Reflection on Design Adherence

Software Modelling Design

Project 3

25th Oct 2015

Tutorial Time:

9:00 - 11:00, Tuesday

Group Number:

Group 13

Group Member:

Nan Lin : 651025

Tessa(Hyeri) Song : 597952

Nihal Mirpuri : 562439

Reflection on Design Adherence

1) Changes we have made from the original design

Create admin controller

In the original design, we thought that admin/scrape could be done by refresh function in article controller and admin/email could be done by adding proper functions in users controller(ex) send_news_digest, interesting_articles, create_msg…). However, when it comes to implementation, we decided to build a new admin controller since that is more natural in terms of object-oriented design. In admin controller, there are two major functions which are email and scrape. Email function deals with sending digest to all subscribers and scrape functions scraping all new articles and they are linked to the url required by the specification.

Compiling and Emailing

According to the specification, we should not include the articles which have already been sent in digests. In the original design, we thought that we could solve out this problem by adding "t.datetime sent_last_digest_at" attribute to the user table. However, we noticed that this way should not function in the way it was supposed to be because of the rule that only 10 most latest articles are sent per one digest. If we kept using the sent_last_digest_at attribute, in the second try of sending digest we could not send any interesting articles even though there actually were in database. Considering that issue, we decided to use many to many relationships between users and articles. By building a articles_users table, every time a article is sent to a subscriber, we put the article in the articles list of that specific user so that next time we send a digest, we can check that if the article has actually been sent to the user or not.

Scraping the article

we removed the original scrape article button and added two admin paths into the route. Therefore, the admin can update the articles and send digest email to subscribers by simply typing in the '/admin/scrape' and '/admin/email'. At the same time, users are not able to scrape articles by themselves anymore in the website and they will see the update automatically with the help of the admin at the backend.

Add a 'categories' attribute in User

We added a categories attribute in User model. This categories attribute stores the values like category or section when articles scraped. The reason for this was caused by one of the requirements which was that scraping and tagging should be separate. All the tagging class-

es can do their work only relying on the database stored by scrapers. If we ignored that kind of value when scraping, it would be impossible for taggers to use them to tag given articles.

Paginating

In order to paginate articles, we added a gem 'will_paginate' and 'will_paginate-bootstrap'. The will_paginate gem is documented online, and we modified articles to paginate when we're viewing all articles, interest-related articles or doing a search. The 'will_paginate-bootstrap' gem was used to render the will_paginate widget in the article view in a style that was consistent with the rest of the website.

Search Engine

In the original design, we decided to add 'weight' attribute to the article table so that we can actually store the weight value which can be used for sorting later. However, it was not a convincing idea. We should have considered the situation where more than one user tries to use the search engine. For that reason, rather than storing weight values in the database, we decided to deal with the values within the method. The two sorting functions (sort_articles_by_pubdate and weight) were removed since there are more powerful built-in methods in ruby to sort.

Removed TaggingArticle class

We decided to remove TaggingArticle class since we could not find any common functions to put in it for all the 5 sub tag classes

2) Challenge and limits

Ruby gems for tagging articles

We used the gem 'alchemy' for tagging article by its summary and 'opencalais' by its title. However, there is a daily limit for using the API keys. Dealing with reasonably many articles, we received an error message saying 'you exceeded the daily limit of the API keys' when we tried to test our application. If you reach the usage limit, the method using that gem will not work anymore and all tags of all articles will be just gone. In order to solve out the limit usage problem, we decided to use 'indico' gem rather than 'alchemy' since it has monthly limit for using their API key. Another problem is speed. It takes quite a lot of time

for those gems to analyse given texts and to produce useful results. We tried to fix the speed problem by converting the external API calls into asynchronous coroutines (using the primitive type Thread), however we very quickly realised that the database (sqllite) can't handle coroutine transactions, despite increasing the default pool and timeout of the database. Therefore we switched back to a non-async method. If we were to fix this problem, it would involve either purchasing an account for the API to allow for bulk requests, or changing our database to a database that can handle multiple transactions simultaneously.

Sending emails using Mandrill in Chrome

We are using the gem called 'mandrill' following the recommendation on the specification. Basically the function 'email' in admin controller which is using this gem is working well. All subscribers will get the required digest properly if this function is called. However, there is some weird issue. If you type 'admin/email' url to invoke this function on Chrome, this function will be called twice in a row therefore all the subscribes will get two digests. Of course, the contents should be different. The first one will contain the first 10 newest interesting articles and the second one the next 10 newest interesting articles.

Searching Engine : Article matches a search iff the article matches all of the keywords in the search.

We had difficulty implementing the search engine, due to the complexity of attempting to filter tags used for the search, and then sorting the resulting articles afterwards. Using a database better suited for searching would have benefited us, especially since we could then do keyword queries and complete a search within a single database query. However this was beyond the scope of a basic search engine, and we decided instead to implement a roundabout method of converting hashes containing search terms and their relevancy into arrays and returning those results.

Possible Improvement

If we had more time to work on this project, we would spend time to do more research on finding some perfect gems so that we could prevent many unexpected issues like getting limited API key, sending an email twice on chrome, getting slow articles loading speed, etc.. At the mean time, we would also try to improve our 'relating tags to interests' part by finding a useful gem to make a more powerful design. Lastly, we would consider to make a radio button instead of a checkbox since radio button is more proper design to our intention

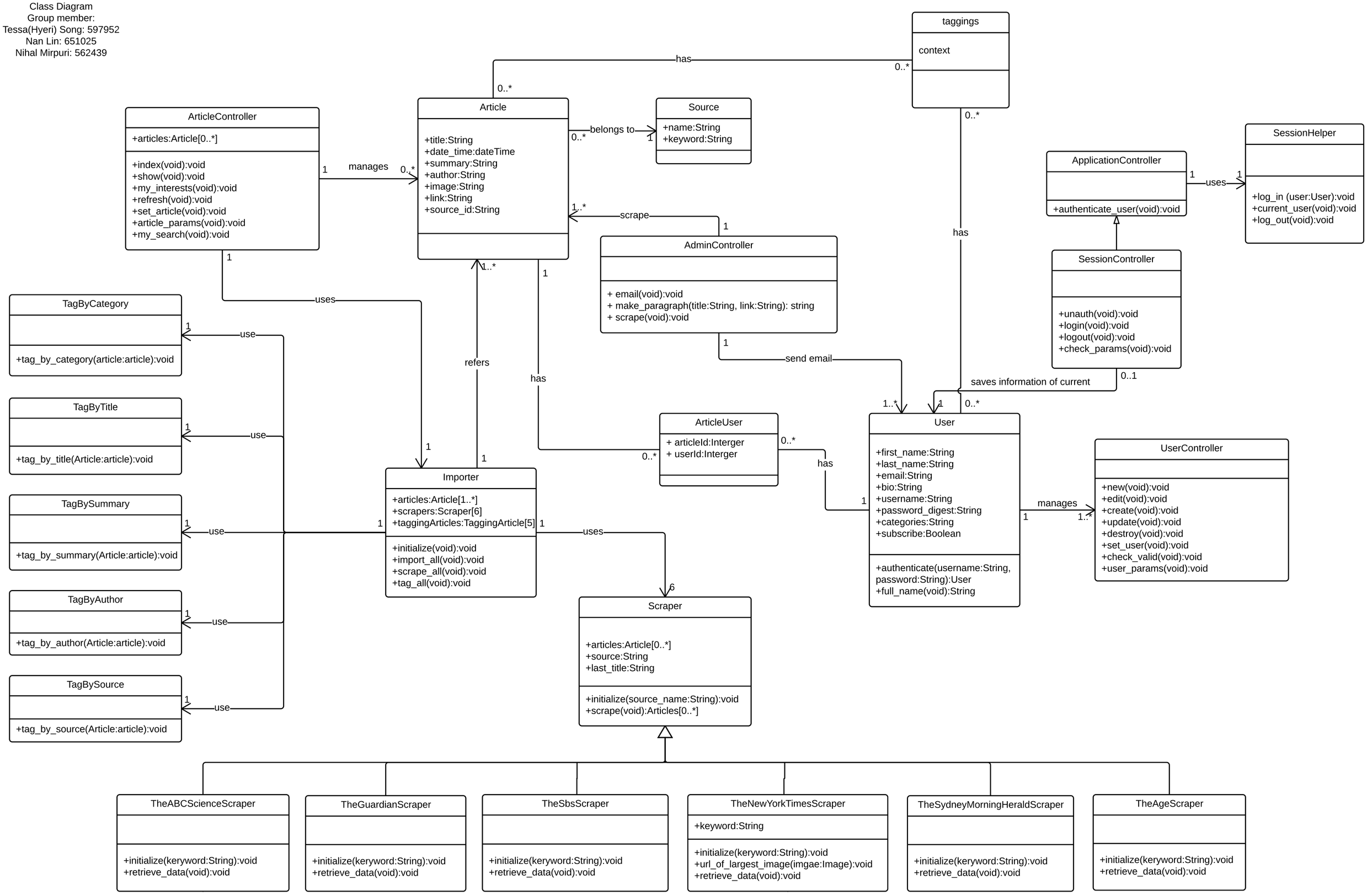
3) The value of spending time and effort developing a thorough design prior to implementation.

It is quite useful to develop a thorough design prior to implementation. It helps us to have a better understanding of the specification before we move to the actual implementation. As a good engineer, we should have a good sense about how the whole architecture looks like and then can make a valuable coding to meet the design requirements. If we do not have a proper design thinking before the actual work, we may keep changing our idea during the implementation and waste much time on it.

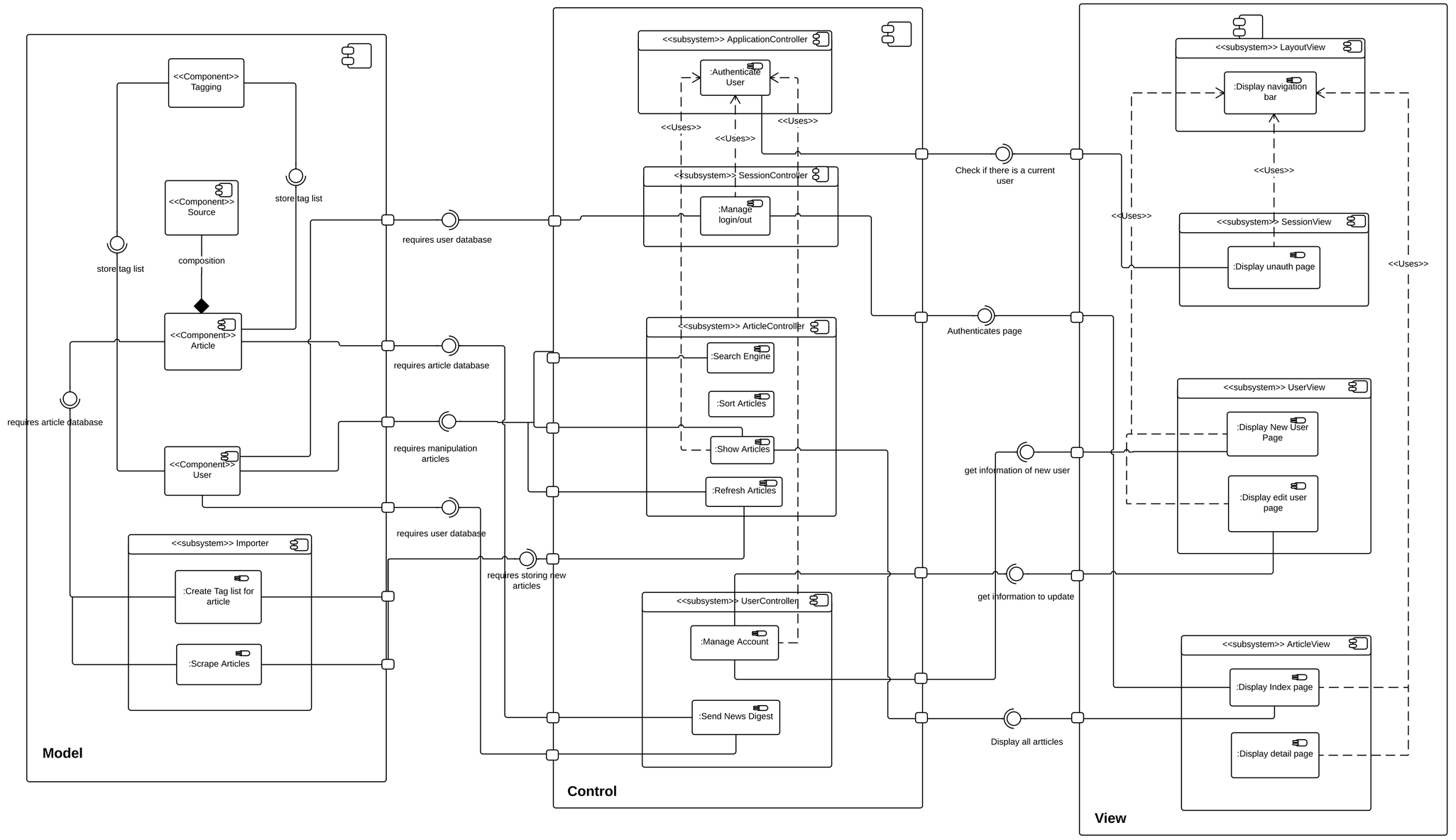
4) How well we followed the original design

Generally, we are following the original design implementation especially in terms of decentralisation although we made a few modifications to implement main functionalities such as emailing and searching. The reason why we made those corrections are that we confronted unexpected issues and problems while doing the actual implementation.

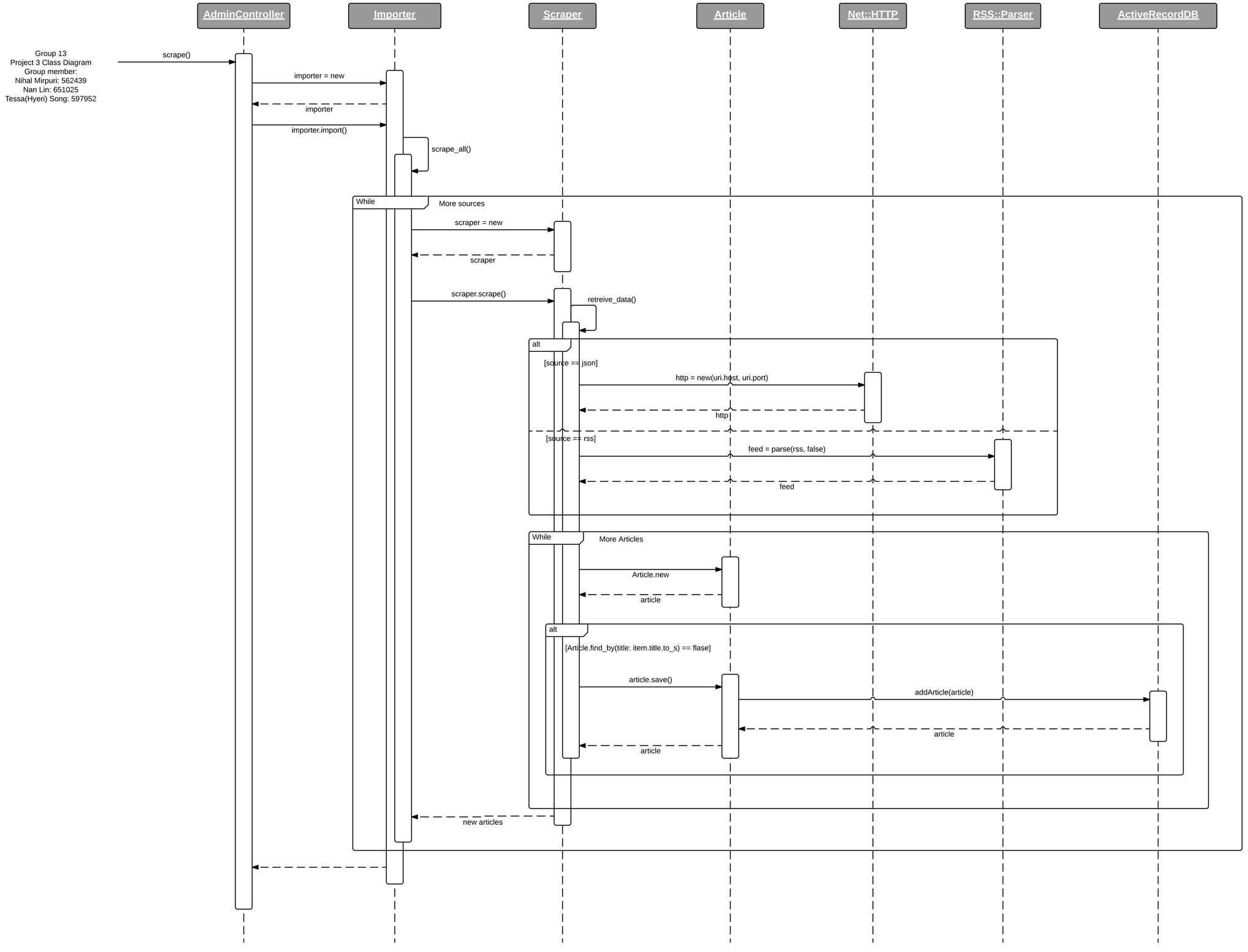
Group 13
Project 3
Class Diagram
Group member:
Tessa(Hyeri) Song: 597952
Nan Lin: 651025
Nihal Mirpuri: 562439



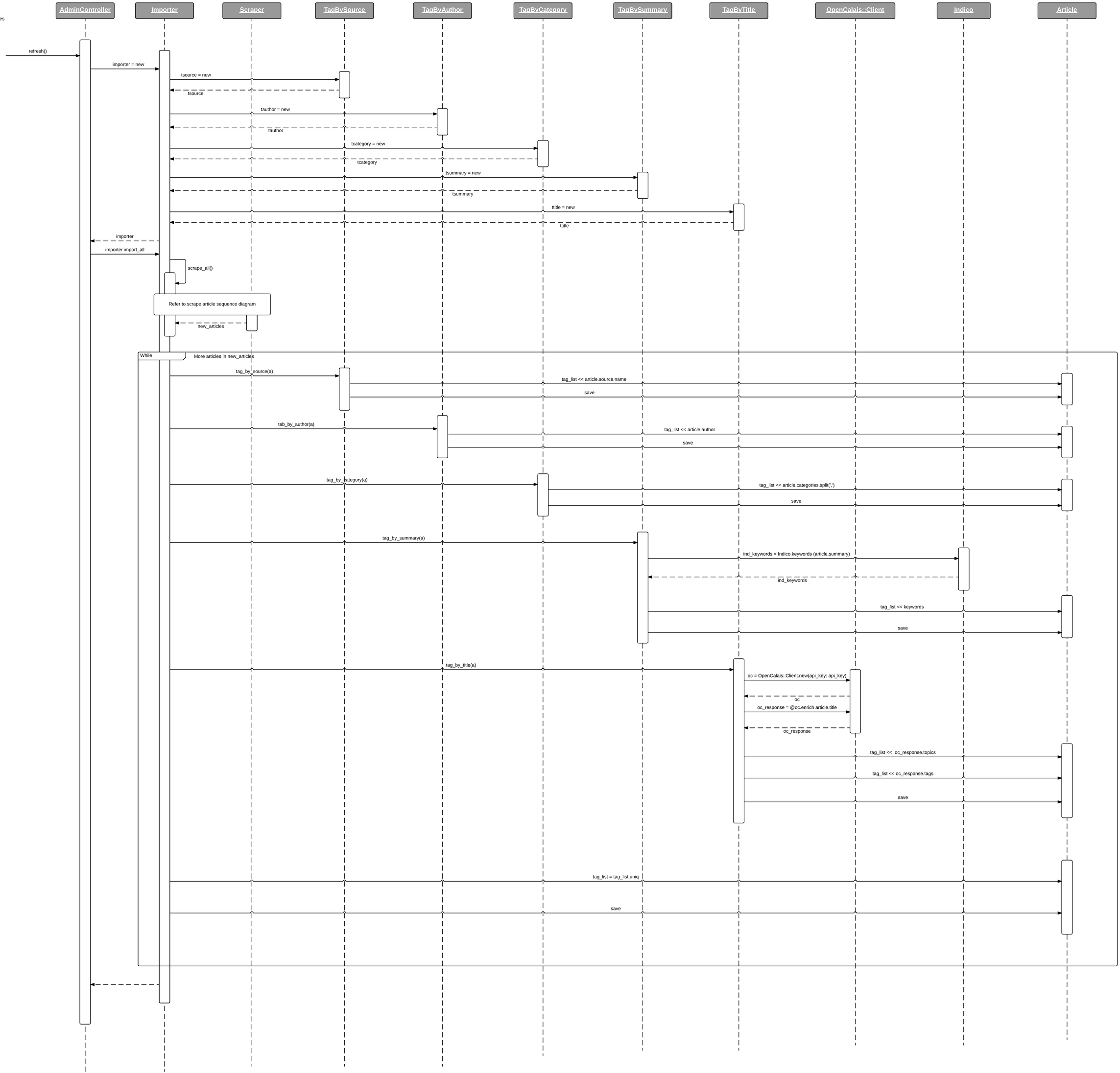
Group 13
Project 3
Component Diagram
Group member:
Tessa(Hyeri) Song: 597952
Nan Lin: 651025
Nihal Mirpuri: 562439



Group 13
Project 3 Class Diagram
Group member:
Nihal Mirpuri: 562439
Nan Lin: 651025
Tessa(Hyeri) Song: 597952



Group 13
Project 3
Sequence Diagram - tagging of articles
Group member:
Tessa(Hyeri) Song: 597952
Nan Lin: 651025
Nihal Mirpuri: 562439



Group 13
Project 3
Sequence Diagram -
Compiling and emailing of a news digest
Group member:
Tessa(Hyeri) Song: 597952
Nan Lin: 651025
Nihal Mirpuri: 562439