Aggies Care: Non-profit organization projects collection system
CSCE 606 -- Software Engineering -- Project Final Report
Texas A&M University

Project Summary:

The customer's problem involves pairing non-profit organizations with students and faculty in order to complete service projects. Currently, there are many students and faculty who would like to do service projects in exchange for experience, as well as many nonprofits and other agencies who have projects to get accomplished with limited budget. However, it is often difficult to raise awareness of these projects, so many go unnoticed. In the course of this project, we designed a website to connect agencies with A&M students and faculty through volunteer projects. Agencies can post projects which A&M students and faculty can then browse through to volunteer for projects relevant to their expertise.

A major emergent requirement became more evident as the project evolved: the need to monitor the website. The customer made it clear that any time students interacted with the public required close supervision, which was provided for by adding numerous admin capabilities. Tamu Users could be blocked, and agencies, projects, faculty status, and completion of projects all needed to be approved or unapproved. Basically, almost every public-facing action from users needs to be approved by the admin.

User Stories (Points, Status):

Update Tamu User Contact Info (1, Accepted): Tamu Users should be able to update how they are contacted.
View Tamu User (1, Accepted): Tamu Users should be visible to other Tamu Users and Agencies whose projects they are working on. This includes an index.
View Agency (1, Accepted): Agencies should be visible to Tamu Users. This includes an index.
View Project (1, Accepted): Projects should be visible to all Tamu Users and the Agency that owns the project. This includes an index.
Edit Project (1, Accepted): Agencies should be able to edit a project including its name, description, tags, and status.
Approve/Unapprove Projects (2, Accepted): Admins should approve projects before Tamu Users can see them, as well as be able to un-approve them. This includes an index of un-approved projects for the admins.
Approve/Unapprove Agencies (2, Accepted): Admins should approve agencies before they can create a project, or before Tamu Users can see them, as well as be able to un-approve them. This includes an index of un-approved agencies for the admins.
Log in as agency through gmail (2, Accepted): Agencies login through gmail so their accounts will be secure.

Create profile as an agency (2, Accepted): When agencies first login their account is created from their gmail information.
Edit agency profile (1, Accepted): Agencies can change their name, email, and phone number.
Log in as Tamu User through CAS (3, Accepted): Tamu Users login through CAS.
Tamu User profile creation (2, Accepted): Tamu Users must enter in their name and role as CAS does not provide this information.
Create a project (2, Accepted): Agencies can create projects that Tamu Users can join. Projects include name, description, tags, and default status of open.
Add a make admin button (1, Accepted): Admins can make other Tamu Users admins.
Delete a project (1, Accepted): Agencies can delete projects.
Subscribe to Project (1, Accepted): Tamu Users can join projects. They show up on the project page as contributors, and the agency that owns the project can see their contact info.
Drop project as user (1, Accepted): Tamu Users can drop projects they no longer want to work on.
Admin approve faculty (2, Accepted): Tamu Users decide their role, so if the role is chosen as faculty, admins must approve this. Before they are approved their role shows as Faculty (Unapproved). This includes a index of unapproved faculty.
Only master admin can delete admins (1, Accepted): The removal of admin status can only be done by master admins.
View completed projects (1, Accepted): The public can see completed projects so they get a sense of the website.
Admin block tamu user (1, Accepted): Admins can block tamu users to protect the quality of the website.
Edit role of tamu user (1, Accepted): Tamu Users can edit their role. For example, if a student becomes faculty.
Show project posted date in show and index (1, Accepted): To know when the project was created, we included project posted date with the project.
View project list with filters (3, Accepted): On project indexes, clicked the column titles (name, date posted, and agency) with sort by those attributes. There is also a search bar that can search by any attribute of a project except date posted.
Admin approve complete projects (1, Accepted): The completion of projects must be approved by admins. This includes a index of completed projects pending approval.

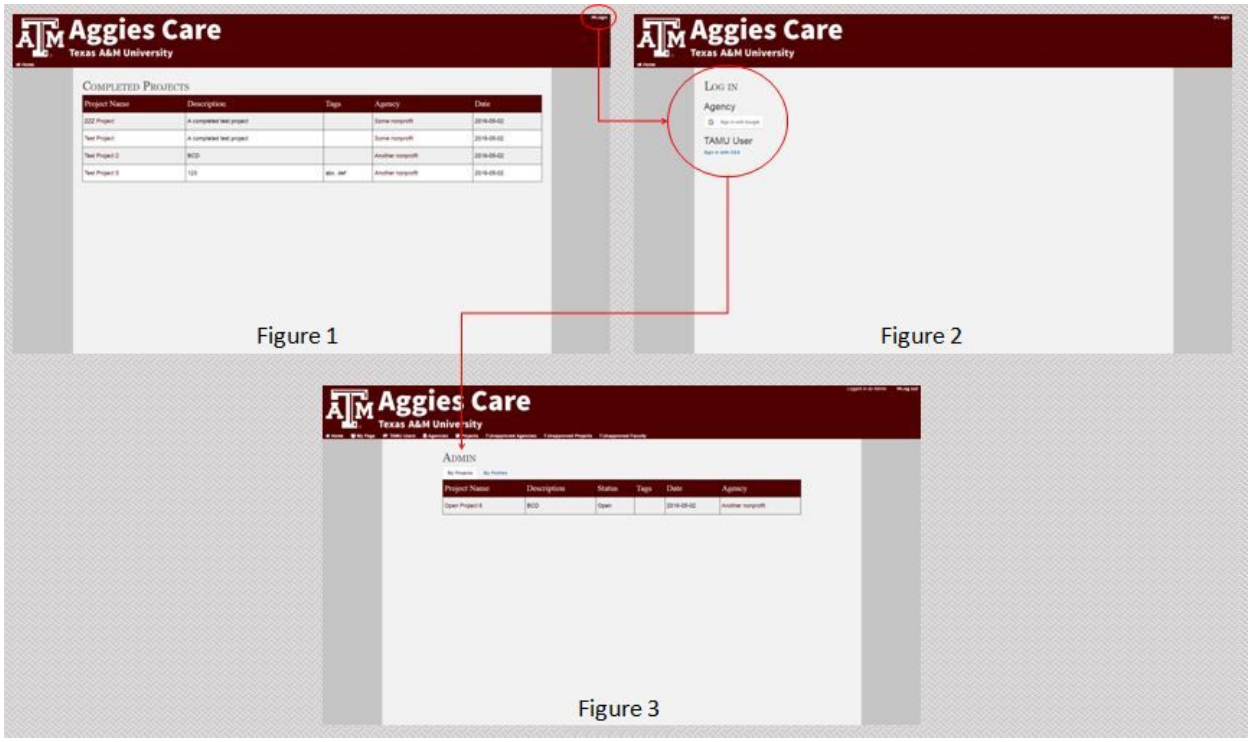

Figure 1 -- Login as an admin user

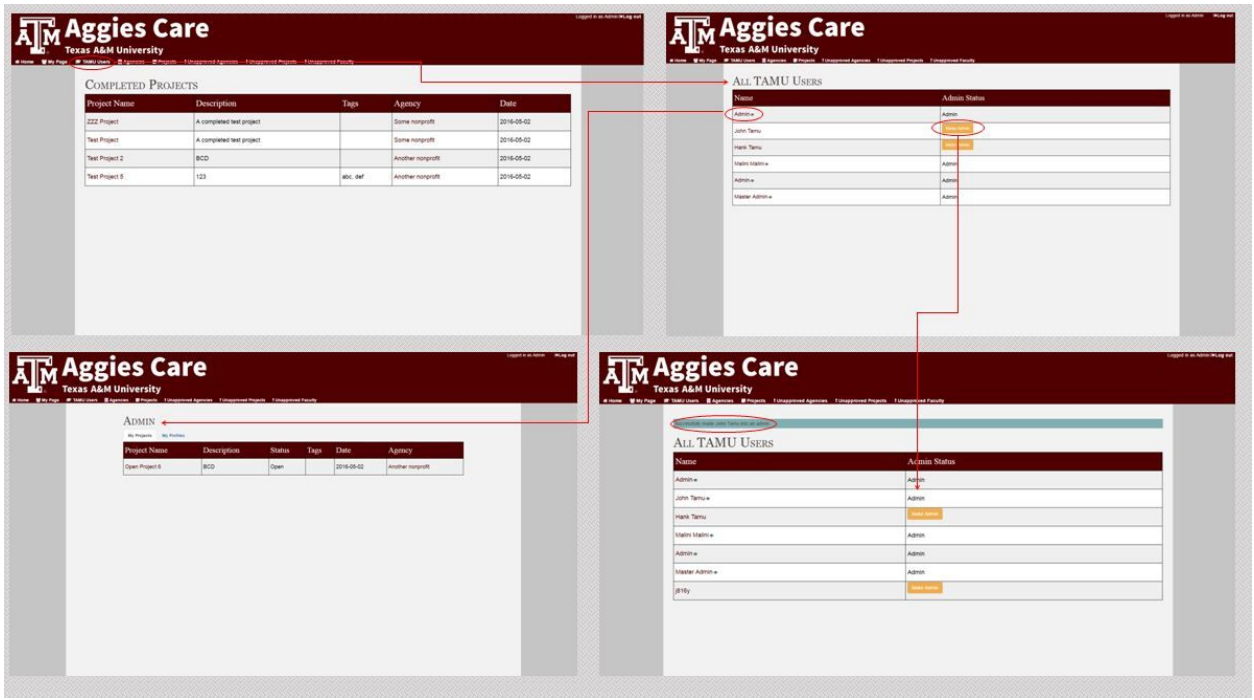

Figure 2 -- View all TAMU users and make user an admin

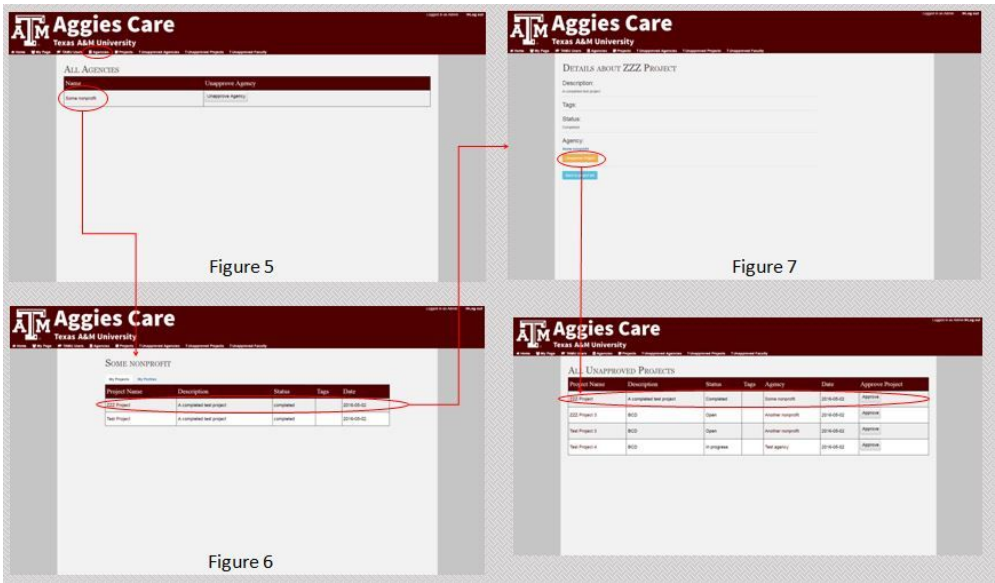

Figure 3 -- View all projects from an agency and disapprove a project

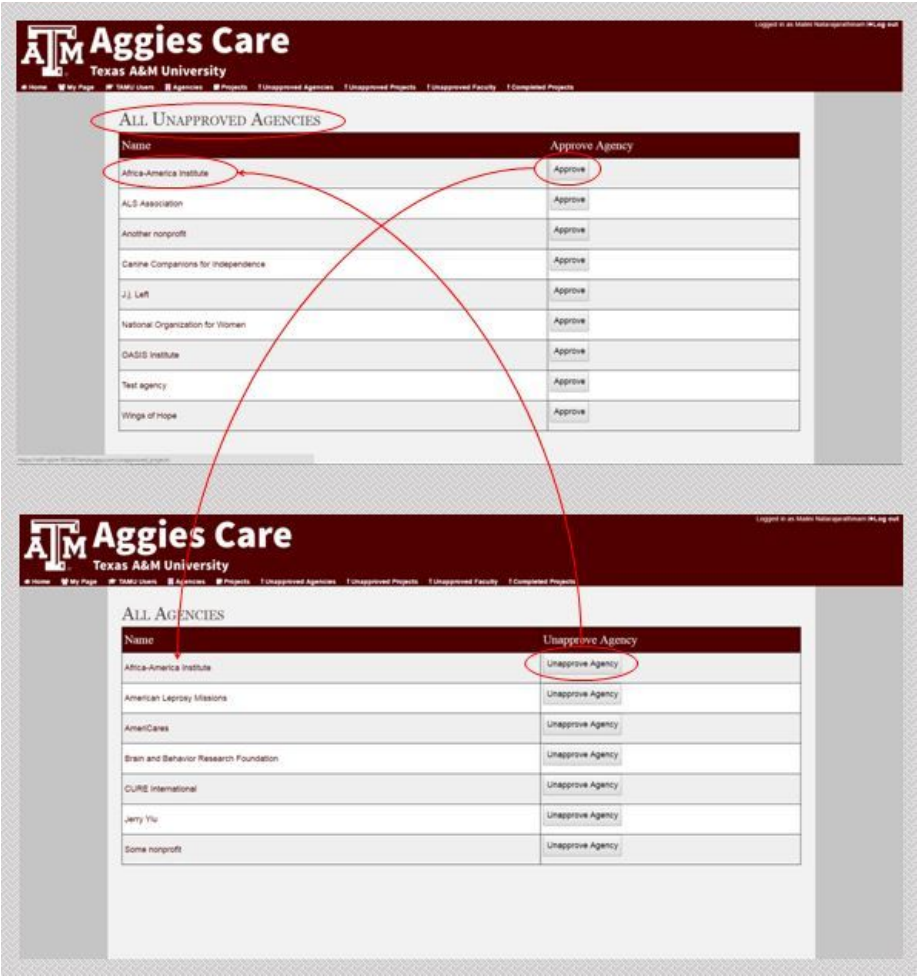

Figure 4 -- As an admin, he/she has the power to approve/disapprove an agency, or a user

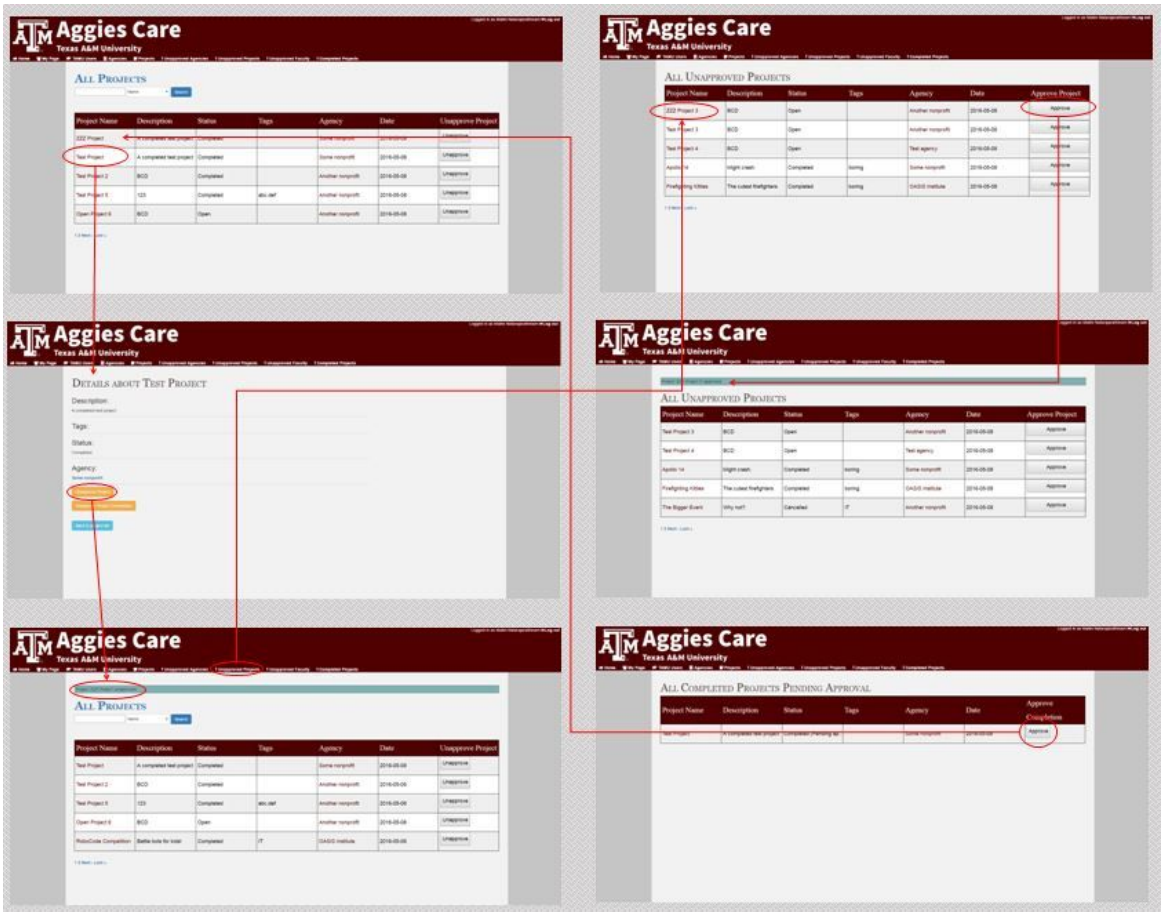

Figure 5 -- Admin also has the authority to approve or disapprove a project, and the completion of a project

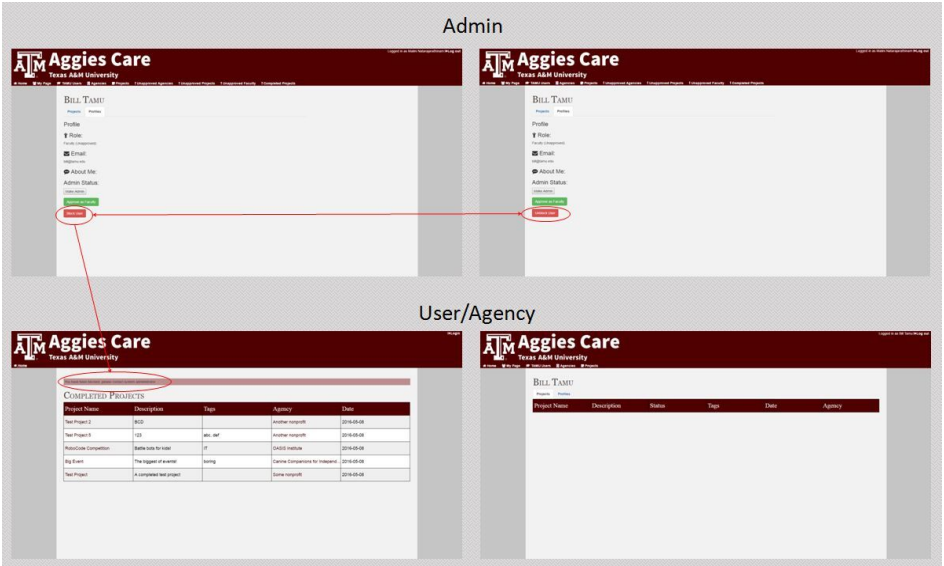

Figure 6 -- Admin can block a user and that user will no longer able to access the website until being unblocked.

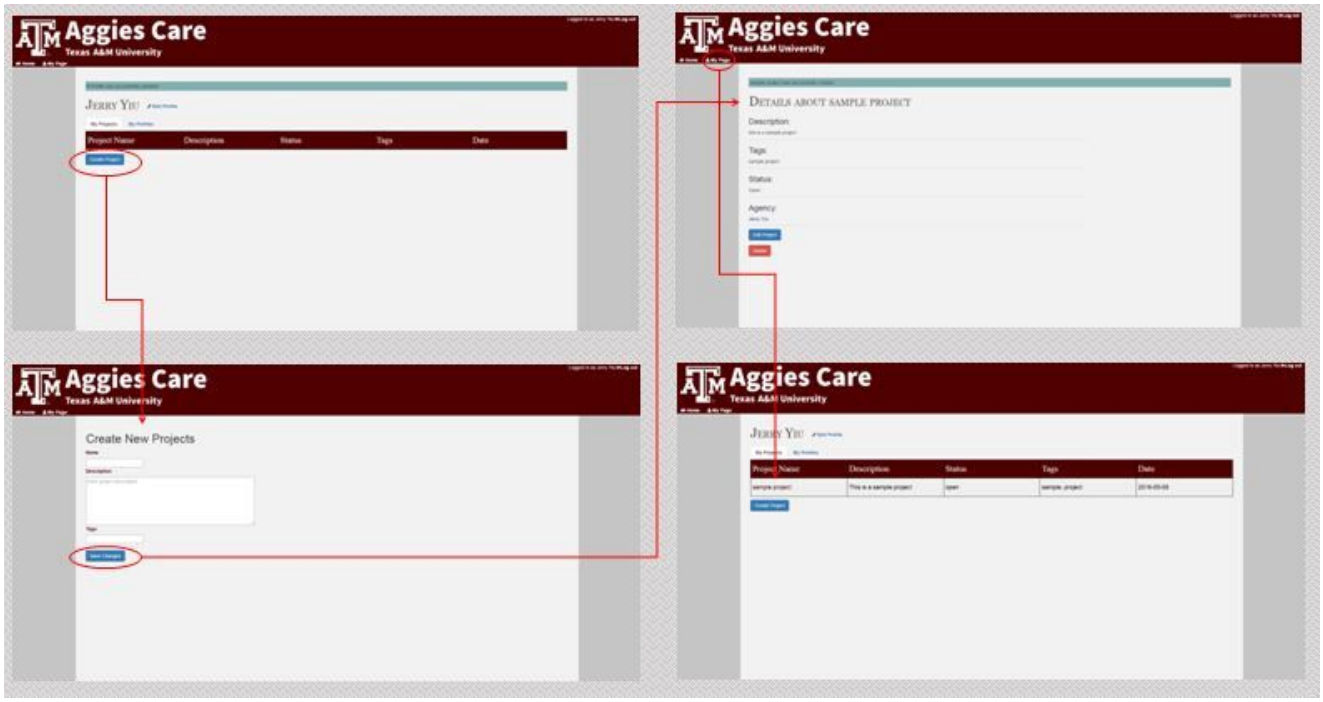

Figure 7 -- Agency can create, edit, and delete a project

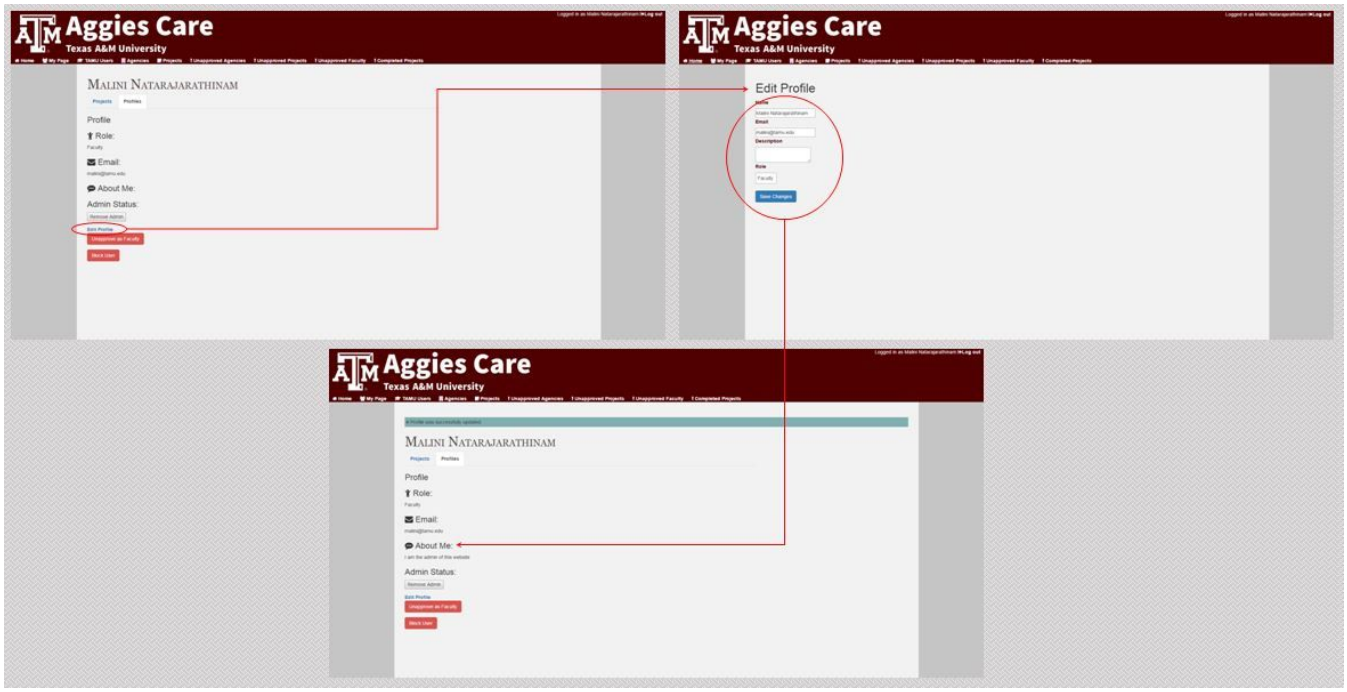

Figure 8 -- User and agency can edit their profiles

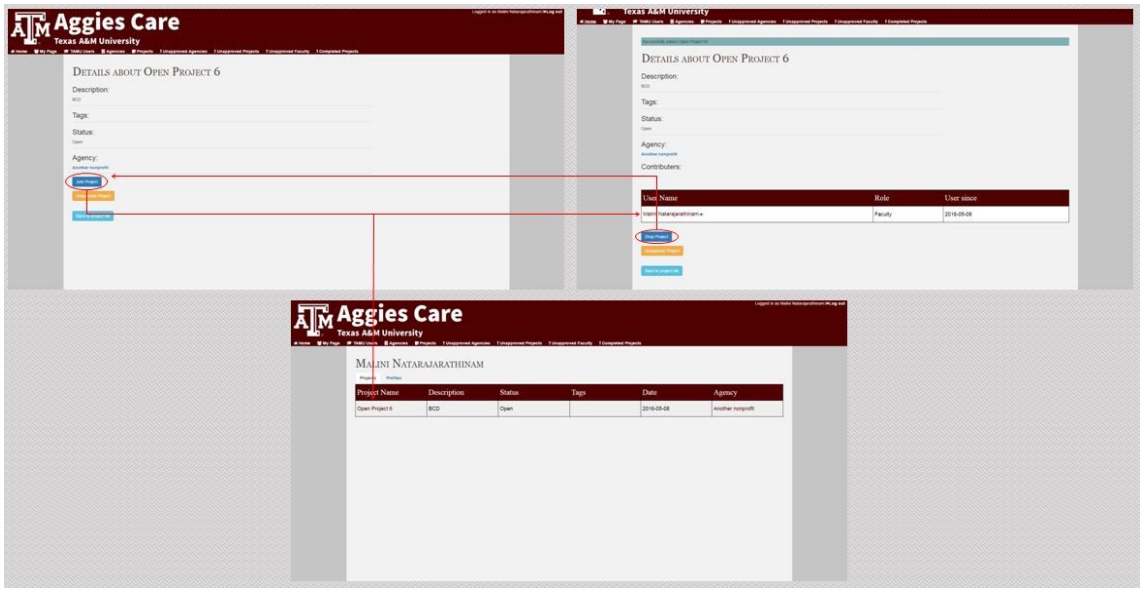

Figure 9 -- After an agency created a project, user can join/drop the project

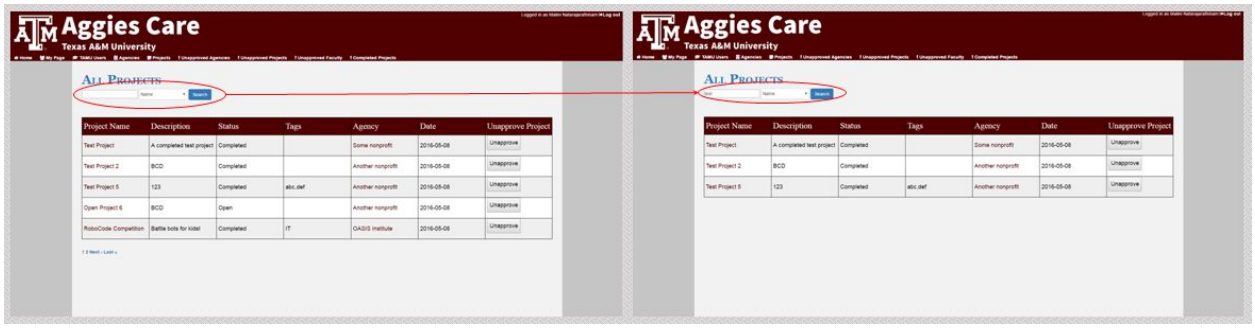

Figure 10 -- Users can search projects by name, agency, status or tags

Team Roles:
Scrum Master: Brennen Taylor
Product Owner: Evan Feiereisel
(no changes to team roles during project)

Scrum Iteration Summarizations:
Iteration 0: Project setup. No points completed.
Iteration 1:
Started with the basic functionality of the website, viewing things and a little bit of editing.
User Stories Implemented: Update Tamu User Contact Info; View Tamu User; View Agency; View Project; Edit Project.
Points Completed: 5
Iteration 2:

Did a lot of chores this iteration, and started with approval/unapproval of projects and agencies.
User Stories Implemented: Approve/Unapprove Projects; Approve/Unapprove Agencies.
Points Completed: 4
Iteration 3:
More chores and finished most everything to do with agencies.
User Stories Implemented: Log in as agency through gmail; Create profile as an agency; Edit agency profile.
Points Completed: 5
Iteration 4:
Added more power to admins, completed Tamu User functionality, and set up interaction between Tamu Users and Projects.
User Stories Implemented: Log in as Tamu User through CAS; Tamu User profile creation; Create a project; Add a make admin button; Delete a project; Subscribe to Project; Drop project as user; Admin approve faculty; Only master admin can delete admins; View completed projects; Admin block tamu user; Edit role of tamu user; Show project posted date in show and index; View project list with filters; Admin approve complete projects.
Points Completed: 22

Customer Meetings:
Date: 3/11/2016
Description: First meeting with the client, nothing to demo. Got an understanding of what the customer wanted for the website and initial user stories, got a 2 minute video.
Date: 3/30/2016
Description: Showed the customer all stories from iteration 1, update Tamu User contact info, view Tamu User, view Agency, view Project, edit Project. Got more feedback, especially admin capabilities such as approving/unapproving projects and agencies.
Date: 4/6/2016
Description: Showed the customer the approval/disapproval of projects and agencies. Got more feedback on the look of the website, agreed on google login for agencies.
Date: 4/20/2016
Description: Showed the customer logging in with google, creating a profile, and editing profile as agency. Learned about more admin capabilities (faculty approval, project completion approval).
Date: 4/27/2016
Description: Final meeting! Showed her all stories from iteration 4. Discussed the process of getting the code up and running on TAMU servers.

BDD/TDD Process:

Our BDD/TDD Process involved writing our cucumber tests first (failing), then our rspec tests (failing), then making our rspec tests pass and then our cucumber tests pass. At first there was a steep learning curve to both the cucumber and rspec tests which slowed down our development, but once we got the hang of it it became very useful in finding bugs introduced by new code and refactoring. Additionally, keeping our development branch at 100% SimpleCov coverage helped us ensure minimal bugs were introduced to that branch. But we found out that 100% statement coverage didn't translate to having sufficient tests as we found a bug in one of our views because we didn't update one of our features that was still passing.

Configuration Management:

We used GitFlow to manage our version control. GitFlow provides 4 different kind of branches: master, develop, feature, and hotfix. Using GitFlow made it easy to create and merge branches in a disciplined manner, so we had a branch for every user story. Whenever we finished a user story and its corresponding tests, we merged it into develop (always keeping develop at 100% SimpleCov coverage). Whenever we finished an iteration we merged all remaining branches to the develop branch so that everyone's branch would sync together. We would then release the develop branch to a demo branch where we could demo the project to the client. In this way, we had stable branches and other developers could keep on developing different features without affecting the demo branch. As for spikes, we have one for each of the login systems (Google and CAS), which are provided by the gems we used.

Heroku Issues:

- CAS: When the project was deployed on Heroku, the CAS authentication would not be available as CAS requires services be added to its registry. When hosting the server locally this was not an issue and when we can deploy to a TAMU server it should not be an issue. However, that meant to provide a fully functional website, we had to use the mock FakeCAS when pushing to Heroku.
- Database: Heroku only supports PostgreSQL not SQLite3 which rails uses by default. As SQLite3 is best to use for rails in the development and test environments, we put it under the test and development and PostfresSQL for production in the Gemfile.

Implementation Environment:

We used Cloud9 and localhost for implementations:

- Localhost: The local environment is fast and flexible, and everything can be highly customized. However, we have to carefully manage the versions of ruby and gems across different projects (if there are other working projects). We use RVM (Ruby Version Manager is) to resolve this issue, but still have to fix some compatibility problems from time to time.
- Cloud9: In Cloud9, each project has its own brand new environment, so we don't have to worry about the version control of gems. The development environment on Cloud9 can

be access from everywhere with browser, and it provides many handy build-in functionalities. However, there are some performance issues – the Rspec and Cucumber tests on Cloud9 is extremely slow (10s start up time)!

Tools/GEMs:

Gems: FactoryGirl, SimpleCov, RackCAS, Kaminari, Omniauth-google-oauth, Sass, Font-awesome

Version Control: Github and GitFlow

- Omniauth-google-oauth2: This gem is for authenticating users by Google API. It provides an interface to send/retrieve data to/from Google API. It also provides handy spike for testing.
- CAS (RackCAS): This gem provides a redirect to a CAS server in response to a 401 code. It has a built-in mock version called FakeCAS which we could use during testing and development to inspect the different page elements for the different types of users. One problem we had with it is the undocumented, hardcoded /login and /logout routes it comes with and the fact it doesn't immediately put user credentials in the session.
- FactoryGirl - This is a gem we used in rspec and cucumber tests. This gem allows us to quickly build and create models with ease for testing.
- SimpleCov - This gem is used to calculate the statement coverage of the codebase when running rspec and cucumber tests. We used this throughout the test writing process in order to ensure that all lines of code were executed by at least one test and ended up with an average of 9 hits per line as we improved our tests.
- Kaminari = This gem is used for pagination of the tables. This is used to split the display of large tables in our website into multiple pages so that not all the data is displayed on a page at once.

Interview Link:

https://vimeo.com/165724779

Demo Link:

https://vimeo.com/165822220