Overview

Our main Georgia's Healing House application is split into two smaller Django applications. The first one is named georgias. This handles most of the outside-of-resident functionality. Things like donation information, incident reports and staff creation are in this application. The other application is where a bulk of our functionality resides — residents. Here, you have the ability to follow through the workflow of someone living at the house. This entails creating application, running an interview and terminating them once they have completed their time. This also includes a brief demographic chart to highlight the types of women that are currently in the house. All of the functionality and logic is within the app's respective view.py file. The templates that are being rendered are located in the template directory of the specific app.

We chose to set our system up like this because when we were going through the requirements, two main areas came to mind. One of these was the residents functionality which was described above — the ability to follow a resident through their workflow. There were also many ulterior functionalities that didn't seem to have much in common so we grouped those into the collective georgia's app. Things like sign in, donation data collection and user management fall under this domain. Previous to where we ended up, there was a desire to create a separate application for more substantial financial tracking and donation related data. However, due to time constraints, we had to drop this feature.

To store the data our application is collecting, we make use of a mysql database. The settings for this are found in the settings.py file within the georgias application. The database is locally on the machine, so the configuration required was limited — filling in the username, password and database name fields in the settings file. To set up our database we created the initial slp_georgias database and then granted all permissions to a user we had created. From there, whenever we wanted to update the database with model changes, all we had to do was run a simple makemigrations and migrate command. The layout of the tables within the database can be seen by taking a look at the models for each application. Each of the models listed in the file works itself out to be a table in the database.

Finally, all of our testing was conducted within the django provided utilities. Our tests for each application are located within the test.py files. To preload our test database, we made use of fixtures. These fixtures are located within each application's fixture direction. When the tests are run they automatically load the data into the database.

Django Packages Used

The django-formtools package was used to handle the size of large forms, namely the Interview Form. The formtools implementation can be found in the residents app views.py. The components relevant to the multi-page form are as follows: resident_interviewForm class, wizard_view, wrapped_interview_form function, named_interview_forms tuple, and templates dictionary.

The resident_interviewForm class holds all the functions and variables need for the wizard to create the multi-page form. It uses the named_interview_forms tuple and creates forms based on the ('name', form) tuples. The get_form_instance creates the Interview Report instance which is saved to the database. The instance variable above it must be initialized to None for the Wizard to properly create a model. The get_template_names function gets the template needed for the form according to the templates dictionary (e.g. page_1 form maps to interview_form_1 template). The done function handles the final steps needed to properly save the interview form for the corresponding resident. This function is called when the last form is validated.

The wizard_view variable uses the as_view function for the resident_interviewForm class to convert the class into the views necessary to handle the form. This wizard_view variable is used in teh wrapped_interview_form function, which functions as a wrapper for the multi-page form created for the resident which includes the resident ID in the url. This ID is used in the done function to save the Interview Form instance to the corresponding resident. This wrapped_interview_form view is what is linked to the urls.py file.

Georgias Application

In our Django project, the “georgias” application handles everything related to users, user profiles, donors, and incident reports. This includes user authentication, donation record keeping and statistics generation, and enabling/disabling users. Like in any Django application, the “georgias” directory contains several files within it that describe how to structure, mainpulate, and present data to the user.

In the model.py file for georgias, we have three main models. The UserProfile, Donor, and Incident Report underpin the three main categories of functionalities in the georgias application. There’s also an extra VolunteerHours model that is currently not being used because of requirements changes. This can be used to log volunteer hours for a particular user. The UserProfile model is linked to the Django user module as a means to extend beyond the basic user functionalities that Django provides — it allows us to add extra fields that we deem necessary. The IncidentReport is linked to both Django base Users and to other Residents to ensure that when a report is filed, we have access to all the members of the house that were involved in that incident.

The forms.py file contains all of the forms that we will be using throughout or application. Most of the code in this file revolves around using the aforementioned models to automatically create fields in a form. However, we also enforce the use of widgets and remove any fields we don’t want to show within the form. Some of the naming of the fields is also changed from the rudimentary variable names to something more human readable.

The urls.py file contains each of the URLs that we link to within our application to accomplish some functionality. Each of these URLs gets appended to the ‘/georgias’ prefix in practice. Many of the URLs are general, without any variable or data passage because these are pages for viewing information or form entry pages. When we do pass a variable through in the URL, that can be seen as “(\d+).” This means an integer is being passed into the URL — the id of the resident, donation or whatever is relevent to that page.

The views.py file in the application consists of the front-facing interface, login operations, volunteer hour functions, admin and staff functions, and reporting features. The front-facing functions consists of all the pages displayed at the home page of the system, with some logic to handle sending emails and validating forms. The volunteer hour functions allow both authenticated and unauthenticated users to submit their hours, and delete them if needed. The admin operation views implement all the functions that are available to an admin, which include enabling or disabling a staff member, adding or deleting a staff member, editing donation information, editing resident reports, and viewing sensitive data.

Resident Application

In our Django project, the "residents" application handles everything related to residents. This includes the application process, the termination process, management, reports and statistics. There are multiple folders and files that work together to keep the resident system running.

The models.py file contains the models for residents, related reports and application information. Each form throughout the application process and each report for managing and terminating residents store their data in the form of a model. Some models contain an attribute named "uniqueid" which is a unique identifier used for querying purposes.

The forms.py file contains all forms used in the website application. Each form is generally created based off of the corresponding model. Forms are used in the views.py file to pass on to the template to prompt user input. All form information is then scraped and stored in a model in the views.py file.

The urls.py file contains the URL paths for all resident affiliated webpages. Each URL in urls.py is appended to the base "/georgias/residents/…" Some URLS contain "/(?P<resident_pk>.*)" or "/(?P<report_pk>.*)" which is used to pass in the model's uniqueid attribute from the URL to the view.

The views.py file contains all the backend logic for each resident related web page. This includes storing inputted information, changing between views/templates and saving/deleting modified models. Some functions contain extra parameters such as "resident_pk" or "report_pk" which represent the resident's or report's unique ID attribute. The value of this parameter is retrieved from the URL and is used to query the necessary data for processing. Webpages that require some form of user input generally has an "if" statement that checks if it is either "POST" or "GET." When "request.method" is equal to "GET," the code displays the logic before any user input. When "request.method" is equal to "POST," the code displays the logic parsing the user input.