Online Scientific Reference System's Documentation

Lesly Villa Jorge (Spring 2018)
Vladimir Palacios Contreras (Spring 2018)
Lisbeth Cardoso Guerra (Fall 2017)
Kyle Walsh (Fall 2017)

CIS 4914 Senior Project
&
EGN 4912 Engineering Directed Independent Research

Advisor: Dr. Mark Schmalz, e-mail:mssz@cise.ufl.edu
Department of CISE, University of Florida
Gainesville, Florida

April 23th, 2018

Structure of SRS

The online Scientific Reference System (SRS) currently includes several views that allow the user to see and interact with data differently. Each view can be accessed from the SRS tab of the application, *Figure 1*.

Figure 1: SRS Select Views

Directory List

The directory list view presents directories and notefiles similarly to how a normal file system would. Notefiles represent the basic file storage units within the system and contain notecards. By clicking on a directory, users are directed to a new location within the file system, which can be tracked by referring to the "Path" field at the top right of the SRS window, as shown in *Figure 2*. By clicking a notefile, users are able to access the notefile and can then proceed to a main menu with each of the notecards stored within that notefile.

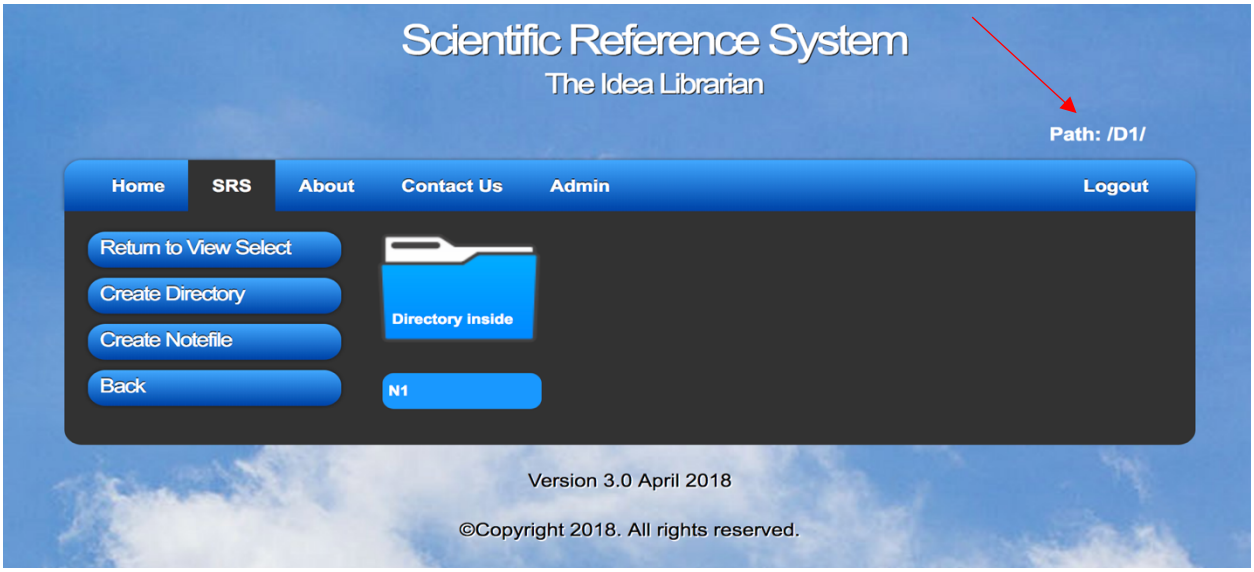

Figure 2: SRS Directory List.

Notefile List

The notefile list view presents all the notefiles in the system, *Figure 3*, regardless of what directory they're stored in. You can access these notefiles by clicking them, in the same way you would in the directory list.

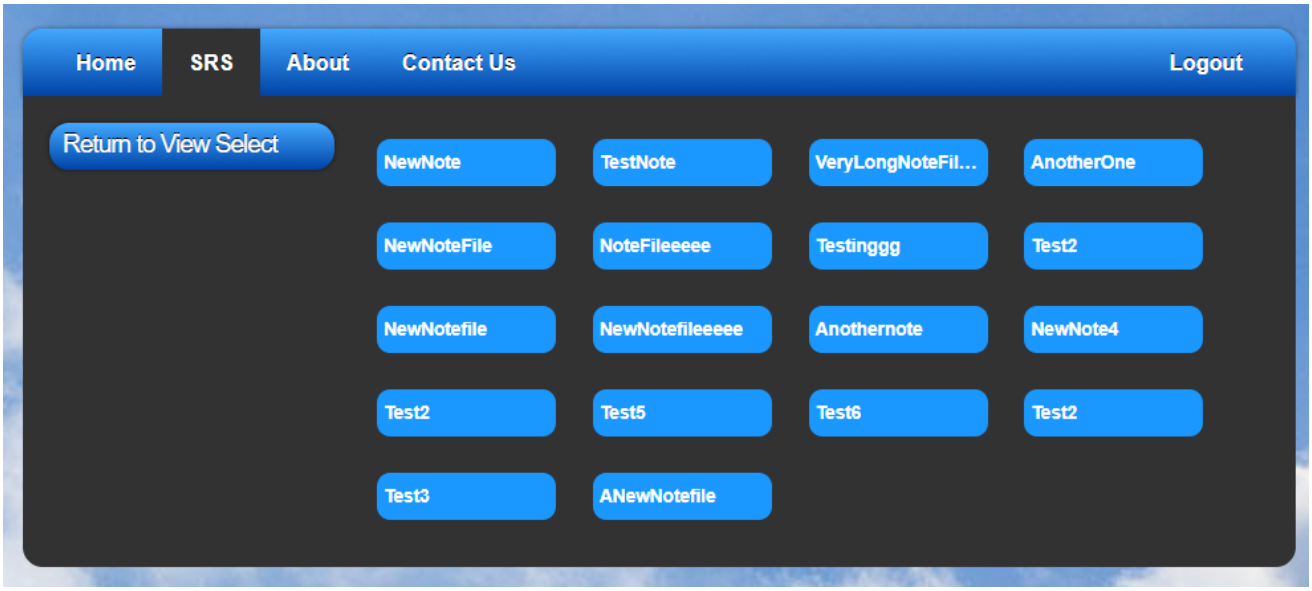

Figure 3: SRS Notefile List.

Image Archive

The image archive view presents all the image files in the system, *Figure 4*, regardless of what notecard they are associated with. The images can also be clicked to expand them and see additional details.

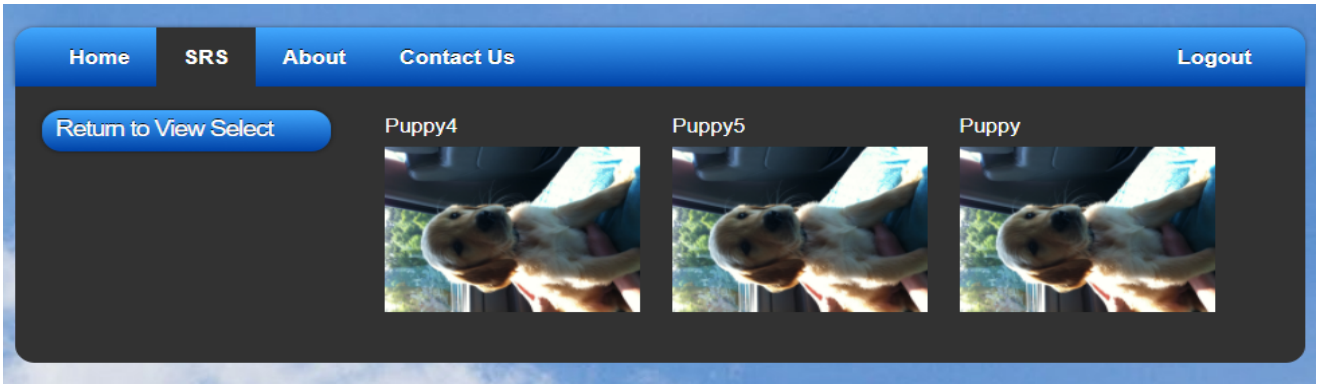

Figure 4: SRS Image Archive.

Video Archive

The video archive view presents all the video files in the system, *Figure 5*, regardless of what notecard they are associated with. The thumbnails can be clicked to open a modal with additional details and play the video clip.

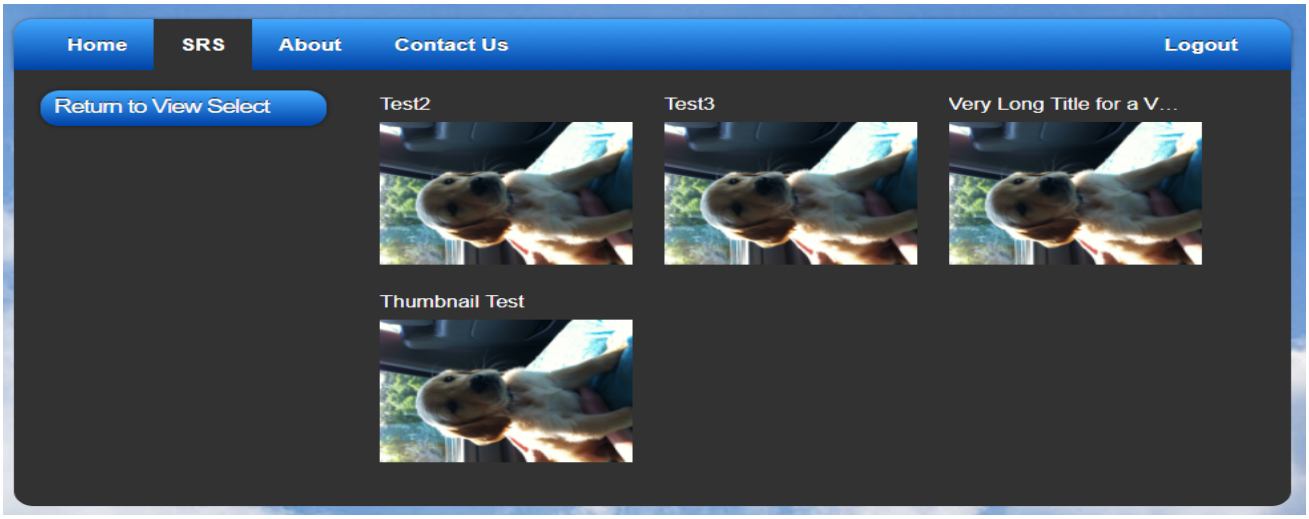

Figure 5: SRS Video Archive.

Audio Archive

The video archive view presents all the audio files in the system, *Figure 6*, regardless of what notecard they are associated with. The buttons can be clicked to open a modal with additional details and play the audio clip.

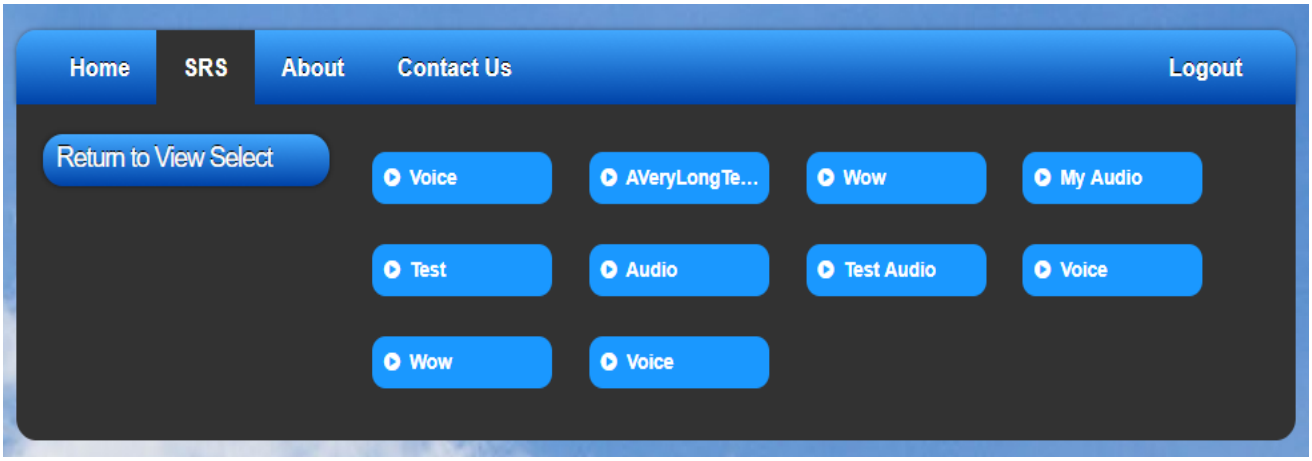

Figure 6: SRS Audio Archive.

Document Archive

The document archive view presents all the documents in the system, *Figure 7*, regardless of what notecard they are associated with. The links can be clicked to download the file.

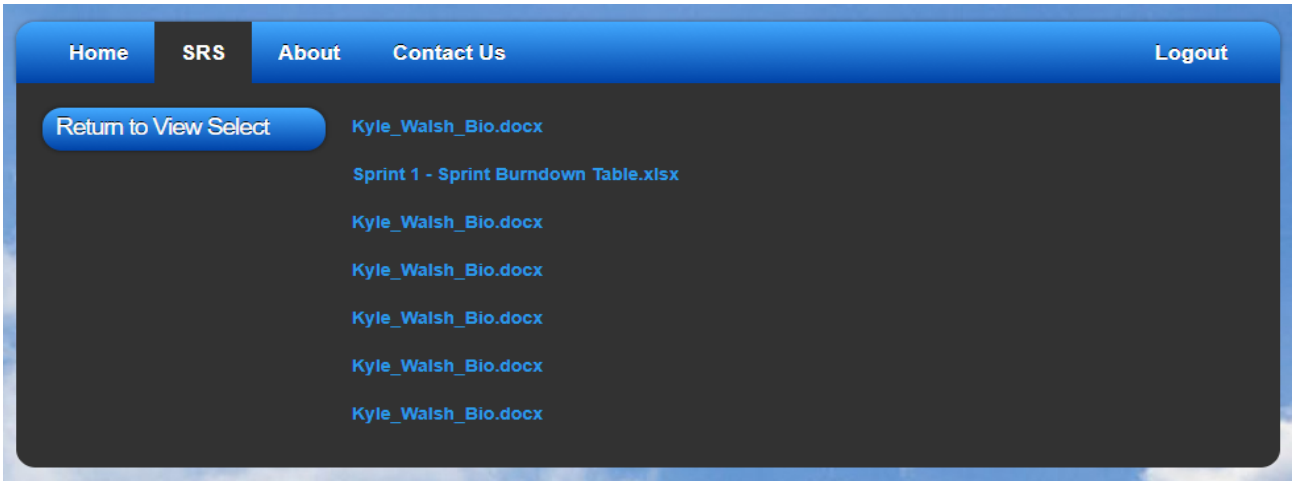

Figure 7: SRS Document Archive.

Functions of SRS

The online Scientific Reference System (SRS) currently has the following features: creation of directories, creation of notefiles, creation of notecards, import and export of notecards, notecard search functionality, addition of equations (both inline and display) to notecards, addition of images to notecards, addition of videos to notecards, addition of audio to notecards, addition of documents to notecards, and the ability to duplicate notecards and select what components you would like to keep. Also, in Spring 2018, the following features were implemented: edition of notecards, deletion (deactivation) of notecard, activation of notecard, selection of notecard, and show notecard's label.

Create Directory

SRS allows to create directories, *Figure 8*. Directories permit to group and organize information within folders. A directory can contain notefiles and other directories. A directory has a name, an author (which is the user that is logged in), a date of creation, and a parent directory. By default, the parent directory is called Home.

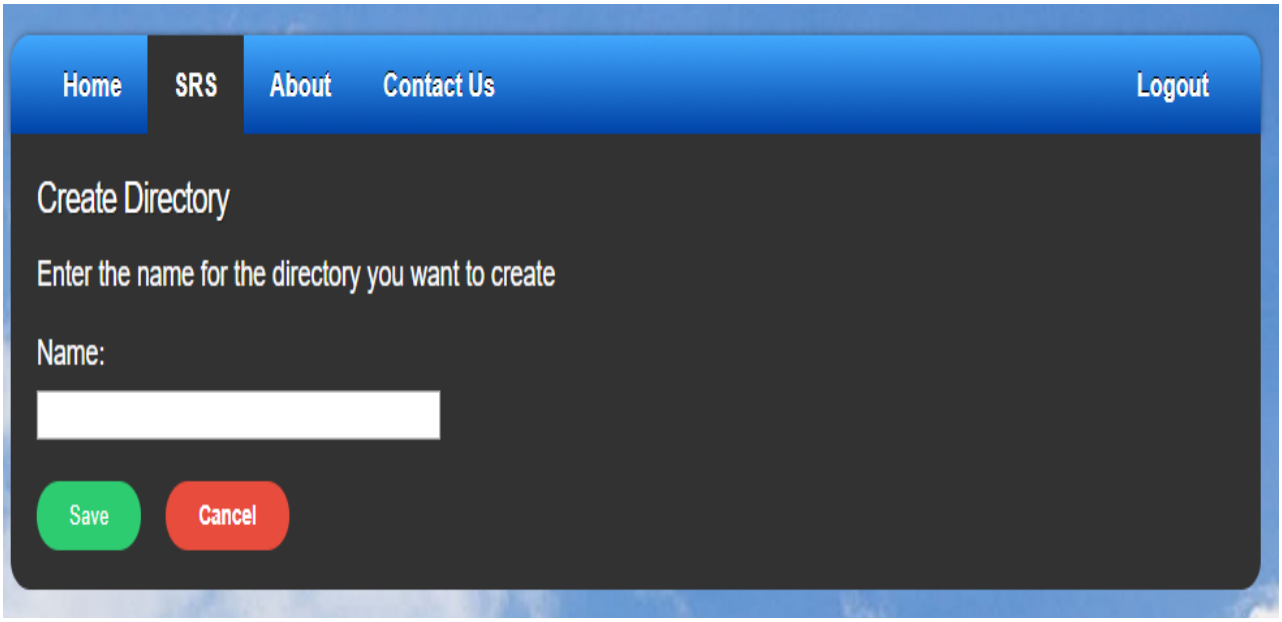

Figure 8: Create a Directory.

Create Notefile

Besides creating directories, SRS allows for the creation notefiles, *Figure 9*. Notefiles are contained within the current directory you have selected. They have name, author (which is the user that is logged in), label, body, date of creation, keywords and a list of notecards associated. A note file can contain zero, one or many notecards. So far, all of these fields are only text-based.

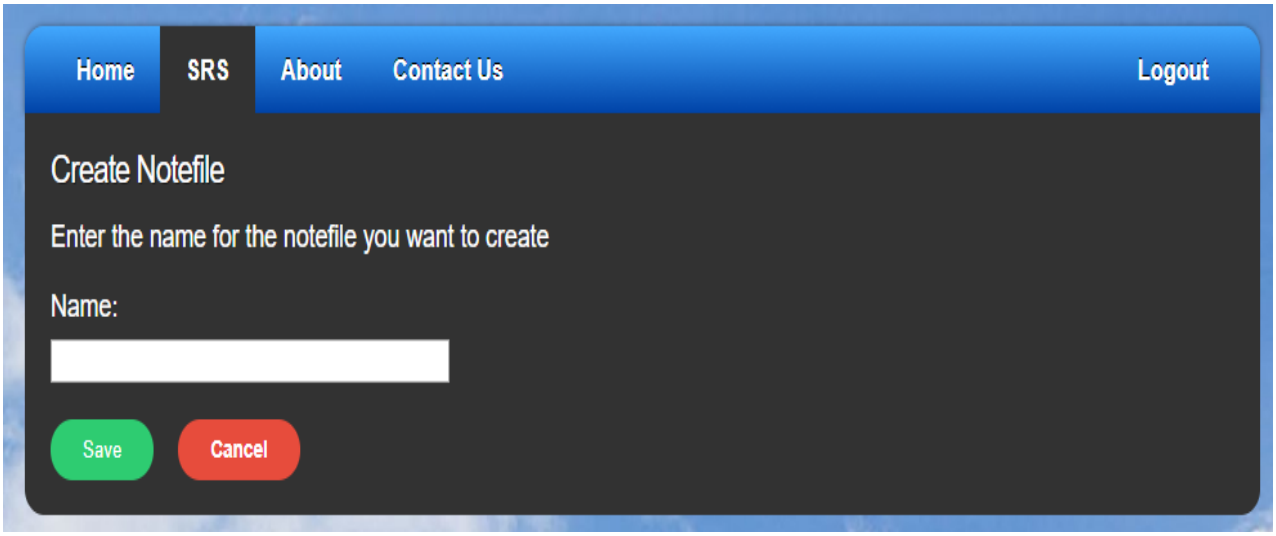

Figure 9: Create a Notefile.

Notecards

The SRS also shows the contents of notecards, *Figure 10*. Notecards contain text-based information regarding a reference, such as name, author (which is the user that is logged in), label, body, date of creation and list of keywords associated. As shown in Figure 10, and similar to SQUARENOTE, SRS shows only some of these fields, particularly name (title), list of keywords and body. Although they are not specifically stored as part of the schema for the notecard, each notecard also has images, videos, audio, equations, and documents that are associated with it as separate database objects in the database and displayed on the notecards page.

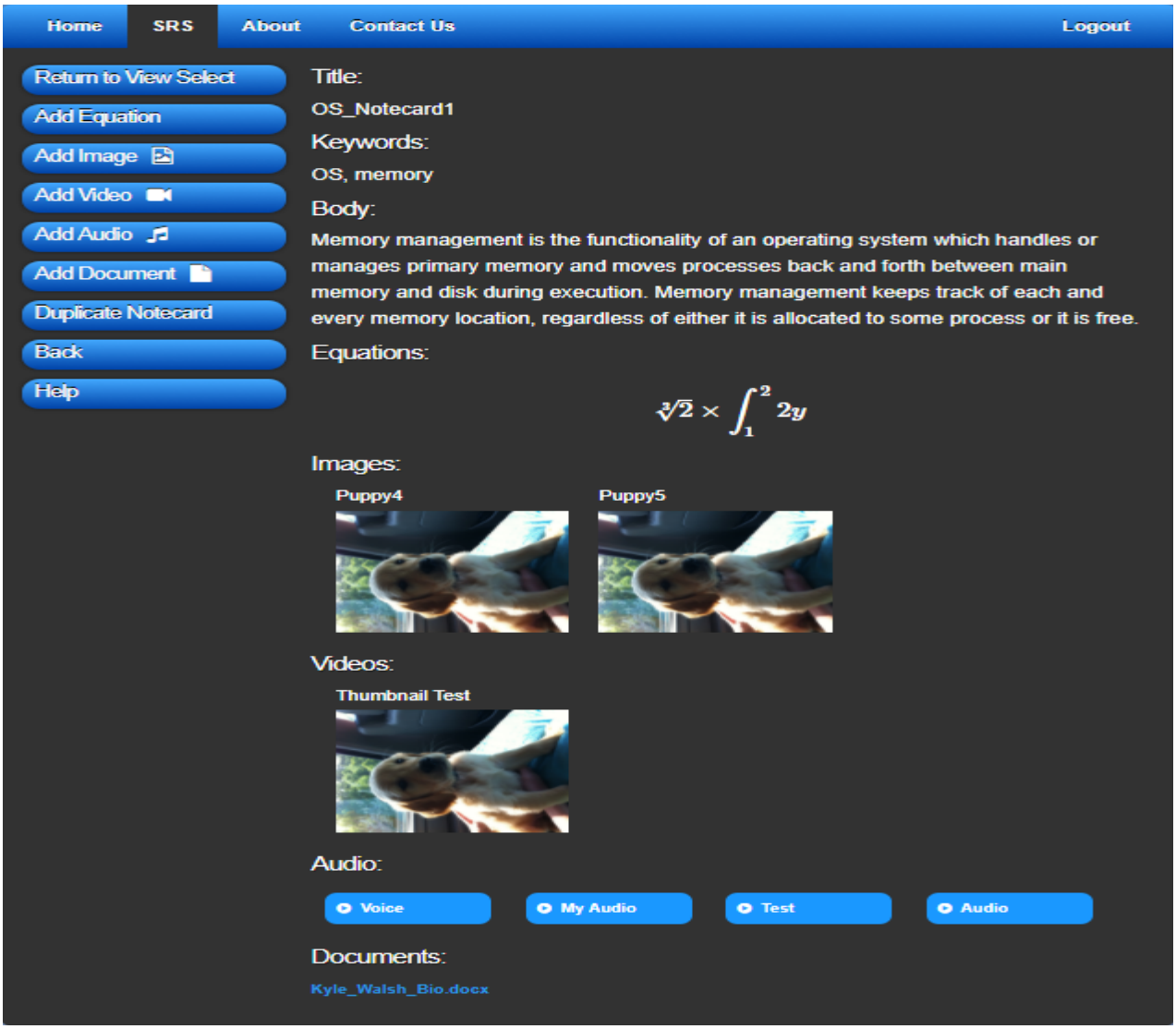

Figure 10: Notecard Layout.

Add / Display Equations

Within a notecard, users will be able to add equations using the equation editor shown in *Figure 11*. The equation editor allows to include a myriad of symbols, such as fractions, variables, matrices, parentheses, limits and series symbols - just to mention a few. Once users have finished creating their equations, they can choose to save or discard them by clicking the button 'Save' or 'Cancel' respectively. Afterwards, users will be redirected to the notecard detail page where they can see the newly created equation and also, they can keep editing the notecard.

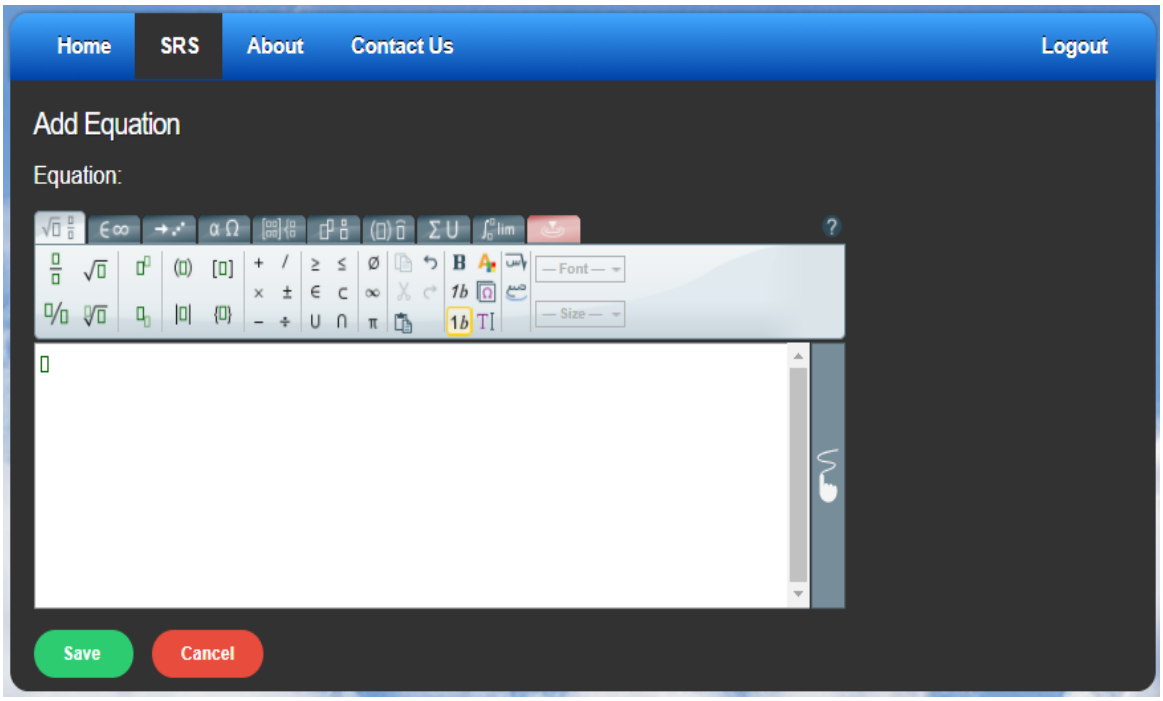

Figure 11: Add Display Equation.

Add Images

Besides adding equations to a notecard, SRS allows adding images to a notecard (see *Figure 12*). In order to add an image, users need to provide a *title* and a *source* path - source path can be a local or an online path. Before saving the image, SRS validates that the source path provided actually contains a supported image extension.

At this moment, supported image extensions are '.ani', '.bmp', '.cal', '.fax', '.gif', '.img', '.jbg', '.jpe', '.jpeg', '.jpg', '.mac', '.pbm', '.pcd', '.pcx', '.pct', '.pgm', '.png', '.ppm', '.psd', '.ras', '.tga', '.tiff' and '.wmf'.

Once users have finished entering the name and path fields, they can choose to save or discard their changes by clicking the button 'Save' or 'Cancel' respectively. Afterwards, users will be redirected to the notecard detail page, so they can keep editing the notecard. The notecard detail page will show the changes recently made.

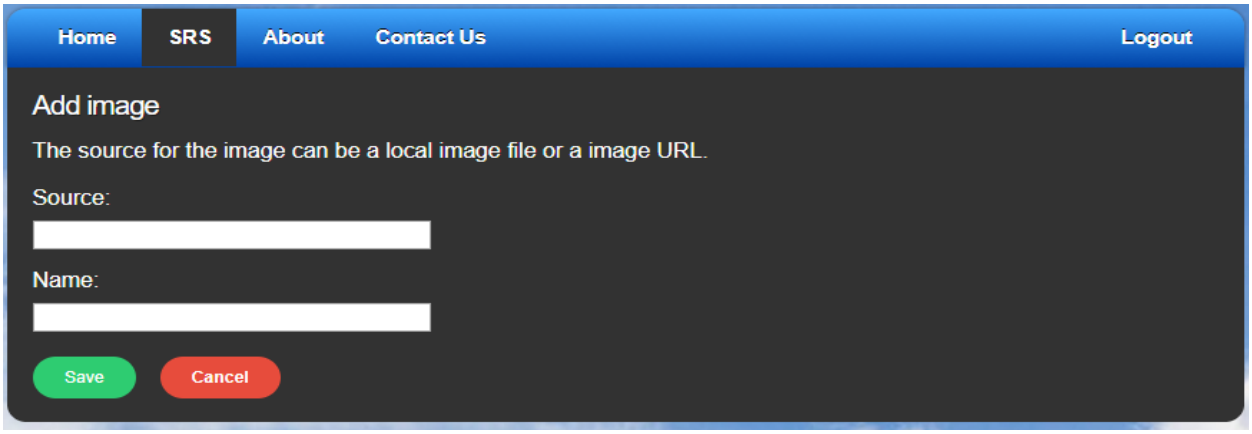

Figure12: Add Image.

Add Videos

In addition to equations and images, SRS allows to add videos to a notecard (see *Figure 13*). Videos can be local (videos stored in your device), online and Youtube videos. Similarly to images, users need to provide a *title* and a *source* path - source path where the video resides. Before saving the video, SRS validates that the source path provided actually contains a supported video extension.

Currently supported video extensions are '.264', '.3g2', '.3gp', '.3gp2', '.3gpp', '.3gpp2', '.3mm', '.3p2', '.60d', '.787', '.89', '.aaf', '.aec', '.aep', '.aepx', '.aet', '.aetx', '.ajp', '.ale', '.am', '.amc', '.amv', '.amx', '.anim', '.aqt', '.arcut', '.arf', '.asf', '.asx', '.avb', '.avc', '.avd', '.avi', '.avp', '.avs', '.avs', '.avv', '.axm', '.bdm', '.bdmv', '.bdt2', '.bdt3', '.bik', '.bin', '.bix', '.bmk', '.bnp', '.box', '.bs4', '.bsf', '.bvr', '.byu', '.camproj', '.camrec', '.camv', '.ced', '.cel', '.cine', '.cip', '.clpi', '.cmmp', '.cmmtpl', '.cmproj', '.cmrec', '.cpi', '.cst', '.cvc', '.cx3', '.d2v', '.d3v', '.dat', '.dav', '.dce', '.dck', '.dcr', '.dcr', '.ddat', '.dif', '.dir', '.divx', '.dlx', '.dmb', '.dmsd', '.dmsd3d', '.dmsm', '.dmsm3d', '.dmss', '.dmx', '.dnc', '.dpa', '.dpg', '.dream', '.dsy', '.dv', '.dv-avi', '.dv4', '.dvdmedia', '.dvr', '.dvr-ms', '.dvx', '.dxr', '.dzm', '.dzp', '.dzt', '.edl', '.evo', '.eye', '.ezt', '.f4p', '.f4v', '.fbr', '.fbr', '.fbz', '.fcp', '.fcproject', '.ffd', '.flc', '.flh', '.fli', '.flv', '.flx', '.gfp', '.gl', '.gom', '.grasp', '.gts', '.gvi', '.gvp', '.h264', '.hdmov', '.hkm', '.ifo', '.imovieproj', '.imovieproject', '.ircp', '.irf', '.ism', '.ismc', '.ismv', '.iva', '.ivf', '.ivr', '.ivs', '.izz', '.izzy', '.jss', '.jts', '.jtv', '.k3g', '.kmv', '.ktn', '.lrec', '.lsf', '.lsx', '.m15', '.m1pg', '.m1v', '.m21', '.m21', '.m2a', '.m2p', '.m2t', '.m2ts', '.m2v', '.m4e', '.m4u', '.m4v', '.m75', '.mani', '.meta', '.mgv', '.mj2', '.mjp', '.mjpg', '.mk3d', '.mkv', '.mmv', '.mnv', '.mob', '.mod', '.modd', '.moff', '.moi', '.moov', '.mov', '.movie',

'.mp21', '.mp21', '.mp2v', '.mp4', '.mp4v', '.mpe', '.mpeg', '.mpeg1', '.mpeg4', '.mpf', '.mpg', '.mpg2', '.mpgindex', '.mpl', '.mpl', '.mpls', '.mpsub', '.mpv', '.mpv2', '.mqv', '.msdvd', '.mse', '.msh', '.mswmm', '.mts', '.mtv', '.mvb', '.mvc', '.mvd', '.mve', '.mvex', '.mvp', '.mvp', '.mvy', '.mxf', '.mxv', '.mys', '.ncor', '.nsv', '.nut', '.nuv', '.nvc', '.ogm','.ogv', '.ogx', '.osp', '.otrkey', '.pac', '.par', '.pds', '.pgi', '.photoshow', '.piv', '.pjs', '.playlist', '.plproj', '.pmf', '.pmv', '.pns', '.ppj', '.prel', '.pro', '.prproj', '.prtl', '.psb', '.psh', '.pssd', '.pva', '.pvr', '.pxv','.qt', '.qtch', '.qtindex', '.qtl', '.qtm', '.qtz', '.r3d', '.rcd', '.rcproject', '.rdb', '.rec', '.rm', '.rmd', '.rmd', '.rmp', '.rms', '.rmv', '.rmvb', '.roq', '.rp', '.rsx', '.rts', '.rts', '.rum', '.rv', '.rvid', '.rvl', '.sbk', '.sbt', '.scc', '.scm', '.scm', '.scn', '.screenflow', '.sec', '.sedprj', '.seq', '.sfd', '.sfvidcap', '.siv', '.smi', '.smi', '.smil', '.smk', '.sml', '.smv', '.spl', '.sqz', '.srt', '.ssf', '.ssm', '.stl', '.str', '.stx', '.svi', '.swf', '.swi', '.swt', '.tda3mt', '.tdx', '.thp', '.tivo', '.tix', '.tod', '.tp', '.tp0', '.tpd', '.tpr', '.trp', '.ts', '.tsp', '.ttxt', '.tvs', '.usf', '.usm', '.vc1', '.vcpf', '.vcr', '.vcv', '.vdo', '.vdr', '.vdx', '.veg', '.vem', '.vep', '.vf', '.vft', '.vfw', '.vfz', '.vgz', '.vid', '.video', '.viewlet', '.viv', '.vivo', '.vlab', '.vob', '.vp3', '.vp6', '.vp7', '.vpj', '.vro', '.vs4', '.vse', '.vsp', '.w32', '.wcp', '.webm', '.wlmp', '.wm', '.wmd', '.wmmp', '.wmv', '.wmx', '.wot', '.wp3', '.wpl', '.wtv', '.wve', '.wvx', '.xej', '.xel', '.xesc', '.xfl', '.xlmv', '.xmv', '.xvid', '.y4m', '.yog', '.yuv', '.zeg', '.zm1', '.zm2', '.zm3', and '.zmv'.

Once users have finished entering the name and path fields of the video, they can choose to save or discard their changes by clicking the button 'Save' or 'Cancel' respectively. Afterwards, users will be redirected to the notecard detail page, so they can keep editing the notecard. The notecard detail page will show the changes recently made.

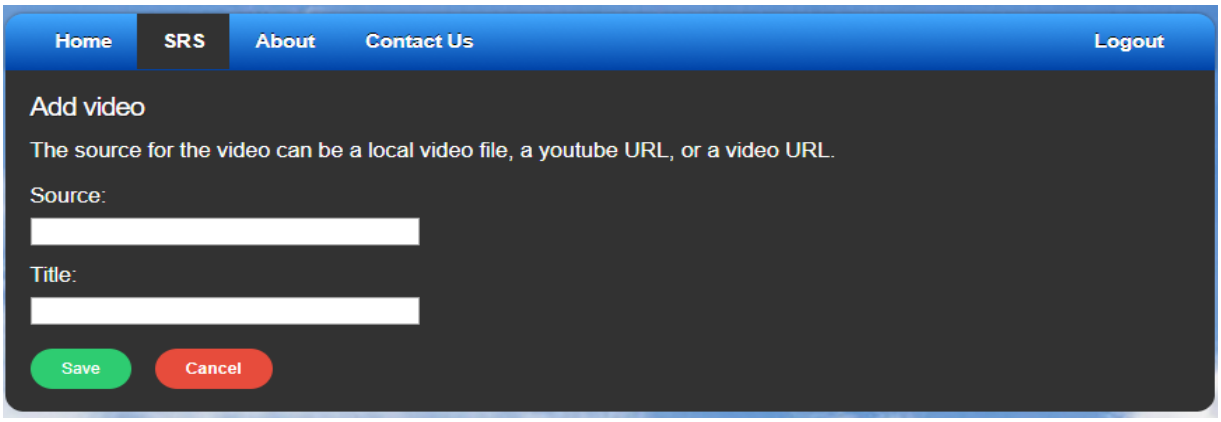

Figure13: Add Video.

Add Audio

Audio files can also be associated to a notecard. This is one of SRS' features allow to save audio files, as shown in *Figure 14*. These audio files can be stored locally in your computer or reside in the internet. Adding an audio file behaves likewise images and videos: users need to provide a *title* and a *source* path - source path where the audio file is hosted. Before saving it, SRS validates that the source path provided actually contains a supported audio file extension.

SRS currently supports the following audio extensions: '.3gp', '.aa', '.aac', '.aax', '.act', '.aiff', '.amr', '.ape', '.au', '.awb', '.dct', '.dss', '.dvf', '.flac', '.gsm', '.iklax', '.ivs', '.m4a', '.m4b', '.m4p', '.mmf', '.mp3', '.mpc', '.msv', '.ogg, .oga, mogg', '.opus', '.ra, .rm', '.raw', '.sln', '.tta', '.vox', '.wav', '.wma', '.wv', '.webm', and '.8svx'.

Same as with image and video files, users can decide whether to associate or not an audio file to a notecard once they have finished entering the name and source fields of the audio. They can do this by clicking the button 'Save' or 'Cancel'. Then users will be redirected to the notecard detail page, so they can keep editing the notecard. The notecard detail page will show the newly associated audio file, if one was created.

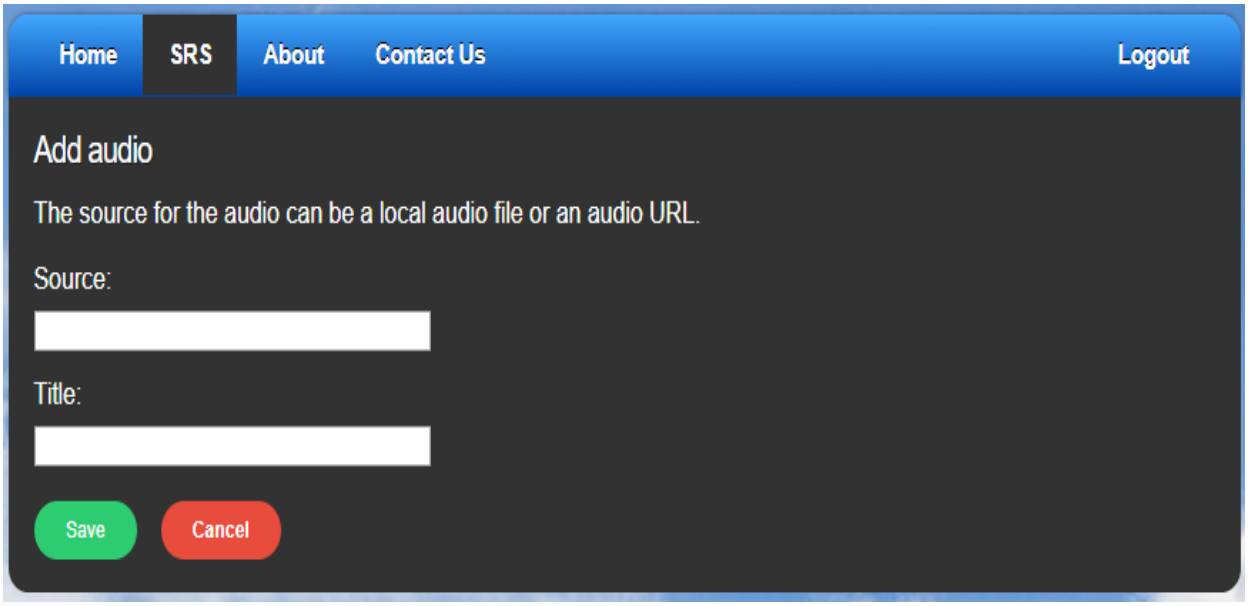

Figure14: Add Audio.

Add Documents

SRS permits to associate equations, images, videos and audio files to a notecard. It also allows to associate documents. As seen in *Figure 15*, SRS only requires a *source* field to allow adding a document. The path to a document stored locally or the link to an online document can be entered at this *source* field. SRS will check if the provided *source* path is one of its currently-supported document extensions.

These document extensions include '.pdf', '.doc', '.docx', '.xls', '.xlsx', '.ppt', '.pptx', '.pps', and '.ppsx'.

Once users determine whether to associate or not a document to a notecard (they can do this by clicking the button 'Save' or 'Cancel') they will be taken back to the notecard detail page. The notecard detail page will show the changes.

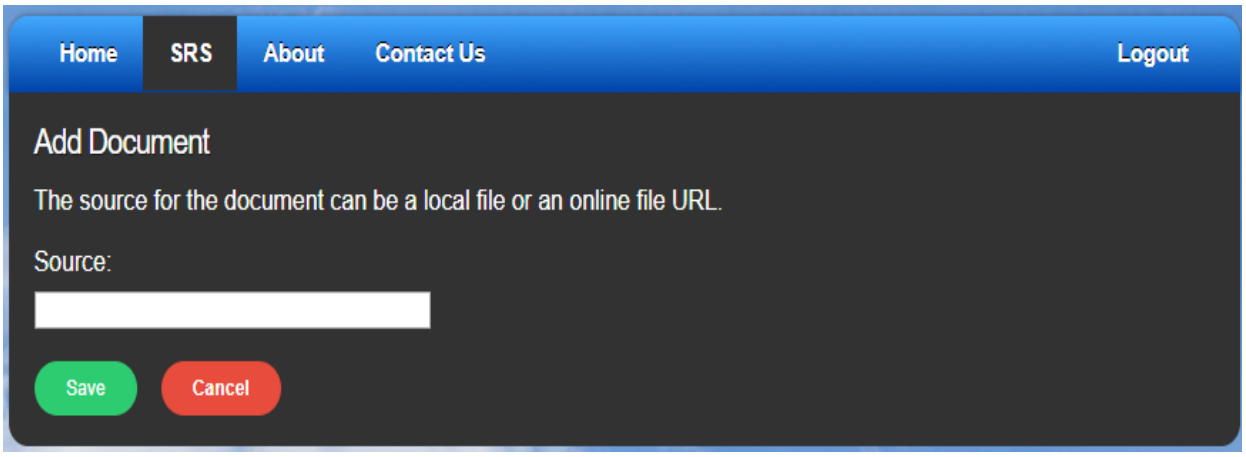

Figure15: Add Document.

Norcard List

As seen in *Figure 16*, one or more notecards can be grouped under one notefile. Similarly, to SQUARENOTE's design, only five notecards will be shown on cascade at any time. Users can move up and down the list of notecards using the keyboard's up and down. The list will be updated accordingly, and users will be able to see all the notecards in the list.

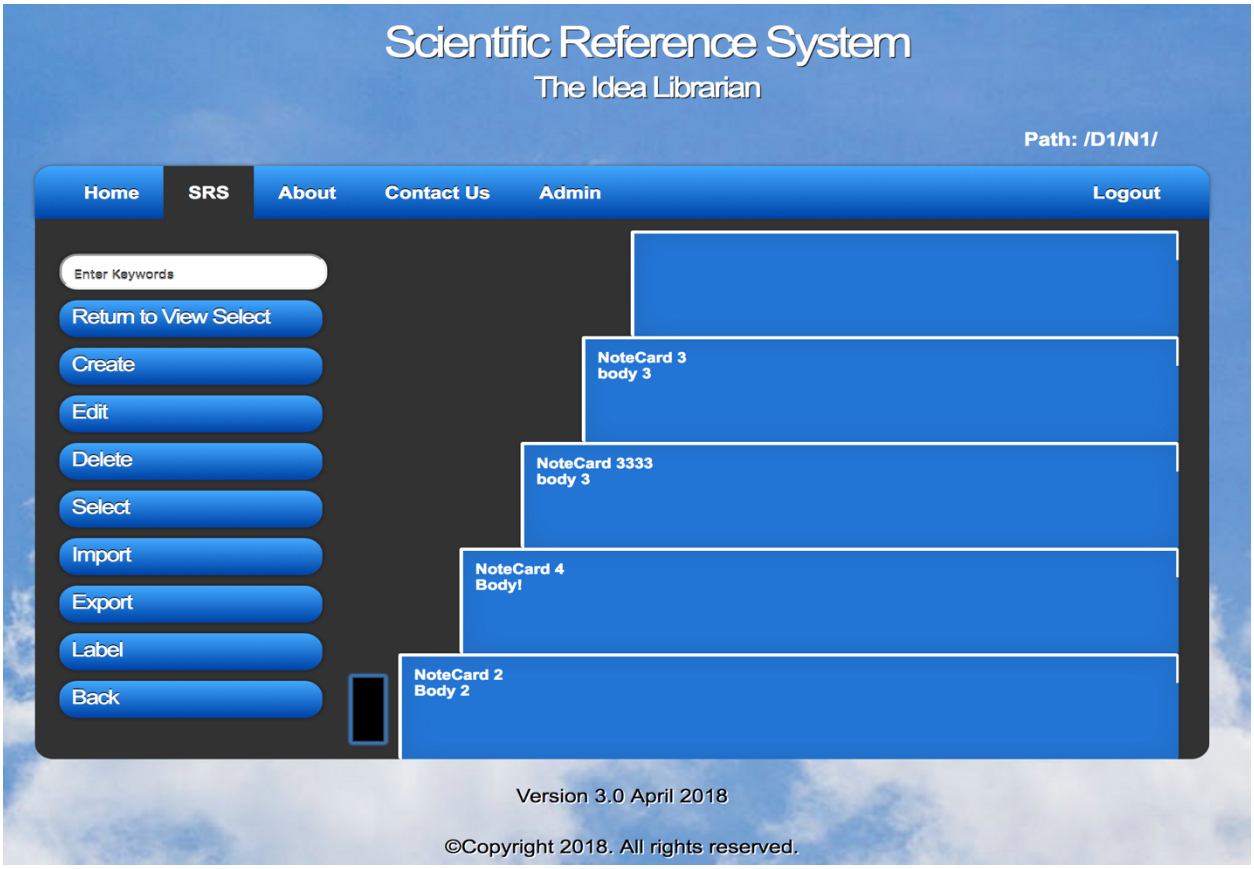

Figure 16: List of Notecards Contained within a Notefile.

Create Notecard

Notecards can be created by being imported (see section below) or by clicking the "Create" button on the notecard list page seen in *Figure 16*. By filling out the form shown in *Figure 17* with the pertinent details, notecards can be initialized and populated. In addition, there are two extra buttons beneath the form that say, "Insert Equation" and "Insert Link". These buttons produce modals that allow you to add inline equations or links into the body of the notecard. The equation modal, see *Figure 18*, will inject a MATHML structured statement into the body that will be rendered as the actual inline equation on the notecard page. The link modal, see *Figure 19*, will inject a link in the form of a tag that will be rendered as a link with the anchor text matching the display text entered and point to the given URL.

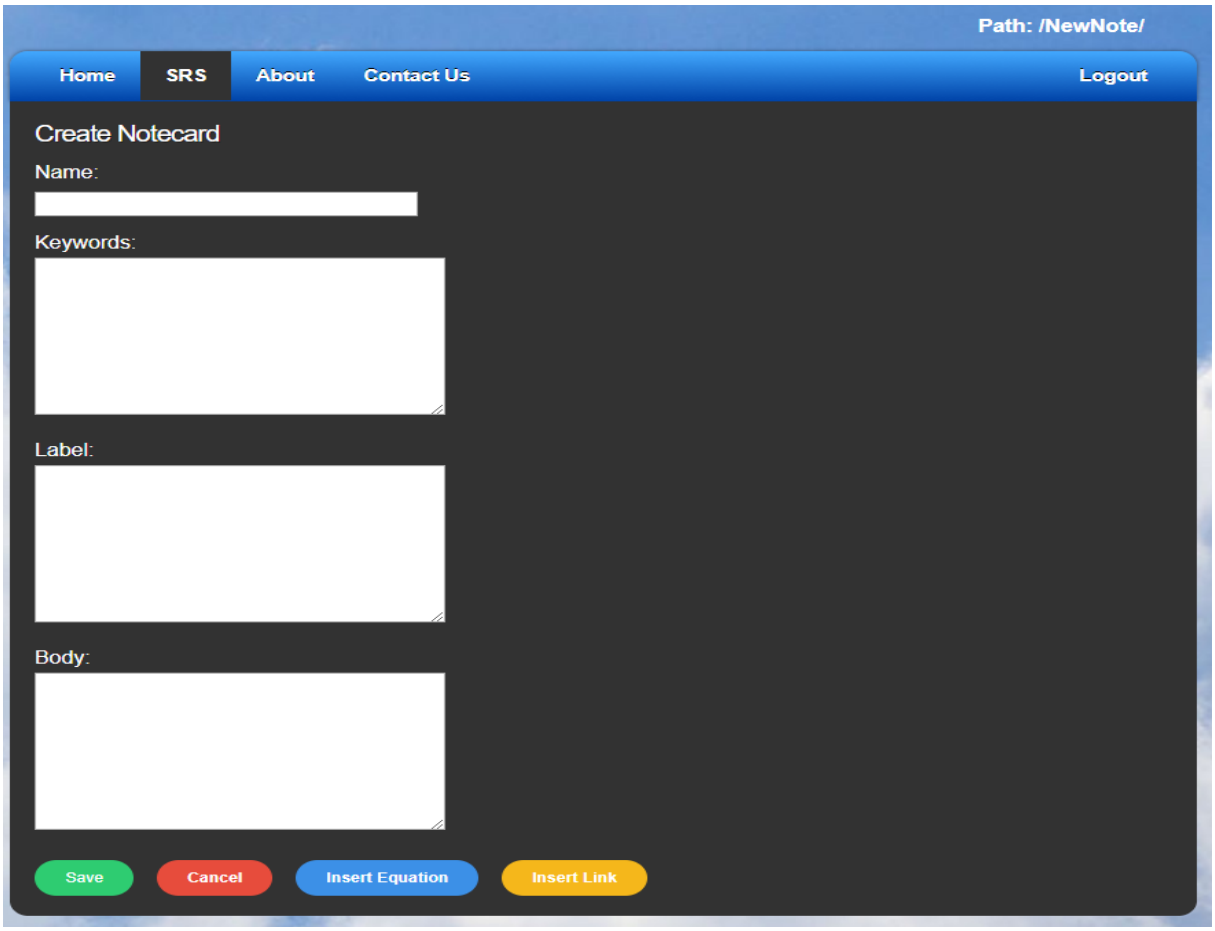

Figure 17: Create Notecard.

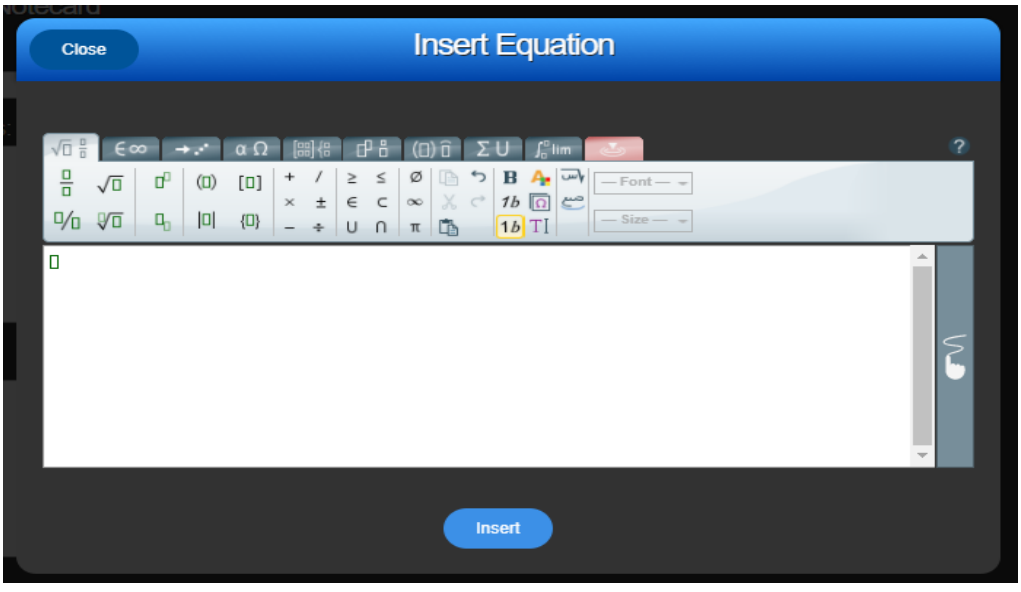

Figure 18: Insert Inline Equation Modal.

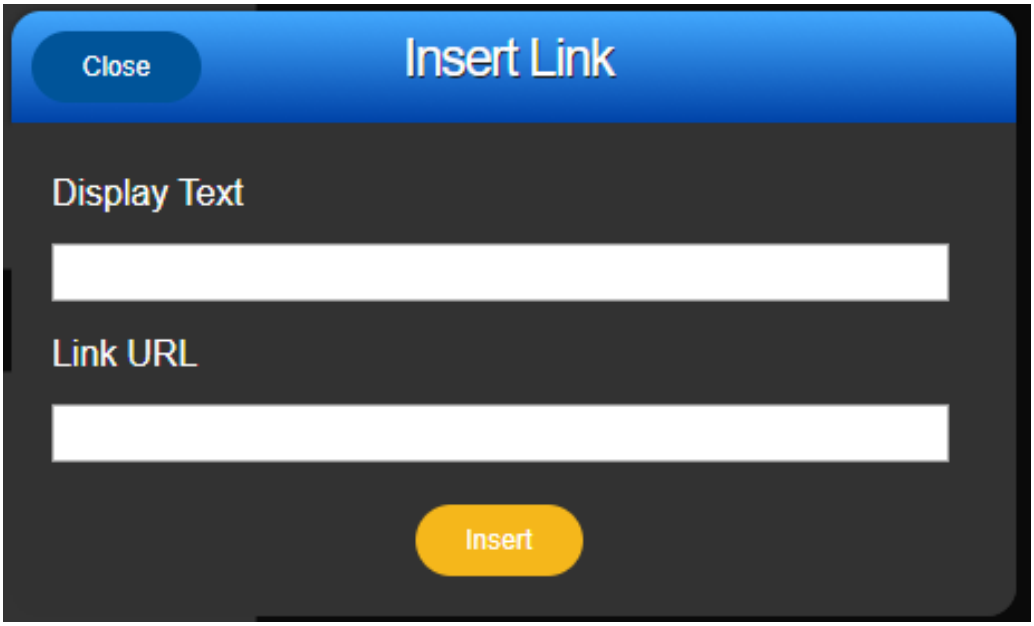

Figure 19: Insert Link Modal.

Edit Notecard

Edit button allows users to make changes to an existing notecard and save those changes into the database. When this button is clicked, the HTML form sends the request and the pk (id of the notecard) to views.py through urls.py. The edit method will then receive these parameters, handle the request, update the database, and send the response back to be displayed on the website.

Delete (Deactivate) Notecard

The delete button does not permanently "delete" the notecard. It rather deactivates the notecard temporarily which it is not shown on the page. The goal of this is to allow researchers to recover back any deactivated notecard. After clicking this button, two popup windows will appear to confirm this action as shown in *Figure 20* and *Figure 21*. The second popup also informs the user that this action can be undone by just pressing the keys Ctrl + Z.

To implement this feature, a Boolean “notecard.activate” variable was added to the “delete_notecard” method in views.py and set as false by default. This method accepts two parameters as well, the request and the pk of the selected notecard, and stores the date of this action. By iterating through all notecards, each deactivated notecard is stored in a dynamic array (list) of deactivated notecards and then sorted by date from the most recent to the old ones. Another dynamic array (list) will contain the activated notecards. Then, it handles the request, sets the flag to false, updates the database, and sends the response back.

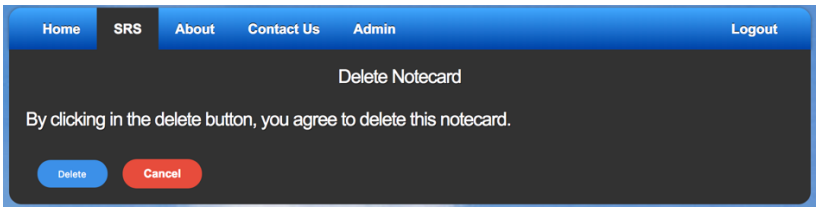

Figure 20: Deactivate View.

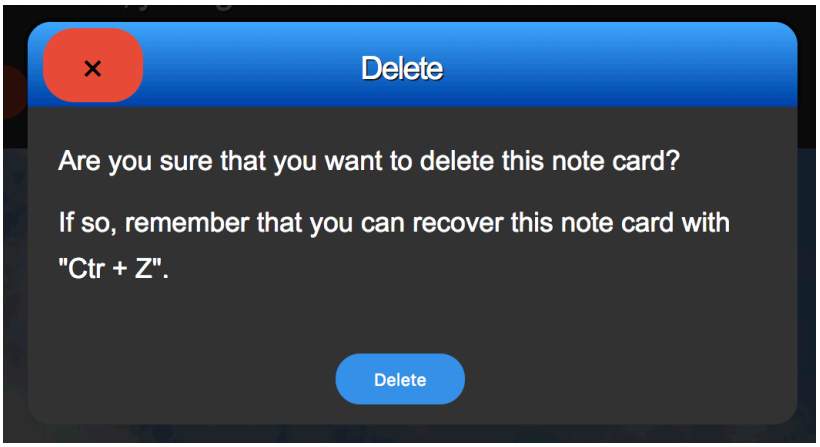

Figure 21: Deactivate Popup View.

Activate Notecard

If a notecard has been deactivated, this option will allow the user to recover as many deactivated cards as there are in the database. After pressing the command keys “Ctrl + Z”, the popup in *Figure 22* will appear.

The implementation of this feature is the opposite of the previous delete functionality. Here the Boolean “notecard.activate” variable is set to true. Since the deactivated notecard’s

array has been already sorted by most recent deactivated notecard, at every Ctrl + Z the deactivated notecard at position 0 will be popped from the deactivated notecard list and pushed to the activated notecard list.

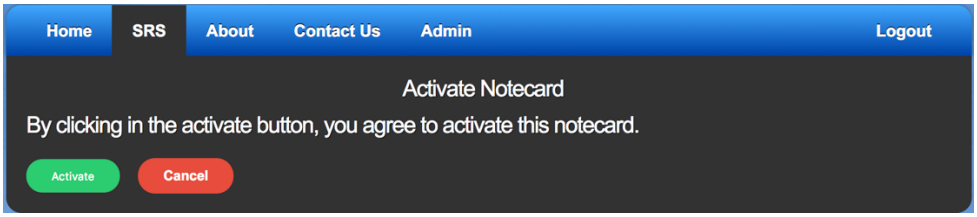

Figure 22: Activate Notecard View.

Select Notecard

As its name indicates, this button can be used to view the selected notecard. Using the up and down arrow keys, the black rectangle next to each notecard will move accordingly checking out the selected notecard.

Label Notecard

The Label button functionality allows viewing the selected notecard label, giving the users the opportunity to just see this feature among all note card's features, see *Figure 23*. This label is a text-based field which explains the purpose of the note card.

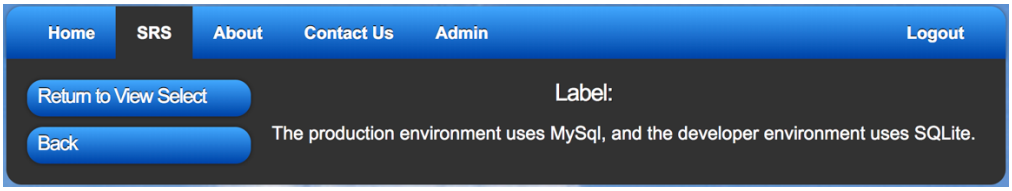

Figure 23: Label Equation View.

Explain Equation

This button will allow the user to see a better explanation of an equation inserted in the notecard, see *Figure 24*. When clicking this button, a popup window will display the actual equation and a brief explanation.

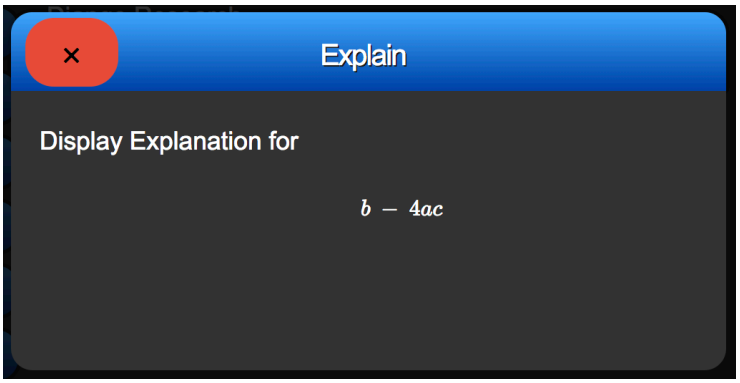

Figure 24: Explain Equation Popup View.

Import Notecards

The SRS also has the ability to import notecards, as shown in *Figure 25*. So far, SRS imports all notecards contain within a binary file with SQI format. In the future, SRS should allow to import notecards from files on other formats, such as SQN format. Moreover, there is no option to import a particular notecard within the list of notecards. This might be implemented in future development as well.

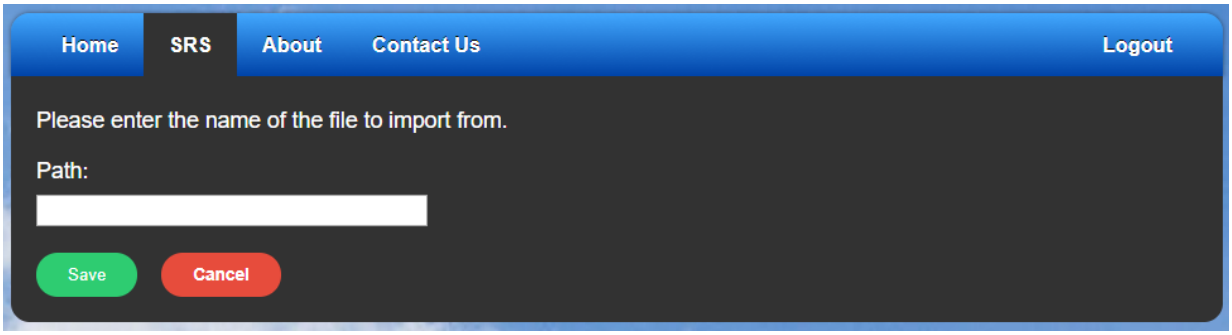

Figure 25: Import Notecards from a Binary File.

In order to import notecards from a binary file, users only need to enter a valid path, assuming that the path points to a correct binary file. If the binary file's format is correct, then SRS will create notecards containing all the information within the binary file. Only when the path is valid and the binary file's format is correct, then SRS will import the notecards. Otherwise, it will show an error message.

As shown in *Figure 26,* the correct format of a binary file includes the *header* for the file, the *keyword-start delimiter*, the keywords (which might be empty if there are no keywords associated to the notecard), the *keyword-end and header-start delimiter*, the name or title of the notecard, the *header-end and body-start delimiter*, the body of the notecard, and the *body-end delimiter*. Except for the first field, the *header* for the file, all other fields repeat themselves for each notecard. Of course, the information regarding keywords, the name or title of the notecard, and the body of the notecard will likely change for each notecard.

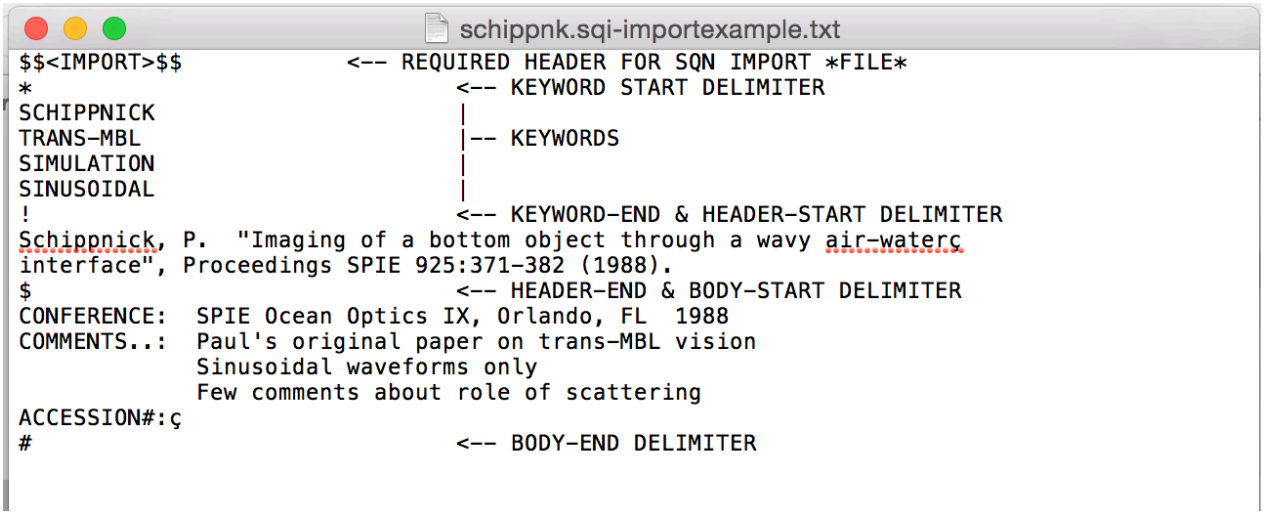

Figure 26: Format of a Binary File.

Export Notecards

In addition to import notecards to a notefile, SRS can also export notecards to a binary file with SQI format. In the future, SRS should allow to export notecards not only to binary files but to SQN files. Similarly, to the import feature, users need to enter a valid path for the binary file to be created (see *Figure 27*).

The correct format of the binary file is depicted in Figure 16. At this moment, SRS exports all notecards contained within a notefile to a binary file; it should also have the option to export notecards one by one, as SQUARENOTE does.

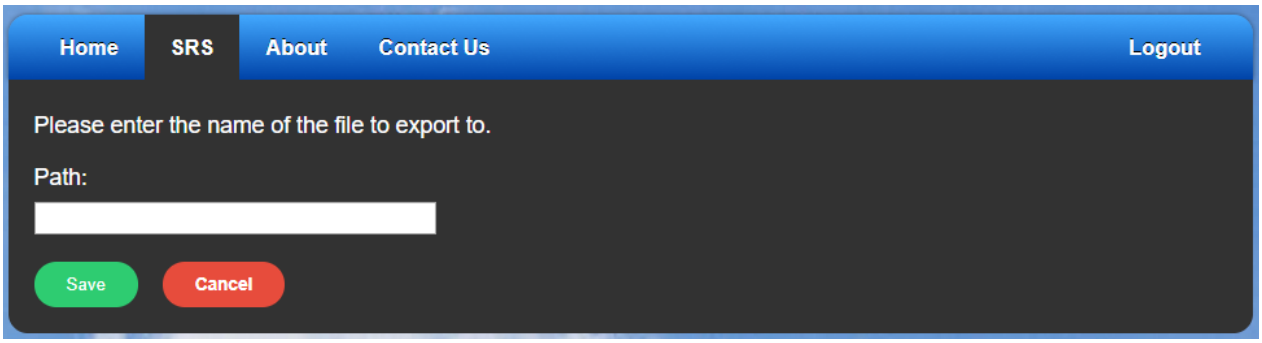

Figure 27: Export Notecards to a Binary File.

Search

Another feature of SRS is the ability to search keyword within notecards contained in a notefile. As seen on *Figure 28*, users can enter the keyword or keywords they are looking for, and SRS will update the list of notecards displayed. The new list of notecards will contain only the notecards that have at least one of the keywords being searched for. In the future, SRS should also have the option to search keywords among notefiles, not only notecards.

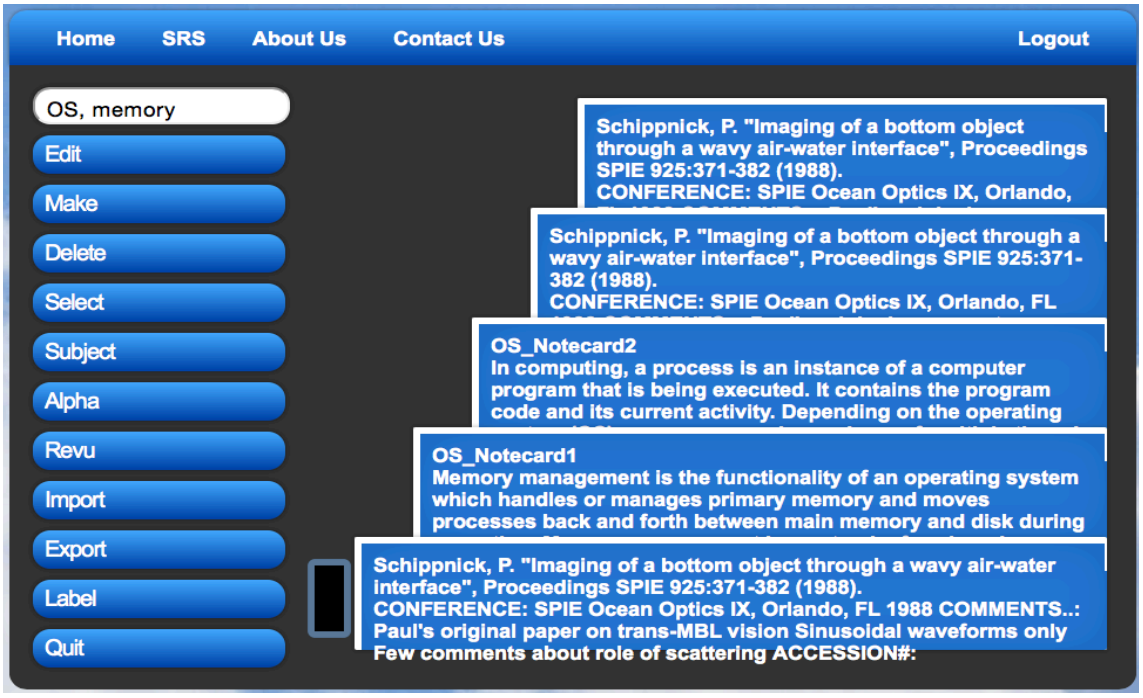

Figure 28: Search Keywords within Notecards and Notefiles.

Deployment

Users can test this application through the URL http://srs-prod.cise.ufl.edu/ or use the IP address 10.36.138.50. A server, "srs-prod", in the UF's CISE department is dedicated to run the production environment of the SRS application. This server is based on Linux, Red Hat operating system, which allows developers to deploy future modifications in an easy way. The deployment process could be done by an authorized developer under the UF's CISE network without the intervention of a system administrator.

In order to make the process of deployment dynamic, the settings.py file was modified to run in both environments, developer and production, see *Figure 29*. If the Django's hostname is developer, the SRS application will use the SQLite local database; however, if the Django's hostname is production, the SRS application will use a MySQL database that lives in the CISE's database. To use CISE's database, the application needs to know the corresponding database credentials. However, to avoid a security breach if these credentials get compromised, for example in GitHub or through emails, these credentials are saved in an encrypted file, my.cnf, inside the server, and the SRS application will know them. In addition, a secure communication protocol, HTTPS, was enforced for this web application to avoid vulnerabilities over the network while the authentication process is in transit.

There are two main reasons why the SRS application currently uses MySQL database in the production environment. First, the UF's CISE department is migrating all databases from SQLite to MySQL for security purposes, thus implementing it now will save time in the future. Second, the SRS application is using a database outside of the server, in the department's MySQL database. Therefore, if accidentally the "srs-prod" server is compromised, the data will be safe, and after pull the application code from GitHub, SRS could be synchronized again with-it MySQL database.

```
if socket.gethostname().endswith('-prod'):
    DJANGO_HOST = "production"
elif socket.gethostname().endswith('-test'):
    DJANGO_HOST = "testing"
else:
    DJANGO_HOST = "development"


# Define general behavior variables for DJANGO_HOST and all others
# Define DATABASES variable for DJANGO_HOST and all others
# Define CACHES variable for DJANGO_HOST production and all other hosts
if DJANGO_HOST == "production":
    DEBUG = False
    STATIC_URL = '/static/'

    # Use mysql for live host
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'OPTIONS': {
                'read_default_file': 'my.cnf',
            },
        }
    }

    # Set cache
    CACHES = {
        'default': {
            'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
            'LOCATION': '127.0.0.1:11211',
            'TIMEOUT':'1800',
        }
    }
    CACHE_MIDDLEWARE_SECONDS = 1800
else:
    DEBUG = True
    STATIC_URL = '/static/'

    # Use sqlite for non live host
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        }
    }
```

Figure 29: SRS settings.py Code Snippet.

Structure of Django Files

Once all the installation and pull-from-GitHub steps are done, the directories of the SRS project should have a structure similar to the one shown in *Figure 30*. We called our main directory *djangogirls*, just as the Django Girls Tutorial prompts us to; you could have chosen any other name. If you followed the aforementioned steps correctly, you should see the nested directories: *mysite*, *myvenv* and *SRS*. There are also important files, such as *manage.py*.

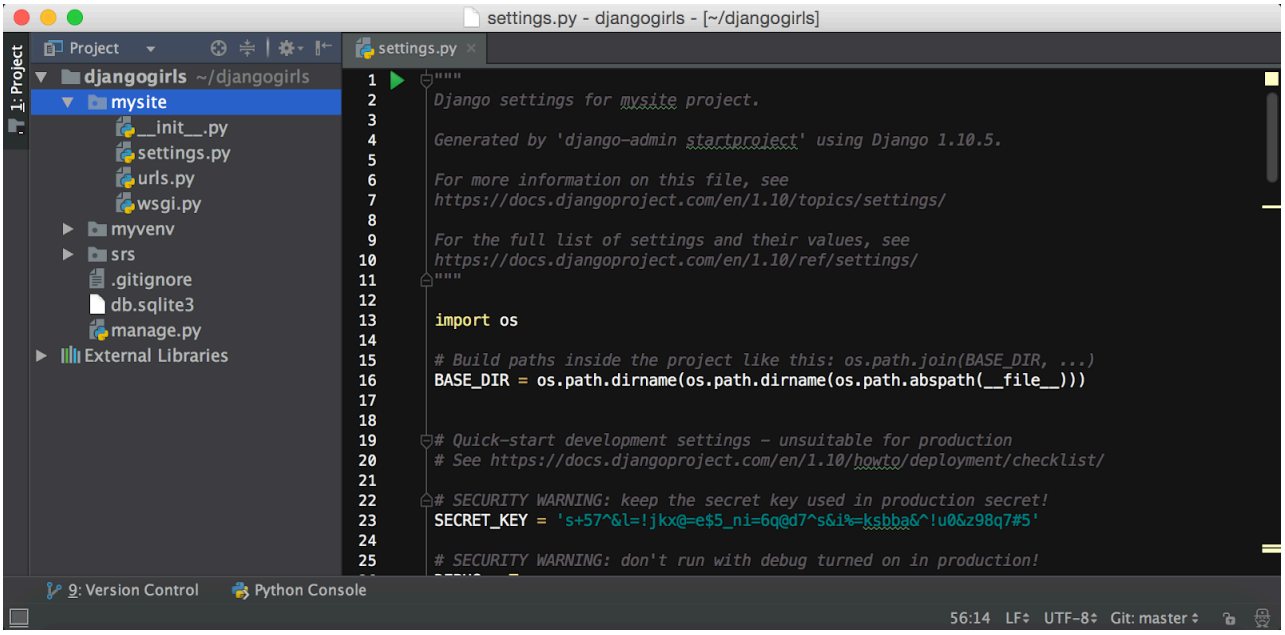

Figure 30: Directory Structure of the SRS Project.

The directory *mysite* contains configuration files for the SRS website. File *settings.py* is of particular relevance, since most of the settings are done within this file.

The directory *myvenv* contains packages and libraries that were added after the installation and configuration steps. It contains the information for the virtual environment to work along the Django framework. This directory does not need to be modified by developers.

The directory *SRS* is where the majority of the source code lives. It contains several nested subdirectories (*migrations*, *static*, *templates*, and *templatetags*) and important files, as shown in *Figure 30*.

- The files within *migrations* are just files that are generated by Django before applying the changes made to the models into the local SQLite database; these files do not need to be changed by the developers.
- The "*static"* subdirectory, however, does contain css and javascript files, icons and images that will probably be modified in future development efforts. This is where the developers want to add or modify images, the website's style or behaviors using javascript code.

- The subdirectory "*templates*" has a nested directory called *SRS* that contains all html files used in the SRS website. If developers want to add or change any html file, this is the place to do so.
- The subdirectory "*templatetags*" contains a couple of python files. One of these files contains the filters applied in SRS.

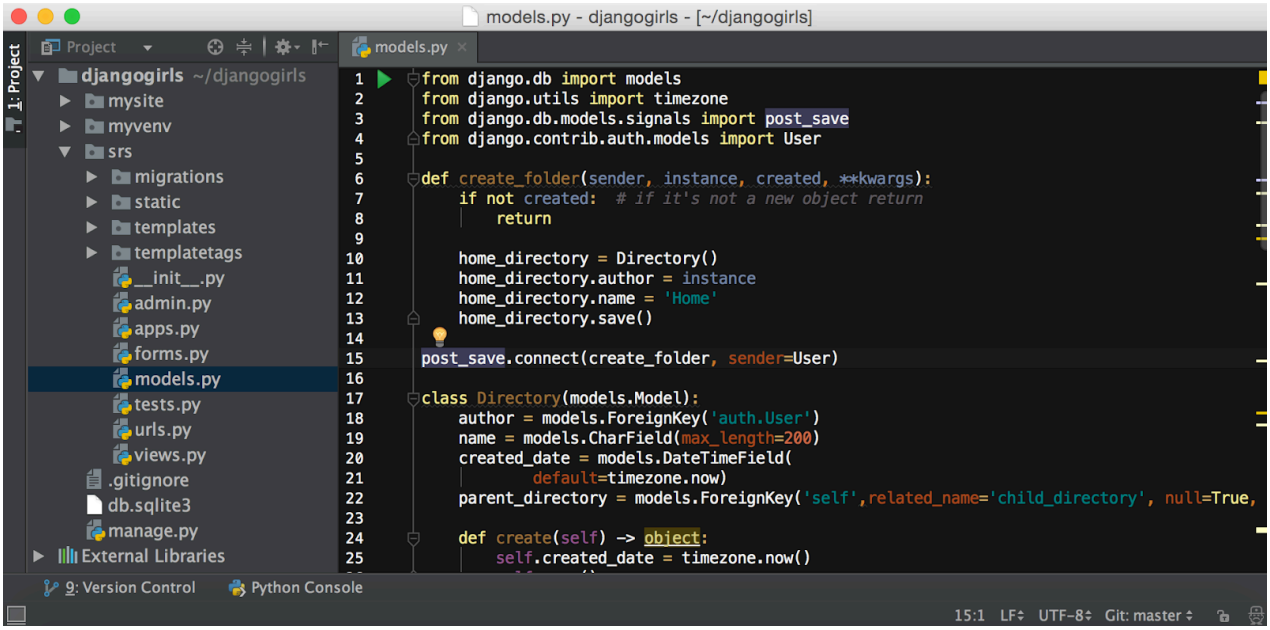

Figure 31: Directory Structure of the SRS Directory.

As mentioned earlier, directory *SRS* nested within directory *djangogirls* contains important files (see *Figure 31*).

- The Django framework facilitates the communication between python code and the local SQLite database. In other words, if developers need to create a new table in the database, they only need to create a model of the database object in *models.py* and apply the changes later on using certain commands in the command prompt (python manage.py makemigrations and then, python manage.py migrate).
- Whenever a model is created in the SRS website, the model needs to be register in *admin.py*.
- SRS uses forms, a feature implemented in the Django framework that eases developers' work. Every time a developer creates a new form, he or she should create it in *forms.py*.

- File *urls.py* contains all urls that the SRS web application uses to render or redirect the html pages.
- File *views.py* contains all the existing views in the source code. A view will define the behavior of a page.

The directories and files aforementioned are the more relevant ones to any development purposes. Nevertheless, developers are welcome and highly encouraged to view all files and directories to have a better understanding of the SRS web application's behavior behind scenes.

Installation Process Explained

In order to set up the environment for SRS, you need to follow certain main steps: install Python, setup the virtual environment for Django, and install Django, a code editor and Git. In addition, you need to create a GitHub account (if you do not have one already) and be able to log into the existing PythonAnywhere account for SRS. Finally, you need to clone the existing source code into your machine and apply the changes made to the models to your local SQLite database before start collaborating in this project.

The instructions listed within the following steps target the Linux operating system. If you have a different operating system, that is OK - just read the Django Girls Tutorial and find your instructions' format [1].

Install Python

Django is written in Python; therefore, we need to have Python installed. The Django Girls Tutorial suggests to install Python 3.5 or later [1]. First, check whether you have an installation of Python 3.5.

```
python3 --version
```

If you have a version of Python 3.5 or later installed, go to the next step. Otherwise, you need to install it.

```
sudo apt-get install python3.5
```

Just to be sure, check which version of Python you just installed.

```
python3 --version
```

Set up Virtual Environment

Once Python 3.5 or later has been installed, you can proceed to install the virtual environment. This step is not necessary, but highly recommended. The virtual environment will isolate your Python/Django setup on a per-project basis; as a result, any changes you make to one website will not affect any other websites you are also developing [1]. In order to create the virtual environment, you need to create or select a directory in which SRS project is going to live. In our case, we chose a directory called *djangogirls*; you can name it as you like.

```
mkdir djangogirls

cd djangogirls
```

Then, we create a virtual environment called *myvenv*. You can again choose any name you like for the virtual environment, but it is highly recommended that you use lowercase and no spaces in your name [1].

```
python3 -m venv myvenv
```

If you encounter any errors, please refer to the Django Girls Tutorial [1]. Otherwise, a folder called *myvenv* has been created. The next step is to start the virtual environment.

```
source myvenv/bin/activate
```

After running this line of code, the prefix in your console's prompt will show (myvenv).

Install Django

Once the virtual environment is running, the next logical step is to install Django. In order to install Django, though, we need to check for the latest upgrade of pip, the program being used to install Django.

pip install --upgrade pip

Afterwards, just install Django using the latest version of pip.

pip install django~=1.10.0

Install Dependencies

There are also several dependencies that must be installed to run the current version of the project:

pip install -r requirements.txt

In addition, the system requires libav so that it can generate thumbnails when videos are added. You can install it by running the following command on Linux:

sudo apt-get install libav-tools

Install a Code Editor

It is highly recommended to use a code editor for your development efforts, such as PyCharm, Gedit, Sublime Text and Atom. One of the reason to use a code editor is that code needs to be plain text, and programs like Word or Textedit produce rich text (with fonts and formatting) which will produce problems at compilation. In addition, most code editors will highlight code or detect problems before you do. In our case, we preferred to use PyCharm, but you are welcome to use the one you feel the most comfortable with.

Install Git

Git is a type of version control system used by many developers. It tracks the changes made to files over time, and it is very useful for development efforts. If you have not done it yet, install Git.

sudo apt-get install git

Create a GitHub Account

If you do not have a GitHub account yet, we recommend you create one as soon as possible by going to GitHub.com. GitHub allows you to collaborate with other developers in real time. It is an extremely useful platform for coders worldwide.

The latest code of SRS is located at https://github.com/lisbecg/srs.

Clone SRS Project from GitHub

Once you have installed git in your virtual environment, the next step is to clone the SRS project to your folder *djangogirls*. Make sure that you are located within the folder *djangogirls*.

git clone https://github.com/lisbecg/srs.git

At this point, you should be able to have the latest source code in your computer.

Create Tables for Models in Your Database

Once you have the latest version of the SRS project's code in your computer, you need to update the SQLite database that Django framework has by default. In order to do this, you need to run two commands. Before running these commands, make sure that you started the virtual environment *myvenv* and that you are positioned within the *djangogirls* directory.

python manage.py makemigrations srs

python manage.py migrate srs

The first command lets Django know that some changes have been made to your models (objects that represent the tables in your SQLite database) and created a migration file. The second command applies this migration file to the database.

At this point, you should be ready to start collaborating towards the completion of the SRS project.

Creating an SRS Executable

The best way to produce an executable for the most current version of SRS is to follow the steps given the Django Girls Tutorial because it will show you the steps you need to follow depending on which operating system you are using [1].

Get Deployment Credential

New developers need to contact the UF's CISE systems administrator, who will upgrade he or she CISE credentials. Dayron E. Acosta is the CISE's administrator who created the server to deploy the SRS application, and he is willing to help new developers. His email is dayron.acosta@ufl.edu.

Deployment

When a feature is done and ready for production, a developer has to deploy it in the "srs-prod" server of the UF's CISE department where the SRS application lives (production environment). If the new feature implementation required modification in one of the forms of models.py file, the developer has to synchronize the MySQL database in the production environment with the SQLite database in the development environment. This synchronization is accomplished with the command "python manage.py migrate"

Bugs and Problems Encountered

Our team is not aware of any bugs that have not been resolved.

References

[1] "*Django Girls Tutorial,"* tutorial.djangogirls.org [Online]. Available: https://tutorial.djangogirls.org/en/. [Accessed: Apr. 25, 2017].