Bermudez is in charge of the user case, technical writing part. Then Aaron Berns and Wenwen Dong complete the sequence diagrams for user case. Hunter Schallhorn is in charge of the ER diagram and data flow diagram.