**Food Trucks Fight Hunger - Paper Prototype**

Group 24

Antonio Bermudez  
Aaron Berns  
Wenwen Dong  
Steven Nowicki  
Hunter Schallhorn

### Group member responsibility for this assignment:

Customer agrees on look through the documents on Wednesday, October 18th. Customer gives some comments, such as how to track legitimate business.

Team used Balsamiq to collaborate on prototypes designing. Team also used Google Doc to collaborate on answering questions. Team communicated on daily bases. All members contributed.

Antonio: Refining requirements definitions

Aaron: Prototypes for use cases 3.1, 4

Wenwen: Summary of changes and why, Prototypes for registration

Steven: Refining requirements, written use cases

Hunter: Prototypes for use cases 1, 2 and ERD and Data flow updates

## 

### 

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

Functional requirements

* + Application requires internet connection
  + Application must run on a device that allows graphical user I/O.
  + Application should be platform independent for smartphones and web browsers.
  + Device must have ability to notify user of incoming messages.
  + Device must be able to gather location of user.
  + Application must be able to store information.
  + Application must be able to communicate between UI and data store.

Non-functional requirements

* + Application should not slow down device nor run slowly on user’s device.
  + Application should be minimally intrusive in terms of device and space requirements.

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

Feature 1 Definition: A database of mobile food vendors and food banks/organizations that is accessible via a UI

Functional Requirements

* Vendors must be able to search for food banks and food banks must be able to search for vendors. The vendors and banks must be able to filter the searches based on vendor/bank names, location, hours of operation, and food types associated with vendors and banks.
* Vendor must be able to post food available to donate and food bank must be able to post food needed. Posts by vendor must contain enough information for banks to make a proper decision on whether they need and can accept the food or not. Posts by banks must contain enough information for the matching service of the application to properly match a vendor and bank.

Non-functional Requirements

* Searches must be accurate and show real-time needs of food banks and availability of vendors.
* UI must make sure vendors and banks submit a minimum amount of information with each post.
* Request and submit interfaces for application must be intuitive and easy to navigate for non-technical users. Users of application do not work in technology sector.
* Request and submit interfaces must be able to be quickly navigable. Vendors and food bank employees are generally pressed for time and have high-demand jobs.

Feature 2 Definition: Text based messaging system

Functional Requirements

* Vendors and banks must be able to communicate with each other regarding donations. This communication must be able to occur within the application. They should not be necessarily required to divulge contact information between each other.
* Vendors and banks must have ability to access conversation and message history.

Non-functional Requirements

* Parties must be able to send and receive communications in near real-time.
* Parties must be able to know when they have received a communication.
* Parties must be able to communicate with multiple other parties at once.

Feature 3 Definition: Automated distance based alerts

Functional Requirements

* Vendors must be notified when there is a food bank available to take their food. This will occur when a bank and vendor have a matching donation need and donation type, respectively, as well as when vendor and bank are within a certain proximity.
* Vendors and banks must provide their location.
* Vendors must provide the maximum distance they are willing to travel.
* Application must notify food banks to confirm donations. Subsequently, food banks must be able to accept/deny donation requests.
* Application must be able to produce route for vendor to use to drop off donations at banks.

Non-functional Requirements

* Distance-based alerts should provide useful information to both bank and vendor on the compatibility of the match.
* Alerts should be relevant. That is, they should only occur when a food bank or vendor is in their hours of operation. Additionally, the location of the food bank should be on an efficient route home for the vendor.
* Donation routes must be efficient and must eventually lead vendor back towards “home.”

Feature 4 Definition: Donation Tracking for taxes

Functional Requirements

* Donation report must be generated after each donation.
* Food bank must be able to verify donation report.
* Vendors must be able to access past records of donations for tax purposes.
* Vendors should be able to search by specific date ranges and by food banks.
* Vendors must be able to print entire list of donations made as well as individual reports for each donation.
* Vendors should be able to request that a report is emailed to them after each donation.

Non-functional Requirements

* Donation reports should contain all information necessary for a vendor to file taxes with the US Federal Government IRS, their local state tax office, and local municipality tax office.
* Vendors must be able to provide their email addresses.

Feature 5 Definition: User Registration

Functional Requirements

* Vendors and banks must be able to register with application.
* System must be able to contact necessary parties to verify information being submitted.
* Vendor and bank must have ability to provide all the necessary information to the application to allow for the entire system to function.
* Vendor and bank must be able to confirm account creation so that no other entity can make an account on their behalf.

Non-functional Requirements

* Registration process must collect enough information from both vendors and banks to allow proper donation records to be created, location-based alerts to work, and mapping processes to work correctly.
* Information supplied in registration must be securely managed by the system; both in transmission and in storage.
* Application must verify that registrants are legitimate entities.
* Application must not allow unverified and unconfirmed accounts to use system.

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

Functional requirements

Hardware

* The connecting device needs an 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, Firefox, and Internet Explorer.
* Smartphones must download an app to use the system. Desktop/laptops use a website.
* 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 manually 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 as shown in the project schema. This database 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. Photos are transmitted via Feature 2 and persisted in the database.

### 

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

**Feature 1 Specification: A database of mobile food vendors and food banks/organizations that is accessible via a UI**

Functional Requirements

This will require building a database and a UI that can communicate with the database. The UI should be able to interface with the database in the following ways.

Search

* A food vendor UI can query the database for food banks. Search fields include: the name of the food bank, the location of a food bank, times the food bank accepts food, and types of food accepted. Search fields are used to construct a query, the results of which are displayed in the UI.
* A food bank user UI can query the database for food vendors. Search fields include: the name of the food vendor, the location of a food vendor, and types of food the vendor can donate. Search fields are used to construct a query, the results of which are displayed in the UI.

Post/Update

* A food vendor UI can post data to the database. Post fields include: what type of food a vendor can donate and at what time.
* A food bank user UI can post data to the database. Post fields include: food bank locations, donation times, and types of food accepted.

Non-functional requirements

* Search results should reflect the most recent state of the database, in real time.
* The UI for posting to the database should enforce a minimum field set. This field set is TBD and will differ dependent on UI type (vendor or bank).

**Feature 2 Specification: Text based messaging system**

Functional Requirements

A text based messaging app should be able to transmit messages between users. The UI should be able to query the database to find a user via Feature 1 and then allow a message to be sent to that user. When a message is sent, it is added to the messages database table and associated to the user account that sent it with a ‘sent’ relation. When a message is received, it is added to the messages database table and associated to the user account that received it with a ‘received’ relation.

Non-functional Requirements

* Messages should transmit within a few seconds.
* 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, sending a message can produce a one-sent-to-one-received update or a one-sent-to-many-received update.

**Feature 3 Specification: Automated distance based alerts**

Functional Requirements

The system itself can continually query the database looking for matches between vendors that have food to donate and banks that need food. When a match is found, an alert is produced in the vendor and bank UI.

* To the system, a match is where a food bank and food vendor have a similar record in the dontation\_type table.
* Matches are also distance based. An alert is only produced if the match occurs between a food bank and vendor and the vendor is within their specified max distance. This max distance is set in the UI and persists as a field associated to the vendor.
* To calculate if a match is within max distance, the system must be able to continually update a vendor’s location in truck.location. The Google Maps API will be used for this calculation.
* Static food bank locations are set during user registration and stored in the database.

Non-functional Requirements

* Alerts should only be produced if the food vendor is active or the food bank is open for donation.

**Feature 4 Specification: Donation Tracking for taxes**

Functional Requirements

* When a donation occurs, a vendor UI is used to create a form containing information about what is donated. Once the form is approved in the UI by the recipient bank, the form updates the donation table detailing who donated what to whom.
* A vendor UI can be used to query the donation table. The UI can query by date/time, estimated cost, and food bank name. Query results are displayed in the UI alongside a Generate Report button. Clicking Generate Report sends the query results to a PDF creation interface that prints the data in table form. The PDF is then automatically downloaded by the connecting device.
* A user setting in the UI can enable automated report generation. If enabled, once a donation is ‘approved’, the email account associated to the donating vendor will receive the type of report described above as a PDF attachment. Creation of this attachment 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.

Non-functional Requirements

* This feature requires the vendor account to have an email address associated to it.
* 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.

**Feature 5 Specification: User Registration**

A user should be able to register for an account, via a UI that posts to the database.

* New food vendor records must have: name, email, phone number, business license, and max distance. Adding a new food vendor updates the truck table.
* New food bank records must have: name, location, phone number, business license, and bank hours.
* Following registration in the UI, the system will send an activation email to the email specified in the registration form.
* The new account will not be useable until a hyperlink in the activation email is clicked by a human user. Doing so will update the activated field to true.
* After a business license is received, the system will either use a verification API to verify that the user is associated to a real business or, if this is not feasible, a human employee of the app will download the license and research the business. If the account is found to be fraudulent, it will be deactivated.
* All confidential information within business licenses needs to be transmitted and stored securely. This will require the system to encrypt the business license file.

### Use Case 1: Vendors finding food banks

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 the vendor has available
* Are currently open
* Are within the max distance the vendor has set in their profile

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 donation 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(s) 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 bank(s) willing to take food that have confirmed.
* Food bank is notified that food vendor is driving towards them.
* Vendor delivers food.

### **Use Case 2: Food bank finds donation**

The second use case is the situation where a bank 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 what they may have available 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, the bank 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 food bank 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: Internal Messages System

The third use case would involve a food bank and a vendor who have possibly had a good relationship in the past communicating and scheduling specific dropoffs. In this case, a food bank 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. The 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: Donation History Log

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, an estimated cost range, or specific food banks 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 download a PDF 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, cost, and food bank donated to.
* Application returns requested list
* Vendor can download list in PDF form if desired
* Vendor can select specific donation for more detailed view
* Application returns specific donation with all detailed information
* Vendor can download list in PDF form detailed view as report

### Summary of changes

Splitting the requirements into definitions and specifications in terms of the environment and interface is a major structural change we made. This change greatly improves the consistency and readability of the requirement document. It also revealed new requirements that were needed. In the definition of functional requirements and nonfunctional requirements in terms of system’s interface, the user can easily identify what features will be introduced. More technical audiences can also easily find how these features will work from the system’s perspective in the specification of functional requirements and nonfunctional requirements in terms of system’s interface. Overall, different audiences of the document do not need to go through the whole document to understand the system, and our requirements as a whole have become much more comprehensive and clear.

Adding the user registration is a major functional change made, after discussions with the customer. To use the system, a food bank or food truck will need to provide credentials and become verified as a legitimate business before they can use the system. A food vendor will provide owner name, phone number, and a legitimate business license. A food bank will provide name, phone number, open hours, and accepted food types. A registered user can access to the system’s features. Registered can use all system features. Unregistered or unauthorized food banks will not be able to use the system’s features to access information. Unregistered or unauthorized food vendors can however view the system. In addition to verification, the user register interface will also allow the user to personalize their profile. Importantly, the system will allow a logged-in user to change open hours to be ‘active’.

We also refined the use cases to make them more precise in descriptions.

* For every use case, we added a function name.
* We made sure every precondition or post condition can be reflected in the ERD diagram.

In use case 1, we update the precondition. When a food truck wants to donate, the donation information cannot be shared with just any food bank. That is because food banks are only open during certain hours, and accept certain types of food. Additionally, the food truck cannot travel too far away from his destination. After adding such preconditions, this will ensure our software developer make better use of the data.

Small changes have been applied to the Entity Relationship Diagram. These changes are made according to the system’s interface specification section.

* The food truck adds an ‘online/offline’ flag. The online flag for food truck means that this food vendor is open to donate. The offline flag means that the food vendor is closed. This flag will improve searching results for food bank who is in short of food. It also matches the open/close hours for food bank and can influence on the automated alerts.
* The food truck adds a ‘max distance’. The ‘max distance’ is the maximum distance food truck is able to travel. The ‘max distance’ field can be changed or updated later. The distance field can influence the automated alerts.

Changes in the Data Flow Chart are made to coordinate with the system’s events. These changes are made according to the system’s interface specification document.

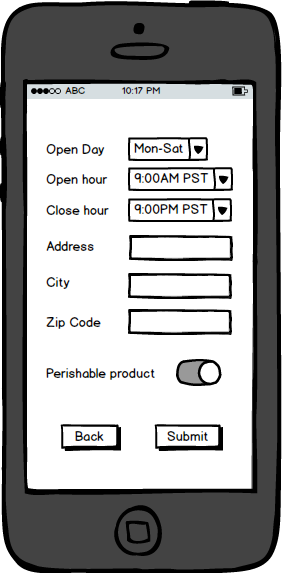
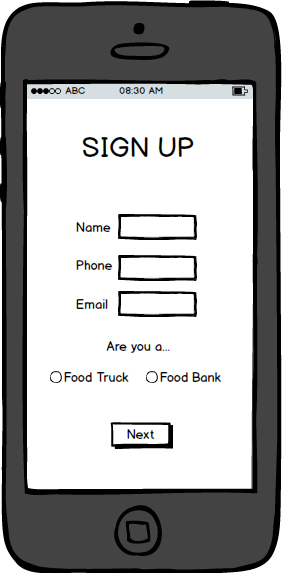
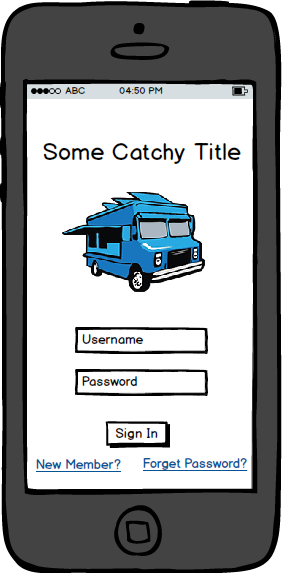
* Overall, the registration for both food truck and food bank users are added in the flow chart.
* For a food truck, the interface will allow a truck owner to search for nearby banks and directly contact food banks, and complete a donation. This is a new add-in event in the flow chart.
* For a food truck, the interface will allow a truck owner to download and view the tax-purposed receipts. This will add two events to the system. One is sending requests to the Database. Another one is the system will prepare the report and allow truck owner to download.

### 

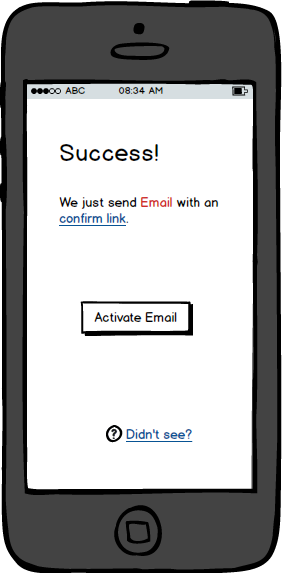
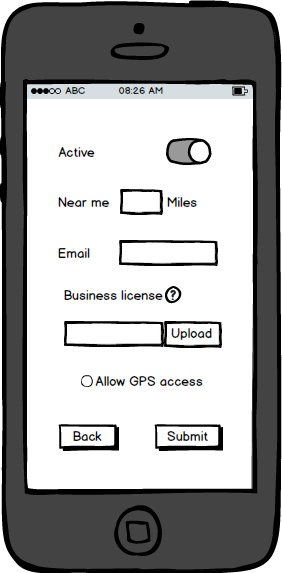
### 

### Paper Prototypes

User registration

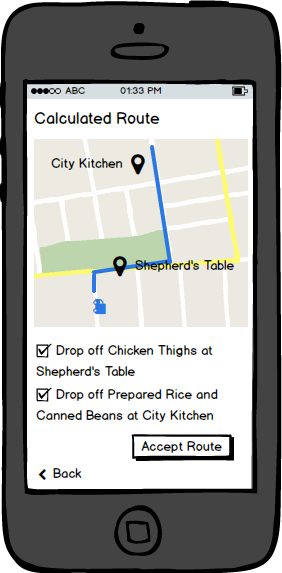
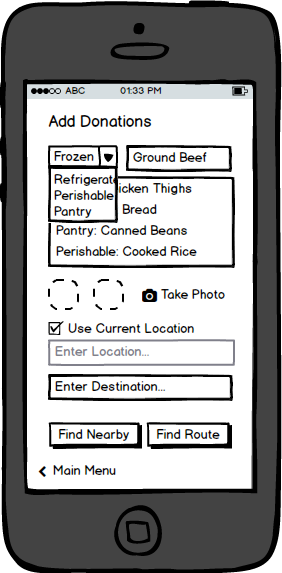
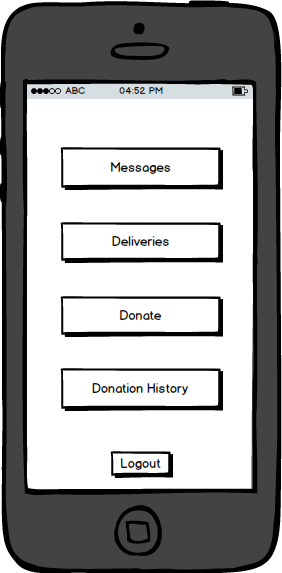


Home Page Sign up Food Bank next page

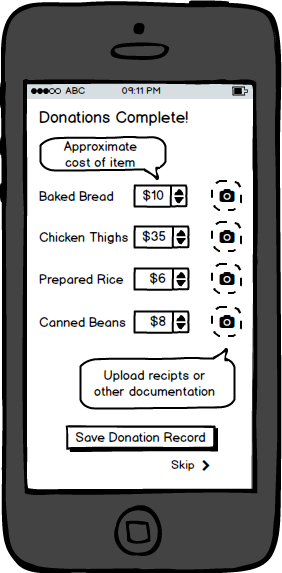


Food Truck next page Sign up complete

UC#1:Food Truck offer donation and complete a delivery

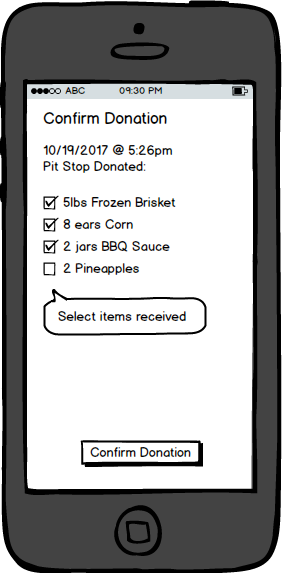
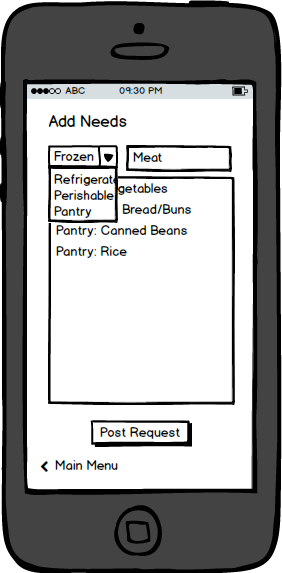
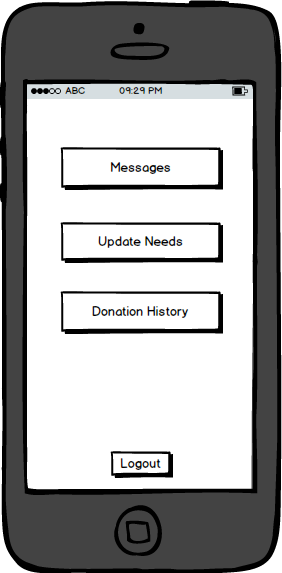


Main menu (food truck) Post Donation Navigation



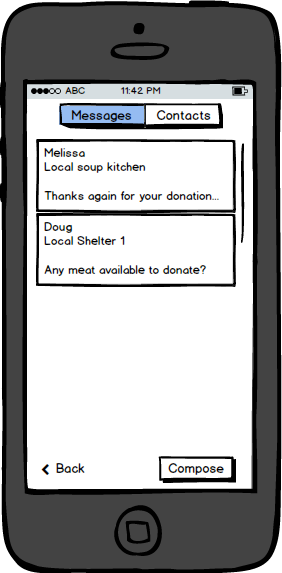
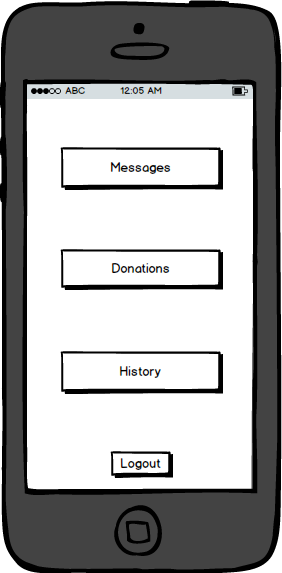
Donation detail

UC#2:Food Bank request donation

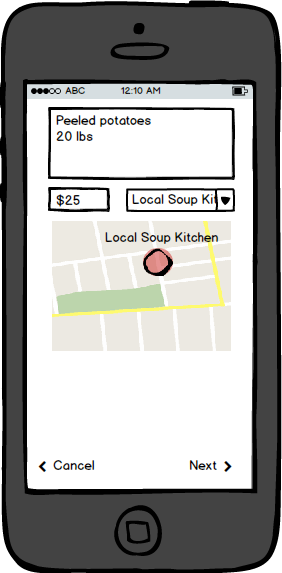
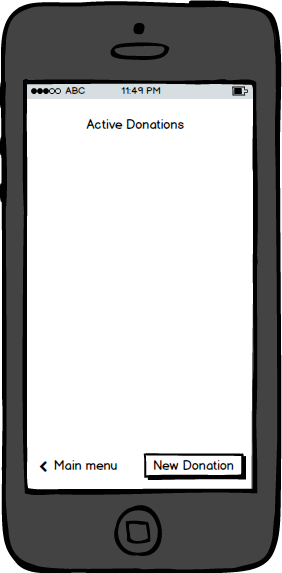
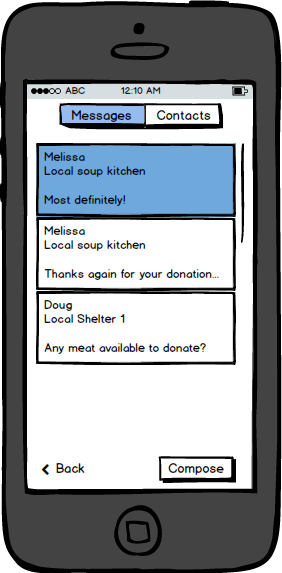


Main menu (food bank) Post Request Confirm Donation

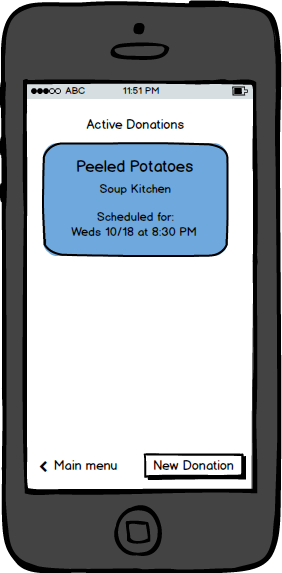
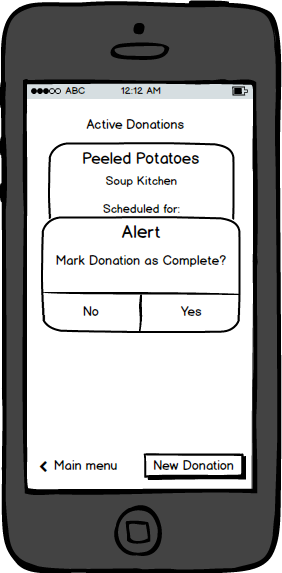
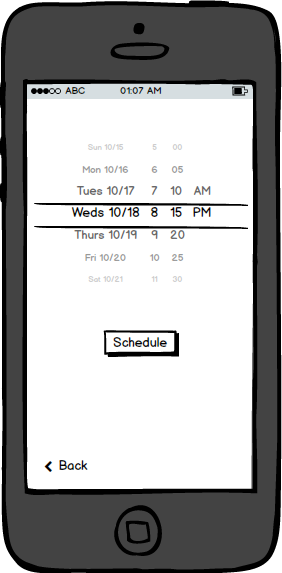
UC#3:Food Truck and Food Bank communicate within app



Main menu (food truck) Main Messages Compose message

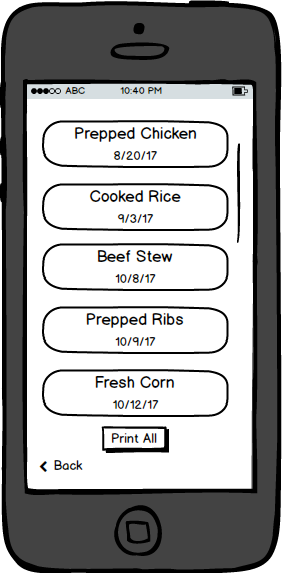
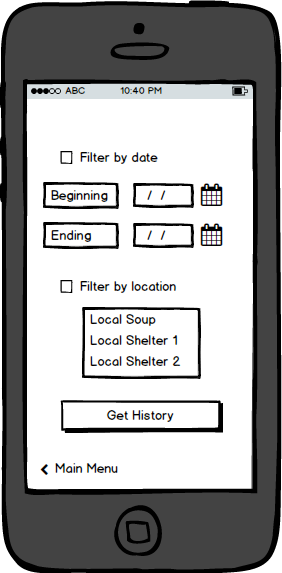
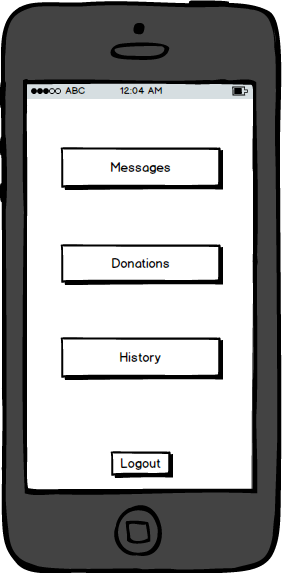


Acceptance message Create new donation New donation

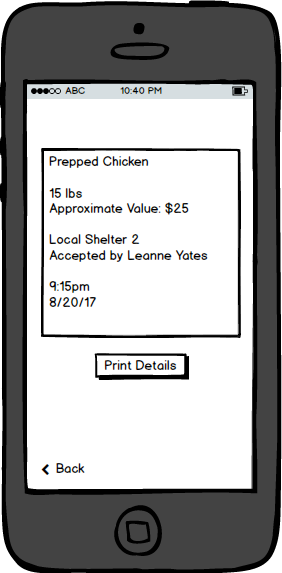


New donation 2 Complete donation Donation List

UC#4:Food Truck get donation receipt

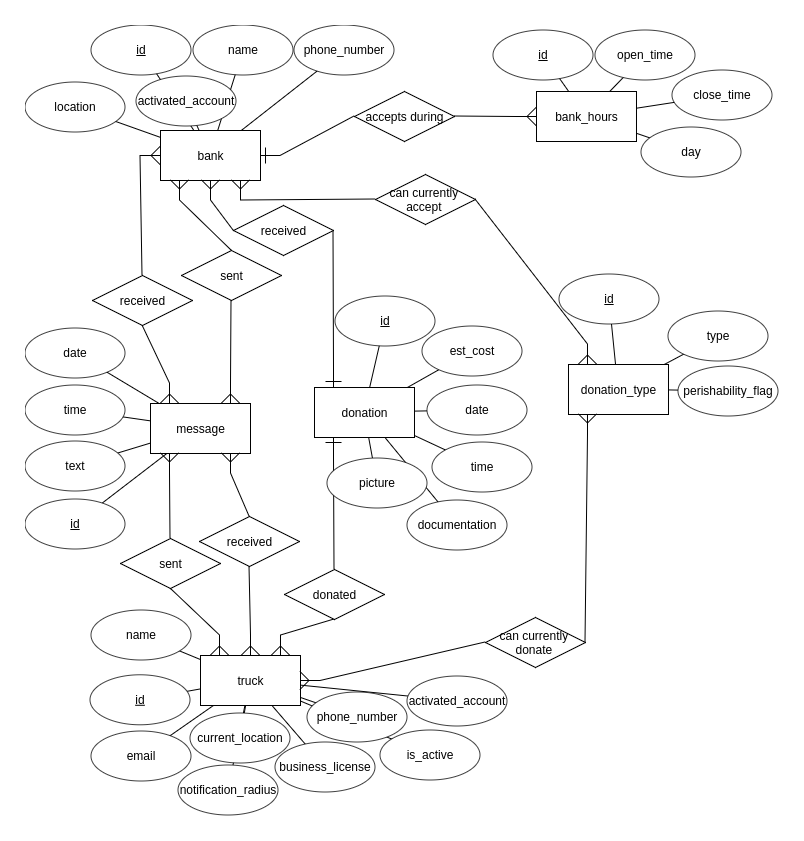


Main menu (food truck) Donation history Donation list

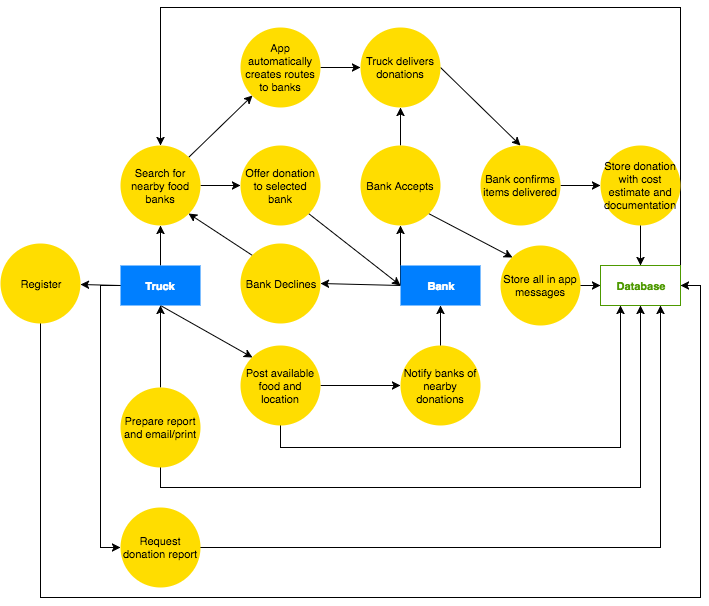


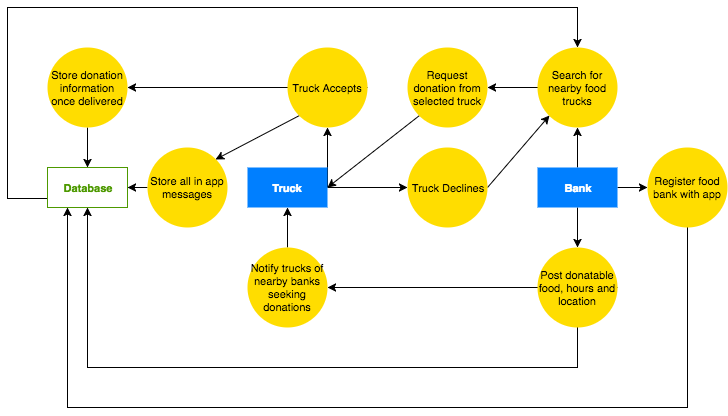
Donation Details

### ERD



### Data Flow



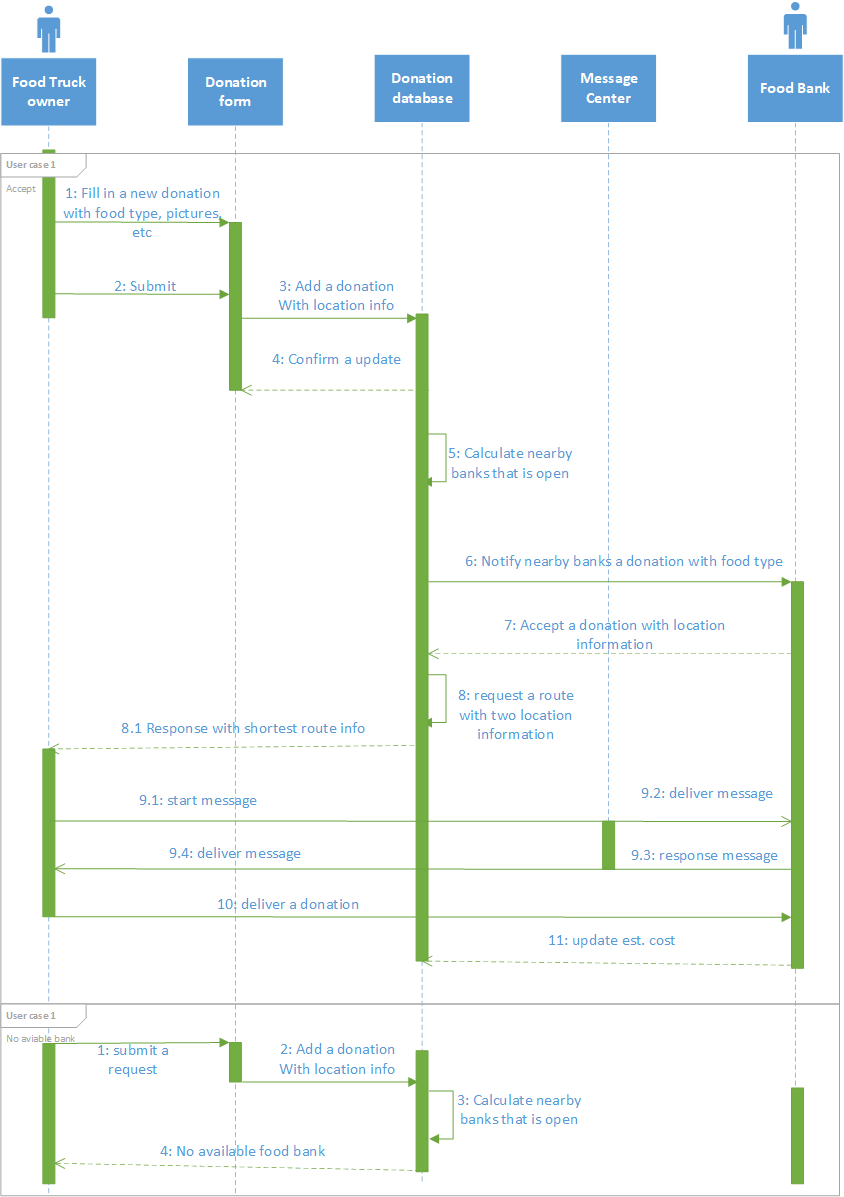


### 

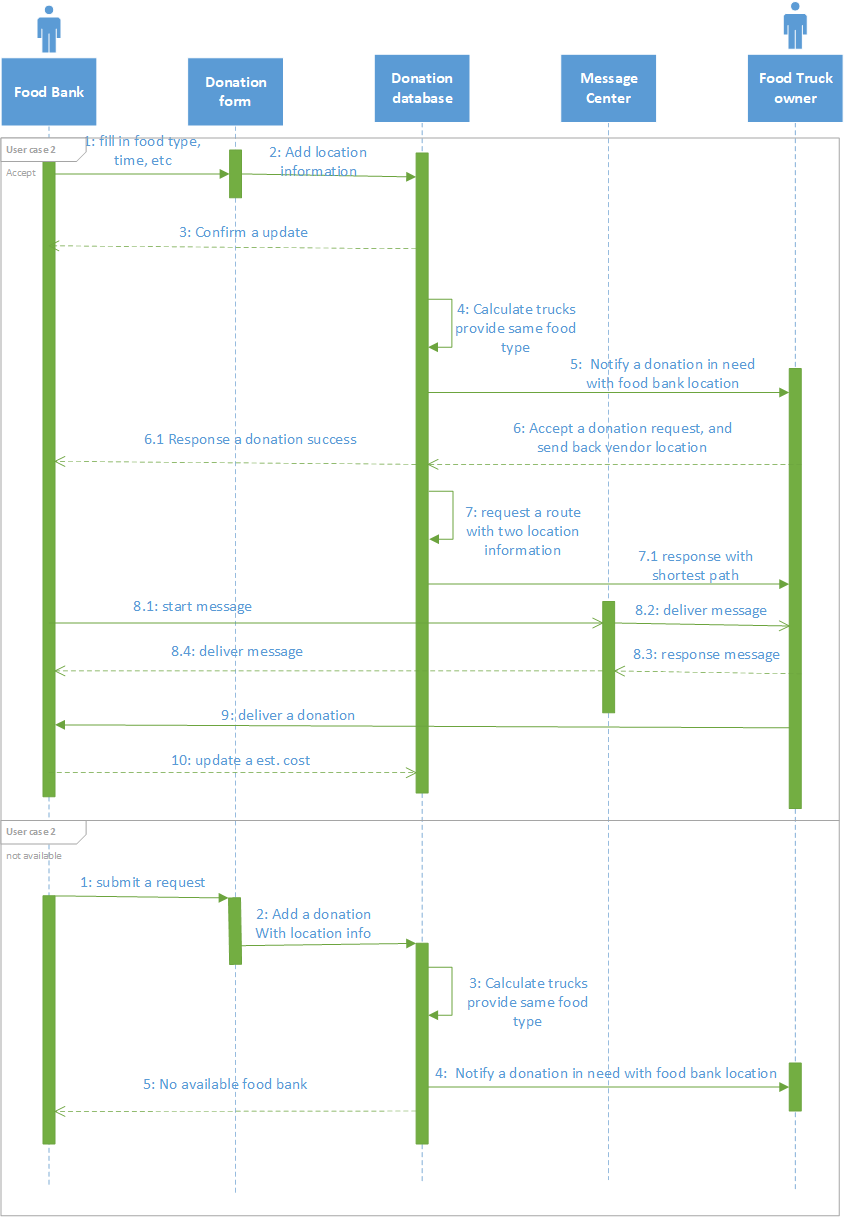
### 

### Sequence chart

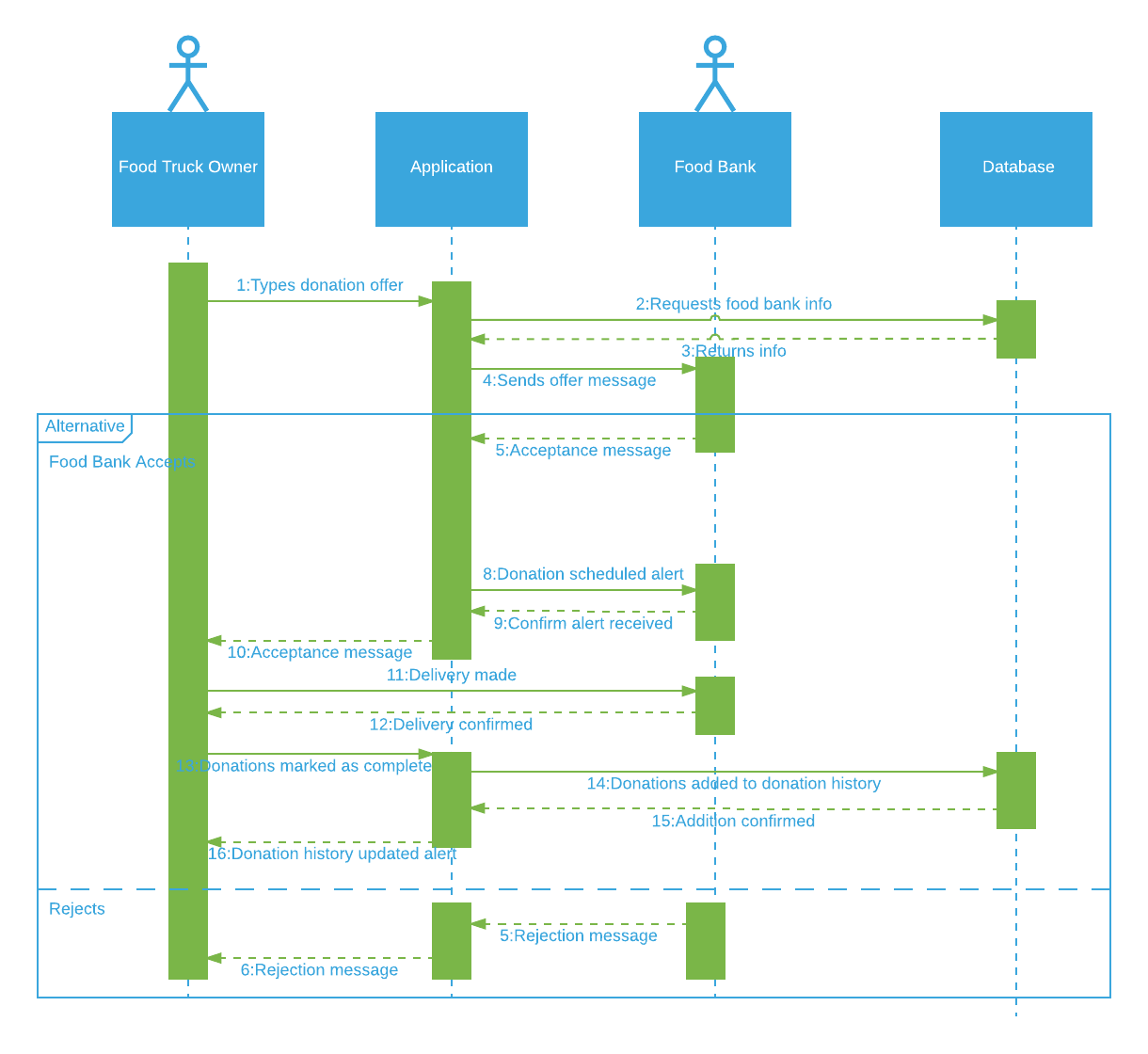
UC#1

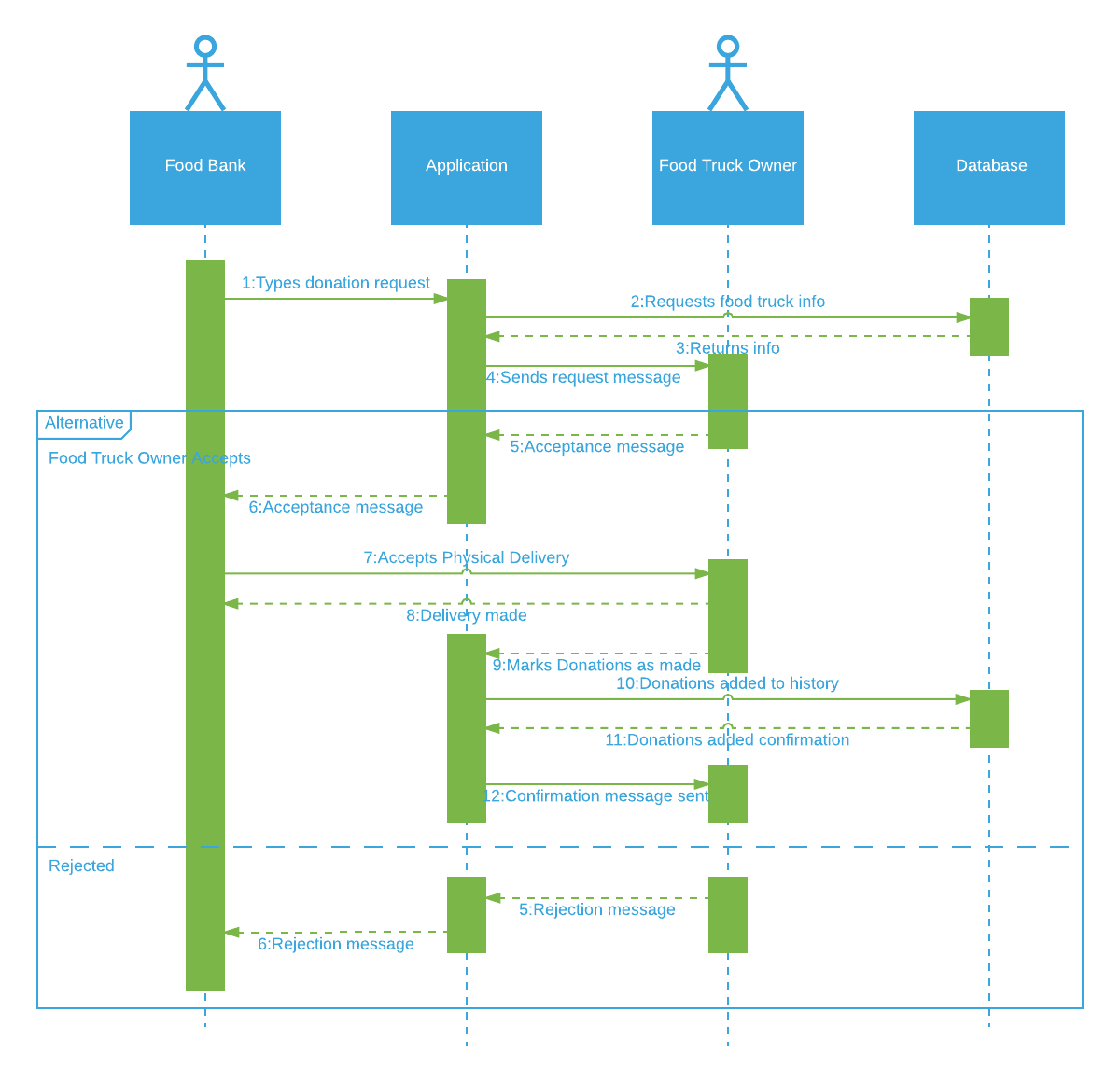


UC#2



UC#3-1



UC#3-2

UC#4

