Contents

[Problem Statement 1](#_Toc352770117)

[Objectives 2](#_Toc352770118)

[Methodology 2](#_Toc352770119)

[Background 2](#_Toc352770120)

[Development 3](#_Toc352770121)

[Tools 3](#_Toc352770122)

[Platform 3](#_Toc352770123)

[Project Management 4](#_Toc352770124)

[Implementation 4](#_Toc352770125)

[Permissions 5](#_Toc352770126)

[Approval System 5](#_Toc352770127)

[Evaluation 6](#_Toc352770128)

[Criteria 6](#_Toc352770129)

[Testing 6](#_Toc352770130)

[Features 6](#_Toc352770131)

[Ease of Use 6](#_Toc352770132)

[Reliability 6](#_Toc352770133)

[Performance 6](#_Toc352770134)

[Failure Points 7](#_Toc352770135)

[Future Work 7](#_Toc352770136)

# Problem Statement

A normal campus organization has some basic records it has to keep, such as membership

and fees owed. This is even more pronounced in a fraternal organization where there may be

certain requirements members have to meet each semester. This can be overwhelming and

hard to coordinate between the various officers of the group. I have seen this problem first hand

in Alpha Phi Omega, a service fraternity I have been active with during my time at K-State. My

goal is to create a website for the Pi chapter of Alpha Phi Omega at K-State that will allow them

to easily manage chapter records.

# Objectives

Since I have been in Alpha Phi Omega as an officer for several years, I had a pretty good idea

of what things would be desirable for the website to be able to do. Beyond the basic static

content that was able to be viewed by non-members, I wanted to include these features:

● An account system with permissions to allow for users to sign in and have access to

appropriate functions.

● Allow officers to create news posts on the front page.

● Allow officers to create events that will be displayed on the site.

● Create an approval system to allow officers to approve various member submitted items.

● Allow members to submit photos for officer approval for the website

● Allow officers to submit minutes to the web page

● Allow members to report service and fellowship hours to officers for approval

● Show members an up to date overview of their standing with the chapter

● Allow officers to post money received from members for their individual accounts

● Show members their current financial obligation to the chapter

● Allow for chapter-wide budgeting, including allocations to various committees.

After speaking to the chapter, there was also interest in creating a chapter blog that members

could post stories from service projects, info about other groups or causes they were interested

in, or really anything that was related to Alpha Phi Omega or our core principles. But, I decided

to leave this out of the core features for the site since it isn’t critical to chapter operations. It is,

however, on a list for other features to implement in the future, which will be covered later.

# Methodology

## Background

The project I chose to work on this semester was to build a new website for the Pi chapter of

Alpha Phi Omega, a community service group at K-State. I have been the webmaster for the

past few years, and the website has always been a pain. We have been hosted off of the main

KSU server that groups have free access to through OSAS. The problem is, this only allowed us

to use basic HTML on the server side, so we couldn’t do much with it. All of our updates had to

be done by hand, which was a pain. I tried to work with it some to make it better by doing things

like loading posts out of a xml file using javascript, but it still was not all that great.

Near the end of last semester, I came across a program from DreamHost where they provide

a free domain name and web hosting to non-profit groups (New Dream Network, LLC, 2011).

After getting in contact with our national office to get our 501(c)(3) letter to prove Alpha Phi

Omega is non-profit, I was able to get our chapter into the program.

DreamHost provided us with a PHP server and access to a MySQL database (Oracle, 2011).

I began work over the summer on several features like basic user management, allowing

members to report service hours, posting minutes on line, etc.. Near the end of the summer,

I finished up enough stuff to make the site manageable, but the code was really ugly. For the

most part, it was one PHP file per web page. I had a few things broken out into classes, but

there weren’t that many. About this point, I decided to scrap my old idea for a senior project and

try to rewrite the site in a more maintainable fashion.

# Development

## Tools

I’m going to go ahead and talk about the IDE I used for this project, mainly because I switched

which one I used part way through the project. I originally started out using notepad++, which

is a really good notepad replacement. It’s not really an IDE, but it does offer basic syntax

highlighting, which was nice.

Eventually, I started playing more with CSS while I was tweaking the presentation. At this

point, the usefulness of notepad++ diminished, as I had to jump over to my array of web

browsers to see how everything rendered (Ho, 2011). Because of this, I started a brief stint with

Dreamweaver CS5 (Adobe Systems Inc, 2011). The main thing I liked about my experience with

Dreamweaver was that the Super Preview function is a nice way to see how things will render

across browsers. It’s basically a screen with sub-windows that holds how IE, Firefox, Chrome,

or whatever other browser you have on your system rendered your page. But, that’s about the

only thing I liked.

From my experience, it seems like Dreamweaver was made more for a graphic designer than a

coder. A lot of the PHP code I used couldn’t be rendered by the system for obvious reasons that

it required database interaction, but even then, it would usually just put a little PHP icon where

the output would go. This can be a problem when the entire page is in PHP. In addition, if you

touch the view window where your current HTML is rendered and dare to make a small change

through that interface, it will do horrible things to your code. I kept finding fixed pixel width tables

being used for my layout because I did something in the view without thinking about it. It also

makes some pretty ugly CSS and sticks it in style tags everywhere. Some of these things might

go away with some configuration, but I didn’t want to play around with it that much. Finally,

on the whole, Dreamweaver was pretty bulky and slow. It just didn’t seem like the best way to

develop.

So, when I began coding on this project again, I was looking for a new IDE. I settled on

NetBeans for PHP, and have loved it so far (Oracle, 2011). It is fast, lightweight, fairly simple,

and even has intellisence for the PHP libraries and functions you define on your own. Basically,

to sum it up, I highly recommend NetBeans as a development IDE.

## Platform

When I got started on this project, I was just learning about the MVC structure at work for a

new project we’re starting on. I really liked this setup, and tried to locate a MVC framework for

PHP I could use on this project. I found CakePHP, and fairly quickly settled on using that for my

project (Cake Software Foundation, Inc, 2011).

Once I started meeting with Dr. Hsu to discuss the project, Drupal was brought up as a possible

alternative for building the site (Drupal Association, 2011). I looked into it more because Drupal

would be much easier for future webmasters to work with if they needed to do something like

add a new page, as opposed to trying to figure out my custom code. I actually decided to go

with Drupal, and began trying to work on it more.

I discovered that while Drupal appears to be a great platform, it takes time to learn. I couldn’t

find any particularly good tutorials right off the bat, so I was working off of several different

tutorials plus some incomplete documentation. In Drupal, over the base website, you have

a theme which customizes how the site looks. I worked with two different theme packages,

Zen and Omega (Wilkins, 2011, Development Geeks, 2011). I was able to get both of them to

the point where the basic features looked the way I wanted them. About this point, however,

I started running into problems with my content. As far as I can gather, what types of content

you can post are configurable, and their presentation changes based on the configuration. The

basic Drupal install I had provided static pages and news articles. Static pages got their own

page, while news articles would render on the front page in a news feed. There were some

presentation things I didn’t like with the news article format, but I left it alone for the time. What

I was concerned with is that I already had my database designed, and was pulling a lot of

information out of it for display to the user. I didn’t see any obvious way in Drupal to imitate that

behavior. I am sure there is somewhere, but about this point I was looking ahead and seeing I

was going to have troubles having anything presentable by the end of the semester at this rate.

Because of this, I decided to drop Drupal and switch back to CakePHP.

When I switched to CakePHP, development sped up considerably. A lot of this was due to the

fact that I wasn’t trying to work with someone else’s system I had to learn anymore; I was writing

my own. However, a lot of my initial progress came through the CakePHP bake script. If you

know Ruby on Rails, I know there’s a script like this with it too; basically, it can look at your

database schema and build basic models, controllers, and views for each of your tables. After

bringing my CSS files over from my old project and adding some controllers and views for static

content, I had the site at the same look we had on the KSU server, and it was done in about a

day.

In summary, I’m sure Drupal is a great platform; the problem I had with it is that it is too

configurable to learn quickly. If I had no time limit or if I knew Drupal better, I’m sure I could

eventually end up with close to the same site as I am coding in PHP. However, based on the

time restrictions of a semester, CakePHP was the better choice for my project.

## Project Management

I’m just going to briefly touch on this, as you can’t have too much of a methodology in a one

man programming team. But, throughout the semester, I was using agile methodologies; scrum

near the beginning of the semester and more of a lean process near the end (Scrum Alliance,

2011, Lean Enterprise Institute, Inc, 2011).

A very basic overview of scrum is that you have a list of tasks that need to be done. The team

identifies which tasks they feel they can accomplish in a sprint (a period of time, typically a few

weeks) at a planning meeting. These features are then worked on, and at the end of a sprint,

the product should be in a workable state, with those features completed. I started using scrum

off the bat because it fell into line with everything pretty easily. At my internships at Softek, we

used scrum for our development, so I was used to the system. In addition, my weekly meetings

with Dr. Hsu lined up perfectly with my usual sprint length of a week. So, every week, I would

pick a few goals to work on and keep moving through the features like that.

However, near the end of the semester, it was getting into crunch time and having just a few

goals wasn’t going to work. I switched over to a lean process, which is like scrum, except

instead of having defined sprints and tasks for a sprint, there is a continual backlog of features

to work on. This allowed for much more flexibility to code as time came up, so I was able to

churn through as many tasks as possible when I had free time.

With both of these processes, I was able to use what is called a kanban board to manage what

tasks I had remaining. Specifically, I used an online implementation called LeanKit Kanban,

which allows users to create different types of tasks such as features, improvements, and

defects, and tracks the process as the cards move through the lanes of the kanban (Bandit

Software, LLC, 2010). In a kanban, the lanes represent the various stages of development, such

as a backlog, planning, in development, in review, and completed. I found this tool very useful

throughout the project, as it allowed for a visual representation of what needed to be worked on.

# Implementation

I’m going to focus in on the implementation of two main features: the permissions system and

the approval system. This is because these two systems really control the flow of everything in

the website; the rest of the site is essentially just acting as a front end for the database.

## Permissions

Back in my prototype application, I had set up a system that allowed different officers certain

admin privileges, e.g. the Vice President of Service has the Service permission, the Vice

President of Fellowship had the Fellowship permission, the President and Webmaster were

given all of them, etc. This way, not only was I able to keep unauthorized people out of the

system, I could filter down what each officer was shown to just what was relevant to them (and

what they were allowed to do).

This system was implemented by linking permission names to different positions in the

database. I chose this setup as it was fairly easy to do and also fairly expandable in the future

for new officer permissions.

When I made the transition to start using CakePHP, I realized that I needed to also give user

groups permissions in order to help differentiate between actives, pledges, alumni, and other

people. So, I linked the same list of permission names to the MembershipType table, so I was

able to reuse the permission names the officers were using. This approach seems to work

fairly well for the MVC framework. Using CakePHP’s Authorization component, I was able to

set up routing so there were three sections to the website: open, member, and admin. To get

to the member section, you have to be logged in, and to get to the admin section, you have

to be an officer and be given the ‘Admin’ permission. Outside of the ‘Admin’ permission, the

different permissions correspond to the different controllers in the website. So, suppose a

member is logged in, is in the members section, and tries to upload a photo to the scrapbook.

What the website does is check the routing prefix first, which is ‘member’ in this case. Since it’s

member, it then looks at the user’s member type, and checks the member type’s permissions

for the ‘Scrapbook’ permission, since these uploads are handled by the scrapbook controller.

If the member has it, then it user is sent through to the upload page. If not, then a message

indicating that the member is not authorized to do that appears. Similarly, if an officer is trying to

look at the photos submitted to the scrapbook, they will be in the admin section, so the system

looks at the officer’s position’s permissions, and checks for the ‘Scrapbook’ permission. If they

have it, then they are let in; if not, then they are informed that they are not allowed to do that. I

should also note that unauthorized messages should be fairly rare unless someone is trying to

get somewhere they are not supposed to be directly, because if the user is not authorized to go

somewhere, then the UI will not show them the link to get there. They would have to be entering

the URL manually or using bookmarks.

## Approval System

The approval system is tied in to just about everything on the website, from events to photos.

This is because while we want to trust our members, we want to make sure that an appropriate

officer checks over something before it gets counted for the member or is posted to the website.

The approval system is fairly simple on the whole; for any piece of data that requires approval,

it contains a reference off to the approval table. The approval table stores data on whether

something has been approved or not, the reasoning behind the decision, who reviewed it, and

the date it was reviewed. If approval on a record is pending (null) or denied, then it is not shown

in normal web content; the only place it can be seen on the website is in the admin section

under the appropriate approval area.

Basic example, say a member submits a service report to the chapter. An event is created in

the database with the information on the report, and an approval record is created as pending.

Eventually, and officer logs in and looks at the report and approves or denies it. If it is denied,

then the member is informed so they can talk to the officer about it if necessary. If it is approved,

then the service event (and the hours associated with it) are now counted for the member in

their standing with the chapter. The only problem with this system is that we don’t want to make

officers approve their own event when they create one for the chapter. Because of this, it is

possible to create an event in the admin section. When an event is created here, the event

is automatically marked as approved by the officer who created the event, since the creating

officer will typically be the one who is reviewing the reports anyways.

# Evaluation

## Criteria

There are a few key requirements I have for the website. First, naturally, it has to be able to

perform all of the features that I outlined for it earlier in the paper. Second, it needs to be easy

to use, as non-technical people will be using and managing the website after I graduate. Third, it

needs to be reliable, as it will be maintaining the chapter records. Fourth and finally, it needs to

perform reasonably well speed wise.

## Testing

### Features

Testing for features was done manually by just poking around with the website with some test

data. I did try to cover the edge cases with the test data, but nothing with automated testing was

done. I think I saw unit testing available in CakePHP, but I didn’t look too deep into it because of

time constraints. If I had more time, it would have been nice to add unit tests though.

### Ease of Use

To test for this, I did was is called Hallway Testing. The idea of this is you grab the first person

walking by and ask them to do something in the app you’re working on. You then are able

to see what someone who is not familiar with your application does when they try to perform

a task. All of my applications passed this testing with my roommates. In addition, I gave a

presentation to the chapter on using the website, and there didn’t appear to be any confusion in

how to use the site.

### Reliability

From what I have been able to see, CakePHP plays fairly nicely with databases, so there

should not be too many problems with data. Just in case, DreamHost provides easy ways to

run backups on the database. I have not implemented this yet, but have begun working on

automating this process using a cron job. The next step would be to automate sending a copy of

this backup off site, so the site is not relying entirely upon the DreamHost servers.

### Performance

Performance testing was minimal overall. For the most part, it was done by just poking around

on the site once it was hosted on the DreamHost server. This is because when I was running

it locally, the site was pretty fast. But, after putting it on the server, it slowed down some.

However, it still performed fast enough for me to consider it acceptable. The slowest page to

load was the main My Account page with all of the reporting on it, and even this page sped up

after the first load, due to caching by CakePHP.

I don’t have too much to go into on this, but MySQL requires indexes on foreign keys in the

database. As most of the things we are looking up go by IDs and we don’t use the values for

lookup, this should help performance as the data begins to build up in the database.

## Failure Points

The one key failure point I see for the site is server load. I have not run any load testing on

the application, so I do not know how either my application or the hosting server will hold up.

However, based on the size of the chapter, I do not expect many issues to come out of this.

Our current membership is about 15 people, so that will be the normal active people on the

site. Even then, they will likely not be logging into the site all that often; perhaps once every few

days for a few minutes at a time if they are extremely active. As there is currently no other main

content to draw them back daily, I think this is a reasonable assumption. A problem could come

from the alumni base. The Pi chapter has approximately 1000 alumni members. However, only

two hundred have email addresses on file, narrowing the range of people that we could contact

to register for the site. Even then, we can assume that just a fraction of them will do much

more than register for the site, as the current members area does not have too much to offer to

alumni.

# Future Work

I have a list of features that while they do not fit in the scope of this project, the chapter has

agreed that they would be useful and would like them to be added in a followup project. Among

these features are:

● Chapter forum - This would be used to help with chapter discussion. A possible implementation for this would use phpBB, a open source forum software that is fairly

widely used (phpBB Limited, 2011).

● Chapter blog - This has been discussed with the chapter to see if it is something they

would be interested in. The idea is to let members post on the blog about things like

service events, or other things that are related to the ideals of Alpha Phi Omega. I still

have not decided if this will be an ad hoc implementation, or if the chapter would want

more features that would be provided through other software, such as WordPress

(WordPress, 2011). The downside of using something like Wordress would be the

difficulty of integrating it with the current website’s look.

● Mailing List - The chapter currently uses a Google group to track who to email for

announcements. However, this is hard to maintain, and we have wound up with people

who did not want to be on this list getting re-added by accident. DreamHost provides a

mailing list implementation I will likely use for this feature.

● The ability to pay dues, etc. through the site using PayPal - A future plan is to automate

some of the payments for things, such as events. We could let a member RSVP for an

event, then issue a charge against their account. To fully automate this process, we

could let members pay their dues online. I held back on this because of PayPal’s fees.

But, I noticed that they only charge their fees on credit card transactions. If the member

pays through a checking account, there are no fees on the transaction. So, one key

thing we have to be able to make sure on this is that we can restrict members from using

credit cards to avoid the fees.

● Off-site database backups - This was discussed earlier earlier, but this should be done

to ensure that we don’t lose data to errors in the DreamHost servers.

I have actually already talked to the chapter about this, and have been given approval to keep

my admin access to the web server and site after graduation in order to continue to implement

new features for the chapter to use. To help with this, I plan to set up a basic contact form in the

admin area to allow officers to request new features or report bugs as necessary. This way, with

how I send the emails out of the site, I can set my email to flag them so I can make sure that I

am checking what things the chapter is requesting.