The Projections App

The projections application is a Django web application written in Python and served using IIS to help members of FE&P to manage their organization’s projections.

To get familiar with the Django web framework, please refer to its official documentation here:

<https://docs.djangoproject.com/en/4.0/>

The admin site is going to be referenced very often. You can access it by going to **Error! Hyperlink reference not valid.** name>/admin

The github repository is hosted here: <https://github.com/gamebot770/Projections>

If you are running from a clean database please make sure to run setup.py before continuing any further!

**Where is this server hosted?**

* It is currently hosted in FE-TS01 at the path “C:\Users\bss13\ProjectionApp”.
* Ask your manager at FE&P to get you access to this server.
* It can only be accessed by remote connection behind staff firewall though, so you will need to either physically be in FE&P’s office, or have some way around the firewall.

**Who has Access? Setting you up for development**

* We use the Django Cas NG project to tie into Rice’s NetID system
  + Settings for this plugin can be found in the projections/settings.py file
* There is a whitelist of people who have access to the site. These netIDs are defined in the AuthorizedNetID table. Staff are able to whitelist new netIDs by creating new records in this table with matching net IDs. Only Rice members with net IDs in this table are allowed to sign in.
* Setup.py should have created initial net ids for login into the system. You must also give yourself admin privileges to access the admin site.
  + To do this we are going to use the Django shell
  + Open up a command line or Powershell prompt and navigate to the project directory (it should have a file called manage.py in it)
  + Run the commands:
  + *python manage.py shell*
  + *from global\_libs.utils.auth import createNewAuthorizedUser*
  + *createNewAuthorizedUser (“<your netID here>”, True, True)*
* *Graphical user interface, application

  Description automatically generated*Go to the admin site
* Congrats you are now able to develop on the site!

**Discussing the Data**

* Organizations – This is the uppermost grouping of the data. Each Organization has a number of ParentLevelE’s associated with it.
  + AccountParentLevelE – This represents a category which accounts belong to. Projections are made with respect to these ParentLevelE’s
    - Accounts – This is the most low-level element in the data. Each account must belong to a ParentLevelE and an Organization

**Discussing the Django Apps**

Accounts

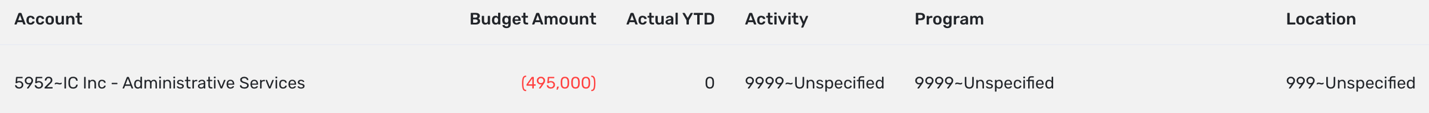
* This app is responsible for managing access to the site
* Notable files here include the access\_list.py file which is referenced when we create the intial AuthorizedNetIDs
* auth\_guard\_cas\_backend.py is where our filtering logic is for new net IDs

Data\_Upload\_System

* This app is responsible for allowing staff to upload their projections for the month
* This is where Records are created which approximately store a row of IO data. The Record objects are referred to very often throughout the code to serve information to the User, particularly in the projections\_update\_system

Projections\_Update\_System

* This app is responsible for serving the web pages that allow you to update an organization's projections
* This app is the largest of the apps in our repo and consists of a uniquely large amount of JavaScript code to accompany the typical python code
  + The JavaScript (JS) will request information from the server using Ajax or other JS methods such as *load*(). This allows for the webpage to operate smoothly without needing to refresh every time you need additional information from the server.
  + It does however mean that as a developer you need to be careful to make sure that the JavaScript you write and edit is hitting the right endpoints on the server. You need to also make sure that the method is correct as well (either a GET or POST).
  + We place all scripts in the {% block scripts %} block in the template.
  + Information sent back to the server could be a number, Boolean or even an entirely new template
    - Graphical user interface, text, application

      Description automatically generatedA key example of this is the detailed view that pops up when you click the info icon on the projections update page for any organization. This table displayed below is entirely served up by a *load()* request.
    - The server treats these as normal Django views. We have made sure to tag any view that could possibly be called by JS in the function descriptions. POST information will be received in request.POST.
* The templates also make use of template tags to execute small amounts of code such as formatting a string. If you need to generate small amounts of data such as the id of a Record being rendered, template tags are your friend and you should make use of them!
  + Examples of custom template tags include “get\_value” which allows us to retrieve a value from a provided map in the template using a given key. Another is the toCurrency tag which automatically formats a string representing money by rounding It to 0 decimal places and placing brackets around it if it is negative.
  + Note that these tags *cannot* be called in JS. More dynamic fields such as the projections field itself which the user writes in and also needs this currency formatting need JS to perform these operations. Hence, we also have a JS implementation of the toCurrency tag called the formatCurrencyField function
* The most critical element that ties the templates together is the element of a data-key.
  + Data-keys are attributes that are on the ProjectionUpdateForms that link all related data on each row in the Projections Update page to that ProjectionUpdateForm.
  + 
  + An example of a data-key is “E818”
  + This data-key will serve as the prefix for the id of all of the related fields for this projection form for example:
  + 
    - This is the related input for the E818 org’s projection form. We can get this using the id “E818\_projection” or by using the helper method getFormID(key, “projection”) which will output “E818\_projection”.
  + Currently the data-keys have 2 types
    - Summary type: This is string containing just the AccountParentLevelE ID
    - Child type: This is a string containing the concatenation of
    - "child\_" + accountID + "\_" + programID+ "\_" + activityID
      * For example
      * In this case the child data-key would be “child\_5952\_9999\_9999”
  + The data-keys are filled out on the creation of the form and passed to the templates
  + Data keys are essential to allowing for the JS to acquire values throughout the form and work with them.
* Using the convertFromCurrencyToFloat method is also essential to obtain numerical values from the formatted strings shown throughout the site.

Additional notes:

* The “Updated This Month” checkmark on the Select Organizations page only refers to whether or not a log exists for the most recent month.
  + If you want to reset these values you will need to delete all of the ProjectionUpdateLog items for that month for that Organization using the Django Shell. For example:
    - Graphical user interface, text, application

      Description automatically generated*ProjectionUpdateLog.objects.filter(org\_\_id=”<org\_id>”, timestamp\_\_month=”<fiscal month>”).delete()*
* I highly recommend using Pycharm Professional (with free student license) to run the application. It comes with a very handy debugger, allowing you to place breakpoints around the codebase.
* For JS debugging use the developer tools in Chrome
  + Right click on any element and inspect it
  + Graphical user interface, text, website

    Description automatically generatedUse the console to see logs from the functions
* While I sincerely hope you will never need to know the pain of setting up a Django server with IIS, if you ever need to do this the following link was very helpful
  + <https://medium.com/nonstopio/deploy-django-application-on-windows-iis-server-93aee2864c41>

About Add Row Function:

* The Add Row function is used to add a new row to a table (on the upper-left corner of the main table), which is implemented using JavaScript and jQuery. It also makes use of several HTML and Django template tags to generate the table rows and the dropdown menu. It is called when the user clicks on the "Add Row" button. The function creates a new row by cloning the last row of the table and inserting it just before the last row. The cloned row is then displayed by setting its hidden attribute to false.
* The new row contains a dropdown menu and several input fields. The dropdown menu allows the user to select an account from a list of accounts that are obtained from the server. The input fields allow the user to enter projections for various time periods.
* When the user clicks on the "Submit" button, the projections entered by the user are submitted to the server using AJAX. The function creates a list of added forms by iterating over the added rows in the table. Each added form contains the account selected by the user and the projections entered by the user. The list of added forms is then passed to a function called createProjectionAddedFormObjects, which creates a list of objects that can be used to update the projections in the database. The objects are then submitted to the server using another function called submitForm.