Reflection Report

We started our design aiming to follow a modular approach to ensure the concept of loose coupling and persist the practice of good software design for our Rails application. To facilitate this, we had our importers and tagging methods separated in different folders to ensure flexibility, and instantiated and called upon them in a separate class called the article processor. We believed that this approach was flexible enough to grant us the necessary separation required allowing us to bring about any further changes as necessary in the near future. Upon implementation though, as we proceeded to map out the design as classes, we identified several possibilities of further segregation along with the ability to incorporate the concept of open and close principle in a well-organized manner for our Importers and tagging methods. Hence, to try out these ideas we created a superclass for the Importers and Tagging Methods each and placed the code to validate the methods implemented by them. Following which, we replaced the tag processor with the Tagging module as we did not need a specific class to do the same process, same for the Importer module. All in all we did not make any major architectural changes, rather we just segregated the already designed classes slightly further to separate concerns up to a good extent.

This exercise of design instilled in us a new viewpoint towards software development as earlier we used to create a basic flow of logic to work through, following which we began implementation, debugging and re-structuring everything as we saw fit. But for this project, we created the designs first, outlined the relevant processes and their calls then mapped those designs into the required class diagrams which is a real life development approach. Although we faced slight difficulty initially trying to draft a complete picture for certain aspects like emailing unique articles only once to users, we did come quite close to our final design and believe that this was good experience for beginners in this field.

Also, we feel that the designing of applications beforehand is not exactly straightforward and requires several iterations in review to get a functional outline for our application. As there are many challenges and constraints involved in real life implementation it is difficult to point out the exact criteria and specifications needed to make the application work. Some of the points we identified were:

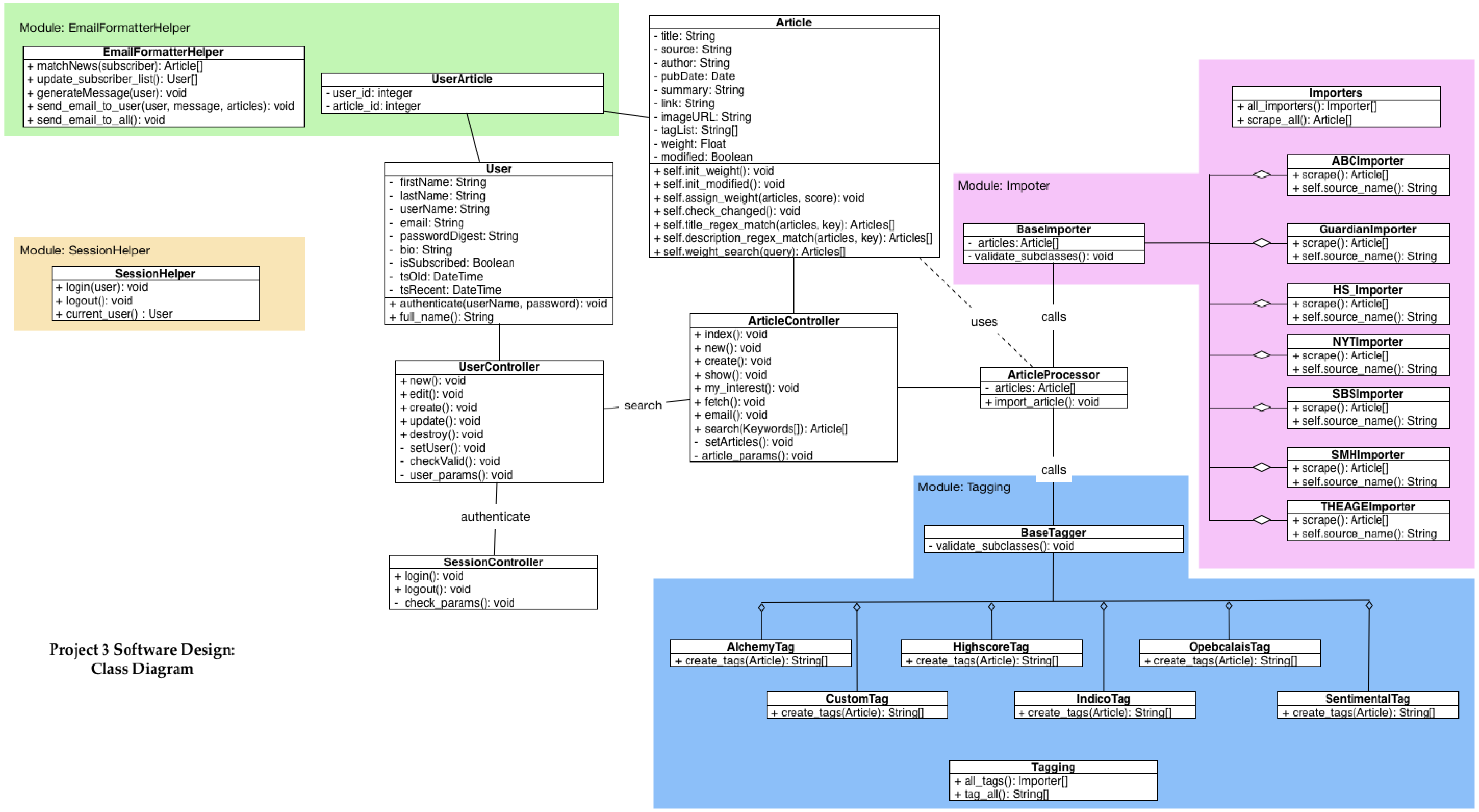
a. Initial designs are never concrete as they are subject to change as per the requirements and depth of scope required.

b. There is always a better design solution possible, but picking the right one which suits the timeline and feasibility is an important decision which involves analyzing the tradeoffs involved.

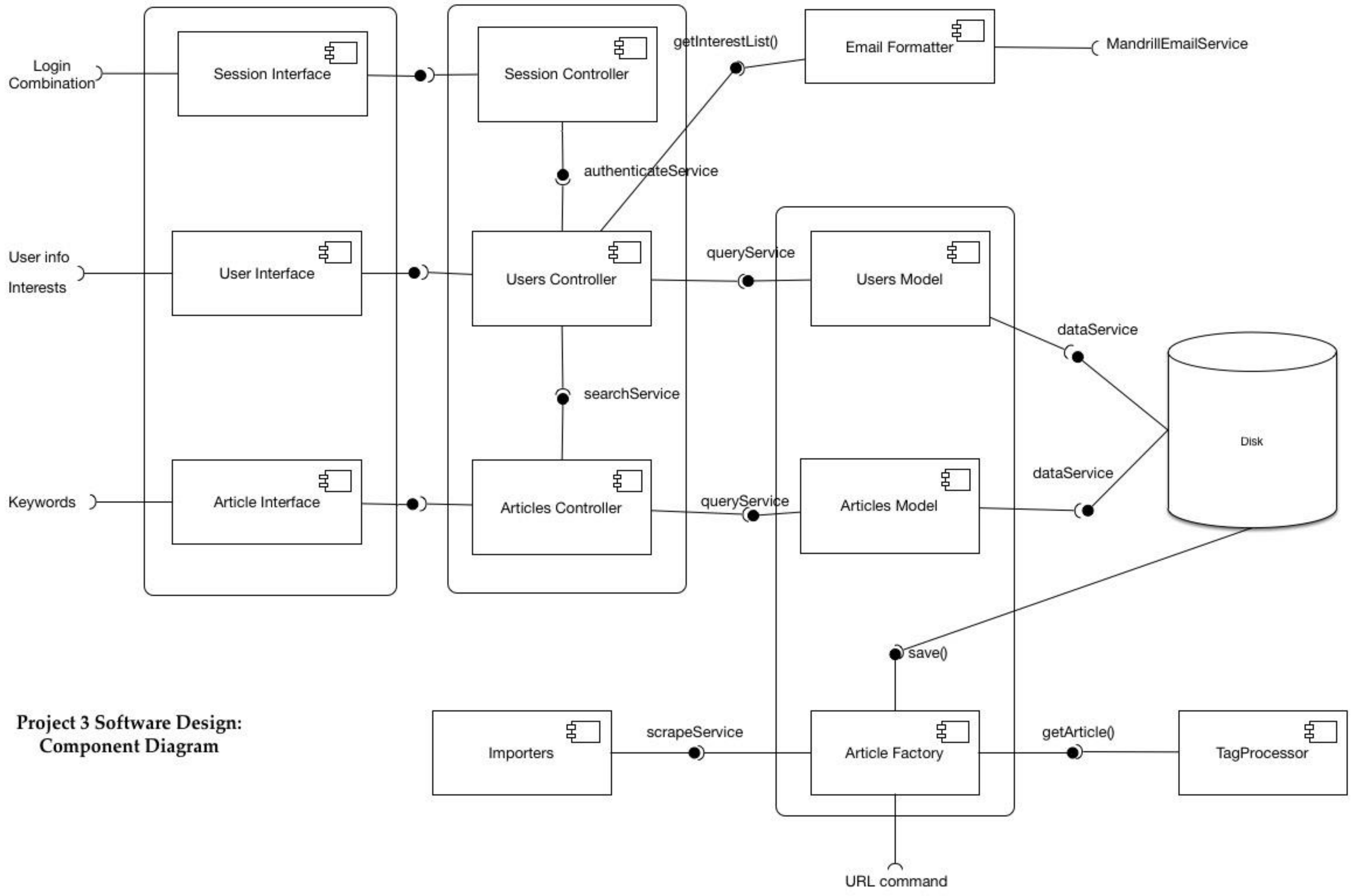
c. Designing of applications can start with the big picture in mind or by breaking the parts into small yet manageable aspects. Depending on the type of application and the amount of time and effort involved in its implementation, either of these approaches can be considered feasible. For this project, since we only needed to implement 3 additional features to our pre—existing application, it was not difficult for us to think of a flexible design keeping in mind the architecture we already had, making our task easier.

d. While translating design to code, we may discover aspects relating to the restrictions and usage of the programming language involved which were not anticipated during design time which makes the task difficult. Hence each design process needs to be considered based on the aspects provided by that programming language up to a certain extent to ensure feasibility in terms of implementation.

Working through this project while managing multiple assessments, these were some of the key points of Software modelling that we have identified so far. Although, given more time we could have automated some of the features while keeping in tune with the present requirements, we have decided to perform those upgrades at our own convenience as part of our future work during the summer.

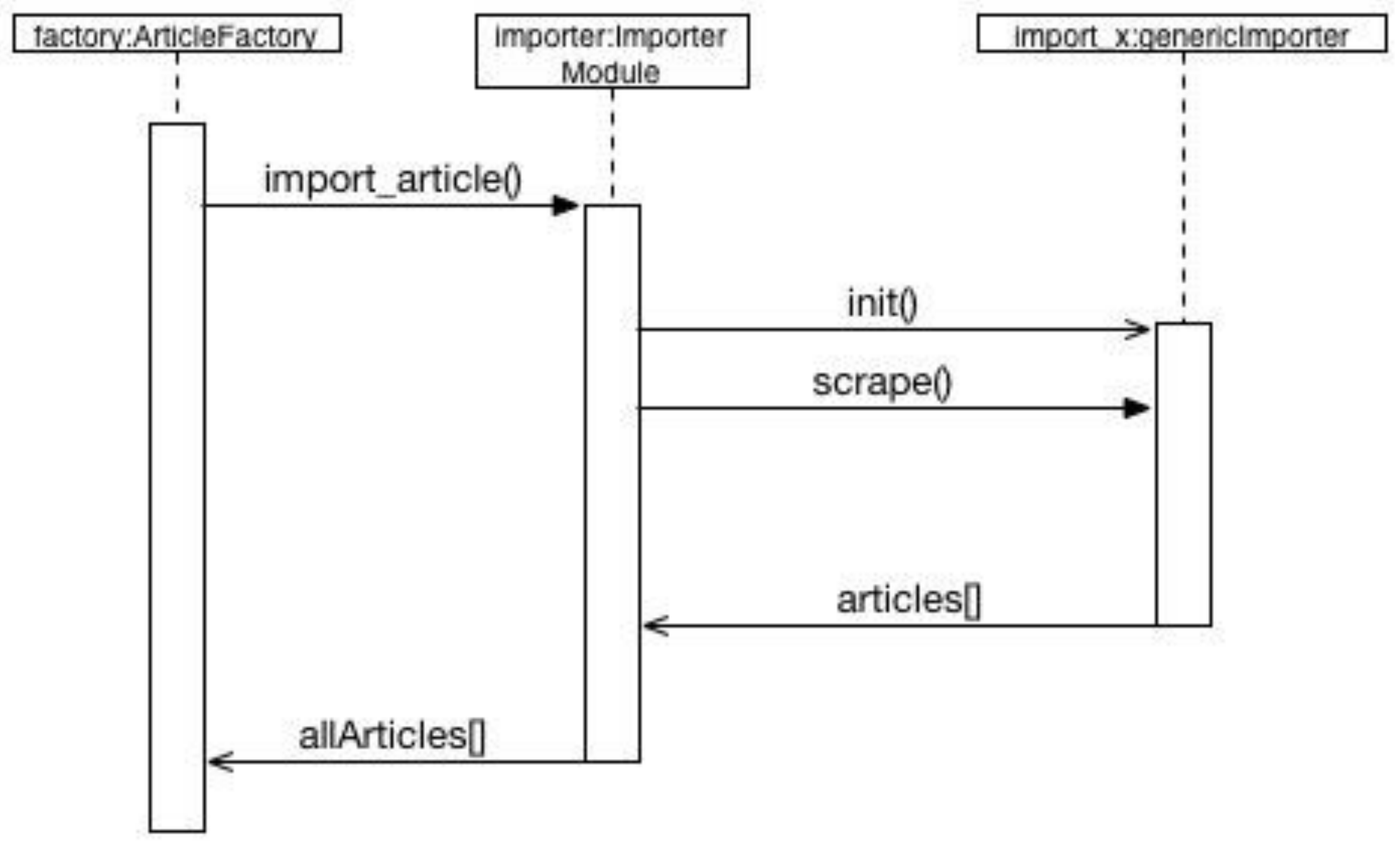


Project 3 Software Design:
Class Diagram

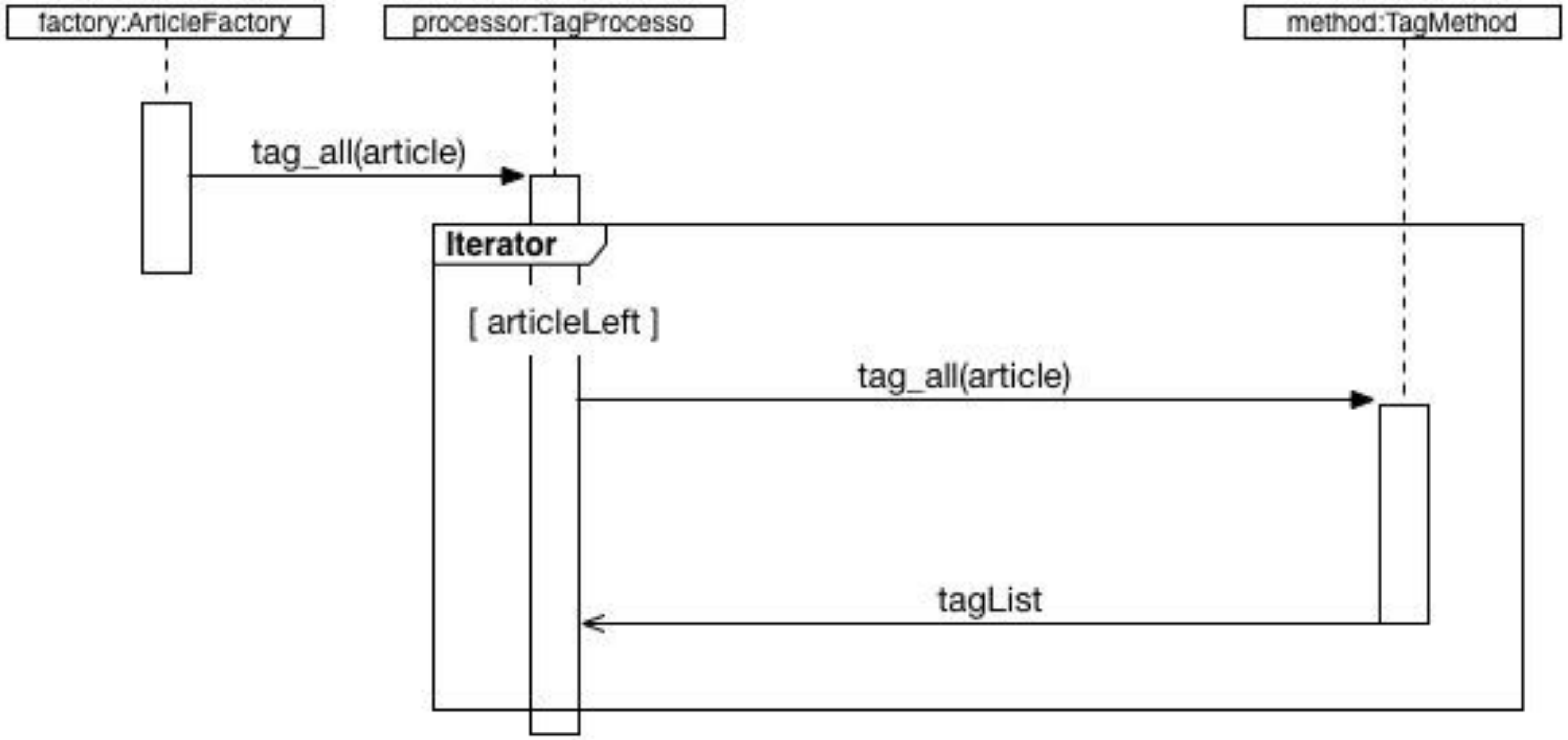


Project 3 Software Design: Component Diagram

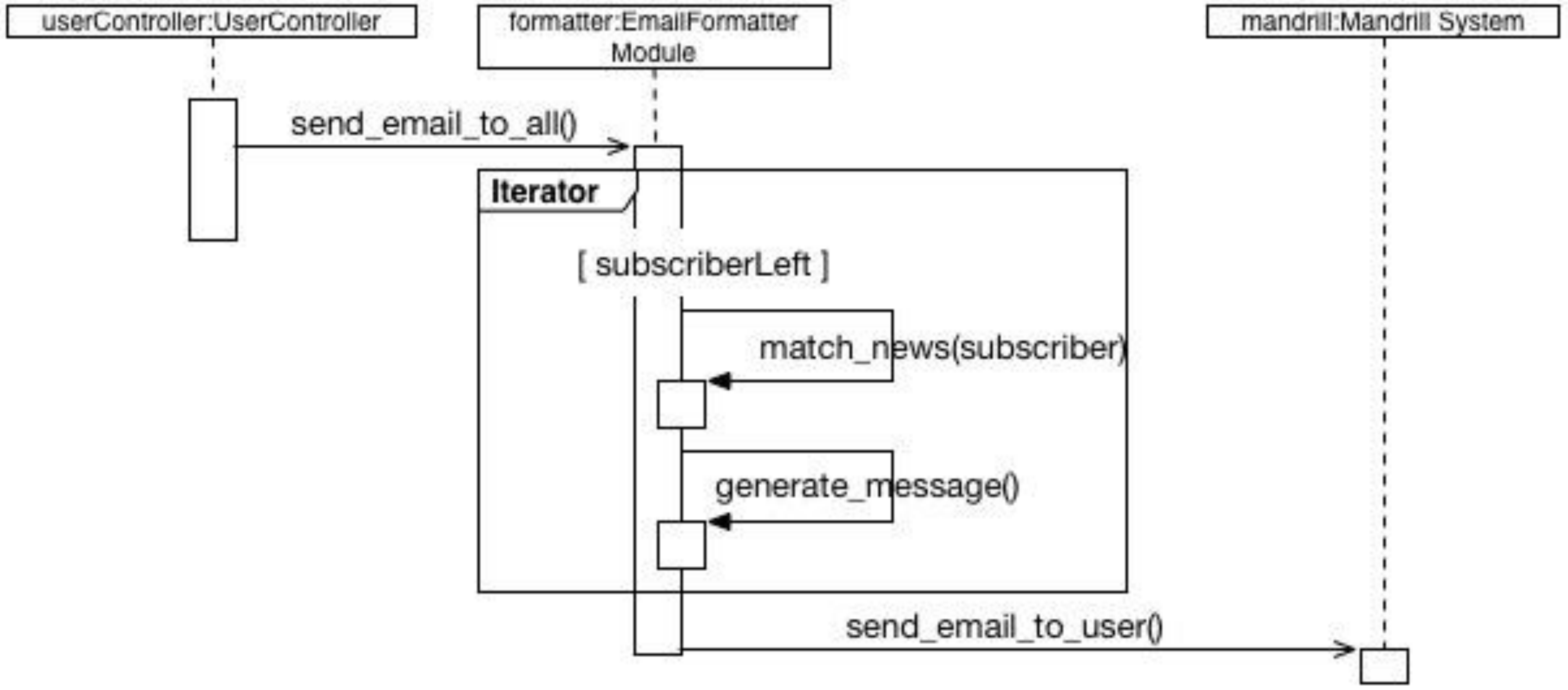
Project 3 Software Design: Sequence Diagram - Scraping



Project 3 Software Design: Sequence Diagram - Tagging



Project 3 Software Design: Sequence Diagram - Compiling and Sending Emails


userController:UserController
formatter:EmailFormatter Module
mandrill:Mandrill System
send_email_to_all()
Iterator
[subscriberLeft]
match_news(subscriber)
generate_message()
send_email_to_user()