# MALBUILDER

Allen Qiu
Andrew Nguyen
David Tang
Zane Nicholson
Harlan Cao
Jaily Zeng

# Table of Contents

# Description

Both Japanese animation (anime) producers and consumers require a tool to keep track of anime titles regular watchers have enjoyed, both to help the ordinary fan determine which show to watch next as well as provide feedback to the industry on which genres are most popular, and therefore more profitable. However, current online anime indices like MyAnimeList (MAL) require many transactions on the user's part to update said lists, discouraging many potential users from updating their MAL and providing better feedback for titles on a regular basis. Our proposed application, MALBuilder, would provide a much simpler interface for modifying anime watchlists in bulk and synchronizing them with the users main MAL. MALBuilder would help regular anime watchers complete an accurate MAL as well as encourage them to update their watchlists more frequently, providing better feedback and watch demographics on whole.

# Process followed by Team

## *Iterative Development*

The software process that our team will be following is extreme programming, which is a relatively new software development project but its methodology has been around for a while. In the early 1990s, object oriented programming was popular it was introducing new practices in the field, such as the concept of "object" programming. Unlike object oriented programming, extreme programming does not introduce new practices as much as it was shifting the emphasis that was placed on some of the practices. The practices that are emphasized in extreme programming include frequent planning, frequent releases, test-first development, refactoring, coding. During the planning process, user stories are written for the purpose of creating time estimates in order to plan meetings. They are usually written by the customers, documenting what they need the system to do for them. User stories are used more often to replace the need for a large requirement documents. The stories focuses primarily on the user needs, and should not include excessive details on specific technology, databases, or algorithms. User stories only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement. Developers are the ones who estimate the approximate time it would take to implement each story.

## *Refactoring*

Refactoring is a powerful technique that is used for improving code quality, but is often used with caution as it may lead to more problems. The biggest safety factor is saving the code as a backup in case something goes wrong and the working code is broken during the refactoring process. Refactoring is conducted in small segments so that the changes which are being made are understood before continuing. Since some refactoring tasks are more complicated than others, they are completed one at a time. Running multiple refractions at once causes confusion in seeing which segment is being affected. Once the refactoring is complete, there should be retests to make sure that there were no changes to the outputs of the program (McConnell 638). The reason for this is because "when programmers make changes to a program, they typically have more than a 50 percent chance of making an error the first

time" (McConnell 638). Therefore, checking to make sure the code is executing properly is expected after a refactoring process.

### *Testing*

The advantage of test-driven development is to force the team to set goals and to develop benchmarks. This aligns with planning as the iterations help designate what module of the code is completed and can have small unit tests to check whether they are functioning properly or not. The decision on dedicating a block of time for unit testing becomes unfavorable as the project deadline lurks around the corner. The benefit of unit testing is that they help determine whether or not a specific feature is working as it is intended. Programmers usually write automated tests to try and break the code. If the code passes all of the tests successfully, then the code can then be moved forward to the next feature or process (Wells). Having frequent unit tests running against the code help with fixing the problem while it is small. Otherwise if left alone, the bug can escalate into a bigger problem when time is scarce. After each small change, such as refactoring, the unit tests can verify that a change in structure did not introduce a change in functionality.

### *Collaborative Development*

A final merit of the XP process is pair programming. This approach introduces the role of a driver and navigator working together at one machine. The driver types while the navigator observes and provides feedbacks, with the programmers periodically swapping roles. Empirical studies indicate that there are numerous advantages of pair programming. It tends to produce higher quality code at a faster rate than when the members work independently. This capitalizes on pair pressure, where the presence of a partner encourages an individual to stay on task. Pair confidence makes debugging and reviewing code easier, since two pairs of eyes on the same material increases the chances of catching any errors. Pair members are also able to learn from and understand each other, making this approach invaluable to overall team building.

## Requirements and Specifications

### *User Stories:*

- Research MAL API
    - Determine whether or not MAL API has capabilities to:
        - Authenticate a user's credentials
        - Download a user's MAL
        - Modify a user's MAL with add, update, and delete
- Research AnimeAdvice API
    - Determine whether or not AA API has the ability to search titles by the following criteria
        - Season first aired
        - Genre
        - Popularity
    - And return the following data:
        - MAL ID

        - Name
        - Season first aired ([Winter/Spring/Summer/Fall] + [Year])
- Flask Framework
    - Layout the basic project framework by:
        - Setting up a default flask project
        - Connecting it to BitBucket
- Front End Mockup Concept
    - Create a mockup concept that contains:
        - The main page interface
        - The interface that queries the user if they have or have not seen specific shows
- Terminal Interface for MAL API
    - Main flow: As a User, I can enter my MAL credentials, so that I can see a menu with options “View current MAL”, “Search anime by keyword”, “Search anime by ID”, “Add new anime”, “Update existing anime”, “Delete existing anime”, and “Exit”.
- Terminal Interface for MALB API
    - Main flow: As a User, I can enter my MAL credentials, so that I see a menu with options “View current lists”, “Add to list”, “Delete from list”, and “Exit”
- Terminal Interface for AA API
    - As a User, I can enter in a list of filters and return columns, so that I receive a list of anime corresponding to these criteria:
        - The interface is unable to connect to the AA API, so I get the prompt "Error: unable to connect to AA" and the program exits.
        - I enter in invalid AA search criteria, so I get the prompt "Incorrect AA credentials" and is asked to try again.
- MAL Initialization (UC MAL Synchronization)
    - As a User, I can authenticate myself with my MAL credentials so that I enter MALB and carry out these basic operations:
        - Load the current state of my saved MALB and display it on the front page.
        - Run the synchronization operation to download the master copy of my MAL from MyAnimeList and refresh the page.
- Search AA for new titles (UC Search Anime)
    - Search metrics include: Type of medium, Airing status, Include and exclude genres, Start date, and End date
- Migrate anime search functionality from AA interface to database instead (UC Login)
    - As of 2/27/2015, AnimeAdvice will be moving development of their anime search functionality to a new API. The deprecation of the API MALB currently uses is expected to be deprecated before the end of the semester. As such, we must prioritize transferring our search functionality from the AA API to our own database
    - Priorities:
        - Create scripts to obtain sample anime data for the database from the new AnimeAdvice API
        - Update database models to support anime data as well as update user data for additional metrics
        - Set up genre storage models and functionality if enough time
        - Map AA Search functionality to database queries using the following fields: Title, type, status, start date, end date, score, genres, and return field

        - Create new tests to target the database search functionality instead of original AA
- Bulk rate all existing titles (UC Update MAL)
    - As a User, I can go down my existing MAL and rate them, so that the ratings are immediately updated for my MAL.
        - User navigates to "Rate Anime" page.
        - System loads the User's current MAL (excluding titles mark as "not watched") from the database into a simple table and populates the User's current rating for each title in a text box.
        - User populates each result with a rating 1-10 (or 0 for "not rated") and submits through "Update Scores" at the bottom of the list.
        - System updates the User's MAL with all scores in the text boxes, flashes "Score Updated", and redirects back to the "Rate Anime" page.
        - User enters an invalid score for some title. System prints the verification error "Error: Invalid score format".
- Bulk add new titles (UC Add Anime)
    - As a User, I can go down a list of anime search results and select which ones I have seen, so that they are added to my MAL.
    - As a User, I can use the Search Page to find a list of new anime, so that I can add them to my MAL,
        - User searches through functionality described in "Search Anime" use case.
        - System returns the results rendered in a itemized form for adding each anime.
        - User chooses the watch status for each anime and submits through "Add Anime".
        - System updates User's MAL with the newly added anime and returns a new form of more anime matching the same criteria, if any.
        - User may continue to add anime matching the same search criteria, or simply start a new search.
- Filter my MAL and chance display specific columns (UC Filter MAL)
    - As a User, I can filter my MAL to match anime search metrics and choose which columns I wish to see in the results.
        - User navigates to index page.
        - System loads the MAL search menu (see "Add Anime" for the example format).
        - User selects which filter criteria to apply (use both Anime and UserToAnime filters) along with which columns to return in the result (use both Anime and UserToAnime columns) and submits through "Search my MAL".
        - System loads the User's MAL that pass the given filters and column values in a simple table.
        - User enters an invalid search criteria. System prints the verification errors at the top of menu.
- Search anime by season (UC Anicharts Search)
    - As a User, I can choose an anime season ([season], [year]) and view the metrics for shows that aired that season in Anicharts format (See http://anichart.net/)
        - User inputs a Year and Season
        - System returns a chart in Anicharts format so that the User can quickly visualize what titles had aired during that time period
- Update all anime metrics (UC Update MAL)

    - As a User, I can update any of my user statistics for any anime entry in my MAL, so that the changes are saved for my MAL
        - Restrict possible range of values for rating and watch status, possibly using dropdowns.
- Enable updates to MyAnimeList (Extend UC Add Anime, UC Update MAL)
    - As a User, I can update any of my user statistics for any anime entry in my MAL, so that the changes are saved for my MAL
        - User makes changes through “Add Anime” or “Update Anime”.
        - If configured to send MyAnimeList transactions, System will send the appropriate queries to MyAnimeList
- Enable Flashcard Interface for Add Anime (Extend UC Add Anime)
    - As a User, I can add anime through the flashcard interface, which draws from common searches and displays titles one by one with options "Completed" or "Not watched".
        - User selects a flashcard sort metric among rating, popularity, and season.
        - System returns the appropriate flashcard stack and displays one by one.
        - User submits with either completed or not watched, and the title is added as such.
- Rebuild Front Page View for MALB (Extend UC View MAL)
    - As a User, I can navigate to the main page so that I can quickly the current status of my MAL.
        - Should be able to dynamically change the columns displayed in table.
- Update Individual Additional Metrics for an Anime (Extend UC Update MAL)
    - As a User, I can select an individual title on the update menu and be redirected to a form that enables editing of all user details for that specific anime (based off MAL input chart)
        - Complete update metrics with start and end dates as well as rewatch number.

## *Use Cases:*

- Login
    - As a User, I provide my MyAnimeList credentials so that the System can verify I am registered with MyAnimeList and allow me to access MALB.
- Search Anime
    - As a User, I can enter in the following search criteria, so that I get back a list of matching anime in tabular form
        - Search metrics include: Type of medium, Airing status, Include and exclude genres, Start date, and End date
- Add Anime
    - As a User, I can go down a list of anime search results and select which ones I have seen, so that they are added to my MAL.
    - As a User, I can use the Search Page to find a list of new anime, so that I can add them to my MAL
        - User searches through functionality described in “Search Anime” use case
        - System returns the result rendered in a itemized form for adding each anime
        - User chooses the watch status for each anime and submits through “Add Anime”
        - System updates User’s MAL with the newly added anime and returns a new form of more anime matching the same criteria, if any

        - User may continue to add anime matching the same search criteria or simply start a new search
- View/Filter MAL
    - As a User, I can filter my MAL to match anime search metrics and choose which columns I wish to see in the results
        - User navigates to index page
        - System loads the MAL Search menu (see “Add Anime” for example format
        - User selects which filter criteria to apply (use both Anime and UserToAnime filters) along with which columns to return in the result and submits through “Search my MAL”
        - System loads the User’s MAL that pass the given filters and column values in a simple table
- Synchronize MALB
    - As a User, I can authenticate myself with my MAL credentials so that I enter MALB and carry out these basic operations:
        - Load the current state of my saved MALB and display it on the front page
        - Run the synchronization operation to download the master copy of my MAL from MyAnimeList and refresh the page
- Update MAL
    - As a User, I can go down my existing MAL and rate them, so that the ratings are immediately updated for my MAL
        - User navigates to “Update Anime” page
        - System loads the User’s current MAL (excluding titles marked as “not watched” from the database into a simple table and populates the User’s current rating for each title in a drop box
        - User populates each result with a rating 1-10 (or 0 for “not rated”) and submits through “Update” at the bottom of the list
- Enable Updates to MyAnimeList
    - As a User, I can modify my MAL through “Add Anime” or “Update MAL”, so that I can view the changes forwarded to my MAL on MyAnimeList
- Anicharts Search
    - As a User, I can search anime by Year and Season and get back a list of anime results displayed in Anicharts form (thumbnails and descriptions for each title arranged in a table so as to give the User an overview of what shows aired that season)
- Flashcard Adding
    - As a User, I can add anime through the flashcard interface, which draws from common searches and displays titles one by one with options "Completed" or "Not watched".
        - User selects a metric to sort anime details he has not seen before such as “Most Popular” or “Highest Score”.
        - System loads anime titles one by one sorted by this metric.
        - For each title the User selects "Completed" or "Not watched”, allowing him to quickly add titles to his MAL while eliminating he has definitely not watched.

## Architecture and Design

MALB was built on top of the Python Flask web application framework due to the familiarity many of our team members had with the framework after completing 242. Following the Flask hierarchy MALB was designed to accommodate the MVC design pattern. Users make requests to MALB by accessing the website along several route, each of which are tied to a single controller action in `views.py`. Upon carrying out the corresponding transaction the controller completes itself by rendering an HTML template with Jinja2 and returning this view to the User. The transaction passes through the `malb.py` interface, which subdivdes tasks among specific backend interfaces. These include `db.py`, which accesses own PostgreSQL database, `mal.py`, which makes API calls to MyAnimeList, and `aa.py`, which parses saved AnimeAdvice data dumps. User input is parsed and verified through models in `forms.py`, built off the WTForms library, and query outputs are passed around in models defined in `models.py`.

Using Flask as the framework of our application forced us to use the MVC design pattern. This suited us just fine, and we probably would have decided to use MVC even if it wasn't force on to us as it promotes clarity of design. We initially intended to offload much of the functionality onto existing frameworks. For example, WTForms offered methods to validate user input and report what errors had occurred, so we selected this library to handle all our form validation. However, we eventually discovered that WTForms did not support dynamically generated forms. We managed to get around this but it cost us development resources that could have otherwise gone into use case development. Similarly, early load testing demonstrated the the MyAnimeList API could not support higher sizes of the bulk transactions we had initially intended. This forced us to rewrite the `mal.py` interface layer to accomdate for their performence bottlenecks. Finally, the initial use of the AnimeAdvice API led us to structure our database tables based on the same information AnimeAdvice had to offer. Even after we stopped using the AnimeAdvice API, we still found their schema of storing user and anime information convenient enough that we continued to work off of their model.

## *UML Class Diagram*

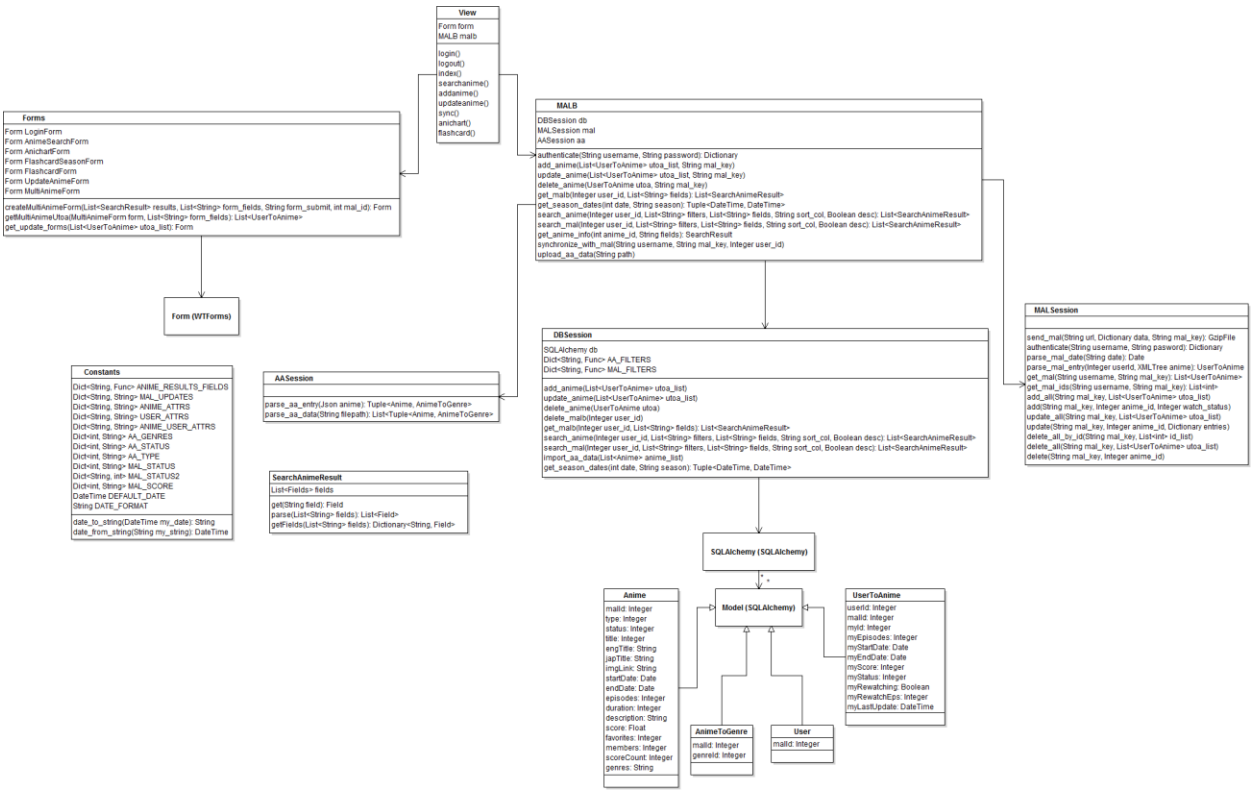


*Figure 1 - MALB Class Diagram*

## *UML Sequence Diagrams*

1. Login

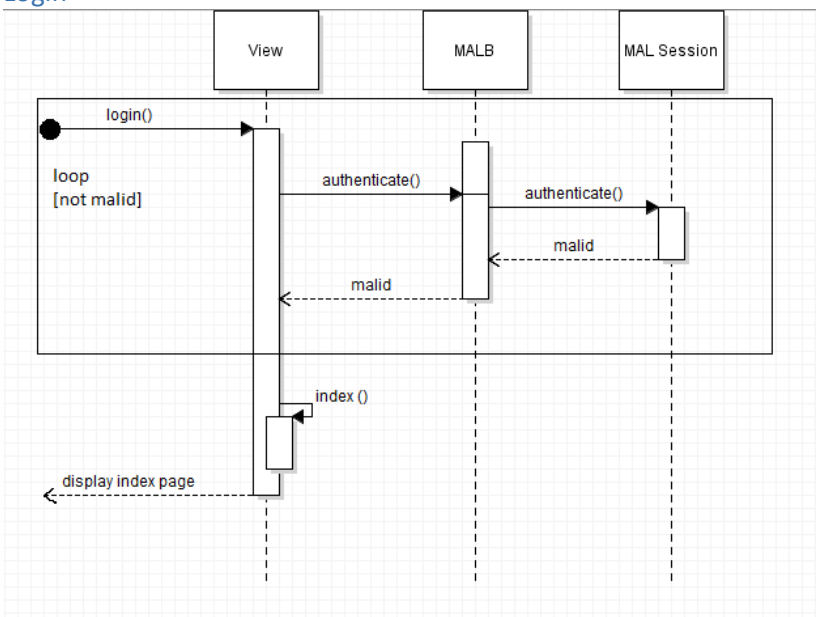


*Figure 2 - Login Sequence Diagram*

2. Search Anime

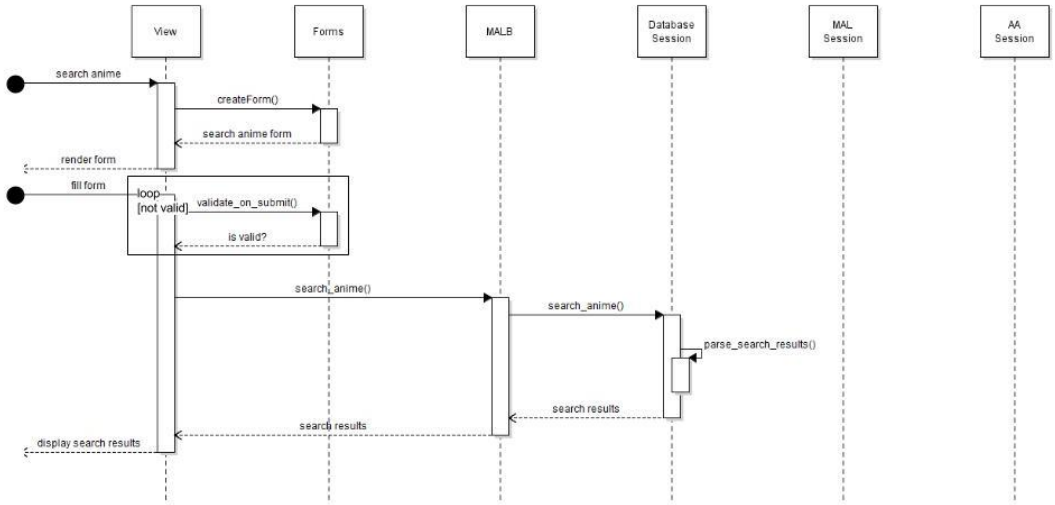


*Figure 3 - Search Anime Sequence Diagram*

3. Add Anime

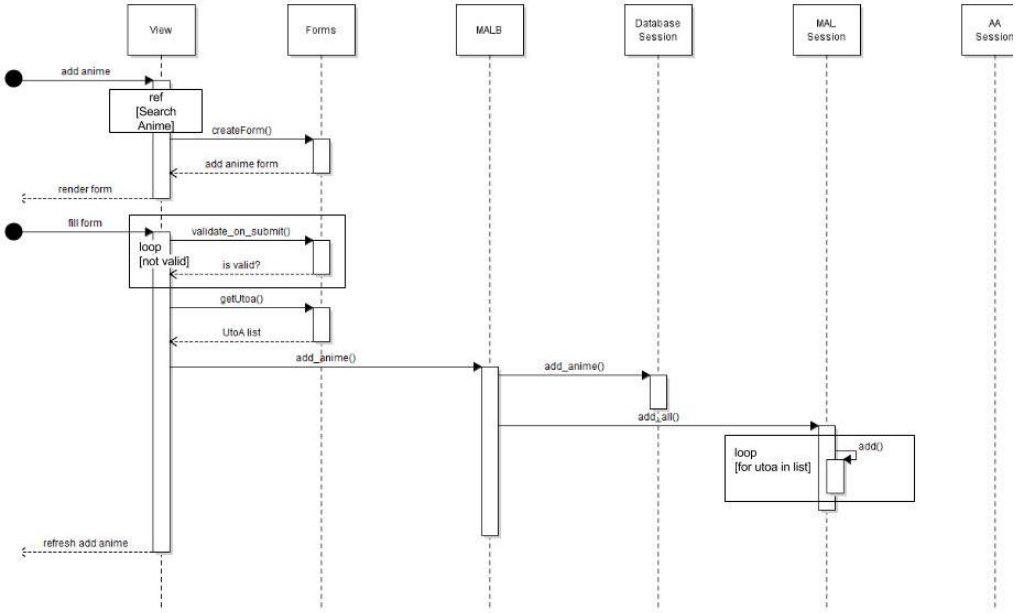


*Figure 4- Add Anime Sequence Diagram*

4. View/Filter MAL

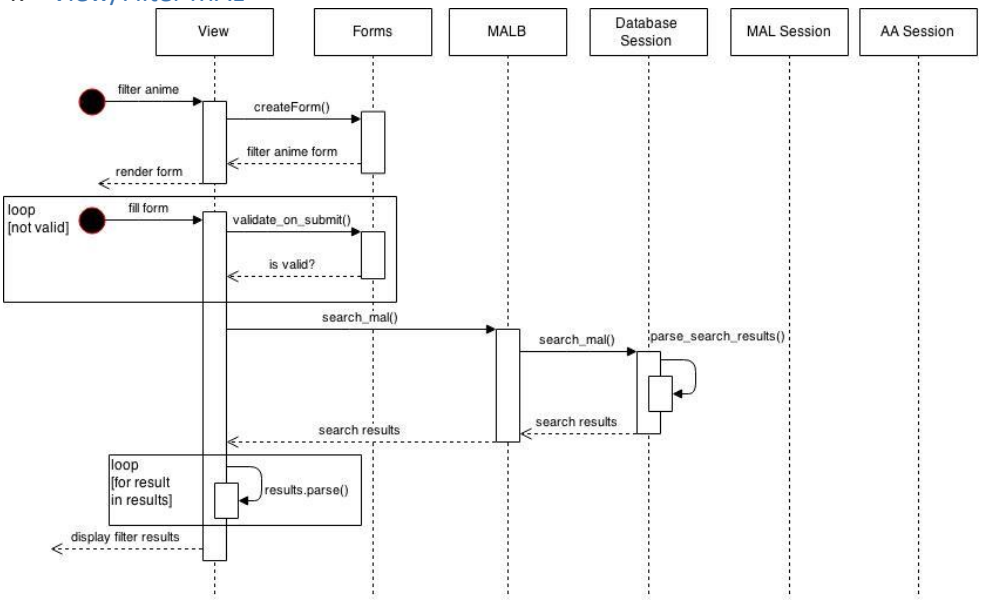


*Figure 5 - View Anime Sequence Diagram*

## 5. Synchronize MALB

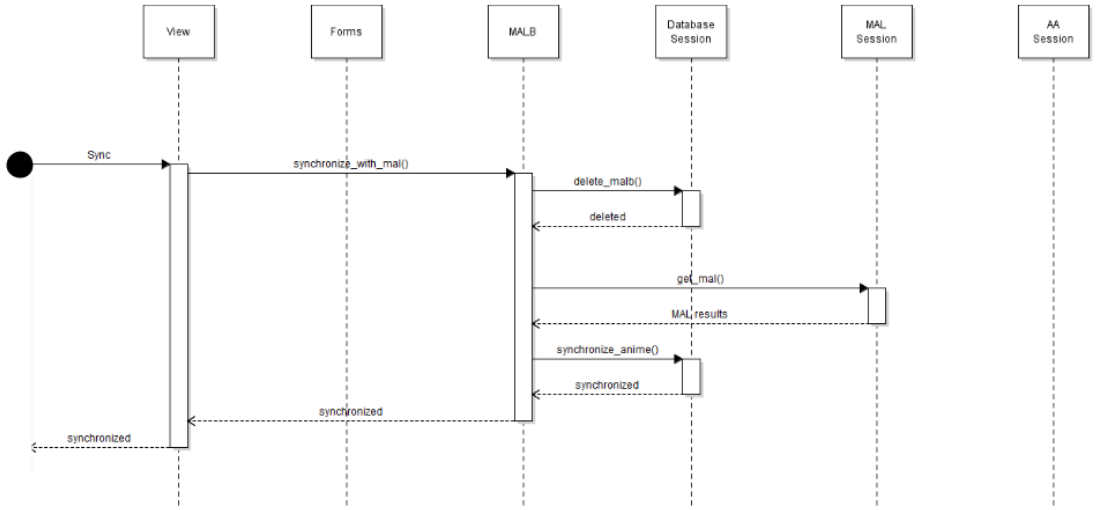


*Figure 6 - Synchronize MALB Sequence Diagram*

## 6. Update/Delete MAL

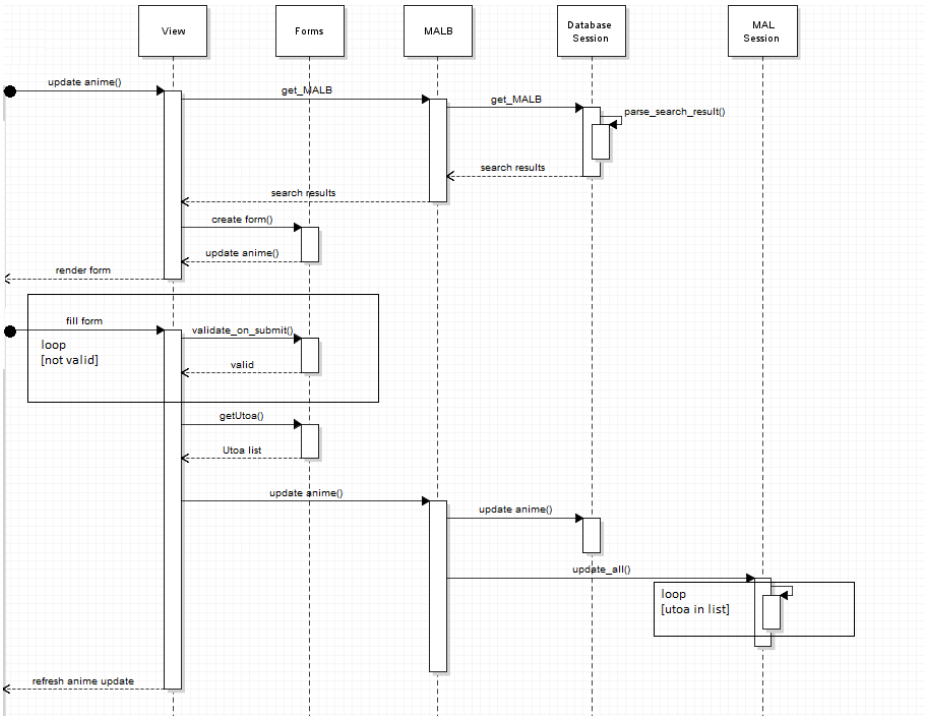


*Figure 7 - Update/Delete MAL*

## 7. Flashcard

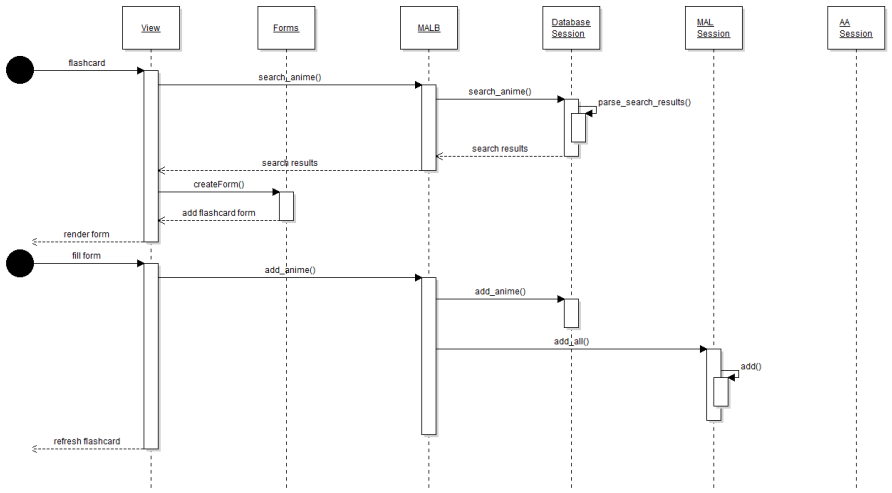


*Figure 8 - Flashcard Sequence Diagram*

## 8. Anichart

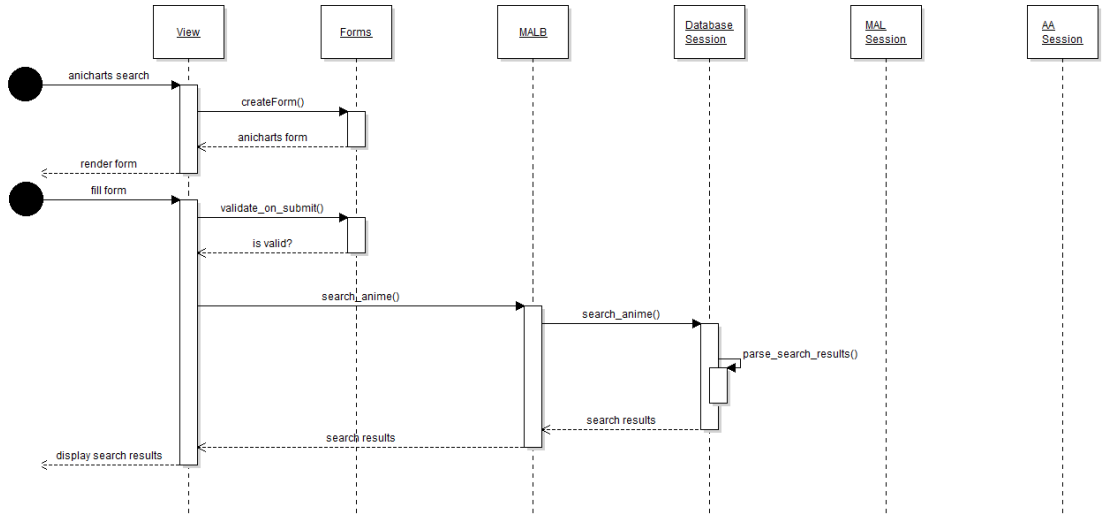


*Figure 9 - Anichart Sewuence Diagram*

Note: UML sequence diagram entities are mapped as follows:

1. View: view
2. Forms: MultiAnimeForm
3. MALB: malb
4. Database Session: dbsession
5. MAL Session: malsession
6. AA Session: aasession

# Future Plans:

## *Andrew Nguyen*

I was already used to XP from Software Engineering I so the XP practices we followed seemed normal. I did learn more about the design of a fully-fledged web application though. Probably the most prominent thing that I learned is how development can be stifled by certain policies. MyAnimeList got acquired by a new company and the new policies they enacted gave us a lot of trouble. In the end we our web deployment of our application fell into a wide blacklist, making all of our API calls fail. This highly discourages development and perhaps trying to work with MyAnimeList was a mistake in the first place.

## *Zane Nicholson*

Zane Nicholson - I learned a lot about how web development works as well as how the MVC framework can be helpful in separating code into easy to understand parts. I got more practice with XP development principles than in 427 and I think I will be able to apply them to any future projects I have in my career. I can also apply what I learned about working in a group and balancing work to future projects which should make me a more complete programmer.

## *Jaily Zeng*

In my Software Engineering I class that I took abroad, we did not learn about the practices that were covered through the same course here at UIUC. Therefore the entire practice of XP was completely new to me. I learned a lot more about front-end design constraints such as our need to switch over to using WTForms for dynamic list populations. The other aspect of the project that I was able to learn was unit testing and how to write unit tests to test the front end of our website. Being able to work on this project made me realize what I will have to deal with when I start working. In the sense where I will be working off other API's and existing databases rather than creating my own. With this, documentation of code is extremely important so that one doesn't get lost in the confusion of what a specific function call does, or why certain mappings are done the way they are.

## *Harlan Cao*

I was already exposed to Extreme Programming from cs427 so I was aware of a lot of the processes we used. I have work a bit with web service frameworks in the past so that also helped a lot. If anything, I refined and polished up those skills. Also, I learned about the pitfall that is WTForms with dynamic forms. Luckily we found alternative solutions.

Working on this project really opened my eyes to the amount of legacy protocols that exist in well-established services. MAL uses XML instead of the more recent such as JSON. We assumed that they were not going to make any major changes but that backfired when they decide to update their API. They also had some management changes in which their policies changed. As a result, we could not fully deploy our app. If anything, I would say that our app is on the level of a client-sided app. The problem is that the setup/install process would not be viable for the standard consumer. This project offers some new insight to various issues that can occur in real life development and could be applied to future projects.

## *Haoyang Tang*

After my experiences with cs427 and other projects, I walked into this project rather comfortable with XP protocols. Additionally, I had already worked with the Flask framework before, so that part of the project was also not a problem. However, during the project, one issue that popped up was understanding how to use WTForms. Another learning experience I encountered during this project was with MAL API calls, which also required some time to figure out. For any future plans, since deployment has been constrained to local services, it will be hard to receive much consumer feedback. Because of this, any future refactorings will also be difficult. However, through these turmoils, I have learned many traits that can be applied to future projects.

## *Allen Qiu*

I initially assumed 428 would build upon the same concepts shown in 427, with only a slightly broader scope to encompass the work on a project from its initial idea to the push to production. However, 428 was far more effective in demonstrating the real threat of risk and unexpected events, and how XP was effective in combating some of these problems while falling short on others. Our MALB was heavily reliant on other libraries and APIs, which became the bottleneck in development since we could not personally change their limitations. First and foremost was the MyAnimeList API itself, through which we attempted to synchronize real user accounts with our own information. MyAnimeList was purchased by another company shortly after the beginning of the semester, and over subsequent iterations we began to see them enact harsher limitations on the use of the API. This forced us to rewrite our own framework and reprioritize stories several time, which was thankfully hastened by the iterative aspect of XP programming. We also assumed our competence in Python/Flask libraries borrowed from previous projects (such as 242), but failed to rigorously explore these libraries for large-scale projects like MALB. It was only halfway through the semester that we discovered the WTForms library did not support dynamic forms creation, forcing us to expend even more time extending the base WTForms functionality to meet our advanced requirements. Our greatest challenge was working with AnimeAdvice, an experimental anime search API that underwent upgrades during the semester. Rather than continually updating our code to match a volatile API, our team decided to stop using AnimeAdvice entirely and establish our own independent anime search functionality. In hindsight we should have prioritized prototyping the advanced functionality first to accurately evaluate the risk of using such libraries, instead of focusing on simpler stories to demonstrate progress right away. Perhaps the most important lesson I gained from 428 is that all untried work carries risk, and that such unknowns should be resolved quickly as possible. If we had spoken with the new MyAnimeList owners before they instituted the API policy changes, or even discussed our initial plans with teams who had previously worked with the MAL API, the impact of these unknowns may have been alleviated. As it stands now, the current MyAnimeList policy now appears to blacklist AWS-based services, preventing us from moving forward to a full-fledged production deployment of MALB. Circumventing this policy by setting up our own server may only be a temporary fix. To completely remove all such risks, I feel that we must actively work together with the new MyAnimeList owners again and establish a new long-term contract to allow MALB to utilize their API before even considering pushing MALB to production. Another option would be to shift the workload of MALB to carry out MyAnimeList transactions on the client's computer, at the cost of completely rewriting the MALB framework.