Benjamin Tate, Clarence Pine, Raj Rikhy, Ted Sanjeevi
Customer: Matthew Weakley
Project B, Group 6
06/11/2017
CS 361 -- Section 400

User stories:

- As a user of this app, I want the app to be available as either a mobile application or via a mobile browser on the major mobile operating systems, such as Android, Mac iOS, and Windows Mobile.
- As a user who lost a pet, I want the app to be easy for local shelters to use, so that I know if my pet is being held at a shelter.
- As a user of this app, I want the ability to mark where the pet was last seen before being lost or where it was found, so that searches can be narrowed.
- As a user of this app, I want to be able to easily create a secure account that will store my contact information and data about my pets, so that I don't have to enter anything more than once.
- As a user of this app, I want notifications to happen through the app itself, so that my private contact information isn't shared with other users if I don't want it to be.
- As a person with a lost pet, I want to be able to receive push or SMS notifications for recent posts so that I know right away if my pet is found.
- As a user of this app, I want posting and searching for lost pets to be quick and painless, so that it doesn't take a long time to get results.
- As a user of this app, I want the app to be accessible from the most popular computer operating systems, such as Mac OS X, Microsoft Windows, Chrome OS, and major Linux distributions.
- As a person with a lost pet, I want a simple search feature, so that I can look only at found pets that match my pet's description in my area.
- As a user who has found a lost pet, I want a pet verification process to screen the potential pet owner and make sure the lost pet goes home with the correct owner.
- As a person who found a pet, I want to be able to receive push or SMS notifications for recent posts if a pet matching the pet's description is found.
- As a user who has found a pet or as a shelter employee, I want to be able to take a photo from a mobile device and upload it to the app so lost pets can be easily added to the database.
- As a person with a lost pet, I want a single place to check for information about my lost pet, so that I can be confident that I am finding any information that is out there.

Priority List:

1. As a user of this app, I want to be able to easily create a secure account that will store my contact information and data about my pets, so that I don't have to enter anything more than once.
2. As a person with a lost pet, I want a simple search feature, so that I can look only at found pets that match my pet's description in my area.
3. As a user of this app, I want posting and searching for lost pets to be quick and painless, so that it doesn't take a long time to get results.
4. As a user of this app, I want the app to be accessible from the most popular computer operating systems, such as Mac OS X, Microsoft Windows, Chrome OS, and major Linux distributions.
5. As a user of this app, I want the app to be available as either a mobile application or via a mobile browser on the major mobile operating systems, such as Android, Mac iOS, and Windows Mobile.
6. As a user who has found a pet or as a shelter employee, I want to be able to take a photo from a mobile device and upload it to the app so lost pets can be easily added to the database.
7. As a person with a lost pet, I want to be able to receive push or SMS notifications for recent posts so that I know right away if my pet is found.
8. As a person who found a pet, I want to be able to receive push or SMS notifications for recent posts if a pet matching the pet's description is found.
9. As a user of this app, I want the ability to mark where the pet was last seen before being lost or where it was found, so that searches can be narrowed.
10. As a user who lost a pet, I want the app to be easy for local shelters to use, so that I know if my pet is being held at a shelter.
11. As a user who has found a lost pet, I want a pet verification process to screen the potential pet owner and make sure the lost pet goes home with the correct owner.
12. As a user of this app, I want notifications to happen through the app itself, so that my private contact information isn't shared with other users if I don't want it to be.
13. As a person with a lost pet, I want a single place to check for information about my lost pet, so that I can be confident that I am finding any information that is out there.

Tasks

(Total 26.75u, 10u available per week) (prioritized by customer):

- 2u - Create accounts:
  - 0.5u - Make a database to store username, password, contact info, pet info
  - 0.5u - Create signup page
  - 0.5u - Create login page
  - 0.5u - Create account management page
- 2u - Search function:
  - 0.5u - Create a table to store information about lost pets
  - 0.5u - Create a table to store information about found pets
  - 0.5u - Create a simple search form page that queries the appropriate database
  - 0.5u - Create a results page that displays matching pets and another page to display detailed information about a pet the user clicks on
- 2.5u - Cross-platform optimization:
  - 0.5u - Ensure compatibility with Chrome, Safari, Firefox, Explorer, Edge browsers
  - 2u - Create mobile version optimized for smaller screens
- 1u - Pet photo:
  - 1u - Create an optional search field for uploading photos of lost/found pet
- 2.25u - Notifications:
  - .25u - Add an option to turn on SMS/push notifications for active searches to the account management page
  - 2u - Implement SMS/push notifications
- 3.25u - Marking location of pet:
  - .25u - Create a search field for address found/last seen
  - 1u - Allow address to be filled from current location
  - 2u - Allow address to be filled by dropping a pin on a map
- 10u - Getting information from other sites:
  - 10u - Find all major lost pet sites and set up functions to scrape entries from each and add them to our database
- 3.75u - Owner verification:
  - .25u - Add “Claim” button to pet details page, which notifies the person who posted that pet
  - 2u - Create a messages page that automatically connects owner and finder when claim button is pressed (messages will also be sent as notifications if the setting is turned on)
  - 1u - If the owner initiated the claim, allow the finder to see the owner’s search terms (including photo if applicable) and judge whether or not it is the real owner
  - .5u - Add ability to send a photo in the message, in case the owner didn’t include one in initial search

Organizing into pairs:

Tackling pair programming of the report was an exercise in coordination. The first effort we did together was to get a high level consensus on user stories and what the flow of the system looked like. This was done in order to help us coordinate exactly what we were looking to build before we began implementing. Beyond simply doing a basic mockup and pseudocode, we wrote down a flow diagram of what the actions were and how they mapped/connected to the different “sub-systems” or parts of the application. We used AWW App to whiteboard out the flow, so this was much easier to understand for everyone involved.

Once we were able to translate the user stories into units of work, we set out to divide the effort between one team that would tackle the front-end side of the application, implementing the user flow, what could eventually become (given more time) multi-user authentication but reduced to a single user auth flow. Secondly, After implementing the authentication and initial user data input, the second task would be to run a search query against the relative inputs. Both of these were well defined as user stories, implemented above - **Search function, and insert**.

The second team was to implement the parts of the user stories listed above that required back-end engagement. This team implemented the parts of the user stories above **Search function, and Insert,** that interacted with the database and worked successfully with the front-end team.

Altogether, we used the Cloud9 IDE & development environment for our editor, and hosted the version tracking of the application and progress on Github so that we were all able to coordinate. However, a major impediment to the pair programming was distance. To overcome this, we used a combination of tools, depending on the time and place of who was working on the front-end. The most successful tool that won out was TeamViewer as it had relatively the lowest latency as well as interoperability between viewing and contribution “Share Screen” mode. This was difficult to get up and running as we kept trying different tools to finally end up with one that was successful. This included trying to rotate on Google Hangouts, Slack, or Skype between git checking in/out before settling on a real-time solution.   Another issue we faced was the database. We used the OSU phpMyAdmin server for our cs290/340 projects but it required a VPN and thus wouldn’t connect on cloud9. Ted read documentation on creating one through cloud9[1] and set up a separate mySQL server for cloud9 that we could all share instead of writing code on separate IDE’s at home. We later switched back to the OSU mySQL server after that for publishing.

Finally, once we were able to settle on a mechanism, we had to implement the unit tests. This was more procedurally complex as we initially came up with ad hoc tests to check the

[1] https://community.c9.io/t/setting-up-phpmyadmin/1723

functionality of the application. One part of the struggle (detailed below) was to understand the comprehensiveness of the unit tests implemented. A few members wanted to address more comprehensive tests cases, all the way to protecting against SQL injection, but given the time constraints it became clear that this was intractable. We were able to work together between pairs to validate the functionality as described in the user stories and ensure that we had a successful product by making our test criteria - both implementation and success - more itemized.

Unit testing:

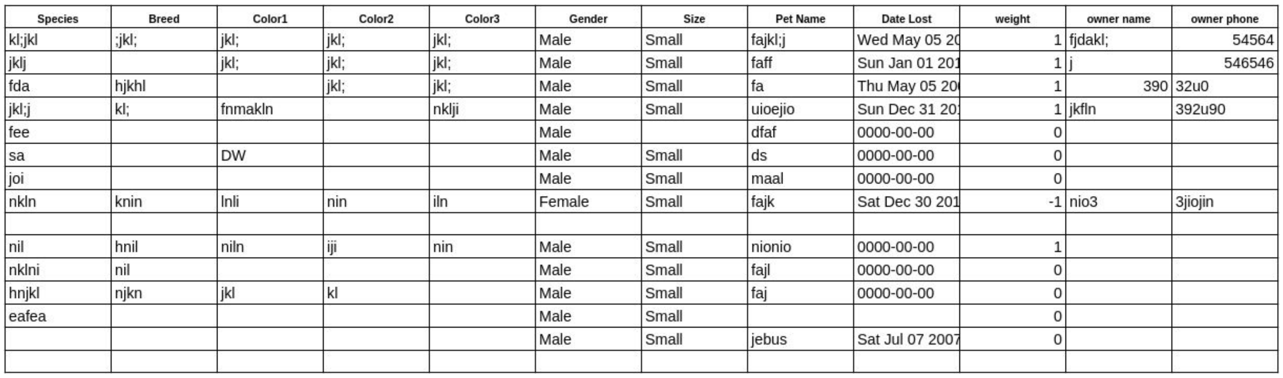
We chose to test the front and back-end of our search feature separately, so as to make identifying the sources of possible bugs easier. Since we were testing the back-end without a front-end to use for input, we used PHPMyAdmin to test our creation, search, and add queries. First, we tested our creation queries at the same time. These queries just created two basically identical tables, one for lost pets and one for found pets. This small part consisted of just one test case, because all we needed to verify was that the tables were getting created and that they contained the required fields. This first test case did not reveal any errors, indicating that the tables were being created properly.

Since there are several optional fields involved in the search, searching required more test cases. However, since the two tables were identical in every respect except name from the back-end point of view, and since we had already validated the tables themselves, we didn’t have to run every test in both directions. The first thing we did to test the back-end search was to run one search query in each direction that included all fields. So first we ran a search of pets that had been found (in other words, we took on the role of a user with a lost pet). Since no data had been added to the found pets table, this search returned no results, as expected.

We then manually checked the lost pets table in PHPMyAdmin for the entry that we had just created by searching the found pets table. Since the entry was present, we moved on to the next step, which was to search for that same entry by sending the same search query to the lost pets table. We then verified that the correct pet entry was returned and that it was now present in both tables. At this point, everything worked as expected, but we realized that repeated searches in either direction would result in duplicate table entries. This could be a problem if one user was repeatedly searching for the same lost pet, but we realized that it also could be valid if it was a result of multiple users looking for different pets that matched the same description. We decided that the only way to accurately determine which case duplicate entries were a valid was with user accounts, which we weren’t implementing yet, so we made a note for this potential bug to be fixed once accounts were implemented.

Now that the searches were working correctly in both directions, we tried searching with once for each field, with that field left blank to ensure that no errors were thrown if some fields were left absent. For these tests, we left out the insert queries to avoid clutter. Finally, we tried inserting an entry into a table with most of the fields blank, to make sure that the filled in columns would still be added correctly. None of these search or insert queries revealed any problems.

For the front end we used the website directly to input different values in hopes of finding new bugs. Sometimes inputted blank values and sometimes inputting nonsensical values to verify system integrity. For example:



Species	Breed	Color1	Color2	Color3	Gender	Size	Pet Name	Date Lost	weight	owner name	owner phone
kl;jkl	;jkl;	jkl;	jkl;	jkl;	Male	Small	fajkl;j	Wed May 05 20	1	fjdakl;	54564
jklj		jkl;	jkl;	jkl;	Male	Small	faff	Sun Jan 01 201	1	j	546546
fda	hjkhl		jkl;	jkl;	Male	Small	fa	Thu May 05 20	1	390	32u0
jkl;j	kl;	fnmakln		nklji	Male	Small	uioejio	Sun Dec 31 20	1	jkfln	392u90
fee					Male		dfaf	0000-00-00	0		
sa		DW			Male	Small	ds	0000-00-00	0		
joi					Male	Small	maal	0000-00-00	0		
nkln	knin	lnli	nin	iln	Female	Small	fajk	Sat Dec 30 201	-1	nio3	3jiojin
nil	hnil	niln	iji	nin	Male	Small	nionio	0000-00-00	1		
nklni	nil				Male	Small	fajl	0000-00-00	0		
hnjkl	njkn	jkl	kl		Male	Small	faj	0000-00-00	0		
eafea					Male	Small			0		
					Male	Small	jebus	Sat Jul 07 2007	0		

As you can see, different inputs were added with different blank spaces to confirm new entries added appropriately. We found a major bug that caused entries to only be blank when developing the database which was solved by specifying "getElementById" as opposed to addPet.elements.color3.value where var addPet = document.getElementById("found_form"). The table above was tested to make sure a confirmation showed, "[your_pet] successfully added to database" and verified that the db had indeed done this. We found that hitting "lost a pet" or "found a pet" multiple times also created multiple forms which we solved using a token mechanism (also mentioned in Acceptance Testing). Now hitting the button multiple times will just open the form or close it.

Acceptance Testing:

Since actual use scenarios for our app require a lost pet, and since the part of our app that we implemented was only for managing and searching database entries, we had to pretend a bit in order to do acceptance testing. We assumed the role of tester, and imagined we were the end users and also sent a link to the portion of the web app to our customer, then played around with the searches, inventing pets that they had lost or found, and tell us what they thought. No major bugs were really found from this round of acceptance testing, which was unsurprising, since we had just run pretty thorough unit tests, and the chunk of the app that had been implemented was fairly straightforward, without much room for hidden bugs. There were little things here and there, such as the "binPlaceholder" text that appearing after each form and staying if the form was closed, or the fact that multiple search forms could be opened at once, but these were all fairly minor appearance things. There was also an issue with 'date_lost' not updating. The binPlaceholder was a leftover bug from when the form kept duplicating every time we pushed a lost-pet or found-pet button. We fixed the issue by using a binary 'token' mechanism to keep track of how many times the button was pressed. We eliminated the binPlaceholder issue by adding it onto our token mechanism (connecting it to the form) and using as a confirmation. The date_lost issue was merely a typo during the copy and paste process in the app.js file. The deleteLost function was nearly a mirror image of deleteFound() so date_found was included in deleteLost by mistake.

We got some comments and suggestions back from our theoretical “end-users”, but for the most part, they were outside the scope of our project. For example, a few suggested entirely new features that they thought would be cool to see in a pet finder app. We acknowledged that these could be brought up with the customer, but that they would have to come after the currently laid out requirements had already been met. The customer was of more help, since he best understood the initial vision of the project and what his goals for it were. He suggested a single page with less buttons since we initially had 4; one of the buttons created two tables to choose from.

Next, after conducting some searches with it, the customer identified the same potential bug that we had in our unit tests -- that repeated searches could result in duplicate entries within either the lost pet or found pet table. We explained that the solution we had come up to for that was to link searches to accounts and validate that one user wasn’t adding duplicate entries. However, the customer came up with a more elegant solution, which was to separate the insert and search functions. This way, inputting a lost or found pet wouldn’t immediately return matching results from the other table, but a user could search for a pet over and over without causing issues. We added this as a new requirement for the second XP cycle.

Finally, one other feature that we came up with during the acceptance testing for the first XP cycle was to give users the ability to edit their entries. This would be necessary if they made a mistake when adding it the first time, or if they gained new information since adding it. An example of the latter case would be if someone found a pet, and gave a best guess for the weight, but then weighed it later and wanted to update the information to make it more accurate. This would also have to be validated through user accounts, but we added it as a new requirement for the next XP cycle.

Changes in second XP cycle:

User stories completed in first cycle:

- As a person with a lost pet, I want a simple search feature, so that I can look only at found pets that match my pet’s description in my area.
- As a user of this app, I want posting and searching for lost pets to be quick and painless, so that it doesn’t take a long time to get results.

For the first cycle, we wanted to focus on the most basic and highest priority feature of the customer's. Since the ability to create user accounts alone wouldn't have any real functionality, we opted to implement the customer's second priority, which was the search feature.

New user stories:

- As a user of this app, I want to be able to edit my lost/found pet entries to fix mistakes or add new information.
- As a user who lost a pet, I want a way to search for pet entries multiple times without creating a new entry.

These user stories were based on new requirements that we found during our acceptance testing in the first XP cycle. We determined from discussing with our customer that the second story, searching for pet entries multiple times, was an important requirement for him, so we made it a priority to implement during the second XP cycle. We compiled the following list of tasks from our testing cycle, and estimated effort for each task:

- Separate search and insert features (0.5u)
- Polish up minor UI bugs (0.5u)
- Add option for a user to edit their own entries (1u)
- Polish design of pages (5u)

Changes to effort estimations:

One big problem we encountered midway through our first XP cycle was that we had drastically underestimated the effort of implementing the front and back ends of the search feature, possibly because we estimated how long we thought it would take if we were working on it full time, rather than how long it would take part time students such as ourselves. As a result, we updated the effort estimations of the remaining tasks from the first XP cycle as follows:

- Create accounts (5u)
- Cross-platform optimization (3.5u)
- Pet photo (2u)
- Notifications (3u)
- Marking location of pet (4u)
- Getting information from other sites (15u)
- Owner verification (4.5u)

In some cases, such as the notifications requirement, we were pretty sure of our initial estimation, but simply rounded up to the nearest day just to be safe, while in other cases, such as the task of creating accounts, we went through each subtask and rethought how long it would take, based on our experience in the first cycle, ending up with a totally different estimation. The total effort for the second XP ended up at 44 units, not including tasks that were already completed.

Changes to priority list:

We talked with the customer about any changes to priorities, and for the most part they remained the same. Creation of user accounts remained as the top priority, and splitting up the search and insert features and fixing the minor UI bugs took the place of the completed search function tasks as the second priority, since the conclusion was that the search function wasn't really complete without that. The other priorities all remained in the same order, despite changes in effort estimation, but the ability to edit entries was inserted above them, at the third spot in the list.

Organizing into pairs (2nd cycle):

The second cycle and experience proved to be a lot easier than the first as some of the basic tools that we used and settled on in the first XP cycle proved extremely valuable. For example, when we ran into issues with the design, instead of implementing different variants for the front-end, we returned to AWW to use the whiteboarding application to iron them out. Additionally, when we settled on Teamviewer for pair programming, it allowed us to be a lot more productive. As we all have work schedules, we found specific time intervals where we would pair program using Teamviewer sessions and Google calendar to coordinate. Using a more regimented approach allowed for faster delivery for a lot of the work required to deliver the user stories. Certainly, in using the same toolset, we were able to achieve a higher throughput and catch up in the lag where the first week suffered a bit. **Our workflow improved significantly.**

When we set out for the second XP cycle, we had established a good working relationship and expertise around some of the tools required to deliver the different components in the first week. Therefore, we decided to continue down the same path of splitting up the front-end and back-end work into two different pairs.

The front-end team, implementing more than just the basic authentication flow and refining the user experience was the goal of that 'iteration;. We had already decided to move ahead with a single user (in lieu of multi-user) to focus on the efficacy of the customer and use case. After discussing with the customer, We realized that zeroing in on the ability to narrow search parameters and isolate multiple pets was an important part of the customer's requirements. Therefore, the front-end pair team worked on improving the delivery of this and working with the back-end team to ensure the search queries and fields were aligned with the database. The front end team also split up the work between implementing searches and inserts as in the first cycle we determined this was the optimal path.

The back-end team, working on improving the query search and inserts, along with tying the features of the new parameters together such that it would work properly with the existing workflow was the focus of the second go-around. This was a lot more difficult to do as a pair-programming exercise as the front-end team had the benefit of visually looking at the data as it was being manipulated in the cloud9 development environment, but the second pair did not. Instead, working with PHPMyAdmin, repeatedly switching back and forth between the GUI and the queries was confusing, and although TeamViewer had relatively low latency, it became cumbersome to jointly work through and switch between interface and query.

The customer proved to be extremely helpful this cycle, as he was able to provide clarity and guidance around the workflow and the prioritization of the requirements in the second go around. Pursuant to this, we were able to understand what to prioritize next for completion, and how best to accomplish the use cases/user stories that mattered. At the end of the cycle, we felt that we had a solid handle on not just passing acceptance criteria but also exceeding expectations.

Reflection:

One major difference that we noticed was how much more focused on user needs the XP model was compared to the waterfall model. With waterfall, we basically just had to think about the discrete units that would eventually come together into the final product, while with XP, we started at the much more detailed scale of what users would want to use the app to accomplish. The result of this difference was that it was much easier to think through all of the requirements and plan out the pieces using XP than it was with waterfall. Another advantage of XP was that it involved actually breaking the work into tasks, estimating the effort for those tasks, and getting them prioritized by the customer, based in part on those effort estimations. In a real-world environment, this would be a very valuable step, as it would ensure not only that the customer and developers were all on the same page regarding timeline, but also that no one would be overworked.

One thing that was helpful about waterfall was making all of the diagrams that clearly showed each part of the project and how the parts would interact with one another. During XP, when we discussing how the final product should look and behave and what attributes each piece would need, there were several occasions that communicating our thoughts to each other was a challenge, and looking back, it may have been easier if we had used diagrams and/or sketches to get our thoughts across. Another thing from the waterfall model that was helpful was drawing out fault trees. While we did consider failure modes this time around, fault trees were a useful way to really visualize how to most efficiently and effectively avoid particular issues.

One more difference between the two, which could be viewed as an advantage for either waterfall or XP, depending on how you look at it, is the process of dividing into pairs for the XP process. This collaboration made it easier to keep the whole team moving in the same direction, but in a big team, it would also limit how many different tasks the team could be working on at once. Even though this difference comes with pros and cons for both processes, our opinion is that pairing up is preferable, because it encourages communication and helps ensure an even workload distribution.

Overall, XP felt like a better process for a project of this large of a scope. While waterfall was quicker and easier, it wasn't as good for planning large, complex projects, as it would have been relatively easy for minor requirements to slip through the cracks, and the fact that wasn't iterative like XP means that those little issues would be harder to eventually catch and fix. XP was less clearly-defined, which sometimes made it less obvious how we should proceed, its user-focused nature and built in iteration made it better suited for a large and complex project.

Code Submission:

- The attached file PetProject.zip contains all of the code for the PetFinder Web app as well as the required node modules.
- Open the file, and README.md provides the step-by-step instructions for using the website and setting up the file locally or on the OSU server.
- The website is hosted at http://flip2.engr.oregonstate.edu:8081/users. To use this site, you need to access from the OSU VPN if not on-campus.
- Since the app is hosted online, the only dependency is a browser. The browser used for testing was Google Chrome, so if the app isn't working on your browser, please download and install Chrome from https://www.google.com/chrome/browser/desktop/index.html and try visiting the above link from that browser.
- The Cloud9 site is at https://ide.c9.io/summonkraken/cs361_projectb_group6_petfinder.