Proposal – JMU-Share

I. Overview

Many college students have had difficulties during their undergraduate career that have led to them seeking online resources to help them with their classes. However, many of these same students can spend hours trying to find online content to help them prepare for a test or supplemental notes to clarify their own notes but never find notes for what they need help with. JMU-Share plans to address this problem by creating a new web application specific to the JMU community that allows users to upload their class notes or view other notes. We plan to limit most features to strictly the JMU community by requiring a @dukes.jmu.edu or @jmu.edu email address to register and login in. With a student’s dukes.jmu.edu account, they will be able to upload their own notes, view other notes, and comment on other’s notes. If a faculty member registers with a jmu.edu account, they will be able to post their own notes for a class as well as endorse student’s notes or comment on a note with corrections or clarifications.

As for the financial market value, JMU-Share will not generate any revenue, our main goal is to help students feel more confident about their classes throughout the semester via peer-assisted learning.

The main transactions that will be supported by JMU-Share are uploading notes to a specific department and course number, searching for notes within first a department and then the course number, commenting on notes for further clarification or explanation, rating notes based on their helpfulness and detail, endorsing notes, and a user may remove a note that they have uploaded. Admin will also be able to remove any note if a note appears to lack as a supplemental resource.

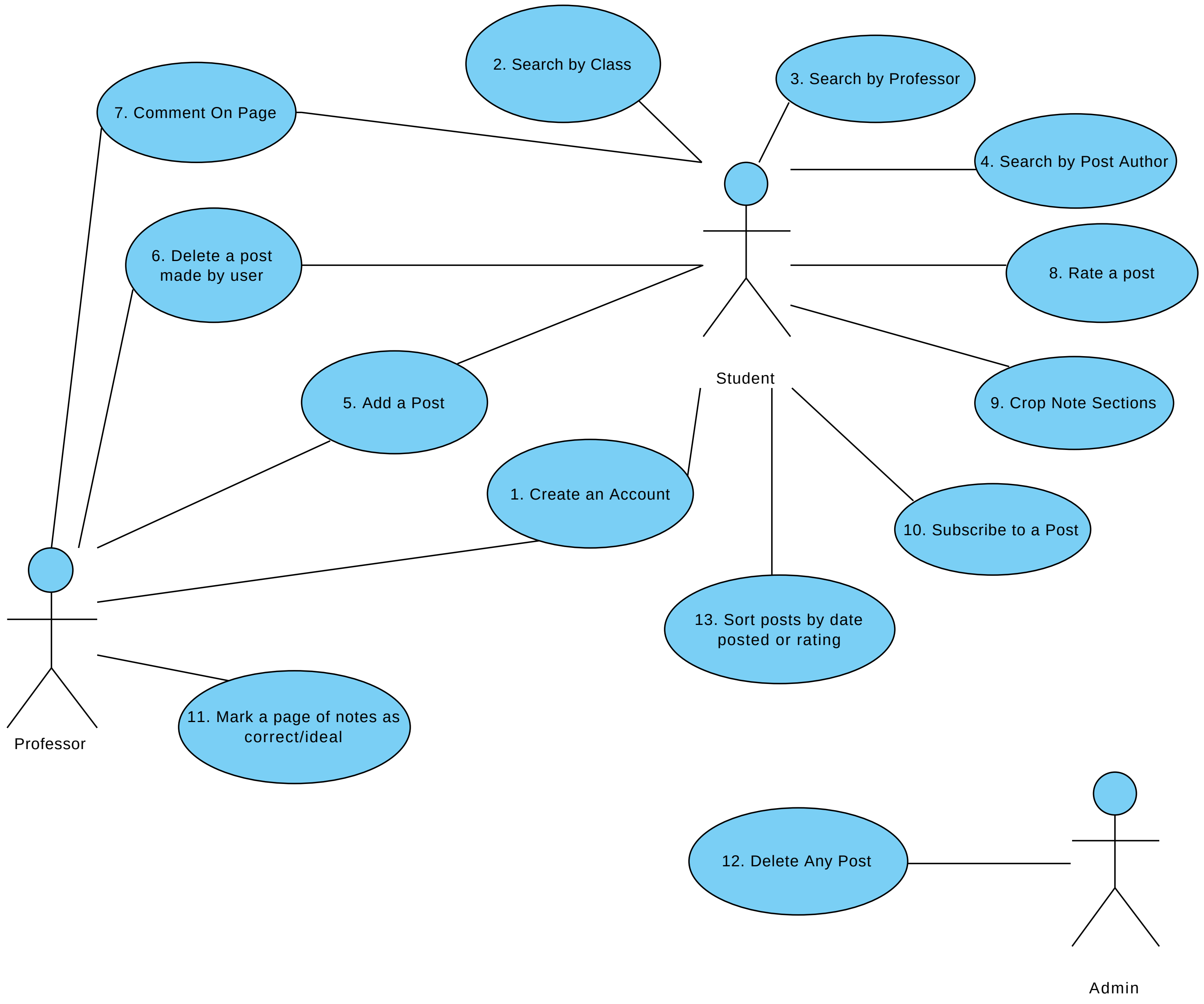
Dynamic Content

All of the web pages related to search results will be dynamic. The results of the search results will depend on the class name, author name, teacher name, note ratings, date ranges, and note titles. Depending on what criteria is used or omitted in the search, results will be different. The results can then be sorted by the previously stated criteria. Each result will include the title of the notes as a hyperlink connecting to the notes and all of the other information as smaller text below the link. All of the search data will be stored in a database table for easy querying. The information in the database about each of the note entries will be gathered when the notes are uploaded to the application except for the rating. Ratings for uploads will depend on the user approval of the notes. Only when a search result is clicked will the actual notes document be retrieved and displayed.

Once notes are in view, the comments on the actual notes themselves will appear as well. Comments are our second form of dynamic content. When a professor or student has a question or comment about someone’s notes they can attach a text to the notes page for all users to see. These comments will be stored in a separate database table that additionally logs exactly where the comment was attached to the notes. This feature adds the ability for the notes to be altered and corrected without requiring the original author to re-upload the notes. In addition to allowing modifications to the notes, we hope that commenting will create discussion among the users of the application and spur interaction between classmates.

Database Tables Needed:

- User Table
  - contains a list of user information:
  - - id (unique identifier for each user; long)
  - - password (used for login; varchar(255))
  - - active (is the user active or not; boolean)
  - - email (used for login; varchar(255))
  - - name (name of the user; varchar(255))
  - - username (username for the user; varchar(255))
  - - role id (foreign key to roles table; long)
- Posts Table
  - contains each posts information including:
  - - author id (foreign key to the user table storing the author of the post; long)
  - - class (foreign key to the class table; long)
  - - text (contains the body of the text for the note; varchar(255))
  - - image (information on the image(if included); jpg)
  - - rating (average post rating from 1 to 5; float)
  - - id (the post's unique identifier in the database; long)
  - - endorsement information (if a professor believes the notes are adequate or should be used as an example; boolean)
- Comments Tables
  - contains a set of comments for the associated posts this information includes:
  - - post id (used to link tables; long)
  - - comment text information (the actual comment; varchar(255))
  - - author (foreign key to the user table; long)
- Roles Table
  - contains possible roles for a user to have
  - -id (unique identifier for each role; long)
  - -role name (the String name for each role; varchar(255))
- Class Table
  - contains all created classes and professors associated with them
  - - id (unique identifier for the database; long)
  - - class number (the number for the class; integer)
  - - department (the department for the class; varchar(255))
  - - professor id (foreign key to the user table, only professors; long)


2. Search by Class
3. Search by Professor
7. Comment On Page
4. Search by Post Author
6. Delete a post made by user
8. Rate a post
5. Add a Post
Student
9. Crop Note Sections
1. Create an Account
10. Subscribe to a Post
13. Sort posts by date posted or rating
11. Mark a page of notes as correct/ideal
Professor
12. Delete Any Post
Admin

<u>**Use Case Descriptions**</u>

1. Create an Account
*Abstract:*A student or professor uses this use case to create their account on JMU-Share.
Precondition: Student or professor has a valid JMU email.
Postcondition: The student or professor has an account on JMU-Share.

Steps:
1. Student or professor chooses to create an account on JMU-Share
2. System requests a valid JMU e-mail to use as the account username
3. Student or professor provides an email.
4. System sends confirmation email.
5. Student or professor clicks link in confirmation email.
6. Account is created.

Alternatives:
3a1. Student or professor provides an invalid email
3a2. System returns to step 2

2. Search Through Posts By Class
Abstract: A student uses this use case to search through note postings by a specific class
Precondition: Student is logged into JMU-Share
Postcondition: Student views all results for their search

Steps:
1. Student selects a class from the options and requests search results.
2. System returns any postings matching the searched class.

3. Search Through Posts By Professor
Abstract: A student uses this use case to search through note postings by a specific professor.
Precondition: Student is logged into JMU-Share.
Postcondition: Student views all results for their search.

Steps:
1. Student enters a professor's name and requests search results.
2. System returns any postings matching the searched professor.

4. Search Through Posts By Post Author
Abstract: A student uses this use case to search through note postings posted by a specific author
Precondition: Student is logged into JMU-Share.
Postcondition: Student views all results for their search.

Steps:
1. Student enters an author's name and requests search results.
2. System returns any postings matching the search author.

5. Add a Post
Abstract: A user uses this use case post a note to the system
Precondition: User is logged into JMU-Share.
Postcondition: System creates the post or adds the new post as a comment to a preexisting post

Steps:
1. User enters their notes into the box provided.
2. System prompts the student for confirmation.
3. System creates the posting with the students information.

Alternatives:
1a1. User chooses an image file to upload.
1a2. System moves to step 2.
2a1. User hits no in the confirmation window.
2a2. System moves to step 1.

6. Delete a Post(student or professor)
Abstract: A user uses this use case to delete a note they posted from the system
Precondition: User is logged into JMU-Share.
Postcondition: User's note is removed from the system.

Steps:
1. User chooses a note that they uploaded to the system.
2. System asks for confirmation.
3. System removes the note.

Alternatives:
2a1. User hits no in the confirmation window.
2a2. System moves to step 1.

7. Comment on pages
Abstract: A user uses this use case to add a comment to a note or notes.
Precondition: User is logged into JMU-Share.
Postcondition: User's comment is logged to the note.

Steps:
1. User chooses a note that has been uploaded to the system.
2. User provides the comment.
3. System prompts for confirmation.
4. System checks the comment to make sure it's valid and appropriate.
5. System adds the comment to the selected note

Alternatives:
3a1. User hits no in the confirmation window.
3a2. System moves to step 2.
4a1. System finds the comment to be invalid or inappropriate.
4a2. System moves to step 2.

8. Rate Posts
Abstract: A user uses this use case to rate a note based on how useful they feel it is.
Precondition: User is logged into JMU-Share.
Postcondition: User's rating is stored in the system.

Steps:
1. Student chooses a note posting from the system.
2. Student selects between 1 and 5 stars for the posting.
3. System stores the choice.

Alternatives:
2a1. Student hits a different star selection on the posting.
2a2. System removes the previous star selection and adds the new selection to the database.

9. Crop Note Sections
Abstract: A student uses this use case to crop a note image to save only the parts they want
Precondition: Student is logged into JMU-Share and are currently on a post with a note image
Postcondition: System saves the part of the note that was cropped.

Steps:
1. Student chooses a note posting from the system with an image file.
2. Student hits crop icon.
3. System opens crop window to allow student to crop whatever part of the image they want to save.
4. System saves cropped portion of the image.

Alternatives:
3a1. Student clicks close-button on crop window.
3a2. System moves to step 2.

10. Subscribe to a Post
Abstract: A student uses this use case to favorite a note posting and mark it to be followed by their account
Precondition: Student is logged into JMU-Share.
Postcondition: Posting is marked to be followed by the student's account

Steps:
1. Student chooses note posting from the system.
2. Student hits the favorite button
3. System marks the post to be followed by the student's account
4. System notifies student whenever a new post is made upon the following note posting.

Alternatives:
2a1. Student hits the favorite button again.
2a2. System removes favorite marking and moves to step 1.

11. Mark a note posting as correct/ideal or endorse an answer
Abstract: A professor uses this use case to mark a set of notes as perfect answer or best answer, similar to what Piazza does.
Precondition: Professor is logged into JMU-Share.
Postcondition: The system displays the note posting as professor-endorsed.

Steps:
1. Professor chooses note posting from the system.
2. Professor hits the endorsement button.
3. System marks the post as endorses as a good answer by the professor.

Alternatives:
2a1. Professor hits the endorsement button again.
2a2. System removes the endorsement marking on the post.
2a3. System moves to step 1.

12. Delete a Post(administrator)
Abstract: An administrator uses this use case to delete any note from the system
Precondition: Administrator is logged into JMU-Share.
Postcondition: The note is removed from the system.

Steps:
1. The Administrator chooses a note that has been uploaded to the system.
2. System asks for confirmation.
3. System removes the note.

Alternatives:
2a1. Administrator hits no in the confirmation window.
2a2. System moves to step 1.

13. Sort Posts By Date Posted or Rating
Abstract: A student uses this use case to sort displayed posts by the date they were posted or by their current 5-star rating.
Precondition: Student is logged into JMU-Share.
Postcondition: System displays all postings in order based upon the chosen sorting method.

Steps:
1. Student chooses the sort drop-down list.
2. Student selects one of the sort options in the list.
3. System displays the sorted results to the student.