CSM ACM: Mozzarella
Field Session 2018

Nick Klonne, Trent Murchison, Levi Petty, Brandon Verkamp
{nklonne,murchison,lpetty,verkamp}@mines.edu

June 19, 2018

Contents

1 Introduction

The Association for Computing Machinery (ACM) club at Colorado School of Mines has a focus of working on Open Source Projects in order to give the members experience working on projects in the Computer Science field that are not required by classes on campus. Mozzarella is an Open Source Project created by ACM with the goal of providing campus computing clubs a simple way to configure a club website and collaborate with other computing clubs. While the website can be customized by hard coding the needed customizations, our client desired project assets to be configurable simply by including the desired files in a configurable directory.

The goals for this Field Session were placed into three categories by the client: absolutely need to be done, should be done, and would be nice to have done. Features that absolutely needed to be done over the six weeks were the custom configurations for different club uses. This would allow any computing club on any campus to use the Mozzarella source code along with custom configuration files to build their own club website. The goals included for the should be done aspect of this project included creating a functioning Wiki that allows users to document projects, presentations, meetings, and any other activity the group wishes. The third goal aspect of what would be nice was to implement plugins for authentication services used by the given university as well as a plugin API that would allow clubs to implement and utilize pluggable site functionality. While this goal was given to the team, it was not necessary that we implement this feature.

2 Requirements

The Mozzarella projects main goal was to extend a web framework by giving it the ability to configure club branding per server. This means that any club at any university would have the ability to make a website their own for the clubs use. Another goal for the project was to make a Wiki that allows for user-editable pages. If time permitted, one other lower priority goal was to integrate a pluggable authentication system for club members using Shibboleth and/or Google OAuth.

2.1 Functional Specifications

- The framework should allow owners of clubs to easily set up and run any computing club website and configure their own club branding and pages.
  - Logos and page styling should be done through a static configuration file.
  - Users should be able to override any default files by placing a file matching the expected file path of the given asset in the custom assets directory.
- The framework should have a functional wiki for documentation, site pages, and archives to be stored.
  - The wiki should be easily editable and keep a revision history for all files.

2.2 Non-Functional Specifications

- Python should be used for program logic, since this is the language mandated by TurboGears.
- Specific libraries (WebOB, SQLAlchemy, Kajii) are used for interfacing with lower-level technologies (WSGI, PostgreSQL, HTML).
- Application should run on a Linux server.
- The new application should extend the existing Mozzarella codebase.
- The wiki should store files in reStructuredtext format and parse the files to HTML for display.

2.3 Potential Risks

There were four major risks involved with working on Mozzarella. The first was that all testing was done on a personal system, which may not have performed exactly as a production server would. This could have possibly resulted in bugs not caught until the website was put into production.

The next risk was that the Field Session team had very limited knowledge of web technologies going into this project. The team had experience in Python, but other technologies were very new, meaning that a large portion of working on this project included reading documentation in order to get the application working properly.

Third, the TurboGears framework is rather complex, integrating many other libraries, so the team had quite a bit of trouble learning about the whole system and how each library works and interacts with the others.

The last potential risk with this project was that Mozzarella had been a work in progress for approximately a year. Extensive work had already done to the code base, meaning that time had to be spent learning how the code had been modified and how it worked as a whole. Jumping in on an unfamiliar codebase took time and adjusting, meaning that the Field Session team had to work on the clients goals and learn the code base in a very short amount of time. This led to and incomplete understanding of how the system worked and a shortage of time to complete the given goals.

2.4 Definition of Done

According to the agreement between the Field Session team and the client, the project would be considered done when we had met the goal of club assets configuration and implemented a Wiki for the site. This meant that the web application had to be fully customizable to any club branding without needing to manually edit XML templates or replace existing assets used by Mozzarella. Once the custom configuration was implemented, the team needed to submit a pull request on GitHub - requesting that the teams changes be integrated into the project - and make further changes according to a client code review. Once that pull request was accepted, we moved on to implement the Wiki. Here, the team focused on providing a page view with reStructuredText formatting, revision history, and page list for the web framework and details on projects. Upon the implementation of the Wiki, the Field Session team then was able to create a second pull request, allowing the client to conduct a code review that marked the completion of the mandatory work.

3 System

3.1 System Hierarchy

Figure 1 shows the system hierarchy of Mozzarella. It begins at the base with web technologies used in the project. This is where the Field Session team started - making a practice website with HTML, CSS, and Bootstrap as a team to ensure that we all knew and understood the basics of how the web framework functions. On the System Level and Tools layer, the application uses Python, WSGI, Linux, PostgreSQL and git. It was mandatory that all members of the team use Linux, as that is where the code base was written and will continue to be developed, and is also the target deployment platform. The team used the Python language to implement new features during the development process and also used git in order to track revision history and implement a Wiki. Libraries used in the code base include WebOb, SQLAlchemy, Kajiki, and Requests - all of which are required and used exclusively by TurboGears, the web framework used by Mozzarella. This framework is built using Python and uses the components previously mentioned in order to simplify the deployment and maintenance of a Python web framework. The middleware of TurboGears allows any developer to quickly test changes to the website without much hassle.

3.2 System Architecture

The system architecture for the entire web application can be seen in Figure 2.

At the core of Mozzarella is TurboGears, which integrates many Python web libraries such as SQLAlchemy for PostgreSQL requests, WebOb for WSGI requests, and Kajiki for XML templating. Essentially, TurboGears ties these libraries together to produce a coherent website. It uses SQLAlchemy to allow programmers to intelligently query for objects in a PostgreSQL database without needing to write complex SQL commands, WebOb in order to resolve a given website URL to a given class and method within the Python application code, and Kajiki for filling developer-made XML templates with instance variables so that the visual framework for a page can be programmed once and automatically filled with different content. Configuration of custom site assets (such as custom CSS styling or image files) was implemented using the statics middleware built into TurboGears, so no external dependencies were required for this functionality. The Wiki functionality used libraries built into TurboGears to fulfill requests for pages and to fill XML templates with page content, docutils to format reStructuredText documents as HTML, and a git repository (manipulated through the pygit2 library) to store pages and keep a revision history for all pages. The system is capable of reading in and using deployment-specific options from configuration files: `development.ini` on a development deployment and `production.ini` for production. This is the primary method by which campus clubs will configure their websites.

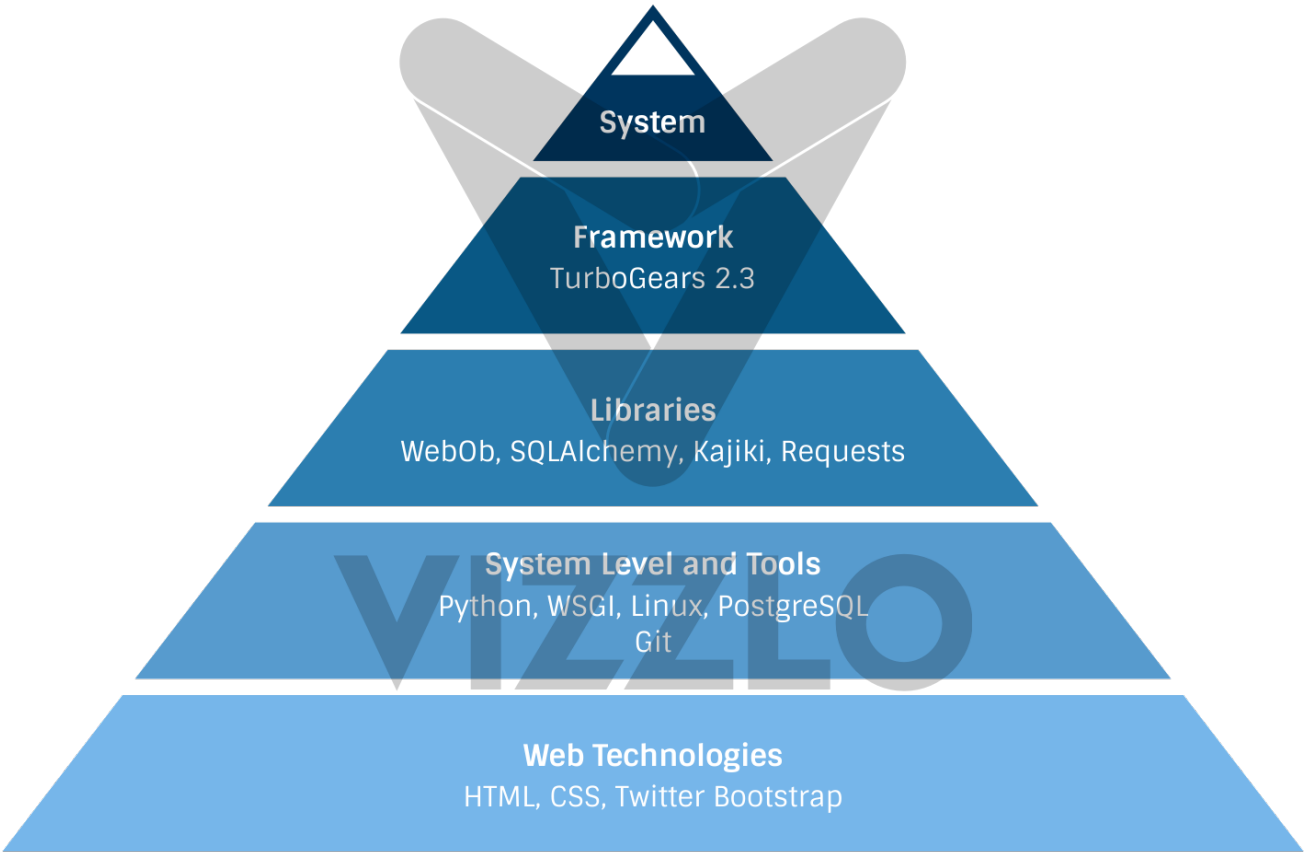

Figure 1: Mozzarella System Hierarchy

4 Technical Design

The ability to configure the website to fit any club on any campus was the biggest and most important part of this project. In order to do this, the team made adjustments in the master.xhtml template file, replacing static text with keys that allowed clubs to replace images, banners, and site css simply by including certain files in a user-specified custom assets directory. Figure 3 shows how this process works - note that `tg.url()` is simply a method that resolves a relative path to a URL in the context of the deployed website. Essentially, when the application is trying to load a given file, it will first look in the static assets directory configured as `site.custom_assets`. If the requested file is found in this directory, that file is returned and served by the application. If not, or if the directory is not configured, the application will then search in the sites public assets dir, which is at `[website directory]/public`. Additionally, clubs may set a configuration option in development.ini or production.ini (depending on the particular deployment) in order to change the club name that is used by every page of the website. All template files needed to be changed in order to display the configured website name in the page titles and header text.

The decision to use a git repo to store and keep track of revisions to Wiki files necessitated extra logic to handle certain cases. The Wiki is optional - its location is set by an option in the deployment configuration file. If this option is not set, the Wiki will not load, returning an HTTP error code to let the user know that the Wiki is not enabled. If the Wiki is enabled and its repository has not been initialized, Mozzarella is able to initialize the repository at the configured location and add to it a standard file outlining how to use and contribute to the Wiki. Figure 4 shows the flow of the Wiki upon implementation.

5 Design Decisions

Most of the potential design decisions for Mozzarella were strictly dictated by the client. Decisions being mandated by the client are a result of the code base already being a working project. Since the Mozzarella codebase has been active for over a year, there was generally no reason to change the framework or the languages being used. While this allowed the team to meet the clients exact expectations, it meant that there was not much creative freedom.

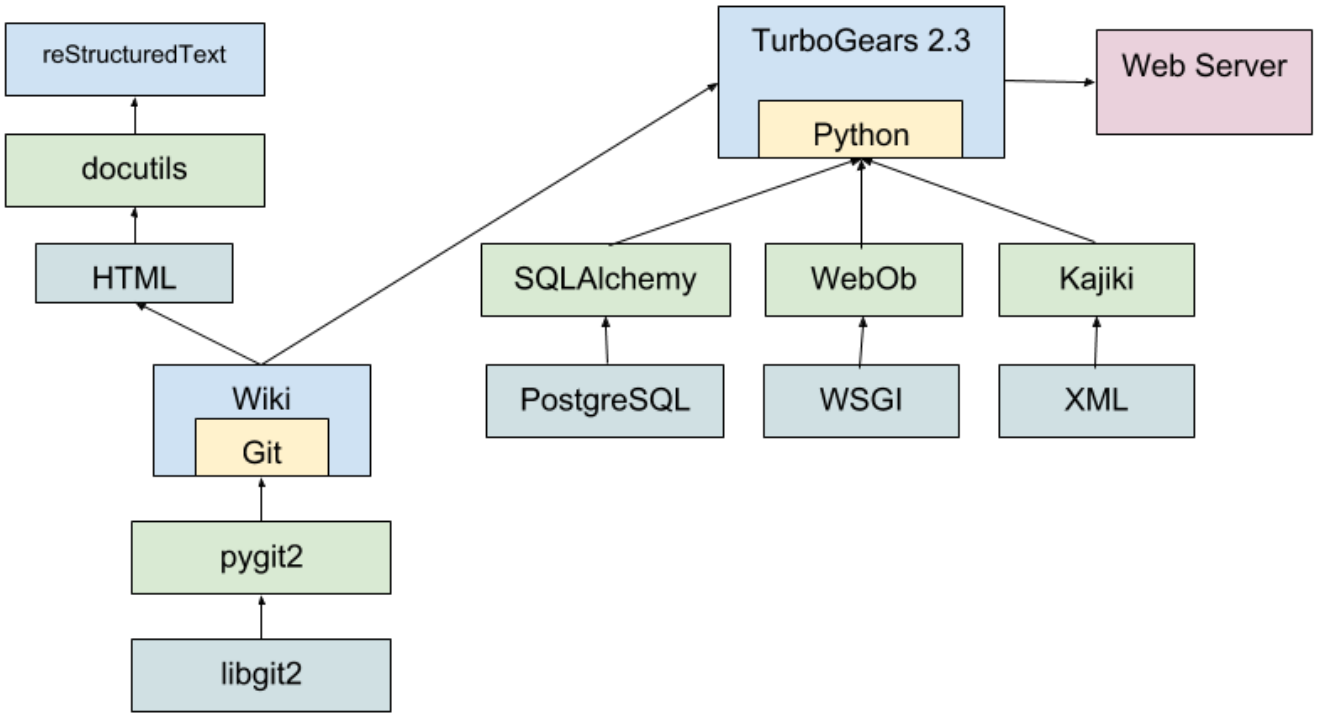

Figure 2: Mozzarella System Architecture

One of the requirements for the Mozzarella website was a method of configuring custom static assets. The team saw two possible ways of accomplishing this: either use the statics middleware built into TurboGears, or program our own custom solution into the application controllers. We decided to use the prebuilt middleware, which saved us the extra programming, reduced complexity, and increased readability in the controller code, and kept the application code in alignment with the rest of the TurboGears philosophy.

One place where we differed from the TurboGears recommendations was in the implementation of the Wiki feature. Instead of using a standard SQL database model, we used a local git repository manipulated through pygit2 for page storage and version control. Using git allowed the team to easily implement a Wiki that used a version control that most computer groups will already know and understand, while eliminating the need to write database migrations or a model for use in application code.

6 Results

In accordance with the clients Definition of Done, the Field Session team completed the Mozzarella goals for the summer. The goal for the Mozzarella web framework was to allow for the already existing code base to be configured for custom use. As shown in Figure 5, the Mozzarella web framework comes ready out of the box but is configured to the Mines ACM theme. Initially the team had some problems implementing the custom configuration due to an incomplete understanding of the code base. However, after a meeting with the client this was resolved and the feature was completed. . As a result, now any club can easily put their logo, name, and custom styling directly onto the site. Figure 6 shows Mozzarella after customization, including site-specific CSS that has been added and a site logo that has been changed. The page information and various other sections can be modified to the clubs liking also.

The second goal for Field Session was implementing a Wiki to track projects and other club information. This feature was implemented, but with similar trouble due to a lack of understanding of the code base.. Additionally, we were asked to implement the Wiki using a git repository as opposed to the TurboGears recommendation of using a SQL database model. After struggling with the beginning, the team finally implemented a git based Wiki that has a generic start up for when the website is first initiated. From there, the club has the ability to change the layout and pages of the Wiki to their own liking. While the team did not have time to implement page editing or creation via a web interface, pages can still be created and edited using git - club members simply need to clone the Wiki

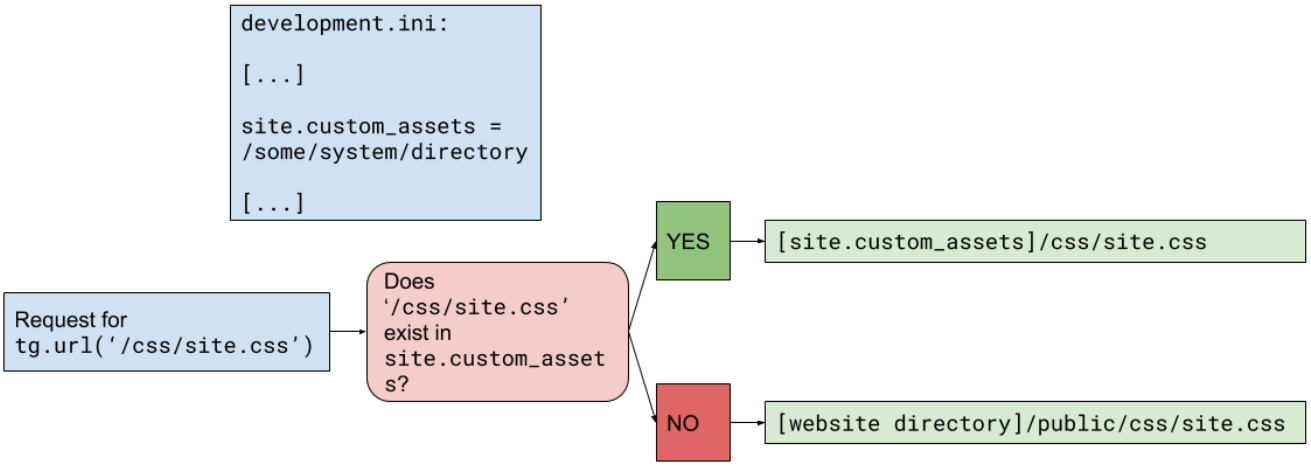

Figure 3: How server assets are loaded

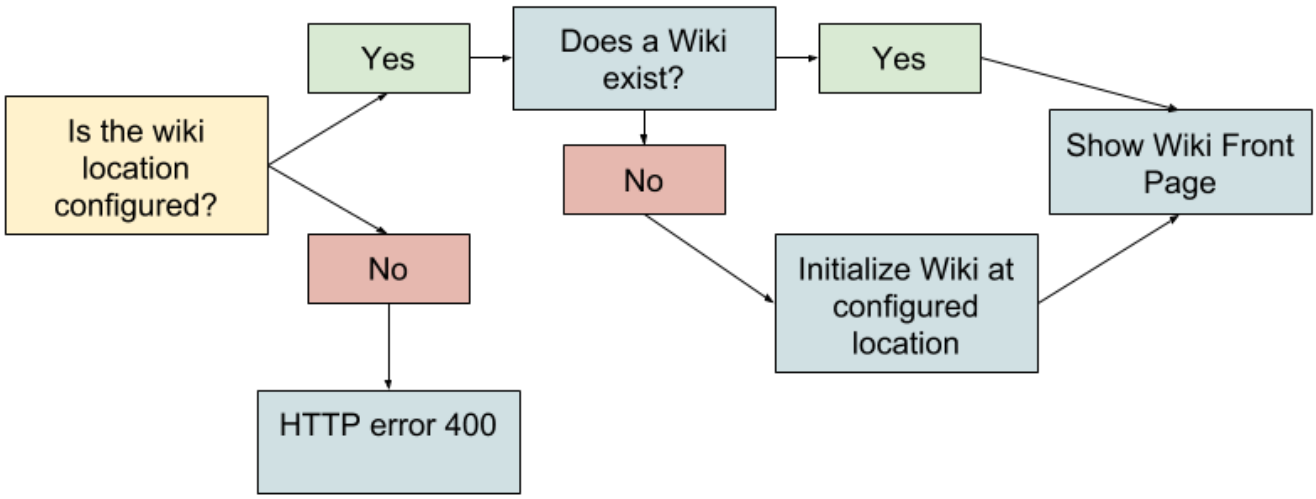

Figure 4: Wiki loading flowchart

repository hosted on the web server and commit and push their changes from there. The Field Session team managed to implement the two major features wanted by our client meaning that according to the Definition of Done, the Mozzarella Field Session Project is officially completed.

7 Reflections

Looking back on the project, the Field Session team recognizes that given more time and a more thorough introduction to the code base, the plugin goal could have been met. Unfortunately, time was not on the teams side when it came to learning the basics of web development, the code base, and the existing system hierarchy. Before even looking at the existing Mozzarella code, the team became accustomed to HTML and CSS by creating an example website to demonstrate basic understanding to the client. While the practice website allowed the team to reach the same basic knowledge for the project, it ultimately took time away from learning the code base and eventually working on features. The implementation of the custom configuration allowed the team to learn more about the Linux operating system, WSGI request dispatch, and TurboGears. Once the configuration was finished, the Wiki implementation became the main issue to tackle. The team spent a lot of time here reading documentation on TurboGears, git, and pygit2 to more understand how the client wanted the Wiki to be implemented. Much of the time spent implementing this was asking questions and conducting more research, as using git to implement a wiki is not a particularly common

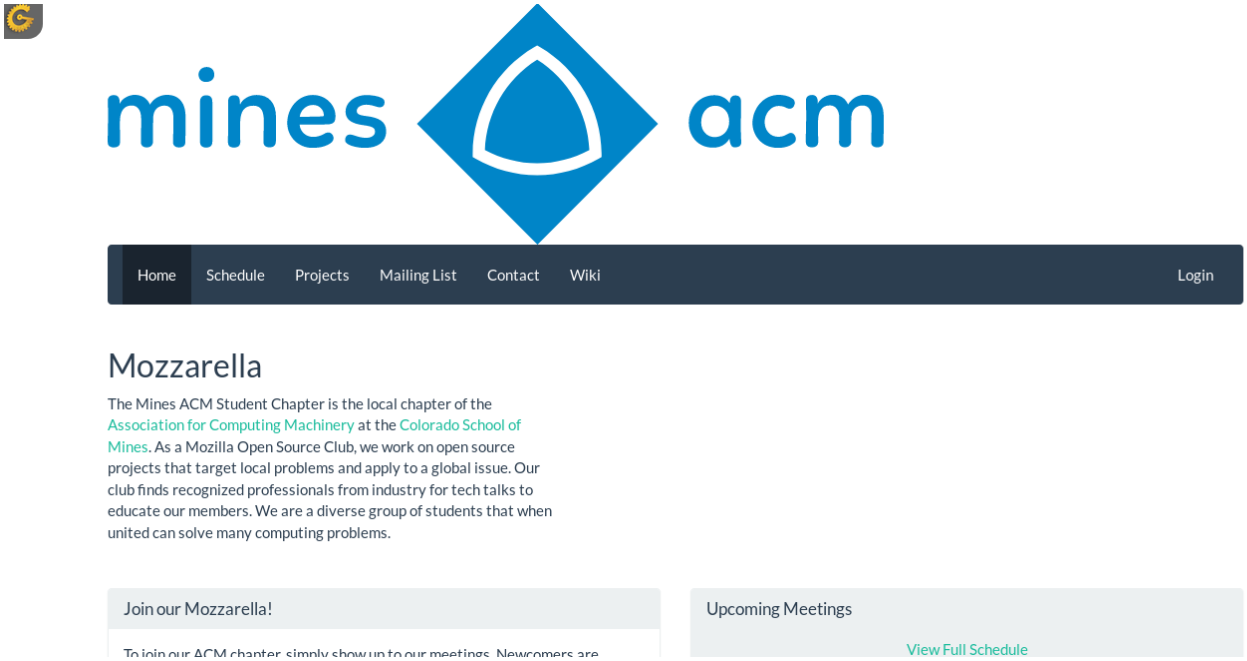

Figure 5: Mozzarella Web Framework before customization of assets

solution. The Field Session team did not have time to attempt to implement an authentication plugin, but the main goals of the project - custom asset configuration and the Wiki - were achieved.

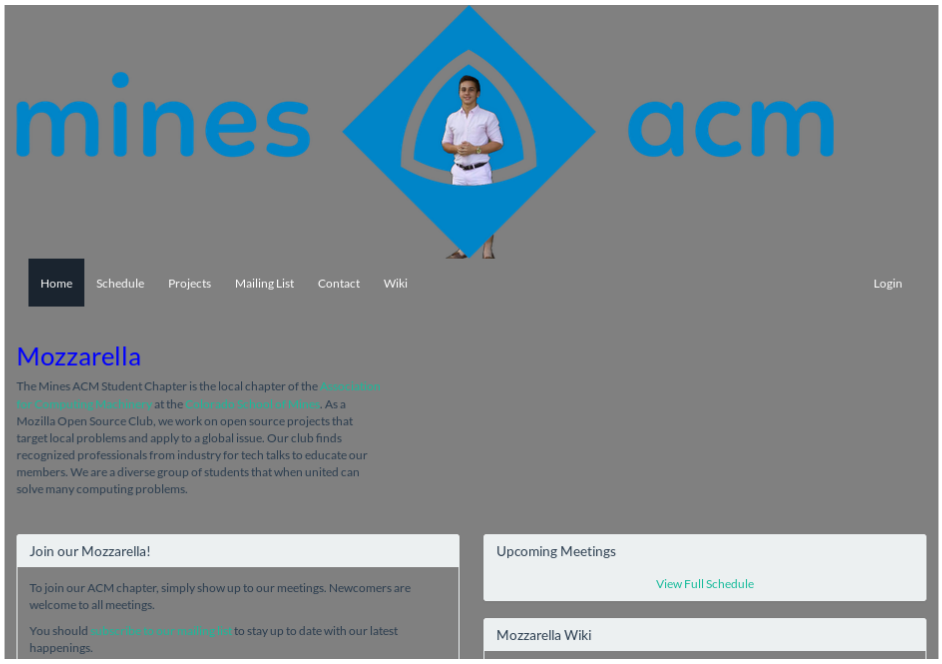

Figure 6: Mozzarella Web Framework after customization of assets