PROJECT CAPx

Prepared for: Dr. Joseph Clark

Prepared by: Steve Truong, Alex Neumann, Jose Recendez, Thomas Rullestad, Sneha Dadhania

Date: April 30th, 2015

Class Time: T/TH, 9:00 AM

Table of Contents

Executive Summary

As part of the required coursework for the Computer Information Systems (CIS) degree at the W. P. Carey School of Business (WPC), students take part in a semester long software development project, where they work with an external client, and demonstrate the skills and best practices that they obtained throughout the degree classes. The professor that teaches the capstone class, where this project is completed, connects with various clients to obtain information about projects that they would like to propose to student teams. In all semesters prior to the Spring 2015 semester, the professors would send out emails to clients and request that they email the professor back with any pertinent information for a project that they want to propose. This became somewhat tedious, as professors needed to print out each email and show it to students in class. The tediousness of this process led to the birth of Project CAPx.

Project CAPx is a web development project that aims to provide a centralized repository that clients can use to propose software development projects to the WPC, CIS seniors, who are in their final CIS course - CIS440: Systems Design and Electronic Commerce. Our team, (Alex Neumann, Steve Truong, Jose Recendez, Thomas Rullestad, and Sneha Dadhania) chose to undertake Project CAPx as our own capstone project. Our client is, Dr. Joseph Clark, a Clinical Assistant Professor at the W. P. Carey School of Business and also, one of the professors for the capstone course.

Throughout the Spring 2015 semester, we designed a website that allows clients to not only learn more about the capstone course, but also take a look at various projects that have been completed in the past, in addition to being able to propose their own projects. The site allows students to view current projects that are available in their semester and also gives them the ability to vote on projects that they think are neat, as well as allowing them to express interest on a particular project. The official site for this project is www.cis440.com, and it is powered by various technologies: Python Flask web framework, Heroku, Git, PostgreSQL, Amazon S3, Bootstrap, HTML, CSS, JavaScript, and JQuery.

Overview

[Client]

Our client is Dr. Joseph Clark, a Clinical Assistant Professor at the W. P. Carey School of Business.

[Problem]

There is a lack of a centralized, simple, straight forward way for potential clients and students to propose projects for the CIS 440 capstone course. Also, there are times when clients will send an email that is multiple pages long that describes their proposal in extreme detail. Initially, students should be given a brief description of the proposal, and the www.cis440.com site allows that.

[Proposed Solution]

Develop a website specifically for the CIS 440 Capstone class. The site will allow both students and clients to post project opportunities. In addition, there will be an area that shows various past projects. Students will be able to view all available projects and rank them with a up/down vote system. Students will also be able to express their interests for projects directly on the site. Each project will have information provided by the company as well as Dr. Clark's comments. Additionally, each project will have tags, which give a quick overview of the desired coding language, required technologies, and other characteristics.

How to Install

If you would like to use our code, please navigate to our Github Repository and download a copy of the entire repository to your machine: https://github.com/asu-cis-capstone/capx . Once you have a copy of the repository, please follow the instructions listed below to get a functioning website:

1. Navigate to https://devcenter.heroku.com/articles/getting-started-with-python#introduction and follow the instructions up to installing the Heroku Tool Belt. The tool belt with allow you to use Heroku commands in your native terminal application. The Heroku guide will also provide links for getting Python installed on your machine. Create a Github account and download Github for your machine. Lastly, you will also need to create a free Heroku.

*** Note that if you are running OS X, you will need to download and install xcode from the App Store to be able to run Git commands from your terminal ***

2. Within your new Heroku account, create a new Heroku database via the Heroku Profile Dashboard. Then, download PGAdmin (http://www.pgadmin.org/download/) for your operating system. PGAdmin will be used to manage the Heroku database.
3. Open PGAdmin, and connect to the Heroku database that your created (check out the Heroku Dev Center if you need assistance with this - https://devcenter.heroku.com/ .
4. Once connected to your Heroku database, use PGAdmin and the CAPxSQL.sql file from the CAPx repository that you downloaded, to create the correct tables, attributes, and dummy data on your Heroku database. You will now need to make some modifications to the *app.py* file, which can be found in the Github Repository folder that you downloaded. Using your favorite text editor, modify the following line of code to match your Heroku database variables:

```
4  conn = psycopg2.connect(database="MODIFY", user="MODIFY", password="MODIFY", host="MODIFY", port="MODIFY")
```

5. Now open a terminal window (on OS X) or Github Shell (Windows), change directories into the CAPx repository folder that you downloaded, and type the following commands:

a. **git init** = initializes a Git repo for all your code
b. **git add –A** = adds all files to the stage for being committed to Github
c. **git commit –m 'INSERT A COMMENT'** = commits all files to a Github Repository
d. **heroku create** = creates a new Heroku App on your Heroku account
e. **git push heroku master** = pushes all the files in the stage to the new Heroku App

6. Next, navigate to your *Heroku Dashboard > Personal Apps*. You should see a new Heroku App listed. Launch the new app. If you receive an *Application Error* page, you most likely need to scale the web dynos on Heroku back to one. In order to do this, open the same terminal window from **Step 5** and type the following command:
   a. **heroku ps:scale web=1**
   b. Try to launch the app again.
7. Once the app successfully launches, the Github Login functionality should be added. To do this, open up the *app.py* file and find the following lines of code:

```
13  app.config['GITHUB_CLIENT_ID'] = 'INSERT YOUR CLIENT ID HERE'
14  app.config['GITHUB_CLIENT_SECRET'] = 'INSERT YOUR CLIENT SECRET HERE'
```

   a. Your app should be registered with Github in order to obtain these variables. If you do not want to have the Github login functionality, you may skip this step.
   b. If you would like to add Github login functionality, replace the appropriate lines of code with your Github Client ID and Secret.
8. Now Amazon Web Services S3 needs to be set up so that the company logos can be stored somewhere. In order to do this, follow these steps:
   a. Install the Boto Extension to make a connection to S3 using Python (from terminal, type *pip install boto* to get boto installed)
   b. Set up a bucket on S3 and give it any name you would like.
   c. Now you just need to add your *AWS_Access_Key_Id* and *AWS_Secret_Access_Key* to the Heroku App Config Variables on your Heroku Dashboard (see the [Application Config Variables] section to learn how to accomplish this). Finally, establish a connection to S3 using the following line of code:
      i. boto.connect_s3(ACCESS_KEY_ID_HERE, SECRET_KEY_HERE)

9. You should now be completely setup and ready to go!

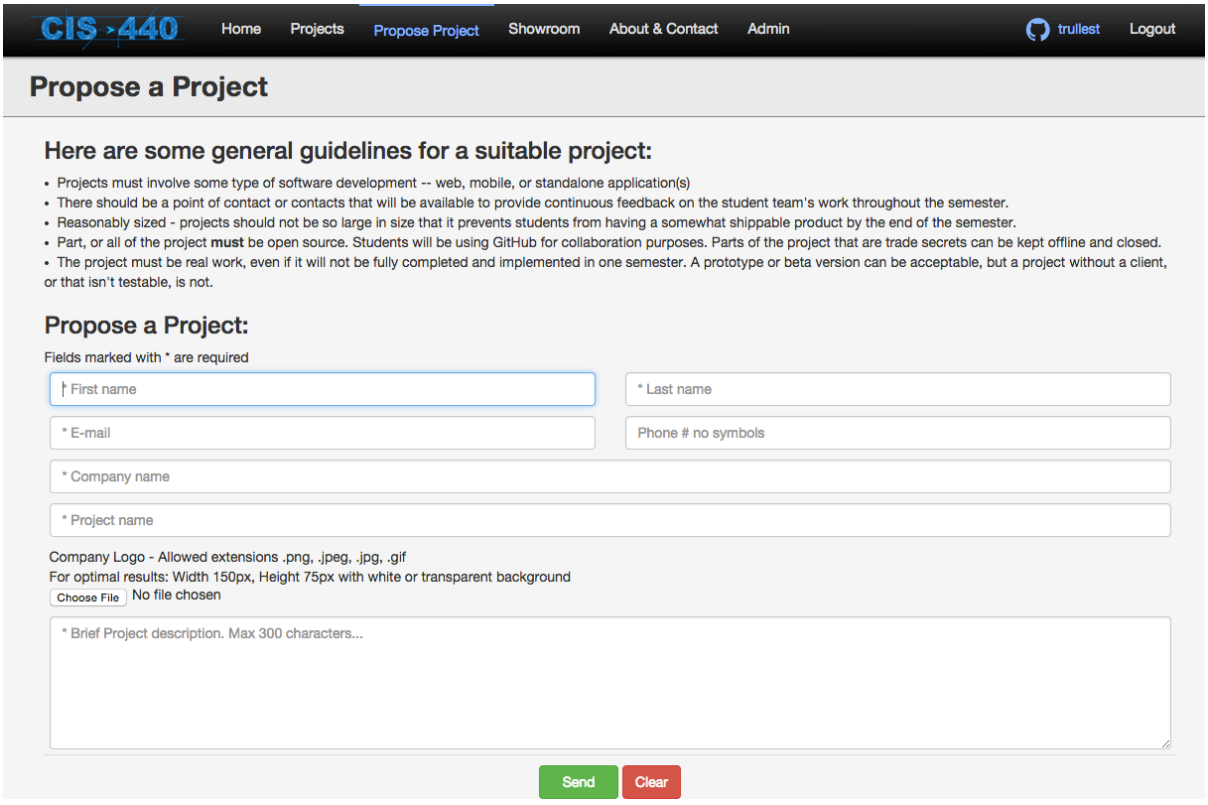
Usage Guide

The CIS440 site has three different users. Depending on the type of user, the site functions vary. This section will walk you through the various functions/features available to each user, as well as how to use them. The first step is the same for all users: Navigate to www.cis440.com .

[Clients]

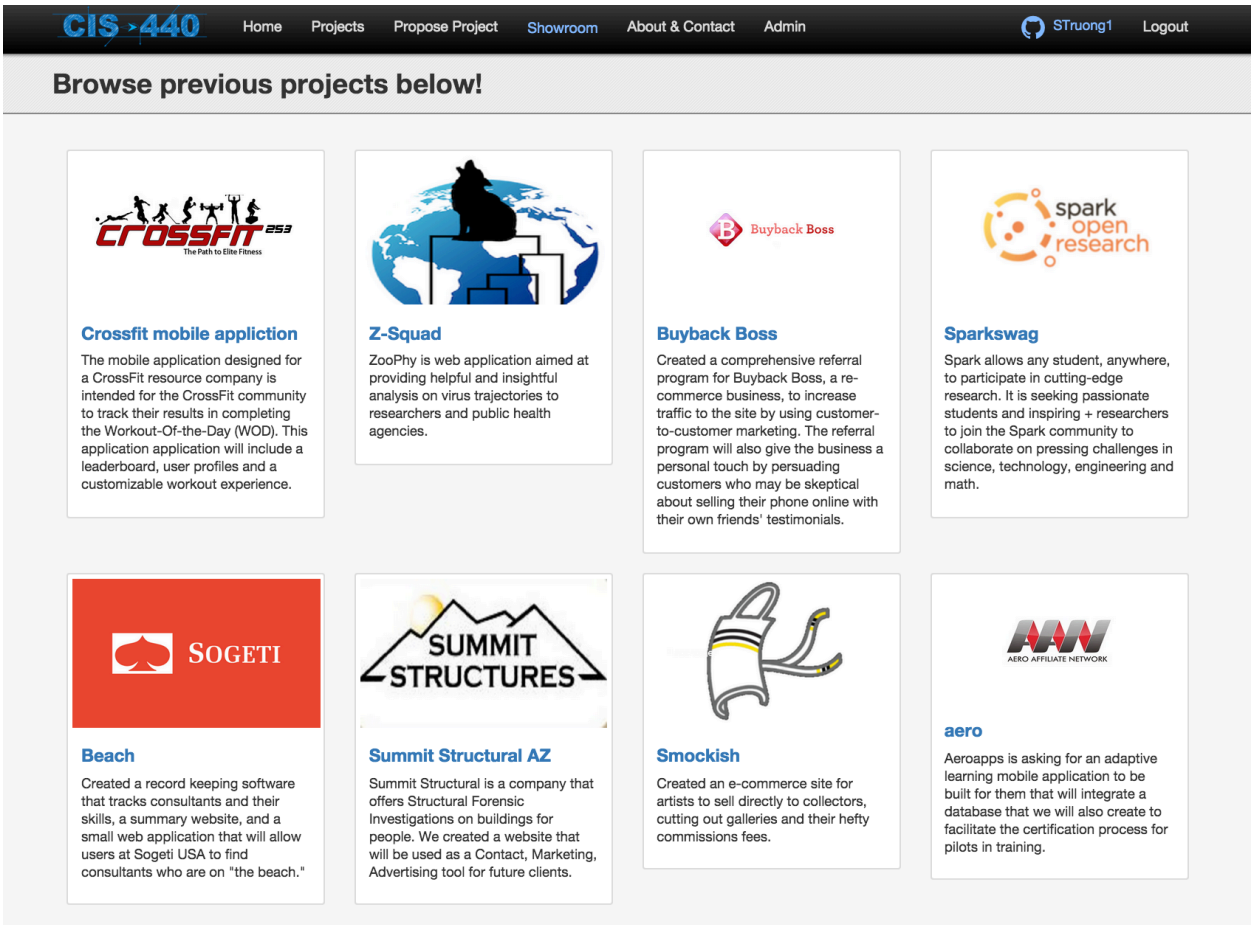
Clients are able to access any page on the site, with the exception of the Admin Panel, which is exclusive to administrators. With that being said, there are only a few features that are really of use to clients.

Clients will mainly be using the site to propose potential projects for students. They can utilize the *Propose Project* page to submit an idea for approval by the course instructor. In order to submit a project, clients will navigate to this page (*Propose Project*) using the navigation bar. The page looks like this:



On this page, clients will find some general guidelines for what a good project should be. Directly below these guidelines, the client will find a form that they can fill out. In order to propose a project, clients need to fill the form out and click on the *Send* button to submit it to be approved by an administrator. Clients are encouraged to upload a company logo, as it makes their posting look a lot more professional. Once a project is approved by an administrator, the project will appear on the *Project* page.

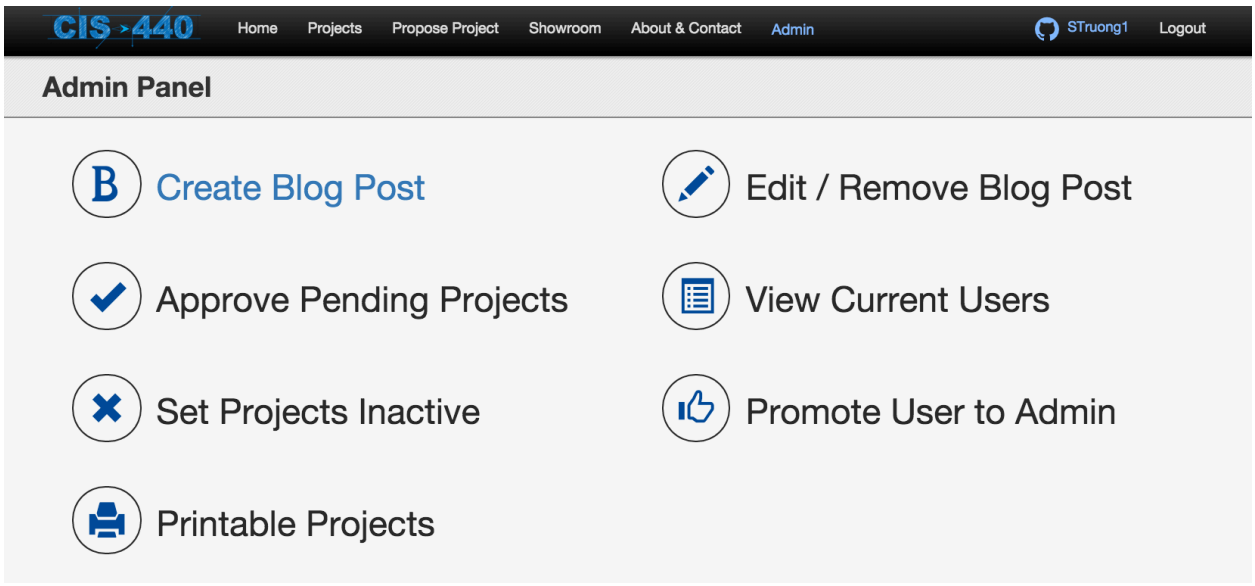
One other page of interest to clients is the *Showroom* page. The *Showroom* page highlights various projects from previous semesters with a description of the project, and a link to the projects Github Repository. This page will allow potential clients to get an idea of the types of projects that have been completed by student teams and gain a better understanding of the types of projects that they should propose. This is what the page looks like:



Clients may also be interested in the *Home* page, where the professor may post various blog updates on how the semester is going or various challenges students are overcoming. In addition, the *About & Contact* page is also useful, in case the client wants to contact the professor with questions before submitting a project or something of similar nature.

[Administrators]

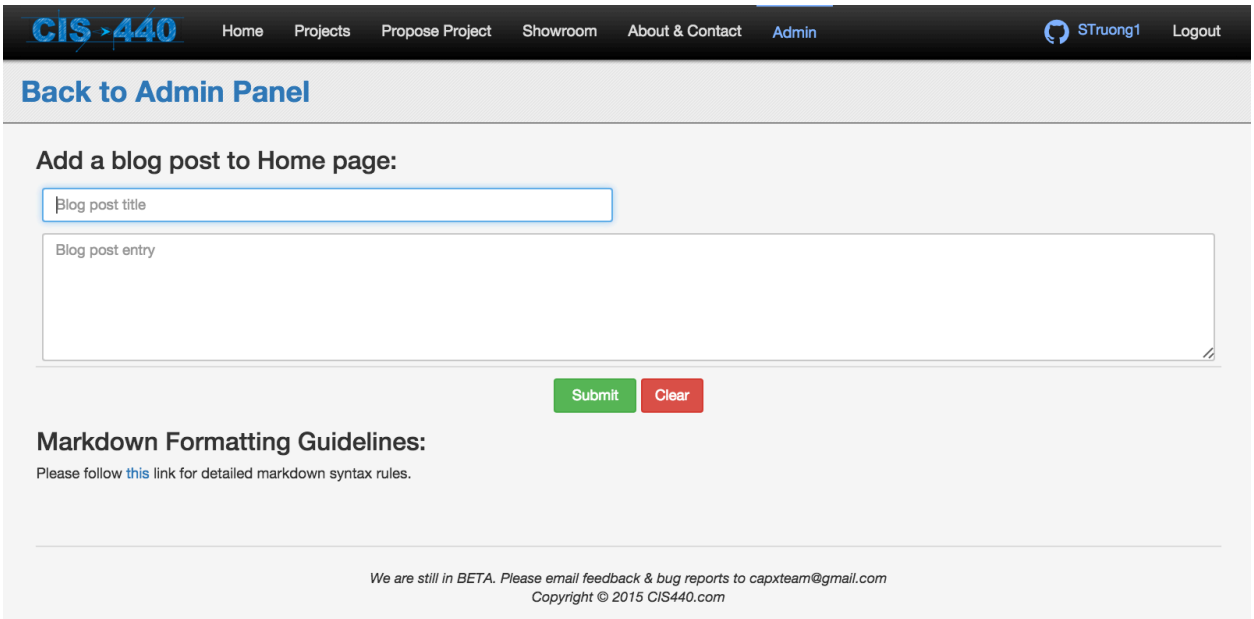
Administrators have a plethora of features that are available to them. First and foremost, they can do anything that a client can, in addition to much, much more. In order to access administrator features, you will need to be logged in and granted administrator rights by a current administrator. Once done, there will be a new page in the navigation bar called *Admin*. This is where all the administrative features are accessed. Here is a quick glance at what the *Admin* page looks like:



All the items listed on the *Admin* page are features that are only available to administrators. Let's take a look at each one in a little more detail:

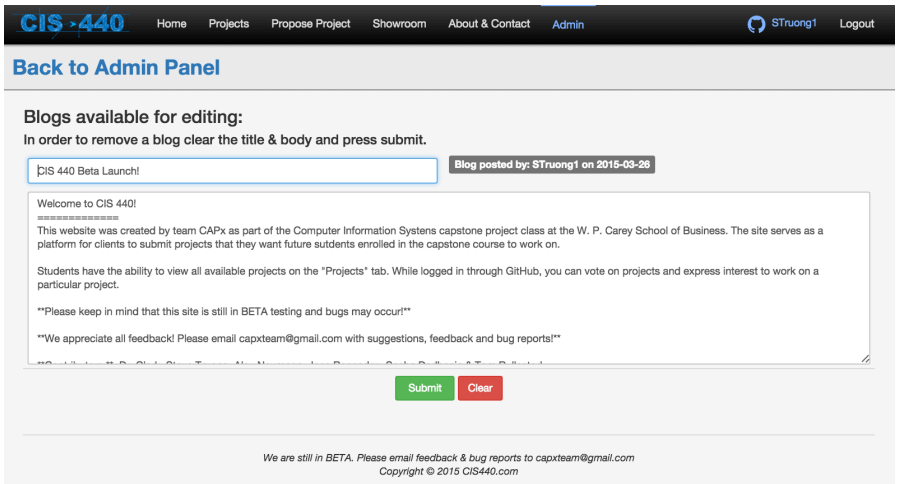
[[Create Blog Post]]

Administrators can create a blog post that is displayed on the *Home* page. The body of the blog post should be written in Markdown (same language used for Github README.md files). All markdown will render correctly when the blog is posted. Here's what the *Create Blog Post* page looks like:



[[Edit / Remove Blog Post]]

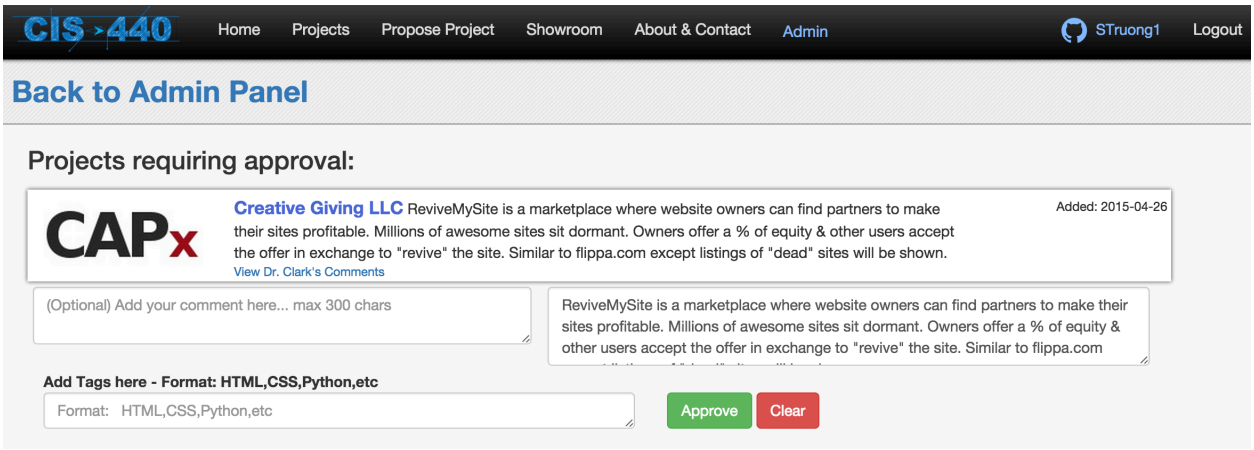
In order to edit or remove a previously posted blog entry, an administrator would click on the *Edit / Remove Blog Post* link on the *Admin* page. This is what the page looks like:



Any blog post that is currently displayed on the *Home* page will be shown here. In order to edit a post, simply make your edits in the appropriate textboxes and click *Submit*. In order to delete a blog post, clear all text/characters from the title and body of the post that needs to be deleted, and click *Submit*.

[[Approve Pending Projects]]

Once a project has been submitted via the *Propose Project* page, administrators have the ability to add comments about the projects, add technology related tags, or update/edit the description that was provided by the person who submitted the project proposal (in case there are any errors). All this is possible with the *Approve Pending Projects* page, which looks like this:

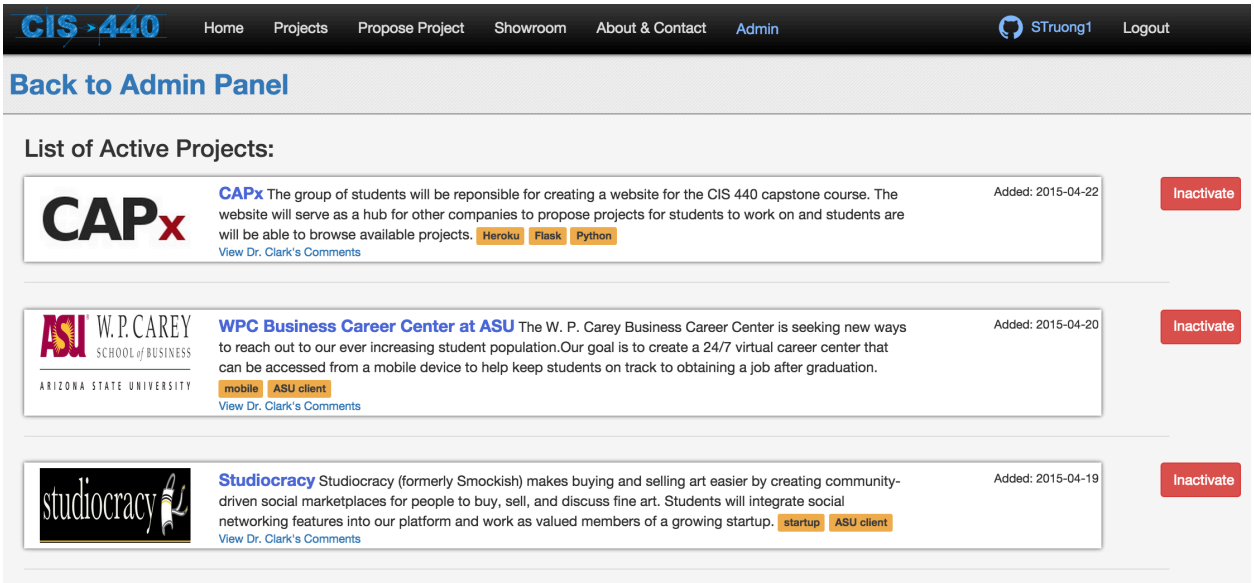


The project posting can be seen with three textboxes under it. The first empty textbox is an area that administrators can use to post their own comments on the project. The second empty textbox is where tags can be added. Lastly, the third textbox, which contains the project description, can either be left alone or edited, if needed. Once complete, clicking the *Approve* button will send the project to the *Projects* page.

[[Set Projects Inactive]]

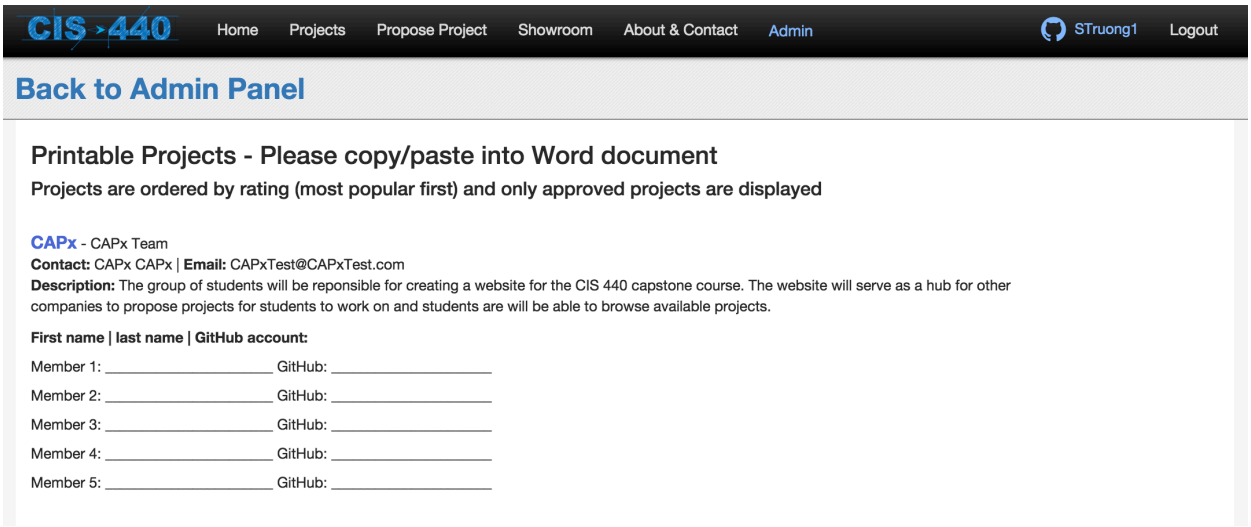
The *Set Projects Inactive* feature allows the removal of projects from the *Projects* page. It's very simple to use. When on the page, a list of the projects that are currently visible on the *Projects* page is displayed. Next to each project, there is a button that says *Inactivate*. When clicked, the

project will be removed from the *Projects* page, and sent back to the *Approve Pending Projects* page. This allows the administrator to keep all projects in the database. Here's an example of what the page looks like:



[[Printable Projects]]

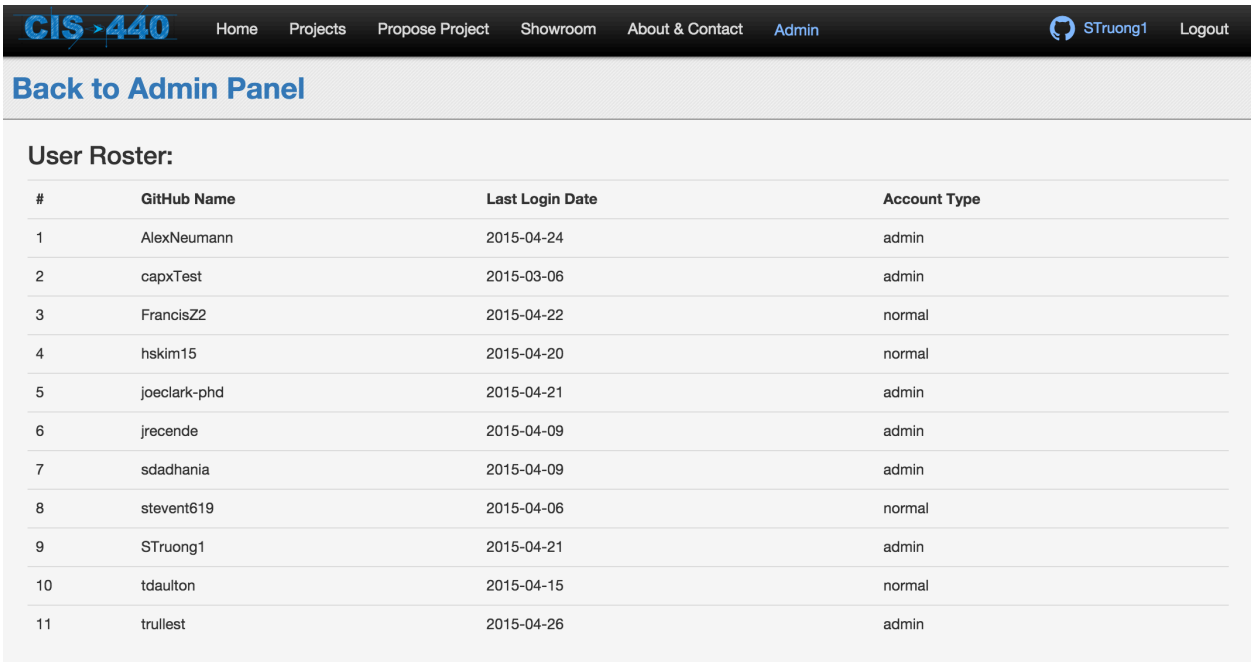
The *Printable Projects* page is simply a neatly formatted sign-up form that professors can copy and paste into a text editor and print out for in-class project sign-ups:



This page also contains contact information for each client, so it is handy to have! Also, only projects that have been approved will show up on the *Printable Projects* page.

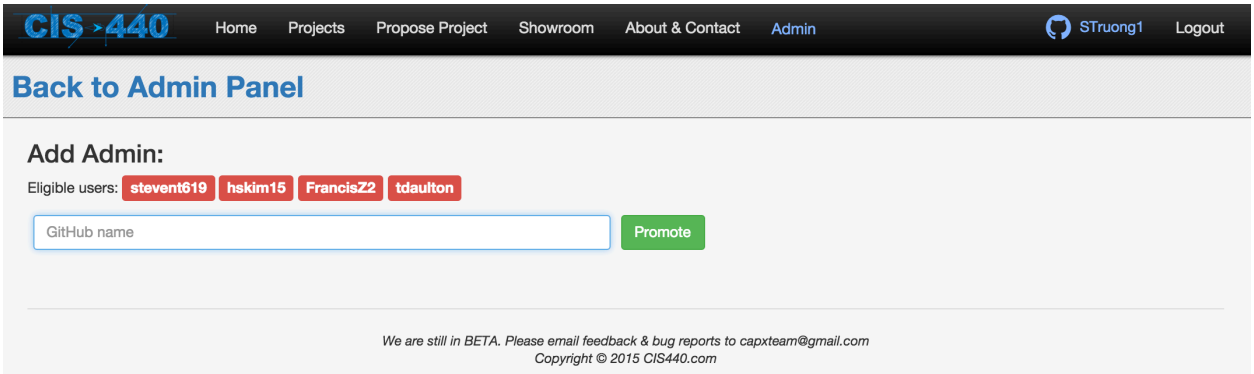
[[View Current Users]]

This page lists all current users of the site in addition to their last login date as well as each users account type (normal or admin):



[[Promote User to Admin]]

The *Promote User to Admin* feature allows a **current administrator** to promote any regular user to an administrator. To do this, navigate to the *Promote User to Admin* page, make sure the user is eligible to be promoted (all users who are not administrators will be shown), and type their Github name ***exactly as it is*** (including case) into the textbox, and click *Promote*:



[Students]

All of the student-specific features are found on the *Projects* page. They are pretty self explanatory, so here's a comprehensive list of what students can do (assuming that they are logged into the system):

- Up/Down vote on projects (using the arrows on the left side of a project)
    - Projects on the *Projects* page are rearranged according to ranking
- Express interest in a project (by clicking *Express Interest* in the right corner of the project box)

The Story

CIS 440: Systems Design & Electronic Commerce is the last course required for the Computer Information Systems (CIS) degree, at the W. P. Carey School of Business at Arizona State University. In this course, students tackle a semester long software development project in conjunction with using best practices and iterative, as well as agile software development methodologies. In the Spring 2015 semester, students were required to have a total of 7 project releases (from 0.1 – 0.6 to 1.0). Each release needed to build upon the last, adding more features/functionality as the semester progressed. Let's discuss how the team made it through the semester and get a good idea of the events that took place before each release.

[Release 0.1]

Before we began working on Project CAPx, we were told by our client, Dr. Joseph Clark, that he wanted the team to use the Python Flaskr Web Development Framework for the backend of www.cis440.com , and Heroku as our hosting service. Because of this, before any actual work was done on the project, the team needed to get an understanding of how to use Flaskr to develop websites. The team tackled the daunting Flaskr documentation (http://flask.pocoo.org/docs/0.10/) and got to work. We soon found out that it was easier to get everything set up and running on an Apple device, but a couple team members only had devices that ran on the Windows Operating System. Although we knew it would be a challenge, we chose to do most of the development with a Windows machine.

After going through the Flaskr documentation, the team met with Dr. Clark to get a better understanding of what he was asking for. In our first meeting, we discussed possible layouts and made a decision on the various features that were an absolute must have for the project. After the meeting, the team took a couple days to compile a couple site mockups to show the client. We wanted to get an "Ok on a mockup before we began coding, this way, we would have a clear understanding of the HTML structure that would be needed for the website. Here are some of the mockups that were put together:

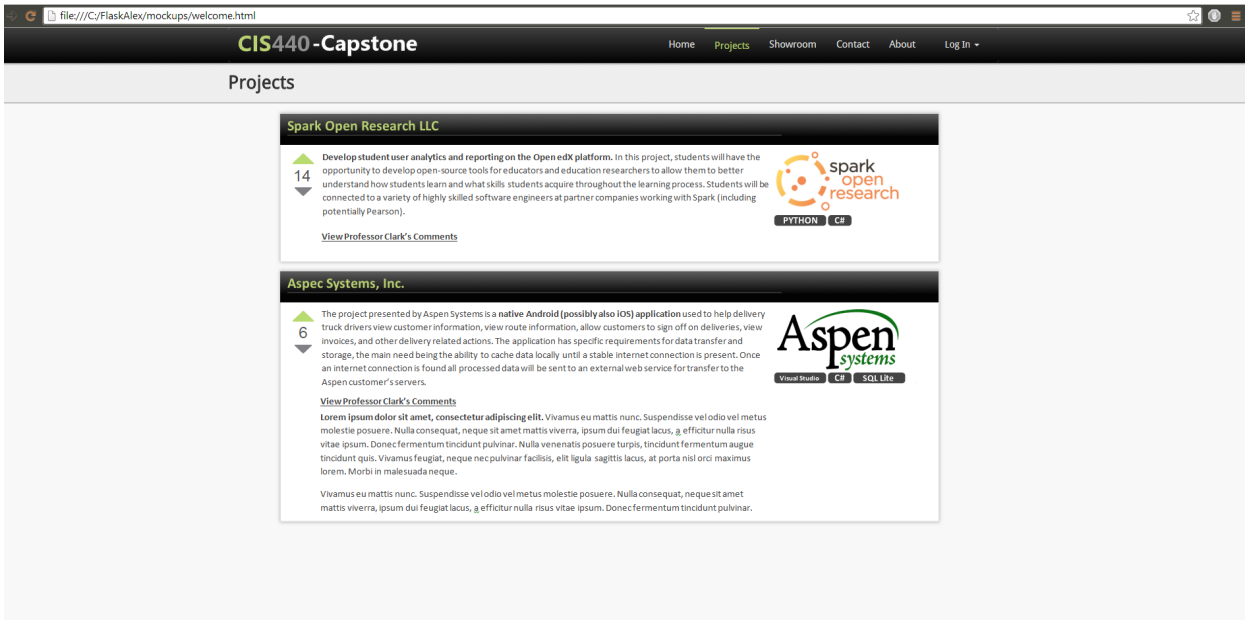


Figure 1: A "light" version mockup that was created with GIMP, a free image editor and creator, similar to Adobe Photoshop.

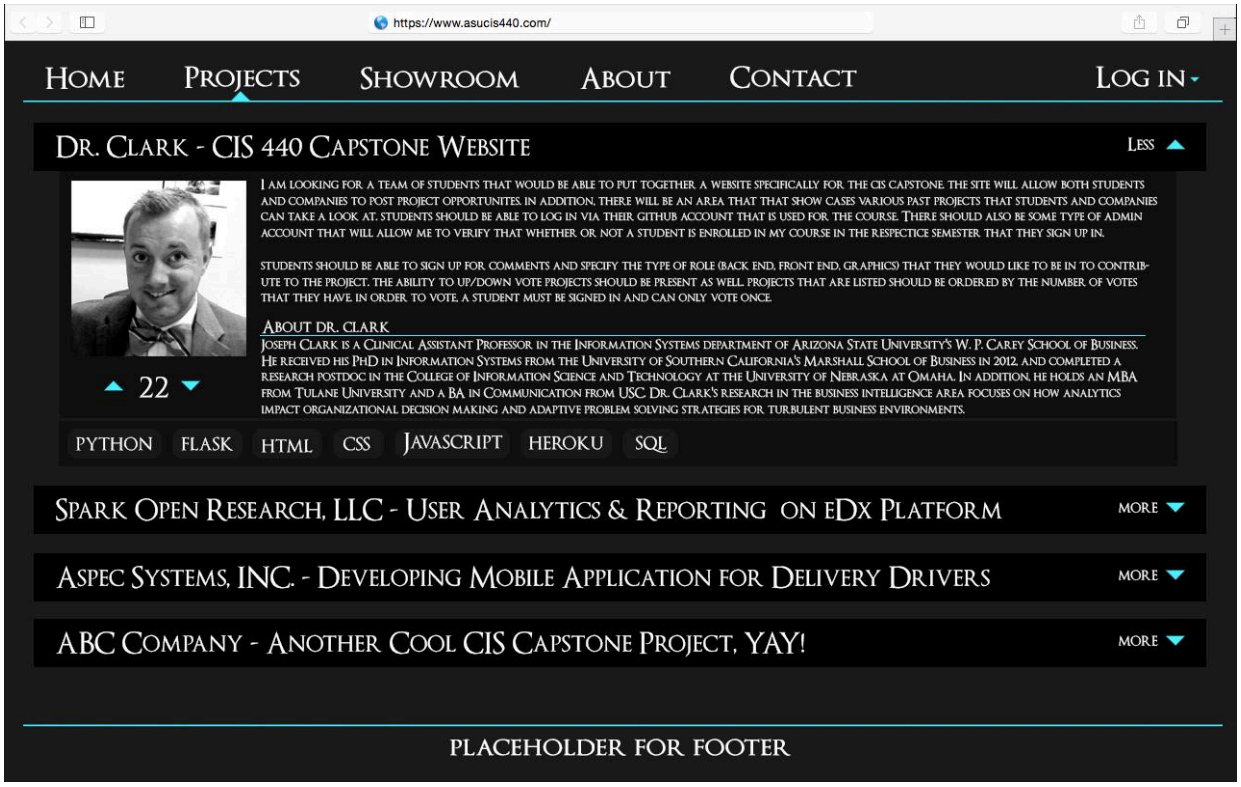


Figure 2: A "dark" version mockup, created using Adobe Photoshop.

After reviewing the mockups with our client, it was decided that the team go with the “light” version. This was the point that got the project “kicked off.”

[Release 0.2]

** Part of Release 0.2 was to create a technology stack diagram. Please see the [Technology Stack] section to see the stack **

Once we understood how to work with the Flaskr framework, we needed to get Heroku setup and running. To do this, we followed the Heroku documentation and setup the Heroku toolbelt on our machines. With the toolbelt, we were all able to push our code to Heroku using Git. Since we are using Github for collaboration purposes, it was easy to learn how to push to Herkou with Git.

Next, the team made the decision to use Twitter’s Bootstrap for the frontend. We chose this particularly because bootstrap automagically scales webpages according to the window/screen size appropriately. We also chose it because of the grid system that it uses. It made creating the project boxes on the *Projects* page a lot easier:

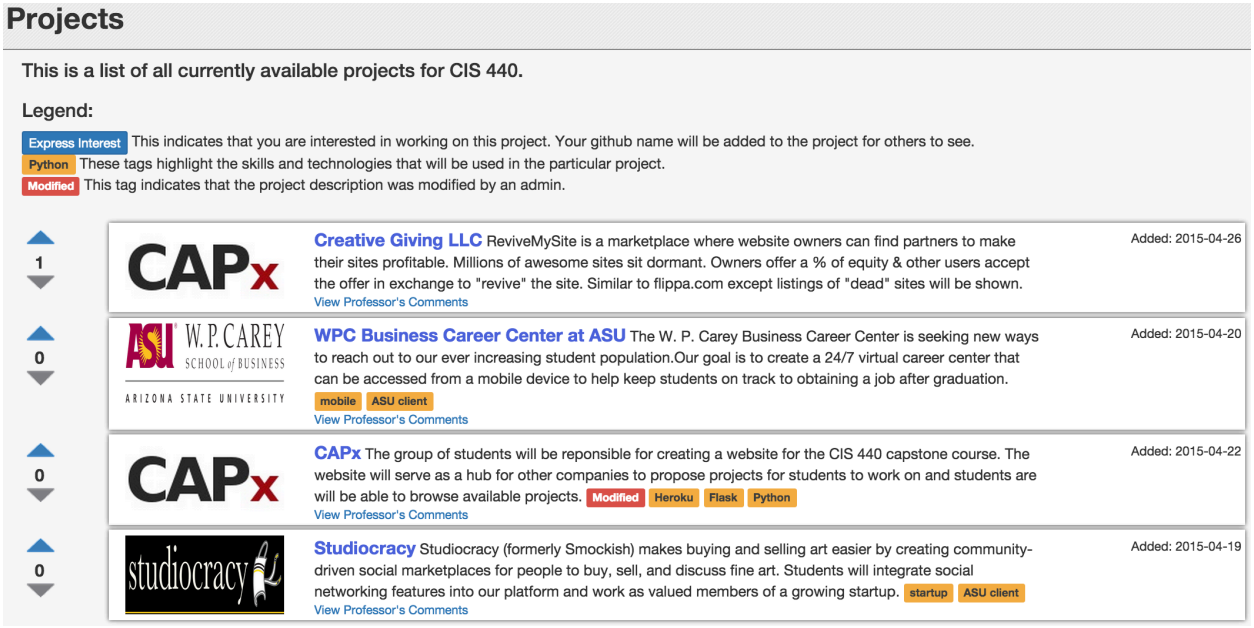


Figure 3: Each project displayed in shown inside a "box." The up/down vote area to the left of each project is also within a "box" that has no borders.

Now you might be asking yourself where all the data is stored. A PostgreSQL database was used, solely because it played very well with the Heroku service. To make things simpler, we also used PGAdmin (a database management tool that is used to manage PostgreSQL databases) to create the necessary tables in our project. All of our table create statements can be found in the CAPxSQL.sql file on our Github Repository (https://github.com/asu-cis-capstone/capx).

In addition to using Postgres, we also used Amazon's S3 service to host company images. As shown in Figure 3, a client can upload a company logo when submitting a project proposal, and the logo appears in the project description. In order to keep things "free", we needed somewhere to host our images. Amazon allows us to store up to 10,000 records for free. So, when an image is uploaded, it gets stored in our S3 database and a link to the image is passed to our Postgres database. When projects get displayed, we use the image URL stored in Postgres to fetch the image and display it next to the appropriate project.

In order to have fancy transitions/animations/hover effects, we used both JavaScript and JQuery, two very popular technologies, especially in the world of web development.

[Release 0.3]

After getting all the initial structure out of the way, we needed to do some testing. We decided to create a Google Form and have it sent out to students in our class. You can view our form at http://goo.gl/forms/TEXaZuHtb6. After compiling all the feedback, we made some adjustments to our site (outside of contentless pages – we had placeholder text for a couple pages up to this point). These were our adjustments:

- User must be logged in via Github to vote
- Made changes to the Up/Down vote
    - The arithmetic was a little off and needed to be updated
    - We also added the ability to reverse your vote on a project

[Release 0.4]

It was finally time to begin cleaning up the code and optimizing the functionality of the site. Outside of the backend coding, the team had finished up a majority of the work that needed to be down. Because of this we created a walkthrough video of www.cis440.com for this release. The video can be viewed at: https://www.youtube.com/watch?v=aK_Lo4ghw9M

[Release 0.5]

After code clean up, we needed to refine all the different site features that were implemented. For this release, we focused primarily on the site blog. Initially, when the blog was implemented, the ability to add images via HTML was not possible. We could not determine how to make this possible at first. Luckily, a member of the team figured that we could use a module called *boto* and use markdown language in our blog feature. So, this is what was implemented and took the longest to fully test for this release.

Other than the blog feature, we also added some other small features to the site. We added a total of three new features for administrators:

- The ability to add technology tags to a proposal before approving it
- The ability to edit the proposal description
- The ability to set projects "inactive" in order to remove them from being displayed on the *Projects* page

Lastly, we created an HTML template for the *Showroom* page that would allow the site administrator to highlight various projects of their choose. The template is used to create the "project container" that you see on the *Showroom* page.

[Release 0.6]

For our last release, we created a series of ten different automated tests. These tests can be found in the *tests* folder on our Github Repository. There is also a description of each test in the folder as well.

[Summary of Releases]

Here is brief, bulleted summary of each release:

[[0.1]]

- Added site mockups
- Created GitHub repo

[[0.2]]

- Added technology stack diagram
- Added Procfile for Heroku
- Added SQL file for DB table and dummy data creation
- Updated live Heroku test site link (in overview)
    - Site now has HTML, CSS, & some JS
    - Content is read from PostgreSQL DB
- Updated "How to Install" section

[[0.3]]

- Link to Google Form used for User Survey: http://goo.gl/forms/TEXaZuHtb6
- Updated the CAPx Test Site (capxtest.herokuapp.com)
    - Added content to "About & Contact" page
    - Added content to "Home" page
    - Added GitHub Login functionality
    - Fixed the Up/Down Vote issues
    - User must be logged in to vote

[[0.4]]

- Link to Release 0.4 video walkthrough: https://www.youtube.com/watch?v=aK_Lo4ghw9M

[[0.5]]

- Added blog edit feature
- Added blog delete feature
- Template for showroom added

- "Companies we have worked with" added to index
- Added "Express Interest" button
- More beta testing
- Cleaned up contact/about page
- Added admin feature: set project inactive
- Added admin feature: add tags to project
- Added admin feature: able to change description before submission

[[0.6]]

- Added a *tests* folder that contains 10 different automated testing items that were created by the team
- Continued to add logo images to the *logos* folder
- Various additions to the site
    - Admin Panel: Added a "Printable Projects" feature (allows client to create a signup form for students)
    - Logo images can now be submitted as part of the propose project form

** Release 1.0 is not included in this documentation because it only contains the completed documentation (this text) **

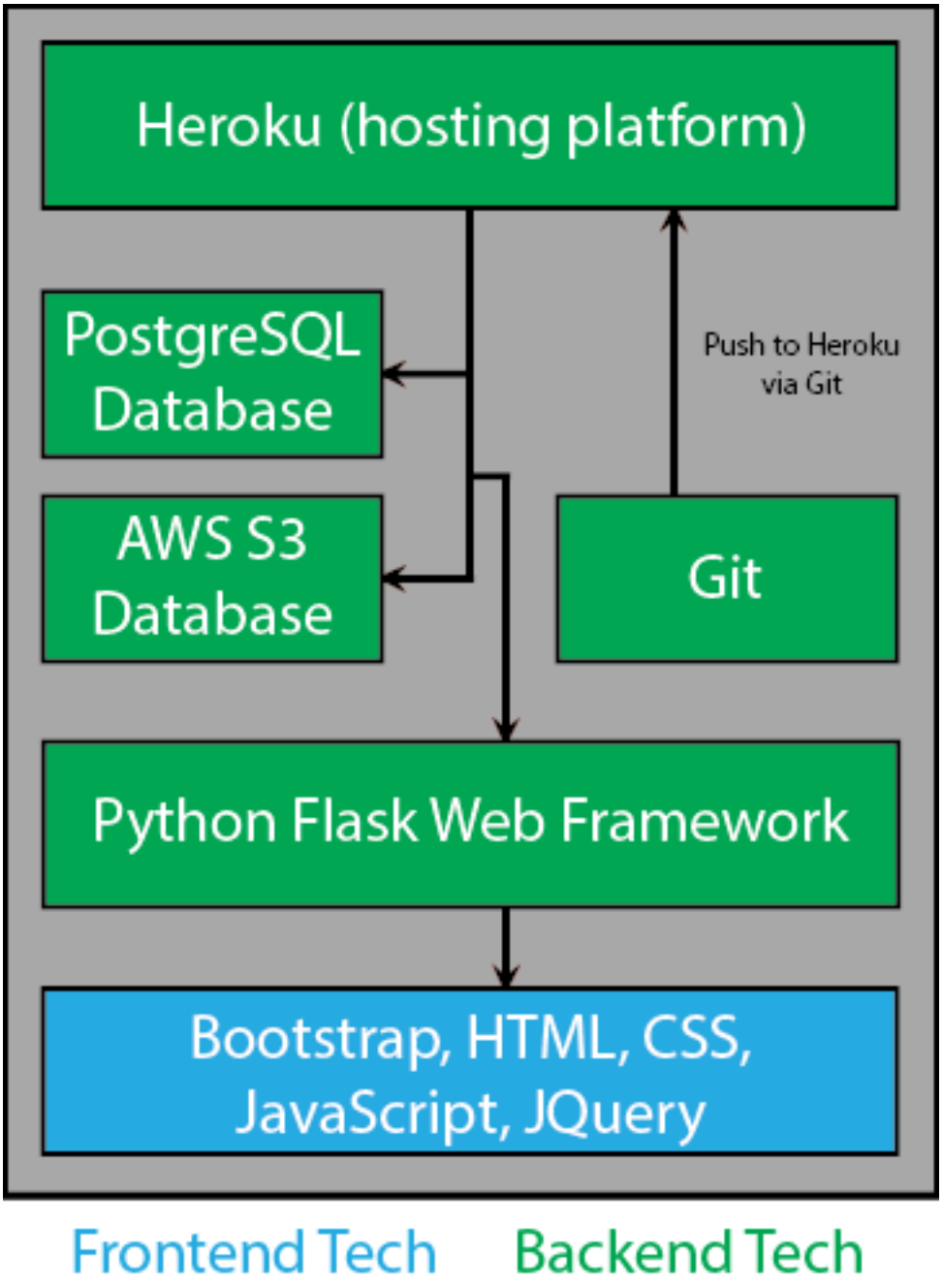
Technologies Utilized

Project CAPx required the use of many different web development technologies. Here is a comprehensive list of each technology and what it is used for:

- Python Flask Web Framework
    - This web framework allows for the unique organization of code files. The basic file structure for this framework is:
        - /projectName
        - /static
        - /templates
    - Any files that never change (such as a cascading style sheet (css)) is placed in the static folder. Any other files, such as basic layouts or navigation bars, is placed in the templates folder.
- Amazon Web Services, S3
    - S3 is used to host all the images that are uploaded by clients when they submit a project proposal
- Bootstrap, HTML, & CSS
    - The bootstrap structure is used heavily for the www.cis440.com site. Our team wanted something that would easily scale for any screen size.
    - In addition, because this is a web application, HTML and CSS were also heavily utilized
- JQuery and JavaScript
    - Both of these technologies were utilized simply for transitions, mouse over effects, and other, various, fancy feats found on the site.
- PostgreSQL
    - Postgres is what we primarily used as a database. We chose it because it is free and open source, and played well with our hosting platform, Heroku.
- Heroku
    - A cloud based web service that supports many different programming languages

- Git
    - As you may have been able to tell, Github was utilized heavily for this project.
        - All code is “open source” and available in a Github Repository
    - All code was also pushed to Heroku using Git

[Technology Stack]



Code Snippets & Project Setup

[Application Config Variables]

Several of our features require us to set specific config variables for our application as soon as someone connects to CIS440.com. In order to set app config variables in the Flask Python framework, we use the following syntax:

```
app.config['name_of_variable'] = ‘value_goes_here’
```

In order for our GitHub login functionality to work, we need to set the GitHub Client ID and GitHub Client Secret, both of which are given to us from our GitHub applications console after registering our app with GitHub.

```
app.config['GITHUB_CLIENT_ID'] = 'our_github_client_id'
app.config['GITHUB_CLIENT_SECRET'] = 'our_github_client_secret'
```

For basic encryption purposes, we also need to set a secret key for our application:

```
app.secret_key = 'secret_key_goes_here'
```

In order to establish a connection with our Amazon Web Services S3 account, we need to provide an Access Key ID and a Secret Access Key. These are given to us from the AWS console after signing up for Amazon S3.

```
app.config['AWS_ACCESS_KEY_ID'] = 'access_key_here'
app.config['AWS_SECRET_ACCESS_KEY'] = 'secret_access_key_here'
```

[Important Modules Imported into our Flask App]

As in most programming languages, we are able to import other modules, extension or libraries that give us access to pieces of code written by others. Our app imports many different extensions, but here we will highlight a few of the significant ones:

1. **Boto** (https://github.com/boto/boto) - a Python package that allows for easy connection to a variety of Amazon Web Service features, including S3 which is our storage device for uploaded company logos
2. **Flask-Markdown** (https://pythonhosted.org/Flask-Markdown/) - a Python package specifically made for the Flask framework. It allows us to render text using the Markdown syntax. This package is used to render our blog post entries on the homepage using markdown.
3. **Psycopg2** - a Python package used to make a connection to our PostgreSQL database. We start by creating a connection to the backend and then perform SQL queries using a cursor object.
4. **GitHub-Flask** (https://github.com/cenkalti/github-flask) - a Python packge made specifically for Flask. The feature we are using is the authentication / login through GitHub. When our users first log in, they are redirected to the GitHub login page and have to grant our app permission to access their public information. After a successful login, they are redirected to CIS440.com as an authenticated user.

[General File Structure Overview]

Here is our basic file structure (only files needed to run the application are shown):

```
static/
        --CSS files
        --JS files
        --Font files
templates/
        --about
        --addproject
        --admin
```

--adminBlog
--adminBlogEdit
--adminInactive
--adminpanel
--adminPrintProjects
--adminProject
--adminPromote
--adminRoster
--base_navigation
--base_navigation_loggedin
--index
--login
--logout
--pleaselogin
--project
--showroom
--welcome
Procfile
requirements
app.py

The *requirements.txt* and *Procfile* are needed for our application to deploy correctly on the Heroku hosting platform. The *requirements.txt* file contains all the modules and extensions that we are using in our application. This allows Heroku to replicate this exact environment on their end, to ensure that our application runs correctly.

In order to create the *requirements.txt* file we run the following command:

pip freeze > requirements.txt

The procfile only contains one line of code: *web: gunicorn app:app --log-file -*
This file is needed for Heroku to recognize our app and deploy it online.

The static directory contains all of our CSS, Javascript and other files related to the styling of our site. The templates directory contains all of the HTML pages. The *app.py* file is the heart of our

application. Everything runs through this file. It determines what happens when users click on certain links, what data gets pulled from Amazon, what data gets pulled from Postgres and what areas of our site the user has access to (i.e admin access or normal user access).

[Example Route from our App.py]

This section will discuss the idea of "Routes" in Flask and we will run over sample code for our Index route. Basically, when a user clicks on a certain link on our webpage, they will be redirected to another page on our site. The route determines what page will be displayed next and what data will be sent to that page.

Our Index route:

```
# Index route
@app.route('/')
def index():
    connect_db()
    # Sets the current users GitHub name
    if 'userName' in session:
        print('User already in session...')
    else:
        session['userName'] = ''
        session['adminCheck'] = ''
        # Tracks if a user is logged in or not
        session['blnLoggedIn'] = ''
        #Tracks the user's oauth code provided through GitHub login
        session['oauth'] = ''

    homeStatus = 'active1'
    if session['blnLoggedIn'] == "yes":
        logincheck = '_loggedin'
    else:
        logincheck = ''


    # pull all blog posts, show the most recent
    g.cur.execute("select * from Blog where active = 'yes' order by datecreated desc, timecreated desc;")
    blogs = g.cur.fetchall()

    close_db()

    return render_template("index.html", homeStatus=homeStatus,
    logincheck=logincheck, admin=session['adminCheck'], userName=session['userName'], blogs=blogs)
```

Lets step through the code!

```
@app.route('/')
def index():
```

The first line @app.route('/') declares a new route for our application. Usually we would have a name after the '/', but since this our our Index or “Home” route, we only need to declare it as '/'. This means that whenever someone navigates to CIS440.com, this route will be executed. Next we define a method called “index()” which contains all of the code that is executed by the home route. We can ignore the connect_db() line for now.

```
# Sets the current users GitHub name
if 'userName' in session:
    print('User already in session...')
else:
    session['userName'] = ''
    session['adminCheck'] = ''
    # Tracks if a user is logged in or not
    session['blnLoggedIn'] = ''
    #Tracks the user's oauth code provided through GitHub login
    session['oauth'] = ''
```

Here we use the Session extension that allows us to set session variables for each user that accesses our site. If we do not have a variable called userName within our session variables, it means that it is a new user and we have to track a new session. If a userName is already present in our session variables, then we know the user is already browsing through our site and a session was already established, therefore we print “User already in session…” to the console.

In order to track a new session, we declare 4 session variables: *userName*, *adminCheck*, *blnLoggedIn* and *oauth* and we initialize them all to a blank value.

```
homeStatus = 'active1'
if session['blnLoggedIn'] == "yes":
    logincheck = '_loggedin'
else:
    logincheck = ''
```

We want to track which page of our site the user is currently on so at the start of the application, we created a variable for each of our navigable pages that holds the value of “active1” if the user is on that page:

```
# These indicate which page has focus
showroomStatus = ''
aboutStatus = ''
projectStatus = ''
homeStatus = ''
addprojectStatus = ''
adminStatus = ''
```

Since we know that the user is on the Index page, we set the value of *homeStatus* to 'active1'. Next we want to check if the user is already signed in. To do so, we create an if statement that sets the logincheck variable equal to '_loggedin' if the user is already logged in or sets it to an empty string. The session variable *blnLoggedIn* will only be set to "yes" if the user has already gone through the GitHub login process on our site.

Now that the basic setup is taken care of, we need to get ready to gather all the data that we want to send to our index.html page. This will mainly involve pulling any non-static data from our database and sending it to our HTML page.

On our index page, we have a blog feature that lets admins post new updates. These are saved in our database and need to be pulled before the index page can be rendered to the user. This is a good time to discuss our connect_db() method.

```
# pull all blog posts, show the most recent
g.cur.execute("select * from Blog where active = 'yes' order by datecreated desc, timecreated desc;")
blogs = g.cur.fetchall()
```

```
connect_db()
```

Whenever a user makes a request to our database, it opens up a new connection. For our PostgreSQL database, we are limited to having 20 concurrent connections because we are using a free Heroku account. This does not mean that we can only have 20 users on our website at the same time. For resource management purposes, we need to close our database connection after we process our SQL query. This will ensure that we are much less likely to hit the connection limit. Basically we want to follow this structure when interfacing with our database on the backend:

open DB connection >> execute query >> store query results >> terminate connection

In order to make this process more efficient, we created a connect_db() and close_db() method to handle creating and terminating new connections.

```
def connect_db():
    g.conn = psycopg2.connect(
        database="MODIFY",
        user="MODIFY",
        password="MODIFY",
        host="MODIFY",
        port="MODIFY")
    g.cur = g.conn.cursor()
```

As discussed previously, we use the imported psycopg2 extension to help with creating a connection to the database. We store our connection in a global (indicated by 'g') variable called *conn*. We call the *psycopg2.connect* method and pass it our DB connection variables. Making the conn a global variable allows us to access it from other methods more easily. Once the connection is created, we call the *.cursor()* method to create a cursor object that will be used to execute SQL statements.

```
def close_db():
    g.conn.close()
    g.cur.close()
```

For our *close_db()* method, we simply call the *.close()* method for both the connection and cursor object to terminate them.

Now we can use our cursor object to execute a SQL statement. We want to pull every active Blog entry from our Blog table and order them by date and time of creation. Next, we store the query results in a variable called *blogs*.

```
# pull all blog posts, show the most recent
g.cur.execute("select * from Blog where active = 'yes' order by datecreated desc, timecreated desc;")
blogs = g.cur.fetchall()
```

```
close_db()

return render_template("index.html", homeStatus=homeStatus,
logincheck=logincheck, admin=session['adminCheck'],
userName=session['userName'], blogs=blogs)
```

Now we can call our close_db() to terminate the connection to our database. Finally, we want to specify which HTML file should be rendered to the user. In this case, we specify that we want to render "index.html" and we pass certain variables. We want to pass our *homeStatus*, *logincheck*, session *adminCheck*, session *userName* and *blogs.* Next we will take a look at what happens in the *index.html* file.

[Template Inheritance and Passing Variables]

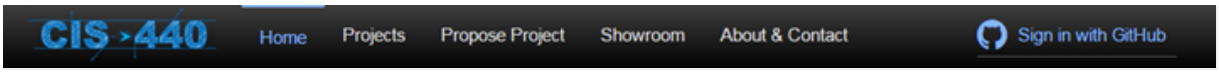
Template inheritance in Flask is very similar to other languages. We can define base templates which other pages can extend. For the base template, we want to include our meta tags, link tags, script tags, title and other header information. Within our body tag, we can define the area that will be populated by the template that extends to the our base (or parent) template.

A parent template could look like this:

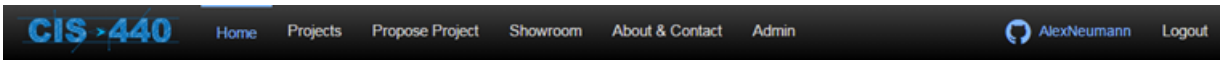
```
<!doctype html>
<html>
    <head>
        <link rel="stylesheet" href="CSS.css">
        <title>CIS 440 </title>
    </head>
<body>
    <div id="content
        {% block content %}
        {% endblock %}
    </div>
</body>
</html>
```

We can now use this as a template for other pages. The content of the new pages would be inserted between the {% block content %} and {% endblock %} tags. For our project, we wanted

to have the navigation bar on every page, therefore it makes sense to create a template for the navigation bar and then insert the content for each specific page within the block content tags. However, the navigation bar changes slightly, depending on whether or not a user is logged in via GitHub. If the user is not logged in they will see this as their navigation bar:



If the user is logged in they will see this one instead:



In order to determine which navigation bar template our index.html (or any of our other pages) should extend, we make use of the *logincheck* variable that we passed from our index route to the index.html page. *Logincheck* will hold the value "_loggedin" if the user is logged in.

The first line of our index.html page will make sure that we always extend the correct navigation bar:

```
{% extends 'base_navigation' + logincheck + '.html' %}
```

We have two navigation bar pages to choose from:

1. base_navigation.html (User is not logged in)
2. base_navigation_loggedin.html (User is logged in)

Since our *logincheck* holds a string value, we can simply concatenate the value to our "base_navigation" text and it will extend the correct navigation bar. After that we can start writing out the content that we want to display on the index page by using the {% block content %} tag.

Now we can add our blog posts to the index page. Here is the code:

```
{% for row in blogs: %}
    <div class="panel panel-default">
        <div class="panel-heading">
            <div class="row">
                <div class="col-md-6">
                    <h3 class="panel-title"><span style="font-weight: bold;">
                    {% print row[1] %}</span>
                    </h3>
                </div>
                <div class="col-md-6" style="text-align: right;">
                {% if row[6] == 'yes' %}
                    Last edit by <span style="font-weight: bold;">{% print row[3] %}</span>
                    on {% print row[4] %}
                {% else %}
                    Posted by <span style="font-weight: bold;">{% print row[3] %}</span>
                    on {% print row[4] %}
                {% endif %}
                </div>
            </div>
        </div>
        <div class="panel-body">
            {% filter markdown %}
                {% print row[2] %}
            {% endfilter %}
        </div>
    </div>
{% endfor %}
```

Since we are using the Flask framework, we can use Python syntax within our HTML pages. We start by creating a for loop that goes through each blog post and renders it to the page in a new panel using the Bootstrap styling.

```
{% for row in blogs: %}
```

The query result from our previous SQL statement is stored as tuples in the *blogs* variable that we passed to our index page. As such, we need to iterate through each tuple (row) and access the relevant information.

Each tuple is very similar to traditional array objects in that we can access its information by using the index location.

```
{% print row[1] %}
```

This statement will print the value of the second column from the current tuple (since indexing starts at 0). In our case, this prints the blog title.

```
{% filter markdown %}
    {% print row[2] %}
{% endfilter %}
```

As mentioned earlier, we want to use the Flask-Markdown extension to render the body of each blog post with markdown syntax. To achieve this, we include the *filter markdown* tag and then post the blog body inside of the filter by accessing the third column of the current row.

The Team

ALEXANDER NEUMANN

alex.neumann92@gmail.com, 11604 E. Del Timbre Drive, 480-252-8912

PROFESSIONAL PROFILE

Technical proficiency in C# programming with Visual Studio, Python, Web Development with Python, API scripting, database design with PostgreSQL and SQL Server, data mining and analysis with SPSS Modeler, text mining with RapidMiner, Microsoft Office application suite; team oriented with great leadership qualities; highly motivated and ambitious; fluency in German

EDUCATION

Bachelor of Science in Computer Information Systems **Fall 2015**
Bachelor of Science in Business Data Analytics **GPA: 4.0**
W. P. Carey School of Business, Arizona State University, Tempe, Arizona
Barrett, The Honors College
Attended school for 14 years in Mannheim, Germany

AWARDS/HONORS

Recipient of Regent High Honors Endorsement (AIMS)
Recipient of the Richard Malone CIS Scholarship
Eligible for ASU President's Scholarship
Member of Dean's List
Top 3rd percentile of graduating class

COMPUTER INFORMATION SYSTEMS & DATA ANALYTICS EXPERIENCE

Web App Development using Python

- Created a website for the capstone course (CIS440) for the Computer Information Systems bachelor degree at W. P. Carey
- Programmed a fully functional, live website that serves as a hub for outside companies to find a group of students to collaborate on projects with them (Live website at: CIS440.com)
- Technologies used: Flask Python Framework, PostgreSQL, Heroku, Amazon Web Services, Javascript, Twitter Bootstrap framework and HTML

Business Data Mining Honors Project – Text Mining in Healthcare

- Created a Naïve Bayes classification model based on question/answer data for healthcare related questions
- Used RapidMiner to conduct text-mining in order to analyze if medical answers found online are more similar to ones written by doctors or to those written by individuals without healthcare expertise

WORK EXPERIENCE

W. P. Carey School of Business, *Teaching Assistant* **January 2015 - present**

- Responsible for setting up the course resources used throughout the semester, including creating all assignments and helping students with the NetSuite online accounting software
- In charge of grading all submissions for every class section taught by the professor as well as responding to student inquiries

SKILLS

Native fluency in German and English, as well as proficient in Spanish
Knowledge in webpage design using Python, PHP, Javascript, HTML, CSS and various frameworks; gathering data via API calls with Python; data analysis and model creation using SPSS Modeler and RapidMiner; cloud hosting services such as Heroku as well as S3 storage on Amazon Web Services; C# programming with Visual Studio

14279 W. Riviera Dr.
Surprise, AZ 85379

Steve Truong
StevenT619@gmail.com

(619)-254-7466
LinkedIn.com/in/SteveTruong

EDUCATION

Bachelor of Science – Computer Information Systems **May 2015**
W. P. Carey School of Business, Arizona State University, Tempe, AZ **3.68 GPA**

Awards/Honors/Groups:

- Dean's List – Fall 2010, Spring and Fall 2012, Spring 2013, Spring and Fall 2014
- W. P. Carey Leader's Academy

Language/Technical Skills

- Fluent in English and Vietnamese
- Microsoft Excel, Word, PowerPoint, Visio, Visual Studios, SQL Server
- Apple OS X, Pages, Keynote, Numbers
- Proficient Coding: C#, SQL, HTML, CSS
- Basic Coding: JavaScript, PHP, MySQL, Visual Basic for Applications (VBA), Python, Bash
- Certificates: Google Power Searcher

PROFESSTIONAL EXPERIENCE

IT Support/Help Desk **November 2014 – Present**
Clearwing Productions, LLC.

- Provide technical support for the Clearwing Productions Arizona office for both Apple OS X and Windows OS users
- Construct "How to" guides for various recurring technical issues, such as, installation of printer drivers, setting up HTML signatures for variety of different mail applications, and mobile device email and calendar set up, in order to maximize utility and cut down time spent on frequent issues
- Create various Excel macros and add-ins using Visual Basic to increase efficiency for the accounting department by reducing the amount of time spent on compiling numerous reports, like individual, end-of-month, employee expense sheets used for reconciliation purposes

Enterprise Risk Services – Technology Risk, Intern **June 2014 – August 2014**
Deloitte & Touche LLP.

- Gathered documentation from clients and reviewed information that was provided to ensure it was complete so IT SOX testing could begin
- Began writing documentation to be served as a guide for new hires to use as a resource when learning how to review evidence provided by client for completeness, accuracy, time-sensitivity
- Provided support for the internal audit team on various tasks, such as creating testing templates for the different internal controls that are tested on client site
- Tested user access controls to ensure that users who requested access to internal applications were provisioned and de-provisioned appropriately
- Gained an understanding of the internal control testing process by learning how to apply the methodologies and best practices set forth by Deloitte

CAMPUS INVOLVEMENT/CLUBS/ORGANIZATIONS

WPC 101: Student Success in Business Facilitator **August 2013 – December 2014**

- Lecture once a week for 50 minutes on topics related to skill building, goal setting, and important steps to take in order to be successful academically, personally, and professionally by engaging students with meaningful and relevant statistical data alongside personal experiences of failures and successes

W. P. Carey School of Business Undergraduate Blog **August 2012 – August 2014**
Link to posts - http://blogs.wpcarey.asu.edu/undergrad/author/struong1/

- Write engaging and thoughtful posts reflecting upon personal, professional, and academic experiences to give prospective and current students a detailed picture of what being a student at one of the top ranked business schools in the nation is like

THOMAS RULLESTAD

515.419.9416 | Thomas.Rullestad@asu.edu | LinkedIn

EDUCATION

Arizona State University | Tempe, AZ | 2011 – 2015 (Expected)
W. P. Carey School of Business | B.S. Computer Information Systems | GPA: 3.51

PROFESSIONAL EXPERIENCE

Charles Schwab & Co., Inc. | Phoenix, AZ | June 2014 – August 2014
Intern – Security Technology & Architecture

- Developed software security requirements for agile development teams
- Utilized the Archer eGRC system to coordinate company security standards with requirements
- Built tool to assist with project – categorizing standards and linking to Archer system
- Worked in teams on business cases for client services and support
- Endorsed by department as a business analyst and technical project manager
- Learned investment strategy and worked in teams on stock market simulation

W. P. Carey Executive Education | Tempe, AZ | September 2013 – May 2014
Intern – Accounting and Business Programs

- Built corporate and local business relationships through program delivery
- Used Advantage financial system and Access to assist with accounting
- Exposure to Salesforce.com and customer relationship management projects
- Created services blueprinting diagrams utilizing Visio
- Assisted in payment and event planning using PayPal and RegOnline
- Provided research and market analysis for marketing campaigns

Willow Creek Golf Course | Des Moines, IA | 2009 – 2011
Pro Shop Assistant

- Assisted customers with transactions and merchandising
- Performed club operational duties and organized golf course events

TECHNICAL SKILLS

- Object-oriented programming in C# and system architecture utilizing UML
- Writing SQL queries for use on MySQL server
- Accounting information systems and internal controls using NetSuite
- Using the Archer eGRC system to write security requirements
- HTML & CSS, JavaScript, PHP, and MySQL for e-commerce and web development
- Network programming and analysis using Ruby

LEADERSHIP AND ORGANIZATIONS

- Department of Information Systems Club
- W. P. Carey School of Business Leaders Academy
- Sports Business Association
- ASU Club Tennis Team (2012-2013)
- Campus ministries for ASU Young Life and CRU
- Young Life – spent the month of June 2013 volunteering at Young Life camp in northern California with fellow college students from around the country

Jose Recendez

2324 N. 29 PL Phoenix, AZ 85008 (602) 930-0391 jrecende@asu.edu

Education:

Bachelor of Science in Computer Information Systems May 2015
W. P. Carey School of Business
Arizona State University, Tempe, AZ

Associates in Network Administration: Cisco December 2010
Gateway Community College

Skills:

Technical - C# VBA Advanced Excel SQL Project Management

Language – Fluent in Spanish & English

Professional Experience:

Early Warning (security administration Intern) March 2014- July 2014

- Created metrics and analysis used by management to evaluate company processes.
- Assisted the Compliance team in ongoing maintenance of information security policies and procedures.
- Aided the Security Administration staff in reporting compliance to security policies and frameworks.
- Collaborated with the Security Administration team in preparation for external audits from Early Warning's clients as well as Payment Card Industry (PCI) audits.
- Automated Excel reports using Visual Basic for Applications (VBA) facilitating the completion of reports.

Sandhills Publishing Internship (IT, Hardware) May 2013 – August 2013

- Developed and edited processes that facilitated and expedited the completion of tasks.
- Created and removed Email accounts for employees and customers, as well as modified existing accounts to meet customer's needs.
- Maintained internal servers by updating client information.
- Worked with Sandhills representatives to solve a variety of customer issues.

Lab Aide, Herberger Institute ASU September 2014 – Present

- Monitor and aide students and staff with computer or printing issues.

Night Crew Grocery Clerk, Fry's Food & Drug Store, Phoenix, AZ June 2006 - May 2013

- Troubleshot departments inventory issues, ordering and receiving supplies proactively to maintain appropriate inventory levels crucial for effective customer service.
- Problem solved and negotiated a diversity of customer service issues.
- Stocked a 2 thousand piece load in a 8 hour shift and assisted customers in a fast pace environment.
- Awarded employee of the month (3 times) and stocker of the period, due to hard work and time management.
- Built displays eliminating the need to hire outside contractors.

Professional Affiliations:

Department of Information Systems Club September 2012-Present

- Network with employers learning about new organizational trends.

Community Service

- Feed My Starving Children- hand-pack meals designed especially for starving children.
- Participated in the refurbishing of the library at Longview Elementary School.
- Painted the cafeterias interior at Maya High School.

SNEHA DADHANIA

Work Authorization Status: U.S. Citizen

2536 W. Alta Vista Rd. Phoenix, AZ 85041 480-442-8275 sneharachhadia@gmail.com

EDUCATION

W. P. Carey School of Business, Arizona State University, Tempe, AZ **December 2015**

Bachelor of Science Computer Information Systems

Bachelor of Science Supply Chain Management

SKILLS

Web Page Design, HTML, CSS, JavaScript, C#, C++,PHP, Database Design, MySQL, Tableau, Visual Basic, Microsoft Access, Word, Excel, Outlook, PowerPoint

PROJECTS

Visual Studio Project – Flight Management System

- Developed Flight management system using Visual Studio which allows the user to manage flight information, searchable passenger manifests and to submit passenger lists for security screening.

Visual Studio Project - Banking Application

- Created Banking application using Visual Studio which allows the user to get balance, deposit to account, withdraw from account, modify user information and display current transactions.

PROFESSIONAL EXPERIENCE

Material Specialist, Avnet, Phoenix Arizona **December 2014 – Present**

- Freight Cost Saving Project- Collaborating with Inbound and Outbound team to understand the freight process.
- Establishing freight cost saving measures for Embedded Materials Department by analyzing customer contacts vs. actual invoicing.
- Determining profitability measures for customer contract negotiation based on various freight charge options.
- Increasing department profitability through logistics optimization.

Supply Chain Intern, Avnet, Phoenix Arizona **May 2014- August 2014**

- Developed & implemented an online tool for estimating and quoting freight charges for Sales Team & Customers
- Studied the supply chain process from Vendor to Avnet and Avnet to Customer
- Collaborated & mapped the process with Transportation Team and Finance Department on how freight charges are applied to customers/internal sales divisions and its impact upon Vendor P&L

Website Manager, Chaos Safety & Industrial Supplies, Phoenix Arizona **July 2013 – May 2014**

- Created, developed and managed content for organization's website
- Handled online marketing projects ranging from SEO to social media marketing
- Maintained a consistent look and feel throughout all web properties
- Assured web-based information is archived for future needs and reference

Marketing Associate, Arista Technologies Private Limited, India **Aug 2008 -Sep 2009**

- Analyzed search engine optimization results to improve page content, ensure site structure was effective, maintained keyword relevancy and promote product names and tags.
- Coordinated email marketing that involved list maintenance, segmentation, execution, monitoring, reporting.
- Arranged promotional events to market workshops and to gain new customers.
- Helped to drive online traffic with web-related campaigns by modifying online content.

PROFESSIONAL DEVELOPMENT

- Vice President of Information Technology at Department Information System Club **Aug 2014 – May 2015**
- Member at Supply Chain Management Club **Jan 2014 – Present**

ACHIEVEMENTS

- Nominated for the Intern of the Year Award **Summer Internship at Avnet**