share’n’complain

share’n’complain Design Report

Version 1.2

Contributors:
Neal Rea, Carlton Welch,
and Daniel Vignoles

Revision History

Date	Version	Description	Author
11/17/2018	1.0	Draft sequence diagrams for each use-case	CDN
11/18/2018	1.1	Added ER and collaboration diagrams, petri-nets, GUI wireframes, method pseudocode, minutes.	CDN
11/19/2018	1.2	Edit ER diagram, complete design pseudocode, aesthetic edits	CDN

Table of Contents

Design Report

1. Introduction

1.1 Purpose

This document serves as a detailed design specification for all subsystems of share’n’complain, complete with scenarios, sequence diagrams, and petri-nets for each use-case, overall system E-R diagram, detailed pseudocode for every class method, and wireframes for all GUI screens.

1.2 Scope

share’n’complain is a document creation platform that allows users edit, save, and share text files. It provides a host of useful tools to this end, aiming to streamline the document creation process through easy information sharing between users.

share’n’complain implements a flexible system that allows a document to be shared with many, few, or just one user. Any edits made to a document are saved in a detailed history file, allowing for rolling back to a previous version of a document. To prevent unwanted changes to documents, users may lock and unlock documents for editing, as well as file complaints with a document owner or administrator should something go awry.

share’n’complain offers a 3-tier privilege system that includes a Super User, Ordinary User, and Guest User. The Ordinary User can access all important document creation, editing, and sharing features, while the Super User has access to additional administrative features, including a list of words that are not allowed to be used in any document. The Guest User has limited feature access, but may read available documents. Each Ordinary and Super user has a profile page with a picture, as well as a list of technical interests that is submitted when applying for Ordinary User status.

To use share’n’complain, a computer is required to download and access the software. All document storage is local, so users must have access to the machine containing documents they would like to access.

1.3 Definitions, Acronyms, and Abbreviations

SU: Super User

OU: Ordinary User

GU: Guest User

Document: A text file with only ONE word per line.

Taboo Word: A “bad” word. A list of Taboo Words is maintained by the SU.

Locked: A locked document cannot be updated by any other users until it is unlocked.

Public: A public document is visible to all users.

Restricted: A restricted document document can be viewed by guest users, and edited by ordinary or super users.

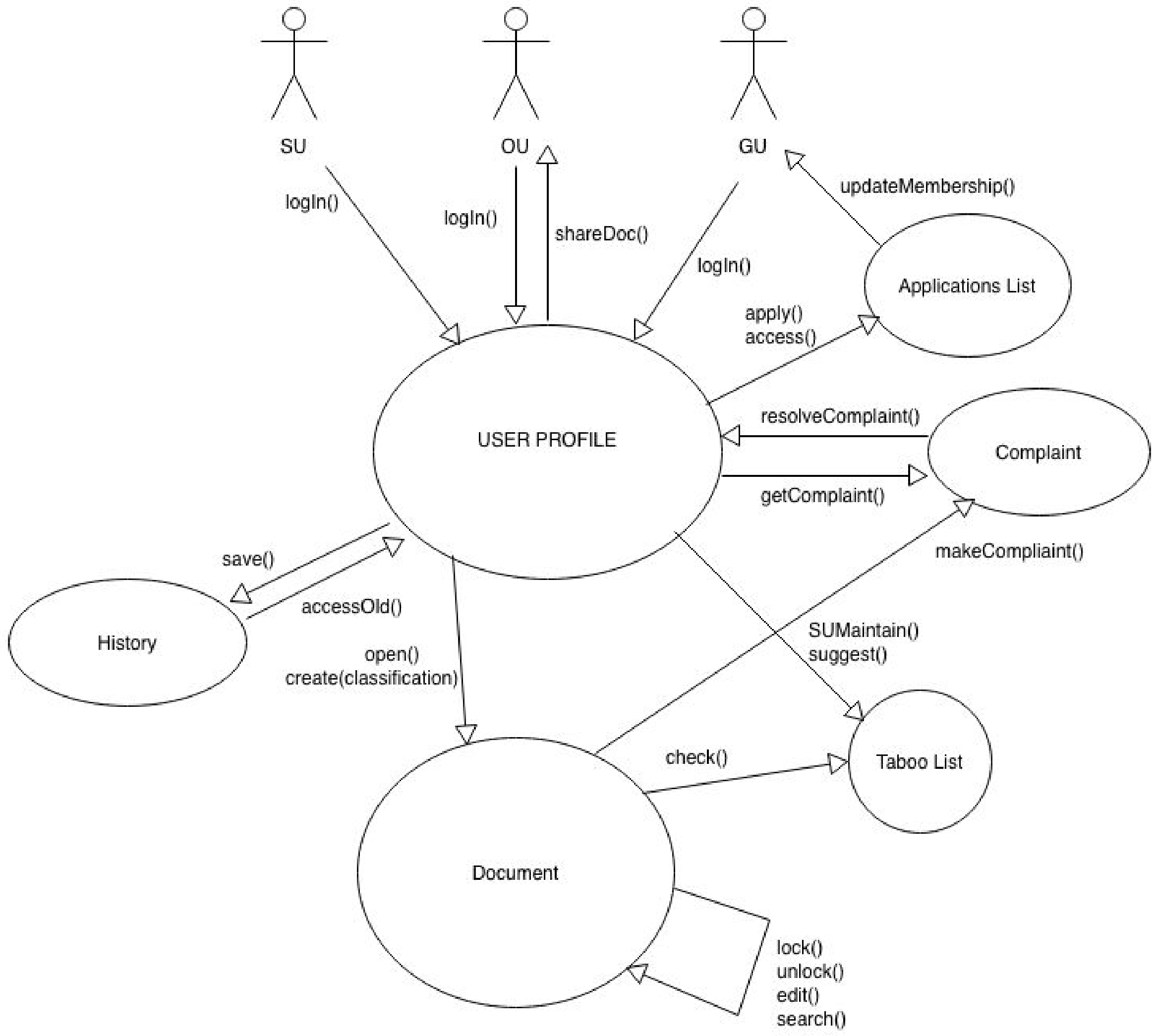
Shared: A shared document can only be viewed or edited by ordinary and super users who are invited.

Private: A private document is only accessible to the OU/SU who created it.

Ownership: A user has ownership of a document if he/she created it.

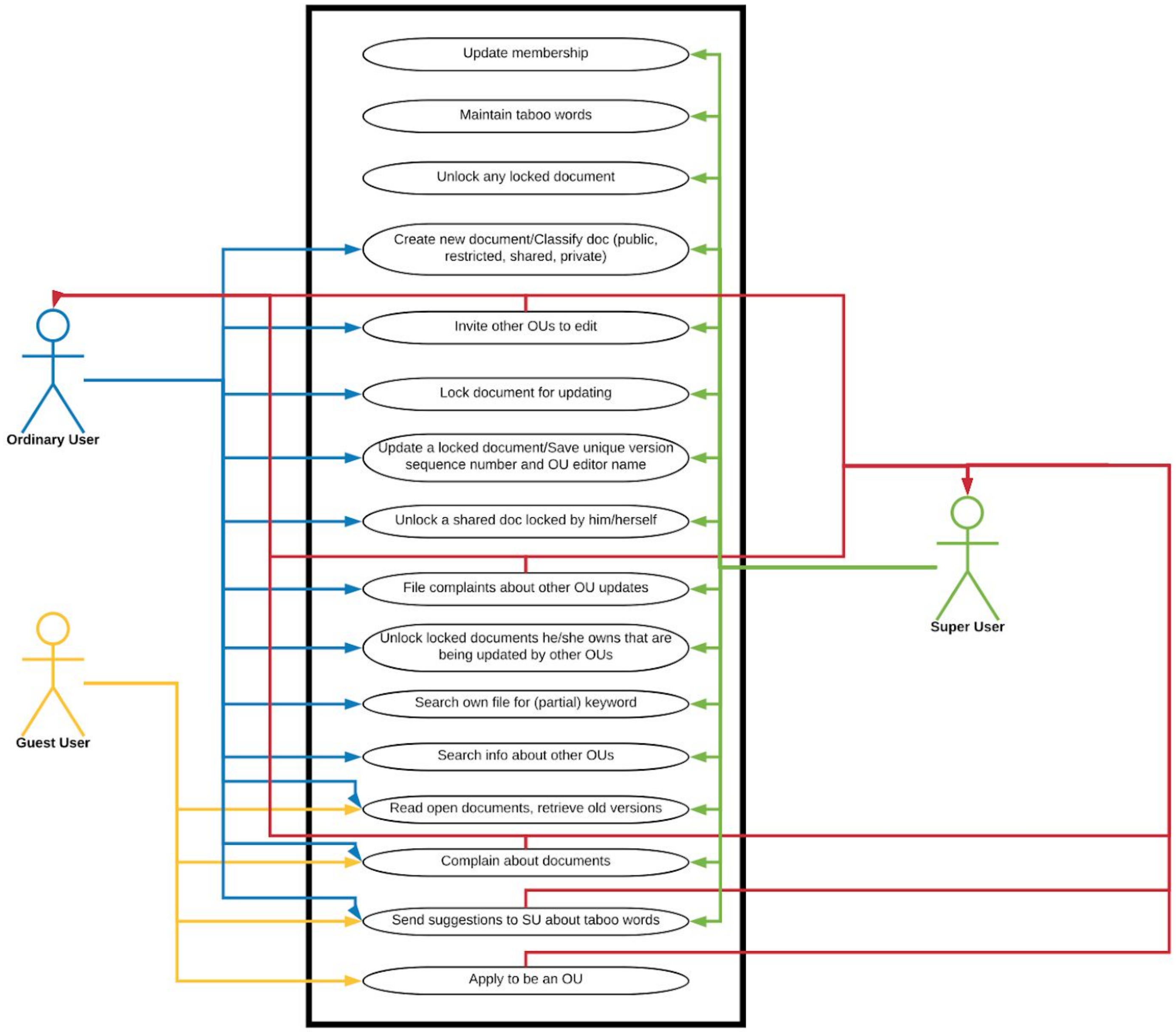
Version Sequence Number: A sequence which references a particular update version of a document.

1.4 Overview/Collaboration Diagram



2. Use-Cases

Use-Case Diagram:

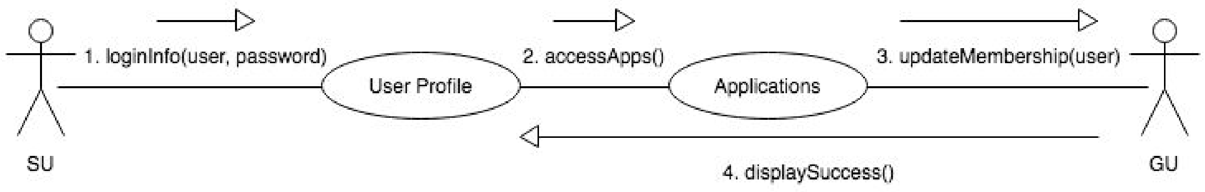

Update membership
Maintain taboo words
Unlock any locked document
Create new document/Classify doc (public, restricted, shared, private)
Invite other OUs to edit
Lock document for updating
Update a locked document/Save unique version sequence number and OU editor name
Unlock a shared doc locked by him/herself
File complaints about other OU updates
Unlock locked documents he/she owns that are being updated by other OUs
Search own file for (partial) keyword
Search info about other OUs
Read open documents, retrieve old versions
Complain about documents
Send suggestions to SU about taboo words
Apply to be an OU
Ordinary User
Guest User
Super User

2.1 Use-Case Scenarios/Diagrams

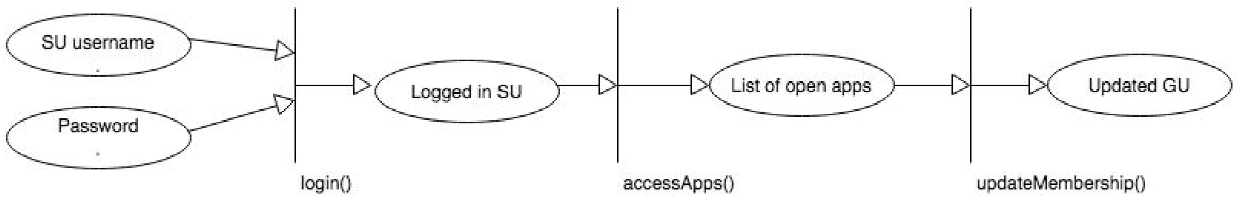
<u>Update Membership:</u>

- Scenarios
  - Normal
    - The SU can view open applications for GUs to become OUs. The SU can either accept or deny the application. The SU can also demote an OU to GU.
  - Exceptional
    - The SU promotes a GU with an empty “interests” field. This will be handled in the application class. When a GU applies to become an OU, he/she must not leave the “interests” field blank.

- Sequence Diagram



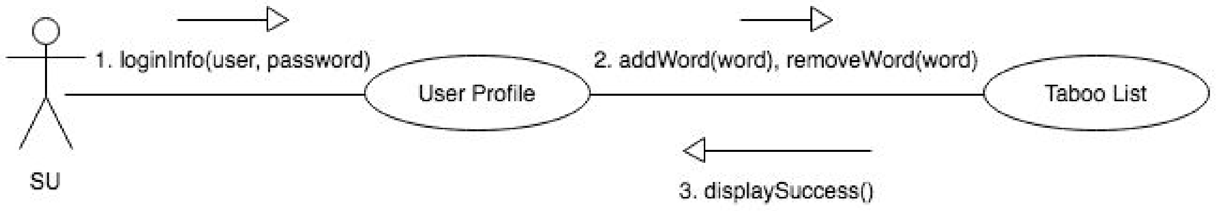
- Petri-net



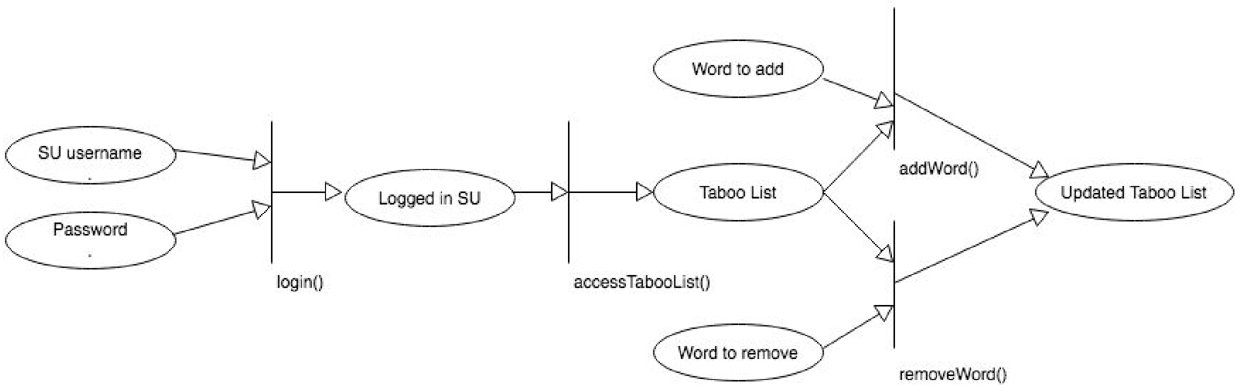
Maintain Taboo Words:

- Scenarios
  - Normal
    - The SU can parse through a list of suggested “taboo” words and decide whether or not to add them to the “Taboo List”. The SU can also remove words from the “Taboo List”.
  - Exceptional
    - The SU tries to remove a word from an empty “Taboo List”. This will be handled by a simple boolean check on whether or not the list is empty.

- Sequence Diagram


1. loginInfo(user, password)
User Profile
2. addWord(word), removeWord(word)
Taboo List
SU
3. displaySuccess()

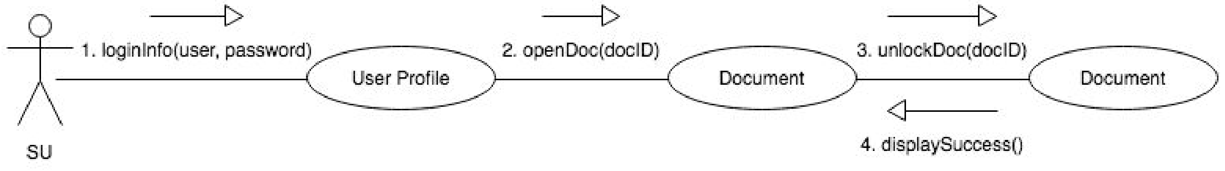
- Petri-net



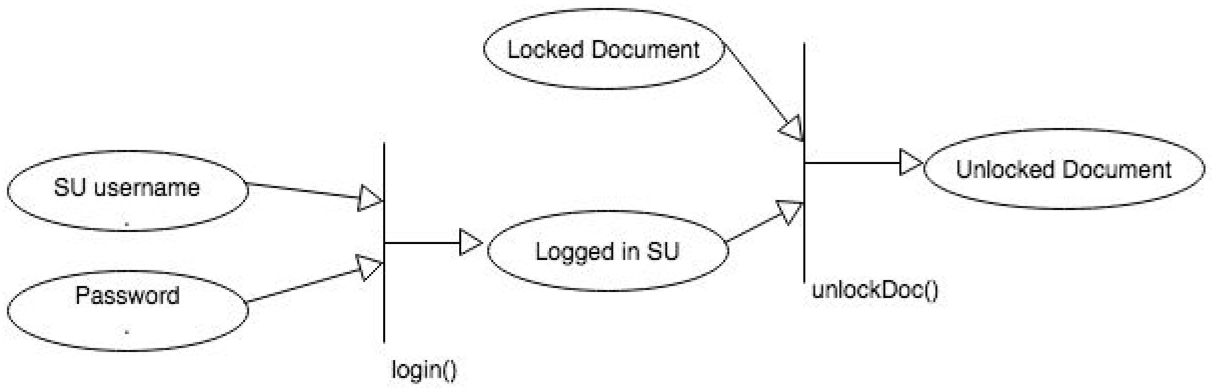
<u>Unlock any locked document:</u>

- Scenarios
  - Normal
    - The SU can unlock any locked document, regardless of ownership.
  - Exceptional
    - The SU unlocks a document by mistake. To solve this, the SU will also have permission to lock any document as well.

- Sequence Diagram



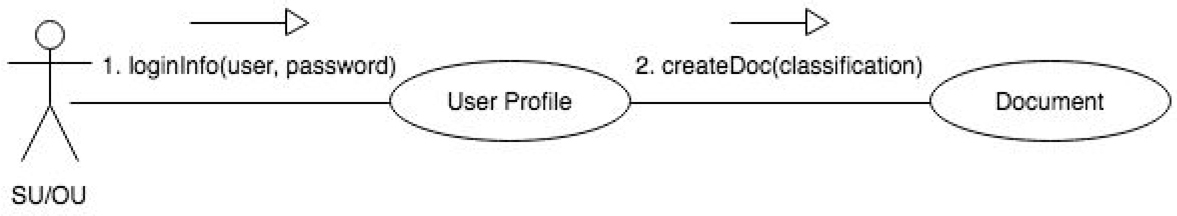
- Petri-net



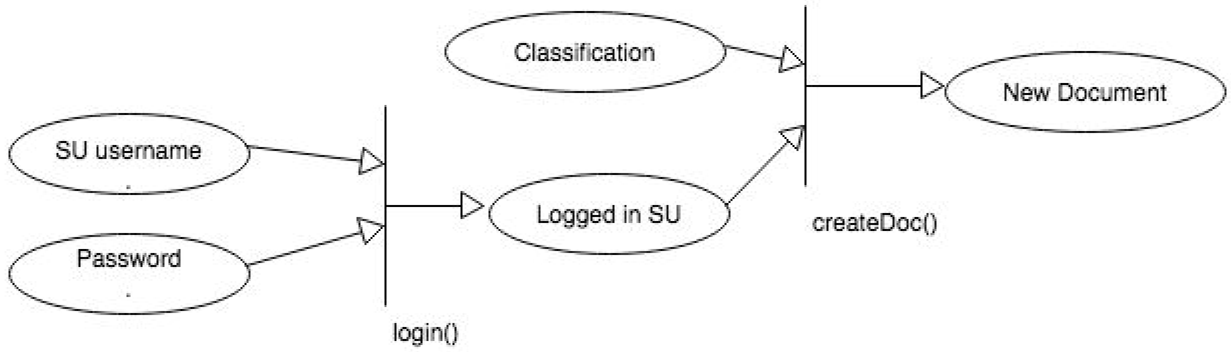
<u>Create/classify new document:</u>

- Scenarios
  - Normal
    - A registered OU logs in using a valid username and password, and selects the option to create a new document from the user profile page. A classification is required from the user (public, restricted, shared, private) before it is rendered.
  - Exceptional
    - A guest user attempts to select the option to create a new document. This is handled on the GUI level by a boolean check of the user's type.

- Sequence Diagram



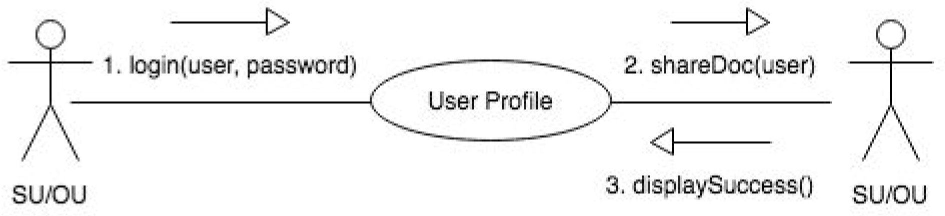
- Petri-net



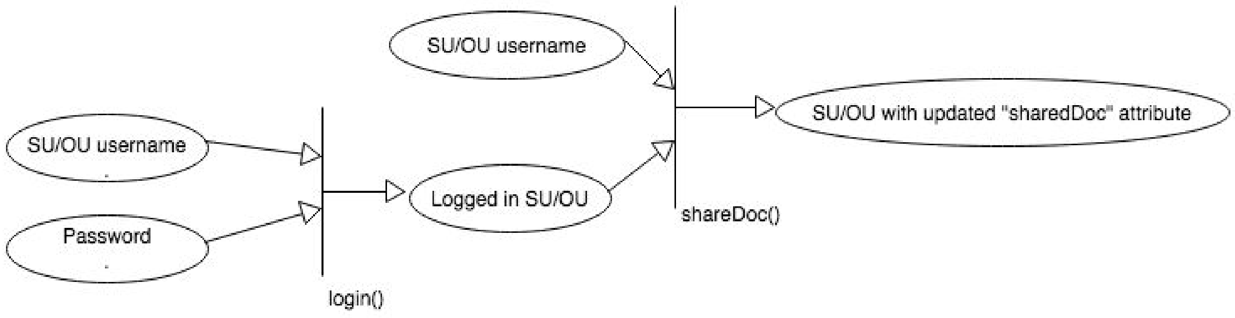
Invite other OUs to edit:

- Scenarios
    - Normal
        - A registered OU sends an invite to another registered OU, using a valid username. The attribute "docsSharedWith" of the invited user is updated.
    - Exceptional
        - A registered OU sends an invite to another registered OU, but the username given is invalid, and does not associate with a registered OU. This is handled by disallowing manual username entries. Invitation usernames will be populated from a database of existing users only.

- Sequence Diagram



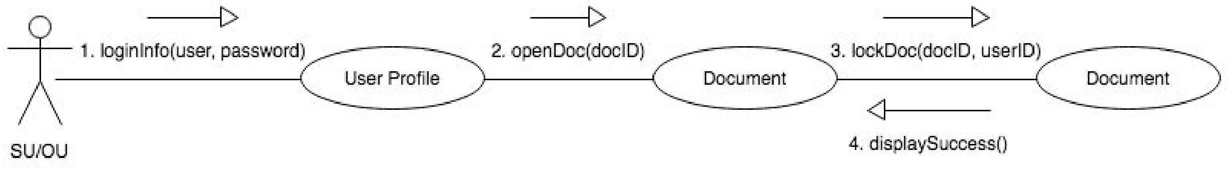
- Petri-net



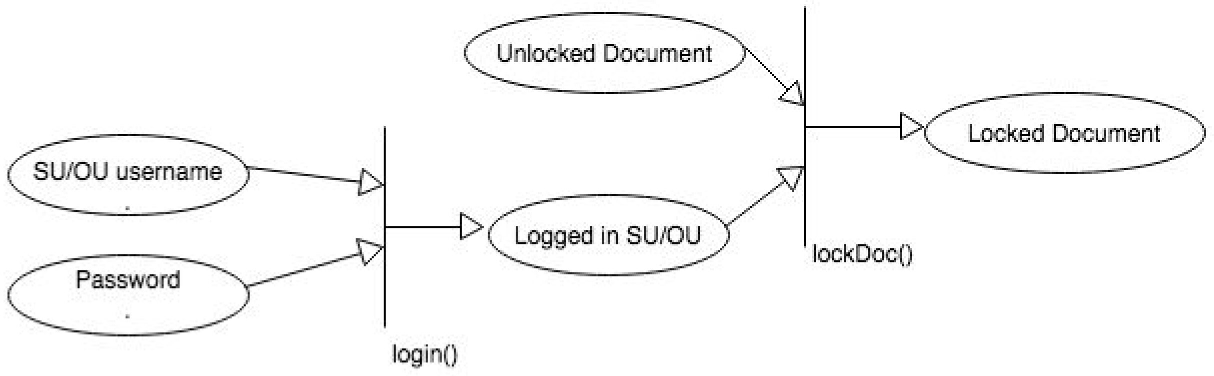
<u>Lock document for updating:</u>

- Scenarios
  - Normal
    - An OU with access to a document locks a document to make edits.
  - Exceptional
    - An OU with access to a document attempts to lock a document that has already been locked. This is handled by the GUI when rendering the document. A locked document will not have the option to “lock” until it has been “unlocked”.
    - An OU with access to a document attempts to edit a document that has already been locked.

- Sequence Diagram



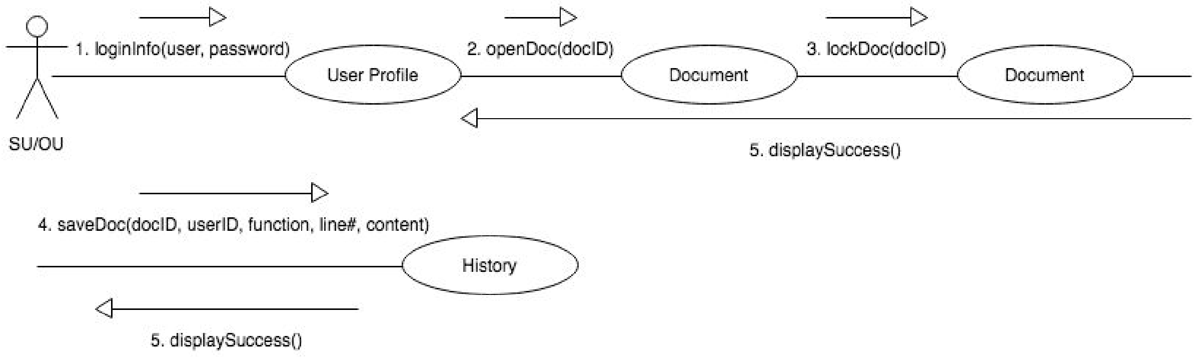
- Petri-net



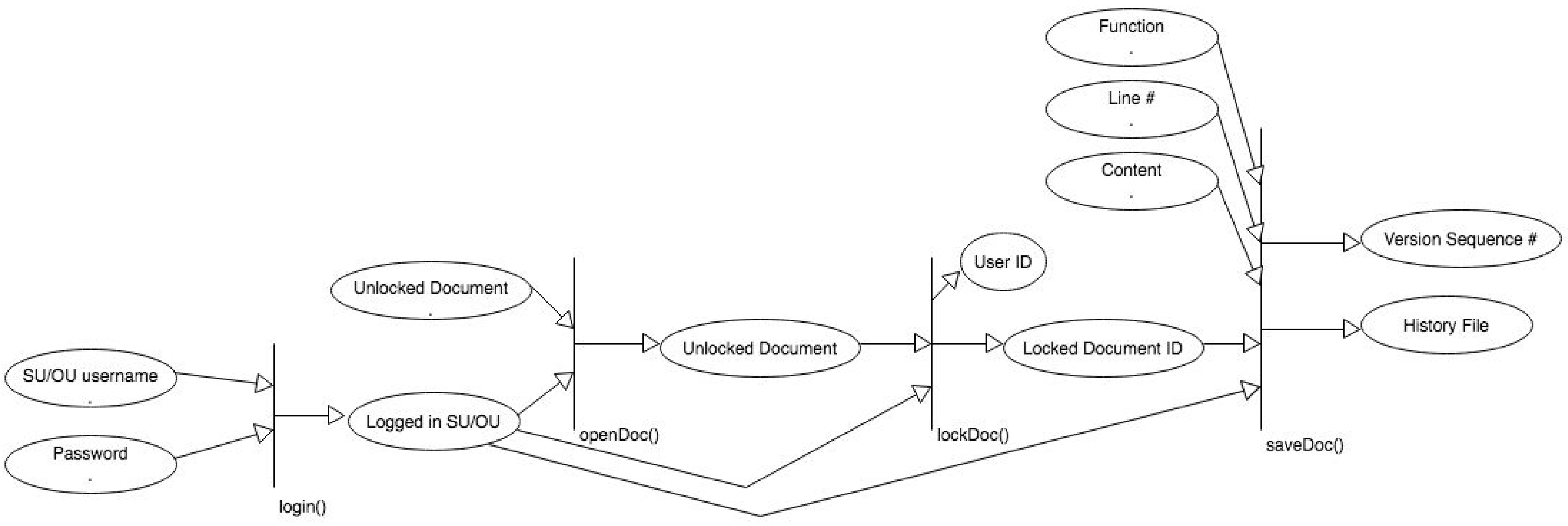
<u>Update/save locked document:</u>

- Scenarios
    - Normal
        - An OU with edit access to a locked document commits edits. The document history is updated, and the version number is incremented.
    - Exceptional
        - An OU with edit access to a locked document commits no edits. This will not harm the version control feature since a version of the document with no edits requires no changes upon reverting.

- Sequence Diagram



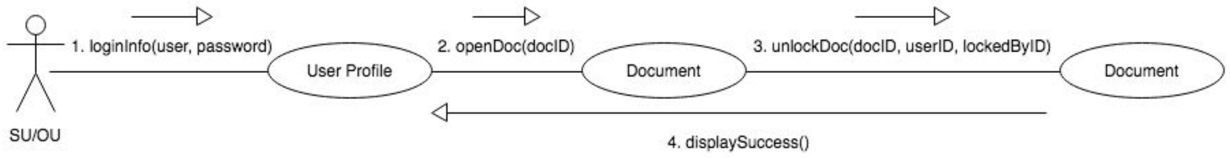
- Petri-net



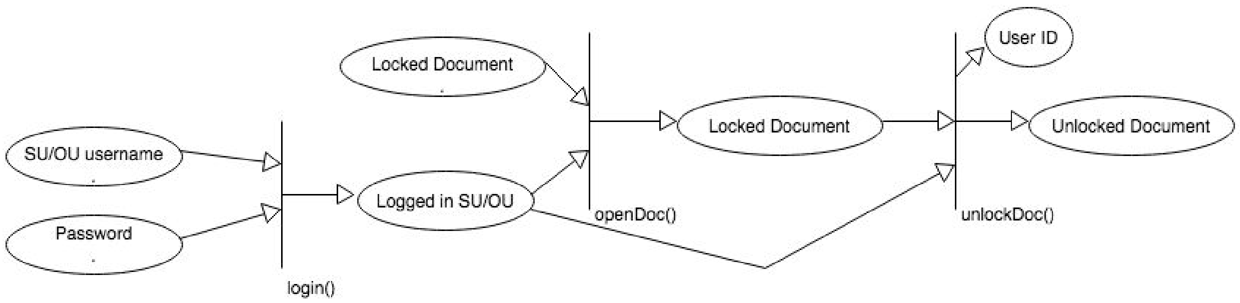
Unlock shared doc locked by him/herself:

- Scenarios
  - Normal
    - A registered OU unlocks a shared document that he/she had previously locked.
  - Exceptional
    - A registered OU attempts to unlock a shared document that another OU had previously locked. The unlockDoc() method will require a check of the document object for the user ID of the OU who locked it last.

- Sequence Diagram



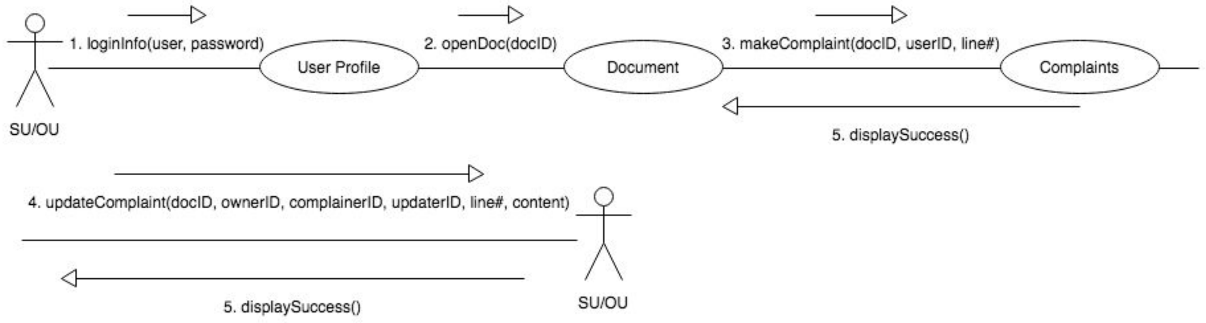
- Petri-net



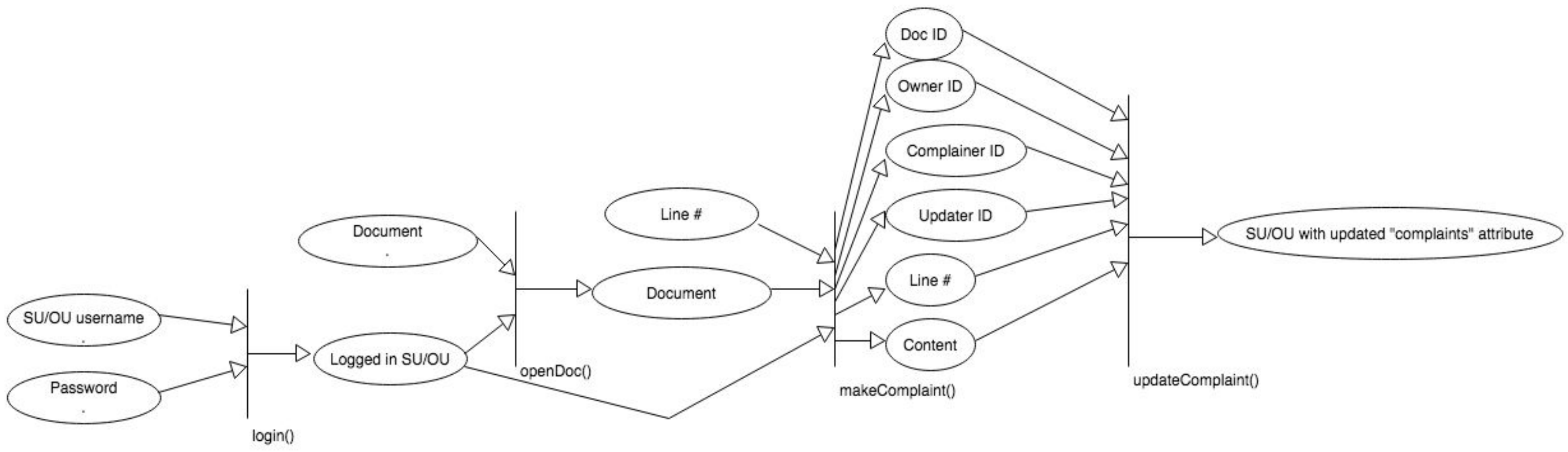
File complaints about other OU updates:

- Scenarios
  - Normal
    - A registered OU opens a document and clicks on the "complain" button. He/she then inputs the line number to complain about. The author of the line in question is calculated via the history file. The owner of the doc is notified via a complaints attribute.
  - Exceptional
    - A complaint is made about a line number that does not exist. A simple check will be made to ensure the line number is less than the length of the document.

- Sequence Diagram


1. loginInfo(user, password)
User Profile
2. openDoc(docID)
Document
3. makeComplaint(docID, userID, line#)
Complaints
SU/OU
5. displaySuccess()
4. updateComplaint(docID, ownerID, complainerID, updaterID, line#, content)
SU/OU
5. displaySuccess()

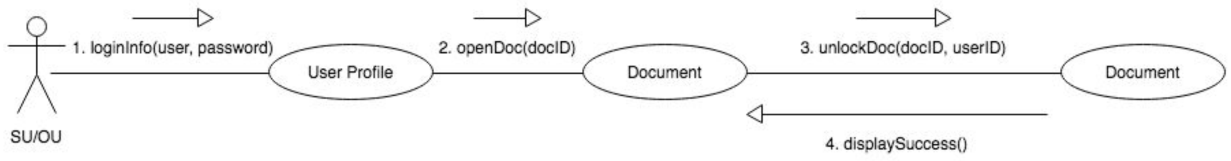
- Petri-net



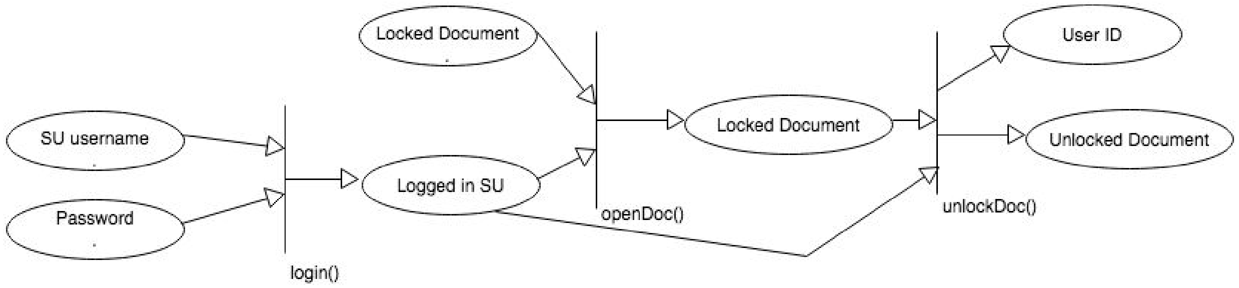
<u>Unlock locked docs he/she owns that are being updated by other OUs:</u>

- Scenarios
    - Normal
        - A registered OU can unlock a document that has been locked for editing by another OU if and only if he/she is the owner of the document.
    - Exceptional
        - An OU tries to unlock a locked document that he/she does not own. The unlockDoc() method must take the user ID and doc ID as input parameters in order to check that the owner ID of the doc matches the user ID.

- Sequence Diagram



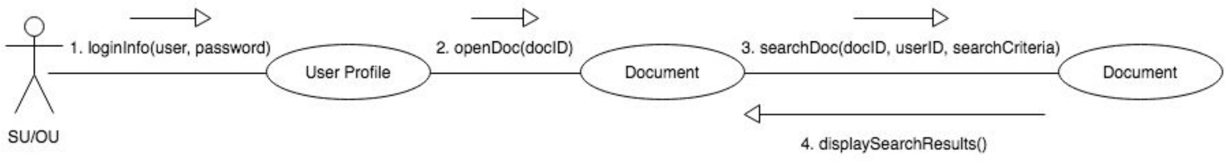
- Petri-net



<u>Search own file for (partial) keyword:</u>

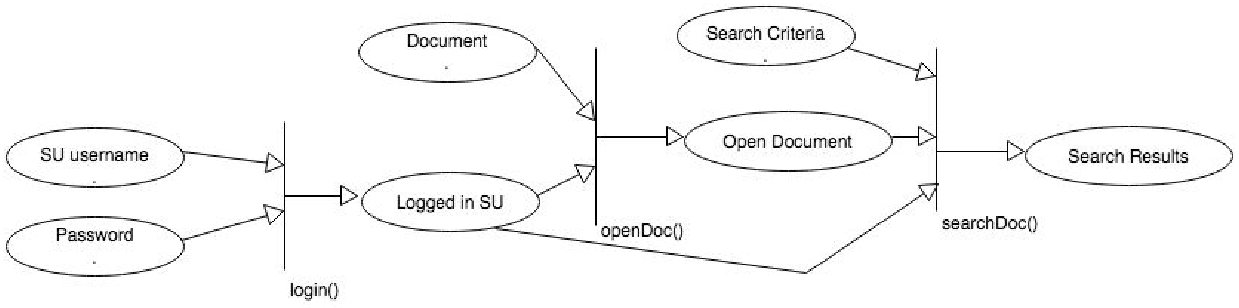
- Scenarios
  - Normal
    - A registered OU can open a document that he/she owns and search for a word or partial word.
  - Exceptional
    - An OU tries to search for a word in a document which he/she does not own. This will be handled on the GUI level. The search bar will not render on documents that the user does not own.

- Sequence Diagram



1. loginInfo(user, password)
2. openDoc(docID)
3. searchDoc(docID, userID, searchCriteria)
4. displaySearchResults()
SU/OU
User Profile
Document
Document

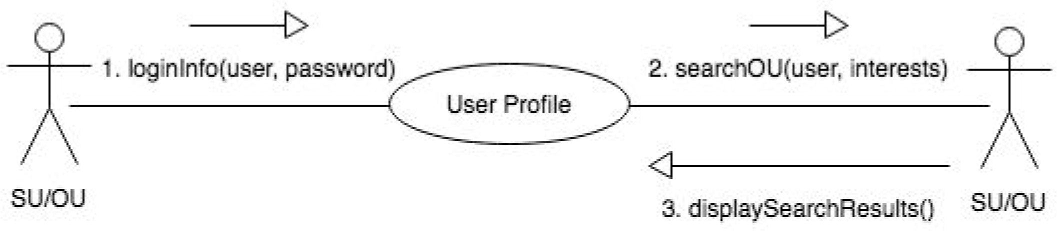
- Petri-net



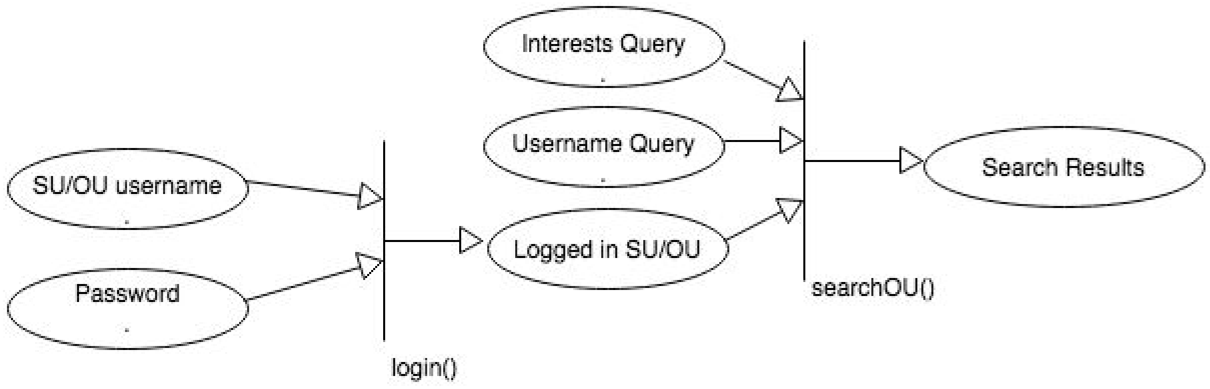
<u>Search info about other OUs:</u>

- Scenarios
  - Normal
    - A registered OU viewing his/her profile page can search through the database of existing users based on username or interests.
  - Exceptional
    - The search criteria does not match any existing users. The search results will be populated dynamically after each character entry. A non-match will simply return an empty list.

- Sequence Diagram


1. loginInfo(user, password)
User Profile
2. searchOU(user, interests)
3. displaySearchResults()
SU/OU
SU/OU

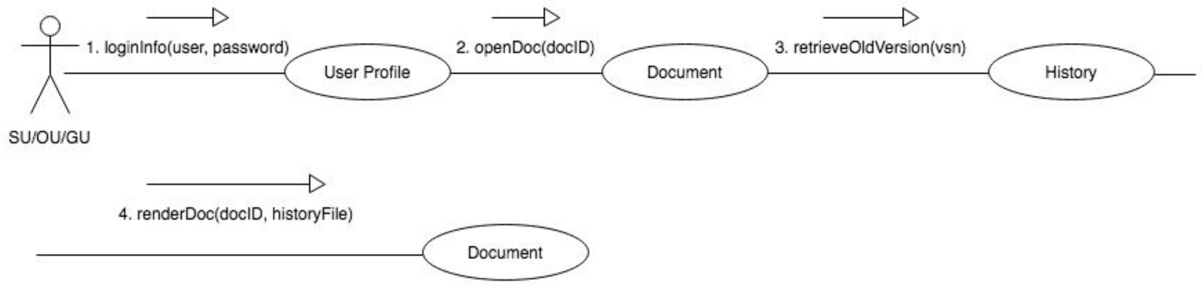
- Petri-net



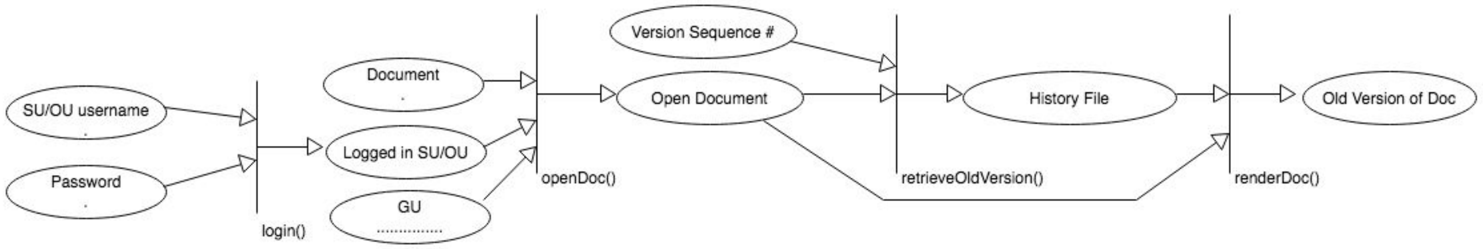
<u>Retrieve old versions of doc:</u>

- Scenarios
  - Normal
    - Any user type (SU/OU/GU) can open a document as read-only. They can click on "view history" and retrieve older versions of the same document.
  - Exceptional
    - If a user tries to retrieve history of a new document, the resulting list of version sequence numbers will be empty.

- Sequence Diagram


1. loginInfo(user, password)
User Profile
2. openDoc(docID)
Document
3. retrieveOldVersion(vsn)
History
SU/OU/GU
4. renderDoc(docID, historyFile)
Document

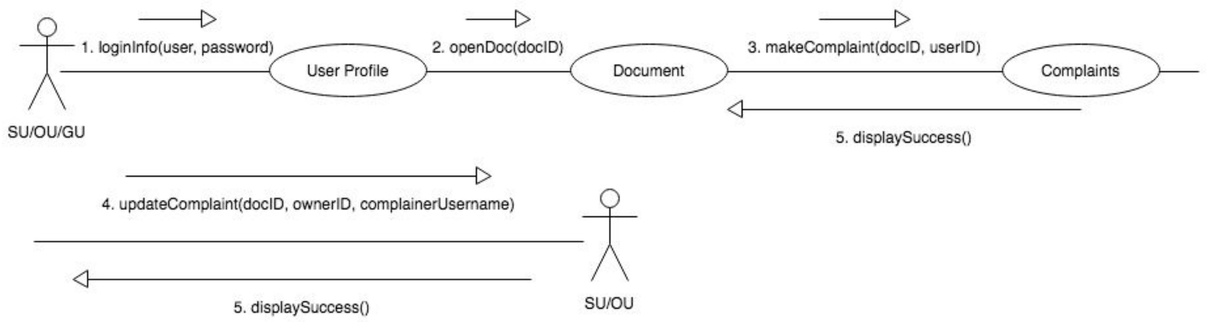
- Petri-net



File complaints about entire documents:

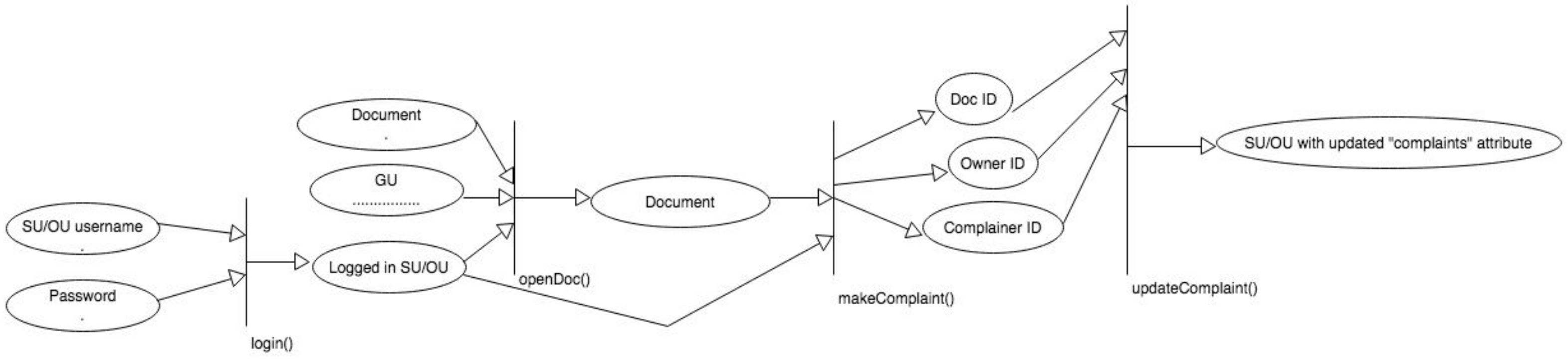
- Scenarios
  - Normal
    - Any user type can open a document and click on “complain” to make a general complaint about the entire document. These complaints are handled by the owner of the document.
  - Exceptional
    - The complainer leaves the “complaint” field blank. This will be handled by requiring an input for the “complaint” field upon submission.

- Sequence Diagram



SU/OU/GU
1. loginInfo(user, password)
User Profile
2. openDoc(docID)
Document
3. makeComplaint(docID, userID)
Complaints
5. displaySuccess()
4. updateComplaint(docID, ownerID, complainerUsername)
SU/OU
5. displaySuccess()

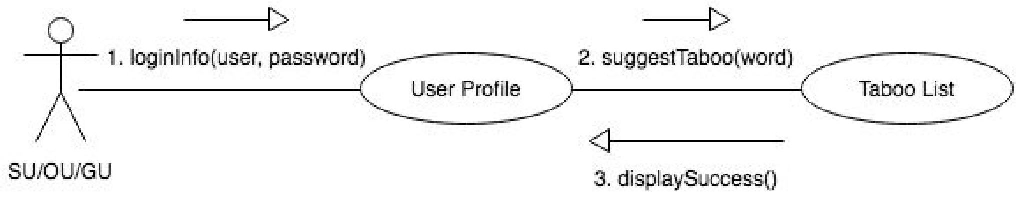
- Petri-net



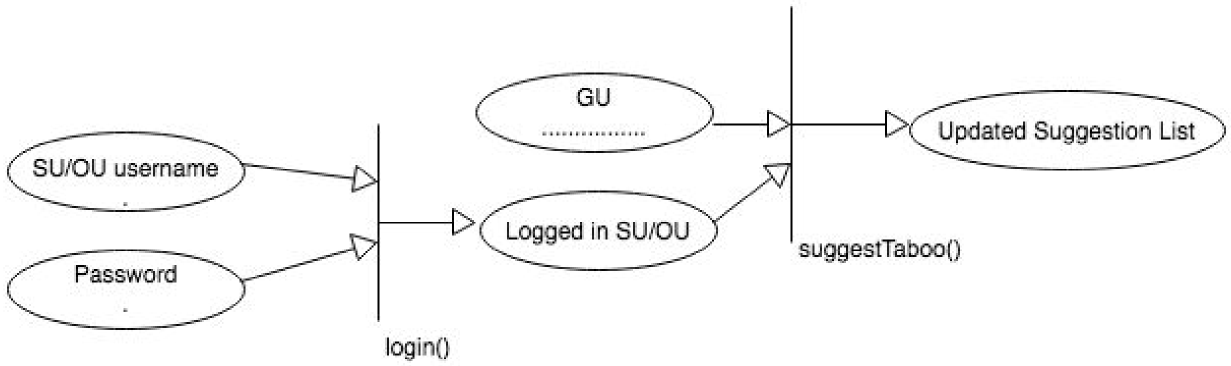
Send suggestions about taboo words:

- Scenarios
    - Normal
        - Any user can send suggestions of "taboo words" to the SU by clicking "suggest taboo". The suggested word will be appended to an existing global array of suggested taboo words.
    - Exceptional
        - If the suggested word already exists in the suggestion array, it will not be added.

- Sequence Diagrams



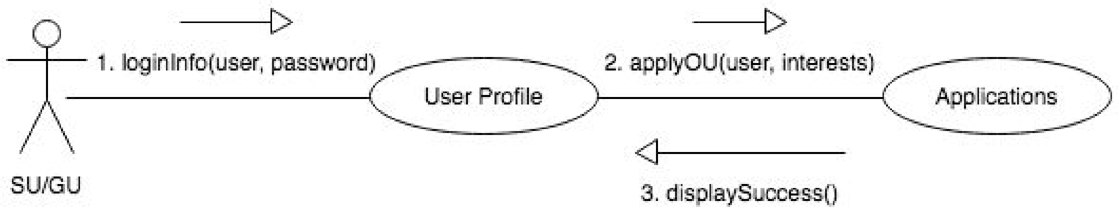
- Petri-net



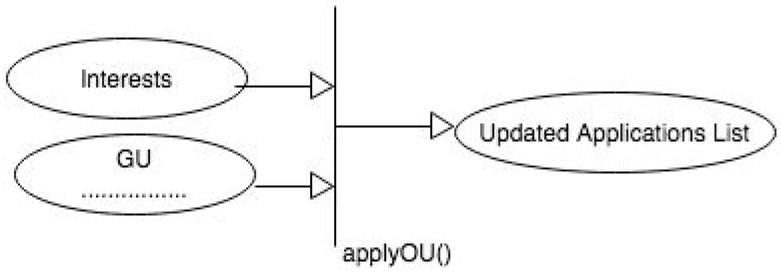
Apply to be an OU:

- Scenarios
  - Normal
    - A GU clicks “apply for OU” in their profile page. They are then prompted to input their technical interests. Upon submission, the application is saved in the application list for review by the SU.
  - Exceptional
    - A registered OU tries to apply to become an OU. This will be handled in the GUI level. The “apply for OU” button will only render for GUs and not for existing OUs.

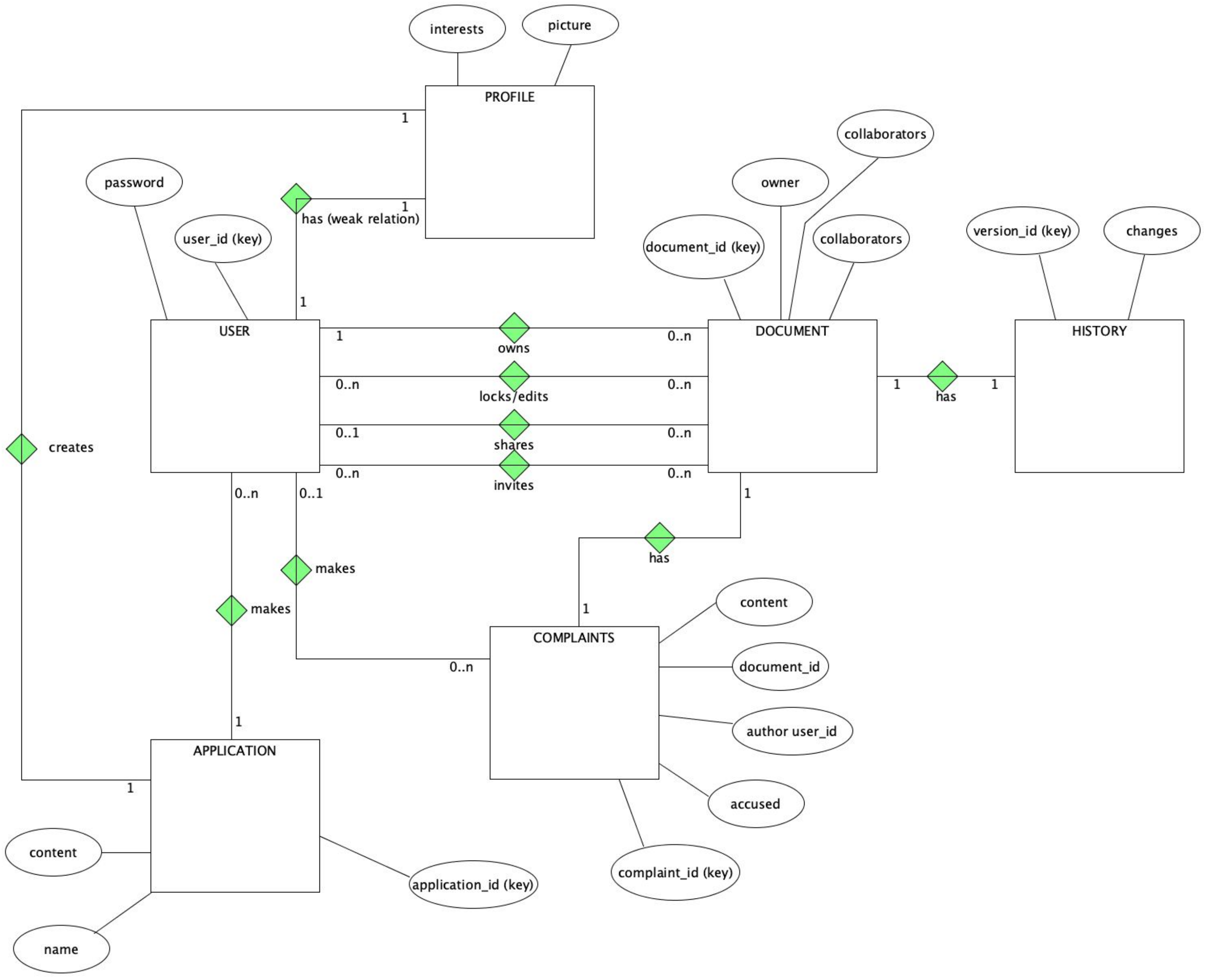
- Sequence Diagram



- Petri-net



3. E-R Diagram



interests
picture
PROFILE
collaborators
password
owner
has (weak relation)
user_id (key)
document_id (key)
collaborators
version_id (key)
changes
USER
DOCUMENT
HISTORY
owns
locks/edits
shares
invites
has
creates
makes
makes
has
COMPLAINTS
content
document_id
author user_id
accused
complaint_id (key)
APPLICATION
content
application_id (key)
name
1
0..n
0..1

4. Detailed Design (pseudocode)

USER CLASS METHODS

User Constructor:

Purpose: Initialize a new user object.

Inputs: name (string), password (string), privilege (categorical variable = SU, OU, GU)

Outputs: *none*

Pseudocode:

```
def __init__(name, password, category privilege)
    self.name = name
    self.password  = password
    self.privilege = privilege
```

Login:

Purpose: Update users logged in status

Inputs: password (string) inputted by user

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def login(self, password):
    if(self.password  == password):
        self.login = True
        return 1
    else:
        self.login = False
```

```
return 0
```

Update Profile (interests, picture):

Purpose: Update users interests and/or profile picture

Inputs: interests[] = a list of interests (strings), profile_picture (string) = path to file image,

Outputs: *none*

Pseudocode:

```
def update_profile(self, interests[], profile_picture):
    self.interests = interests[]
    self.profile_picture = profile_picture
```

Create Application:

Purpose: Create an application object

Inputs: desired_name (str) = desired username for OU, content (str) = argument for promotion

Outputs:

Pseudocode:

```
def apply_ou(self, desired_name, content):
    assert(self.privilege == GU)
    app = application(desired_name, content)
    return app
```

Update Membership:

Purpose: Change user privilege between SU, OU, GU

Inputs: categorical variable desired_privilege = SU/OU/GU

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def change_privilege(self, category desired_privilege):
    if(desired_privilege != self.privilege):
        self.privilege = desired_privilege
        return 1
    else:
        return 0
```

DOCUMENT CLASS METHODS

Document Constructor:

Purpose: Initialize a new document object.

Inputs: doc_name (string) = name of document, owner = user who created document (user),

share_type (categorical variable = Public/Shared/Private)

Outputs: *none*

Pseudocode:

```
taboo = [] #static class variable
taboo_suggestions = [] #static class variable

def __init__(doc_name, owner, category share_type):
    self.name = doc_name
    self.owner = owner
    self.content = []
```

```
        self.locked = False
        self.locked_user = self.owner
        self.share = share_type
        self.history = history()
        self.complaints = []
```

Add Taboo Word:

Purpose: Promote taboo suggestion to taboo master list

Inputs: suggest_index: index of suggestion to promote from tabbo_suggestions list

Outputs: taboo list is updated with new word, word is removed from taboo_suggestions list

Pseudocode:

```
def taboo_promote(suggest_index):
    taboo_suggestions.remove(suggest_index)
    taboo.append(taboo_suggestions[suggest_index])
```

Remove Taboo Word:

Purpose: Remove taboo word from taboo master list

Inputs: remove_word: word to be removed

Outputs: taboo list updated with targeted word removed

Pseudocode:

```
def taboo_remove(remove_word):
    taboo.remove(remove_word)
```

Lock/Unlock Document:

Purpose: Lock/Unlock document and save user who called method

Inputs: state (Boolean) = True/False , user (User) = A collaborator of this document instance

Outputs: self.locked is updated, self.locked_user is saved

Pseudocode:

```
def switch_locked(self, state, user):
    self.locked = state
    self.locked_user = user.name
```

Update Line:

Purpose: update an existing line in a document

Inputs: line_number (int) = desired line number to change, line_content (str) = content to replace line with

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def update(self, line_number, line_content):
    if(line_number is valid and line_content not in taboo):
        self.history.append([‘update’, line_number,  self.content[line_number])
        self.content[line_number] = line_content
        return 1
    else:
        return 0
```

Add Line:

Purpose: add a new line to the end of document

Inputs: line_content (str) = content to store in added line

Outputs: *none*

Pseudocode:

```
def add(self, line_content):
    if(line_content not in taboo):
        self.content.append(line_content)
        self.history.append(‘add’, length(self.content), line_content)
```

Delete Line:

Purpose: delete an existing line in a document

Inputs: line_number (int) = desired line number to delete

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def delete(self, line_number):
        if(line_number is valid):
            self.history.append(‘delete’ , line_number,
                        self.content[line_number])
            self.content.remove(line_number)
            return 1
        else:
            return 0
```

Share a Document:

Purpose: add a user to the list of collaborators for a document

Inputs: userid = valid key for a user in the system

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def add_collaborator(self, userid):
    if(self.share is applicable):
        if(self.collaborators does not contain userid):
            self.collaborators.append(userid)
            return 1
        else:
            return 0
    else:
        pass
```

Unshare a Document:

Purpose: remove a user from the list of collaborators on this document

Inputs: userid = valid key for a user in the system

Outputs: return of 1 or 0 indicating success/failure

Pseudocode:

```
def remove_collaborator(self, userid):
    if(self.collaborators does not contain userid):
        return 1
    else:
        self.collaborators.remove(userid)
```

Complain about an OU’s update:

Purpose: Complain about collaborators of document

Inputs: complainer (user) = user making complaint, accused (user) = user being complained about, reason (str) = reason for complaining

Outputs: self.complaints list appended to with new complaint object

Pseudocode:

```
def complain(self, complainer, accused, reason):
    self.complaints.append(complaint(complainer, accused, reason))
```

Search a Document for Keyword:

Purpose: Search document for Keyword

Inputs: key_word (string) = word to search for within document

Outputs: False if word not found, (x,y) integer tuple where x = line, y = word

Pseudocode:

```
def search(self, key_word):
    for line in self.content:
        for word in line:
            if(word == key_word)
                return (line,word)
    return False
```

HISTORY CLASS METHODS

History Constructor:

Purpose: Initialize a new history object

Inputs: *none*

Outputs: *none*

Pseudocode:

```
def __init__():
    self.content = {} #empty dictionary
```

Update History:

Purpose: Add a new entry to the Revision History

Inputs: type (categorical) = ‘update’, ‘add’, or ‘delete’, line_number (int) = line_number of document change takes place, content (string) = what what changed/added/removed

Outputs: self.content updated with new version code

Pseudocode:

```
def append(self, category type, line_number, content):
    generate valid version_code
    self.content[version_code] = [type, line_number, content]
```

Revert Document to Previous Version:

Purpose: Retrieve previous Version# changes

Inputs: desired version code to retrieve changes of

Outputs: list of changes which can be applied to a document object to display a previous version

Pseudocode:

```
def retrieve(self, version_code):
        return self.content[version_code]
```

COMPLAINT CLASS METHODS

Complaint Constructor:

Purpose: Initialize a new complaint object

Inputs: complainer (user) = user making the complaint, accused (user) = user being complained about, reason (string) = reason for complaint

Outputs: *none*

Pseudocode:

```
def __init__(complainer, accused, reason):
    self.resolved = False
    self.complainer = complainer
    self.accused = accused
    self.reason = reason
```

APPLICATION CLASS METHODS

Application Constructor:

Purpose: Initialize a new application object

Inputs: user (user) = a user instance to associated with the application, content (string) = reason for promotion

Outputs: *none*

Pseudocode:

```
def __init__(user, content):
    self.user = user
    self.content = content

    self.status = False
```

5. System Screens

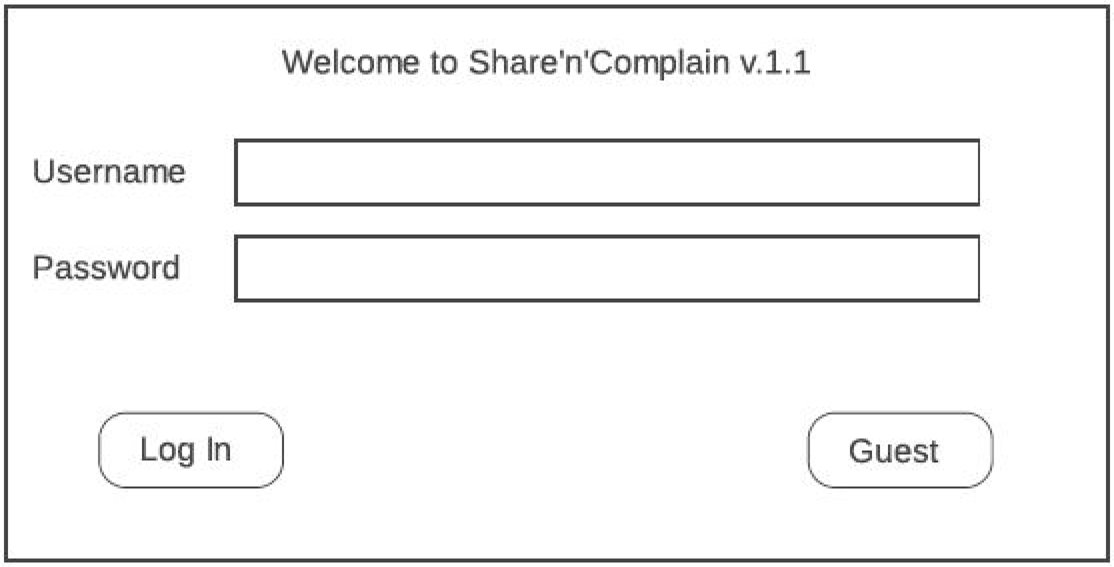


Fig. n, Login

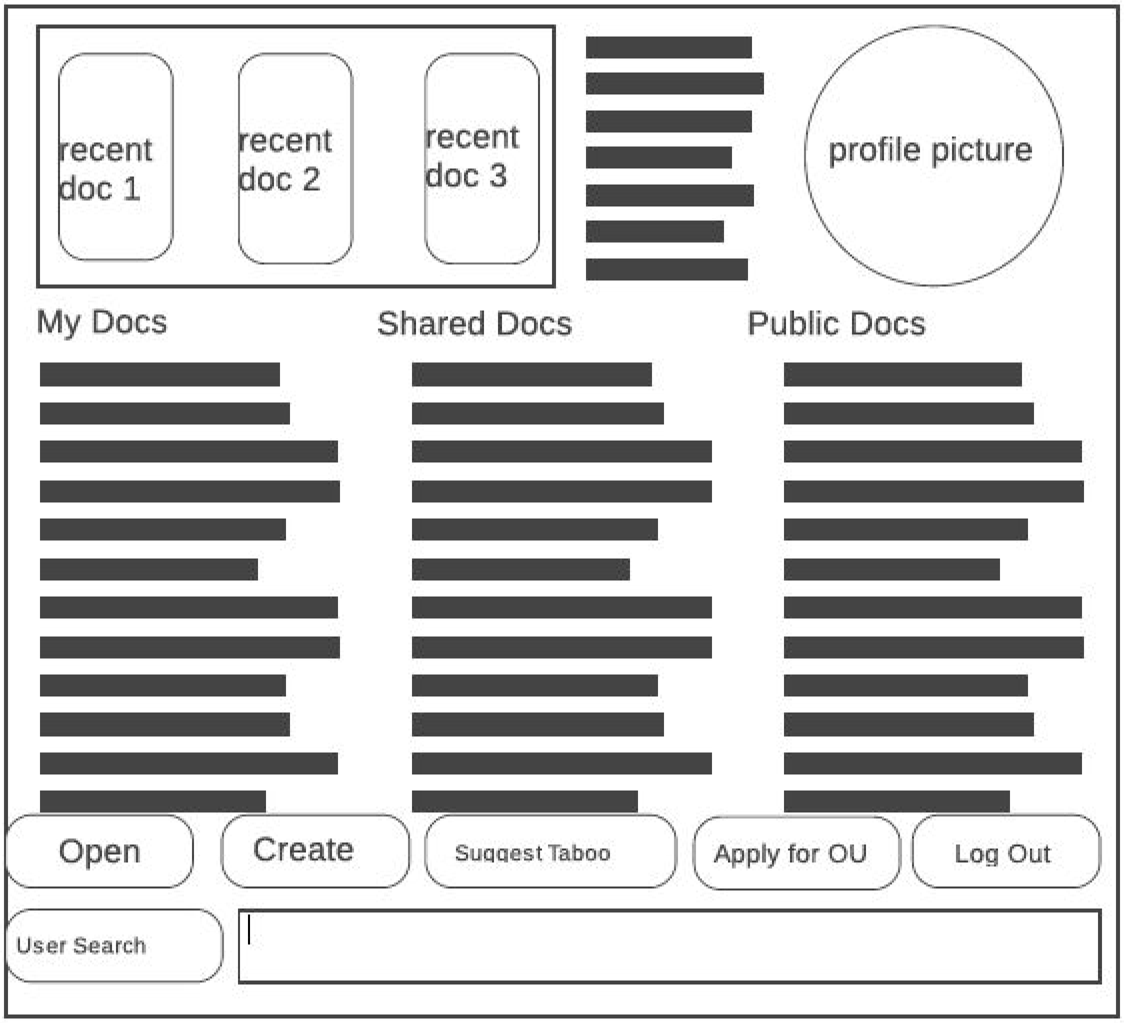


Fig. n, Profile page

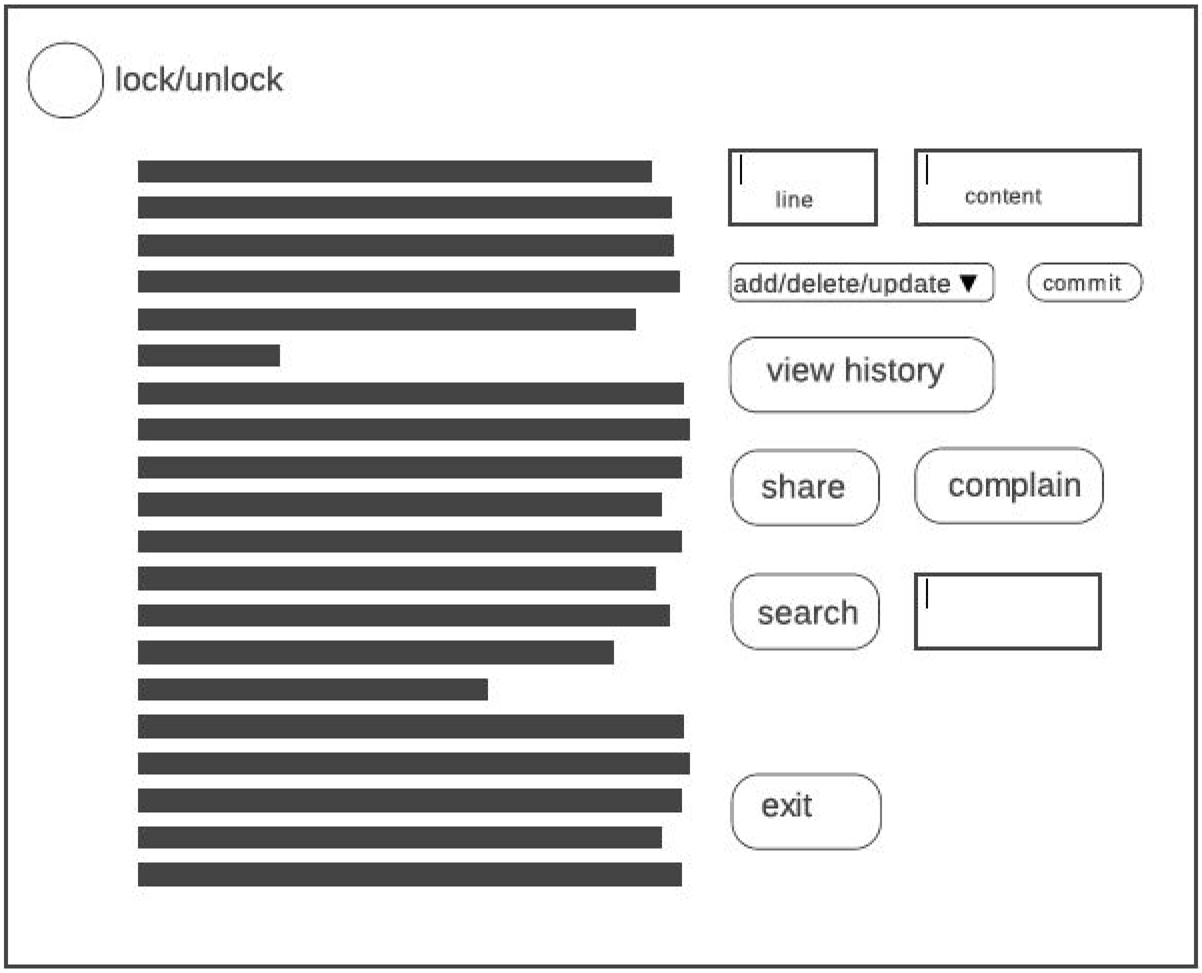


Fig. n, Document view

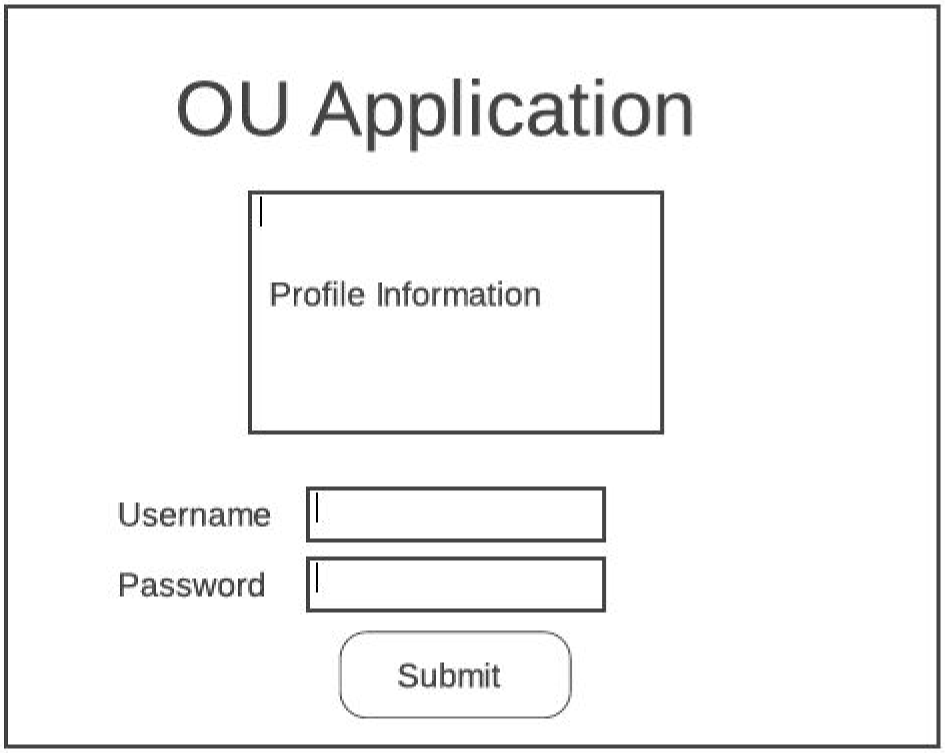


Fig. n, Application page

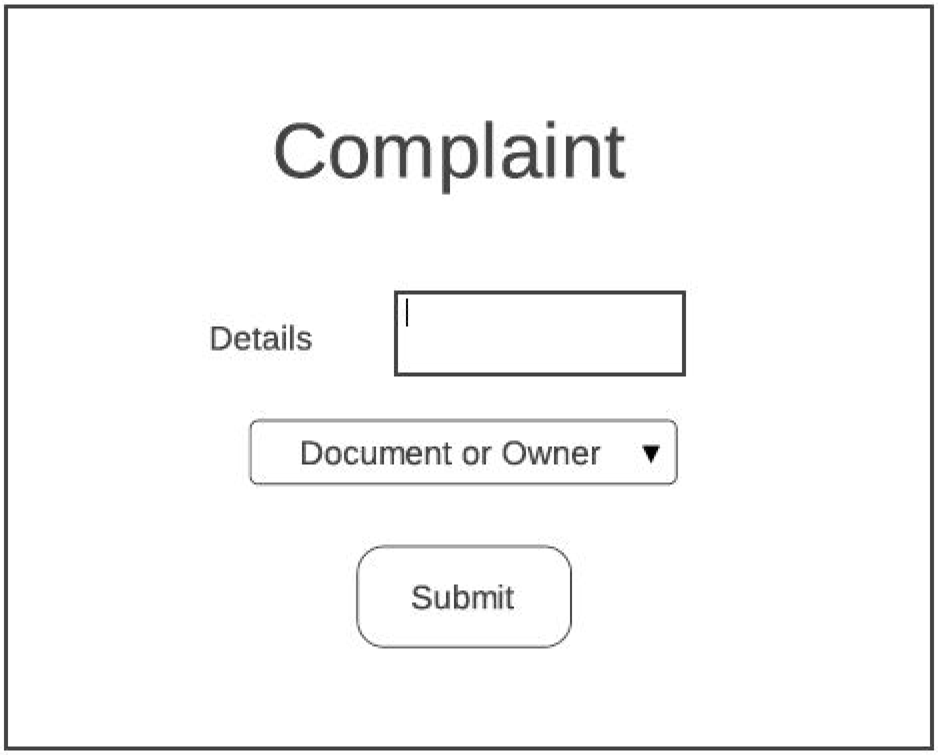


Fig. n, Complaint page

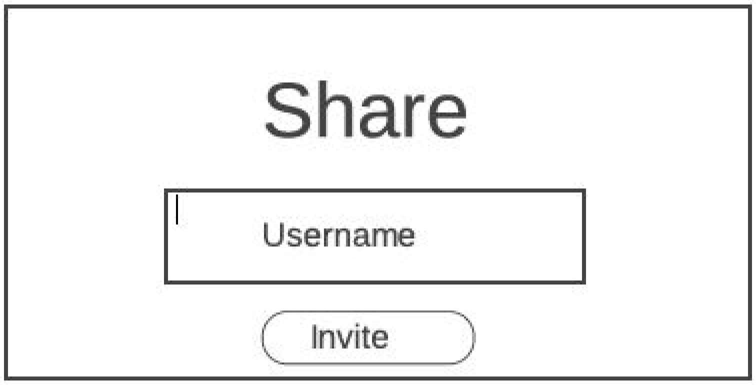


Fig. n, Share page

6. Minutes

11/9/18, 1:00 PM - 3:00 PM

- All members present.
- Design report review, general planning and improvement brainstorming.
- ER diagram discussion and development.
- Entity brainstorming.
- Use-case and functionality discussion.

11/16/18, 12:00 PM - 3:00 PM

- All members present.
- General check-in, review of required tasks.
- Collaboration class diagram whiteboarding.
- Digitization of collaboration class diagram.
- Use case sequence diagram development and digitization.
- Wireframe whiteboarding.
- Division of work for future meetings.

11/18/18, 12:00 PM - 5:30 PM

- All members present.
- Check-in and review of required tasks for the day.
- ER diagram whiteboarding.
- Digitizations of ER diagram.
- Pseudo-code method development.
- Wireframing digitization.
- Use case petri-net development and digitization.

7. Git Repository

Our Git repository can be found at: https://github.com/dvignoles/dss