**REGISTER & LOGIN DETAILED DESIGN**

1. References
   1. Flask-Security-Too 4.1.0. Retrieved 8/7/21 from <https://pypi.org/project/Flask-Security-Too/>
   2. Flask-Security Documentation [for Flask-Security-Too]. Retrieved 8/14/21 from <https://flask-security-too.readthedocs.io/en/stable/>
   3. Website model for our input form formats: <https://sa.www4.irs.gov/irfof/lang/en/irfofgetstatus.jsp>.
      1. A copy of this page is provided as Screenshot(1).png.
   4. The current version of Data Design for User Authentication yymmdd.docx
   5. The current version of ‘Website standards yymmdd.docx.’
   6. The current version of ‘Create & drop yymmdd.sql’ (sometimes provided with a .txt suffix).
   7. The current version of ‘Insert & delete yymmdd.sql’ (sometimes provided with a .txt suffix).
   8. The current version of ‘Idea detailed design yymmdd.docx’
   9. The current version of High level module design Task 10 yymmdd.docx
2. **It is helpful to read Ref 1.i., Ref 1.d. and briefly reviewing Ref 1.f. before reading this document.**
3. Important notes:
   1. ***Throughout this and all our documents you will see many notes like “see Ref 1.d.”. Such notes always refer to a set of Reference documents like above, listed at the top of the document.***
   2. Please follow the standards for our Website as described in the current version of Ref 1.e. *for all work performed on every project*.
   3. Items highlighted in lavender describe functionality we hope to obtain from Flask Security.
   4. When statements are made in this document like, “process xyz in template abc does something” it is intended that the process occurs where appropriate and more likely in the Python or Javascript code associated with the template.
4. Purpose:
   1. The purpose of this task is to add user authentication and related functions such as registration and login to the SFF website.
5. Background:
   1. Pages will be added to the website with confidential information and users will be added with various roles, ideas (grant applications), and projects. The objective of this process is to securely connect the users to their authorized pages based on their roles, associated ideas and, in a future task, projects.
6. Design approach:
   1. Users visiting the SFF website have two specific authentication needs:
      1. Volunteers need to access various pages of confidential information based on their roles, and
      2. A different group of users need to connect to their confidential Ideas entered and displayed by the “Share My Idea” and “Complete My Idea” Flask forms connected to a MySQL database.
      3. Flask-Security-Too will be used to provide as much functionality as possible such as:
         1. Registering users by email address.
         2. Users changing email addresses.
         3. Change password
         4. Forgot password
         5. In using Flask-Security-Too please continue to adhere to SFF Website Standards (Ref 1.e.) to the extent reasonably possible.
7. Registration and login – general notes.
   1. Three input forms will be used for this process:
      1. Login.html
      2. Register.html
      3. Registercpt.html (Registration completion).
   2. All three forms will likely be smaller than our typical website pages.
   3. SFF users log in via login.html and register via register.html.
      1. See Ref 1.e. for our standard file locations including templates.
   4. Accessing Login.html. The template can be accessed at either of two website locations:
      1. Via the existing website menu, by selecting the last menu item at the end of the ‘About’ section, titled ‘login’, and
      2. The process currently initiated by the “Share My Idea” button located at the bottom of the ‘Share My Idea’ page, in the ‘Grants’ section.
         1. Immediately above this existing button, the following instruction will be added: “To create a new Idea or continue working on one you must be logged in. You may also log in here for all other purposes.”
         2. The title on the button will be changed to “Log In”.
         3. The new login.html page will be inserted first in the process initiated by this button.
8. Registration
   1. Purposes
      1. Determine the user’s identity
      2. Create and save an initial password for the user
      3. Determine the user’s role(s), if any.
      4. Input or update user information and populate table user.
   2. Users register via register.html.
      1. Register.html is accessed from a link on login.html.
      2. This and another input form, registercpt.html, collect, edit, and save user data to tables user and userproj, respectively.
   3. Format & format modifications
      1. The register.html page looks somewhat like the right side of screenshot(1) but with the format defined in Ref 1.e., which should be followed in all respects.
      2. Compared with the right side of screenshot(1), the following text changes should be implemented on register.html:
         1. Top title
            1. If the user clicked “Update My Info” in login.html to get here:

The top title will say, “Update My Info”

The row under the top title will read, “Users can review and update their SFF contact information here.”

* + - * 1. If the user clicked “New to the SFF Portal? Register here.” to get here:

The top title will say, “User Registration”

The row under the top title will read, “Interested users can register here to send in their ideas for grants and for all other purposes.”

* + - 1. Labeled boxes will be provided to enter each item on table User except ‘lastaccessed’ which is populated by the system.
    1. Required items should be identified on the form by a red asterisk (per Ref 1.e.).
    2. The Captcha process should be added at the bottom. See <https://support.google.com/recaptcha/?hl=en>
  1. Processing
     1. Validation (whether data is entered by typing or cut-and-paste).
        1. Follow the validation standards in Ref 1.e. (identified at the top of this document).
     2. When the form is displayed:
        1. The bottom “Continue” button is displayed but grayed out.
        2. If the user is logged in:
           1. The user is updating his or her profile.
           2. The email address is used to access table user and all possible boxes will be populated from table user.
           3. The user revises the data as needed.
        3. If the user is not logged in:
           1. The user is registering.
           2. A blank form is displayed and must be fully filled out.
     3. Email address
        1. If email address is populated and the user attempts to change it, the system responds as follows:
           1. If any errors exist in the data on the screen, the system will display an additional message stating that other errors must be corrected before changing your email address.
           2. If no errors are present, the system displays the message, “Changing the email address will change the ownership of your Idea(s) to the new email address. After this, the old email address will no longer be a registered user, you will be logged out, and, for your security, you will need to re-register. The system will, however, save all your old data to the new user email address likely making re-registration easier, and your old password will be valid for the new email address. Are you certain you want to do this?”
           3. The user is presented with ‘yes’ and ‘no’ alternatives.
           4. If the user confirms yes, the system performs the following in the sequence shown (due to database foreign key constraints):

The data as it then exists in the populated screen is saved to table user.

Insert rows into table userrole using the new email address and duplicating the remaining columns for all existing rows with the old email address.

Insert rows into table useridea and userproj using the new email address and duplicating the remaining columns for all existing rows with the old email address.

Insert a new row into table user using the new email address and duplicating the remaining columns for the existing row with the old email address.

Delete the row in table user for the old email address. Since both the userrole and useridea tables have been created with ‘on delete cascade’ clauses for their user foreign keys, the corresponding rows in both these tables will be deleted automatically.

The user is logged out and returned to login.html.

* + - * 1. If the user confirms no then the system changes the entry for email address back to the old email address.
      1. If the email address is not populated, it will display an error until it is.
         1. The user enters an email address:

The system will hopefully have a full-featured Flask extension to confirm the email address given is that of the person who will receive the password. It might operate something like:

The system sends a randomly generated numeric sequence to the email address and responds something like, “We have sent a confirmation code to your email address. Please enter it.”

A smaller box to the right of the email address box having the title “confirmation number” (or other title if provided by the Flask framework) is provided to enter the number.

Once the email address has been confirmed:

Hopefully the framework confirms this to the user by displaying a message to the user.

Hopefully the framework generates a temporary password until the user’s first login.

* + - 1. Enter & validate all other fields.
    1. Once error free, the bottom grayed out “Continue” button becomes its normal color.
       1. This “Continue” button is required because the user must confirm satisfaction with all data elements including those that are optional.
    2. The user presses “Continue”
       1. The system commits the data to table user.
       2. The ‘lastaccessed’ timestamp on table user is updated to now() for the user.
          1. Note: this timestamp update occurs even if no changes were made to the user’s data.
    3. (At this point we have four possibilities:
       1. User is either updating or registering.
       2. The user is either a volunteer or an Idea applicant.)
    4. Check userrole
       1. Whether or not the user is registering or updating their contact data, using the email address the system looks for rows on table userrole.
       2. If the user has at least one row on table userrole the user is an SFF volunteer.
          1. Display the note: “Thank you for volunteering with Strengthening Future Families. The work is important and your contributions *are gratefully appreciated!* Your new password is <password>. Please retain for future log ins.”
          2. If the user is registering, a “Return to Log In” button is displayed at the bottom.
          3. If the user is updating, an “Update” button is displayed at the bottom.
          4. In either case, when the button is clicked the user is returned to login.html.

This button is necessary so the user can save their password.

* + - * 1. The process is complete whether updating or registering since the user is definitely a volunteer.
      1. Whether or not the user is registering or updating their contact data, if the user has no rows on table userrole the user is a grant applicant.
      2. No bottom button is displayed.
      3. Registercpt.html is immediately displayed.
      4. (At this point we have an idea applicant who is either registering or updating their contact info.)
  1. Registration Completion (Registercpt.html) input form.
     1. Input boxes are provided for all columns on table Userproj (see Ref 1.f.) except emailaddress.
        1. The fields are populated with any available data from table userproj.
           1. Some users may have no data on table Userproj.
     2. The user modifies the data as desired and completes at least all the required items.
     3. As always, data is validated as it is entered.
     4. Either ‘CV’ or ‘About’ must be populated. If neither is populated, best is if both display red borders and error messages but at least one must display them and an error message provided.
     5. When the user is satisfied
        1. If the user selected register to come here from login.html:
           1. A “Register Me” button at the bottom of the form is displayed & clicked.
           2. The system updates the row on table Userproj and responds with something like:
           3. “Congratulations! You have successfully registered. Your new password is <password>. You can change it at next login.”
        2. If the user selected Update My Info to come here from login.html:
           1. An “Update” button at the bottom of the form is displayed & clicked.
           2. The system updates the row on table Userproj and responds with something like:
           3. “Update successful. Your new password is <password>. You can change it at next login.”
        3. The system displays a “Continue” button at the bottom of the page.
        4. If the Flask extension has a standard dialog for this process it is OK to use providing the resulting page format can be made similar to our standards and the dialog is similar to the preceding.
     6. When the “Continue” button is clicked, login.html is again displayed.

1. Login
   1. Purposes
      1. To determine what the user is authorized to see and do.
      2. To facilitate the user connecting to confidential data and processes.
   2. Format
      1. The login.html page will look somewhat like the left side of screenshot(1) but with the format defined in Input Form Standards, Ref 1.e., which should be followed in all respects.
         1. Top title
            1. The top “Welcome!” in screenshot(1) will be replaced with “Welcome to Strengthening Future Families’ grantmaking portal!”
         2. The text immediately below will read, “Interested users can register and log in here to send in their ideas for grants and for all other purposes.”
      2. Compared with the left side of screenshot(1), the following text changes should be implemented on login.html:
         1. The items should be centered since the right side items from the screenshot are not used on this form.
         2. For the items under ‘Registered user?’:
            1. The first row will be “Email address”.
            2. Password will not change.
            3. The “sign in” button will read “log in” and will be our standard oval button per Ref 1.e. but with a color indicating a log in button. The button may need to be a bit smaller than the navigation buttons around the website because of the smaller size of this page.
            4. Three links will be provided under the “sign in” button (where Screenshot(1) instead of the single “Forgot Password?” link).

Format: They will look like the “Forgot Password?” link in Screenshot(1) except

They should use our standard crimson color for text links on a white background (per Ref 1.e., paragraph “5. Links”).

The blue “Click Here” part will be omitted.

The link lines will be at least double and probably triple spaced to separate them so the user understands these are separate links.

First link

Text: “Forgot or change password?” (see our General Input Form Standards page for link standards).

It will connect to a standard process to recover a forgotten password.

Hopefully this functionality is provided by our user authentication framework module.

When this process is complete, the user will be returned to this login.html page.

If no authentication framework module is used, simply reuse the logic described for register.html.

Second link

Text: “Update My Info”

The entire string will be the link.

Underneath this and indented a little will be, “You must be logged in for this.”

The Register.html page is displayed but with different top title and bottom notes.

Third link

Text: “New to the SFF Portal? Register here.”

Probably use one size larger (if this still looks OK) for the font for this line so new users notice it most.

Only the “Register here” part will be the actual link.

Processing

The selected menu choice is saved for processing in register.html.

When each of these processes completes, the system returns the user to this login.html page.

* + - * 1. Omit the ‘Subscribe to our newsletter’ section.
  1. Validation
     1. Please follow the standards described in Ref 1.e. (see the References section at the top of this document)) and especially the standards in the **Validation** section.
  2. Processing
     1. Login.html is used for users to login.
     2. If the user logs in successfully column ‘lastlogin’ on table User is updated to now().
     3. Flask-Secuiryt-Too (Ref 1.a.), if possible, will be used to require the password to be changed at first login after registering. If not, a data element can be added to table user, which would be populated during registration to accomplish this.
     4. Two standard SFF buttons (see Ref 1.e.) are provided at the bottom center of login.html:
        1. One is an azure navigation button with the word ‘SFF Home’ inscribed in the center.
           1. When clicked it returns the user to the website home page.
           2. If the user was logged in when clicked, the user remains logged in.
        2. A second button inscribed ‘Create a New Idea’ is displayed only if the user is logged in.
           1. When clicked this opens a new “Share My Idea” input form with no items populated.
           2. See Ref 1.h. for processing associated with this alternative.
           3. *The process to which this button connects is the subject of a separate project.*
           4. Any user logged in may create an idea. No further authentication is required.
        3. Whether one or two buttons are displayed, they are centered.
     5. Above the buttons may be a set of links (if any) to pages the user is authorized to access.
        1. *Each heading and subheading for the links is displayed only if there are links to be displayed under the heading or subheading.*
        2. A list of zero or more links allow the user to open zero or more pages for each of the user’s roles under the heading, “Page links for My Roles”
           1. Table ‘**userrole**’ has a primary key of: <emailaddress> and ‘rolecode’ and each row on the table identifies a unique combination of a user with a role.
           2. The system provides access to each page authorized for the user’s role(s) but the details remain to be defined based on Ref 1.a. which we expect to provide much of the functionality.
        3. Additional lists of links to the user’s ideas may be displayed under the role links.
           1. Under a heading “My Ideas”, links for the user’s ideas, if any, are listed in two rows separated by a blank line. Each of the two rows may extend over multiple rows, if needed, to display all the ‘ideanum’:

Subheadings for the two rows are created as follows:

If there are no Idea numbers to be displayed for the SECOND row, no subheadings are displayed for either row.

Othewise:

The first row subheading is “My Initial Idea(s)”.

The second row subheading is “My Final Idea(s)”.

Some Ideas may be listed in both rows.

The first row lists, in ascending order, links for the Idea numbers (‘ideanum’). One link is created for each row for this user on either table ‘**userideatemp**’ or ‘**useridea**’.

The second row lists in ascending order, links for the Idea numbers (‘ideanum’). One link is created for each row for this user on table **‘userideafinal’.**

* + - * 1. All ‘ideanum’ of both types should probably be listed in a larger (and perhaps partly bold) typeface.
        2. If no rows are found for of any of these Ideas there is no expansion between the roles and the bottom one or two buttons.
      1. The system allows and facilitates this user opening the page(s) associated with these roles and Ideas but no one who is not so authorized.
  1. Other items:
     1. Hopefully our Flask extension will provide a process to prohibit all unauthorized persons/sources from accessing confidential pages.
     2. After logging in the user may navigate away from the SFF website or leave their PC unattended for an extended period. What options are there for automatically logging the user out?
        1. The ideal alternative would be one or more database parameters to control these functions.