Software Project
Iteration 3
Bear Feed

Mushfika Rahman
Sadia Nasrin Tisha
Muhammad Ashfakur Arju
Tingshuo Miao
Noorah Ashmeel

3rd December 2022

Abstract

Bearfeed is a website where users can create their own articles and event or visit all kinds of articles from Twitter and BaylorNews. It has some fun features like tags. This documentation describes the details of our project Bearfeed. It contains sections that include project Vision, project analysis and design, diagrams, implementation, Trello and Gantt charts, and Git.

1 Project Vision

The vision of this project is to create Baylor University news feed (Bear Feed), news feed portal for all Baylor University students and faculties.

Functional Requirements

- Users can log in to the system
- Users can register to the system
- User can ask for reset password
- Users can update interst list based on tag.
- Users can create posts in the text format
- Users have to add tags to the post
- Users can get notifications
- Users can search by tags
- Users can write comments
- Users can delete comments

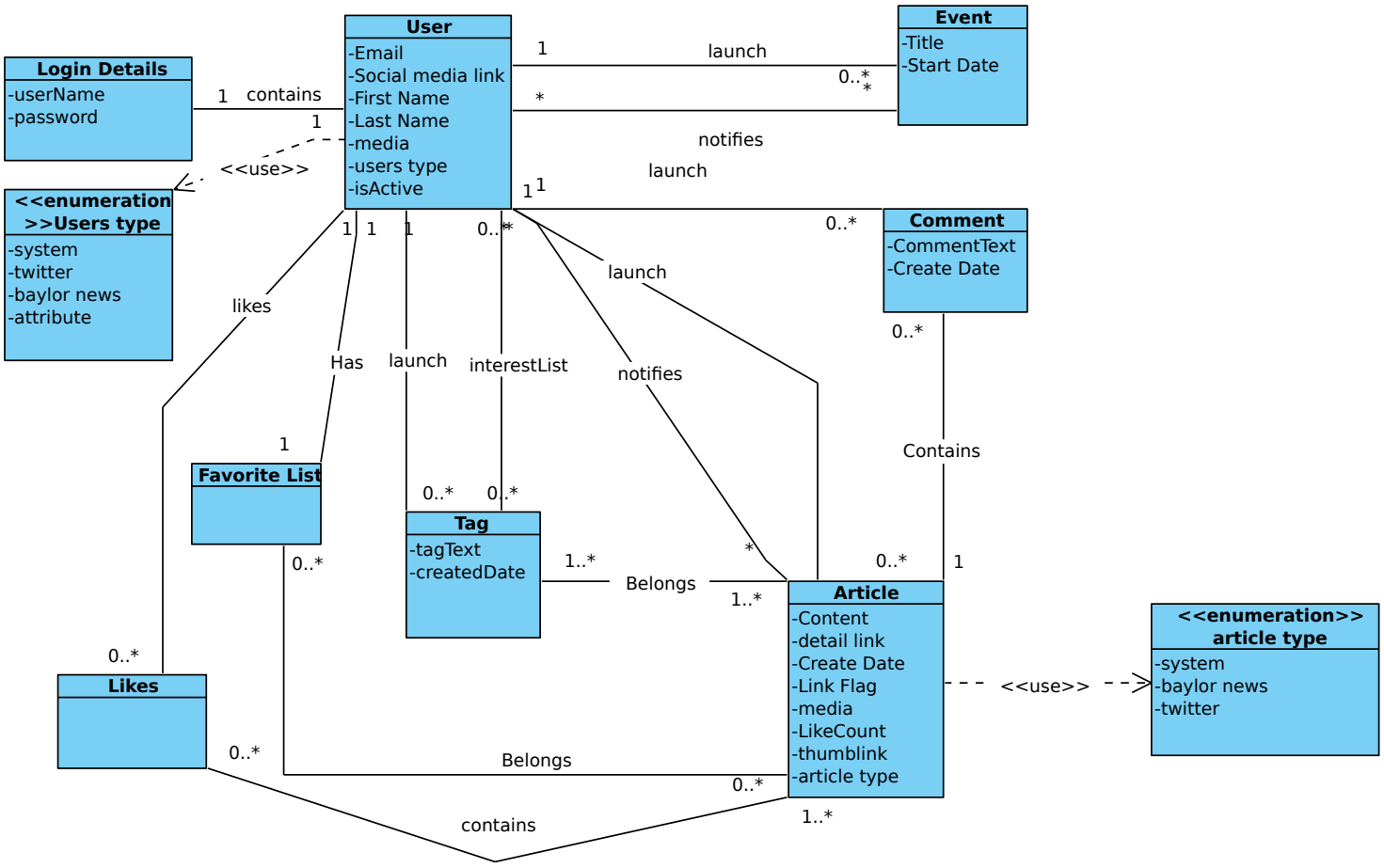
- Users can add posts add their favorite
- Users can view other posts
- Users can filter news feeds by tags
- Users can create new tags
- Users can share on social media
- Users can like other posts in the feed
- Users can create events
- The system can fetch information from the external source

Non-Functional Requirements:

- Authentication to user login
- Password encryption
- Auto-suggestion
- Message queue

2 Project Analysis and Design

2.1 Domain Model

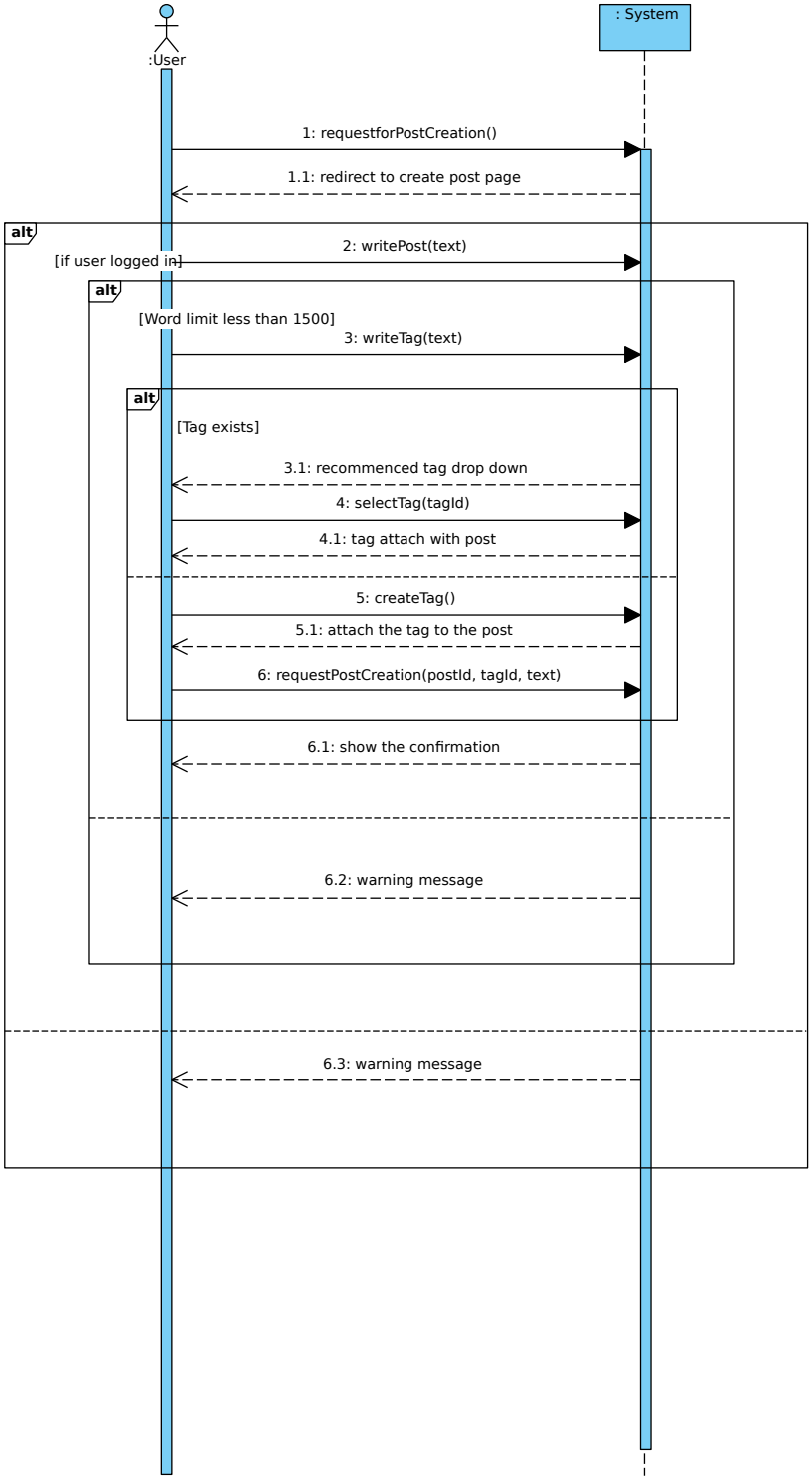


2.2 Create Article

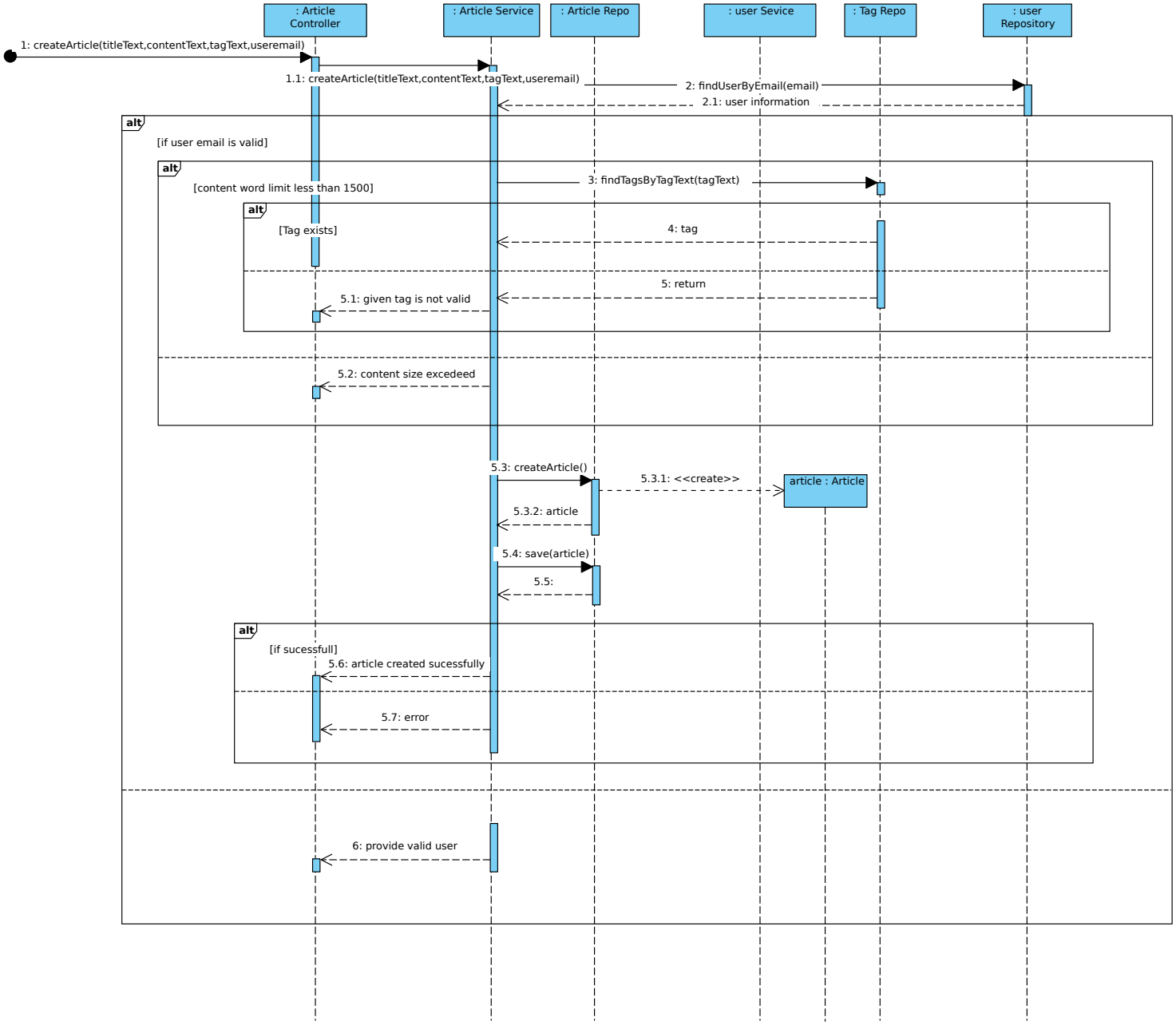
2.2.1 Fully-dressed Use Case-1: Create Article

Use Case: Create Article
ID: BN001
Actors: User
Preconditions: The user is logged in and clicks on the create article button
Flow of events: 1. User types/pastes his post in the text box. 2. If the article does not exceed the word limit (1500 characters) 3. User types for available tags. 4. The system auto suggests tags. 5. The user selects tag from suggestion. 6. The system attaches a tag from the article. 7. User clicks on create button.
Postconditions: The system will display confirmation message
Alternative flow: The user is not logged in.
Post Conditions: The system show an error "valid user info needs to be provided".
Alternative flow: 2.1 User writes an article exceeds word limit.
Post Conditions: The system will display a warning message.
Alternative flow: 3.1 User writes tag not available in the system. 3.2 User clicks on create tag option.
Post Conditions: The system attaches a new tag to the search field.

2.2.2 System Sequence Diagram(SSD): Create Article



2.2.3 Sequence Diagram(SD): Create Article

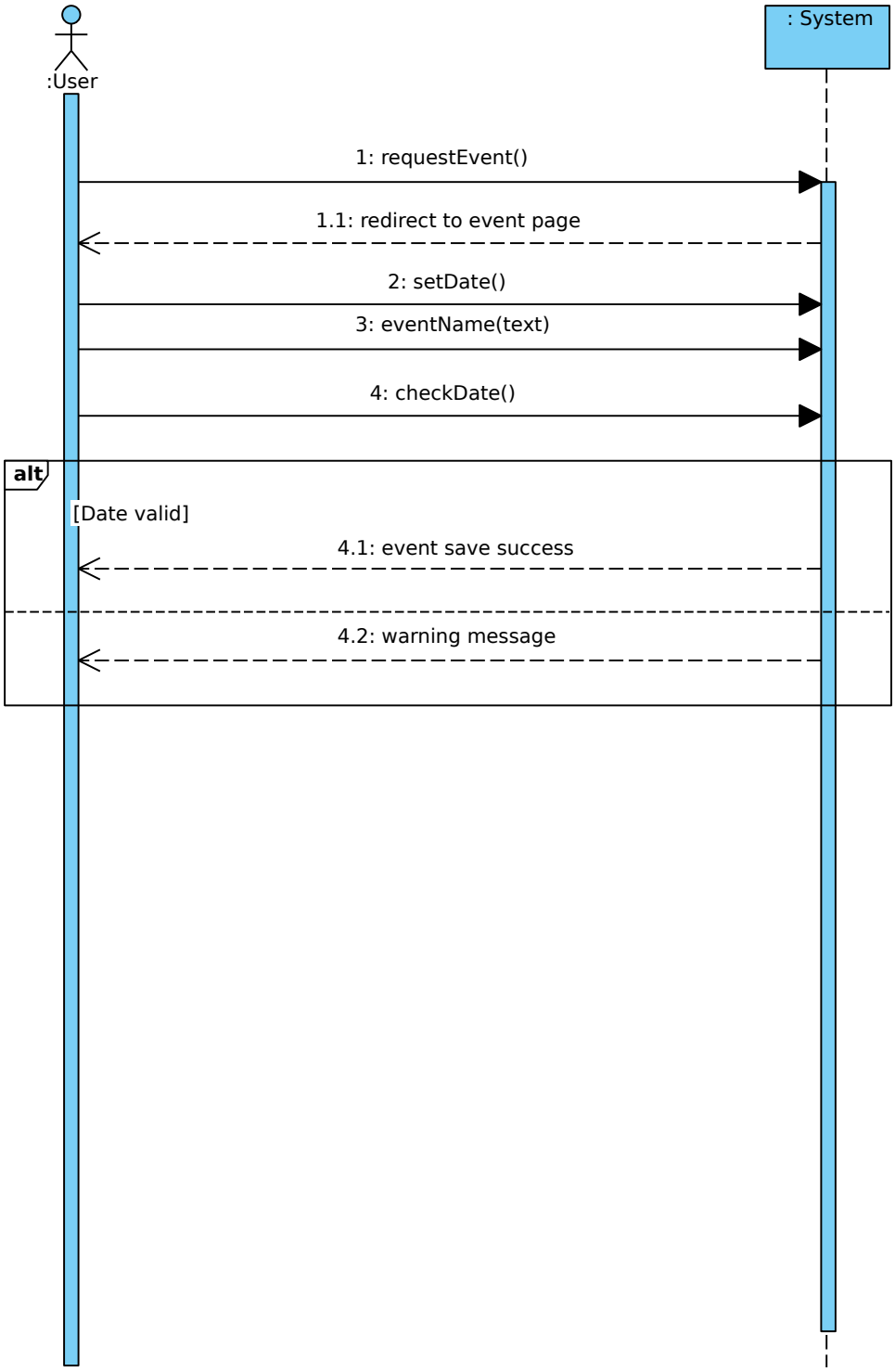


2.3 Create Event

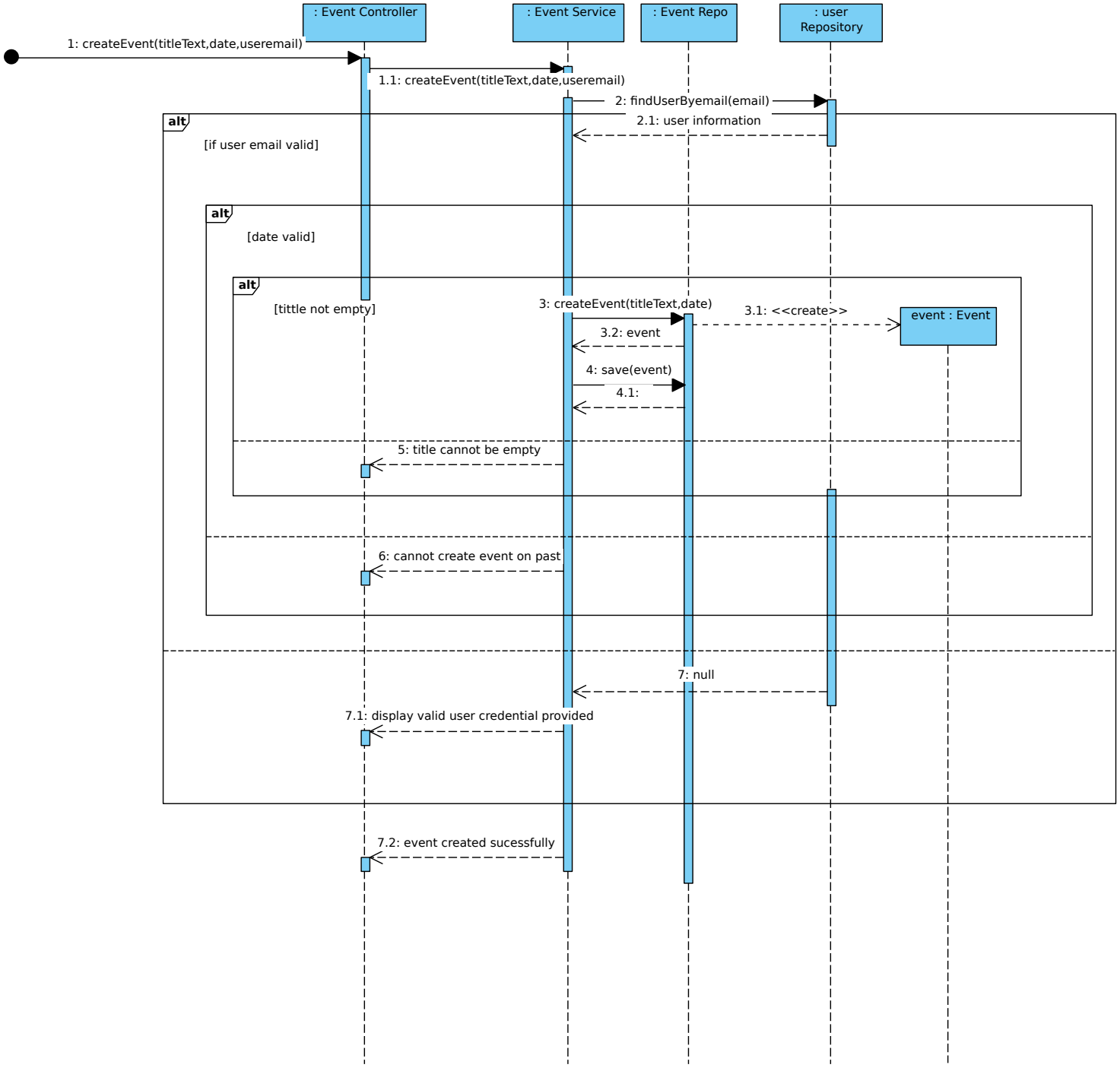
2.3.1 Fully Dressed Use Case-2: Create Event

Use Case: Create Event
ID: BN0002
Actors: User
Preconditions: The user is logged in and clicks on the create an event button
Flow of events: 1. User types/pastes title to the post in the text box. 2. The user sets a specific date for the event. 3. User clicks on create event button.
Postconditions: Display confirmation event creation is successful and display created event on the right
Alternative flow: 2.1 User sets a date that is in the past.
Post Conditions: The system will display a warning message.

2.3.2 System Sequence Diagram(SSD):Create Event



2.3.3 Sequence Diagram(SD):Create Event

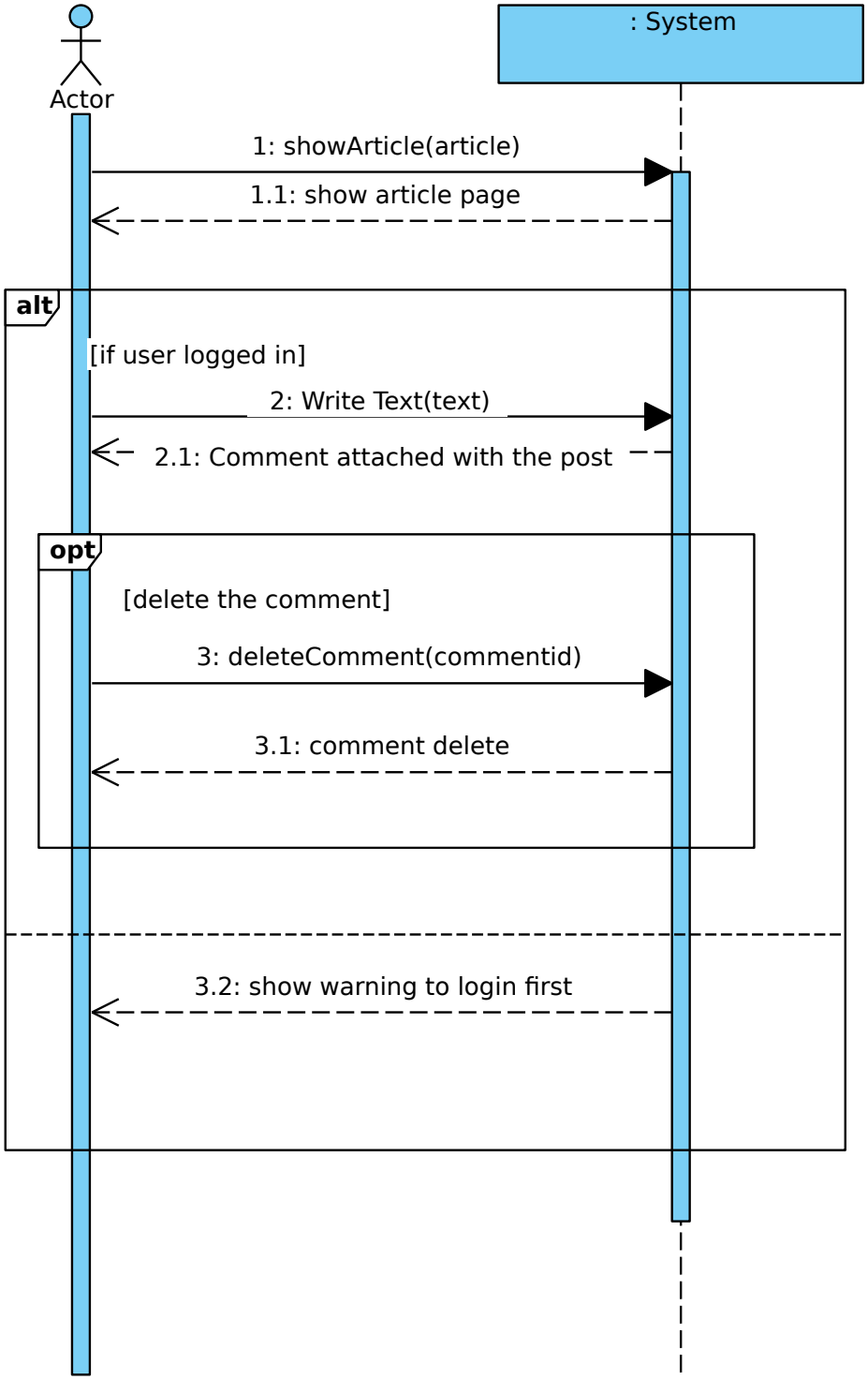


2.4 Create Comment

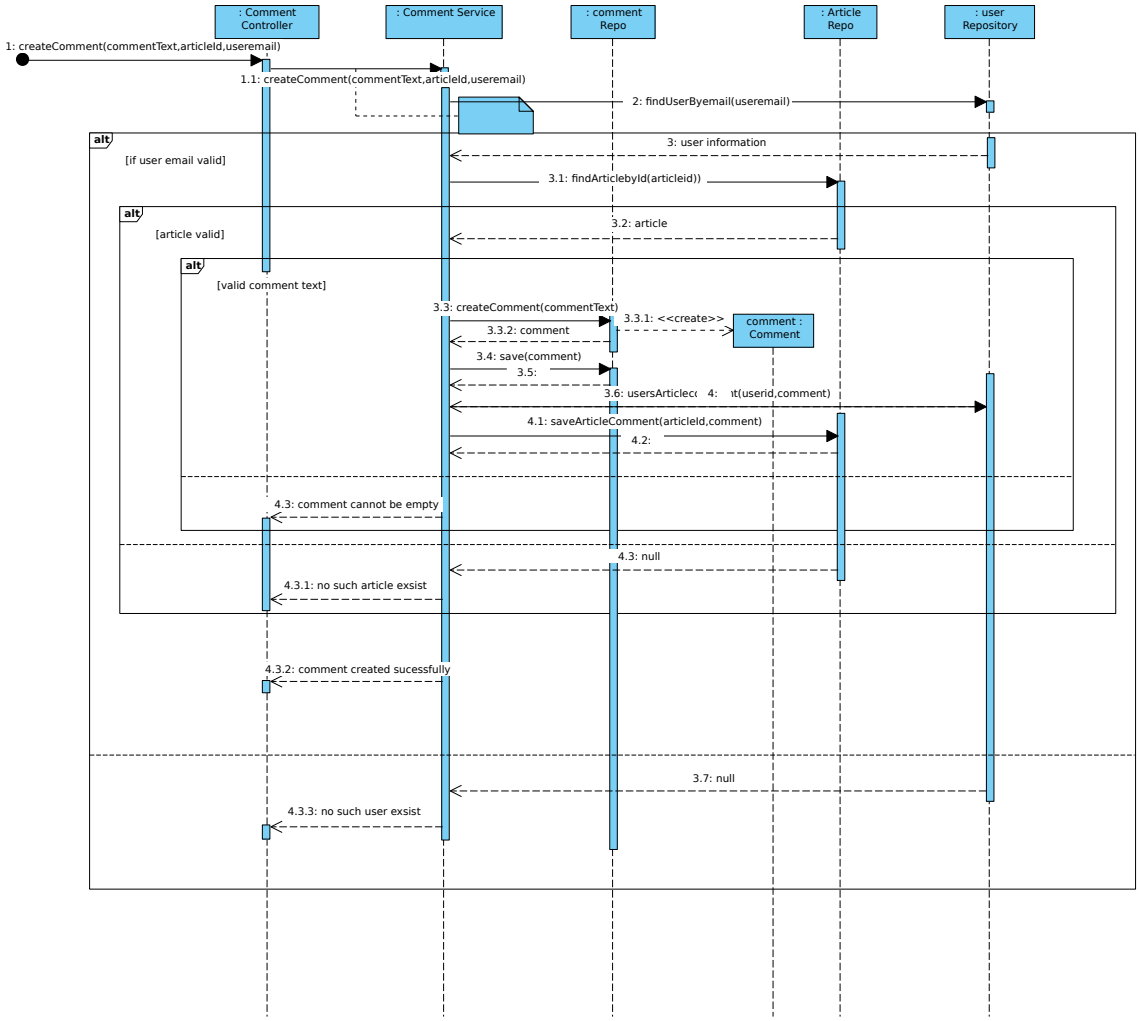
2.4.1 Use Case-14: Create Comment

Use Case: Create Comment
ID: BN014
Actors: User
Preconditions: User is logged in the system and in the view full article page.
Flow of events: 1. The user writes text to the comment. 2. The user clicks on the comment button. 3. The system pops up confirmation message.
Postconditions: The comment is attached to the post.
Alternative flow: The user chooses to delete the created comment.
Post Conditions:. The system removes the comment from the list.

2.4.2 System Sequence Diagram(SSD):Create Comment



2.4.3 Sequence Diagram(SD):Create Comment

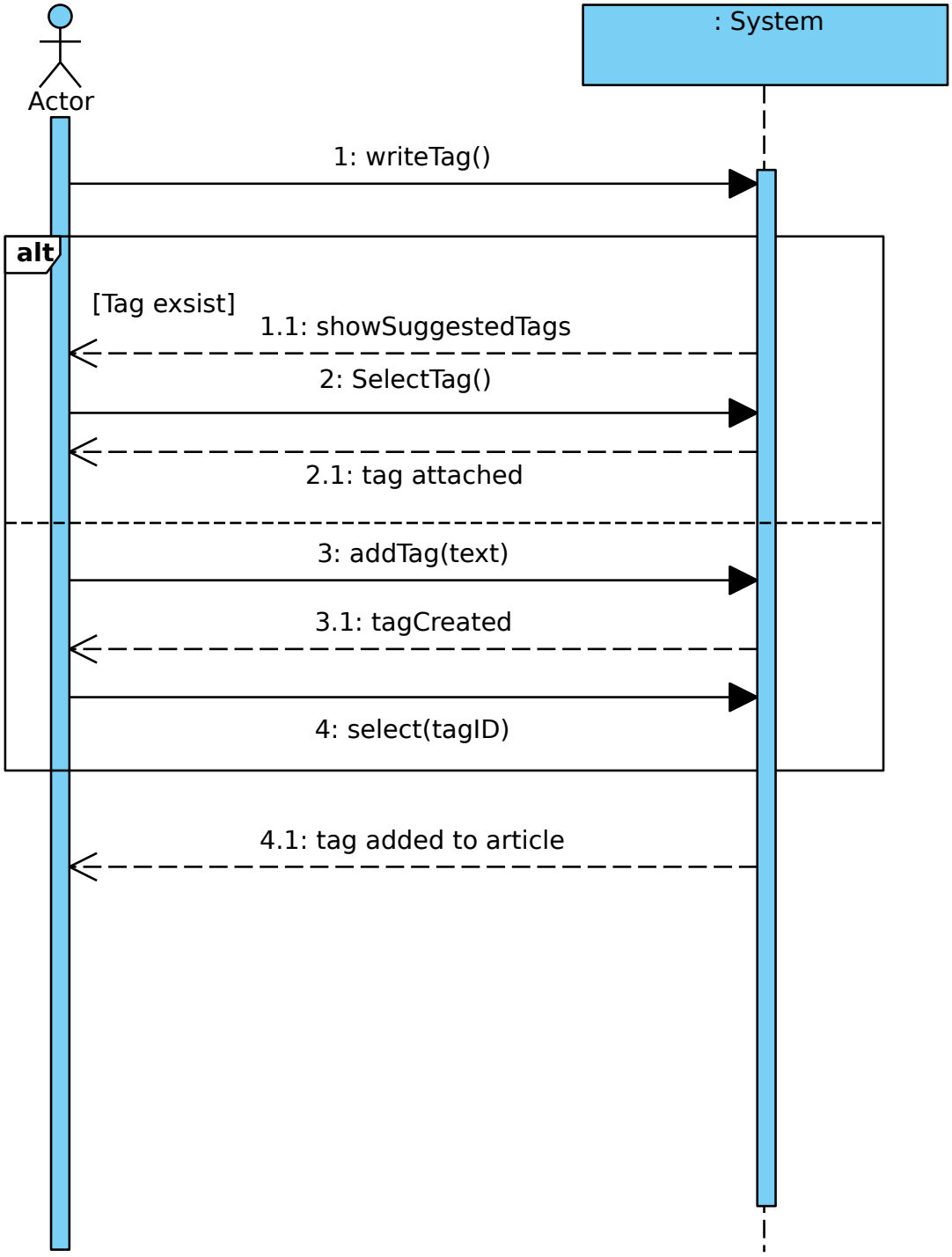


2.5 Create Tag

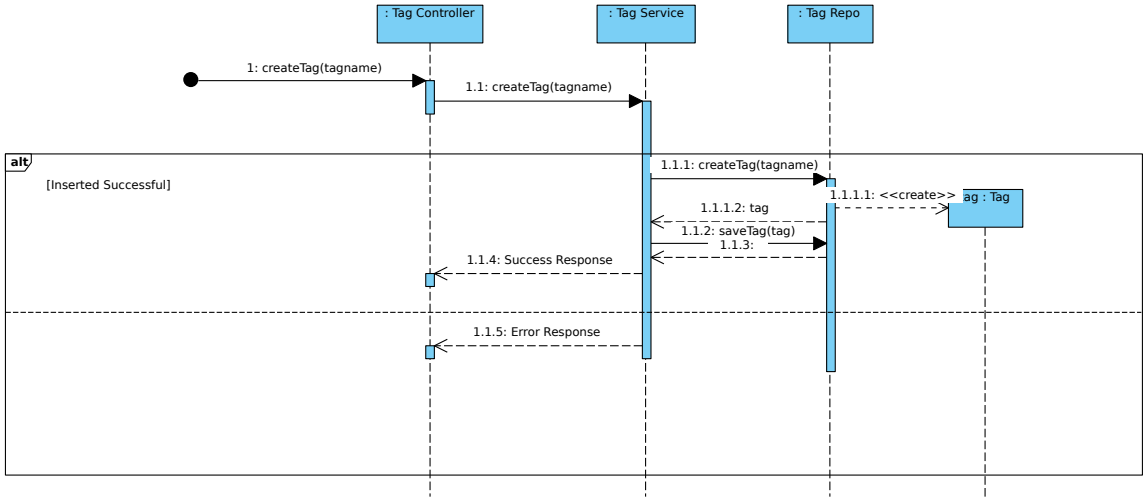
2.5.1 Use Case-10: Create Tag

Use Case: Create Tag
ID: BN010
Actors: User
Preconditions: User must be logged on and creating an article in create article page.
Flow of events: 1. User writes text for the tag in the tag box. 2. If the tag already exists. 2.1 System auto-suggests existing tags of the systems. 3. Else, the user writes a tag that is not in the system. 4. The user clicks on the create tag option. 4.1 The tag creates. 5. Add the tag in create article tag box
Postconditions: The system adds the tag to the list of already created tags by the user.
Alternative flow: The user can leave the page at any time.
Post Conditions: The system redirects to the desired page.

2.5.2 System Sequence Diagram(SSD):Create Tag



2.5.3 Sequence Diagram(SD):Create Tag

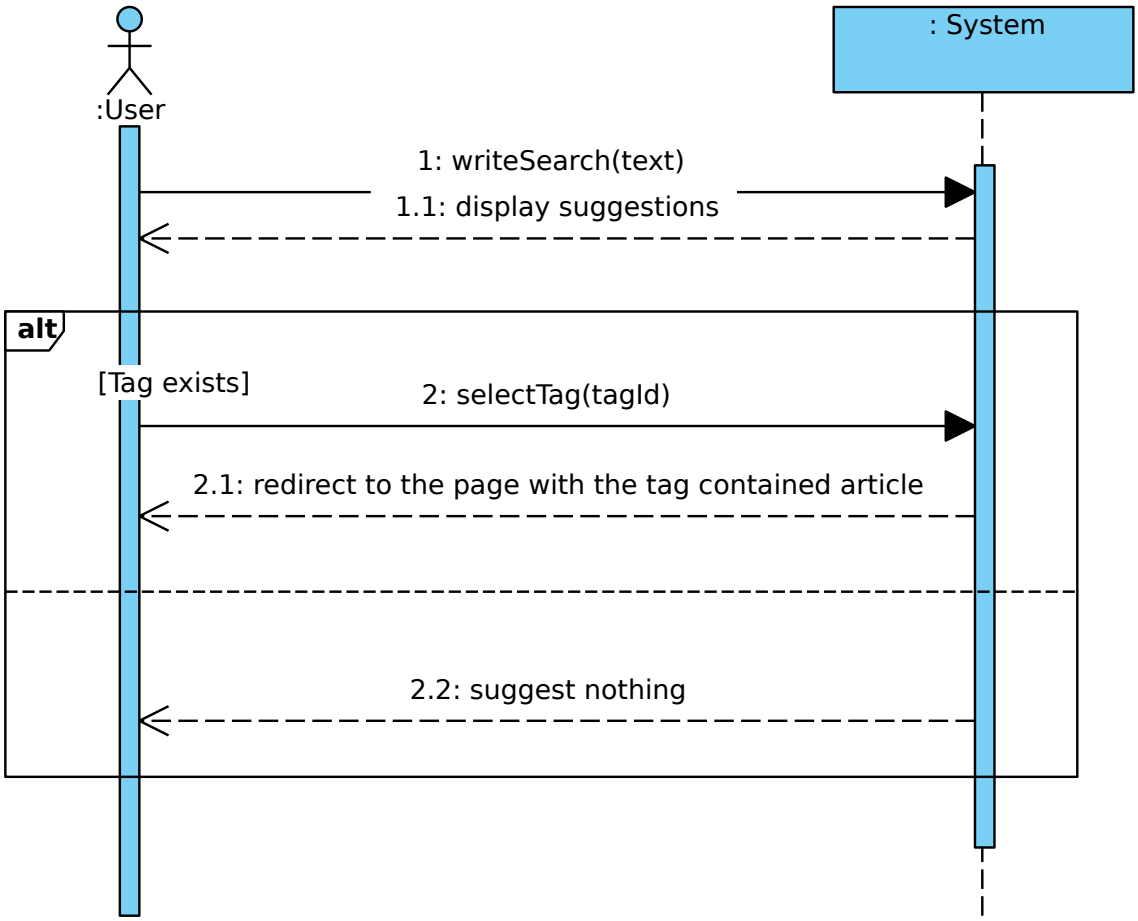


2.6 Show Articles By Tag

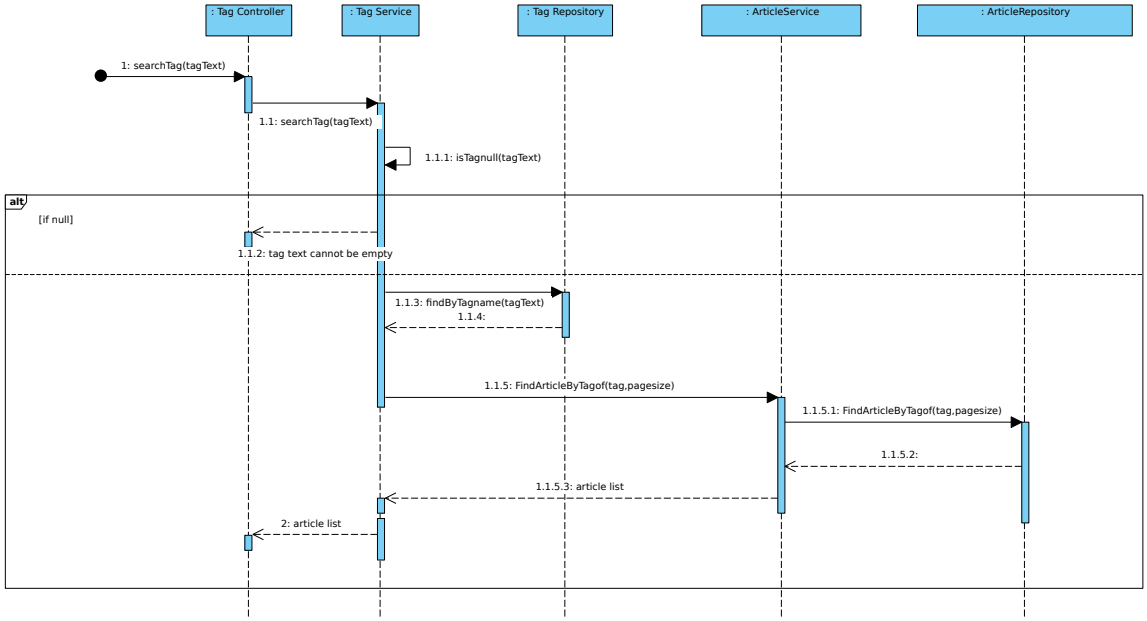
2.6.1 Use Case-5: Show Articles By Tag

Use Case: Show Articles By Tag
ID: BN005
Actors: User
Preconditions: User is logged in the system
Flow of events: 1. User writes the tag text in the search box. 2. The system auto suggests tags. 3. User selects tag from suggestions. 4. User clicks on the search icon
Postconditions: User is redirected to the page loads preview of the articles with the selected tag.
Alternative flow: 4.User writes a tag that doesn't exist in system
Post Conditions: The system shows no suggestions
Alternative flow: User might not click the search icon.
Post Conditions: The system will keep the user in the current page.

2.6.2 System Sequence Diagram(SSD):Show Articles By Tag



2.6.3 Sequence Diagram(SD):Show Articles By Tag

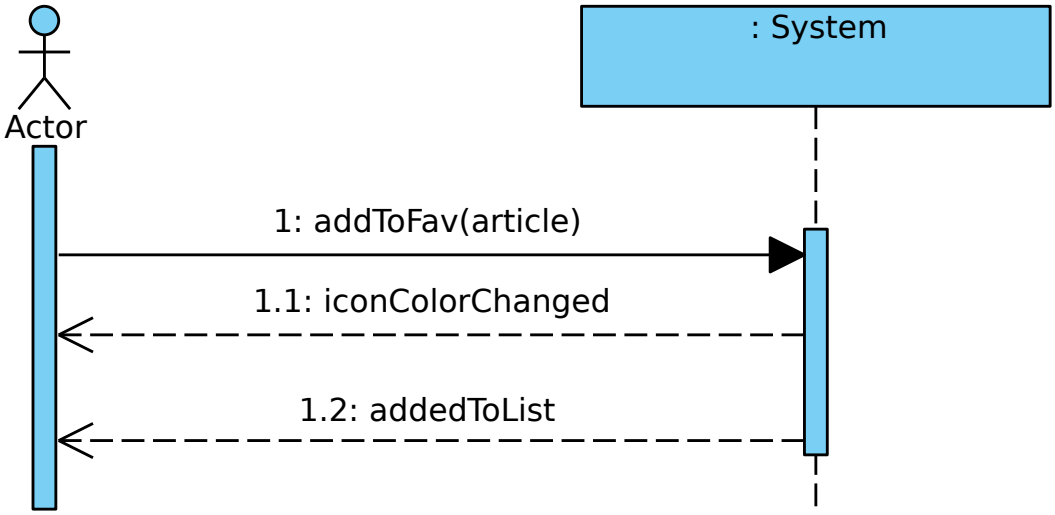


2.7 Add to favorites

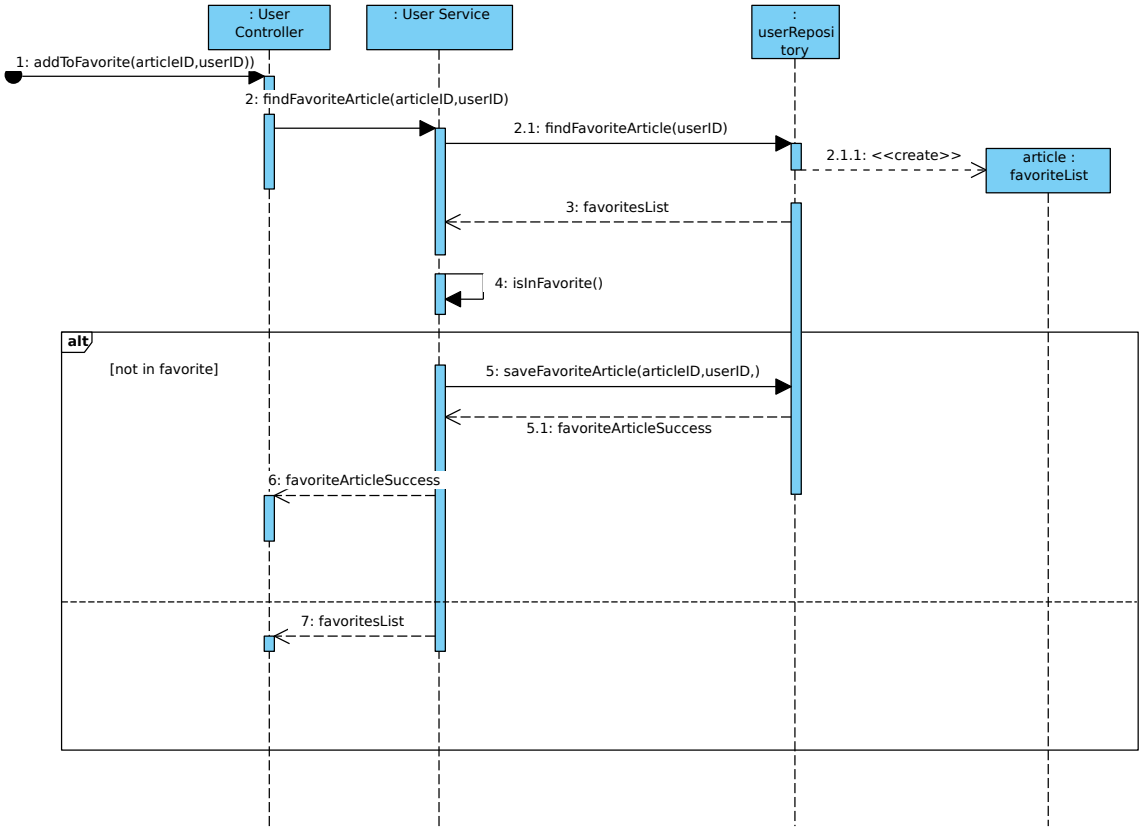
2.7.1 Use Case-11: Add to favorites

Use Case: Add to favorites
ID: BN0011
Actors: User
Preconditions: User must be logged on and in the view article page.
Flow of events: 1. User clicks on the add to favorites icon.
Postconditions: The System notifies with confirmation.
Alternative flow: User have the article previously added liked article.
Post Conditions: The system notifies with already was in the list.

2.7.2 System Sequence Diagram(SSD):Add to favorites



2.7.3 Sequence Diagram(SD):Add to favorites

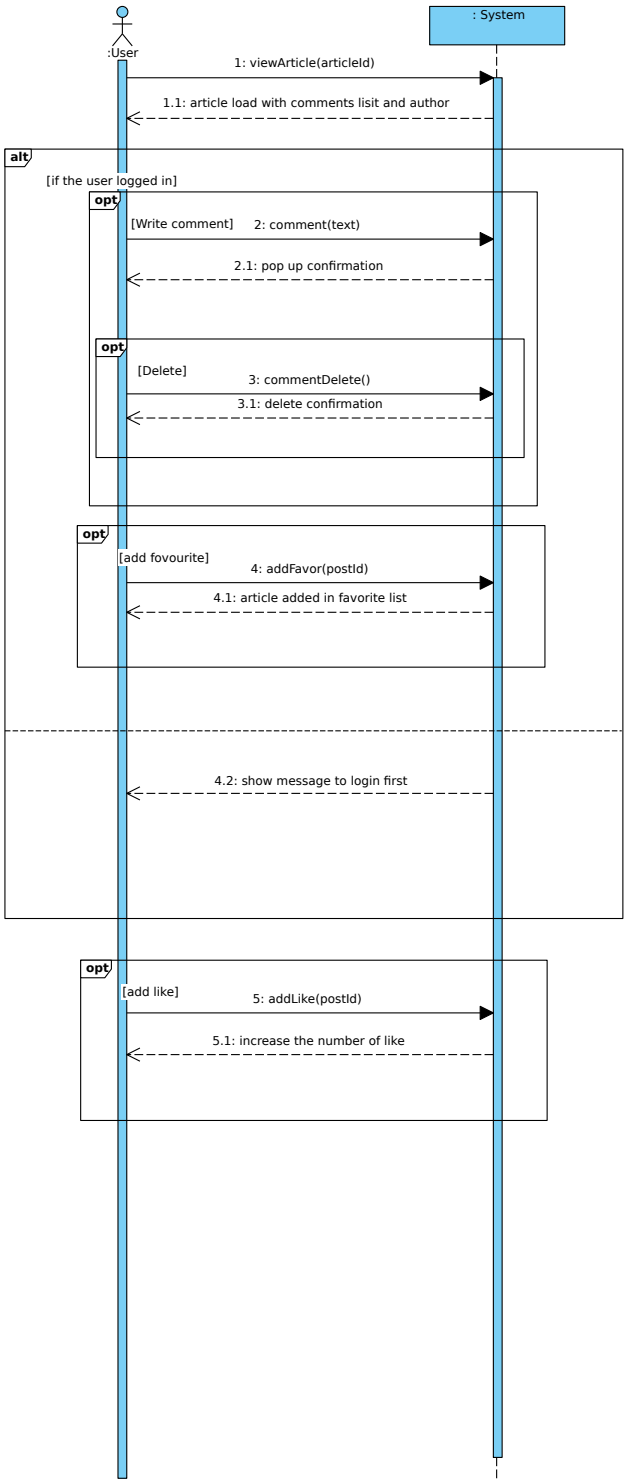


2.8 Show Article and Comments

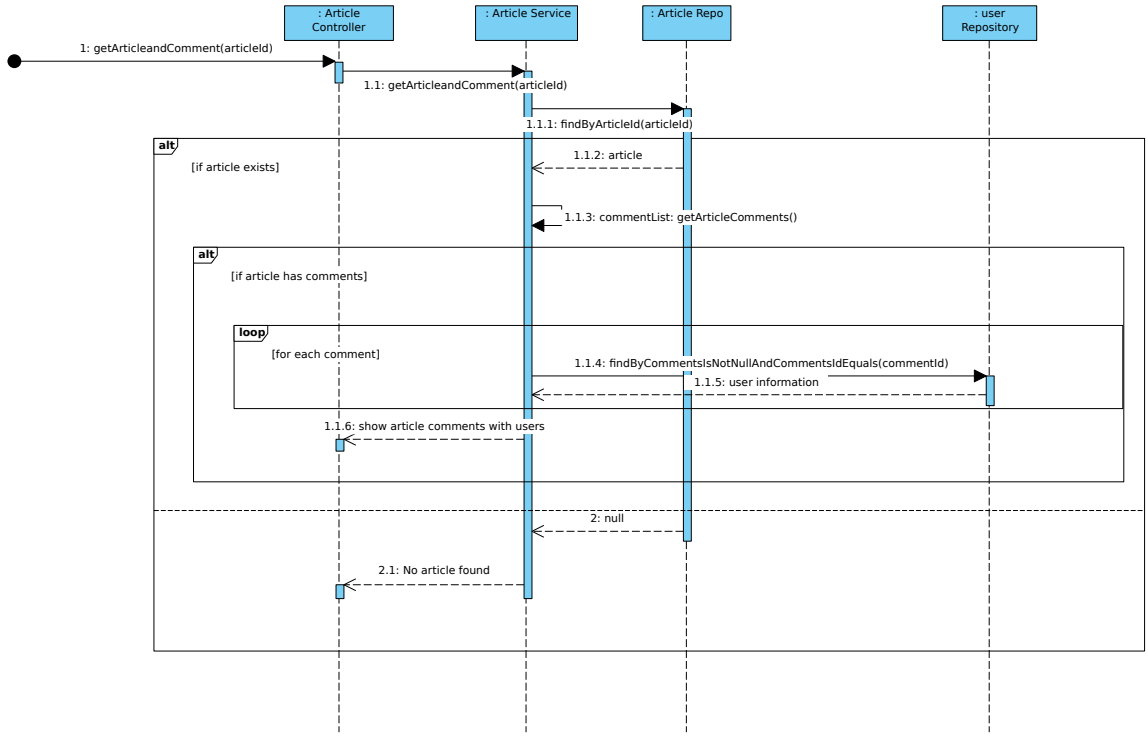
2.9 Use Case-3: Show Article and Comments

Use Case: Show Article and Comments
ID: BN003
Actors: User
Preconditions:
Flow of events: 1. The user clicks on the view full article button. 2. The system redirects loading the article with the list of comments and the author of comments.
Postconditions:
Alternative flow: If the user is logged in and writes a comment in a text box and clicks on the add comment button.
Post Conditions: The system will display a the list of comments in the article page.
Alternative flow: If the user clicks on the like button.
Post Conditions: The system will increase the like count of that post.
Alternative flow: If the user is logged in and click on the favorite icon
Post Conditions: the article will add to his/her favorite list.
Alternative flow: The user closes the article without doing anything.
Post Conditions: Redirects to the home feed.

2.9.1 System Sequence Diagram(SSD):Show Article and Comments



2.9.2 Sequence Diagram(SD):View Article Comment and Comment creator

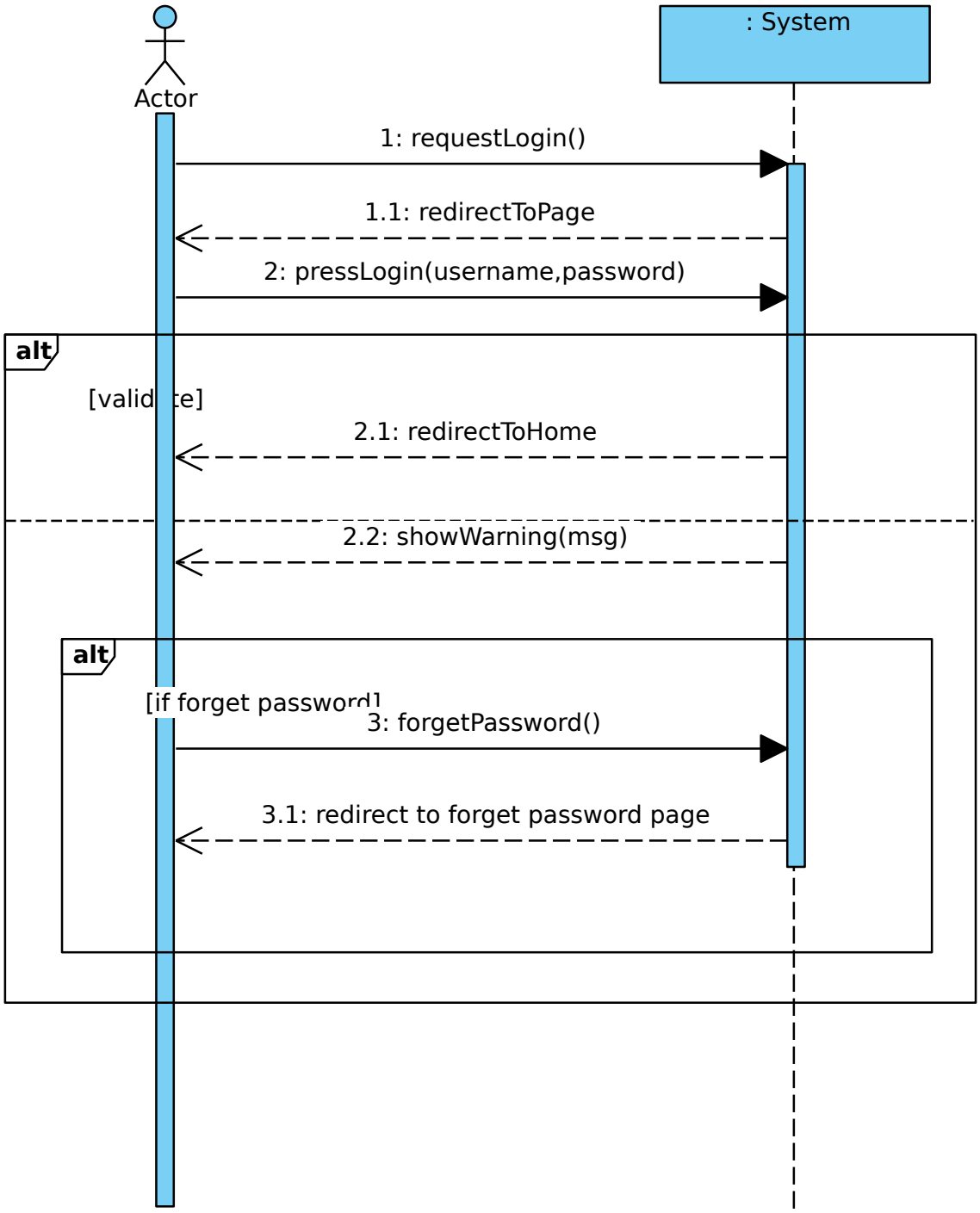


2.10 User Login

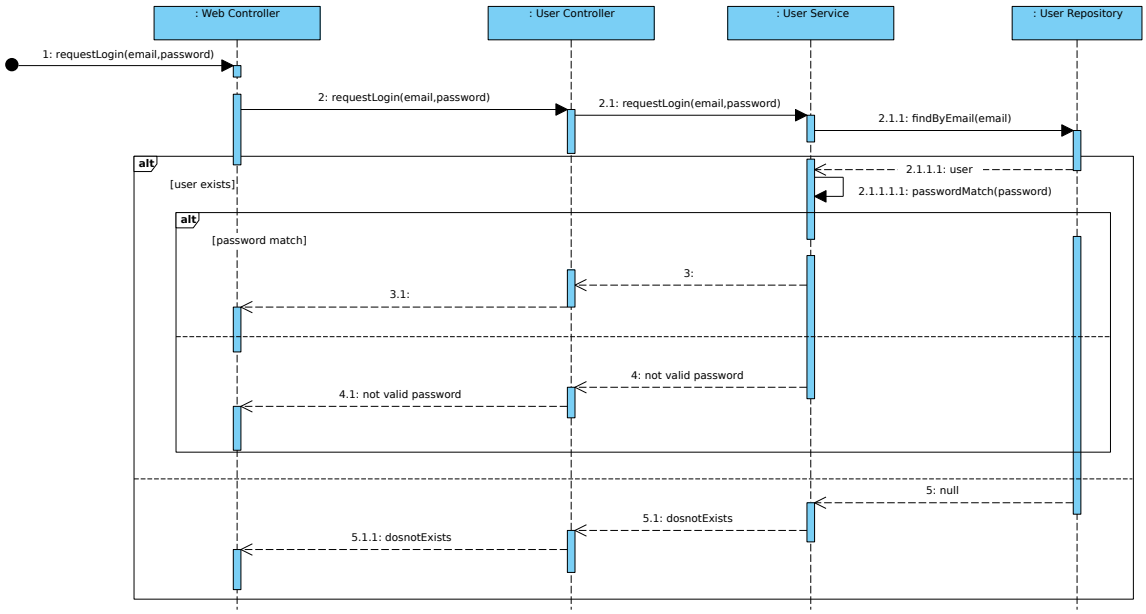
2.11 Use Case-6: User Login

Use Case: User Login
ID: BN006
Actors: User
Preconditions: The user goes to the login page.
Flow of events: 1. The user enters his/her name and password. 2. Press Login 3. If the system validates credentials. 4.1 The system redirects the user to the home feed. 4. Else the credentials are not validated the system will show a warning message.
Postconditions: The system shows the profile icon.
Alternative flow: The user interacts with the forgot password.
Post Conditions: The system will redirect to the reset password page.
Alternative flow: The user leaves the login page.
Post Conditions:

2.11.1 System Sequence Diagram(SSD):User Login



2.11.2 Sequence Diagram(SD):User Login

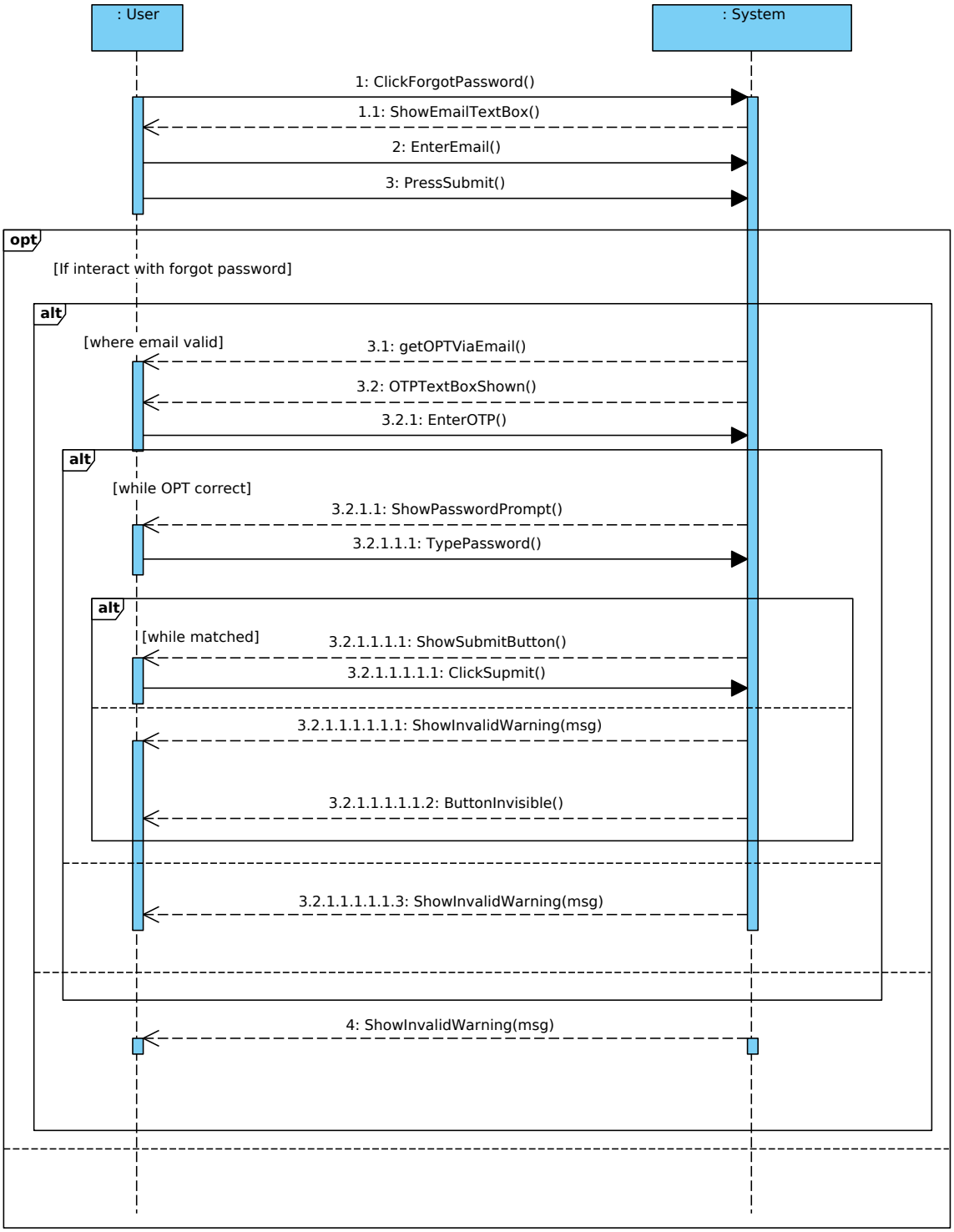


2.12 Forgot Credential(NOT DONE)

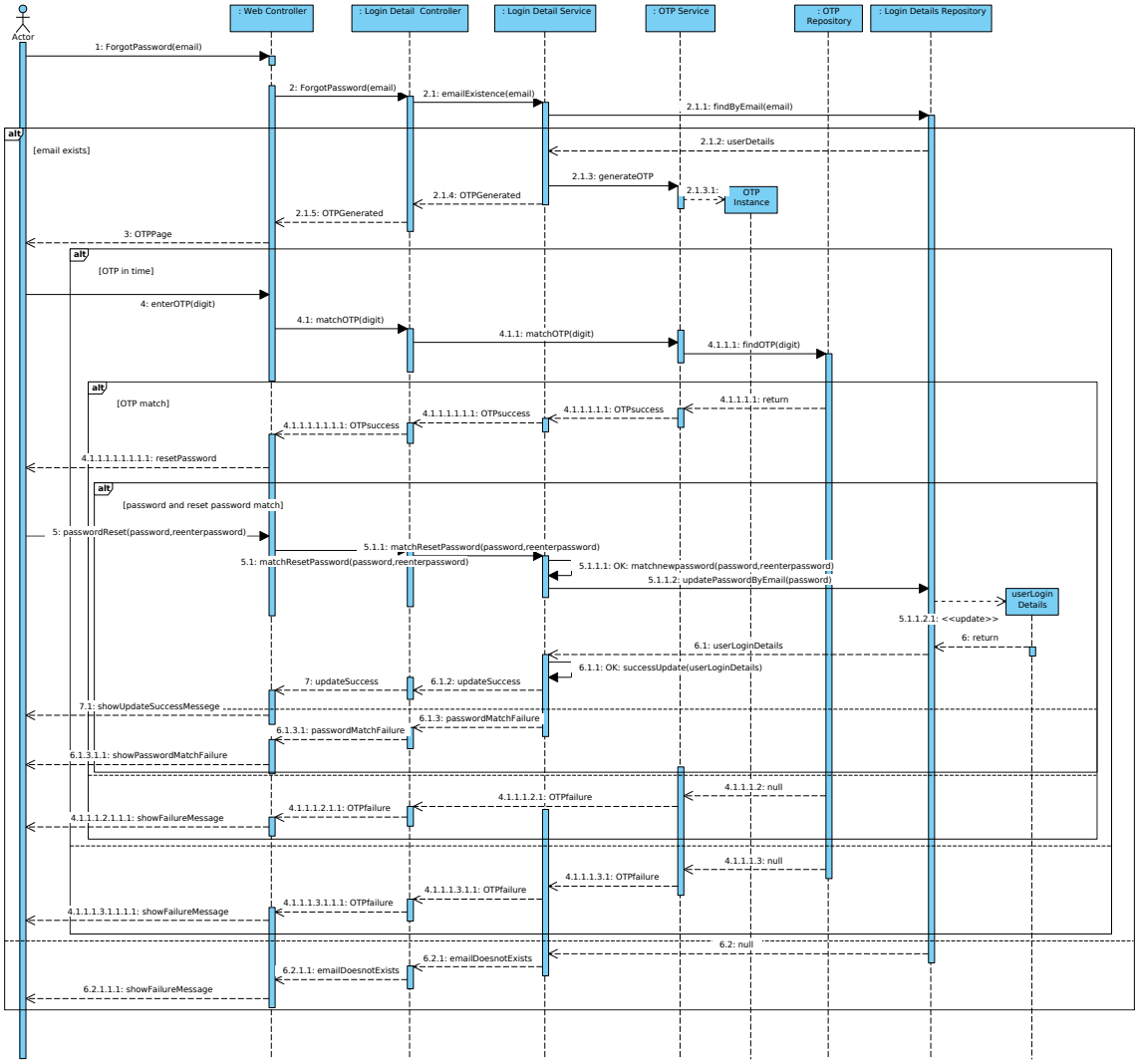
2.12.1 Use Case-7: Forgot Credential

Use Case: Forgot Credential
ID: BN007
Actors: User
Preconditions: If the user clicks forgot password icon
Flow of events: 1. The user enters his/her email id. 2. Press submit icon. 3. If the email id exists with the system. 3.1 The user will get his/her OTP in the email id. 3.2 Redirect to the reset password page 3.3 The entered OTP is correct and entered in time and passwords match.
Postconditions: The system redirects to the login page
Alternative flow: The OTPs mismatch.
Post Conditions: The system displays warnings OTP mismatched.
Alternative flow: The entered and reentered passwords don't matc.
Post Conditions: The system displays passwords mismatch.

2.12.2 System Sequence Diagram(SSD):Forgot Credential



2.12.3 Sequence Diagram(SD):Forgot Credential



2.13 User Profile(NOT DONE)

2.13.1 Use Case-13: Create Profile

Use Case: Register Profile
ID: BN013
Actors: User
Preconditions: User clicks on the signup page.
Flow of events: 1. System loads the register profile page. 2. User enters useremail, password, re enters password, first name, last name. 3. User clicks on the sign up button. 4. If the user email is valid 4.1 The system redirects to the enter otp page. 4.2 The otp is correct and entered in time.
Postconditions: The system will redirect to the home feed with profile icon.
Alternative flow: User can have invalid otps.
Post Conditions: System will redirect to the new homefeed with the profile icon visible.
Alternative flow: User can click on the login page.
Post Conditions: System redirects to the login page.

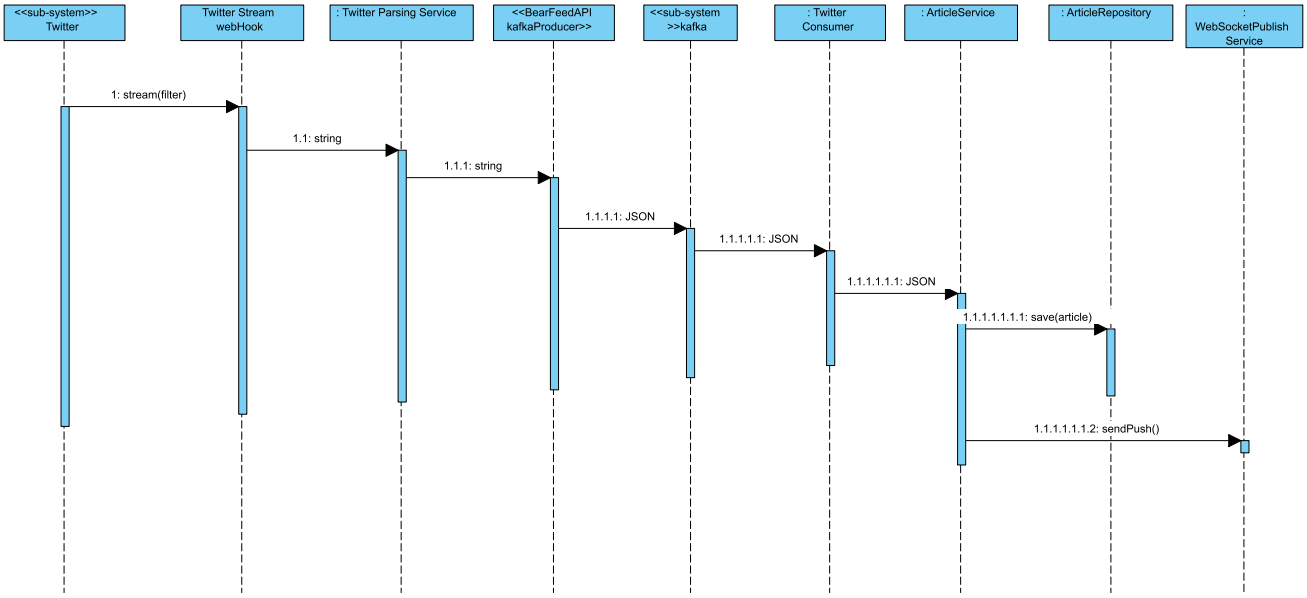
Use Case: User Interest List Update
ID: BN006
Actors: User
Preconditions: The user needs to logged into the system.
Flow of events: 1. The clicks on profile button. 2. User types for interest tags in the field 3. The system suggests tags. 4.1 The user selects tags from the suggestion. 4.2 The user clicks on submit button.
Post Conditions:

2.14 Twitter Crawler

2.14.1 Fully Dressed Use Case: Twitter Crawler

Use Case: Twitter Stream About BearFeed
ID: BN0013
Actors: Twitter Users
Preconditions: Twitter users post public posts about BearFeed.
Flow of events: 1. Twitter user post a public tweet with "bearfeed"/"BearFeed"/"BEARFEED" keywords. 2. TwitterStreamer picks all the streams with the above keywords. 3. Parse Twitter string message for hashtags and URLs. 4. Send the parsed JSON to Kafka producer. 5. BearFeed backend picks the stream asynchronously. 6. BearFeed backend process the stream message. 7. Beafeed backend pushes a new notification to the client webpage.
Postconditions: The event is created
Alternative flow: The user cancels without submitting.
Post Conditions: Redirects to the home feed.
Alternative flow: 2.1 User sets a date that is in the past.
Post Conditions: The system will display a warning message.

2.14.2 Sequence Diagram(SD):Twitter Crawler

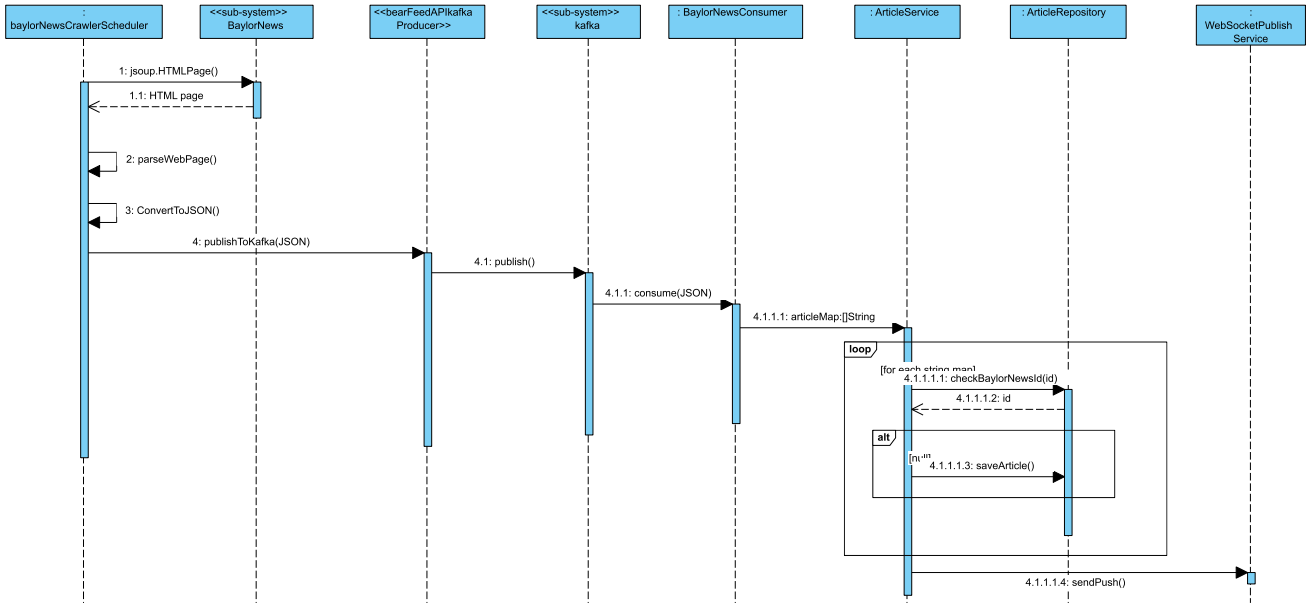


2.15 Baylor Crawler

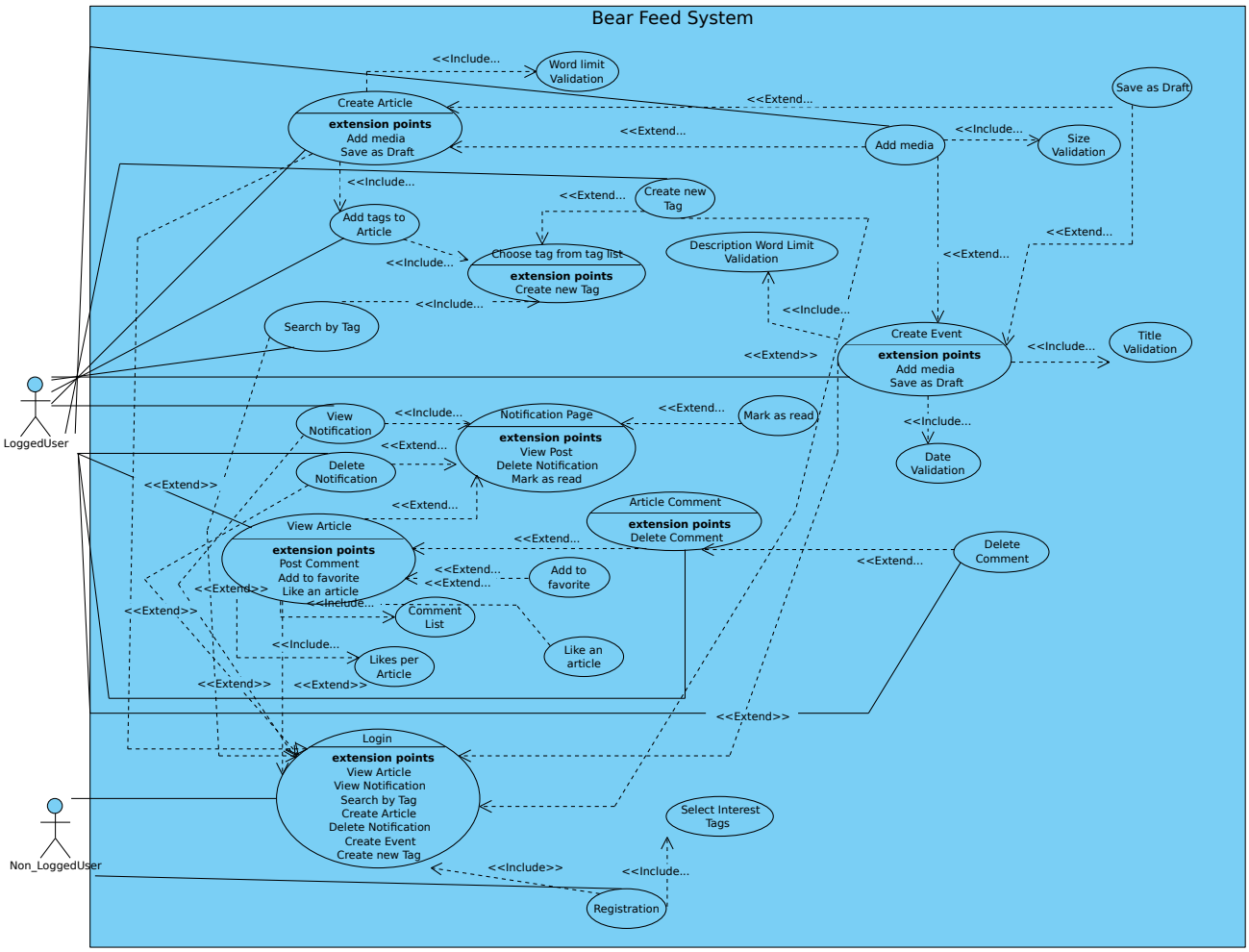
2.15.1 Fully Dressed Use Case: Baylor Crawler

Use Case: Baylor News Processing
ID: BN0013
Actors: BaylorNews Scheduler
Preconditions:
Flow of events: 1. BearNews scheduler parses www.baylor.edu/bn/ and www.baylor.edu/bn/news.php at 5:00 PM everyday. 2. BearNewsCrawler parses the webpage and turns the pages into parsable JSON. 3. Publishes the JSON to Kafka. 4. BearFeed backend picks the stream JSON message. 5. BearFeed backend checks the database for the existence of NewsId. 6. If NewsId is not found, then the backend service saves the JSON message as an article. 7. Send a push for a new article to the user client.
Postconditions: The event is created
Alternative flow: The user cancels without submitting.
Post Conditions: Redirects to the home feed.
Alternative flow: 2.1 User sets a date that is in the past.
Post Conditions: The system will display a warning message.

2.15.2 Sequence Diagram(SD):Baylor Crawler



3 Use Case Diagram



3.1 System Operation

3.2 GRASP in Domain Model

3.2.1 GRASP pattern: Information Expert

1. *User*- For Authenticate and Login Details, it is necessary to know about the user. It is necessary to know about the user for creating Event. User is responsible to create a Tag.
2. *Twitter Crawler*- Twitter Crawler have a message subscriber which will listen to notification from twitter. It will validate the new post with predefined logic and if everything satisfied it wil send the post to Article service for saving it to Database.
3. *Baylor News Crawler*- Baylor News Crawler has periodic call to Baylor news portal and it will face any new article. Baylor news crawler have a record of last fetched article from the news portal to commit duplicate article save. Article save is done asynchronously via message queue.

3.2.2 GRASP pattern: Partial Expert

1. *Article*- To create Article, we need to assign Tag and need to know about User. To fulfill the new responsibilities require information that is spread across these classes of objects Tag and User.
2. *Favorite List*- To modify the Favorite list, two responsibilities will be assigned, User and Article. To fulfill the new responsibilities require information that is spread across these classes of objects Article and User.
3. *Comment*- To create Comment, two responsibilities will be assigned, User and Article. To fulfill the new responsibilities require information that is spread across these classes of objects Article and User.
4. *Notification*- To create Comment, three responsibilities will be assigned, User and Article and Event. To fulfill the new responsibilities require information that is spread across these classes of objects Article and User and Event.

3.2.3 GRASP pattern: Creator

1. *Article creates Article*- Whenever a User creates an Article, the system will create Article instance and attaches specific tag from Tag class. Therefore, Article class is responsible for creating an article.
2. *Tag creates Tag*- Whenever a User creates a Tag, the system will create Tag instance. Therefore, Tag class is responsible for creating an tag.
3. *Event creates Event* Whenever a User creates an Event, the system will create Event instance. Therefore, Event class is responsible for creating an article.
4. *Comment creates Comment* Whenever a User creates a Comment, the system will create a Comment instance. Therefore, the Comment class is responsible for creating a comment.
5. *Media creates Media* Whenever a User uploads Media, the system will create Media instance. Therefore, Media class is responsible for creating a media.

3.2.4 GRASP pattern: Low Coupling

1. Low coupling exists between Login Detail and User via intermediate table. Because Login Detail will keep some attributes of User.

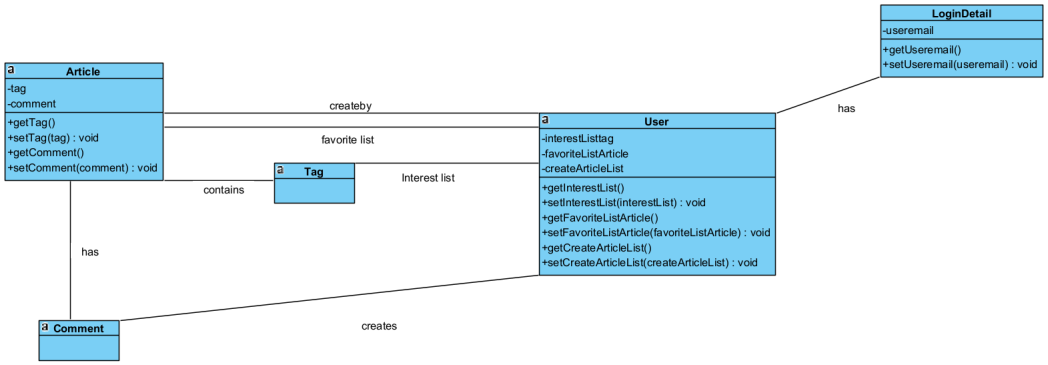


Figure 1: GRASP Pattern Creator, Information Expert, Partial Information Expert, Low Coupling

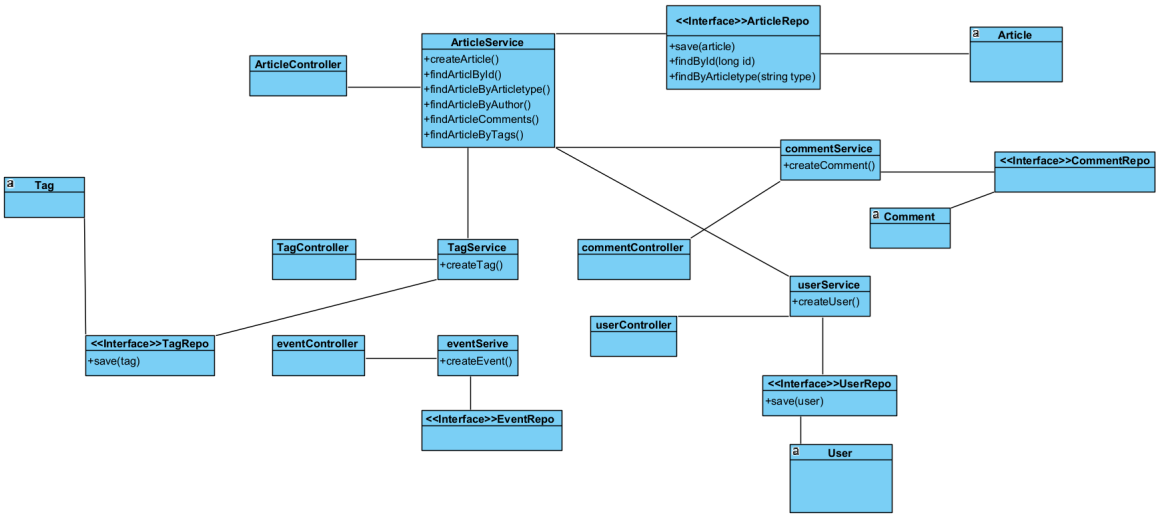


Figure 2: GRASP Pattern High Cohesion, Low Coupling, Pure Fabrication, Controller

2. Low coupling exists between Crawlers and Article via intermediate table
3. Low coupling exists between Article and Comment via intermediate table. Because Comment will keep some attributes of Article.
4. Low coupling exists between Article and Tag via intermediate table.

3.2.5 GRASP pattern: High Cohesion

1. User has high cohesion because a user collaborates with other classes to fulfill the responsibilities.
2. Article has high cohesion as it has the responsibility to create the Article and Add to Favorite.
3. Event has high cohesion as it has the responsibility to create Event

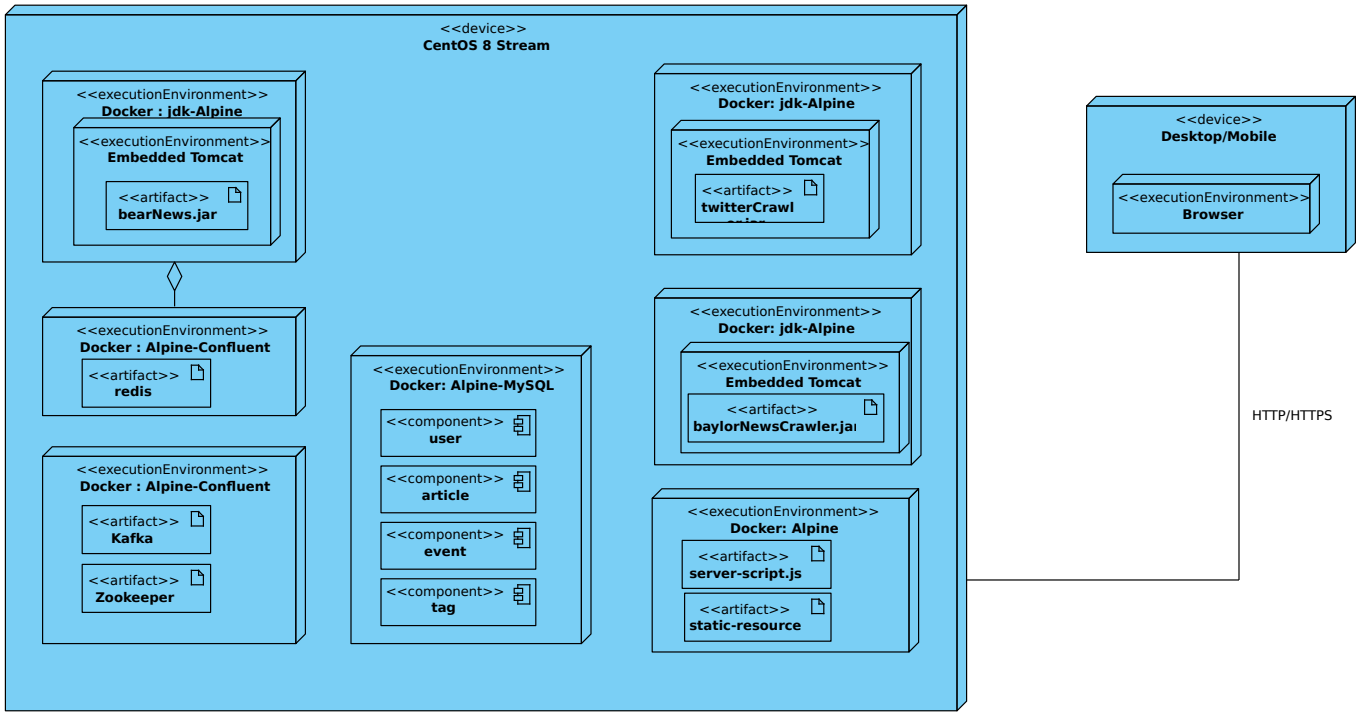
3.2.6 GRASP pattern: Controller

1. *Articles Controller*- It handles a view routes request to Article entity to create article. Gives us the full view of the full article. It redirects us to a view which has the full contents of the article and all comments related.
2. *Event Controller* It handles a view routes request to Event entity to create event.
3. *Login Controller* It will handle the user's authentication process. It will handle the request to reset use password.
4. *Search Controller* It will handle the user tag search and return a view with all specific articles.
5. *User Controller* It will handle the request of updating profile from user.
6. *Tag Controller* It handles the creating tag.

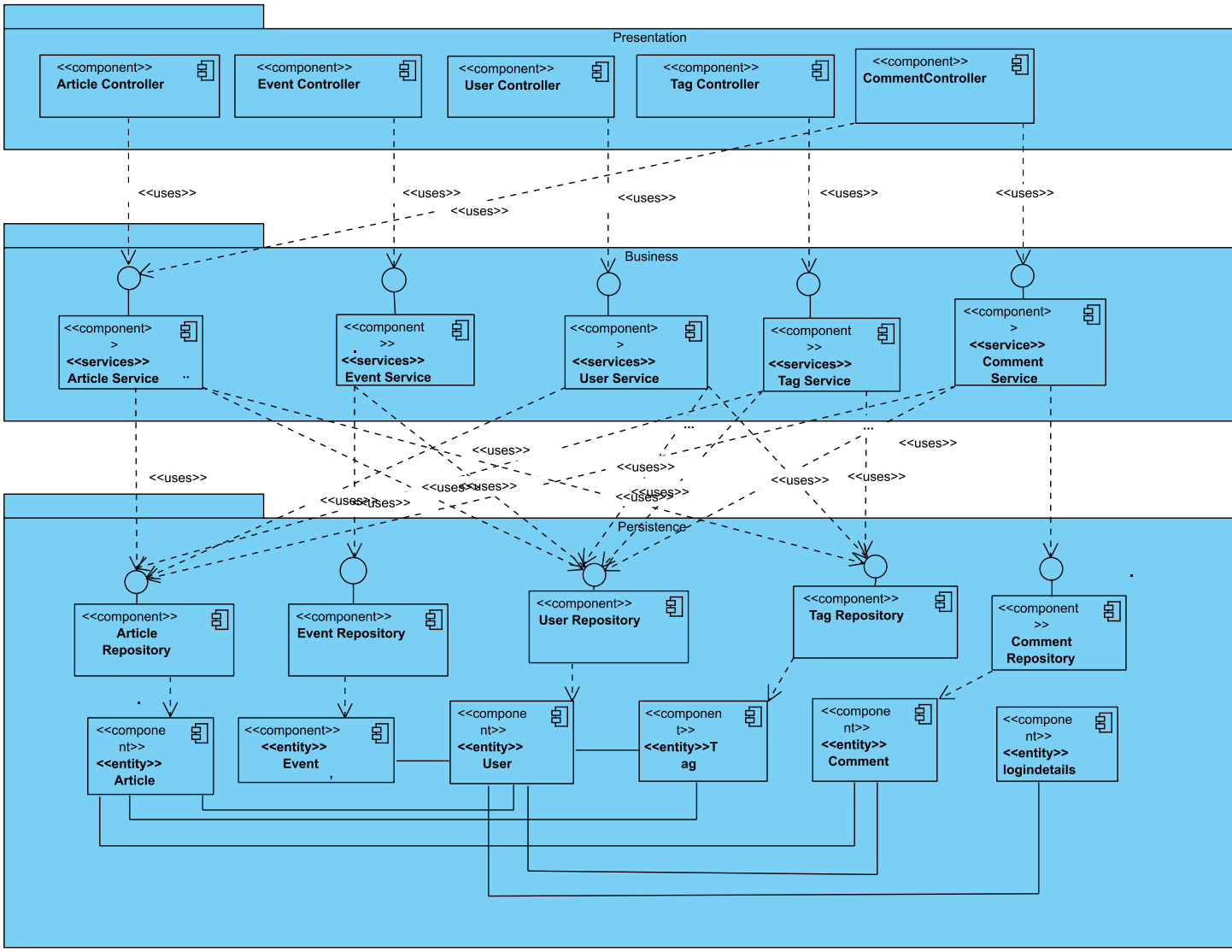
3.2.7 GRASP pattern: Pure Fabrication

In our whole system, we need to persist with the database. For the persistent and fetching data, we access it through interface. so our classes such as user, article, event is coupled with database interfaces.

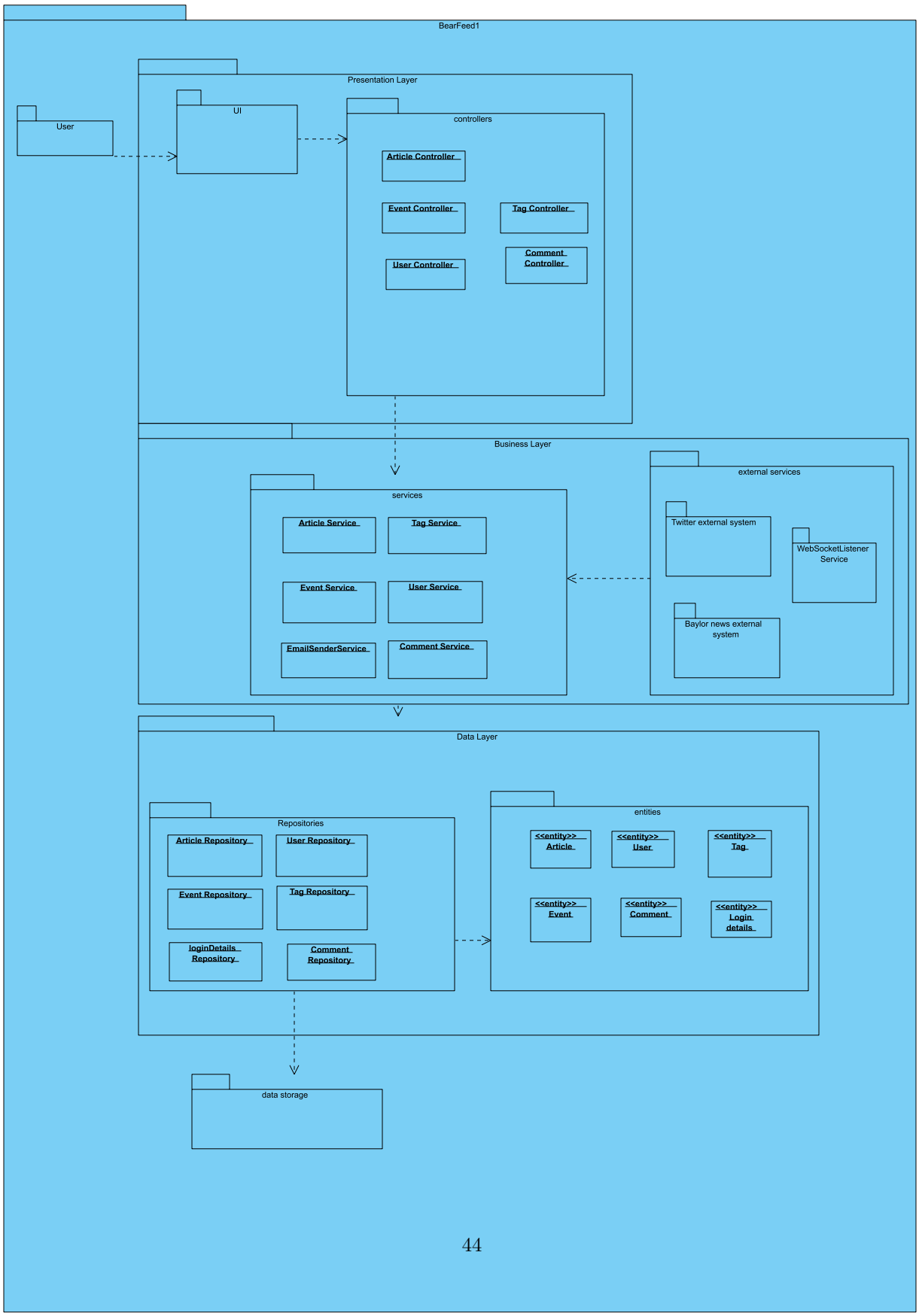
3.3 Deployment Diagram:



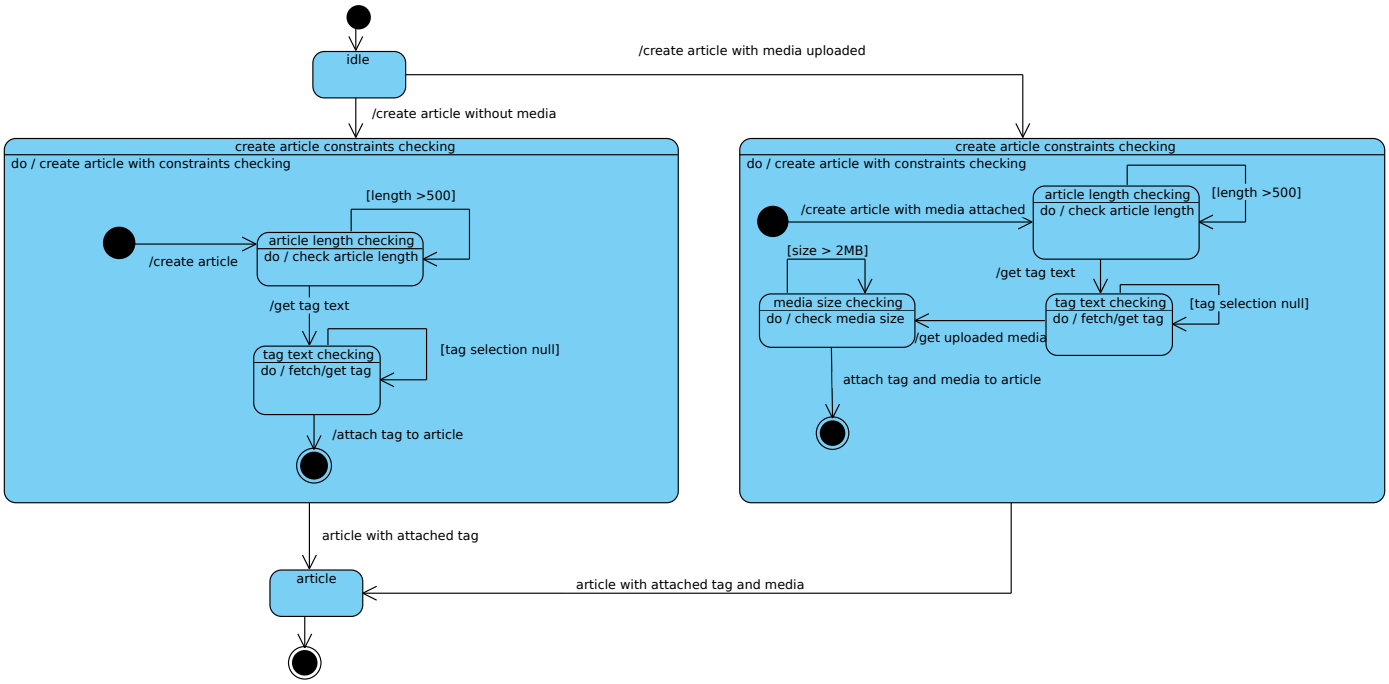
3.4 Component Diagram:



3.5 Package Diagram:



3.6 State Machine Diagram For add Media to Post:



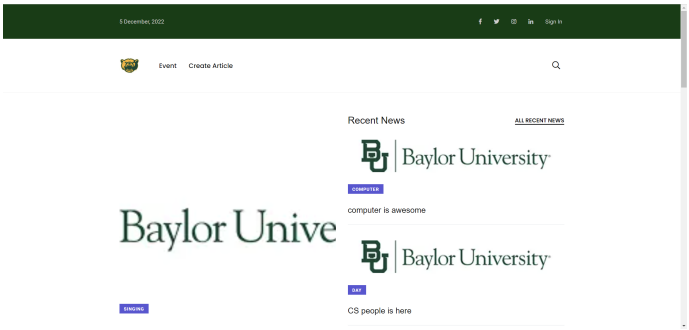


Figure 3: Main page with articles created by user

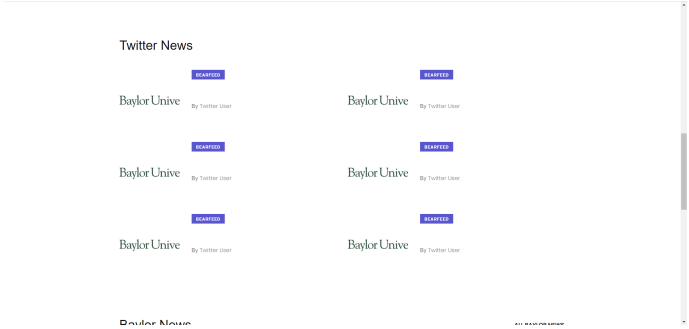


Figure 4: Main page with articles from Twitter

4 User Interface Overview

Here is the full demo of our Project:

4.1 All the UI Images

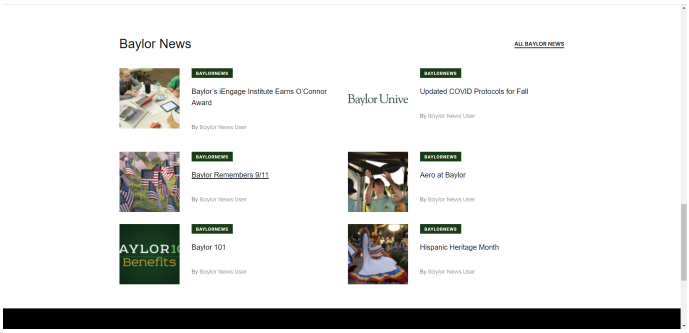


Figure 5: Main page with articles from Baylor News

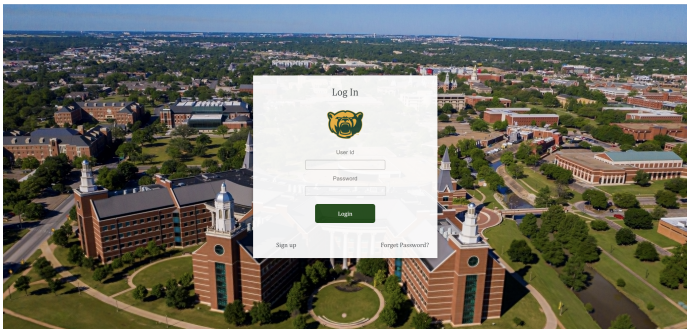


Figure 6: Sign in page

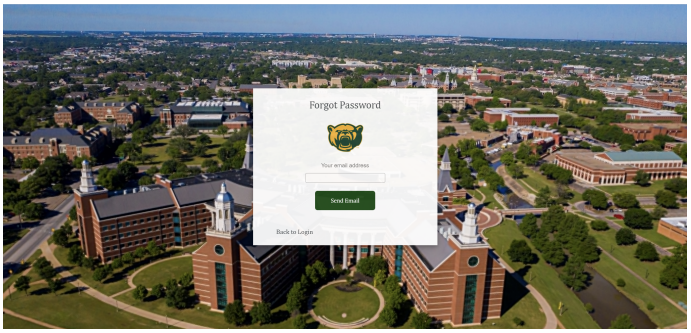


Figure 7: Forgot password page

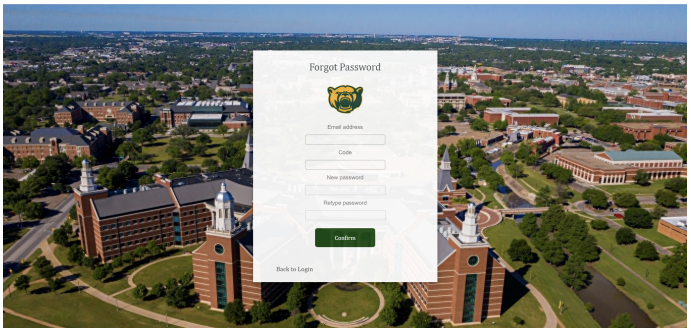


Figure 8: Reset password page

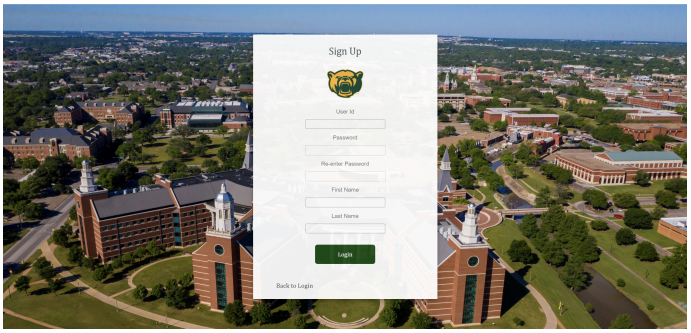


Figure 9: Sign up page

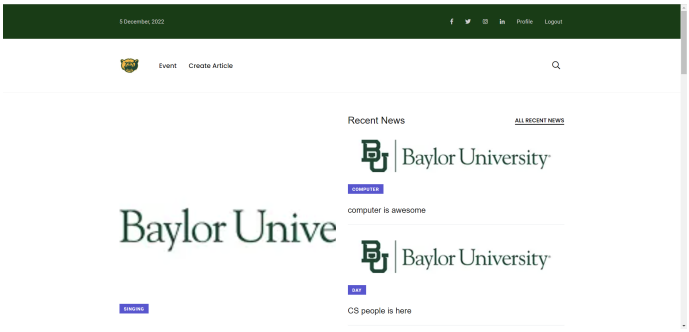


Figure 10: Logged in page

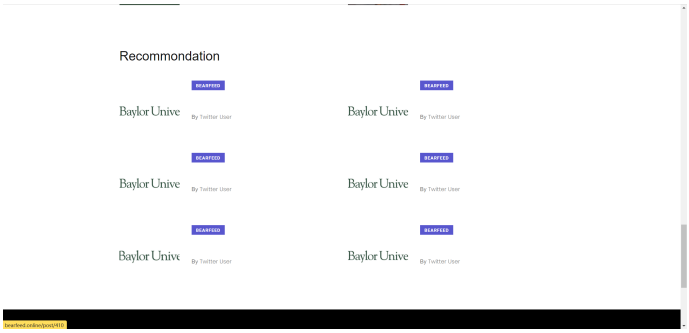


Figure 11: Recommendation articles showed by logged-in user's interest tags

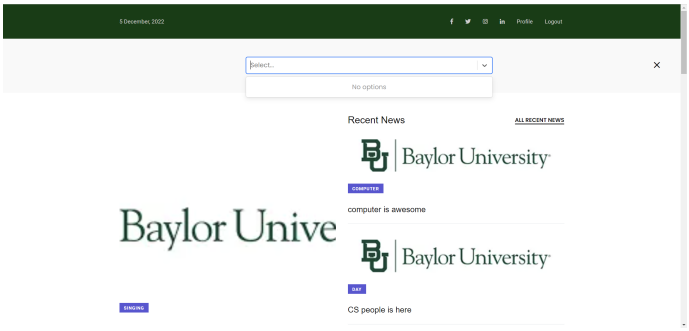


Figure 12: Search section

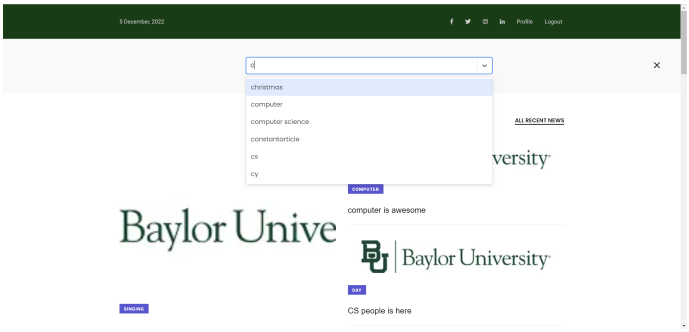


Figure 13: Search recommendation

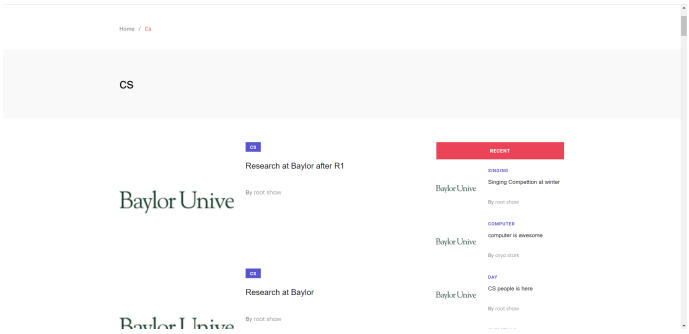


Figure 14: Search result page

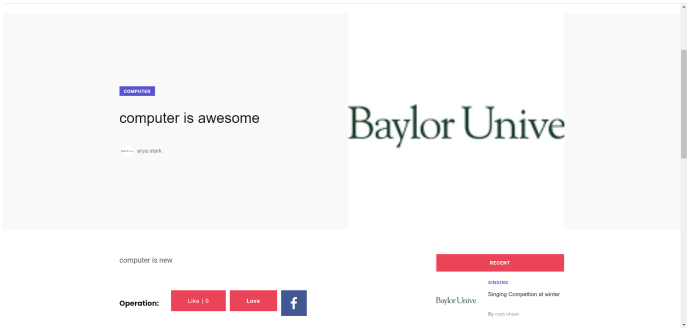


Figure 15: User created article's detail page

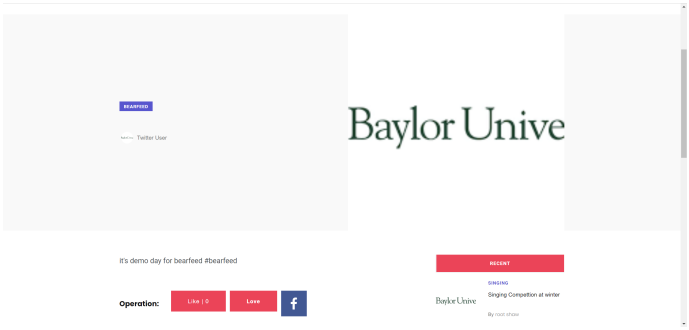


Figure 16: Twitter article detail page

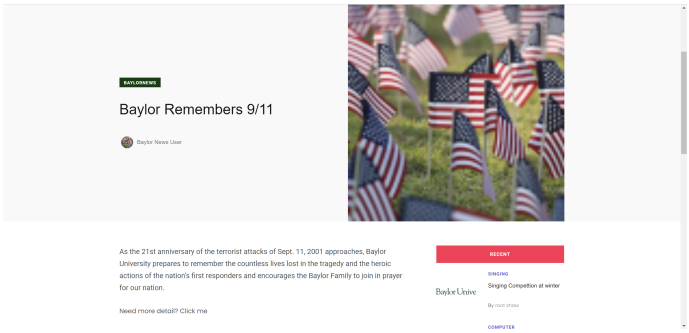


Figure 17: Baylor News article detail page

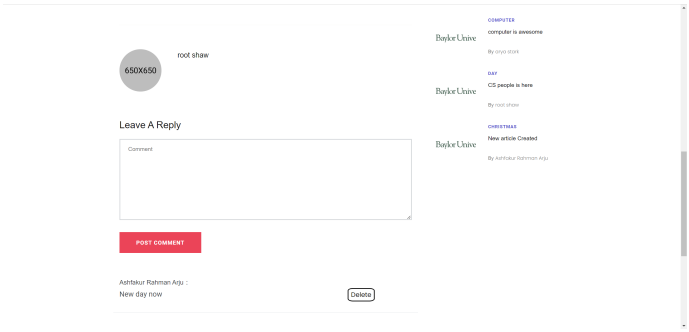


Figure 18: Comment Section

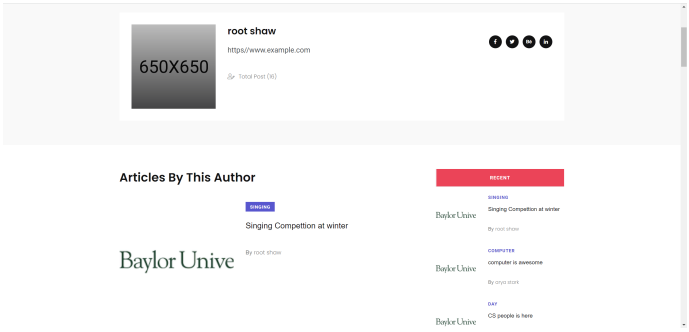


Figure 19: One specific author's article page

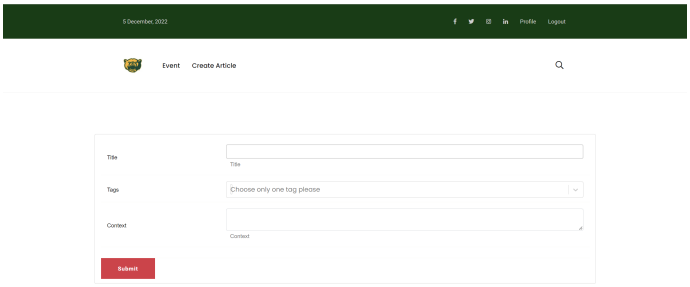


Figure 20: Create article page

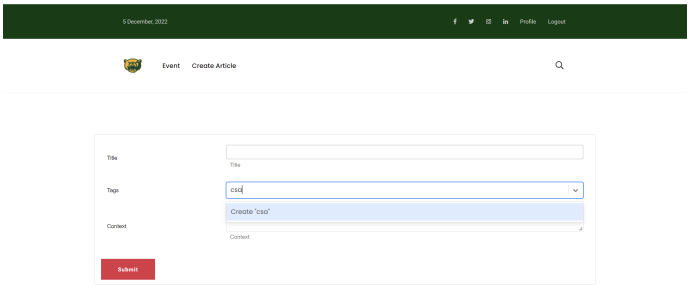


Figure 21: Create tag section

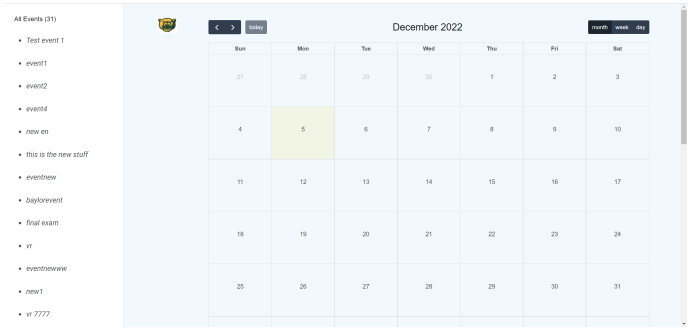


Figure 22: Event page

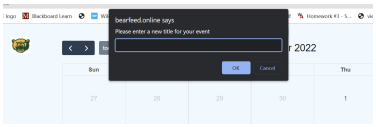


Figure 23: Create event section

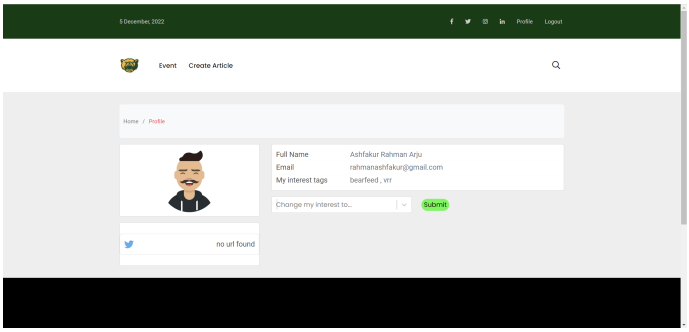


Figure 24: Profile page

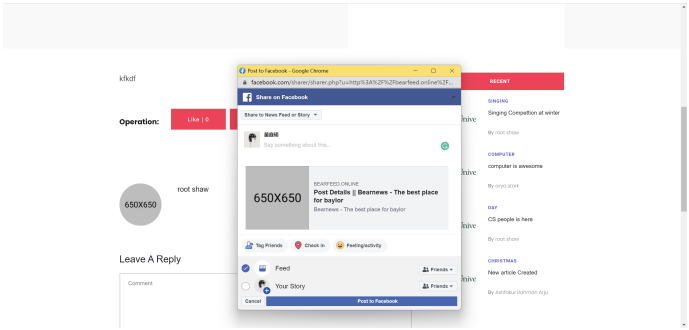


Figure 25: Share to social media section

5 Implementation

Our backend server is developed using Spring Boot. The core maven dependencies are:

5.1 Backend

1. Spring Boot Web Starter
2. Spring Boot Security Starter
3. Spring Boot ActiveMQ Starter
4. Spring Boot Stomp-websocket
5. MySQL
6. Lombok
7. Spring-boot-starter-test
8. Spring Boot Starter Kafka
9. Jedis

5.2 Frontend

Our frontend client is developed using Next JS. Next JS was chosen because of its very efficient async rest handling and DOM manipulation. Also, NextJs helps with the routing of sub-DOM very easily. Frontend communicates with the Backend server using REST API. The Frontend has Runtime dependencies and Development dependencies. For Development, we use the following tools:

1. eslint
2. eslint-config-next

For Runtime, we use the following dependencies

1. babel/preset-react
2. emailjs/browser
3. fortawesome/react-fontawesome
4. fullcalendar/common
5. fullcalendar/interaction
6. fullcalendar/react
7. fullcalendar/timegrid
8. stomp/stompjs
9. axios

10. bootstrap,
11. gray-matter
12. luxon
13. mdb-react-ui-kit
14. moment
15. next
16. next-transpile-modules
17. ng-fullcalendar
18. prop-types
19. react
20. react-animated-heart
21. react-bootstrap
22. react-cookie
23. react-dom
24. react-easy-button
25. react-notifications
26. react-responsive-masonry
27. react-router-dom,
28. react-select
29. react-share
30. react-slick
31. react-stomp
32. remark
33. remark-html
34. sharp
35. sockjs

5.3 Backend Authentication and Authorization

As our system has only two user types, we don't have any role-based authentication. We are implementing spring-security JWT token-based security. When the user type the correct credential, the backend provides a JWT token to the user and the browser caches the token for a time being, 30 min. Whenever there is a request to the API backend, the JWT token is picked from the cache.

5.4 Frontend Authentication and Authorization

In our system, a user can see the generalized feed but can't make any comments or like the feed. If the user is logged in, they can comment, add to favorites and delete comments.

5.5 REST API

In our application API backend, we have used spring-boot-web rest controller for exposing the endpoints. In our whole system, we have three subsystems. In total, we have more than 40 endpoints. In our application API backend, we have used spring-boot-web rest controller for exposing the endpoints. In our whole system, we have three subsystems. In total, we have more than 40 endpoints.

5.6 Email Verification

The user must confirm email upon registration. Following registration, a confirmation code will be sent to the user's email. The user account will stay disabled until the email is validated. To send emails to user-specified emails, we used the Zoho Mail client.

5.7 Reset Password

The forgot password option allows the user to reset his account's password. He must enter his registered email. He will receive a confirmation number via email, which he can use to create a new password. To produce and email confirmation codes, we used the same technique as during registration.

5.8 Websocket Implementation

We have implemented WebSocket with STOMPJS and spring-boot-amq. When the user is logged in, front end client establishes a connection with the backend with its username. Whenever there is a new article say from a system user, twitter or Baylor news the user will receive a push notification to its UI. Upon receiving a new push, the feed will be refreshed automatically.

5.9 Kafka Implementation

In our current system, we have three subsystems: API backend, TwitterStreamer, and BaylorNewsCrawler. Both TwitterStreamer and BaylorNewsCrawler use Kafka to publish a new post. When there is a new message in the topic, the API backend picks the message and processes it as an article. When the article is processed, Backend sends a push notification using the WebSocket

5.10 Design Patterns

1. **Singleton:**
   We have some seed data to function like the default tag, default image, and URL. We are using @EventListener to make sure that those are instantiated only once.
2. **Builder**

   (a) Kafka config builder
   (b) Redis config builder
   (c) Fasterxml-Jackson builder for parsing date and custom fields.
3. **Observer:** We have implemented WebSockets and Kafka which follow the Observer pattern.

6 Deployment

We have used DigitalOcen droplets for our host deployment environment. Our host machine has the following configuration :

1. OS: CentOS 8 Stream
2. RAM: 8GB
3. CPU: 4 Physical Core
4. Disk: 160 GB
5. Rate Limiting Bandwidth 4MBps

In the host machine, we have two types of execution environments. CentOS screen and Docker. The following services are running in the CentOS screen environment :

1. Nginx
2. BearFed API Backend
3. Bearnews Crawler
4. Twitter Streamer

The following services are running on the Docker host:

1. Kafka-Zookeeper (with local volume)
2. Redis (with local volume, on-disk persistence)
3. MySql (with local volume, periodic backup to disk)

We have also configured Nginx with our public domain (www.bearfeed.online). For nameserver resolution, we are using DigitalOcean and Hostinger Nameservers. Nginx is currently configured to reverse proxy all the request to localhost:3000, where our front-end react application is running in the development build.

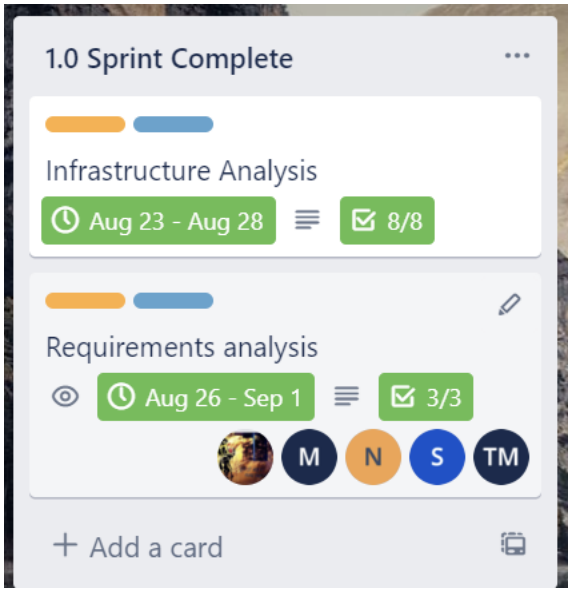


Figure 26: Spring 1 task

7 Trello and Gantt chart

7.1 Trello

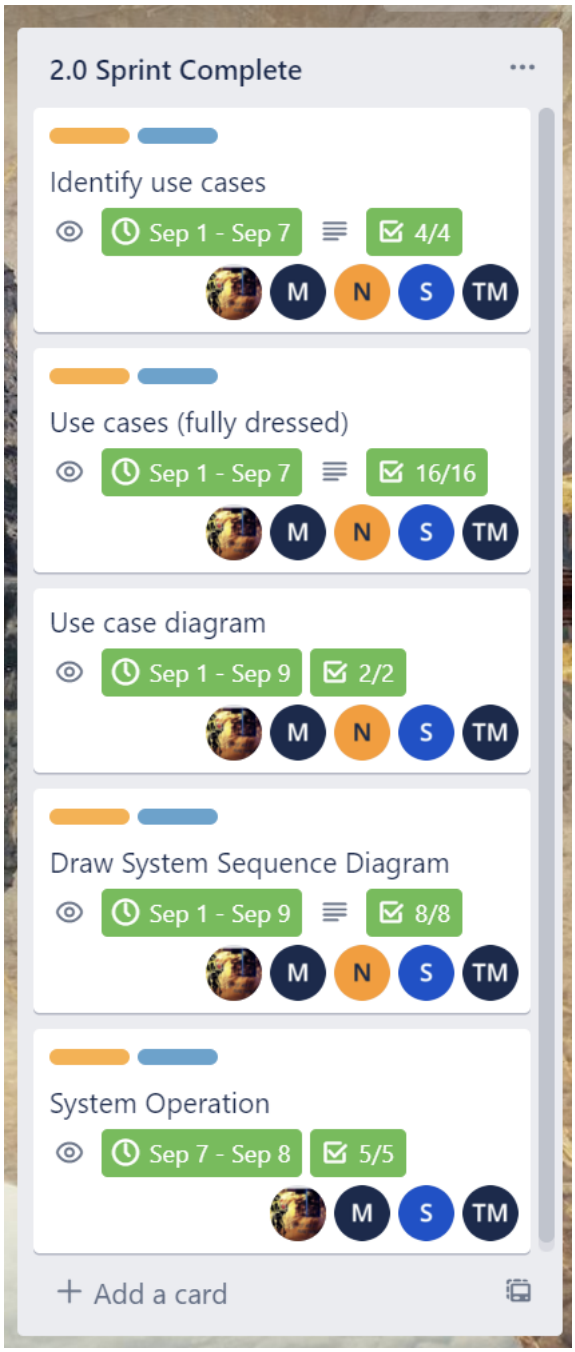


Figure 27: Spring 2 task

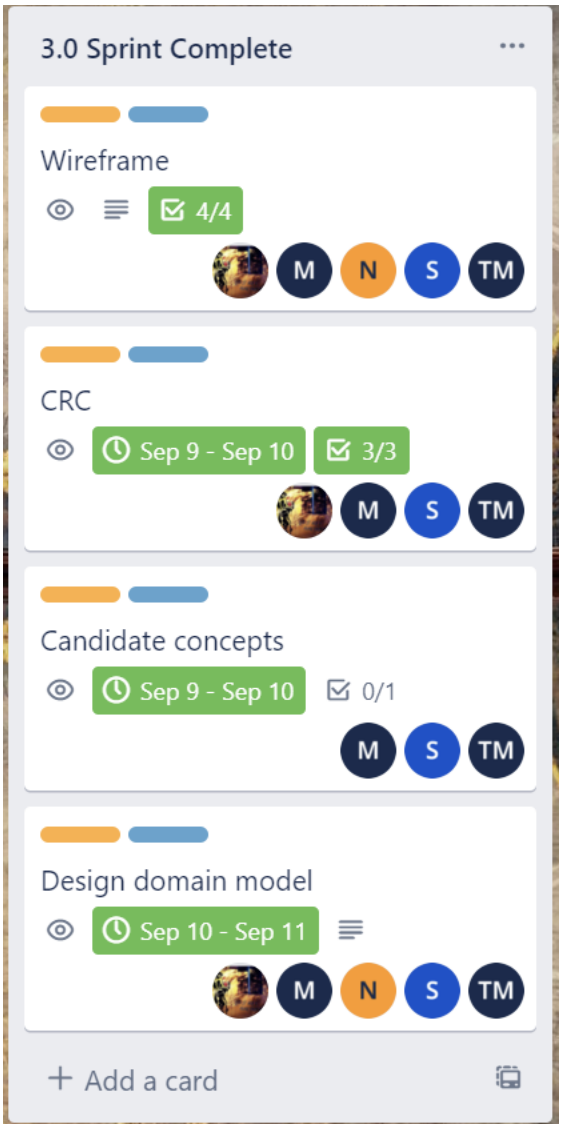


Figure 28: Spring 3 task

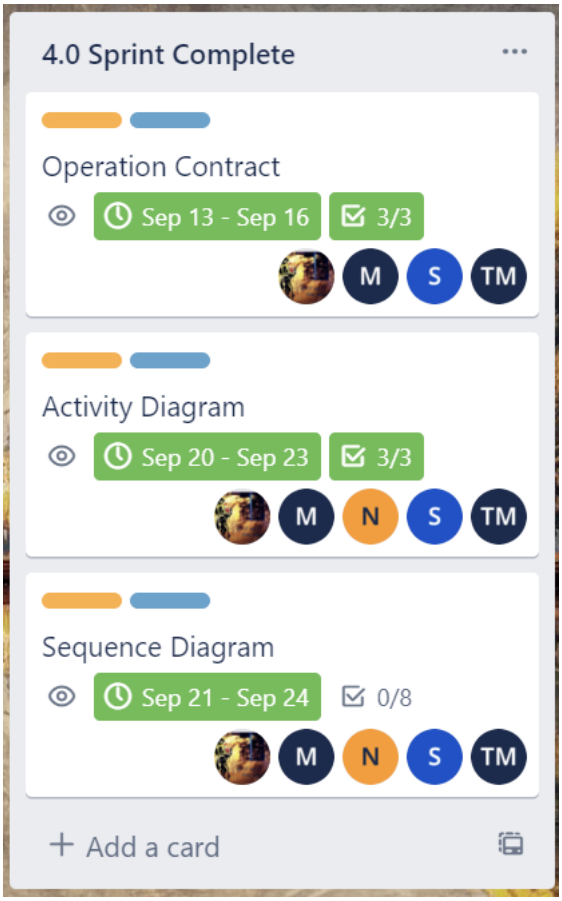


Figure 29: Spring 4 task

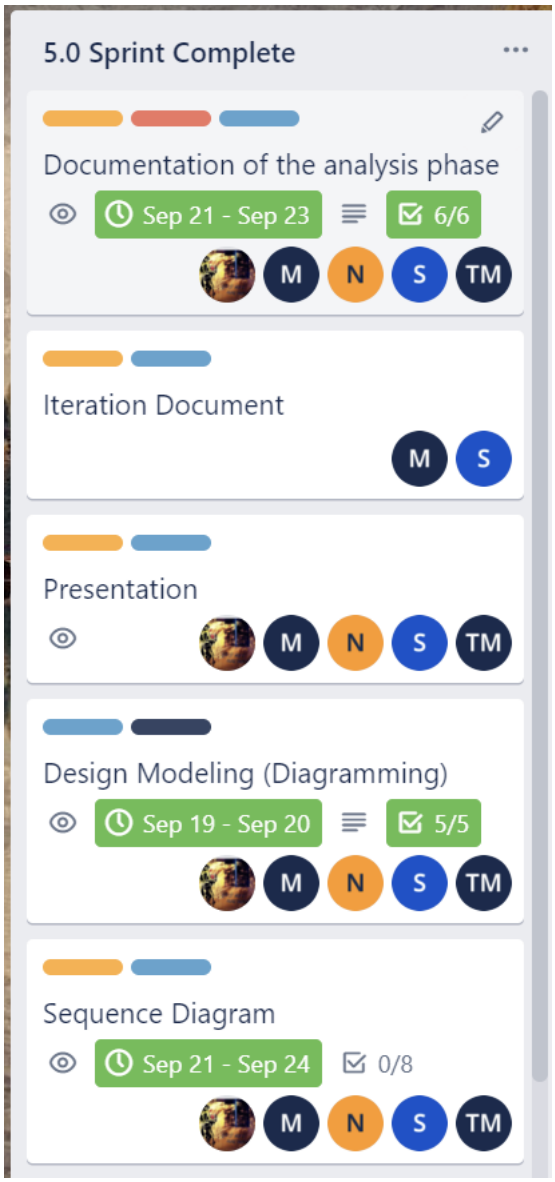


Figure 30: Spring 5 task

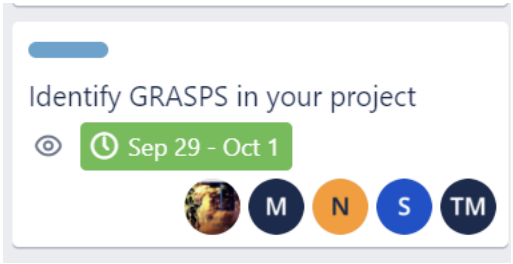


Figure 31: Spring 5 task cont.

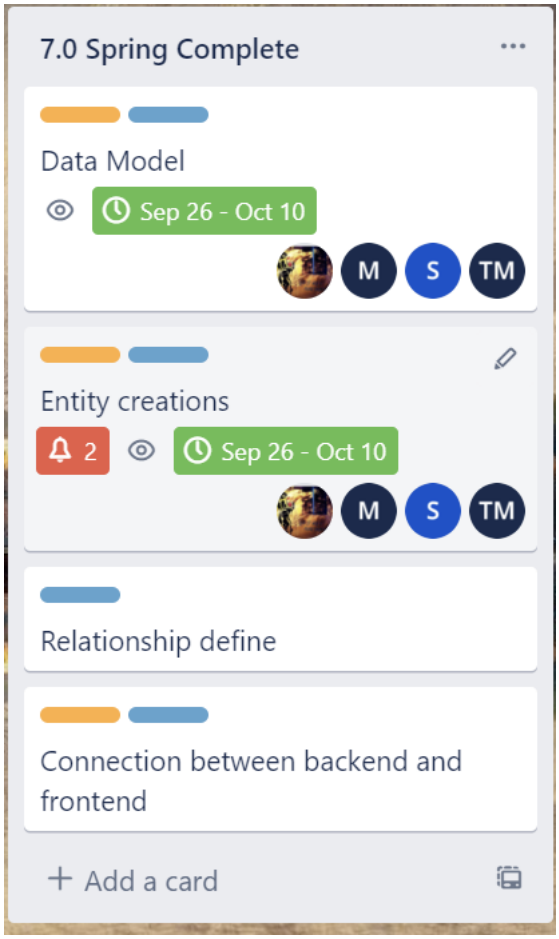


Figure 32: Spring 7 task

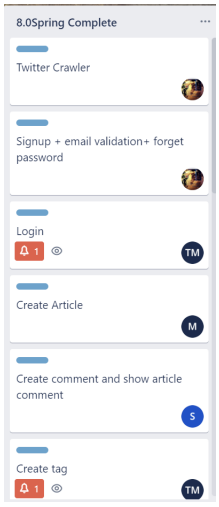


Figure 33: Spring 8 task

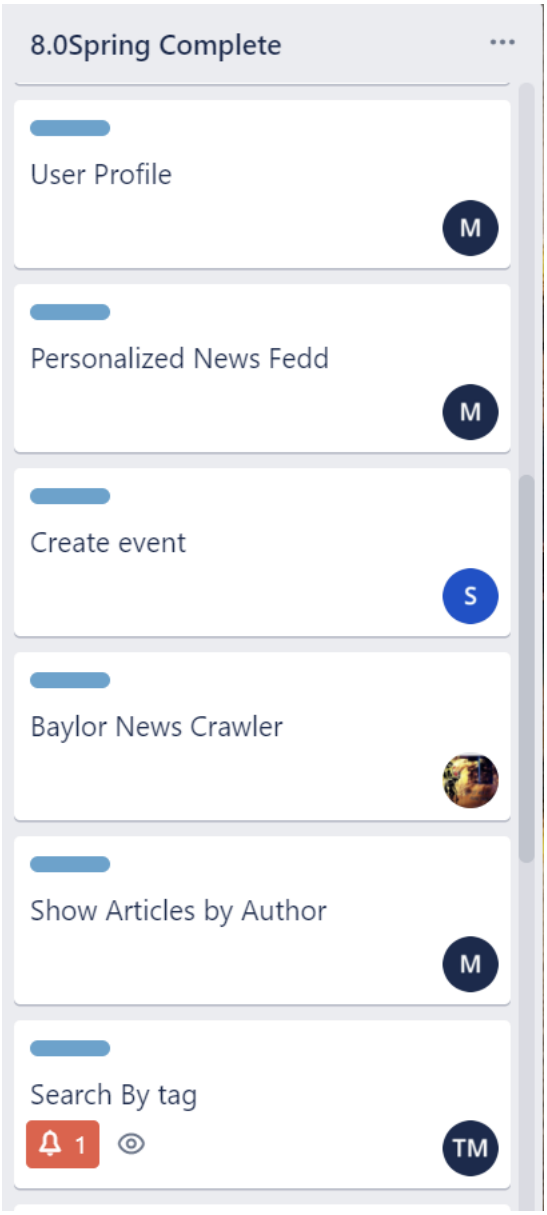


Figure 34: Spring 8 task cont.

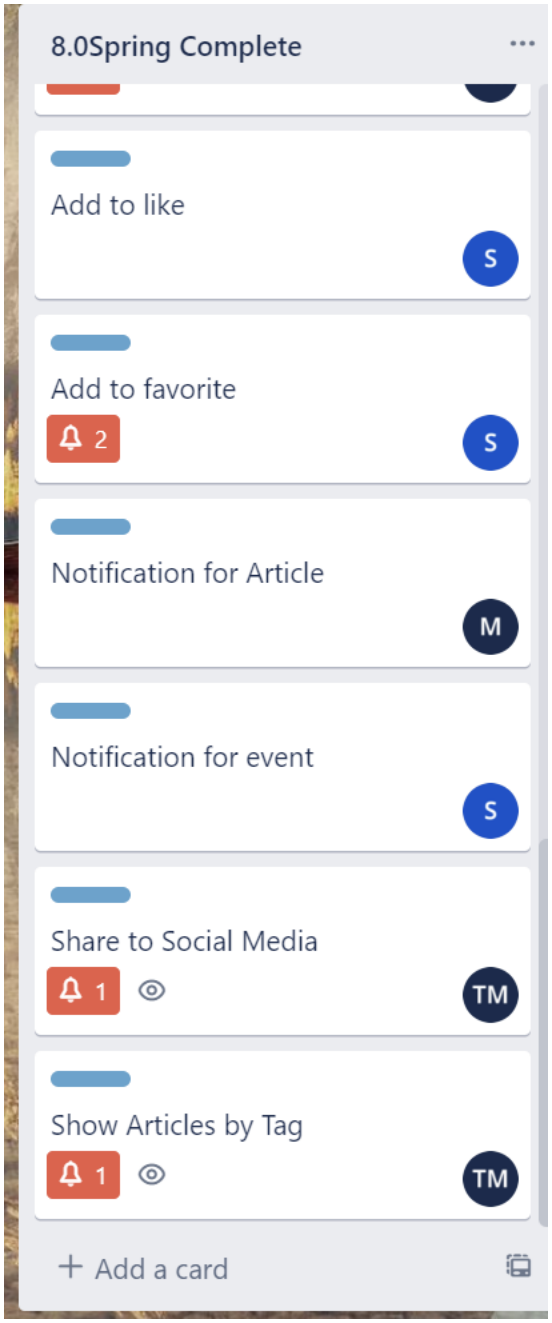


Figure 35: Spring 8 task cont.

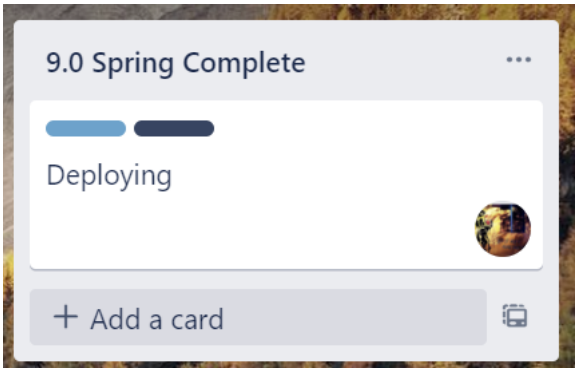
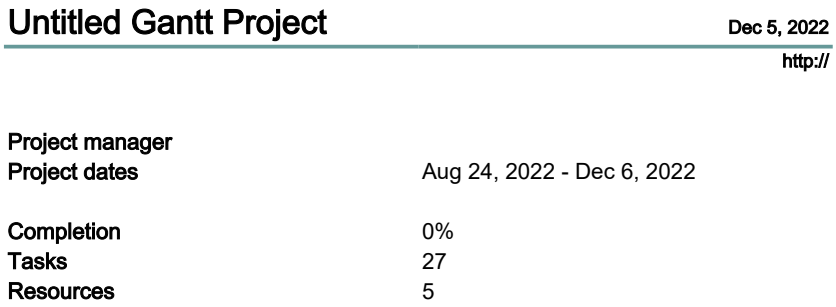


Figure 36: Spring 9 task

7.2 Trello Work hour

Name	Arju	Sadia	Mushfika	Tingshuo
Time	105hr	105hr	105hr	105hr

7.3 Gantt Chart



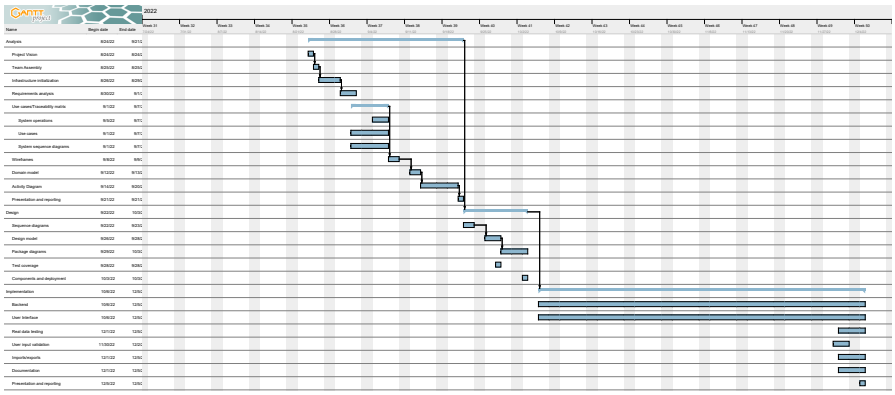
Tasks

Name	Begin date	End date
Analysis	8/24/22	9/21/22
Project Vision	8/24/22	8/24/22
Team Assembly	8/25/22	8/25/22
Infrastructure initialization	8/26/22	8/29/22
Requirements analysis	8/30/22	9/1/22
Use cases/Traceability matrix	9/1/22	9/7/22
System operations	9/5/22	9/7/22
Use cases	9/1/22	9/7/22
System sequence diagrams	9/1/22	9/7/22
Wireframes	9/8/22	9/9/22
Domain model	9/12/22	9/13/22
Activity Diagram	9/14/22	9/20/22
Presentation and reporting	9/21/22	9/21/22
Design	9/22/22	10/3/22
Sequence diagrams	9/22/22	9/23/22
Design model	9/26/22	9/28/22
Package diagrams	9/29/22	10/3/22
Test coverage	9/28/22	9/28/22
Components and deployment	10/3/22	10/3/22
Implementation	10/6/22	12/5/22
Backend	10/6/22	12/5/22
User Interface	10/6/22	12/5/22
Real data testing	12/1/22	12/5/22
User input validation	11/30/22	12/2/22
Imports/exports	12/1/22	12/5/22
Documentation	12/1/22	12/5/22
Presentation and reporting	12/5/22	12/5/22

Resources

Name	Default role
Arju	Team Leader
Scott	Analyst
Tisha	Developer
Rahman	Developer
Noorah	Tester

Gantt Chart


2022
Name
Begin date
End date
Analysis 8/24/22 9/21/2
Project Vision 8/24/22 8/24/2
Team Assembly 8/25/22 8/25/2
Infrastructure initialization 8/26/22 8/26/2
Requirements analysis 8/30/22 9/1/2
User cases/Fcossibility matrix 9/1/22 9/1/2
System operations 9/1/22 9/1/2
Use cases 9/1/22 9/7/2
System sequence diagrams 9/1/22 9/7/2
Wireframes 9/8/22 9/8/2
Domain model 9/12/22 9/13/2
Activity Diagram 9/14/22 9/20/2
Presentation and reporting 9/21/22 9/21/2
Design 9/22/22 10/3/2
Sequence diagrams 9/22/22 9/23/2
Design model 9/26/22 9/26/2
Package diagrams 9/29/22 10/3/2
Test coverage 9/26/22 9/26/2
Components and deployment 10/3/22 10/3/2
Implementation 10/6/22 12/9/2
Backend 10/6/22 12/9/2
User Interface 10/6/22 12/9/2
Real-date testing 12/7/22 12/9/2
User input validation 11/30/22 12/2/2
Imports/exports 12/7/22 12/9/2
Documentation 12/7/22 12/9/2
Presentation and reporting 12/9/22 12/9/2

Resources Chart

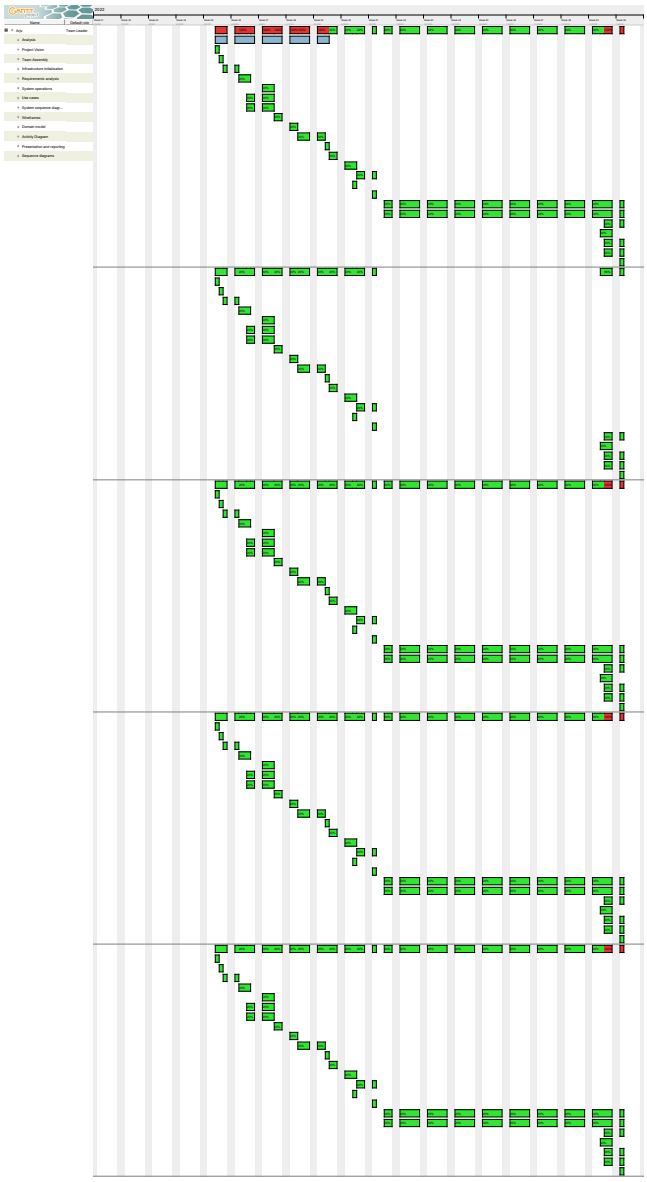


Figure 37: Git contributors

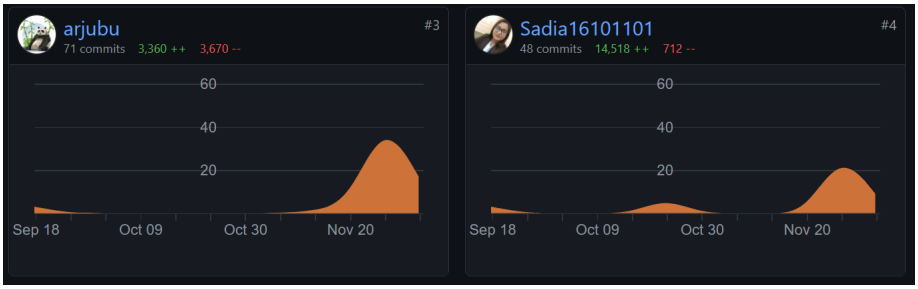


Figure 38: Git contributors Cont.

8 Github

GitHub repository:

9 Github Work Distribution

10 Value Estimation

Each of us spent about 105 hours on this project. The followings are the breakdown for each iteration

1. Iteration 1: 10 hours
2. Iteration 2: 45 hours

3. Iteration 3: 50 hours, extra time due to Thanksgiving.

Assuming an average of $33 hourly wage for a software developer in US, total labor value is about $10395 (105*3*33). We approximated the value of the idea and innovation to be about $7500, and the cost for deployment to $200. This leads to a total value of $18095 for the whole project.