Software Design Specification

For

Disability and Accessibility Services

Alternative Text Format Database

Submitted by

John Hellrung

Version 1

Document template copyright (c) 2005, Gregory W. Hislop. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation.

Table of Contents

[Table of Contributions 3](#_Toc458804680)

[Revision History 3](#_Toc458804681)

[1. Introduction 4](#_Toc458804682)

[1.1. Purpose 4](#_Toc458804683)

[1.2. Scope 4](#_Toc458804684)

[1.3. Definitions, Acronyms, and Abbreviations 6](#_Toc458804685)

[1.4. References 6](#_Toc458804686)

[1.5. Overview 7](#_Toc458804687)

[2. The Alternate Format Text Conversion Process 7](#_Toc458804688)

[1.1. Current Process 7](#_Toc458804689)

[3. Hardware/Software Requirements 8](#_Toc458804690)

[4. Use Case Diagram 9](#_Toc458804691)

[5. System Diagram 10](#_Toc458804692)

[6. Architectural Description (see section 6.2.1.2) 10](#_Toc458804693)

[6.1. Overview 10](#_Toc458804694)

[6.2. Models 10](#_Toc458804695)

[6.3. Controllers 12](#_Toc458804696)

[6.4. Views 16](#_Toc458804697)

[6.5. Authorization & Authentication 17](#_Toc458804698)

[7. Interface Description 18](#_Toc458804699)

[7.1. User Interface 18](#_Toc458804700)

[7.2. Main Interface 19](#_Toc458804701)

[7.3. Admin Interface 23](#_Toc458804702)

[8. Detailed Design 27](#_Toc458804703)

[8.1. User 27](#_Toc458804704)

[8.2. Role 28](#_Toc458804705)

[8.3. UserRole 29](#_Toc458804706)

[8.4. File 29](#_Toc458804707)

[8.5. Post 31](#_Toc458804708)

[8.6. ER-Diagram 31](#_Toc458804709)

# Table of Contributions

The table below identifies contributors to various sections of this document.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Section | Writing | Editing |
|  | Entire Document | John | John |
| 1 | Introduction | John | John |
| 2 | The Alternate Format Text Conversion Process | John | John |
| 3 | Hardware/ Software Requirements | John | John |
| 4 | Use Case Diagram | John | John |
| 5 | System Diagram | John | John |
| 6 | Architectural Description | John | John |
| 7 | Interface Description | John | John |
| 8 | Detailed Design | John | John |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 08/11/2016 | 1.0 | Document created | John Hellrung |

## Introduction

### Purpose

To provide a database with intergraded systems that improves the Alternative Text Format Conversion process that the Disability and Accessibility Services need. The application will provide support and simplify the Conversion process for Disability and Accessibility team decreasing the amount of time needed in the Conversion process.

### Scope

This application is for the Berea College Disability and Accessibility Services. The Application will have authentication with only allowing these three user roles: Admin, Labor Students and Student. Only a Disability and Accessibility staff or student can have access to the Admin role. The Admin role will have full access to all features and pages of the database and its intergrade systems. The Labor Students on the other hand will have full access except for management features or pages for example accounts management. The Labor student will be able to make file uploads and make small edits to files in the database. Then there is the Student role, which can only download and view the index page after the authentication process. The Database will only allow hold these flowing data types of files to be downloaded: Doc, Docx, PDF, XML. Based on future updates the allowed download list can be expand if needed.

#### Associate Task/ Stakeholder

C:\Users\hellrungj\Downloads\Data Access.png

This database will serve the following users and tasks:

DAS Admin

* Upload textbooks and materials needed for accessible format conversion projects
* Able to report on active conversion projects
* Able to approve pending conversion projects from DAS Labor Students
* Able to accept conversion projects requests from Students

DAS Labor Students

* Able to upload textbooks and materials for an active conversion project
* Able to submit conversion projects to DAS Admin

Students with Print Disabilities

* Obtain their own conversion projects
* Able to submit a request for a conversion project

### Definitions, Acronyms, and Abbreviations

* + 1. Design view: A subset of design entity attribute information that is specifically suited to the needs of a software project activity.
    2. Software design description (SDD): A representation of a software system created to facilitate analysis, planning, implementation, and decision making. A blueprint or model of the software system. The SDD is used as the primary medium for communicating software design information.
    3. Create, Read, Update, Delete (CRUD): This refers to the common operations of creating (in this case, creating and posting), displaying, updating and deleting an entity (e.g. a question post).
    4. Model View Controller (MVC). A common design pattern in web applications where program control is divided into three sections:
  + Model: A database API for a single database table. Allows for full interaction with records without needing to type SQL.
  + View: A text file that can be rendered into HTML with appropriate instance variables as determined by the calling controller action.
  + Controller: Receives incoming HTTP requests, instantiates appropriate instance variables based on request parameters and renders the appropriate HTTP response,
    1. Architectural Description: a set of practices for expressing, communicating and analyzing software.
    2. Detail Design: A description of top-down design in an application.
    3. Interface Description: A description of the graphical interface.
    4. Authentication: The act of confirming the truth about an attribute of data.
    5. Authorization: The function of specifying access rights to an application.
    6. Intergrade Systems: A defined process of bringing together subsystems into one system and ensuring that the subsystems function together as one system.
    7. Print Disability: An inability to read printed texts effectively.
    8. Repository: A storage location from which software packages may be retrieved and installed on a computer.
    9. ER-Diagram: Describes inter-related things of interest in a specific domain of knowledge.

### References

* + 1. Project Repository: <https://github.com/Hellrungj/DAS-Alt.-Format-Text-Database>
    2. Flask: <http://flask.pocoo.org/>
    3. PeeWee: <http://docs.peewee-orm.com/en/latest/>
    4. Flask-PeeWee: <http://docs.peewee-orm.com/projects/flask-peewee/en/latest/>
    5. Flask-Admin: <https://flask-admin.readthedocs.org/en/latest/>
    6. Flask-Security: <https://pythonhosted.org/Flask-Security/>
    7. SQLite database: <https://sqlite.org/>
    8. The Jinja Documentation: <http://jinja.pocoo.org/docs/dev/>
    9. DataTables: <https://datatables.net/>
    10. The Configure Documentation: [http://configure.readthedocs.io/en/latest/#](http://configure.readthedocs.io/en/latest/)
    11. Section 508 standards: <http://www.section508.gov/summary-section508-standards>
    12. Web Content Accessibility Guidelines (WCAG) 2.0: <https://www.w3.org/TR/WCAG20/>
    13. IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions: <https://standards.ieee.org/develop/wg/1016_WG.html>

### Overview

This document, will address the Architectural Description, Interface Description, and Detail Design of the Disability and Accessibility Services Alternative Text Format Database. The Architectural Description will address the overarching implementation of the application. The Interface section will cover the structure of how the user interacts with the application and how that will affect the user. Finally, the detail design section will describe how the source is mapped and translated throughout the project.

## The Alternate Format Text Conversion Process

#### Description

The alternate format text conversion process is a process for student who has a print disability where then DAS takes a Document that they requested like a PDF or book and turns it into a format that is accessibility

### Current Process

#### Description

The process starts with DAS sending an Email asking the Student in Question if they would like alternate text of one of these courses. If the student’s emails back then DAS would then go into Banner and look up the student’s course information. If DAS finds the Course information then DAS looks up the book information through the bookstore use the student course information. Then if DAS find all the information about each texts, DAS has to take in account of the student types of disability. Then after that, we then after find where the text is available. When all said and done the file is either, a Word or PDF file which is saved on a staff member’s computer.

#### Flow Chart

The flow chart is too big to include in this document. Please follow this link: [Flow-Chart-Full](https://bereacollege-my.sharepoint.com/personal/hellrungj_berea_edu/_layouts/15/guestaccess.aspx?guestaccesstoken=6pAPMQ4T6OPyW%2bX8zM7cjcz4bN2RsauPXEFklYfIZKc%3d&docid=2_06acb3a6daacf4a829078a525ac64884e&rev=1)

## Hardware/Software Requirements

These are the Hardware/Software requirements for running this software.

* Ubuntu 16.04
* Apache 2 Web Server
* WSGI
* Flash Web Framework
* Python 2.7

For more information about this application requirements please check out the application’s repository. (See Section 1.4.1)

## Use Case Diagram

C:\Users\hellrungj\Downloads\Use Case.png

## C:\Users\hellrungj\Downloads\Flask System Diagram.pngSystem Diagram

## Architectural Description (see section 6.2.1.2)

### Overview

The Alternative Text Format Database is a Flask application and as implements a Model View Controller (MVC) framework design. The following will going to detail about each part of the MVC framework.

### Models

The Application using Pee Wee model for Flask. This provide a database API for their respective tables:

#### User Table

|  |  |
| --- | --- |
| **Class**: User | Interacts with: N/A |
| Methods   * username * email * password * active * confirmed\_at * notes | Attributes   * TextField() * TextField() * TextField() * BooleanField(default=True) * DateTimeField(Null=True) * TextField(null=True) |

#### Role Table

|  |  |
| --- | --- |
| **Class**: Role | Interacts with: N/A |
| Methods   * name * description | Attributes   * CharField(Unique=True) * TextField(Null=True) |

#### User Roles

|  |  |
| --- | --- |
| **Class**: UserRoles | Interacts with: Role and User |
| Methods   * user * role * name * description | Attributes   * ForeignKeyField(User, related\_name='roles') * ForeignKeyField(Role, related\_name='users') * property(lambda self: self.role.name) * property(lambda self: self.role.description) |

#### File Table

|  |  |
| --- | --- |
| **Class**: File | Interacts with: User |
| Methods   * title * author * edition * size * filename * file\_type * created\_at * last\_modified * last\_ modified\_by * file\_path * hidden | Attributes   * TextField() * TextField() * TextField() * IntergerField() * TextField() * TextField() * DateTimeField() * DateTimeField() * ForeignKeyField(User) * TextField() * IntergerField() |

#### Post Table

|  |  |
| --- | --- |
| **Class**: Post | Interacts with: User |
| Methods   * title * text * date * user | Attributes   * CharField(max\_length=120) * TextField(null=False) * DateTimeField() * ForeignKeyField(User) |

### Controllers

The Controller classes handle incoming requests, instantiate appropriate instance variables, and render appropriate views.

#### Admin.py

|  |  |
| --- | --- |
| **Name**: AdminController | **Imports**: allImports \*, switch, flash, url\_for, admin, form, ModelView, FileAdmin, FileUploadField, BaseView, expose, Security, roles\_accepted, login\_required, current\_user, rules, and AdminIndexView |
| Methods: | Description: |
| * **Class**:MyHomeView * **Parameters**: AdminIndexView * **Route**: admin/index | * Interacts with AdminIndexView which overwrites Flask\_Admin defaults Index View. Checks if the is “Admin” and if not redirect to their previous page. |
| * **Class:** UserAdmin * **Parameters:** ModelView * **Route**: admin/user | * Gets a database quarry and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. |
| * **Class**: RoleAdmin * **Parameters:** ModelView * **Route**: admin/role | * Gets a database quarry and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. |
| * **Class**: UserRoleAdmin * **Parameters:** ModelView * **Route**: admin/userrole | * Gets a database quarry and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. |
| * **Class**: PostAdmin * **Parameters:** ModelView * **Route**: admin/post | * Gets a database quarry and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. |
| * **Class**: FileDataAdmin * **Parameters:** ModelView * **Route**: admin/files | * Gets a database quarry and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. |
| * **Class**: MyFileAdmin * **Parameters:** FileAdmin * **Route**: admin/file | * First File Admin reads the system for files and the result of that quarry get put into a table that can be searched, sorted, and filtered. It also implement CRUD features for the tables users. Then The class overwrites File\_Admin to enable and modify the File\_Admin class. |
| * **Class**: AdminUpload * **Parameters:** BaseView * **Route**: admin/upload | * Logs out the admin from the session and sends them to the login. |
| * **Class**: StundentView * **Parameters:** BaseView * **Route**: admin/logout | * This class sends the admin to student main part of the application to where the admin can access the student views. This help with debugging. |

#### Auth.py

|  |  |
| --- | --- |
| **Name**: AuthController | **Imports**: allImports \*, switch, flash, Security, login\_required |
| Methods | Attributes |
| * **Class**: Login * **Parameters:** * **Route**: /login | * Logs in the user into the session, if the validation equals true, send a flash message, and then send the user to index. If not user has not validation equal true then the user is resend back to login. |
| * **Class**: Logout * **Parameters:** * **Route**: /logout | * Logs out the user form the session and sends them to the login. |

#### Edit.py

|  |  |
| --- | --- |
| **Class**: EditController | **Imports**: allImports \*, switch, send\_file, session, current\_user, login\_required, roles\_accepted, os, datatime |
| Methods | Attributes |
| * **Class**: file * **Parameters:** * **Route**: /file | * Gets a database quarry and the result of that quarry get put into a table |
| * **Class**: edit * **Parameters:** * **Route**: /edit | * Gets a database quarry and the result of that quarry get put into a table and then allows that user to update the edits to a file. |
| * **Class**: delete * **Parameters:** * **Route**: /delete | * This set the file attribute hidden to 1 and update the database that way to file appear to delete but are not so in the event of unexpected file deletion the Admin can restore the file in the database. |

#### Main.py

|  |  |
| --- | --- |
| **Class**: MainController | **Imports**: allImports \*, switch, session, flash, abort, send\_file, redirect, login\_required, current\_user |
| Methods | Attributes |
| * **Class**: Home * **Parameters:** * **Route**: / | * This class checks the user’s role and sends them to their view. For example, if the user is the Admin, they will send to the Admin. |
| * **Class**: Index * **Parameters:** * **Route**: /index | * Gets a database quarry and the result of that quarry get put into a table. |
| * **Class**: filedownload * **Parameters:** * **Route**: /download | * This class searches for the file system and then send that file to the browser for the user to download. |

#### Upload.py

|  |  |
| --- | --- |
| **Class**: UploadController | **Imports**: allImports \*, switch, session, flash, abort, send\_file, redirect, login\_required, current\_user |
| Methods | Attributes |
| * **Class**: uploadfrom * **Parameters:** * **Route**: /upload | * This class just send the user to /upload.html |
| * **Class**: uploading * **Parameters:** * **Route**: /uploading | * This class uses drop.js to collect the file submit by the user and along with any proved by the form. The class then updates the database and updates the filesystem with the data. |

### Views

#### Index View

The index view shows all entries of sorted by A – Z. On this page the user can search and sort use the table thanks to datatables.js, which is a JavaScript open source project that makes it easy to create a data tables with sorts and search feature.

The index page shows the user all current files in the database but it does not show all the data on whose files. The data shown by a table is:

* Title
* Author
* Edition
* Size

#### Upload View

The upload view allows the user to download documents but currently only doc, docx, XML and PDF thanks to DropZone.js.

The upload view is a form that only the user with the follow can use:

* Admin
* Labor Student

The form asks the use for the following:

* File
* Title
* Author
* Edition

#### Edit View

The edit view allows the user to edit the title, edition and author. In addition, on the edit page a delete button hides the document from viewing and send the admin a message. The reason this is the case to ensure that no file is delete without admin knowing about it. These features are only available to the Admins and Labor Students.

The edit view is a form that only the user with the follow can use:

* Admin
* Labor Student

The form asks the use for the following:

* Title
* Author
* Edition

### Authorization & Authentication

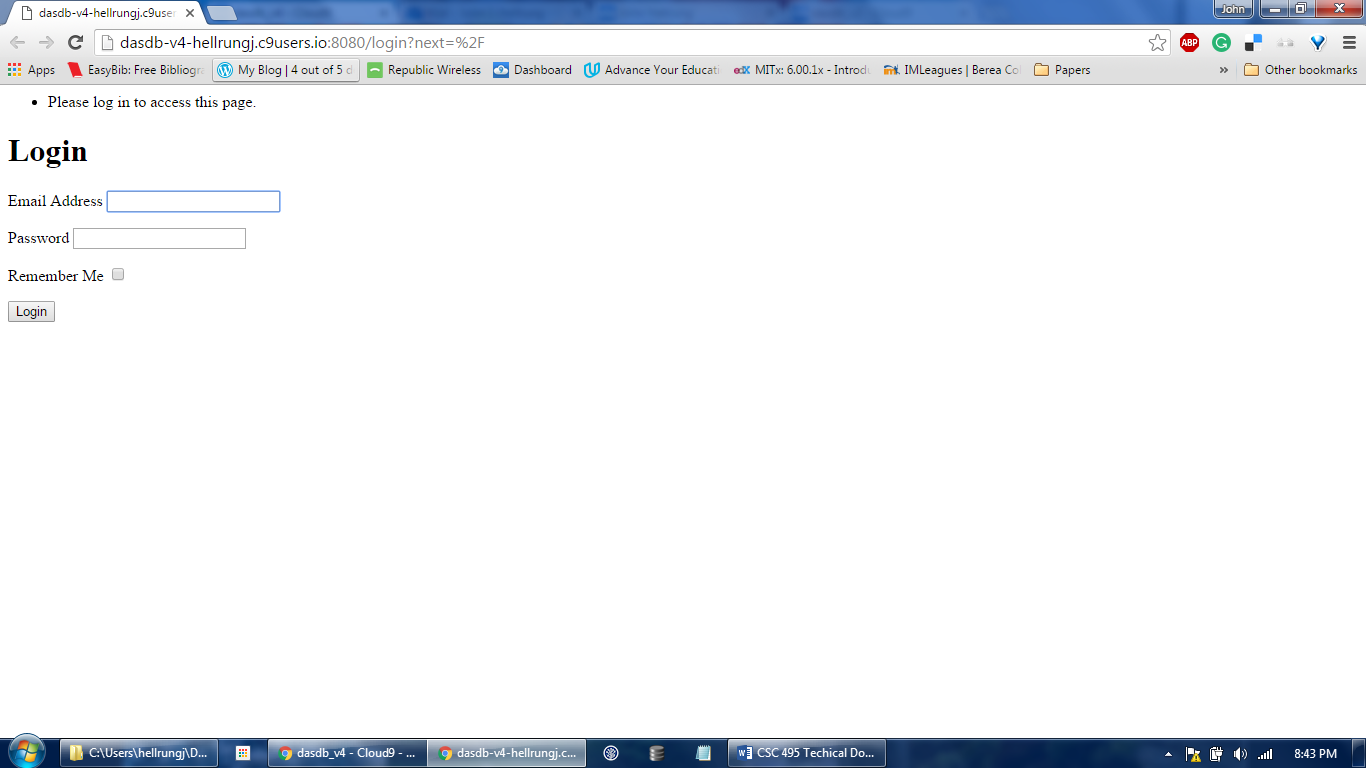
The Alternative Text Format Database implements authorization by using Flask Security. Flask Security allows:

* Session based authentication
* Role management
* Password encryption
* Basic HTTP authentication
* Token based authentication
* Token based account activation (optional)
* Token based password recovery / resetting (optional)
* User registration (optional)
* Login tracking (optional)
* JSON/Ajax Support

These features are possible by using these Flask extensions:

* Flask-Login
* Flask-Mail
* Flask-Principal
* Flask-Script
* Flask-WTF
* itsdangerous
* passlib

Using flask security I was upload to secure my database and now the user have to login and there is user varication all each view. Resulting in this page if the user is not login.



As you see the page has you login with your email and password before entering. The database will stop of you try to access a part of the database with login and will also stop user who do not have access to that page. For example, if a student try to access to admin page by typing /admin in the Url bar, they will be redirected back stopping the student form accessing the page.

## Interface Description

### User Interface

The user interface is split into two major sections:

* Main Interface
* Admin Interface

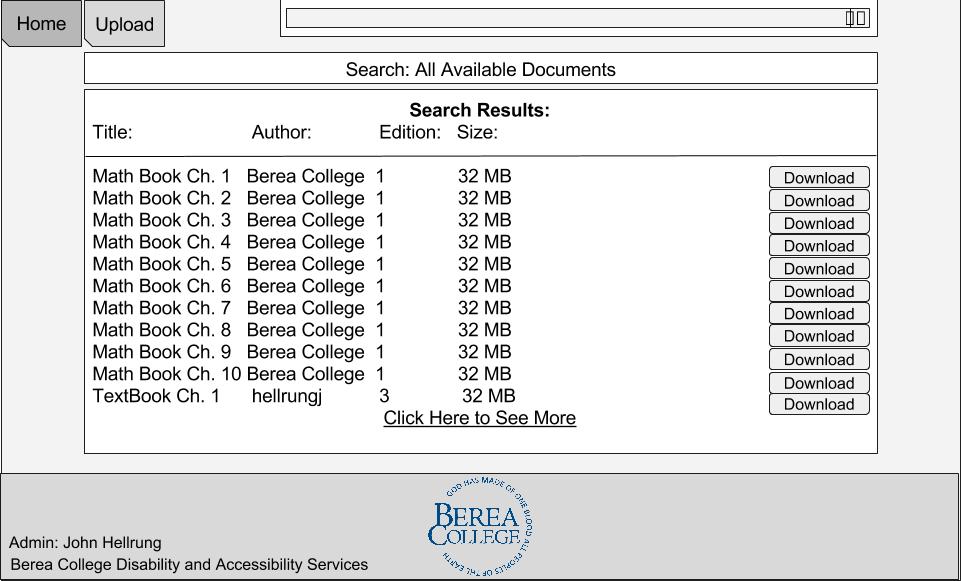
This to make easier to implement user roles between the Admin and every role in the database.

The user interface consists of a set of pages that each allow the user to do different tasks related to posting, editing, deleting and viewing a question. The pages that make up the Q&A forum are the "main", "New/Edit" and "View/Respond" pages. Each page will contain buttons and/or text fields that will allow the user to perform some CRUD action on a question. The user will right click once on buttons and simply type in text for the text field.

### Main Interface

#### Index View

##### An Early Version:



As you can see the index page has a table layout with a search and two tabs. One for Home and the other for uploads.

The early table displays the following data:

Title:

Author:

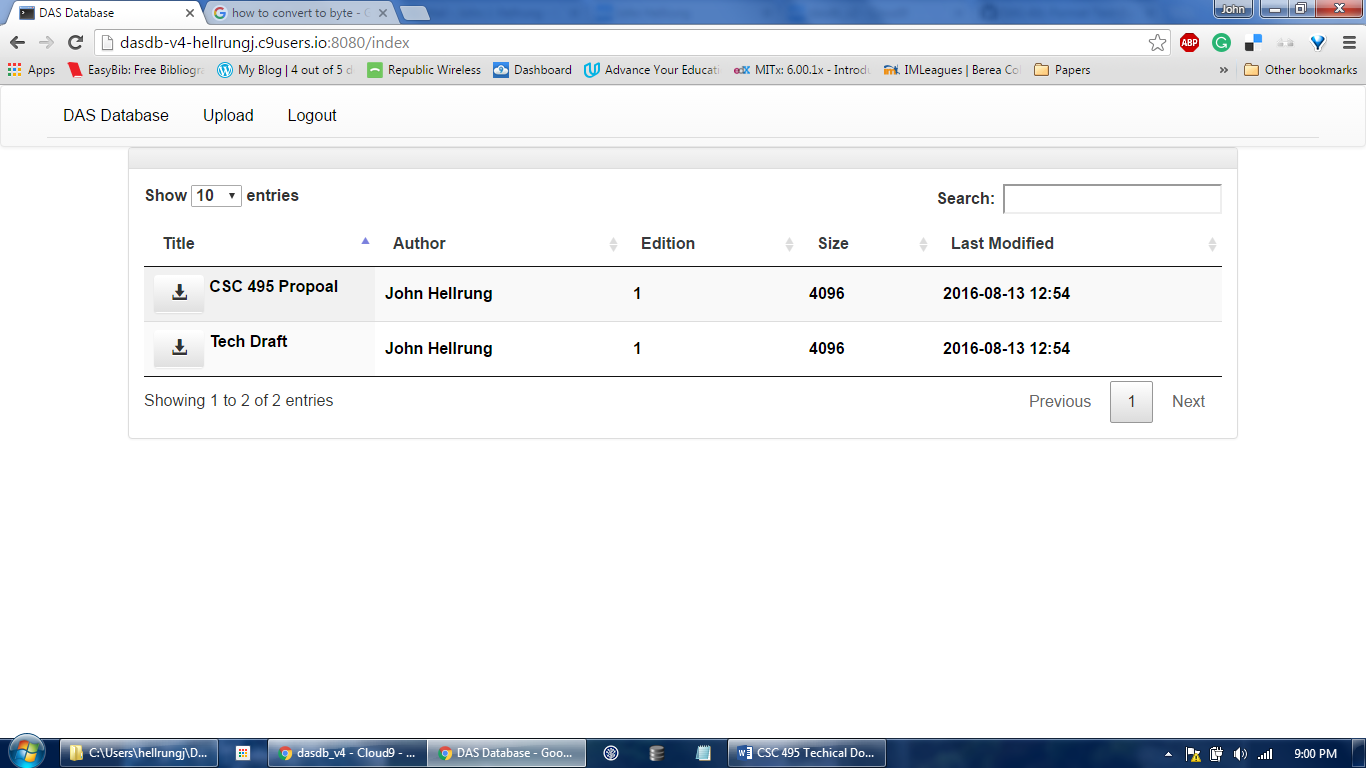
Edition:

Size:

This would change in later versions.

The search bar would search for title of the file in the database and display all file with that title. On the right side you have download button for each file in the table.

##### Current Version:



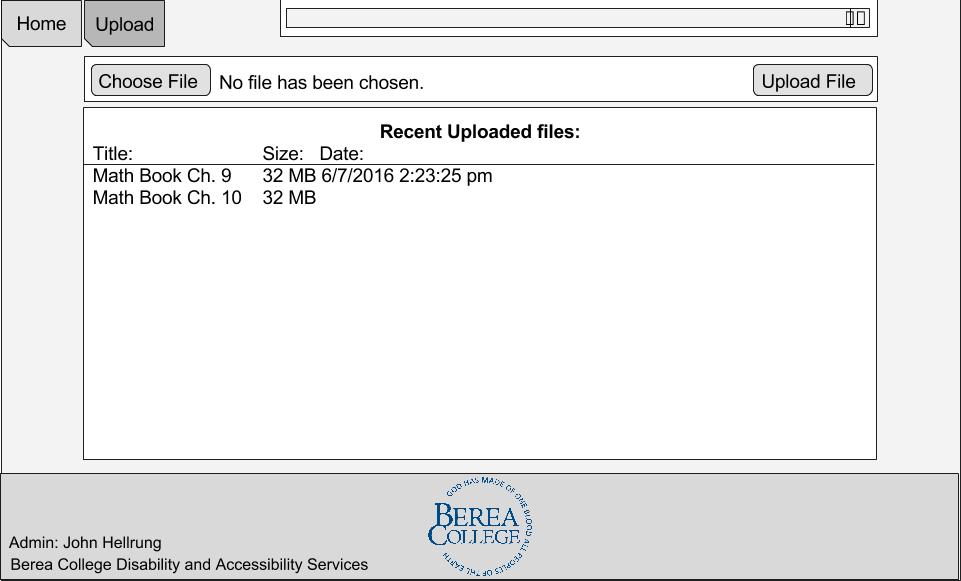
As you can see the current version look about the same expect now it has a logout tab and a search that is not located in the Navibar.

This is because DataTables a great JavaScript application does this for me and plus it gives filtering and pages of the table which is nice.

Then the next part is the download is now on the left rather than the right and the table displays a download button with a Glyphicons from Bootstrap rather than text saying download. The table also displays the last modified time. These changes were made because we at DAS through it would easier for us to know the last known date modified. The download is still access thank to alt text with the glyphicons.

#### Upload View

##### An Early Version:



As you can see the upload page also has table layout with a search and two tabs. One for Home and the other for uploads. The page also has a upload button with a choose file button that would upload the file into the database.

This early table displays the following data:

Title:

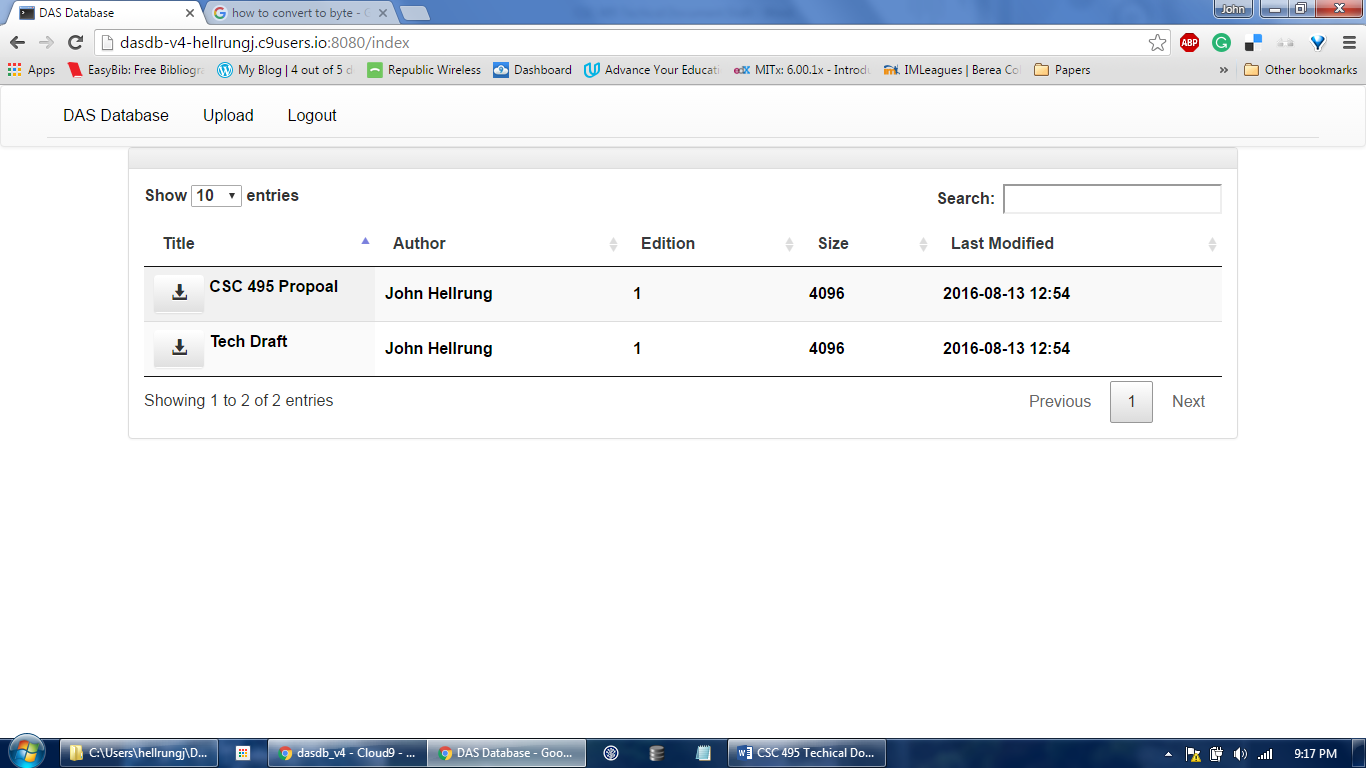
Author:

Size:

Date:

This would change in later versions.

##### Current Version:



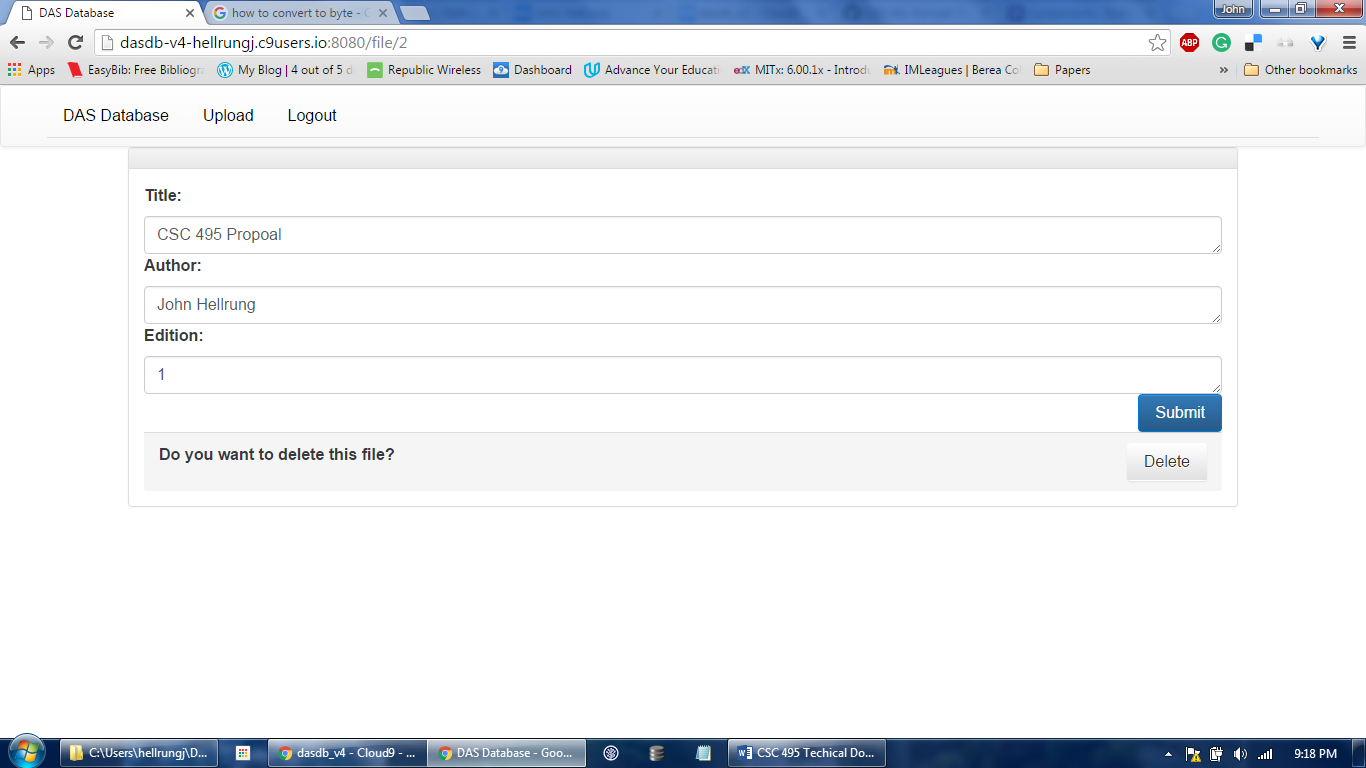
The current view now asks the uploader for the file, title, author, and edition. This discussion was made because having to table display simple data was not useful and redundant. Now the upload view act as form that is effective for are needs. This is because of the help of Drop another JavaScript application that makes file upload a lot easier.

#### Edit View

##### An Early Version:

Now, the edit page was not planed and through about at first but came later in development after realize that it would a great idea for someone to edit the files rather than just uploading again.

##### Current Version:



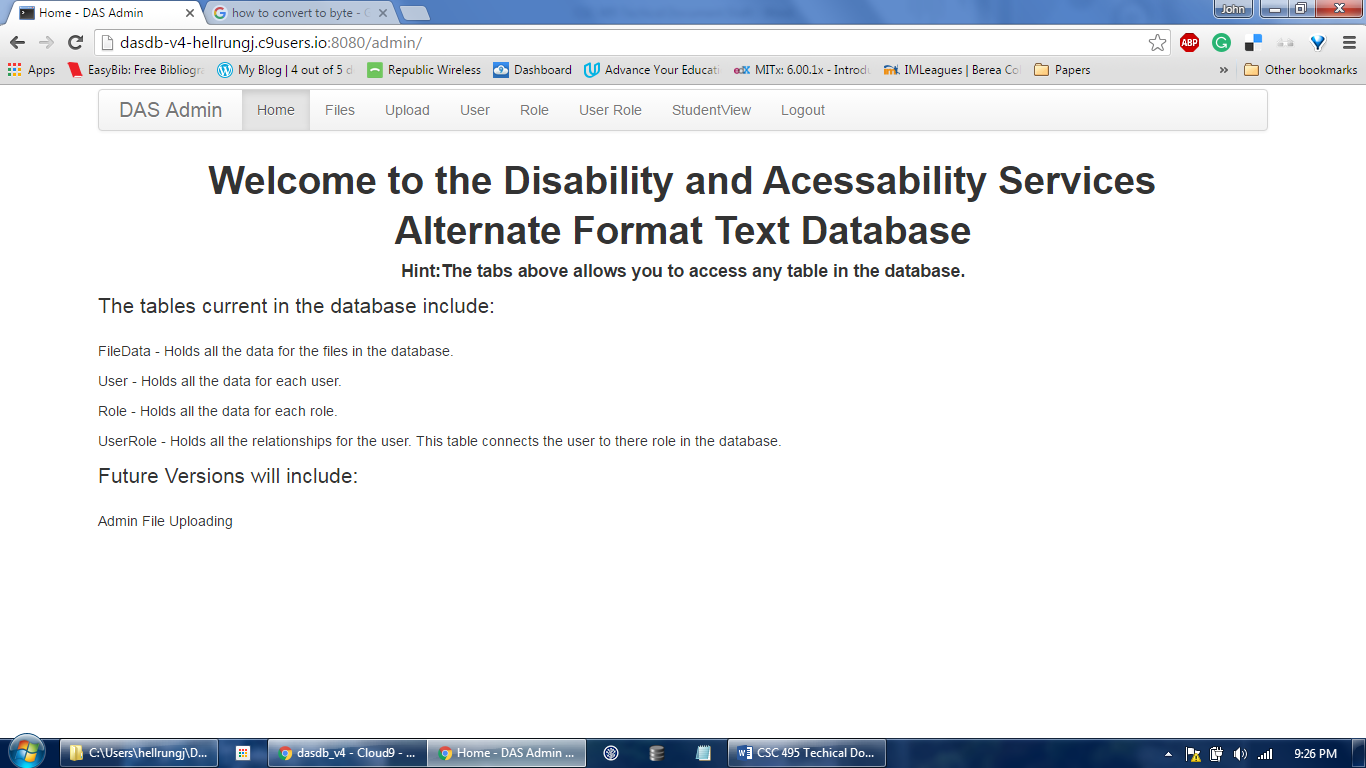
Now, as you see edit view gives current title, author, and edition within textboxes. Below, the textboxes is the submit button that submit the changes to the file data in the database. At the very bottom is a button that sets the file data attitude “hidden” to ‘True” effective deleting the file from the student table but not the database that can only be done by admin. When clicked both of these buttons send the user to index page.

### Admin Interface

The Admin Interface is Flask-Admin a Micro Framework that allows an admin interface with an existing data model. This is very helpful because it allow the Admin to use any of their CRUD ability in the database.

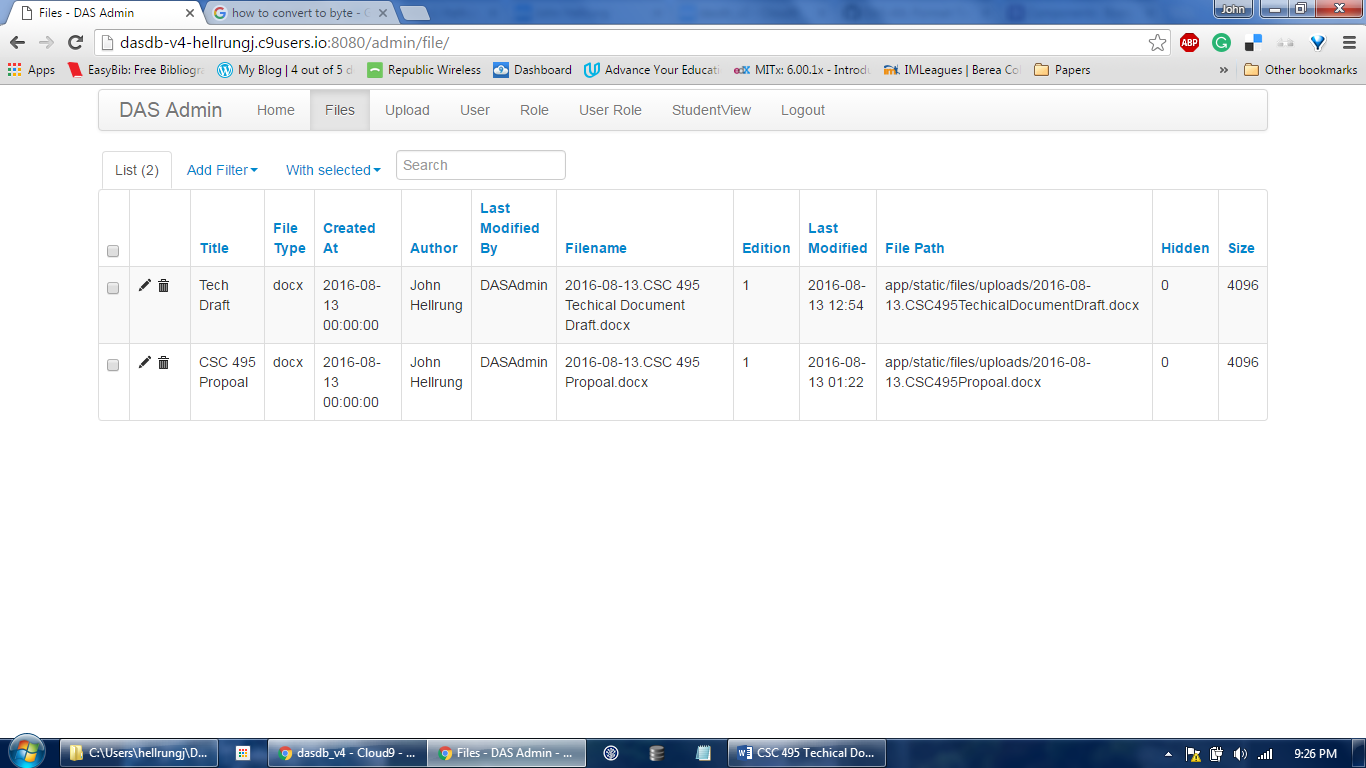
The Admin Interface has six parts:

#### Index View



Current the index view will gives a brief discretion on how to use the database and what will be included in future updates. The future of this page is act dashboard with updates of notification of requests form students.

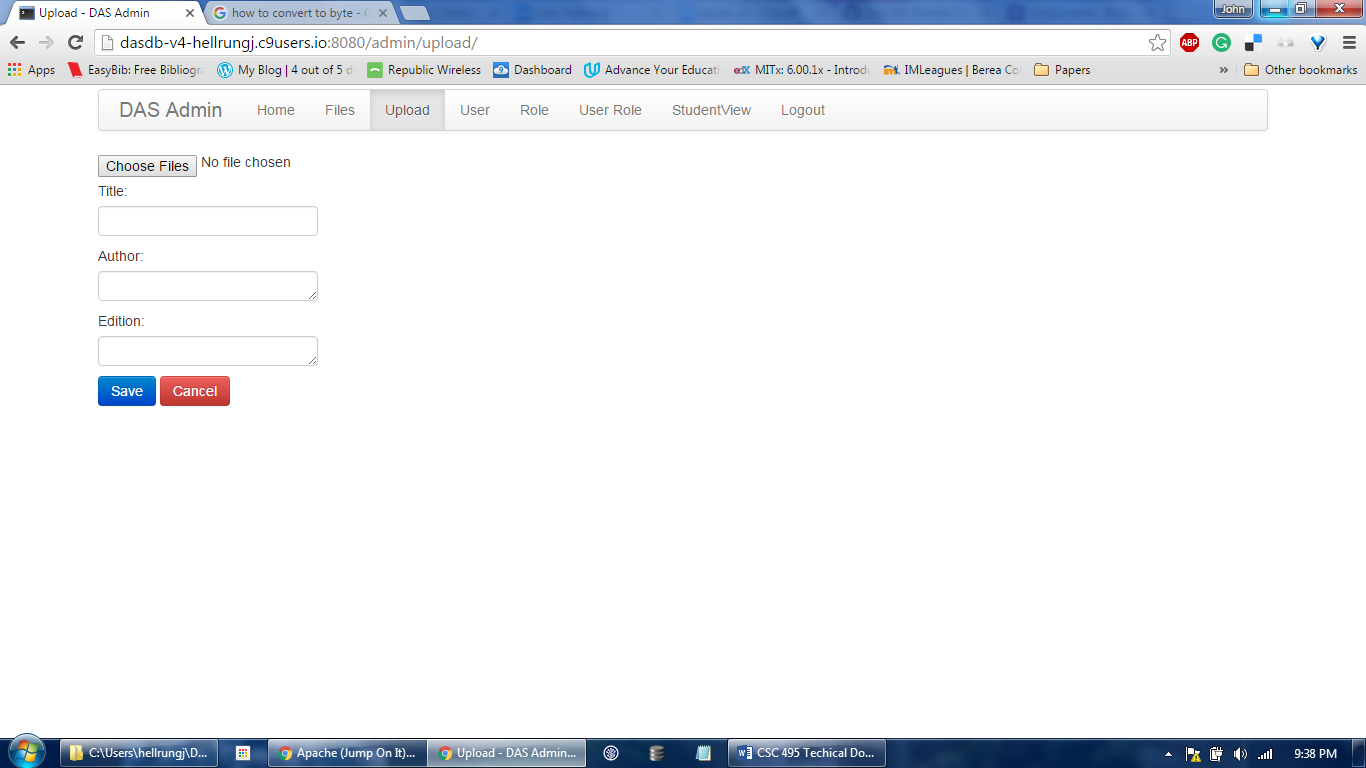
#### File View



The Files View includes a table that holds all the data files in the database. Here the admin can edit and delete file if needed. This table can sort, filter, select and search but it cannot create a file yet and it only deletes the file data not the file in the database. These are both two bugs that are currently being worked on.

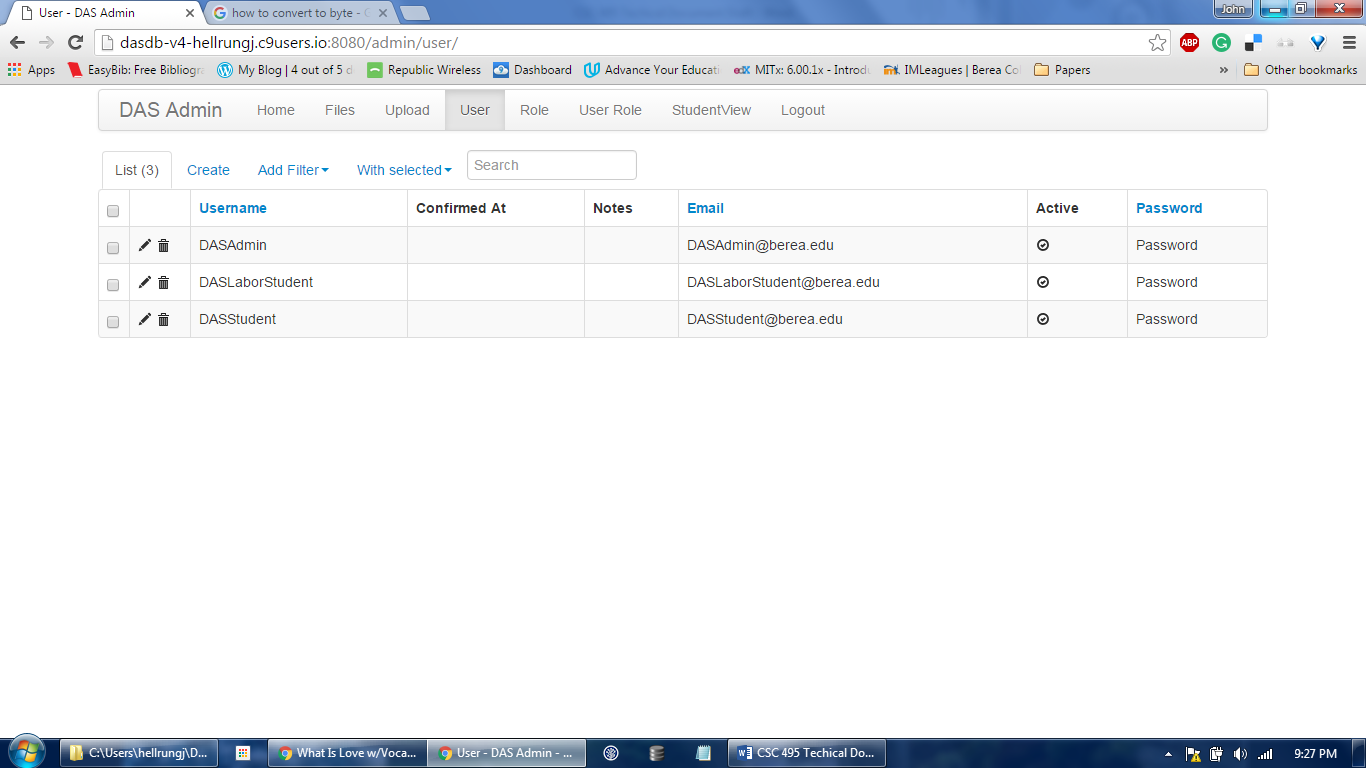
For more information about the database columns, (see section 8.4)

#### Upload View



The upload view is the answer to one of the bugs that is currently being answered. Now, the file view can upload data but it does not currently able to update the database data along with handling uploads. This view solves this current by allow the admin to upload file with a from like the upload file in the main application.

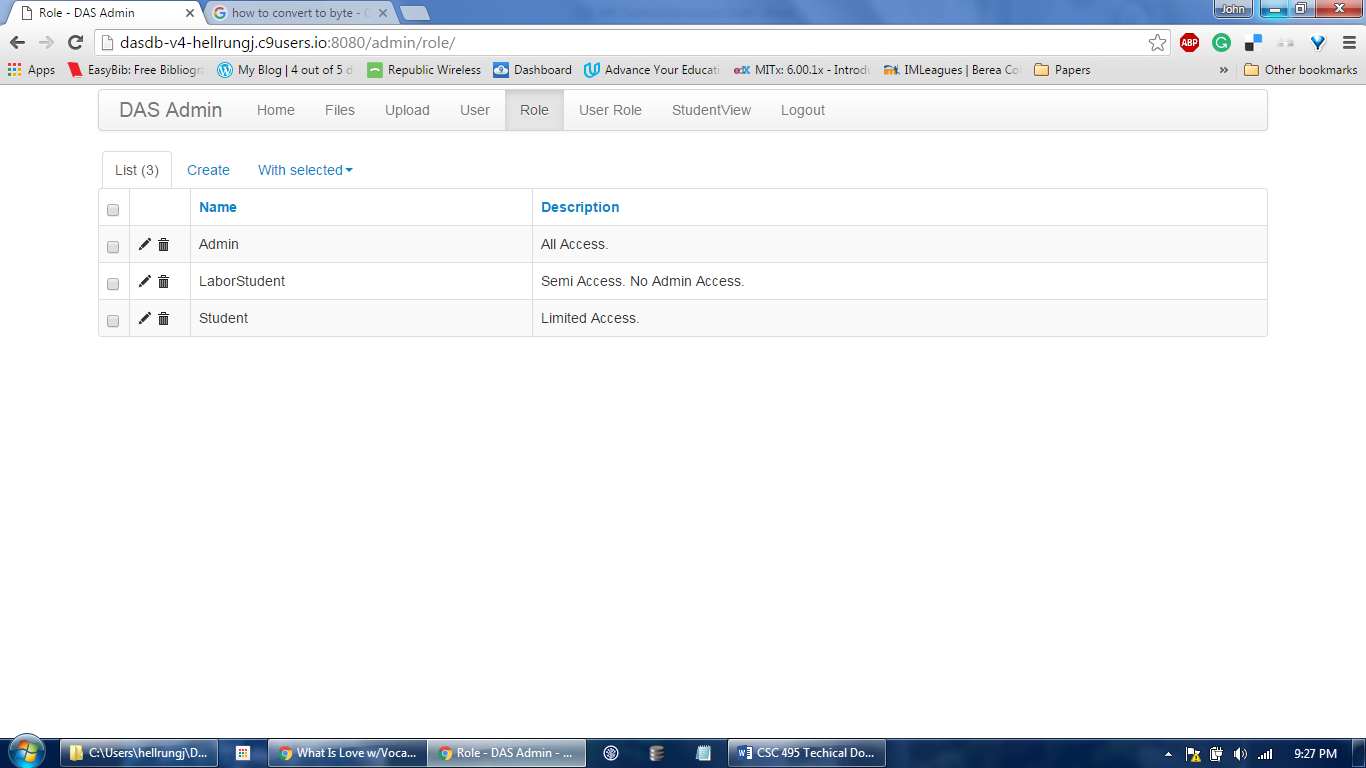
#### User View



The User View allows the admin to create, delete, edit, and update any data for any user. This view display all data of user table in the database in a clear table.

For more information about the database columns, (see section 8.1)

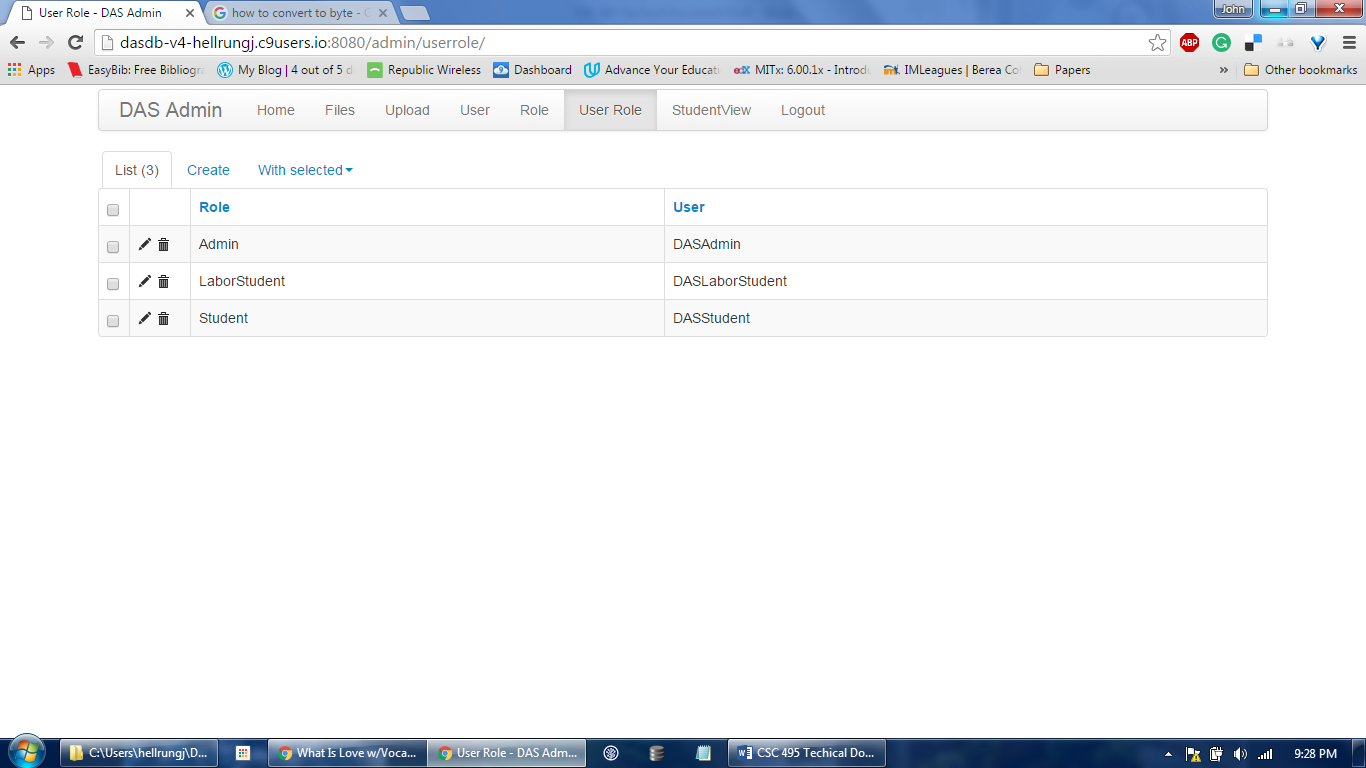
#### Role View



The Role View allows the admin to create, delete, edit, and update any data for any role. This view display all data of role table in the database in a clear table.

For more information about the database columns, (see section 8.1)

#### UserRole View



The Userole View allows the admin to create, delete, edit, and update any data for any user role. This view display all data of user role table in the database in a clear table.

For more information about the database columns, (see section 8.1)

## Detailed Design

This section aims to go over the purpose of different entities that will be involved during the process of building CodeWorkout’s question and answer forum. Since we are sticking to a very basic design for the beginning of a small web application and because forums like this generally focus on the flow of data to and from a database, these entities are mainly necessary additions to the preexisting database system that CodeWorkout uses.

### User

* + 1. **Purpose**

User table is to store all the users in the application. The User table helps by acting as an authorization tool that helps Flask\_Security secure the application. (see Section 5.8) Plus, User table acts as a record link the other tables.

#### Requirement

The user table must has a username field and password field at the minimum.

#### Description

Please see the following attributes of the table “User” (see section 2.2.1)

##### ID

The ID data acts as a unique definer for the table.

##### Username

The Username data acts as a definer for the user.

##### Email

The Email data acts as a definer for the user but allow for confirmation by email.

##### Password

The Password data acts as a unique definer for security.

##### Active

The Active data acts as a definer for the application to tell if the user is still active account.

##### Confirmed\_at

The Confirmed\_at data acts as a definer for the application to tell if the admin has confirmed the account and at what time.

##### Notes

The Notes data acts as a user notes for the admin.

#### Notes to the Future

If I had more time I would love to put two-point factor authentication which would call for a phone number with could be add to user table.

* 1. Role
     1. Purpose

The purpose for the Roles table is too identify a group of users.

#### Requirements

The Role table must have a name for the name of the role and description for the admin.

#### Description

Please see the following attributes of the table “Role” (see section 2.2.2)

##### ID

The ID data acts as a unique definer for the table

##### Name

The Name data acts as a unique definer for the admin.

##### Description

The Description data acts as notes for the admin.

* 1. UserRole
     1. Purpose

The purpose for the UserRole table is to set a many to many relationship for the user and role tables.

#### Requirements

The UserRole needs a User and Role.

#### Description

Please see the following attributes of the table “UserRole” (see section 2.2.3)

##### ID

The ID data acts as a unique definer for the table

##### User

This ties the user to role.

##### Role

This ties the role to user.

##### Name

This give a name to UserRole, which is the save as the Role Name.

##### Description

The description helps the admin get some information for the role for example the roles purpose.

* 1. File
     1. Purpose

Holding all the file data for a file in the file system in the database.

#### Requirements

The file needs file\_path and filename.

#### Description

Please see the following attributes of the table “File” (see section 2.2.4)

##### ID

The ID data acts as a unique definer for the table

##### Title

The title data for text of the title.

##### Author

To keep track of the people that wrote the text

##### Edition

To keep track of the texts edition information.

##### Size

To information the user when downloading.

##### Filename

To keep track the file in the database own filesystem.

##### File\_type

To let the user know what type of file that they are downloading.

##### Created\_at

To keep information about the time of file creation.

##### Last\_modified

To keep of modify data.

##### Last\_modified\_by

##### To keep track of the last user who modify the file data.

##### File\_path

To keep track of file location in the database.

##### Hidden

This is hide data in the database. Therefore, if a Labor Student delete a file then the admin can recover it.

#### Notes to the Future

I would to add a way of tracking track of file changes and who changed them rather than the last one.

* 1. Post
     1. Purpose

The purpose of the post table is to hold the post data for any post made in the database.

#### Requirements

The post table just needs a field for text.

#### Description

Please see the following attributes of the table “Post” (see section 2.2.5)

##### ID

The ID data acts as a unique definer for the table

##### Title

The title is for text data and summering the actual text giving the reads an idea of that they are about to read.

##### Text

The text data holds the message of the post.

##### Date

The date data is for hold the time the post was publish.

##### User

The user is for hold information about the author of the post of reads.

#### Notes to the Future

This feature is not fully intergraded yet but in the future, I would like if the Admin could send a post to the main index page as a notification or update to Labor Students.

### ER-Diagram

The following ER-Diagram is a small example of how the each of above tables relate to each other and other tables in the database system.

C:\Users\hellrungj\Downloads\Untitled Diagram.png