Abstract

PauseLab is a system to manage community engagement in proposing and selecting public art projects. The system serves three primary purposes: collection and exposition of ideas, compilation of proposals to and from the community, and facilitation of community input on which ideas will receive funding. Residents of the Charlottesville community will now be able to streamline their voices.

Introduction

The objective of our Computer Science Service Learning Practicum project is to design a system for Charlottesville-based non-profit, PauseLab, that would facilitate projects in Charlottesville-area communities. PauseLab is an organization that promotes civic-engagement in the Charlottesville area by sourcing ideas for projects from the residents themselves. These projects range from setting up traffic lights to building public art pieces.

* Participatory budgeting
  + How it works
  + Existing systems that have used it before
* The system in place before
  + Paper forms
  + Wordpress site
  + Excel data organization
* Desired features of the system that was built
  + Idea collection, proposal submission, voting
* Impact that this system will have on the community
  + Encourage civic engagement and fuel betterment of the Charlottesville area

System Design

The PauseLab project uses a Ruby on Rails framework with a MySQL database. We used various gems (Ruby libraries) and a few client-side libraries to accomplish pre-solved tasks such as authentication, authorization, file attachments, and map displays. At a high level, our application is just a schema exposed by templates and glued together with gems and framework-related code.

User Roles

The application is designed around six different roles that a user can take. The roles are mutually exclusive, in that a user can only occupy one at a time.

* Admins: Site administrators who have the highest privileges, including creation (and deletion) of other users in other roles. They control and maintain the system.
* Moderators: Trusted users who are permitted to manage any data in the system unrelated to users (e.g. ideas, proposals, blog posts). They revise and filter site content.
* Steering Committee: Users who are permitted to see most data in the system, even if it not accessible to residents (see below), and are able to use this data to communicate with each other. They direct PauseLab’s focus and draw insights from the community.
* Artists: Users who propose public art projects to address community needs.
* Super Artists: A subset of artists who are selected to implement projects and report on their progress. They carry out the will of the community.
* Residents: Users to view the system’s published content and may submit ideas for improvements to their area. They are community representatives.

Application Phases

There are several steps in the chronology of one cycle (or as it is known in the code, an “iteration”) of the project. The first phase is ideas collection, where residents may submit ways that they perceive the community can improve. The next phase is proposals collection, where artists can submit more fleshed out plans that address the previously submitted ideas. The last phase is voting, where residents submit votes on which artist proposals should get funded. Once voting is complete, some number of projects are selected to be implemented, and users can visit the site to view news on progress.

Models

There are several models in the system that are implemented through a relational database schema (i.e. SQL); some are related to the project-specific tasks, while others are linked to auxiliary tasks like authentication and authorization.

* Idea: Submitted by residents in the first phase of the application. Contains freeform text about the idea itself, a category that the idea belongs to, a geographic location (i.e. latitude and longitude), and some personal information about the resident submitting the idea, including name, email, and phone number. When an idea is first submitted, it has an “unchecked” status, and is not publicly available. Only when an admin or moderator approves the idea does it become visible.
  + Category: Created by moderators as a means of grouping ideas together. Contains a short piece of text. Some example categories include: “Health and Safety,” “Art and Culture,” “Infrastructure,” and “Other.”
* Proposal: Submitted by artists in the second phase of the application. Contains freeform text about the proposal itself, a title, a link to the artist’s portfolio. Similar to ideas, proposals must be approved by admins or moderators before being publicly visible. Proposals may also be marked as “funded” to indicate to users which ones were selected in the voting phase.
  + Proposal Budget: Contains cost fields for various parts of the plan. This encourages artists to think about how they will allocate their funds if their project is selected.
  + Proposal Comment: Contains text for internal comments between members of the steering committee regarding a proposal.
* Vote: Submitted by residents during the last phase of the application. Has a many-to-many relationship with proposals, as well as personal information on the voter such as name, email, and phone number.
* Blog: Created by super artists or moderators during any phase of the application, but particularly after voting is complete, to report any news about PauseLab. Contains a title and freeform text body that may be embedded with formatting, images, or hyperlinks.
* Iteration: Captures the notion of cycles of the application. PauseLab can conceivably operate the ideas-proposals-voting process for community projects at least once per year, so each time it needs to reset it creates a new iteration. Each iteration has many ideas, proposals, and votes, and when each of these three models is shown on the site they are enumerated according to the current iteration.
* Landing page: Created by moderators for specific but mutable page content. Contains freeform text that may be embedded with formatting, images, or hyperlinks. Landing pages are exposed on the About page, the Ideas collection home page, and the pages that artists and steering committee are redirected to once they log in.
* Mass Email: Created by moderators for sending emails to certain role groups. For example, a moderator may want to send out an email to all artists reminding them to send in proposals before that phase of the application is over.
* User: Contains the profile of a users who authenticate into the system, including name, email, password, phone number, role, and an avatar image. Users may be associated with proposals, proposal comments, and blogs. However, they are not associated with ideas or votes because those submissions would typically be made by “one-time” users; the personal information collected in those submissions reside in those models themselves.
* Ability: Defines the permissions for each user role. Roles generally fall into a hierarchy where those further down the chain possess a subset of permissions belonging to those higher up. Admins are the top, followed by moderators, steering committee, super artists, artists, and residents, in that order.
  + Admins: Can manage (i.e. create, read, update, destroy (CRUD)) any type of data
  + Moderators: Like admin, but cannot manage user data
  + Steering Committee: Can create proposals, proposal comments, ideas, votes, and manage anything that they created. Can read all ideas, regardless of approval status. Can only read approved proposals.
  + Super artists: Like steering committee, but can create blogs, and cannot create proposal comments. Can only read approved ideas and proposals.
  + Artists: Like super artists, but cannot create blogs.
  + Residents: Like artists, but cannot create proposals.

Libraries

For most models (i.e. Idea, Proposal, Vote, Blog, Landingpage), much of the Rails code was boilerplate. We simply generated routes and controller code for the CRUD operations and defined the permissions (i.e. which users could take which actions) in Ability. The templates were designed to be clean and functional, and parts of the HTML were generated based on authorization. A few of the more interesting models and the gems used to address them are described below.

For user management by an admin, we created routes and controller code similar to typical models. However, the User model connected to the Devise gem to allow for some easy-to-use methods regarding authentication (e.g. checking if a user was signed in and automatically hashing passwords). Devise also allowed us to create forms for users who sign up for accounts themselves and automatically handled common user management tasks like password reset.

For user authorization, we used CanCanCan to enforce the permissions declared in Ability. For the CRUD operations this gem built authorization in automatically, such that if users do not have permission to see or do something, they are redirected to the root URL with an error message. The view templates used simple helper methods inside if-statements to only generate links or buttons that a user was authorized to follow or press.

Some models (i.e. Blog, Landingpage) necessitated the use of a “what you see is what you get” (WYSIWYG) editor, where the user could format text (size, style), insert images, and create hyperlinks without typing out HTML. To accomplish this we used Trumbowyg, which transformed <textarea> elements into easy-to-use GUIs.

To avoid automated bots from spamming certain form submissions (namely, user account creation and votes), we added Recaptcha (a CAPTCHA service provided by Google) as an extra validation step. The user needs to check a box and in some cases perform another task to prove they are human. A Recaptcha element can be embedded into the HTML and verified in controller code with some simple Ruby helper methods.

For internationalization, we used the i18n library native to the Rails framework. Any strings that are part of templates are indirectly referenced through a call to a translate method which looks up a key-value mapping in a YAML file. By creating different YAML files for each language and setting a locale variable in the request parameters, the proper string is shown on the page.

For several templates that show an instance of a model intended for public consumption (i.e. Idea, Blog), a Disqus thread is embedded at the bottom of the page. This platform allows users to post comments after they authenticate through a common service such as Google (or create a Disqus account). Even though this separates the application-specific account and the common one (possibly inconveniencing users who may need both), it captures a demographic that may want to contribute to the discussion without investing time into making a new account.

The final significant library is Google Maps, which is embedded in pages that display ideas. Since an idea contains coordinates about the area it addresses, Google Maps is an intuitive way of allowing location input on a form and then displaying that location to the public. Each approved idea shows up as a marker on the map with an icon that corresponds to the idea category. Zooming and panning features are built into Google’s Javascript code.

A variety of other libraries were used, including Paperclip for file attachments, Simple Form for easy form code, Bootstrap and Font Awesome for clean, elegant, and responsive styles, Rubyzip for zipping multiple files into one download, and Chartkick for producing line graphs and pie charts on a webpage.

Procedure

With the overall architecture explained, we will elaborate on system design on a per-phase basis. For each phase, we will describe how a user of each role interacts with the system.

Idea Collection

The first stage is idea collection. During this time, residents are able to submit project ideas for by completing the form and choosing a location. An email is sent afterwards to assure the resident that their idea is pending approval. Moderators will take those ideas and approve/disapprove them. Steerers can only edit ideas for content. Once approved, they appear on the homepage for all residents to view. Approved ideas can be shared via facebook or twitter. Clicking “ideas console” in the ideas tab allows for residents to like ideas.

Proposal Collection

Once all ideas are collected, the admin will change phase from ideas to proposals. During this time, artists are summoned to submit proposals. An email is sent once more to alert an artist about the status of proposals. Like ideas, proposals need approval from moderators/admins as well. Steerers can only edit information within proposals. Approved proposals remain unseen to residents. Once the phase changes to voting, residents will see approved all the proposals. Meanwhile, the home page will display all current ideas. Users can still like ideas during this time, but they may not submit ideas.

Voting

After the admin changes phase from proposals to voting, the home screen changes once more. All approved proposals will be displayed for residents where they can choose three proposals to vote for. To do this, the resident must fill out the voter ballot stating their name and contact information. They must also check the voter pledge saying they are within the residence they are voting from.

The proposal with the highest number of votes will become the project for the year. Artists that are associated with the project can create news posts that update residents about its status. These posts can be seen from the home page as well, where it displays three of the latest posts.

Iteration Complete

After all three of the phases are completed, the admin can end the iteration. This system flushes out the information associated (ideas, proposals, votes) and creates a new iteration. Users are not affected remain the same for the site. Within the iteration console, the admin/moderator can look at past iterations and export information about that iteration as a zip file.

Results

Some of the main issues PauseLab wanted improved was creating a way to vote on projects online, increase overall participation in idea submission and voting, streamlined project proposal submission, and a way for steering committee members to easily view and comment on proposals.  In the past PauseLab only offered on online form to submit an idea.

Conclusion

Further Work

To be discussed…

-Better visualizations: see how users interact and make changes from that

-> how many votes are being submitted over time

-More complicated for web security (prevent like-hacking)

-> Likes are based on cookies, remove the cookie and you can re-like the thing