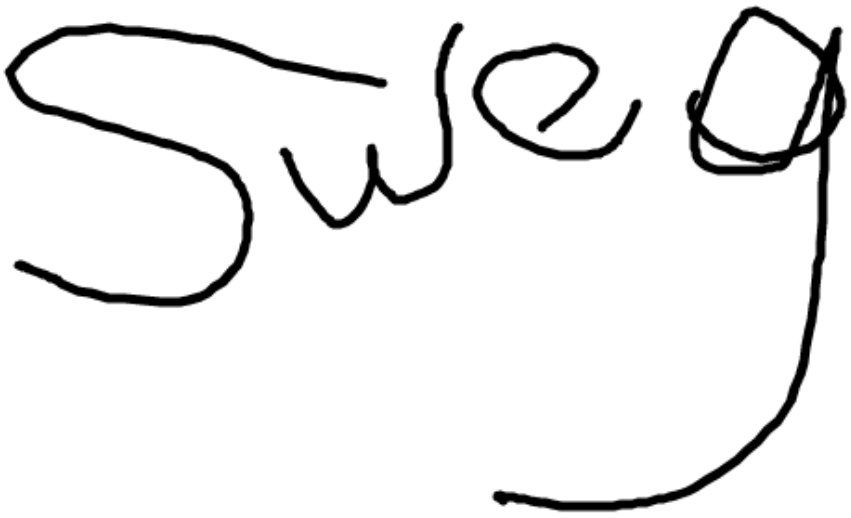
Design Use Cases

Students With A Goal (S.W.A.G.)

June 5, 2015



Melody Jeng	System Architect
Arno Gau	Senior System Analyst
Rachel Lee	Software Development Lead
Laura Hawkins	Project Manager
Rohan Rangray	Algorithms Specialist
Andrew Buss	Database Specialist
Phuong Tran	Quality Assurance Lead
Chung Kang Wang	Business Analyst
Masud Rahman	User Interface Specialist
Kevin Mach	User Interface Specialist

Contents

1 Accounts **3**
- 1.1 [A1] User Registration . . . 3
- 1.2 [A2] User Login . . . 5
- 1.3 [A3] Add a Class . . . 6
- 1.4 [A4] Drop a Class . . . 8
- 1.5 [A5] Password change . . . 9
- 1.6 [A6] Password reset . . . 11
- 1.7 [A7] Limit Student Access to Courses . . . 13
- 1.8 [A8] User Logout . . . 14
- 1.9 [A9] Contact Admin . . . 15

2 Flashcards **16**
- 2.1 [F1] Push Flashcard . . . 16
- 2.2 [F2] Edit Flashcard . . . 17
- 2.3 [F3] Pull Flashcard . . . 18
- 2.4 [F4] Flag Inappropriate Cards . . . 19
- 2.5 [F5] Filter Flashcards . . . 20
- 2.6 [F6] Blank Out Words in Flashcard . . . 21
- 2.7 [F7] Fix Flashcard . . . 23
- 2.8 [F8] Hide Cards . . . 25
- 2.9 [F9] View a Feed . . . 26

3 Decks **27**
- 3.1 [D1] Remove a card from a deck . . . 27
- 3.2 [D2] Viewing cards in deck by pull time . . . 28

4 Review **29**
- 4.1 [R1] Study Deck . . . 29
- 4.2 [R2] Review Notification . . . 31
- 4.3 [R3] Configure Account Notifications . . . 32

1 Accounts

1.1 [A1] User Registration

Description	To create flashcards and save them into decks, the system requires the user to create an account prior. The user will be able to create an account by registering.
Desired Outcome	An account will be created for the user from the specified username, password, and email. The user will be able to log into the aforementioned account.
User Goals	The user shall have an account to participate in the application's activities.
Primary Actor	User (student)
Dependency Use Cases	None
Priority Level	"Must"
Status	Implemented
Pre-conditions	None
Post-conditions	The user has an account registered with the system.
Trigger	The user wants to create an account.
Workflow	1. The frontend renders the login form described in login.html. 2. The user shall click the sign up tab. 3. The frontend shall render a registration form and display it to the user. 4. The user shall fill in the form. 5. The frontend checks that the provided email is not invalid, and that the password is not invalid. If anything is not valid, it displays an error and returns this to the user. 6. If the form is valid, the frontend POSTs the form in JSON format to the backend API at /api/verify_email/. 7. The backend deserializes and validates the data in /api/register. If it's not valid, it returns an error to the frontend. 8. The backend creates a new user object from the provided data, filling in the email and password fields in models.py in model method create_user.

9. The backend saves the user object in models.py.
10. The backend marks the user's email as unverified.
11. The backend sends an email to the user's address with a link to validate the user's email address from models.py in function send_confirmation_email.
12. The backend responds with success and logs the user in on a new session for the User.
13. The frontend reports success to the user and stores the sessionid for future requests.

Verification Workflow:

1. The user receives an email with a link to verify their email. The user clicks the link
2. The frontend in VerifyEmailController submits a POST request to the backend.
3. The backend validates the code in views.py in function veryify_email. If the code is not valid, it returns an error to the frontend.
4. The backend marks the user's email as confirmed and saves the user object again in models.py in function confirm_email. The user's account is now active.
5. The frontend's VerifyEmailController.js reports success and logs the user in to the application.

1.2 [A2] User Login

Description	The user is able to login to personal account.
Desired Outcome	The user shall be able to provide their username and password to access their courses and flash cards. The user shall gain access to the list of the courses they have added, and be able to review the flash cards they have added to their decks.
User Goals	The user wants to resume use of the site on another device, or use the site after logging out.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration
Priority Level	"Must"
Status	Implemented
Pre-conditions	• The user has registered an account. The user knows their email and password.
Post-conditions	• The user can navigate the site and see their data.
Trigger	The user wants to use the application and its core features.
Workflow	1. The frontend shall render the login form described in login.html and implemented in LoginController.js 2. The user shall type in their email and password. 3. The backend shall verify that the user has submitted their email and password in views.py in function login. 4. The backend shall check that the user is active in views.py in function login. 5. The backend shall log the user in using the Django login feature. 6. The frontend shall displays the home page for the user.

1.3 [A3] Add a Class

Description	The user shall be able to add a class and access the flashcards associated with that class.
Desired Outcome	The class shall be connected to the student's account and the user shall have the ability to read and publish flashcards for this class.
User Goals	The user wants to publish flashcards for this class, and read the flashcards for this class that are published by other users.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login
Priority Level	"Must"
Status	Implemented
Pre-conditions	• The user has registered an account. • The user has successfully logged in to the application.
Post-conditions	• The user can view the flashcards being published for the class and add them to their deck. • The user can publish flashcards to be viewed and added by other users in the same class.
Trigger	The user wants to make flashcards and view flashcards for a class.
Workflow	1. The user shall click "Add a Class" from the "Classes" drop down menu in the upper left hand corner. 2. The frontend shall navigate to the new page described by addclass.html. 3. The user shall begin to type the class department code, course title, course number, or instructor in the text box. 4. The frontend in ClassAddController.js shall send a get request to the backend at api/sections/search/. 5. The backend shall search the database for the classes matching department code, course title, course number, or instructor in models.py in function search. 6. The frontend shall show the classes that the backend has found. 7. The user shall select from the classes the one that they want to add. 8. The frontend shall enable the 'Add Class' button. 9. The user shall click 'Add Class' button to the right of the text box.

10. The frontend shall send a post request to api/sections/.
11. The backend shall in views.py in function enroll shall call the method enroll in models.py.
12. The backend in models.py in function enroll shall add the class to the User's sections.
13. The frontend shall redirect the user to the class's Live Feed using the ClassAddController.js.
14. The frontend shall present the live feed for the class by using feed.html and FeedController.js.

Alternate Workflow: User is whitelisted.

Precondition: The class has a whitelist, a whitelist is a list of email addresses that can add themselves to the class.

4. The user shall select their desired class, which has a whitelist, by clicking on "Add Class".
5. The frontend shall submit a POST request to /api/me/sections with the course ID that the User typed.
6. The backend shall check if there is a whitelist for the class that the user has selected in models.py at function is_whitelisted.
7. The backend shall check if the user is on the whitelist in models.py in function is_user_on_whitelist.
8. The backend shall add the user to the class following steps above.
9. The frontend shall redirect the user to the class's live feed followed the last step in the Workflow above.

Alternate Workflow: User is not whitelisted.

Precondition: The class has a whitelist, a whitelist is a list of email addresses that can add themselves to the class.

8. The backend shall will check that the user is in the whitelist in models.py.
9. The frontend shall display that the class is whitelisted and the Add Class button will not be enabled.

1.4 [A4] Drop a Class

Description	The user can remove themselves from a course he/she is registered to
Desired Outcome	The user has dropped the class and no longer need access to the flashcards for said class. This means they can no longer see the flashcards associated to the class that they dropped
User Goals	The user shall not have access to a class and the flashcards associated with that class. They will no longer see the class listed in their drop down menu.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class
Priority Level	"Should"
Status	Implemented
Pre-conditions	• The user has a valid account. • The user is logged in. • The user has enrolled in at least one class. • The user has navigated to the Settings page.
Post-conditions	• The user shall no longer have access to dropped class, or associated flashcards
Trigger	The user no longer wants to see flashcards from that class.
Workflow	1. The user shall click 'Drop' button to the right of the class they wish to drop. 2. The frontend shall submit a DELETE request to /api/me/sections/<section ID number>. 3. The backend shall check that the user is enrolled in the section in models.py in function drop. 4. The server shall remove the selected class from a user's list of section in models.py in function drop. 5. The frontend shall remove the class from the list in setting and show toast reading 'Dropped'.

1.5 [A5] Password change

Description	The user shall be able to change the password for their account in the account settings.
Desired Outcome	The user's password shall be changed and the User shall be able to log in with the new password in the future
User Goals	The User wants to use a different password in the future
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login
Details	The user provides their current password and a new password in a form. If the current password is correct, the system updates their password to the requested new password.
Priority Level	"Must"
Status	Implemented
Pre-conditions	• The user has registered an account. • The user has logged in. • The user knows their current password. • The user has navigated to the Settings page.
Post-conditions	• The user's password is changed • The user can log in with the new password
Trigger	The user wants to change their password.
Error Handling	**Missing Fields**: • The frontend shall enforce required fields • The backend will return an HTTP Bad Request error to the frontend. **Incorrect Old Password**: • The backend will return an HTTP Forbidden error to the frontend. **Blank New Password**: • The backend will return an HTTP Bad Request error to the frontend.
Workflow	1. The frontend shall render a form that requires the user to enter their old password and new password from settings.html

2. The user shall type in their current password, the desired new password, and retype the desired new password to confirm.
3. The frontend shall submit the data to the server by sending a PATCH request to /api/me containing "old_password" and "new_password" values in SettingsController.js.
4. (The)
5. The backend shall check the user's current password in views.py in function patch under UserDetail.
6. The backend shall update the user's current password in views.py in function patch under UserDetail.
7. The frontend shall present a toast reading "Password successfully changed" and redirect the user to the Add a Class page.

1.6 [A6] Password reset

Description	The user shall be able to reset their password without being logged in.
Desired Outcome	The user's password shall be changed to one that they remember.
User Goals	The user will be able to log into the site with a new password.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration
Details	The user provides their email in a password reset form. If the address is valid, the site sends a password reset link with a random token to that address. If the address is invalid, the site does not send a link. For security reasons, we do not reveal to the User whether the email address was valid, and we expire the link after 24 hours. When a user visits the link emailed to them, they are able to specify a new password in a form. When they submit the form, their password is updated if the token is correct.
Priority Level	"Must"
Status	Implemented
Pre-conditions	• The user has created an account before. • The user knows the email address attached to their account for our application. • The user can log into their email account.
Post-conditions	• The user's password is changed. • The user can log in with the new password.
Trigger	The user wants to log into the site but does not know their password.
Workflow	1. The User clicks "Forgot Password" text on the login page 2. The frontend shall render the password reset page from the template requestpasswordreset.html. 3. The user shall type in their email address for their account and click the "Reset" button. 4. The frontend shall send a POST request to /api/request_password_reset/ from RequestResetController.js. 5. The backend shall check if an account exists with the email entered by the User in views.py in function request_password_reset.

6. The backend shal send an email to the User with a link to the password reset page in models.py in function request_password_reset.
7. The user will navigate to their email account, open the email, and click the link.
8. The frontend shall render the password reset page from the template resetpassword.html.
9. The user shall type in their new password and re-type it to confirm it.
10. The frontend shall check if the new password is long enough and if the confirmed password is the same as the new password in ResetPasswordController.js.
11. The frontend shall send a POST request to /api/reset_password from ResetPasswordController.js.
12. The backend shall change the user's password in views.py in function reset_password.
13. The frontend shall redirect the User to the login page.
14. The user shall use their new password to login to the application.

1.7 [A7] Limit Student Access to Courses

Description	The instructor shall be able to limit access to the courses that they are in charge of by whitelisting only those students that are actually in the class.
Desired Outcome	The class will have limited access and only those Users who are whitelisted may enroll in the class.
User Goals	The instructor wants to limit access to their class so only the students who are actually in their class participate, and nobody else.
Primary Actor	Instructor
Dependency Use Cases	[A1] User Registration, [A2] User Login
Priority Level	"Should"
Status	Implemented
Pre-conditions	• The user has valid instructor's account
Post-conditions	• Only those users who were whitelisted may add the class.
Trigger	The course instructor wants to limit the students who can add their class on our application.
Workflow	1. The instructor emails the administrators from their UCSD email address and requests to limit access to their course. They provides a list of emails of the students that are to be whitelisted. 2. The administrators visit a custom admin page, select the course, and paste the list of emails. They submit the page directly (no separate frontend here) to the server. 3. The backend shall create a WhitelistedAddress for each provided email, attaching it to the section taught by the instructor. 4. The backend shall add any existing users whose email addresses appear in the whitelist to the class.

1.8 [A8] User Logout

Description	The user shall be able to log out of their account on the site.
Desired Outcome	The user's information and data will no longer be accessible after logging out.
User Goals	The user is done with his/her session of using the website, and wants to make sure others cannot access the data in their account.
Primary Actor	User (student)
Dependency Use Cases	None
Priority Level	"Must"
Status	Implemented
Pre-conditions	The user is logged into their account
Post-conditions	The user's data can no longer be accessed.
Trigger	The user no longer wants to be logged into the application.
Workflow	1. The user clicks the 'Logout' button in the upper right hand corner. 2. The backend shall log the user out using the Django logout feature. 3. The frontend renders the application login page using login.html.

1.9 [A9] Contact Admin

Description	The user shall be able to contact the admin.
Desired Outcome	The user shall send a message to the admin; the admin shall receive the message. User Goals:The user sends a message to the admin.
Primary Actor	User
Dependency Use Cases	None
Priority Level	"Must"
Status	Implemented
Pre-conditions	None.
Post-conditions	The admin receives the user's message.
Trigger	The user wants to contact the adiminstators of the site with bug reports, questions, or praise.
Workflow	1. The user shall click the '?' button in the right corner of the navigation bar. 2. The frontend shall render the page using the help.html template. 3. The user shall scroll to the bottom of the page. 4. The user shall click the text reading "Send Us an Email!" 5. The frontend shall open Microsoft Outlook or a similar application using the HelpController.js. 6. The user shall see our email address in the recipients text box.

2 Flashcards

2.1 [F1] Push Flashcard

Description The user shall be able to create a flashcard with their input.

Desired Outcome The user shall have the flashcard added to their own deck and the Live Feed.

User Goals The user will see their flashcard in their deck and the flashcard will be shared with others.

Dependency Use Cases [A1] User Registration, [A2] User Login, [A3] Add a Class

Pre-conditions

- The user has an account.
- The user has added at least one class.
- The user has navigated to the live feed for a class.

Post-conditions

- The user has flashcard added to their deck.
- The Flashcard is shown in the Live Feed.
- Other users can add this flashcard to their decks.

Trigger The user wants to make a flashcard to study.

Workflow

1. The user is at the live feed for the class from feed.html.
2. The user shall click on the '+' button at the bottom right of the screen.
3. The frontend shall present a modal for inputting the flashcard text to the User.
4. The user shall type in the content of their flashcard.
5. The user shall highlight keywords in the flashcard text.
6. The user shall click on the 'Contribute' button.
7. The frontend shall generate a POST request consisting of the flashcard text, blanks, and material date, in JSON form in FeedController.js and send it to /api/flashcards/.
8. The backend shall obtain the flashcard information by deserializing the JSON in the POST request in views.py.
9. The backend shall create a new record for this flashcard in the Flashcards table in the database in models.py.
10. The frontend shall publish the newly created card to the live feed of the class using FeedController.js.

2.2 [F2] Edit Flashcard

Description The User shall be able to duplicate the flashcard authored by a different user and make changes to it.

Desired Outcome The new flashcard is edited and saved appropriately.

User Goals To create a flashcard similar to an existing flashcard but with minor changes.

Dependency Use Cases [A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard, [F3] Pull Flashcard

Pre-conditions

- The flashcard to be edited is visible in the user's feed view or deck view.
- The user is currently in the feed view or the deck view of the course the card to be edited belongs to.

Post-conditions

- A duplicate flashcard with some differences is created.

Trigger The User wants to duplicate a card and make some changes to it and thus create a new flashcard.

Workflow

1. The user shall hover over the card to be edited and and click on the pencil button that appears on the bottom left of the card.
2. The frontend shall produce an editable dialog box containing the flashcard text
3. The user shall make desired changes
4. The user shall click 'Save Changes'
5. The client shall generate a PATCH request for a new flashcard in CardGridController.js and send it to the server at /api/flashcards/
6. If the user changed only the blanks of the cards, the server shall create a new FlashcardMask object and update the appropriate User-Flashcard object with a reference to it.
7. If the user changed the text of the card, the server will instead:
   a) Create a new flashcard for the section
   b) Push it to the feed
   c) Add it to the user's deck
   d) Hide the old card from the user
   e) Return the new card to the user

2.3 [F3] Pull Flashcard

Description	The User shall be able to add flash cards to their own deck from the Live Feed.
Desired Outcome	The User shall have the flashcard added to their own deck
User Goals	The user will be able to review that flashcard.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashard
Priority Level	"Must"
Status	Implemented
Pre-conditions	• A flashcard has been created and is in the Live Feed for a class. • The User is currently viewing the Live Feed for that class.
Post-conditions	• User shall have that flashcard added to their deck. • User can review this flashcard later.
Trigger	The User wants to save a flashcard for review later.
Workflow	1. The user shall hover over the flashcard he wants to add to his deck and click on the '+' icon that appears in the center of the card. 2. The frontend shall make the pulled Flashcard appear in the user's deck. 3. The frontend shall generate a POST request in FlashcardFactory.js and send it to /api/flashcards/<flashcard id>/pull 4. The server shall call the FlashcardViewSet.pull to handle the POST request sent by the frontend. 5. The FlashcardViewSet.pull method should call the user.pull method in models.py with the flashcard object to be pulled as an argument 6. The user.pull method shall create a UserFlashcard object associated with the request's user and the passed in flashcard object and save it in the database. 7. The server shall notify connected clients about the new card rating, if any 8. Connected clients will take the new rating into account when next rearranging the feed.

2.4 [F4] Flag Inappropriate Cards

Description	Cards may be flagged indicating inappropriate content
Desired Outcome	The flashcard's inappropriateness variable is adjusted.
User Goals	To notify the admins if a card is inappropriate and should not be displayed in the feed.
Primary Actor	User (Student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard
Priority Level	"Should"
Status	Implemented
Pre-conditions	• The user is enrolled in a class • The flashcard has been created. • The flashcard is viewable in feed. • The user has clicked the red 'eye' icon on the flashcard which is the "Hide Card" button.
Post-conditions	• Flashcard is hidden from user. • The flashcard is marked as being inappropriate.
Trigger	The User does not like the particular flashcard.
Workflow	1. The frontend displays a toast that offers the user the report text. 2. The user clicks the "Report" text on the flashcard. 3. The frontend shall hide the flashcard from the user in FlashcardFactory.js. 4. The frontend shall generate a POST request in Flashcard.report in FlashcardFactory.js and send it to /api/flashcards/<flashcard id >/report 5. The backend shall check if the user already hid the flashcard in views.py function report. 6. The backend shall hide the flashcard in models.py in class FlashcardHide.

2.5 [F5] Filter Flashcards

Description	The user is able to filter for flashcards by material date and text.
Desired Outcome	The user shall see flashcards based on the filter options: the material date and the text of the card. User Goals:The user can find what he/she is specifically looking for.
Primary Actor	User
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard, [F3] Pull Flashcard
Priority Level	"Should"
Status	Implemented
Pre-conditions	• The user has added a class. • The class in question contains non-zero number of flashcards.
Post-conditions	• The user only sees the flashcards that match his filter criterion.
Trigger	The user wants to only see cards for a specific date range or with a specific substring in their text.
Workflow	1. The frontend shall render the page described in study.html using CardListController.js. 2. The user shall select the weeks that the filtered cards should belong to. 3. The user shall type in the appropriate search filter in the text input box on the top right of the screen. 4. The frontend generates a POST request with the flashcard dates and the flashcard text filter as parameteres and sends it to /api/study. 5. The system displays only the cards that match the filter criteria specified by the user.

2.6 [F6] Blank Out Words in Flashcard

Description The User shall be able to blank out keywords in any flashcard in his deck.

Desired Outcome The blanked out words in the flashcard notify the System that they are keywords.

User Goals The User shall mark some words as keywords so the System may later quiz theirself by blanking out the keywords and having the User guess what they are.

Primary Actor: User (student)

Dependency Use Cases [A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard

Priority Level "Must"

Status Implemented.

Pre-conditions

- The User has an account and is logged in.
- The User shall be on the Live Feed for the class.
- The User has clicked the 'c' hotkey or the 'plus' button to start making a card.
- The User shall type in information relevant to their class.

Post-conditions:

- The blanked out words in the flashcard are marked as keywords.
- The System shall blank out the keywords and let the User guess what they are when it presents the flashcard to the User for reviewing.

Trigger The User wants to make a flashcard and be quizzed on parts of the flashcard while they study.

Workflow

1. The User shall highlight the words that he wishes to blank out.
2. The frontend shall enclose the selected text in <b> tags to alter its appearance.
3. The User shall click the 'Contribute' button.
4. The frontend shall convert the blanked portion of the text into a list of offsets to be sent to the backend in FeedController.js.
5. The backend shall mark those words by updating the field mask for that userflashcard object in views.py in function create.

6. The backend shall save the new mask in models.py.

Alternate Workflow

1. Precondition: The User has clicked the edit flashcard button instead of creating a new card.
2. The User shall highlight selection text and click either ctrl-b or 'Blank Selected Text'.
3. The frontend shall enclose the selected text in <b> tags to alter its appearance.
4. The frontend shall convert the blanked portion of the text into a list of offsets to be sent to the backend in CardGridController.js.
5. The backend shall mark those words by updating the field mask for that userflashcard object in views.py in function create.
6. The backend shall save those in models.py.

2.7 [F7] Fix Flashcard

Description The User shall be able to alter a flashcard he/she made originally.

Desired Outcome The system changes the flashcard in the user's deck so that it doesn't affect any other users who added the previous version of the flashcard to their decks.

User Goals The user shall either change the text or the blanks of the flashcard.

Primary Actor User (student)

Dependency Use Cases [A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard

Priority Level Must

Status Implemented

Pre-conditions

- The user has created the flashcard
- The user is either in the feed view or the deck view of a class and can see the flashcard that he wishes to fix.

Post-conditions

- The flashcard in the user's deck gets updated to reflect the changes made by the user.

Trigger User (creator of original card) has clicked on the button blue pencil icon at the bottom left of the flashcard.

Workflow

1. User shall hover over one of the Flashcards they authored.
2. User shall click on the blue pencil icon at the bottom of the flashcard.
3. The frontend shall display a dialog box to edit the flashcard.
4. The User shall change the text and the blanks of the flashcards appropriately.
5. The User shall click on the "Save Changes" button.
6. The frontend shall convert the blanked portion of the text into a list of offsets to be sent to the backend in CardGridController.js.
7. The backend shall update the flashcards content in FlashcardViewSet.partial_update in views.py.
8. The backend makes sure that no other user has the card in question in his/her deck in Flashcard.edit in models.py.
9. The backend makes the appropriate changes to the card and saves them.

Alternate Workflow	1. The backend finds that other users have added the card that is being edited. 2. The backend shall create a new card with the passed in changes applied and add it to the user's deck after removing the older card from the user's deck.

2.8 [F8] Hide Cards

Description	The user shall be able to hide cards from feed
Desired Outcome	The card is no longer visible to the user
User Goals	The card has been looked at and should be hidden to reduce screen clutter Primary Actor: User (Student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class, [F1] Push Flashcard
Priority Level	"Should" Status:Implemented
Pre-conditions	• Flashcard is created and is viewable by user. • The user is at the live feed or the deck view for the class the flashcard in question belongs to.
Post-conditions	Flashcard is not viewable by user
Trigger	The user no longer wants to see a card in their feed and deck.
Workflow	1. The user shall hover over the card they want to hide. 2. The user shall click on the red 'eye' icon on the bottom right of the flashcard. 3. The frontend shall make a request to the backend from FeedController.js. 4. The backend shall check if the user has already hidden that card in views.py. 5. The backend shall change the falshcard to hidden in the models.py. 6. The frontend shall no longer show that card when the feed is rendered.

2.9 [F9] View a Feed

Description	The user shall be able to view Live Feeds for different classes
Desired Outcome	The system shall only show the user Live Feeds for specific classes.
User Goals	The User will see only one Live Feed at a time.
Dependency Use Cases	[A1] User Registration, [A2] User Login, [A3] Add a Class
Priority Level	"Must"
Status:	Implemented
Pre-conditions	User has added a class.
Post-conditions	User shall see the Live Feed for that class
Trigger	User shall select a class
Workflow	1. The user logs into their account. 2. The system verifies the User's credentials and saves their session. 3. The user selects a class from the dropdown menu at the left side of the navbar. 4. The frontend displays the Live Feed for the class.

3 Decks

3.1 [D1] Remove a card from a deck

Description The user can remove flashcards from their deck.

Desired Outcome The user will not be notified about that card.

User Goals To only review cards that the user wants to review.

Primary Actor User (student)

Dependency Use Cases Add a class [A3], Add Flashcards to Deck [F1], Make a Flashcard [F3]

Priority Level "Must"

Status Implemented

Pre-conditions

- The User has an account with the application
- The User is logged in

Post-conditions

- Desired cards are hidden to the user.

Trigger The User no longer wishes to be able to study a card.

Workflow

1. The frontend shall shows the user the navigation bar from /home.html/.
2. The user shall select the appropriate class.
3. The frontend shall route the user to the feed controller in /scripts/feed.js/ for the appropriate class.
4. The user shall select deck view.
5. The frontend shall route the user to the deck controller in /scripts/deck.js/.
6. The user clicks a flashcard's remove button.
7. The frontend shall send a DELETE request to the server at /api/flashcard/<flashcard ID>/remove.
8. The backend removes the flashcard from the user's deck.
9. The backend updates the flashcard's position in the live feed.
10. The frontend updates the user's Deck.

Alternate Workflow

1. The frontend shows the user the navigation bar from /home.html/.
2. The user shall select the appropriate class.
3. The frontend shall route the user to the feed controller in /scripts/feed.js/ for the appropriate class.
4. The user clicks a flashcard's remove button.

5. The frontend shall send a DELETE request to the server at /api/flashcard/<flashcard ID>/remove
6. The backend removes the flashcard from the user's deck.
7. The backend updates the flashcard's score.
8. The frontend updates the user's deck.

3.2 [D2] Viewing cards in deck by pull time

Description	The user is able to organize the deck by time stamp.
Desired Outcome	The user views the deck in time ascending/descending order.
User Goals	To organize the card for easier editing purpose.
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [F3] Pull Flashcard, [D1] Create Deck
Priority Level	"Must"
Status	Implemented
Pre-conditions	• The user has an account with the application. • The user is logged in. • The user has a deck with at least 1 flashcard. • The user is at the live feed for a class.
Post-conditions	• Cards in the deck are in time order.
Trigger	The user wants to view their cards in a meaningful order.
Workflow	1. The user shall select the deck view from the navbar. 2. The frontend renders the view of the deck from the template deck.html.
Alternative Workflow	1. The user shall select the list view from the navbar. 2. The frontend renders the view of the deck from the template cardlist.html.

4 Review

4.1 [R1] Study Deck

Description	The user shall be able to look at the cards in his/her deck in order to study them
Desired Outcome	The user shall be presented with individual flashcards in an optimized order. Blanks will be empty and the user will have a text boxes to fill in.
user Goals	The user shall be able to study all flashcards in his/her deck from the appropriate class.
Primary Actor	user (student)
Dependency Use Cases	[F1]
Priority Level	"Must"
Status	Implemented
Pre-conditions	• user is logged in • user has added cards to his/her deck
Post-conditions	• user shall see a single card with blanks. • user shall be presented with text boxes to fill in
Trigger	user wishes to study flashcards compiled for a specific class.
Workflow - Blanked	1. The user shall view a class. 2. The user shall click on the button "Study Deck". 3. The frontend shall display study.html to the user. 4. The user shall select which classes he/she would like to study. 5. The user shall click on the button "Fetch" to begin studying. 6. The frontend shall send POST request to the UserFlashcardQuizViewSet in views.py for a flashcard (ordering based on a hidden algorithm). 7. The user shall enter a response for the flashcard. 8. The frontend shall send the user's response back to the backend in POST. 9. The backend shall inform the frontend of how correct the user's repsonse was. 10. The frontend shall display said results to user.

11. The frontend shall produce two buttons for the user, indicating whether or not the card was answered correctly based on the provided results.
12. The user shall select their desired response.
13. The frontend shall relay response to the backend in a POST request.
14. The backend saves the response and updates statistics about the flashcard.
15. Loop back to 6.

[Alternative Workflow - No Blanks]

1. The user shall view a class.
2. The user shall click on the button “Study Deck”.
3. The frontend shall display study.html to the user.
4. The user shall select which classes he/she would like to study.
5. The user shall click on the button “Fetch” to begin studying.
6. The frontend shall send POST request to the UserFlashcardQuizViewSet in views.py for a flashcard (ordering based on a hidden algorithm).
7. The backend shall inform the frontend that this card had no blanks.
8. The user shall click on the button “Next” to retrieve the next card.
9. Loop back to 6.

4.2 [R2] Review Notification

Description	The User shall be notified when to review each specific card.
Desired Outcome	The User shall receive a notification when it is time to review a specific card using a spaced repetition formula.
User Goals	The user shall see the notification and which card needs to be reviewed
Primary Actor	User (student)
Dependency Use Cases	[A1] User Registration, [A2] User Login, [F3] Pull Flashcard
Priority Level	"Must"
Status	Implemented
Pre-conditions	• User has an account with the system. • User is logged in. • User has cards in his/her deck.
Post-conditions	• System shall calculate when to next present the card based on a memory decay formula. • User shall be notified.
Trigger	Time has passed for a card to be reviewed.
Workflow	1. The backend analyzes a User's flashcard's statistics and when they last reviewed the card to determine when to send the flashcard notification. 2. The backend shall send a notification to the User at the specified review time notifyusers.py in directory management/commands. 3. The frontend receives a "Review Flashcard" notification and handles in in SettingsController.js. 4. The frontend displays the notification.

4.3 [R3] Configure Account Notifications

Description	The user shall be able to turn notifications on or off at will
Desired Outcome	The user shall only receive notifications at the times that were specified.
User Goals	The user shall not be bothered needlessly or at inconvenient times.
Primary Actor	User (student)
Dependency Use Cases	...
Priority Level	"Must"
Status	Not Implemented
Pre-conditions	• The user has a verified account.
Post-conditions	• The user only receive notifications at specified times.
Trigger	The user indicates that they want to change their notification settings.
Workflow	1. The user selects "Settings" button from the navigation menu bar. 2. The frontend displays a settings screen using settings.html. 3. The user shall modify notifications and settings as they see fit. 4. The user shall click "Save Settings" button. 5. The frontend shall display a message confirming the settings are saved from the SettingsController.js. 6. The backend shall save notification settings to the user's attributes in models.py.