dekū
System Design Document

Table of Contents

Page

1. Introduction 1

1.1. Purpose of This Document 1
1.2. References 1

2. System Architecture 2

2.1. Architectural Design 2
2.2. Decomposition Description 3

3. Persistent Data Design 11

3.1. Database Descriptions 11
3.2. File Descriptions 11

4. Requirements Matrix 12

Appendix A - Agreement Between Customer and Contractor 13

Appendix B - Peer Review Sign-off 13

Appendix C - Document Contributions 13

1. Introduction

1.1 Purpose of This Document

This document provides a high level and low level perspective into the functions of the dekū system. The high level documentation serves as a guide to any end users and/or customers. It provides a brief overview of how the different systems interact. The low level documentation serves as a guide to any programmer or developer of the system that is creating and/or maintaining it. There are workflows which show how the user can move through the system.

1.2 References

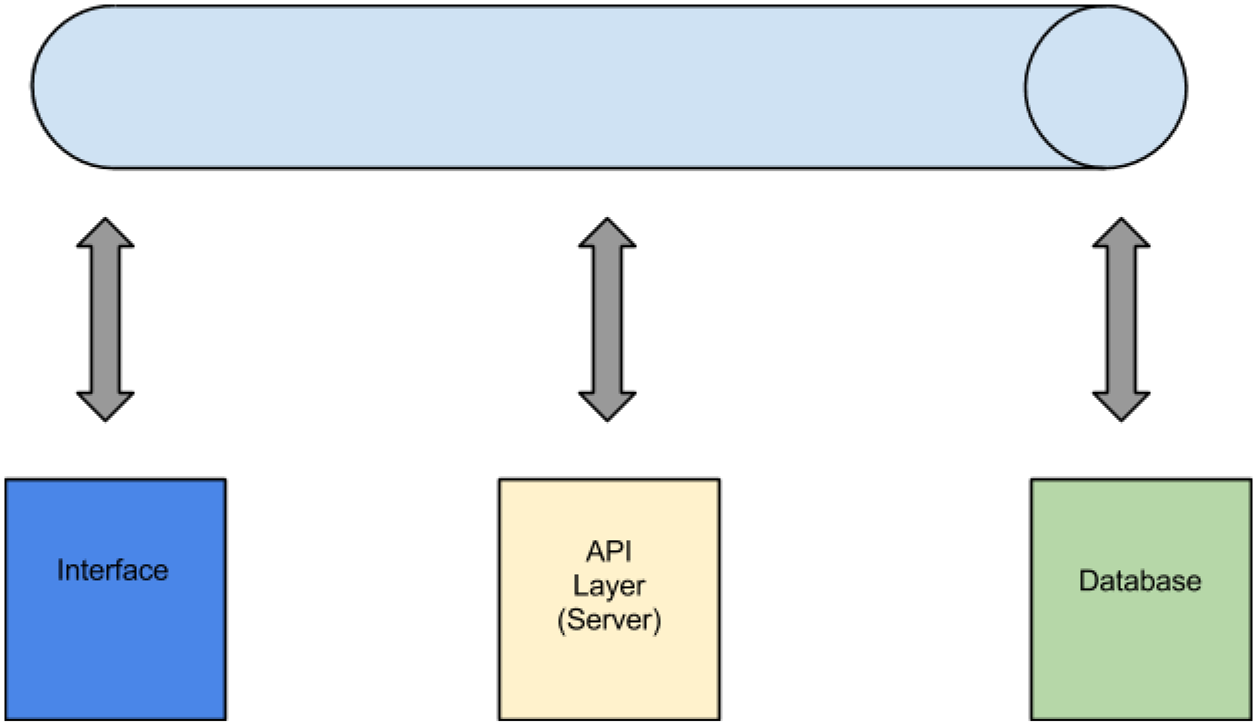
Here are links to the technologies that we are using for this application.

● Flask:	http://flask.pocoo.org
● Twitter Bootstrap:	http://getbootstrap.com/2.3.2/
● BackboneJS:	http://backbonejs.org
● Pydenticon:	https://github.com/azaghal/pydenticon
● Slidebar:	http://plugins.adchsm.me/slidebars/

All of the view and operation design structures were based on the use cases from the SRS. Refer to this document for use case numbers which will be used in the Requirements Matrix. The UI Design Document is an expansion of the information covered in 2.2.

2. System Architecture

2.1 Architectural Design



This application uses a Service Oriented Architecture (SOA). What this means it that any component of the application can be replaced, and the application will still run as needed. For example, if the application needed an Android or iOS front end, the interface is the only component that needs to be built, and the other parts will hook in.

The client-side portion of this application is divided into the functional and graphical sections. The functional component is written in BackboneJS which is a Model-View-Controller architecture that allows view logic, business logic and data storage to all be handled in an uncoupled manner. BackboneJS allows the front end to act primarily as a single page application (SPA). This reduces the number of transitions from page to page, keeping a more consistent and responsive user experience. Twitter Bootstrap will provide the groundwork for a responsive graphical interface that scales to device dimensions gracefully.

Our backend is a RESTful API built with Flask, a Python microframework. This setup allows us to easily swap out interfaces as needed, while still retaining access to the backend and database. The API will allow GET, POST, PUT and DELETE for internal app usage, and GET calls from external mediums. Users will be able to generate new content with in the application, but will be able to query it externally if they wish to. The database is a MySQL database with two tables: one for users and one for posts (referred to as cards).

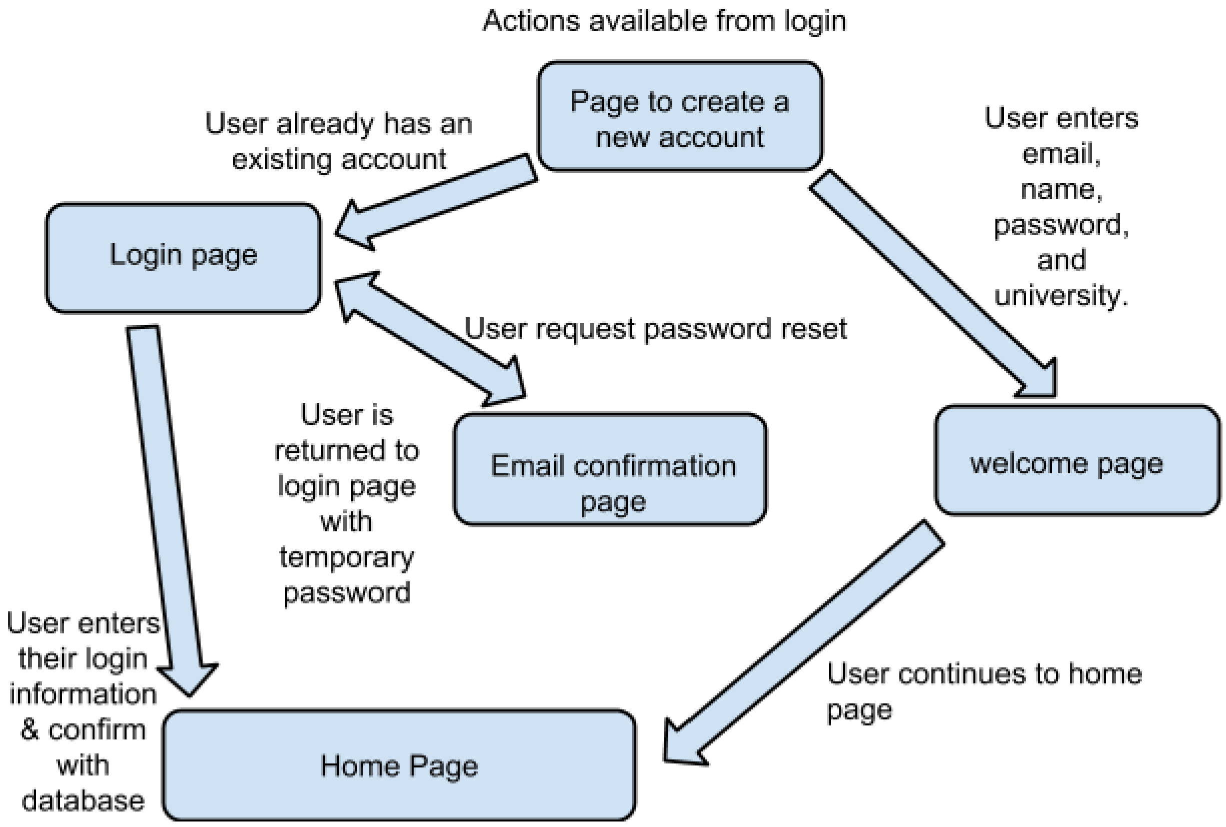
All of these will be running on a 32 bit Ubuntu Server installation. As stated above, any of these components can be substituted.

2.2 Decomposition Description

Everything on this application can be broken down into two groups: views and functions. Since we are running a minimalist architecture, the operations exist of a few unique views which allow the user to perform their functions. Functions are any actions that the user carries out on the site.

2.2.1 Sign in View

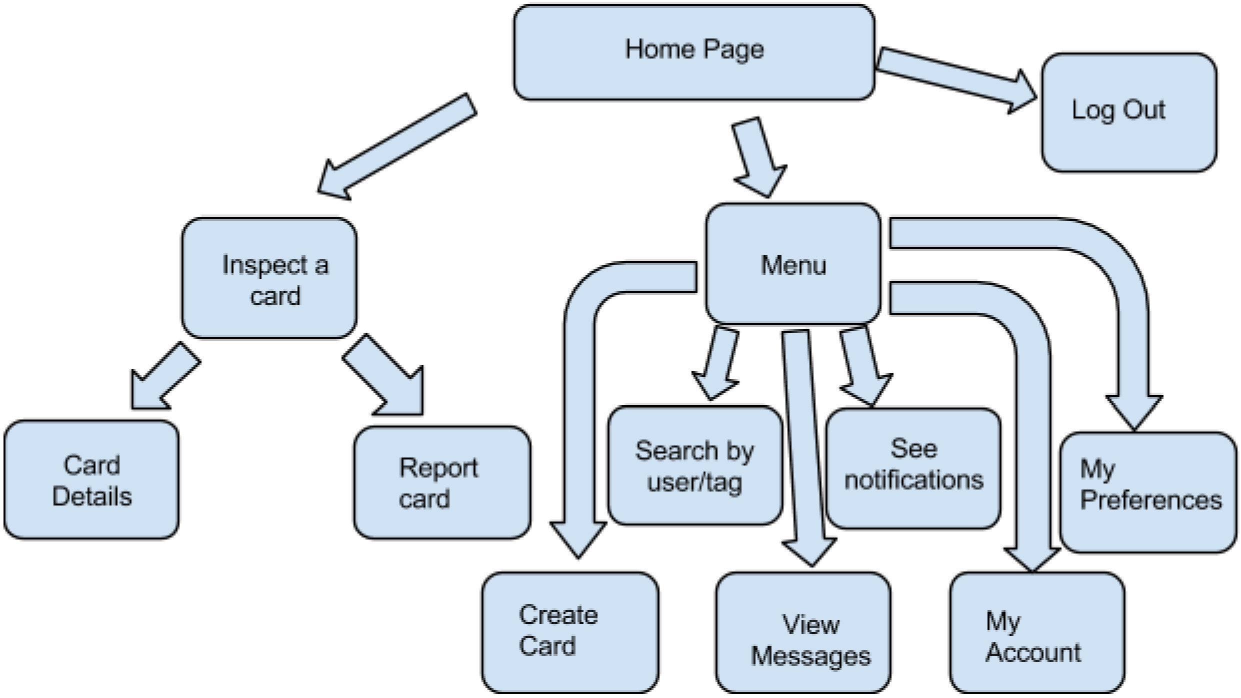
When a user goes to the website, they will first reach the sign in page. They will initially see the page to sign up or the site. Creating an account requires the user to enter their first name, last name, email, which university they attend, and their desired password. If they already have an account they can click the Login for existing users. In the existing user view they can enter their username and password, or if they have forgotten their password, they can click the forgot password button. The user can request for their password to be reset by sending a temporary password to their email. The end state is the home view.


Actions available from login
Page to create a new account
User already has an existing account
User enters email, name, password, and university.
Login page
User request password reset
User is returned to login page with temporary password
Email confirmation page
welcome page
User enters their login information & confirm with database
User continues to home page
Home Page

2.2.2 Home View

This is the main view that the user will spend their time on. From here the user can see all of the cards that are shared through the campus community. Here they can inspect a card which will bring up additional information about that card. They can go to the profile view for their account or for the profile of any other user simply by clicking on the avatar/name of that user. They can also go to the menu view which will allow the user to create a card, search, check messages, check notifications, view their account, and change their preferences. In the home view, the cards will be scaled to different sizes, but they will all share the same aspect ratio. Since all the cards on this site are public, there needs to be some way to signify popularity other than resharing, because that will clutter up the feed. If a card is more popular, it will be larger in the feed by a moderate amount. There will be three tiers of sizes to make sure that users can quickly tell the popularity without the content becoming overwhelming. This will be handled by a script that extracts statistics on the card from the database, and the upgraded size will be based on for Bootstrap to configure.

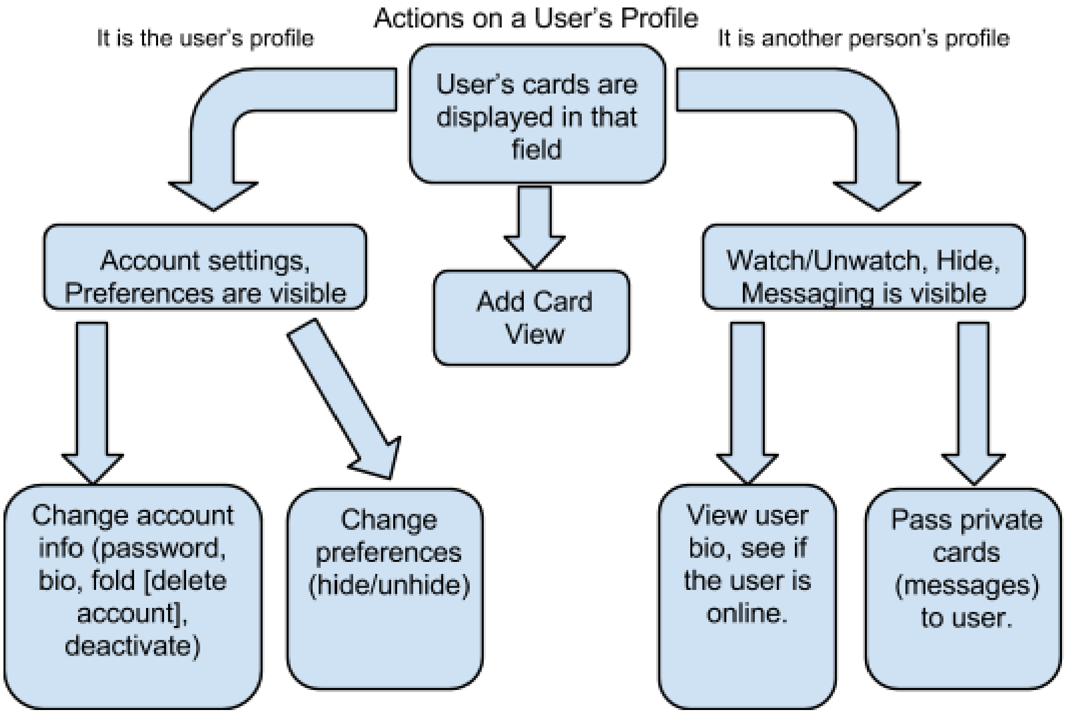
Actions available on home page.



2.2.3 Profile View

This is the second important view in the system. These views are dynamically generated using an account bio and taking cards for the database with tags that match this user. Depending on whether it is the user's account or someone else's, the view will have different options. A user can edit the account (password, email, bio) by and all those settings are saved to the database.

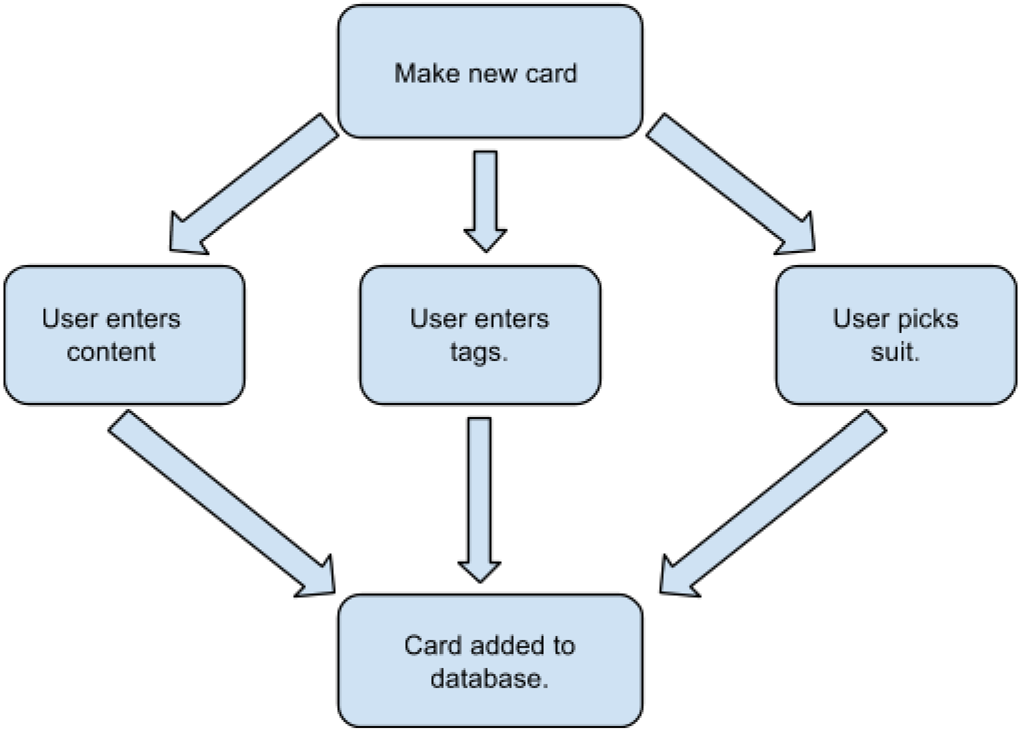
In addition, they can adjust their preferences which will change their hide list and watch list. Every user will be given a reputation that is calculated by the API and stored on the database. There is an aggregated rank and categorized ranks based on the following: how many watch you; how many of your cards are in other people's decks; how many total 'marks' do your cards have. Each category will have its own weighted value when it is totaled up for a user's aggregate score which can be seen on their bio tab. From the deck tab, the user can be sent to the Add Card View, and can search by tag, and receive notifications.


Actions on a User's Profile
It is the user's profile
It is another person's profile
User's cards are displayed in that field
Account settings, Preferences are visible
Add Card View
Watch/Unwatch, Hide, Messaging is visible
Change account info (password, bio, fold [delete account], deactivate)
Change preferences (hide/unhide)
View user bio, see if the user is online.
Pass private cards (messages) to user.

2.2.4 Add Card View

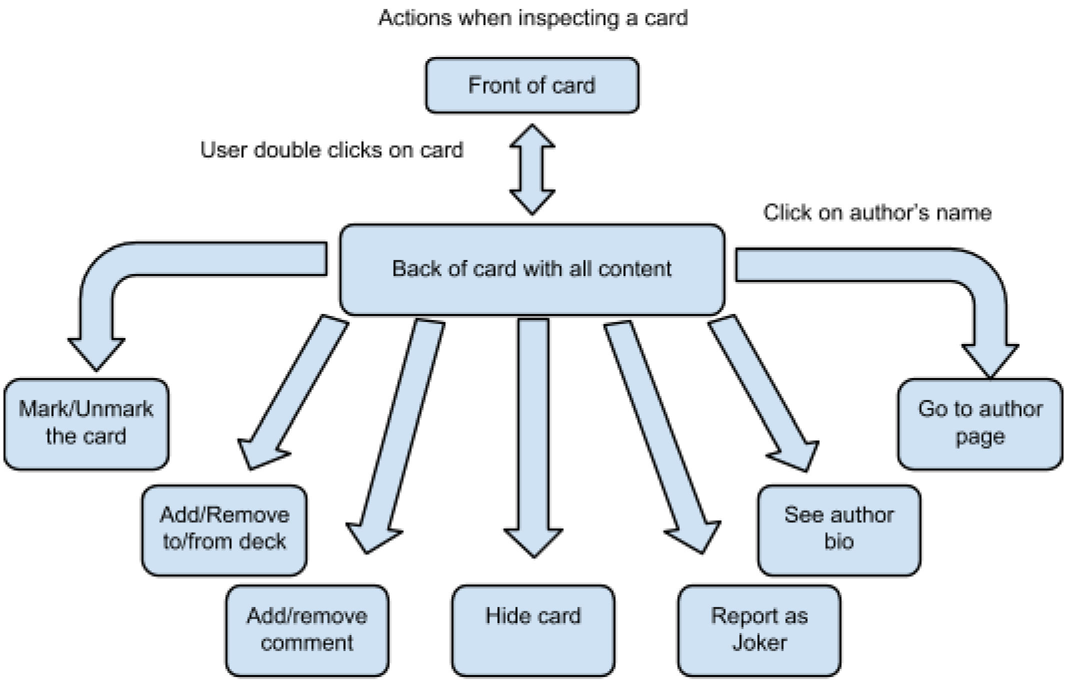
In this view the user can create a new card to add to their deck. This card is required to have a suit to determine its category, and must have at least one tag. The set of possible tags is stored on the database and the user will be provided suggestions when adding tags; it is recommended at least one tag be location specific. Finally, the user simply needs to enter the content within the hard character limit (currently 240 characters) using only text, no upload. This card is processed by the server and stored in the database. The view will update to show the new card in the feed so it is public to the whole school. The card will have the appropriate suit and a color-coded header that will match with the tags of the card. This is all handled by the Twitter Bootstrap.

Actions about making a new card.



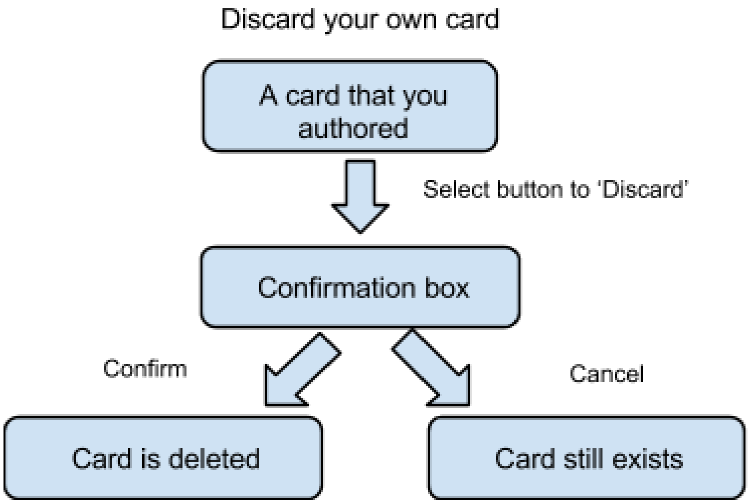
2.2.5 Inspect Card View

The home view is populated with the front of cards. Double click to go to inspect view. This will include the number of people who marked the card, the number of people who added it to their own deck, and an ability to directly comment on the card or to other users in a single thread format. All these actions can be undone by a user, and the database will store/remove the data accordingly. In addition, the user can hide this card, report it as a Joker so it can be removed, see the author bio through a side panel swipe, or go to that users profile page. Once they are done, just double click to return to the home view.



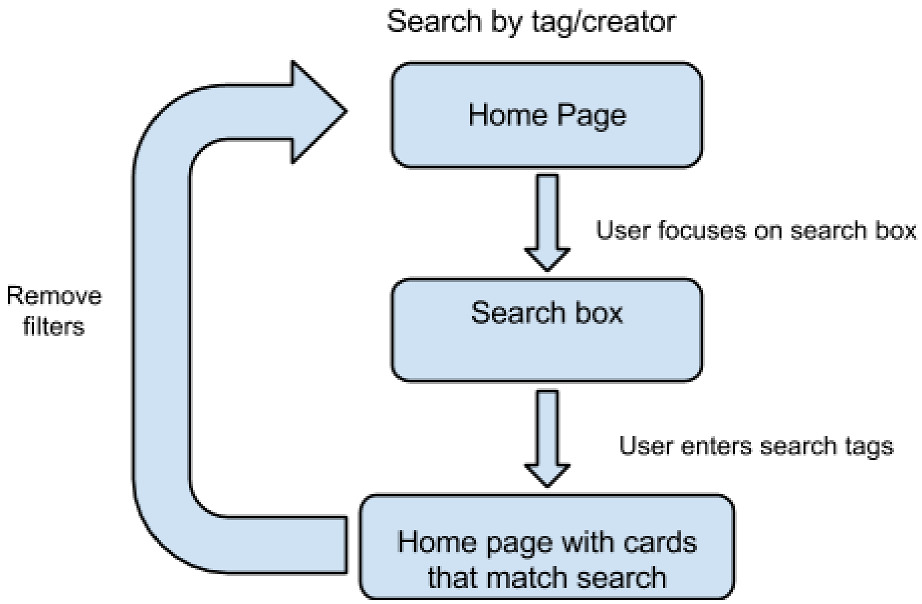
2.2.6 Discard Card Operation

Users have the ability to undo their actions, and this includes the ability to delete a card from their deck. If they are inspecting a card that they authored, the user will have the added option to 'Discard'. Upon confirming the action, that card is deleted from the users deck. If that was the only place that card existed, then there would be no further trace of it. However, if other users have added this card to their deck, it will be visible there. It will be listed as have a 'Null Author'



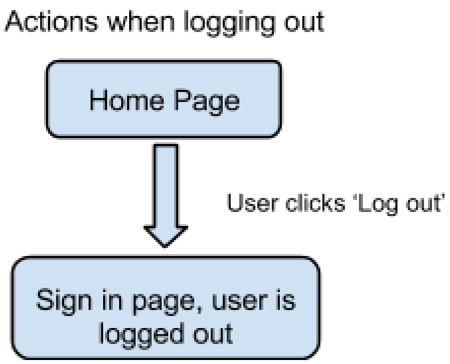
2.2.7 Search Operation

From the home view the user can swipe from the left to get a menu with a search function. They can search for cards by tag, author or category. The API retrieves matching content from the database and the front-end will adjust to show only this content. The user can clear these filters.



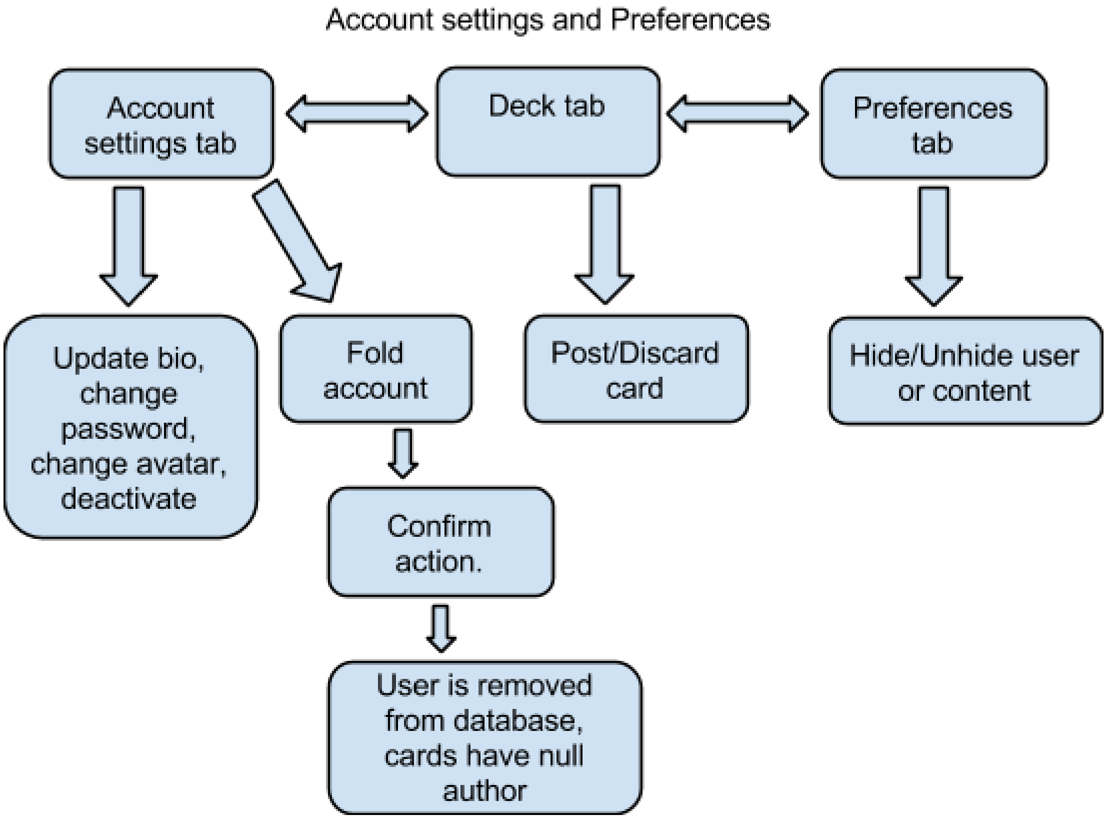
2.2.8 Logout Operation

On the top, by the user's avatar, there will be an icon that will allow the user to log out of their account. It will log them out from the API and send them to the Sign In View.



2.2.9 Account Operations

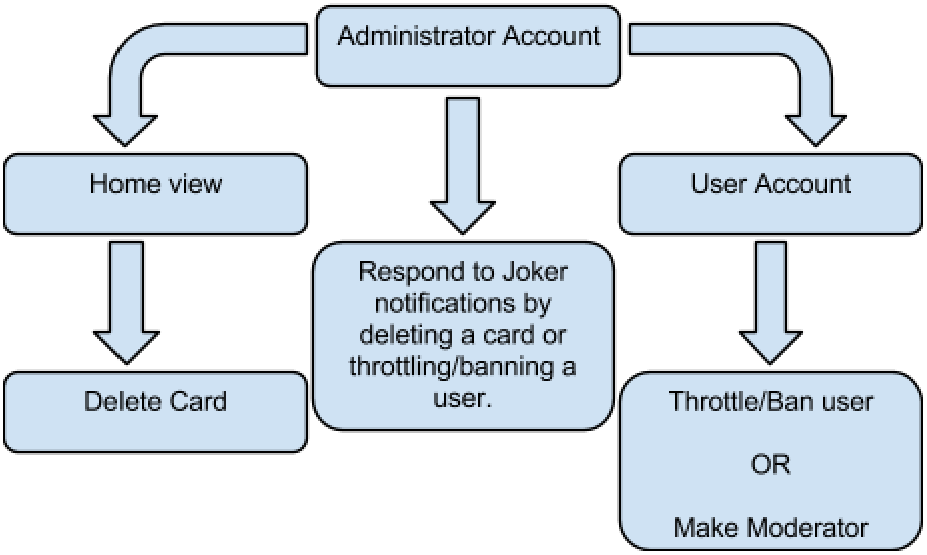
When on their own profile page, a user will have the ability to enact changes to the account and preferences settings. Account settings contain any information that is used in public content or the user's security information. Their default avatar is from Pydenticon, but the user can change it. The user can change their password or email, and these will be updated in the database. They can also deactivate their account or fold (delete it). Folding will remove them from the database completely, while deactivating will only prevent them from posting content until they login again. Preferences will contain customizable content such as the user's desired color scheme, a watching list, and a hiding list. All changes will be set for storage in the database.


Account settings and Preferences
Account settings tab
Deck tab
Preferences tab
Update bio, change password, change avatar, deactivate
Fold account
Post/Discard card
Hide/Unhide user or content
Confirm action.
User is removed from database, cards have null author

2.2.10 Administrator Operations

So far, everything that has been covered are general actions that any user can perform. The administrator for each campus dekū site will have extra control over content control. Their primary function is to ensure that cards and users are not offensive or spam. If that is the case, the administrator has two courses of action. For a first offense, a user's content is throttled. If that does not solve the issue, that user can be banned from the site. Individual users are able to report cards and users as Jokers, and request they be removed from the deck. It is the administrator's responsibility to process those reports and decide if they should act on it. In addition, they can assign a user to be upgraded to a moderator to help them manage the tasks. In the different views, an administrator may see several extra options, such as: 'Make moderator', 'Delete card', or 'Throttle user'. This added privilege is stored on the user table in the database, and there can only be one administrator per university.

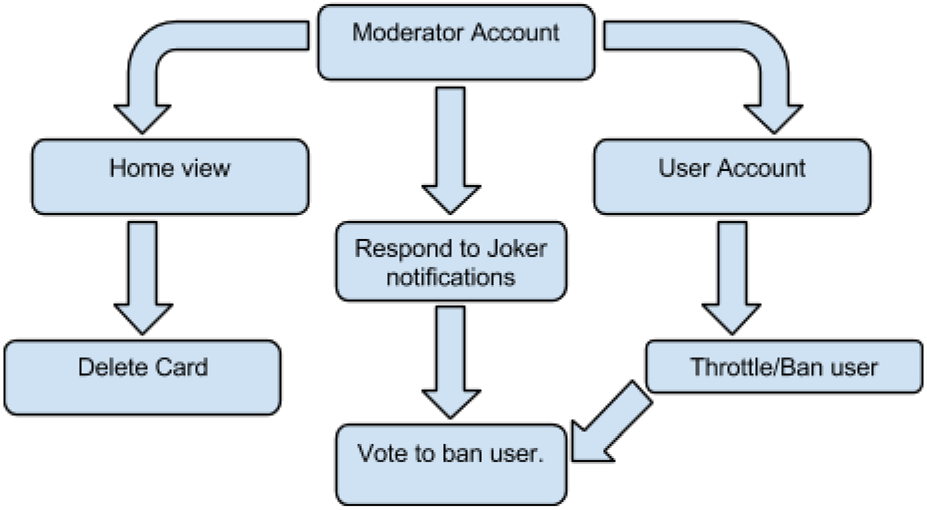
Actions the Administrator can perform



2.2.11 Moderator Operations

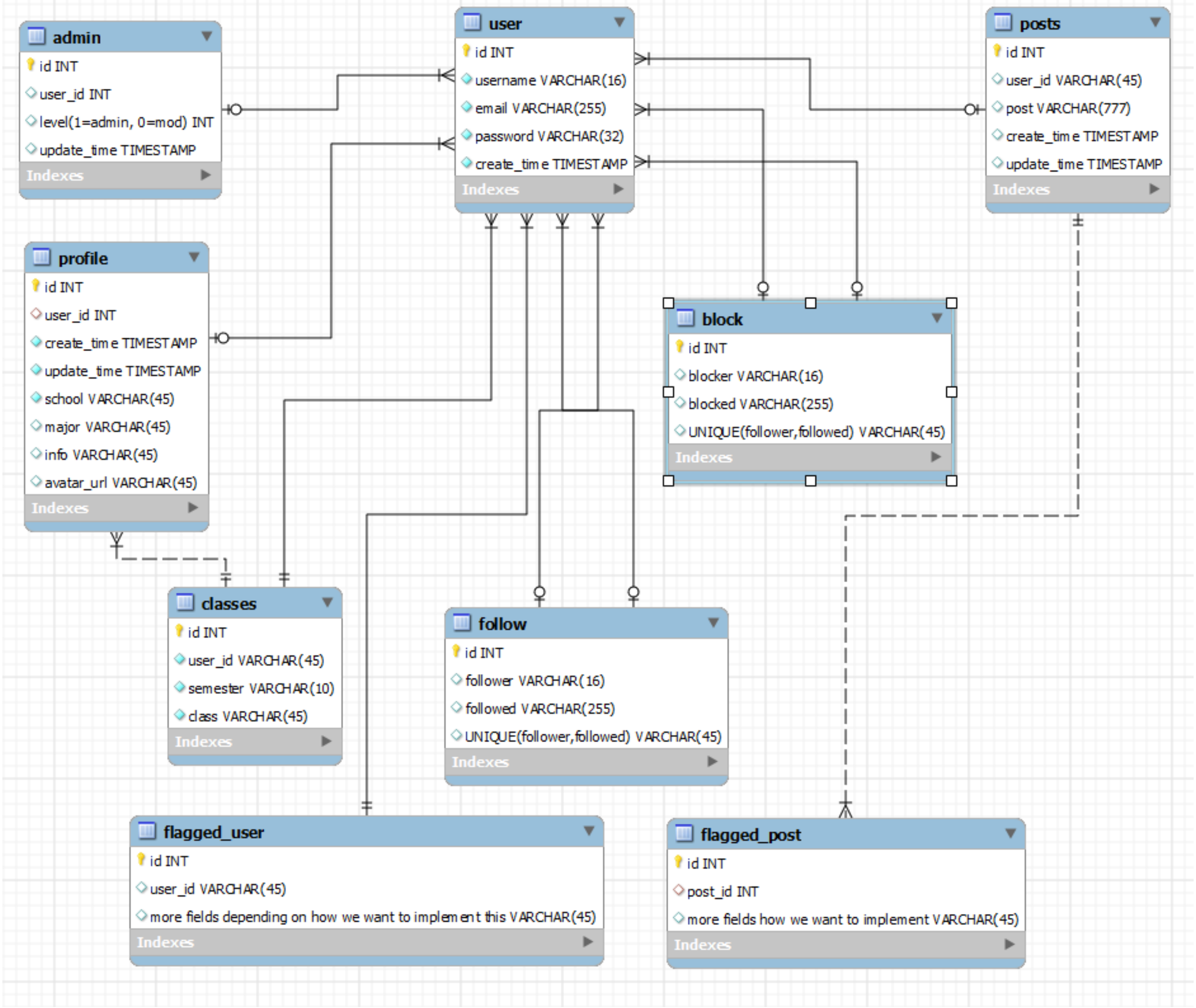
Moderators have virtually all the same operations that can be done as an administrator, except they cannot assign moderators, though they can recommend them to the administrator. They are responsible for dealing with spam and banning offensive users. In order to process the banning request, there will be a vote among the moderators on whether or not to proceed. There must be three of a kind for the user to be banned. If the vote is not successful within 48 hours, the request is dropped from the system. They will have similar additions to their view as do administrators.

Actions the Moderators can perform



3. Persistent Data Design

3.1 Database Descriptions



3.2 File Descriptions

More information on actual implementation will be included as the files begin to actually be made. At this time the descriptions will be mostly overviews of what the file will do.

Card class - This file would contain all the data for information and the method handling of the cards. It would have a method editing and deleting, be of its own type and contain mostly strings of descriptions, IDs, links, and an image for the front of the card. It would also have an object helping define which subsections the card falls under (student affairs, administrative, etc).

Deck class - This file would be a handler class for cards. This is what links to and interacts with the users account for the user functions such as search or save cards. The deck class will hold the searching function for other users to search through the database.

User class - This will hold the functions for user interactions. It will link to the database and the deck file to handle information displaying on a persons account. It will also have a way to distinguish between common user, moderator, and admin, and whether a person is being throttled back for spamming too many posts.

Describe the file(s) used by the system. Include a diagram of the file structure. For each field in a file, give its name, data type (e.g., int, double), size (e.g., strings), and description of what it represents. Basically, give all of the information that a programmer must have to implement the file. If no files are used, simply state so. Supplement your description with a sample file(s). [Length is whatever it takes]

4. Requirements Matrix

SRS Requirement	**SDD Component**
1	2.1.1
2	2.1.1
3	2.1.1
4	2.1.9
5	2.1.9
6	2.1.2, 2.1.3, 2.1.4
7	2.1.2, 2.1.3, 2.1.7
8	2.1.2, 2.1.3, 2.1.5
9	2.1.3
10	2.1.3
11	2.1.3, 2.1.5
12	2.1.2, 2.1.3, 2.1.5, 2.1.6
13	2.1.2, 2.1.3, 2.1.9
14	2.1.3
15	2.1.2, 2.1.3
16	2.1.2, 2.1.3, 2.1.8
17	2.1.4
18	2.1.1, 2.1.9
19	2.1.2, 2.1.3
20	2.1.4
21	2.1.4
22	2.1.10
23	2.1.11
24	2.1.4

Appendix A - Agreement Between Customer and Contractor

The customer for the dekū social content and sharing site has agreed to this site and all its functionality, including creating accounts within a university context, sharing public content based on location, and being able to mark costs and pass messages through them. This document includes use cases for all of the functional requirements, as well as a collection of the non-functional requirements.

If changes to the requirements develop in the future, this document will be amended and all parties will sign off on the document to make sure that the changes are in accordance with the wishes of the customer.

Client

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Team

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Name ____ Date __________________
Print
Name ____ Date _________________

Appendix B - Team Review Sign-off

This document affirms that all of the members of this team have contributed to and reviewed the material within this document. Any minor disagreements between members are listed below.
Team

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Comments____
____

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Comments____
____

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Comments____
____

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Comments____
____

Name ____ Date __________________
Print
Name ____ Date __________________
Signature

Comments____
____

Appendix C - Document Contributions

Identify how each member contributed to the creation of this document. Include what sections each member worked on and an estimate of the percentage of work they contributed. Remember that each team member must contribute to the writing (includes diagrams) for each document produced.