Food Trucks Fight Hunger - OO Design

Group 24
Antonio Bermudez
Aaron Berns
Wenwen Dong
Steven Nowicki
Hunter Schallhorn

SUMMARY

Customer met group on Tuesday 12 PM CST. We used Google Hangouts to communicate. Customer was reasonable for the requirements. Every member was able to join the meeting.

Team uses Google Hangouts as main communication tool on daily basis. Team uses Google Docs to collaborate on the project. All members contributed.

Collect user stories, and prioritize story	All members with customer
Identify tasks, Estimate efforts	Aaron, Hunter, Antonio, Steve
Sequence diagram and spike	Aaron and Wenwen
Outline your plan for implementing the stories due next week	Steve
Set up server, database, and environment	Aaron, Antonio, Wenwen

User Stories

1 - Story: Register a Food Truck
Description: A food truck user should be able to open the app and register for a new account. After registration, the food truck user should be able to log into the app.
Due/Priority: High (First week)
Tasks

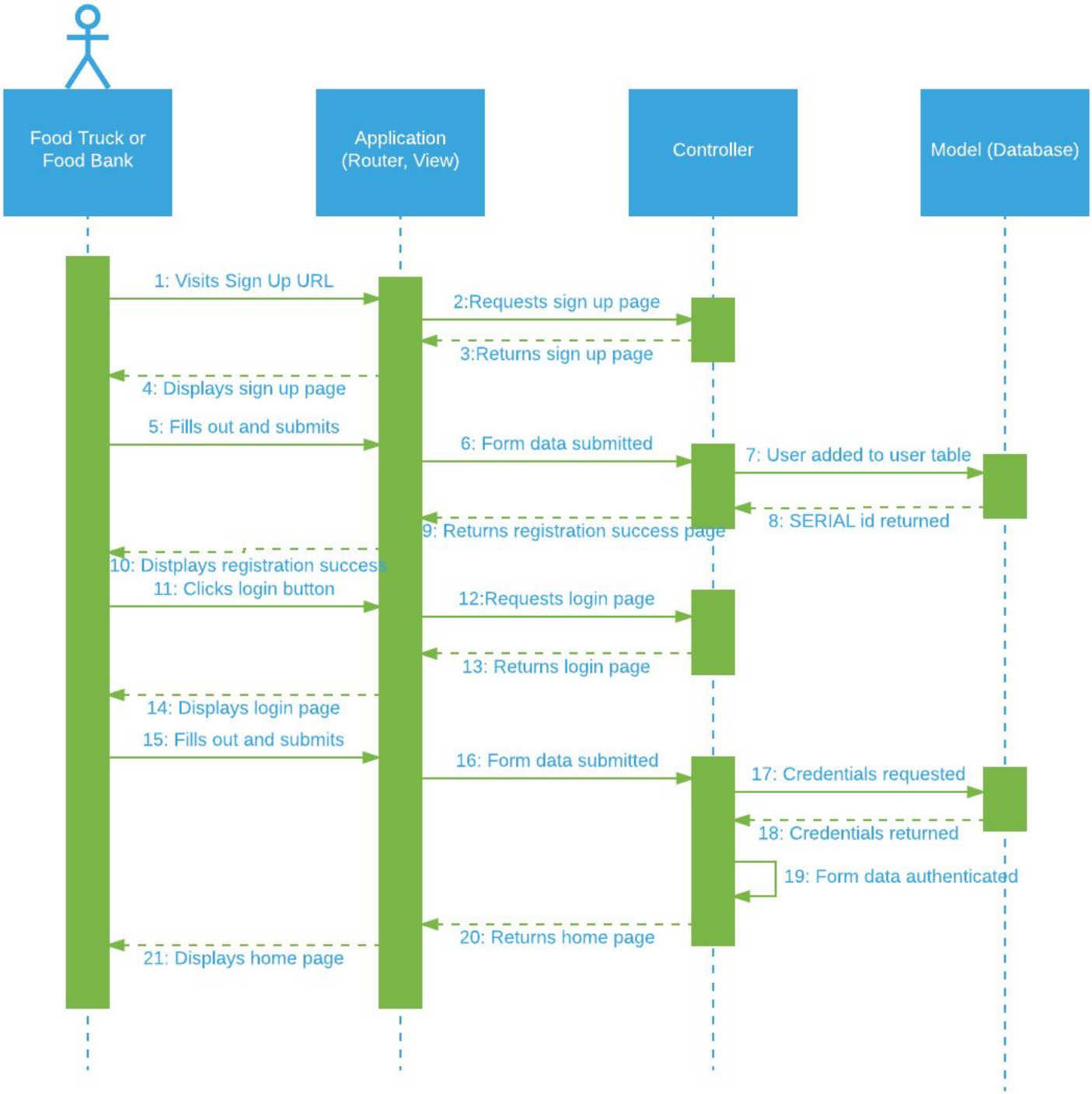
Task	**Time Estimate**
Set up a Postgres database to store food truck member info. https://www.postgresql.org/files/documentation/pdf/10/postgresql-10-US.pdf	8 hrs
Set up an API server to access the database.	8 hrs
Create a webpage to gather and submit new member info.	8 hrs

Order of tasks: 1 and 2 need to happen before the webpage can communicate with the database. Creation of the database also needs to happen before pretty much every other task.

2 - Story: Register a Food Bank
Description: A food bank user should be able to open the app and register for a new account. After registration, the food bank user should be able to log into the app.
Due/Priority: High (First week)
Tasks

Task	**Time Estimate**
Set up a table in the database to store food bank member info.	4 hrs
Set up an API server to access the database.	4 hrs
Create a webpage to gather and submit new member info.	4 hrs

Order of tasks: This story's tasks are almost identical to Story 1; so, if these tasks are performed after Story 1 tasks are completed, they should take less time, presumably because we will have already figured out how to do them.



Sequence Diagram: Stories 1 and 2

3 - Story: Look for Food Banks Nearby
Description: A food vendor should be able to use the app to search for food banks. Search is performed by the user specifying a maximum distance from their current location.
Due/Priority: High(Second week)
Tasks

Task	**Time Estimate**
Create a webpage that takes location information and submits it, displaying results of nearby food banks.	4 hrs
Create route on the API server that queries database for banks with a location within a specified range from the user's current location.	4 hrs

Order of tasks: These tasks can only be completed once the database is up and contains data about users. Within this story, these tasks may need to be completed simultaneously.
Spike:

- Functionally sub-tasks:
    - get user's current location information
    - get user's radius data
    - get user's input search content, it should allow a general search or a specified name search.
    - return a list of search result, and each result can be traced from google maps for further routes requests.
- Technically
    - get Google Maps API keys and create a client object to call google maps places service

create a client:

```
var googleMapsClient = require('@google/maps').createClient({
  key: 'your API key here'
});
```

*refer:https://googlemaps.github.io/google-maps-services-js/docs/

    - need to install google maps service into node js environment
    - save location data by longitude and latitude data
    - execute a radar search with location data, either by 'type' for a general search, or by 'keyword' for a specified search.

This is an example of how to perform a radar search showing shelters in Cupertino that users have described as West Valley. It will return a list of with “West Valley community service” and other related shelters named with west valley.

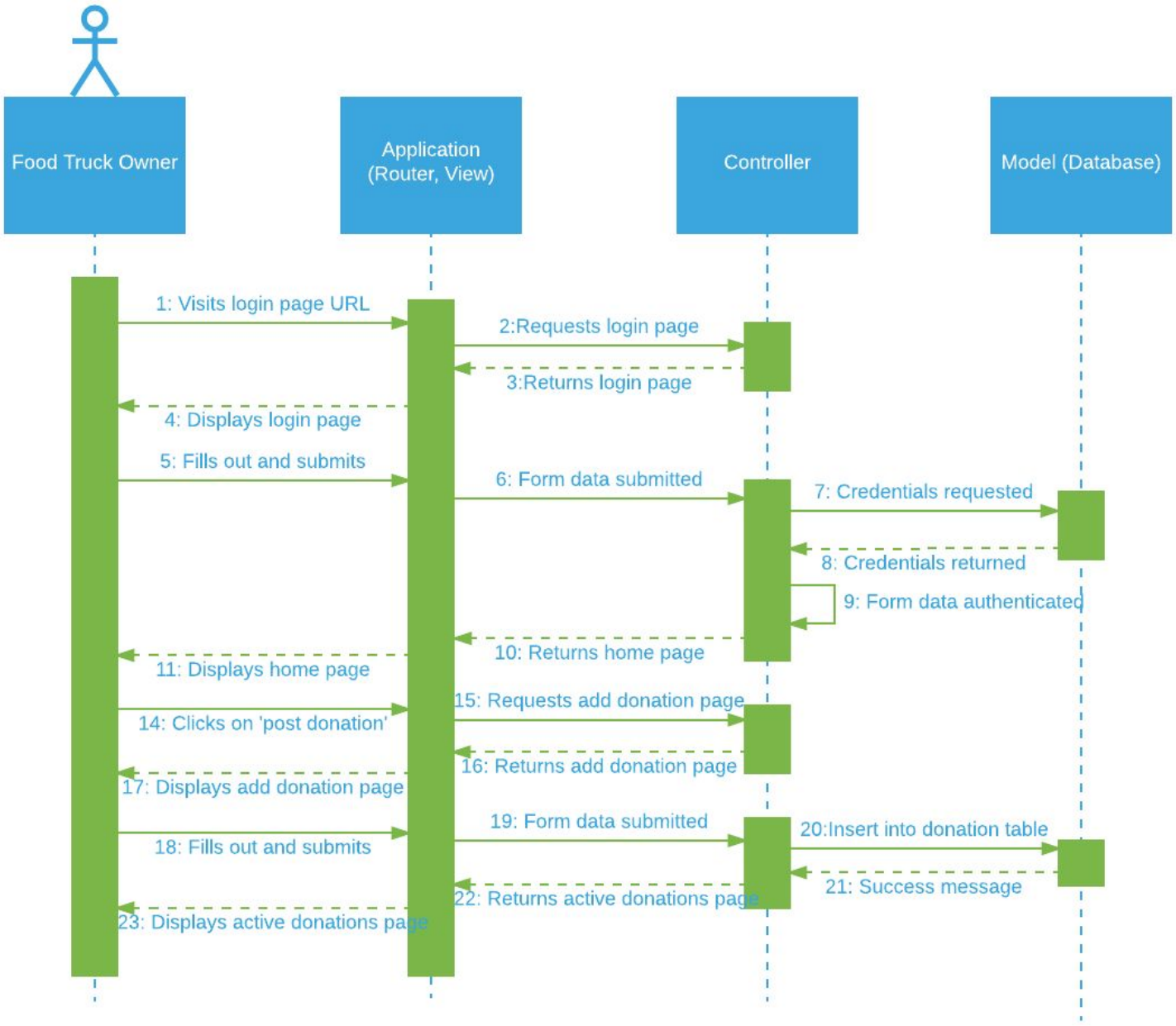
https://maps.googleapis.com/maps/api/place/radarsearch/json?location=37.323191,-122.039645&radius=20&type=shelter&keyword=westvalley&key=YOUR_API_KEY

- get a return page with result list after opening Google maps APP.

4 - Story: Post Donatable Food
Description: A food truck user should be able to post what food they have available to donate. The posting should include details about the food, such as type and perishability.
Due/Priority: High (Second week)
Tasks

Task	**Time Estimate**
Create table in database for available donation entities.	4 hrs
Create routes on the API server to get and post to a truck user’s entities.	4 hrs
Create a web page to list and edit available food and details.	4 hrs

Order of tasks: These tasks can only be completed once the database is up. Also, the first two tasks must be completed before the third.



Sequence Diagram: Story 4

5 - Story: View Account/Settings (Food Truck)
Description: After a food truck user has registered an account, they may want to view their account details. There needs to be functionality for them to do this.
Due/Priority: Low (Second week)
Tasks

Task	Time Estimate
Create get route in the API server to query database for account details.	4 hrs
Create webpage to display these results.	4 hrs

Order of tasks: These tasks can only be completed once food truck users can register. Within this story, these tasks can be completed simultaneously.

6 - Story: Edit Account/Settings (Food Truck)
Description: After a food truck user has registered an account, they may want to edit their account details. Importantly, they may want to change their max distance for a donation setting or password. There needs to be functionality for them to do this.
Due/Priority: Low (Second week)
Tasks

Task	Time Estimate
Create post route in the API server to update account details in the database.	4 hrs
Create webpage form to edit and save account details.	4 hrs

Order of tasks: These tasks can only be completed once food truck users can register and view their account. Within this story, these tasks can be completed simultaneously.

7 - Story: View Account/Settings (Food Bank)
Description: After a food bank user has registered an account, they may want to view their account details. There needs to be functionality for them to do this.
Due/Priority: Low (Second week)
Tasks:

Task	Time Estimate
Create get route in the API server to query database for account details.	4 hrs
Create webpage to display these results.	4 hrs

Order of tasks: These tasks can only be completed once food bank users can register and view their account. Within this story, these tasks can be completed simultaneously.

8 - Story: Edit Account/Settings (Food Bank)
Description: After a food bank user has registered an account, they may want to edit their account details. Importantly, they may want to change their hours of operation or password. There needs to be functionality for them to do this.
Due/Priority: Low (Second week)
Tasks:

Task	**Time Estimate**
Create post route in the API server to update account details in the database.	4 hrs
Create webpage form to edit and save account details.	4 hrs

Order of tasks: These tasks can only be completed once food bank users can register and view their account. Within this story, these tasks can be completed simultaneously.

9 - Story: Generate Alert for Donation Match
Description: An alert would be generated for a food vendor and a food bank when a matching donation type was found within a user-set mile radius.
Due/Priority: Low (Second week)/Won't get to
Tasks:

Task	**Time Estimate**
Create routine that intermittently queries database for available donations and needed donations with matching criteria (food type, within max distance, etc.).	8+ hrs
Create webpage UI that will alert users when a match is found for them.	8+ hrs

Order of tasks: These tasks can only be completed once the database is established and filled with data. Within this story, these tasks can be completed simultaneously, although creating the routine will be the most time consuming task and should be tackled first.

10 - Story: Confirm Food was Donated

Description: When a food truck user makes a donation, they will need to submit a form containing donation details to the recipient food bank user. The recipient food bank user then needs to approve the donation.
Due/Priority: Low (Second week)/Won't get to
Tasks:

Task	**Time Estimate**
Create web page that allows food truck user to submit a donation to food bank user for approval.	4 hrs
Create web page that allows food bank to approve a donation.	4 hrs
Create post route in the API server to update donation approval state in the database.	4 hrs

Order of tasks: These tasks can only be completed once the database contains tables for users and donations.
Spike:

- Functional sub tasks:
    - When a donation is posted, it will have a owner id and trying to find a matching id. The owner id and matching id is abstracted from the user manager's id. from there it can be matched its user type, either a food bank or a vendor.
    - When a donation find a match, the status of the form will change into 'processing'.
    - After the vendor finish a donation, they can change the donation's status from 'processing' to 'to evaluate'.
    - After a food bank received a push message to confirm the donation's evaluation, the status of the donation can be updated, either be 'rejected' or "confirmed".
- Technically
    - It can be done via request/response from app.get().

11 - Story: Get Route to Food Bank(s)
Description: Once a food truck user has decided to make a donation, they should be able to use the app to navigate to the food bank(s).
Due/Priority: Low (Second week)/Won't get to
Tasks:

Task	**Time Estimate**
Integrate navigation API into application backend	8+ hrs

Display navigation in web page	8+ hrs

Order of tasks: These tasks can only be completed once the database contains tables for users that contain location information. Ideally, it would need to be tested by a mobile user on a smartphone, so the mobile UI would also need to be functional.

12 - Story: Send a Message
Due/Priority: Low (Second week)/Won't get to
Tasks:

1. Set up a database to store message info.
2. Create a webpage that handles messaging.
3. Select 'send to' and submit the 'message' to server.

Description: Users of the system should be able to send messages to one another, using the system. Messages are in text form and may optionally include pictures.

13 - Story: Receive a Message
Description: Users of the system should be able to receive messages from other users, using the system. They should be alerted when messages are received.
Due/Priority: Low (Second week)/Won't get to
Tasks:

Task	**Time Estimate**
Set up a database to store message info.	8 hrs
Create a webpage and push to a paired 'send to' user with 'message' info.	8 hrs
Alert sent to recipient user in the web page UI.	8 hrs

Order of tasks: These tasks could presumably be completed as soon as the database has user entities. It is not dependent on other tasks.

14 - Story: View details of a donation
Description: The food truck user should be able to view a donation they have made through the web page. When they view it, basic details should be displayed.
Due/Priority: High (First week)

Tasks:

Task	**Time Estimate**

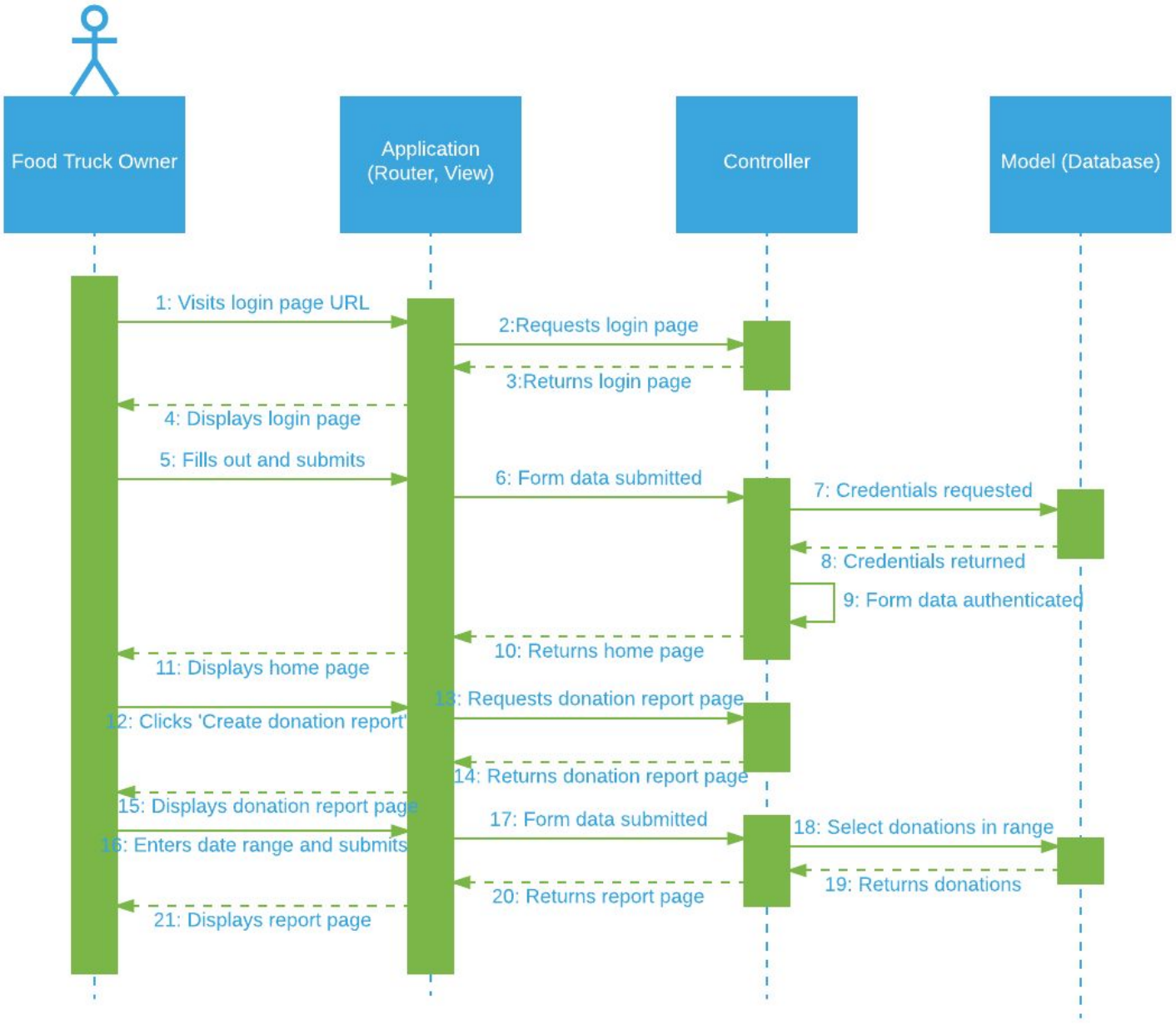
Add functionality to donation history webpage that allows for a donation to be selected and submitted to server. The server should return donation details, which should be displayed in the web page UI.	4 hrs
Create route to query and return specific donation information	4 hrs

Order of tasks: These tasks require the database to have a donation entity set up. Within this story, these tasks can be completed simultaneously.

15 - Story: Generate Donation Report
Description: The food truck user should be able to enter a date range and receive a list of donations they made during that date range.
Due/Priority: High (First week)
Tasks:

Task	**Time Estimate**
Add functionality to donation history web page that accepts range criteria, submits range data within query to database, and displays returned results in the web page UI.	4 hrs
Create route to query and return applicable donation information.	4 hrs

Order of tasks: These tasks require the tasks from Story 14 to be completed. Within this story, these tasks can be completed simultaneously.



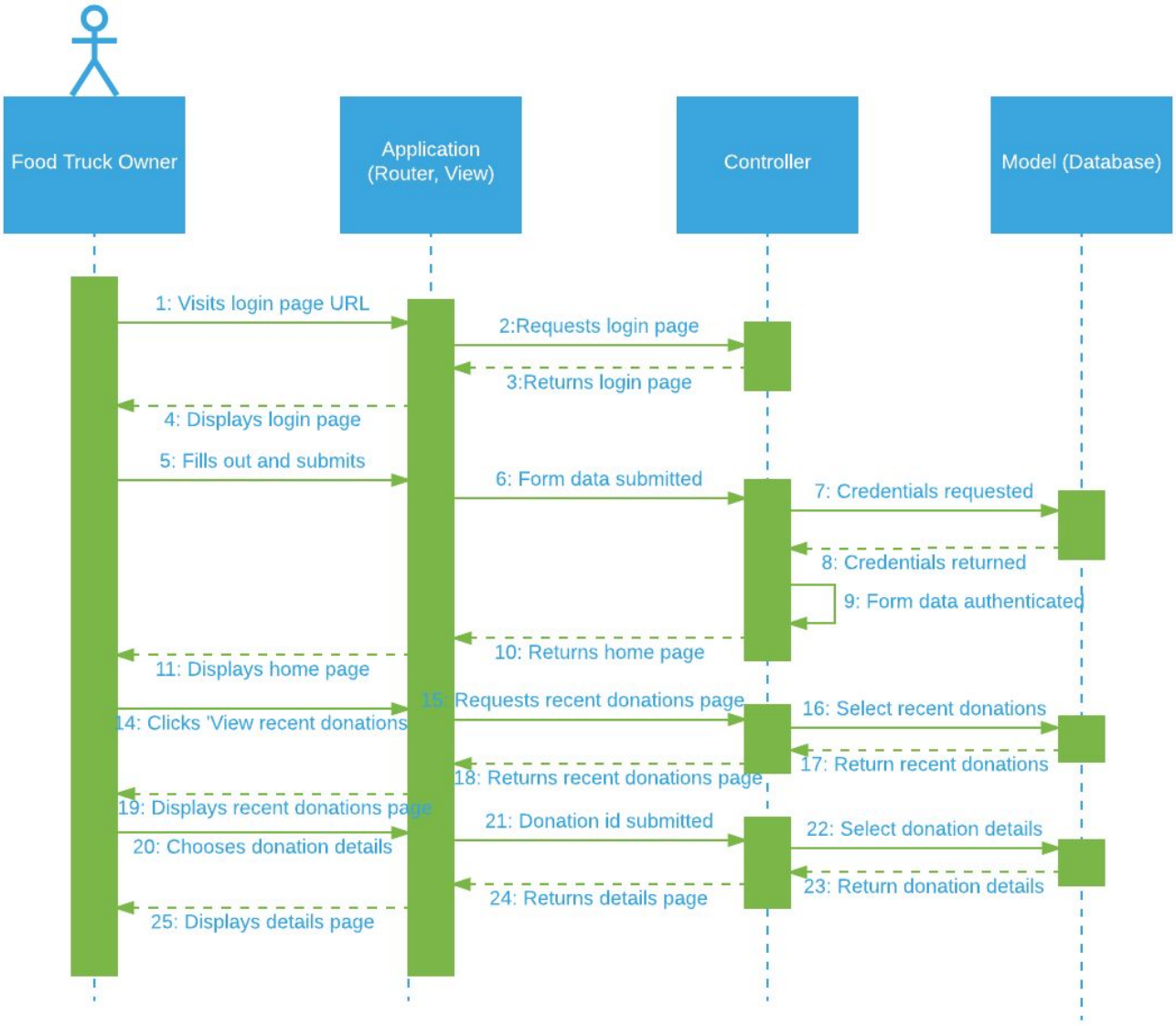
Sequence Diagram: Story 15

16 - Story: Browse Donation History
Description: The food truck user should be able to browse recent donations in a list form. This list form will not display donation details. To see donation details, the user must use Story #14.
Due/Priority: High (First week)

Tasks:

Task	Time Estimate
Create a web page that requests and displays history of most recent donations.	4 hrs
Create routes to query and return donation history.	4 hrs

Order of tasks: This story is somewhat linked and similar to Story 14. It can be completed alongside Story 14.



Sequence Diagram: Stories 14 and 16

Outline of Plan for next week

1. Setting up the Development Environment

The customer has given us plenty of tasks that are categorized as high priority and due next week, however before any of them can be accomplished we need to get a development environment up and running. As we are already finding out, this is a significant amount of work.

Specifics that are needed here include:

- Setting up a Postgres database
- Setting up an application server on Heroku
- Setting up a group repository on Github. Each developer must also set up a local repository and verify that they can pull from and push to the master branch.
- Linking Github to Heroku, so that whenever we push new code, the application redeploys
- Choosing a specific implementation for communication between server and front end. Right now, we are running node.js server side and using Jade to render the front end. Due to limited experience with these technologies, we may need to switch to AngularJS.

2. User Stories

Our goal is to have every User Story marked High Priority (First Week) delivered by the end of the first week. Given the challenges we may face in setting up the development environment, this may or may not happen.

Within our High Priority User Stories, the following are the ones that we will focus on first:

1. Register a food truck or food bank user (Stories #1 and #2)

This task is essentially adding a user to the user table in the database through the front end web page. The breakdown of subtasks needed to make this happen are listed under the user stories themselves. Prerequisite to these subtasks is the full implementation of our development environment. Specifically, we need to nail down exactly how the system's front end will talk to the back end.

2. Look for food banks nearby (Story #3)

This task is essentially allowing the user to use the front end web page to query the database and receive results displayed on the web page. Prerequisites for this story's subtasks are the same as those mentioned above for registration. Note however that the registration stories do not need to be completed before this task. We can simply add food bank users to the database manually to help us build and test this story.

3. Post donatable food (Story #4)

This task is essentially to allow the user to use the front end web page to post donatable food entries to the database. Prerequisites for this story's subtasks are the same as those for the other stories. Additionally, this story should be completed after Story 1, as a potential donation needs to be linked to the specific food truck user who is posting it. As a way around this during development, we could construct the UI form to require that the user enter their user name.

4. Donation Reporting (Stories #14, #15, #16)

This task is essentially to allow the food truck user to use the front end web page to view: details of a specific donation, a listing of recent donations, and a history of all donations they have made. Prerequisites for this story's subtasks are the same as those for the other stories. Additionally, the database will need user and donation information in it. Donation info can be manually added to facilitate completion of this task; however, completion of Story 1 may be perquisite to this task, since reports need to be specific to the user that is logged in. As a way around this during development, we could construct the UI form to require that the user enter their user name.