Food Bank Wish List

Group 20: Jason Goldfine-Middleton, Emiliano Colon Perez, Andrew Elmas, Allister Laurel, Kyle Grotelueschen-Schlatter

Client: Maxwell Goldberg

User Stories

Note: First and Second cycle user stories and estimations are combined. Changes made to user stories and estimations are noted within the list, with changes in estimations made using (->).

Food Bank Representative Wish List Item Entry

- Food bank representatives enter items into a list which contains items the food bank gives preference to receive through donations.

Donor Wish List Item Entry *(Completed: First XP Cycle)*

- Donors can create a personalized list of items they would like to donate.

Match User List items to Food Bank items

- System attempts to match items on food bank lists to items on donor lists.

Compile Recommended list of items for Donor

- System creates a recommended items wish lists for each donor user based on items on local food bank lists.

Generate Cash Donation Suggestion for User

- If there is no match between donors list and food bank lists, the system generates a cash donation option to allow for the purchase of items on the food banks list.

Update User and Food Bank Lists *(Completed: Second XP Cycle)*

- Donations made by donors are tracked and the food bank wish list is updated to reflect the items donated.

App verifies User's confirmed donation

- Donor users use the app to indicate a donation was made at a donation center, once the donation center verifies the donation was made the app indicates the donation was completed.

App accumulates User rewards

- Donor user donations are tracked and awarded points based on the amount donate

Registration and login is needed to access internal pages of the application, including a personal profile and food list *(Completed: Second XP Cycle)*

- A registered and authenticated user, either a donor or food bank representative, can modify either a request or offer list of food items.

Multiple Food Bank Users can add and edit the same list *(Added and Completed: Second XP Cycle)*

- Food Bank Users can access their Food Bank's list so that they can add, edit, and confirm the list.

Donor attempts to access the list of another Donor but is blocked by permissions *(Added and Completed: Second XP Cycle)*

- A Donor might be curious to see another Donor's list, while attempting to they are denied access by permissions and security protocols.

Estimations on Effort for Tasks

In the following list of Tasks, 1 Unit is analogous to 1 weeks' worth of work performed by 2 team members working in tandem under pair programming. With a 5 member team, approximately 4-5 units can be completed in a 2 week timeframe.

Initial Setup *(Completed: First XP Cycle, effort as estimated, Entire team)*

- Create a shared repository for the development team (0.25 units)
- Choose and configure a server for production (0.75 units)
- Set up local development environments/platforms (0.75 units)
- Create a barebones web app to build off of with templates to ease development of new pages (0.25 units)

User Profile Setup *(Completed: First XP Cycle, effort half as estimated, Emiliano)*

- Create page for signup for Donors and Food Bank Users (0.5 units -> **0.25 units**)
- Create login page for users who have already signed up (0.5 units -> **0.25 units**)
- Create database of current registered users (1 unit -> **0.5 units**)

Food Bank User Wish List Item Entry *(Completed: Second XP Cycle, effort minor, Jason & Allister)*

- Create a page with an expandable table with a form that allows Food Bank Users to add items they wish to have donated to their Food Bank (1 unit -> **0.25 units**)
- Create a database and tables to persist the items and quantities requested (1 unit -> **0.25 units**)

Donor Wish List Item Entry *(Completed: First XP Cycle, effort as estimated, Andrew & Jason)*

- Create a page with an expandable table with a form that allows users to add items they wish to donate to a Food Bank (1 unit)
- Create tables to persist items and quantities offered by donors (0.5 units)

Match User List items to Food Bank items

- Create Search button at bottom of user generated list to be compared with local Food Bank wish list (1 unit)
- Compile Recommended list of items for Donor (0.5 units)
- Create algorithm that efficiently matches donations from user and items needed by food banks (1.5 unit)

Generate Cash Donation Suggestion for User

- Create a page for donors that will show the monetary value of the item just scanned that also shows related items that can be donated from current food bank wish lists. (0.5 units)
- Create algorithm where the system will search local groceries and wholesale websites to find the price of the item scanned and show the donation price. (1.0 unit)

App verifies User's confirmed donation

- Create page for Food Bank to confirm when a particular user has completed a donation transaction (1.5 units)

Update User and Food Bank Lists

- Create logic that updates User List and Food Bank list based on the confirmation of reception from the Food Bank provided by a user given above (1 unit)

App accumulates User rewards

- Create rewards page listing all accumulated "points" based on user's donation history (0.5 unit)

Registration and login is needed to access internal pages of the application, including a personal profile and food list (*Completed: Second XP Cycle, Entire team)*

- Create registration page and form allowing user to optionally select a food bank affiliation (0.5 unit)
- Create login page and ensure session stays live (0.25 unit)
- Create user table and plug it into Symfony's "user provider" feature (0.25)

Multiple Food Bank Reps Access Food Bank List (*Added during Second XP Cycle. Completed: Second XP Cycle, Jason*)

- Create route restrictions that allow Food Bank Reps access to their Food Bank's list. (0.25 units)

Donor attempts to access the list of another Donor but is blocked by permissions (*Added during Second XP Cycle. Completed: Second XP Cycle, Jason*)

- Create route restrictions that allow user to access his list items but not those of other users (0.5 units)

User Story Priority List (Assuming 5 time units for the project)

1. **Initial Setup** – All bullet points – Ranked top priority because it's the precondition of everything else in the project. (3 units remain)
2. **User Profile Setup** - All bullet points – Prioritize either Food Bank users or regular users so that there's time to implement the third priority. Return to the type of user not selected if time permits. (1 unit remains)
3. **User Wish List Item Entry** – Page creation – Prioritize page creation over building the database, but try to do both if time allows. I know this exceeds the time units estimate by 1 unit, so I understand if both items under the task list can't be achieved in the two-week period. (-1 units remain, but see the note above on prioritizing one or the other type of user)

Second XP Cycle Summary

During the first XP cycle, we were able to complete the user story: **Donor Wish List Item Entry.** No new user stories were added during the First XP Cycle. After receiving the priority list from our client we did not need to make changes to the rest of our user story list or tasks as we continued with our work. At the start of the Second XP Cycle we added two new user stories: **Multiple Food Bank Users can add and edit the same list** and **Donor attempts to access the list of another Donor but is blocked by permissions.**

As for the tasks, the **Initial Setup** and **Donor Wish List Item Entry** were on par with our initial estimations. The **User Profile Setup** was created earlier than expected, roughly half of the time

initially estimated. Our repurposing of pages created for Donor profiles have also effectively decreased many of our estimations for completing pages used by Food Bank Representatives. We also added new tasks with estimations that not only derived from our initial user story list but also the new user story we added (noted above).

Following consulting with the client as the start of the Second XP Cycle, it was determined that our priorities and task list were still consistent with our client's expectations. Thus, we made no changes to our priority since it was established in the First XP Cycle. Though we only completed one user story in the First XP Cycle, we still completed a variety of tasks that put us in an advantageous position to complete many of the user stories (as noted in the list).

Pair Programming in First XP Cycle

During our second meeting, when we first hashed out the various user stories and tasks, we agreed that we would each try to pair program with as many other members of the group as possible. Additionally, we planned to pair up on all sorts of different tasks in order to learn as much as we could individually.

We generally used Google Hangouts to pair program because it allowed for concurrent video/audio chat and screen sharing. This turned out to work very effectively, although we had a few latency issues at times.

During our first XP cycle, Jason and Emiliano kicked off the pair programming with a focus on understanding the basics of Symfony, configuring the local development environments properly, and establishing a Git workflow for the project. Jason was the driver. It was a productive meeting and helped start the bonding process between members. There was significant discussion of how the different pieces of a Symfony project fit together and some coding done for the controller. No issues were encountered during the meeting.

Emiliano later was the driver when he pair programmed with a couple other members of the group in order to spread some of the new knowledge. Although Kyle had little difficulty getting his local development setup, he had some personal issues to take care of and later pair programmed with Andrew to get up to speed, with Andrew as the driver.

Ultimately, there were very few issues with pair programming during the first week, although a significant amount of the session was spent searching through the Symfony and Doctrine documentation in order to figure out how to achieve the session's goals. In order to allow for more coding time during future sessions, members of our group started trying to read the relevant documentation ahead of time and even programming some sample methods in order to test their understanding before logging into Google Hangouts.

In addition to our pair programming, we were actively involved in a Google Hangouts group chat semi-continuously in order to keep our development coordinated and to synchronize our Git workflow. This greatly enhanced all of the benefits of pair programming, especially in the areas of group bonding and awareness of what each other was working on at any given moment.

Unit Testing

Due to the fact that our implementation of the Food Bank Wish List web application was entirely within the framework of Symfony, there were a few different reasons that we were unable to do extensive unit testing.

For one, we relied heavily on controllers, which handle the different routes and build the responses that will be rendered for the client. The Symfony documentation specifically recommends against unit testing controllers, stating that they are better candidates for functional tests. Once we had implemented authentication sessions, it was even more difficult to come up with justifications to unit test these classes.

Secondly, our application relied heavily on Doctrine to persist data to the database, and unit tests for database interactions is a serious undertaking. A lot of prep work and also additional components are needed to create effective unit tests. When Jason asked his coworker how he usually writes unit tests for Doctrine, he mentioned that they create a SQLite database just for

testing purposes. We decided as a group that this would be far too much work and would take away precious development time that we could use to integrate the various features we were working on.

A third challenge we faced was the fact that Symfony works via “magic” functionality. Most of the functions we coded are called by the engine and either do not return unit testable values or change the state of internal Symfony classes only.

Therefore, we found that our best opportunity for unit testing would come from refactoring some of our custom logic in controllers, the authentication voter, and entity classes out into a DataValidation class with public static methods. We were able to come up with unit tests for six different static methods:

isValidStateAbbr(), normalizePhone(), isValidZip(), stripTabsAndNewLines(), validateUserAndUserListMatch(), validateFoodBankUserAndListMatch()

All unit testing was done with the help of PHPUnit. The first four methods involved string manipulation and validation, so the unit tests were simple. On the other hand, the last two methods involved objects and references to associated objects, and required mocks created with Mockery to test properly. We did not find any bugs through our unit tests but we did refactor our code in a fair number of instances and encouraged better object-oriented design.

Acceptance Testing

Some of our preliminary acceptance testing involved making sure that following registration a user would be able to login to their account. We did not find any bugs but this did help us realize the importance of making sure that users can only access pages that they are allowed to view. For example, in our system there are two types of users, Donors and Food Bank Representatives. Donors are only allowed to view their own lists, whereas Food Bank Representatives have access to the list of their respective Food Bank’s.

Once we finished the necessary functionality of displaying the lists, we added a few items in our database to make sure that they showed up on the interface properly. We then created the interface to alter the items in the database. Thus we tested the functions that added, edited, and deleted the items with the database. We did this by adding items through the interface and

checking to make sure that they populated appropriately in the database. We then edited and deleted those items in the interface and again checked back at the database to make sure the changes took effect.

Another test we conducted was to ensure that representatives from the same Food Bank were able to access and manipulate the same list. We had created two users that both had been given the permission to access the Food Bank list. After we were able to access the same premade list we began to alter the list via the interface in one of the users. When we noticed that the second user was able to see the changes from the first user, we knew that our database schema was in line with our desired use case.

Through acceptance testing at various stages, we discovered a few bugs. One was that the user lists showed id values for each row, instead of a simple counter. This was resolved through the use of variables in Twig, the templating engine.

Another was that, early on, one could access internal pages even without logging in if the user typed correct routes into the browser navigation bar. This was resolved through the use of roles and verification methods added to the DefaultController class.

Similarly, we found that a user could edit a food item not in his own list by navigating directly to the corresponding route in his browser. We resolved this using Symfony's voter feature, which uses custom criteria to determine if an authenticated user has permission to modify or interact with a particular entity in a certain way.

Pair Programming for Second Cycle

Coming into the second cycle, the group started working with more fluidity. Once development had picked up during the first cycle, pairing up on certain tasks started becoming easier. This happened because our tasks started to connect and overlap with each other and thus pair programming become much more advantageous when compared to the first cycle.

We continued to use Google Hangouts for the same reasons mentioned above. Although many tasks had overlapped, the complexity of the topics we were covering had increased. Thus pair

programming has helped some of the group members gain more understanding on the sections of the program they weren't directly involved in.

A good example of this was when Allister and Jason pair programmed. Jason had begun work on the unit testing and was able to provide a detailed look into the code he had written to describe the usefulness when it came to utilizing unit testing in our program. Allister was not as well versed so pair programming provided him with a better understanding of the sections that were most important to test. Through describing his work, Jason was given a better idea of where else we should take our unit testing. Since neither were extremely experienced in implementing unit testing in a program such as ours, this was a necessary learning experience to better get a handle on unit testing in general.

The collaboration definitely started becoming more frequent halfway into the second cycle. Andrew and Allister had tasked themselves to work on capturing the user id so that it can be used to connect with the user's list. Andrew took the reins of the driver as both delved into the documentation to work on a section that neither had much experience in. Allister was able to find specific sections in the documentation that helped provide guidance to utilizing functions to capture the user id for future use.

Kyle, Emiliano and Jason started work on developing the addition of food banks to the registration form. Jason was the driver while Kyle and Emiliano helped with the information lookup and with the functionality of the feature. We worked for a while on this and finally Jason found specific sections that allowed this functionality. A prepopulated list of food banks from each of our home locations was created.

Objectively, the second cycle moved more smoothly than the first in terms of collaboration and task allocation. Since we had become more accustomed to each other's schedules, I believe that pair programming become much easier to schedule. And the complexity of the project increased, pair programming became much more helpful to help the group members see how all the different parts functioned with each other.

Reflection

Comparing the different methods used in Project A and Project B for CS361, we found XP preferable to waterfall in that it was much more iterative. Our team of five members worked separate or together, depending on the task, creating new features and editing existing ones. Along the way, we learned new and better ways to do these tasks, repeating the process as many times necessary. We believe XP is a more learn-as-you-go approach of doing work; this is one of its strengths but it is also one of its weaknesses, as it can take a lot of time to get the features to work or the new implementation of the code could conflict with the existing dependencies, making the process very arduous. Another strength of XP is the constant involvement of the customer in the process of software development and the pair programming experience.

On the other hand, waterfall is sequential, and its testing is only done in the final stages of work. In waterfall, most of the time is spent just planning out what we were going to do without actually implementing it; this takes a lot of preparation time, but it also gives you a more goal-oriented direction on what should be done next. One weakness of waterfall is that you are not coding anything until everything is planned out and if something does go wrong, the process has to be started all over again which can make the task you were set out to do take much longer than expected.

QUESTIONS FOR AND ANSWERS FROM CLIENT

1. **Should we allow the user to manually input items offered/desired or have him/her select from a predefined list with generic items?**

From the perspective of making a matching algorithm, I think it will be significantly more difficult if we allow users to choose their own inputs. I would say to use a predefined list of generic items for simplicity's sake. Perhaps a future iteration of this project (not during the two-week period) would allow users to suggest items to add to the predefined list.

2. **The app will require a web server portion as well. Do you want to prioritize the mobile app or the web app?**

I want to prioritize whatever portion of the app is easiest for the group to implement. If you are all more familiar with web programming, make the web app. Otherwise, if you feel more comfortable with mobile development you can go in that direction.

3. **For the app, is an iOS application okay or were you envisioning a different platform?**

iOS is totally fine if you guys want to go with mobile development, but feel free to choose whatever the group is most familiar with from web or mobile platforms and technologies.

4. **Do we need to validate that food bank staff are representing a true food bank, and how many users should be allowed to represent a single food bank?**

I do think that in the full version of this application we would need to do some kind of validation on food bank users. If you would prefer to defer this feature to a future development cycle, I'm totally fine with that.

5. **What kind of login information/personal details do you expect/require from users?**

It depends on how sophisticated the rewards program side of the app is going to be. If we're shipping gift cards to people and filling out W-9 statements on their behalf, we essentially need

to verify that they are real individuals. For the purposes of this portion of the project, I think it would be okay if we collected only names, general locations (So that we can estimate which local food banks are closest to them), email addresses, and perhaps phone numbers optionally.

6. **Could you expand a bit on the team-building and competition portion of the vision? How do you picture teams being set up (by location, by "interest", by some form of social networking)? Furthermore, how do you envision users being able to discover each other? Some kind of profile?**

This is an interesting problem that I only touched on in a cursory way in the vision statement. The following is just speculation on what the team game might look like and I definitely wouldn't require this portion of the app to be implemented in the two-week cycle (so the user profile answer above still holds true).

I think each team competition would have to be implemented within a particular geographical region, and I think a full-fledged version of the app would involve a social-networking profile system that allows regional users to form teams. Team sizes need to be limited in some way, and there need to be clear start and end dates for a given competition. There would also need to be a mapping between cash/in-kind donations and points received by a team. The aim of the game would simply be to score the most points as a team by the end of time period. There could also be daily goals of a particular kind or scavenger hunts for food items that are especially needed by a food bank.

A lot more thinking needs to go into how this could actually be done, but I hope this gives some idea of what I was thinking of in the vision statement. Please let me know if I can clarify the team game aspect of the project further.

7. **How would you like the donations to be handled? Through the web/mobile app or by connecting users with food banks and letting them arrange donations with them directly through their portals? Is PayPal an option?**

My intention was that donations would be handled through the web and mobile apps, themselves. As for PayPal, it's difficult to say which payment processors would actually be used

for the project because I haven't looked closely at the fees and conditions that these vendors charge. For simplicity's sake, I don't think payment processing needs to be implemented in any way for this two-week portion of the project. If you guys want to implement payment processing, then I'm ok with whatever means you select to implement it (through PayPal or something else), but it's by no means a requirement of mine.