# Debias

EECS4314 - Advanced Software Engineering

Professor Marin Litoiu

Prepared by - The Agile Alliance

**Team members**

Siddharth Bhardwaj

Michael Borg

Ruizhe Liu

Arshdeep Saini

Hengchao Xiang

# Table of Contents

# Abstract

This document presents the conceptual architecture of Debias software system. Examine the system in different architecture styles, and present a set of use case diagrams in unified-modelling-language (UML).

# Introduction

Debias is a news aggregation tool. It aims to solve the difficulty in finding articles with different viewpoints by aggregating daily news articles and displaying their tone to allow users to differentiate articles pertaining to a topic based on their tone. Users can search for news articles in the front end, and articles pertaining to these topics will be selected from the database of processed news articles and returned to the user displaying the name, author, date, of the article as well as scores indicating the tone of the articles. The users can further filter down the articles within the topic by the tone, which can allow them to get a wider range of viewpoints from a topic.

The motivation came from the need to make it easier for individuals to be able to filter their news feed in a way that will allow them to see articles with different tones, which could give them a more balanced understanding of the topic they are searching for.

# Functional Requirements of Debias

- Any registered client should be able to login.
- Any client should be able to register.
- Any client should be able to search for articles by a given keyword.
- Any client should be able to filter articles of a given keyword by their tone.
- Any registered client should be able to combine different tone filters.
- The tone filters can have one of the following values: anger, joy, fear, sadness, analytical, confident, tentative.
- The news article item will have the title, author, it's abstract, image, tone score data, link to the full article, and a feedback button.
- Any registered client should be able to provide feedback on what kind of bias they believe the article they read had.
- Any client should be able to view the Tone Score for each news article.
- Any client should be able to add or delete topics from their subscription list.
- The system's subscription service should be able to send periodic notifications (push notifications, email and SMS) to the client based on their subscription list.
- The system's central manager should have a scheduled timer that will periodically fetch raw data from the pre-set external news APIs.
- Administrators should be able to login.
- Administrators should be able to add or delete a manager from the system.
- A manager should be able to login.
- A manager can manage the Tone Filter.
- A manager can manage news articles available to clients.

# Use Cases of Debias


Figure 1. Use case diagram

# Non-functional requirements

- The system should limit api requests to 100 per hour for each client querying the news database for articles.
- The system should allow clients to sign-up or sign-in through federated identities (Facebook, Apple, Google).
- The system should notify the users to change the password every four months and the previous passwords can never be used again.
- The system should not be unavailable more than 1 hour per 1000 hours of operation.
- The system should be able to process 1000 user requests per second in peak load.
- The application should verify the user's request before allowing them to use its features.
- Installation of a new version should leave all database contents and all personal settings unchanged.
- Encryption in-transit (eg.HTTPS) and at-rest (eg.encryption of DB) will be used for privacy.
- Client's search history for articles and subscription topics will be stored locally on their device.
- The web app should be installable as a standalone app on the client's device.
- The front-end interface should be accessible to the client and have a score of 80+ on lighthouse performance score.

# High Level Architecture Overview

## System Functionality

Based on considering feasibility, social and economic benefits in the Project Proposal document, previously submitted, the functionality Debias will include the following: 1) provide scores by providing scores that depict the tone of the articles we can make it much easier for people to find articles that differ greatly on the same subject, and may offer the user a different perspective on the topic; 2) enable users to search the news with a specific tone for a topic; 3) allow users to give feedback on the accuracy of the results, which they get from our product. The feedback will be collected by the system and make the result more accurate; 4) allow all registered users safety and security interact/communicate with the Debias system over the world-wide-web (WWW), and store sensitive information such as login credentials, and email in the Debias system following the recommendations of the National Institute of Standards and Technology (NIST)- special publication 800-95: *Guide to Secure Web Services*[1]; 5) allow entities to publish and subscribe to topics; 6) system collects raw data (e.g. news metadata, tone scores) periodically(e.g update frequency every four hours).

## Description of Subsystems

According to the currently defined functions and the principle of single responsibility, we roughly define the following component:

Central Manager

First of all, it is responsible for handling the request, and invokes other components to process corresponding requests. Secondly, it also has the function as a scheduler, which is responsible for periodically triggering the task of updating article data and coordinating the function of applying user feedback to the article metadata. According to a predefined timer, it should regularly invoke the News processing pipeline to fetch the data from the external news API to perform the analysis and then update the article metadata at least on a daily basis.

News processing pipeline
This is an article processing module. Including Article content extractor, a job queue, and Result processor. Article content extractor will be responsible for obtaining the URLs of corresponding articles one by one from the preset news sources, extracting the content, storing them in the Article content database. The completed extraction task is marked in the job queue and then sent to the third-party tone analyzer for analysis. The returned results are processed by the Result processor and stored in the article metadata database. The data here will be the main data displayed by the client to the user.

Feedback learner
Responsible for collecting user feedback data, storing it in the feedback database and performing learning on it, it will periodically send requests to the central manager to apply user feedback.

Authentication Service
When the central manager receives a request related to user registration/login, it will call the authentication service to process it. We divide it into a single component in order to maintain the possibility of continuously improving the requirements for security and privacy.

Searching Service
According to the requirements of the use case, the Searching service provides three types of search functions: search for news based on keywords, add tone filters to the search results, and search for topic tags for subscription. It mainly relies on the article metadata database to realize the search function.

Subscription Service
When the central manager receives a request to add/delete a topic tag from the front end, it will call the subscription service to complete the request. The subscription service is also responsible for managing publishings and push subscriptions.

Management Service
Mainly responsible for fulfilling the management requests of administrators and managers. It contacts the user data database and article metadata database to realize the functions of managing user permissions, the visibility of the article and the tone filters.

## Comparison of overall software architecture

**MVC style**
Based on our software's functional requirements, the MVC architecture style is very compelling. The software has a modular approach and several different components communicating with each other. The front-end component handles client interaction and is

responsible for invoking the central manager. After this, the central manager is responsible for providing functionalities such as client authentication, news article search and feedback collection. Splitting the the components into Model, View and Controller, we get the following:

Model: The data model handles the application's state. For our software, this includes a database of all the news articles fetched from news APIs (such as NY Times etc.), a database of client information such as their credentials and feedback. This state is periodically updated as more news articles are fetched from new APIs and are then fed into a tone analysis tool that provides the core value for our software, an unbiased news feed. From this analysis it is apparent that the data model is relatively simple.
View: The front-end provides the view component for our MVC style. The view presents the client with a news feed that is refreshed in sync with the data model's periodic updates. The view allows clients to consume news articles ordered and filtered based on their tone, and presents them with the full article if they choose to interact with one. Clients also have the option to provide feedback on how accurately a news article's tone represents it's content.

Controller: The client's ability to search for articles based on topic and tone are facilitated by the controller in this architecture. As the client enters a search query for a topic they desire, the news feed is populated with the most relevant news articles. These articles are also ordered and filtered based on the client's preference. Throughout the search process, the controller continuously retrieves data from the model. As client's provide feedback on news articles and their tone accuracy, the controller invokes the data model to incorporate their input into its analysis method.

Studying this Model-View-Controller architecture style for our software, we see that the client's interactions with the data model are simple. The search and filter features rely on simple search queries performed on the data model. The data model itself also has a relatively stable state, updating only periodically and without any client input. The feedback provided by the clients helps the model passively by allowing it to better perform search queries over it's data. Considering these insights, an MVC architecture style demands higher code complexity and additional maintenance in general. It also underplays the role of the controller, as it only performs simple operations, while the data model handles the main intelligence of gathering appropriate news articles. In addition, MVC allows the view to directly communicate with the database, which reduces the security and at the same time makes the data model change affect both controller and views.Therefore, we decided to not continue with this architecture style, instead opting for the Three Tier architecture which better suits ours and our softwares needs.

**Three Tier style**
From an overall point of view, the entire application adopts the three tier architecture style, which is divided into the front end, business logic part and database part. The front end is mainly responsible for providing user interface, handling user interaction but without any business logic and direct communication with the database; business logic is mainly responsible for responding to front-end requests, applying business logic and sending requests to the database according to demand; and the database is responsible for data storage and extraction. The reasons for choosing this structure as the main architecture style are as follows:

- It can maintain the independence between the tiers. This provides certain support for fast development, easy maintenance and scalability. We can flexibly allocate

development tasks according to the areas that developers are good at. The front-end, business logic, and database levels can be developed concurrently using different programming languages, and communication through REST APIs. It will also improve development efficiency.

- Provides flexibility in scalability. Allows us to plan independent scalability plans between each tier based on the analysis of actual usage.
- Provide the possibility of achieving physical independence during deployment and improve the availability of the entire system. Considering that we need to periodically update the database and apply user feedback, high availability has higher value.

## Component diagram

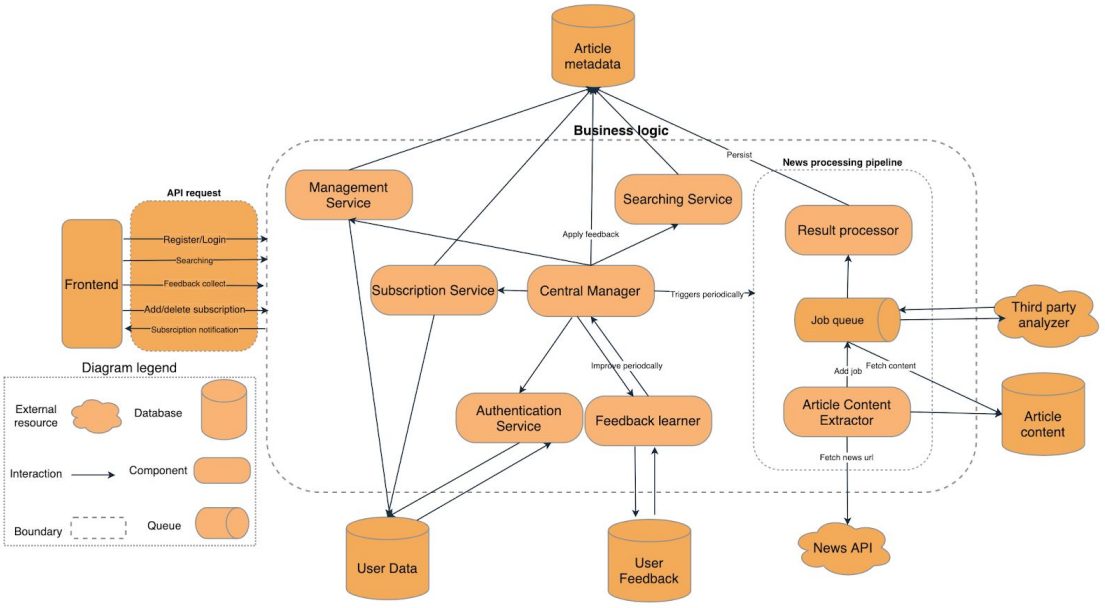


Figure 2. Component diagram

# Subsystem architecture

### News processing pipeline

The subsystem news processing pipeline uses pipe and filter style, as naturally, the data flow in our system could be modeled as a pipeline. The main reason for choosing this style to process data is that it is easy to maintain and enhance, which retains the possibility of strengthening our data processing capabilities. We can always achieve more powerful data processing functions by adding components in the pipeline.

### Comparison with Repository style

One of the most important features of our software is its ability to periodically refresh content available to our clients. The content primarily constitutes news articles that are filtered and ordered based on the client's search query and tone filter selection. This content is presented on the front-end as a news feed.

This periodically refreshing nature of the content requires an optimal architecture design to ensure the performance of the remains consistent. For this, we considered the repository style as an alternative:

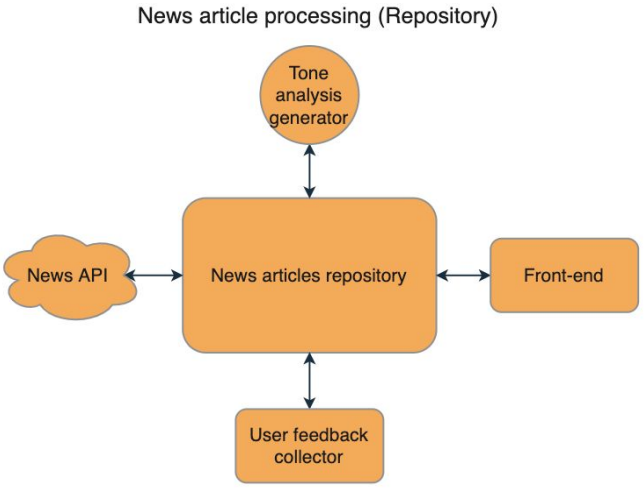


Figure 3. Repository architecture

Because there is only one common place to store the data, the data can be represented efficiently. And it can also reduce the overhead of transferring data among different components. But It lacks the ability to accommodate the changes in the data structure since this change will affect all the other components. And the other components are highly dependent on the central data store.

**Subscription service**

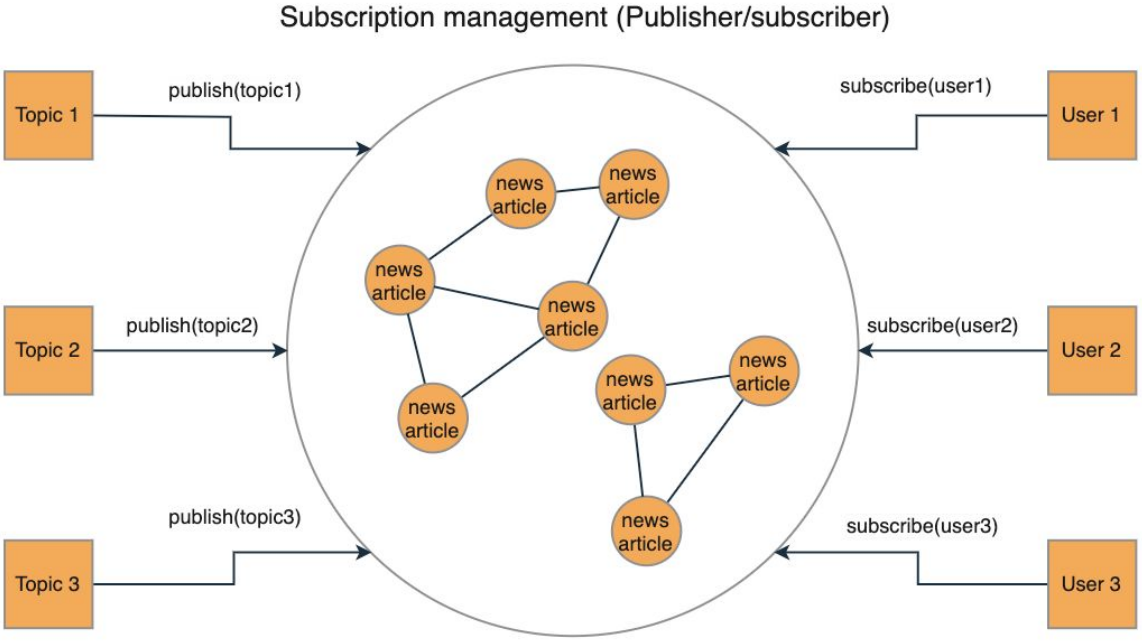


Figure 4. Implicit invocation architecture

For architecting the news subscription functionality, our software uses the Implicit Invocation architecture style. Clients express interest in subscribing to select news topics, and these topics periodically publish structured information for the clients to consume. Due to the flexibility of the Publisher/Subscriber architecture style, our software can add or remove new publishers (topics) as the clients express their interest to subscribe to them. It also gives clients the convenience of editing their subscription list, which in turn adds/removes publishers they are subscribed to. As apparent from these advantages, the Implicit Invocation architecture style is a good choice for designing the subscription service subsystem.

# Comparing with Similar Existing Systems

There is no existing system that exposes their architecture to the public. All current systems that are similarly owned by a private company, thus, this section is not applicable to our team's project.

# Activity diagrams and sequence diagrams

Use case 1: Client searches the article with tone filters.

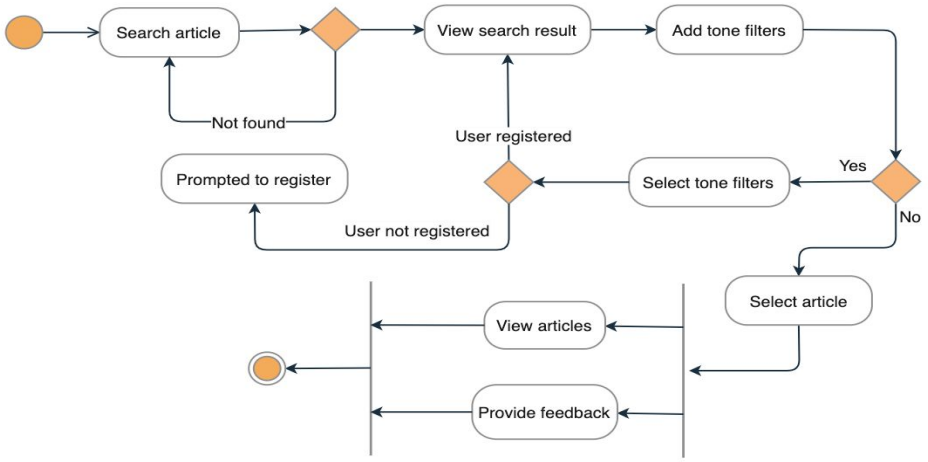


Figure 5. Activity Diagram

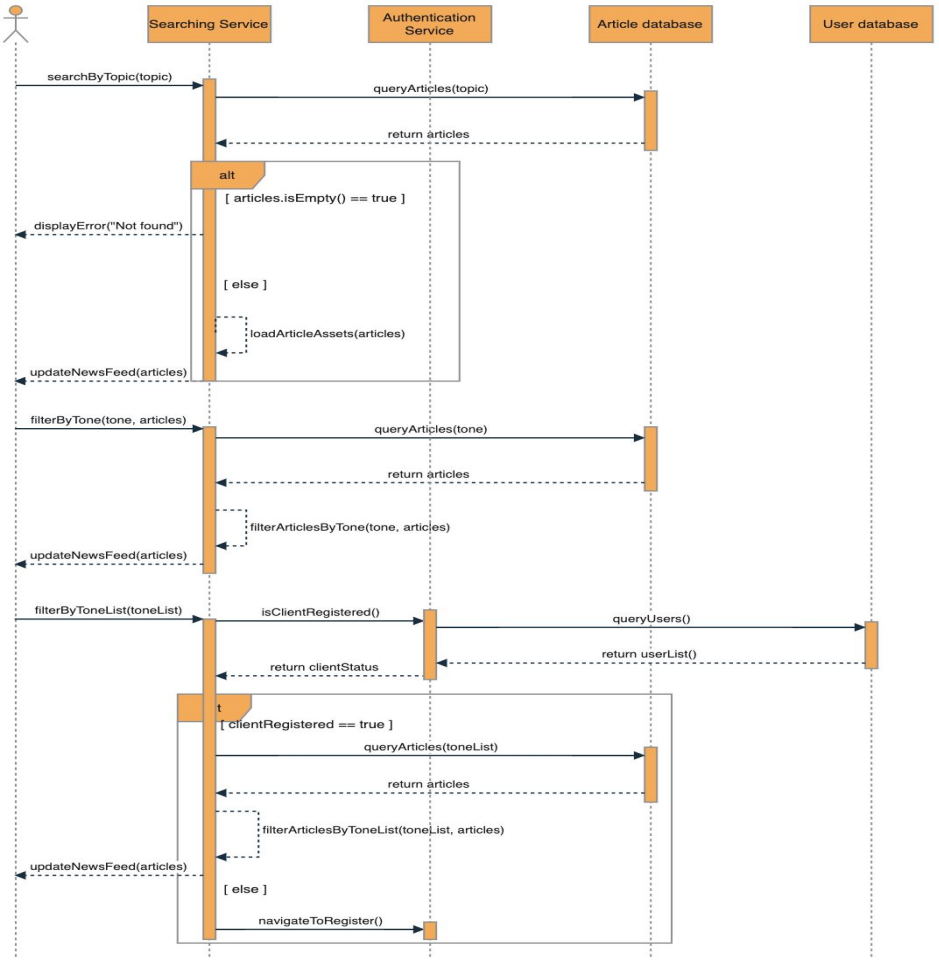


Figure 6. Sequence Diagram

Use case 2: Client adds topic tag to the subscription list.

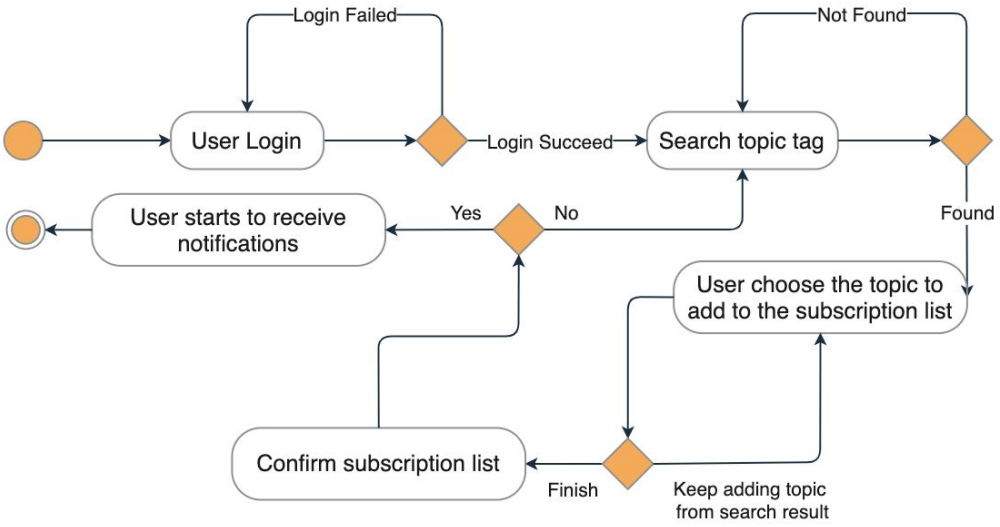


Figure 7. Activity Diagram

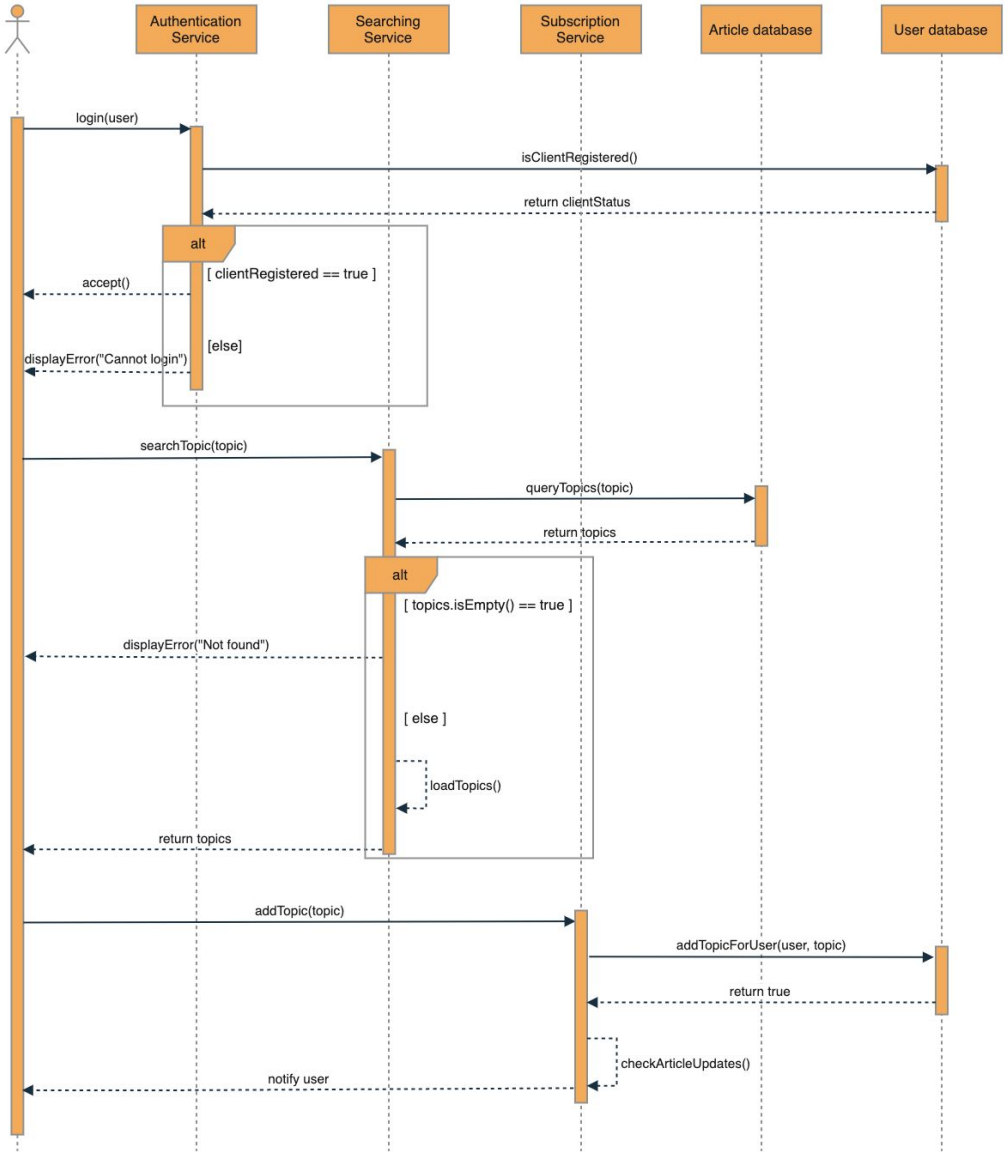


Figure 8. Sequence Diagram

# Concurrency Control

Being able to handle concurrency is necessary for any product used by thousands of users, in order to ensure that Debias is able to handle high traffic from processing large amounts of articles, and thousands of users trying to view the processed information debias will use a queuing system in addition to the concurrency control offered by our chosen relation database. News sources will be periodically querying for new news articles. Once new news articles are retrieved from a news source, they will be placed in a NOSQL database that holds the information from each article in a queue where it will await processing via the WATSON Tone Analyzer. Once the response has been retrieved from the WATSON Tone Analyzer, the response will be processed and inserted into a relational database. The selected database will handle reading and writing locks while the data is being inserted with an INSERT SQL statement. When a user searches for articles, they will query the table containing the article information, and the WHERE clause will contain the parameters indicating the date and topic being searched, along with any of the tones that might be used as filters, multiple users reading from the same table or row will not have issues as multiple process can read the same row from a table.

# Division of responsibilities

**Cloud Engineer:** Responsible for ensuring that communication between the frontend, backend, and the database works as intended, and that the connections are secure to avoid various attacks. The cloud engineer will be responsible for the initial set up of the chosen database on a cloud server.

**Backend Developer:** Responsible for creating the database schemas for the tables in which data will be stored as well as backend code that will get news articles from sources, process them, and insert them into the database. The backend developer should also create various functions that will query the database for information.

**Full Stack Developer:** Responsible for making sure that the API's in the front end are sending the correct parameters and receiving the expected response. The full stack developer will create and maintain the frontend code that will call the backend API's and get a response, as well as creating and maintaining the backend code that will call the necessary functions in the backend to create an appropriate response.

**Front End Developer:** Responsible for maintaining the front-end code, ensuring that the data received from the API requests is correctly displayed in the UI and that the user experience matches expectations.

# Software Process for Project

Since Agile is good for small projects[5], a lightweight and people-based approach rather than a plan-based, our team decided to use the Agile software process with scrum methodology.


Source:[4]

# External Interfaces of Debias

Some APIs integrated into Debias are: New York times API for articles, Google, Amazon, facebook API for Federated Identity, IBM Cloud SDK APIs for Tone Analyze. All the aforementioned APIs are RESTful API-- based on HTTPS requests and JSON responses**.** Thus, Debias since communicates with the above external interfaces using REST, it is a RESTful system. In essence, this allows Debias to support HTTP inputs from the user via request making implemented by the REST server.

# Data Dictionary

**Central Manager:** subsystem component responsible for routing the api and function as a scheduler-periodically triggering the task of updating article data and coordinating the function of applying user feedback to the article metadata.

**News processing pipeline:** subsystem component composed of three separate subcomponents- article content extractor, a job queue, and result processor. It is responsible for processing the article and presenting it to the user.

**Feedback learner:** subsystem component responsible for collecting user feedback data. Communicates with the Feedback database and Central manager to manage user feedback.

**Authentication Service:** subsystem component that communicates with the central manager to handle registration and login requests- following NIST guidelines to ensure compliance with security and privacy requirements.

**Searching Service:** subsystem component that relies on the article metadata database to provide search functionality. Uses keywords, tone filters, and topic tags (for subscription) to search.

**Subscription Service:** subsystem component that interacts with the central manager to handle requests to adding or deleting a topic tag from the front end, and manage publications and push subscriptions requests.

**Management Service:** subsystem component responsible for fulfilling the management requests of administrators and managers. Interacts with user data database and article metadata database to handle functions invovinging managing user permissions, article visibility to Clients, and visibility of tone filters available to Clients.

**User Data Database:** Stores the user information, including user credentials, subscription list information etc.

**Article Metadata Database:** Main database of the system, will store the article information with tone analysis scores, visibility, abstract, url and other necessary information.

**User Feedback Database:** Stores the user feedback data

**Article Content Database:** is a table of a single system database, which defines a scheme that will be used to store and retrieve text of articles, stored as an text file object in an object-based storage solution.

**Agile:** Software development in which requirements are discovered and solutions are discovered through the collaborative effort of self-organizing and cross-functional teams, and the customer/end user.

**User**: a Client of the System. Extents WebUser.

**WebUser**: a unauthenticated, anonymous user of the Internet.

# Naming Conventions

API: **A**pplication **P**rogramming **I**nterface. Is an intermediary software that allows two applications to communicate with each other.

REST: **RE**presentational **S**tate **T**ransfer. A software architecture style that defines a set of constraints when creating web services.

SOAP: **S**imple **O**bject **A**ccess **P**rotocol. It defines an RPC mechanism using XML for client-server interaction across a network

MVC: **M**odel-**V**iew-**C**ontroller. Architectural pattern commonly used in developing client-facing applications.

NIST- The **N**ational **I**nstitute of **S**tandards and **T**echnology. Is a government entity that helps organizations to better understand and improve their management of cybersecurity risk.

Scrum: is an agile framework that helps teams work together. Encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

# Conclusion

Debias is a news aggregation tool that aims to solve the difficulty in finding articles with different viewpoints by aggregating daily news articles and displaying their tone to allow users to differentiate articles pertaining to a topic based on their tone. Based on requirements generated from customer/user goals, the Debias system is designed with a central manager to handle news article ingestion, storage and processing. Also it can handle feedback learning-to make the system smarter, authentication, searching, publishing and subscribing- using pub-sub service system subcomponents, interactions

with external interfaces- NY times API, and handle concurrent control on API calls, and database read-writes using system subcomponents.

# Lessons Learned

In this phase we learned: Determining the data flow may be a good start for the component planning. After determining the system workflow, and considering the division of components in combination with use cases, it will be easier to sort out the ideas. I now understand better why many large-scale applications adopt staged development iterations. Defining too many use cases beyond the basic functions in the initial stage will increase the difficulty of system design, but complex use cases will indeed help the design to support complex functions. It may be that there are some trade-offs like choosing the architectural style.

# References

1. “Guide to Secure Web Services”. The National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf
2. “Tone Analyzer”. IBM Cloud Docs. Retrieved from https://cloud.ibm.com/docs/tone-analyzer?topic=tone-analyzer-about
3. Account Plans. IBM Cloud. Retrieved from: https://www.ibm.com/cloud/free
4. “Agile 101”. The Agile Alliance. Retrieved from: https://www.agilealliance.org/agile101/
5. “Software Processes Requirements”. Marin Litiou’s 4314 class slides. Retrieved from:https://eclass.yorku.ca/eclass/pluginfile.php/775191/mod_resource/content/2/Lecture%203-%20Software%20Processes.pdf
6. “Scrum”. Atlassian Agile Coach. Retrieved from https://www.atlassian.com/agile/scrum