

# CS Department Automated Information Timeline Iteration 2 Submission

Matthew Hays, Pawan Bhandari, Sarah Faron, Tim Klimpel  
The Incredibles

April 1, 2024

# Contents

<b>1</b>	<b>Change Log</b>	<b>5</b>
<b>2</b>	<b>Vision Statement</b>	<b>5</b>
<b>3</b>	<b>Requirements</b>	<b>6</b>
3.1	Functional Requirements . . . . .	6
3.2	Non-Functional Requirements . . . . .	6
<b>4</b>	<b>Glossary</b>	<b>7</b>
<b>5</b>	<b>Domain Model</b>	<b>7</b>
<b>6</b>	<b>Use Cases</b>	<b>9</b>
6.1	Use Case Diagram . . . . .	9
6.2	Use Cases . . . . .	10
6.2.1	UC01: View Media Library . . . . .	10
6.2.2	UC02: View All Posts . . . . .	12
6.2.3	UC03: View Event Calendar . . . . .	12
6.2.4	UC04: Create Event . . . . .	12
6.2.5	UC05: Edit Event . . . . .	14
6.2.6	UC06: Delete Event . . . . .	16
6.2.7	UC07: View Event . . . . .	19
6.2.8	UC08: Submit For Review . . . . .	20
6.2.9	UC09: Review For Approval . . . . .	21
6.2.10	UC10: Create Post . . . . .	24
6.2.11	UC11: Edit Post . . . . .	26
6.2.12	UC12: Delete Post . . . . .	29
6.2.13	UC13: View Post . . . . .	31
6.2.14	UC14: Edit Page of Event . . . . .	32
6.2.15	UC15: Manage Displayed Posts . . . . .	33
6.2.16	UC16: Manage Displayed Media . . . . .	35
<b>7</b>	<b>Activity Diagram: Create Event</b>	<b>39</b>

<b>8 Wireframes</b>	<b>40</b>
8.1 Login Screen . . . . .	40
8.2 Create Post Screen . . . . .	41
8.3 Review Post Screen . . . . .	42
8.4 Faculty Dashboard . . . . .	43
8.5 TV . . . . .	44
8.6 Web UI . . . . .	45
<b>9 Class Diagram</b>	<b>46</b>
<b>10 GRASP Patterns</b>	<b>46</b>
10.1 Annotated Sequence diagrams . . . . .	47
10.1.1 Sequence Diagram: Create Event . . . . .	48
10.1.2 Sequence Diagram: Review for Approval . . . . .	49
10.1.3 Sequence Diagram: Edit Post . . . . .	50
10.1.4 Sequence Diagram: Manage Displayed Media . . . . .	51
10.1.5 Sequence Diagram: Add Page to Event . . . . .	52
<b>11 Object Constraint Language</b>	<b>52</b>
11.1 Describing Constraints with OCL . . . . .	52
11.1.1 Constraint 1: Staging Approved Posts . . . . .	52
11.1.2 Constraint 2: Displaying Posts . . . . .	52
11.1.3 Constraint 3: Notifications . . . . .	53
<b>12 State Machine Diagram</b>	<b>53</b>
<b>13 Package Diagram</b>	<b>54</b>
<b>14 Deployment Diagram</b>	<b>55</b>
<b>15 Component Diagram</b>	<b>56</b>
<b>16 Code</b>	<b>56</b>

## List of Figures

1 Domain Model for CS Department Automated Information Timeline . . . . .	8
---	---

2	Use Case Diagram for CS Department Automated Information Timeline . . . . .	9
3	System Sequence Diagram: View Media Library . . . . .	11
4	System Sequence Diagram: Create Event . . . . .	13
5	System Sequence Diagram: Edit Event . . . . .	15
6	System Sequence Diagram: Delete Event . . . . .	18
7	System Sequence Diagram: View Event . . . . .	20
8	System Sequence Diagram: Review For Approval . . . . .	23
9	System Sequence Diagram: Create Post . . . . .	26
10	System Sequence Diagram: Edit Post . . . . .	28
11	System Sequence Diagram: Delete Post . . . . .	30
12	System Sequence Diagram: View Post . . . . .	32
13	System Sequence Diagram: Manage Displayed Posts . . . . .	34
14	System Sequence Diagram: Manage Displayed Media . . . . .	37
15	Activity Diagram: Create Event . . . . .	39
16	Wireframe: Login Screen . . . . .	40
17	Wireframe: Create Post Screen . . . . .	41
18	Wireframe: Review Post Screen . . . . .	42
19	Wireframe: Faculty Dashboard . . . . .	43
20	Wireframe: TV . . . . .	44
21	Wireframe: Web UI . . . . .	45
22	Class Diagram . . . . .	46
23	Sequence Diagram: Create Event . . . . .	48
24	Sequence Diagram: Review for Approval . . . . .	49
25	Sequence Diagram: Edit Post . . . . .	50
26	Sequence Diagram: Manage Displayed Media . . . . .	51
27	Sequence Diagram: Add page to Event . . . . .	52
28	State Machine Diagram for review and staging of submitted posts, events and pages. . . . .	53
29	Package Diagram . . . . .	54
30	Deployment Diagram . . . . .	55
31	Component Diagram . . . . .	56

# 1 Change Log

1. Added wire frames for the following screens in section 8
  - (a) Login
  - (b) Create a Post
  - (c) Review Posts
2. Identification and discussion about GRASP patterns & Sequence diagrams on section 10
  - (a) Discussion on GRASP patterns pertaining to the project
  - (b) Annotated sequence diagram for below use cases
    - i. Create event
    - ii. Review for approval
    - iii. Edit post
    - iv. Manage displayed media
    - v. Add page to event
3. OCL on section 11
4. State machine diagram on section 12
5. Package diagram on section 13
6. Deployment diagram on section 14
7. Component Diagram on section 15

# 2 Vision Statement

The Baylor University Computer Science Department needs a system for faculty to passively share information on department news and events with students and other faculty members. To fulfill this need, our team is developing an automated, web-based system that will populate faculty content, perform content rolling, and automate content updates.

Using our system, any faculty can propose a post, but only one admin/reviewer can approve the post for publishing. In addition to the embedded post timeline, the system also has an event calendar. Similarly, any faculty can propose an event to add to the calendar, but only one admin/reviewer can approve the event for publishing to the event calendar.

This content is available for viewing through a TV screen in the department lobby, which is manually managed by our CS Department office manager. The system is configured to either (1) show only the most recent content on the lobby TV or (2) show staged media tagged manually by the office manager. The system uses WebSockets to roll and update recent content automatically, as well as a WYSIWYG editor to modify and tag static content pages.

Since not everything can be displayed in the lobby, the system supports a web view accessible to all visitors with a mobile device via QR code. Through the web application, all audience members can browse all

content available beyond the posts and events tagged for the lobby TV.

The system uses Spring middleware with database and REST API to support the two described web-based and UI-responsive frontends.

## 3 Requirements

### 3.1 Functional Requirements

**REQ01:** Faculty members can create a post for review, which will initially default to a proposed state until approved.

**REQ02:** An admin/reviewer must approve a post before it can be published to the site.

**REQ03:** A user can add events to an event calendar that can be displayed on screen.

**REQ04:** A user can add pictures to posts.

**REQ05:** A user can add videos to posts.

**REQ06:** A user can add HTML pages to posts.

**REQ07:** A user can tag approved posts for display on a large format screen.

**REQ08:** The system will allow management of which posts are displayed.

**REQ09:** The system will allow management of which media is displayed.

**REQ10:** The system will fall back to the 10 most recent posts if no posts are tagged, or less than 10 posts are tagged.

**REQ11:** The Office manager will use the WYSIWYG editor to modify static content posts.

**REQ12:** The Office Manager will use the WYSIWYG editor to tag posts for display on screen.

**REQ13:** The system will allow audiences to use their mobile devices (e.g., through a QR code that brings them to the website) to browse all content beyond the 10 most recent posts.

**REQ14:** The system will allow audiences to use their mobile device to browse the calendar.

**REQ15:** A user will authenticate and be assigned a valid role or view the content as a non-authenticated user.

### 3.2 Non-Functional Requirements

**REQ16:** The system will have the following user roles: Faculty, Admin, and Office Manager.

**REQ17:** Content rolling and updates to content will be completed via WebSockets.

**REQ18:** The backend will consist of a Spring Framework REST API with a database and appropriate middleware.

**REQ19:** The system will manage a media library which consists of all posted pictures and videos.

**REQ20:** The system will consist of a front-end display: a television screen.

**REQ21:** The system will consist of a front-end display: a web-based display optimized for all screen sizes.

**REQ22:** The design of the user interface must utilize Baylor's official color scheme.

- Baylor Green: #154734
- University Gold: #FFB81C
- Secondary and accent colors can be chosen from the lists provided here.

**REQ23:** The system will have a WYSIWYG editor.

## 4 Glossary

The following terms are important to the development of the project:

- **WYSIWYG Editor:** What You See Is What You Get, a word processing editor that allows users to view and update content exactly as it appears on the web UI
- **WebSocket:** A computer protocol which provides bidirectional communications over a single TCP connection
- **Post:** A picture, video, or manually created HTML page
- **Page:** An HTML page which can be added to a post or event
- **Event Calendar:** A calendar which is displayed on screen which faculty can add events to
- **Media Library:** A library which stores both picture and video content
- **Faculty:** A university employee who uses the system under discussion to create posts and events
- **Admin:** A privileged user who holds all access rights
- **Reviewer:** A university employee who holds special privilege over the system under discussion to approve or reject posts and events submitted by Faculty
- **Officer Manager:** A university employee who holds special privilege over the system under discussion to manage which posts are displayed at any given time

## 5 Domain Model

The below Domain model represents a Draft version of the current planned domain model. Classes/objects are noted with a *\_D* to indicate they are *draft* classes.

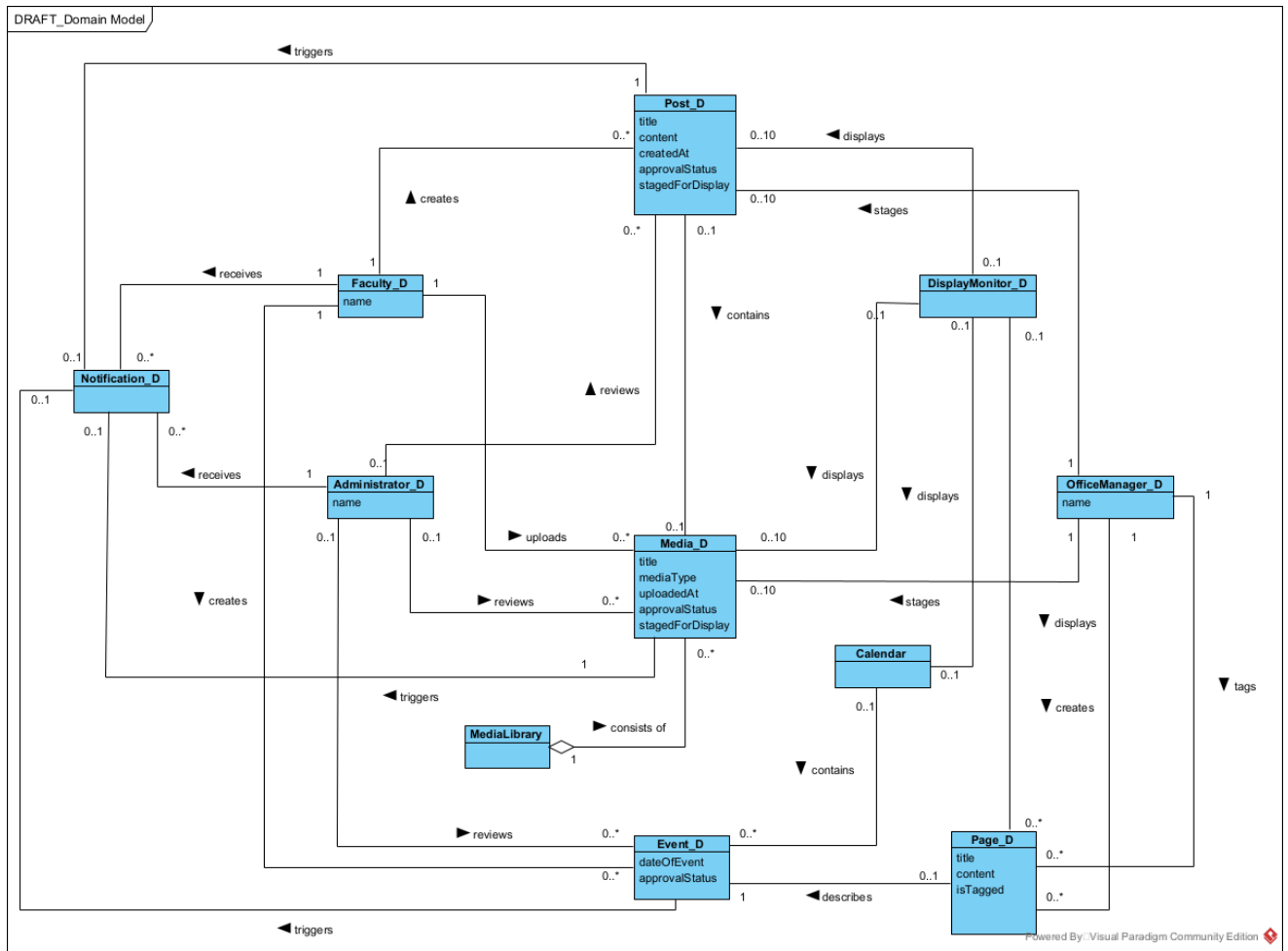


Figure 1: Domain Model for CS Department Automated Information Timeline



## 6 Use Cases

### 6.1 Use Case Diagram

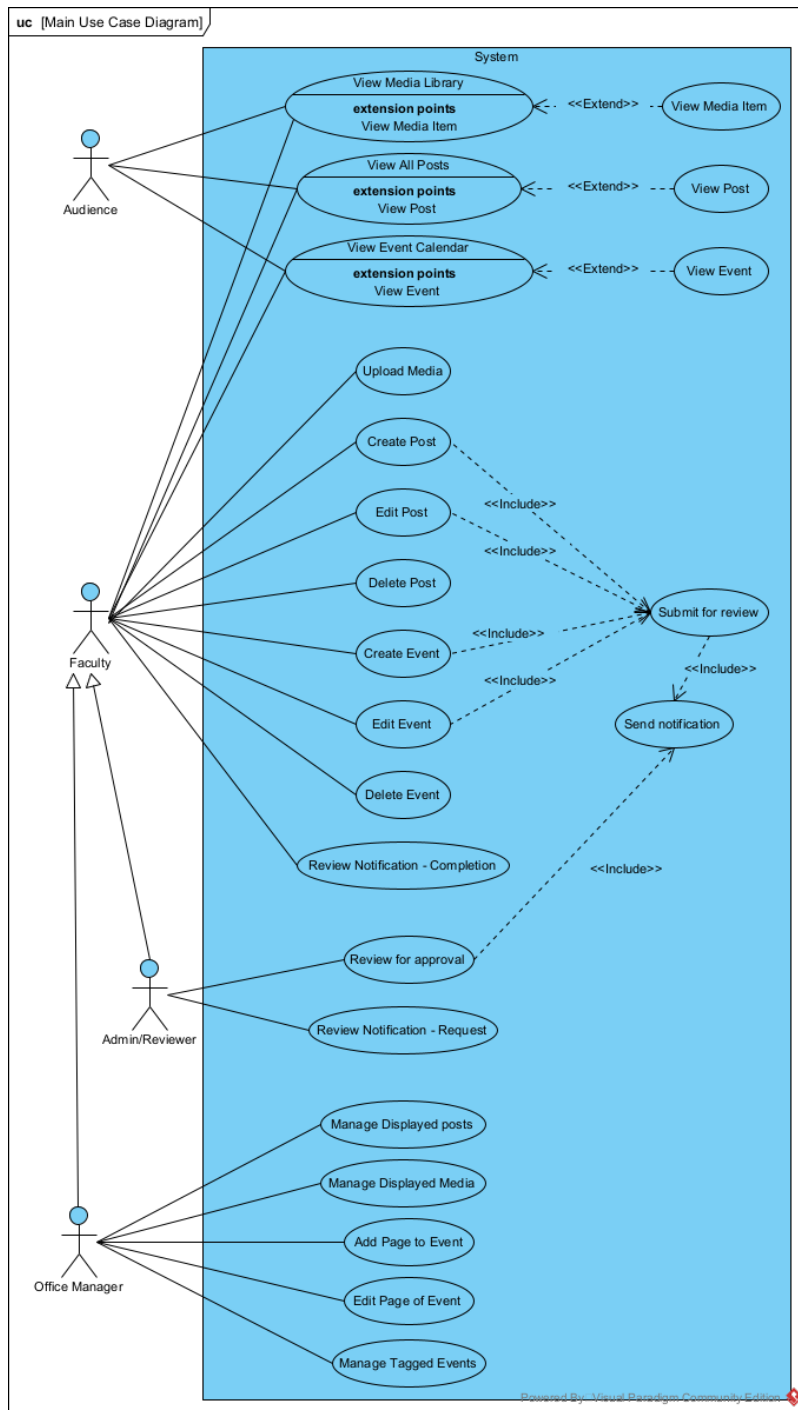


Figure 2: Use Case Diagram for CS Department Automated Information Timeline

## 6.2 Use Cases

There are a total of 16 use cases written at this stage in the project. Each team member was assigned 3 use cases to fully analyze. Each of these 12 use cases are written in fully-dressed form and accompanied by System Sequence Diagrams (SSDs) and Operations Contracts (OCs). The remaining 4 use cases are written in brief/casual form.

The assigned (fully-dressed) use cases are as follows:

Bhandari	Faron	Hays	Klimpel
UC11	UC05	UC04	UC15
UC12	UC06	UC09	UC16
UC13	UC07	UC10	UC01

### 6.2.1 UC01: View Media Library

**ID:** UC01 (View Media Library)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Audience

**Stakeholders and Interests:**

- Audience wants the ability to view all shared media through the media library
- Faculty wants to be able to allow guests to view media shared by them
- Office Manager wants to ensure that all media is available to be seen through the web portal
- Admin wants to be sure that approved media can be displayed to the audience

**Preconditions:**

- Media library contains approved media items

**Postconditions:**

- Media library is shared with the audience

**Main Success Scenario:**

1. Audience member scans available QR code
2. System provides appropriate URL for main home page
3. Audience member navigates to URL
4. Audience member selects “Media library” option
5. System presents full media library view

**Alternative Flows:**

3A: URL is down

1. Audience member alerts faculty or office manager
2. Office manager or faculty review system status

#### 5A: Multiple pages available

1. System presents paged view with limit of items
2. Audience member selects 'next page'
3. System returns next set of results
4. Continue until end of media library is returned

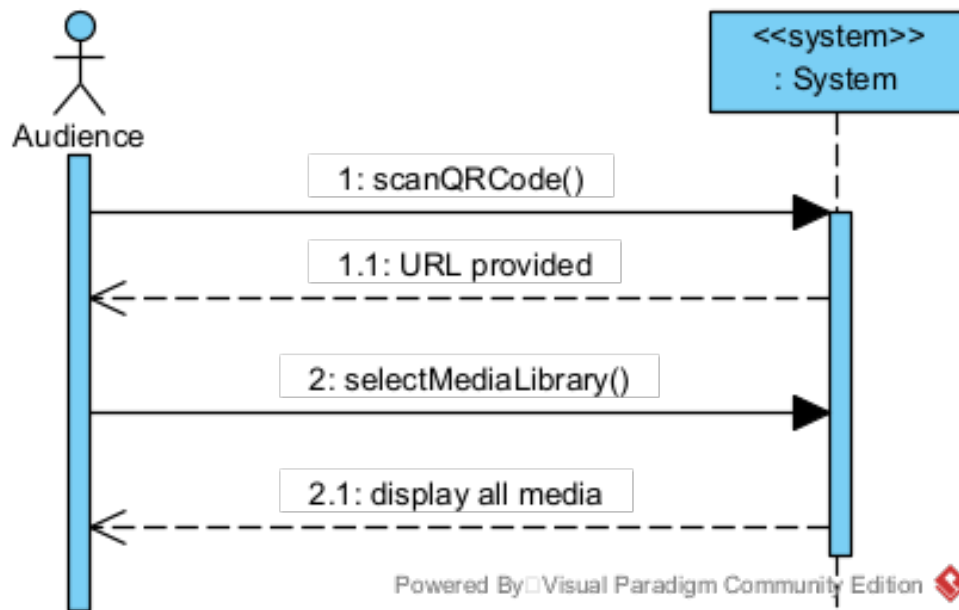


Figure 3: System Sequence Diagram: View Media Library

**Operation:** `selectMediaLibrary()`

**Cross-References:** UC01 (View Media Library)

**Pre-conditions:**

- Media items,  $m$ , exists in system
- $m$  is approved for view

**Post-conditions:**

- List of media,  $ml$ , is created
- $ml$  is returned to user

### 6.2.2 UC02: View All Posts

**ID:** UC02 (View All Posts)

**Main Success Scenario:** An audience member has access to either QR code or URL of web application version of system. User navigates to the home page and selects “View All Posts” option. The system returns paginated view of all posts. If more than the page limit exists, multiple page options are presented to user for navigation.

**Alternate flow:** None.

### 6.2.3 UC03: View Event Calendar

**ID:** UC03 (View Event Calendar)

**Precondition:** Faculty is identified and authenticated in the system.

**Main Success Scenario:** An audience member has access to either QR code or URL of web application version of system. User navigates to the home page and clicks “Event Calendar” link. The system returns view of Event Calendar, defaulted to open on the current month.

### 6.2.4 UC04: Create Event

**ID:** UC04 (Create Event) **Scope:** CS Automated Information Timeline

**Level:** User Goal

**Stakeholders and Interests:**

- Faculty: A person that works for the university and is interested in gaining visibility of their post and/or event.
- Admin/Reviewer: A person that works for the university and approves and/or removes posts and/or events from the system.

**Preconditions:**

- Faculty has been identified and authenticated.

**Postconditions:**

- Event has been persisted.
- Event has been placed in the proposed state.
- An event notification has been persisted.

**Main Success Scenario:**

1. Faculty writes the event using the event creation tool.
  - (a) The event title is written by the faculty user.
  - (b) The event body is written by the faculty user.

2. Faculty reviews the event draft.
3. Faculty submits the event to the system.
  - (a) An event id is generated by the system.
  - (b) A timestamp is generated by the system.
  - (c) The user id of the faculty user is attached to the event object.
  - (d) The system assigns the status of the event to the proposed state.
4. The system generates a notification object.
  - (a) The event object is attached to the notification object.
5. The notification object is persisted.
6. The event object is persisted.
7. The system returns the persisted event object to the faculty user.

**Extensions:**

- \*.a. Anytime the system does not respond,
  1. Faculty will notify the Admin/Reviewer.
  2. Admin/Reviewer will restart the system.
- 5.a. If the notification is not persisted,
  1. The system returns an error message to the faculty user.
  2. The faculty user resubmits the event.
- 6.a. If the event is not persisted,
  1. The system returns an error message to the faculty user.
  2. Faculty reattempts the submission of the event.

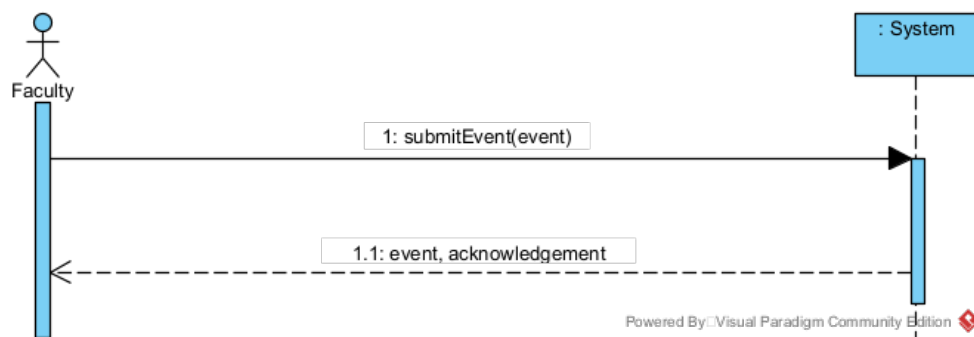


Figure 4: System Sequence Diagram: Create Event

**Operation:** submitEvent(event: Event)  
**Cross References:** UC04 (Create Post)  
**Preconditions:**

- Faculty, f, has been identified and authenticated.

**Postconditions:**

- Event, e, was created.
- Event e.status was set to “proposed”.
- Event, e, was persisted.

### 6.2.5 UC05: Edit Event

**ID:** UC05 (Edit Event)

**Scope:** CS Automated Information Timeline

**Level:** User Goal

**Primary Actor:** Faculty

**Stakeholders and Interests:**

- Audience: Wants to view up-to-date information about events in the CS Department
- Faculty: Wants the ability to provide updated information on events they submitted
- Office Manager: Wants to display accurate event information on the calendar on the Lobby TV
- Admin/Reviewer: Wants to update previously approved events with up-to-date information provided by the Faculty who submitted those events

**Preconditions:** Faculty created an event (UC04) and submitted it for review (UC08). Admin/Reviewer reviewed the event (UC09) and approved. Admin/Reviewer is identified and authenticated in the system. Faculty is identified and authenticated in the system and is viewing the event calendar (UC03).

**Postconditions:** Event status is in a proposed state.

**Main Success Scenario:**

1. Faculty clicks the “View My Events” link on the event calendar page.
2. System returns a list of existing events authored by Faculty that have been approved.
3. Faculty selects the event they wish to edit from the returned list of their previously approved events.
4. System returns the event page for the selected event.
5. Faculty clicks the “Edit This Event” button on the event page.
6. Faculty enters the updated event information into the submission form:
  - Event date
  - Event time
  - Event description
7. Faculty submits the updated event information for Admin/Reviewer review.
8. System provides Faculty with a submission confirmation message.

**Extensions (or Alternate Flows):**

3-4a. Faculty begins editing the event, then decides not to submit the updates for review:

1. Upon leaving the web page, any changes to event date, time, or description entered into the submission form are erased.
2. Selected event remains unchanged in the system.

3b. Faculty does not see the event they wish to edit in the returned list because the event has not yet been approved by the Admin/Reviewer:

1. Faculty must wait for approval from Admin/Reviewer on the original event before the event is eligible for editing.

**Special Requirements:** None

**Technology and Data Variations List:**

4a. Date input is required in the format MM/DD/YYYY.

4b. Time input is required in the Standard Time format in the Central Time Zone.

**Frequency of Occurrence:** Could be nearly continuous if Admin/Reviewer also reviews edited events and approves them (UC09) nearly continuously.

**Open Issues:** None

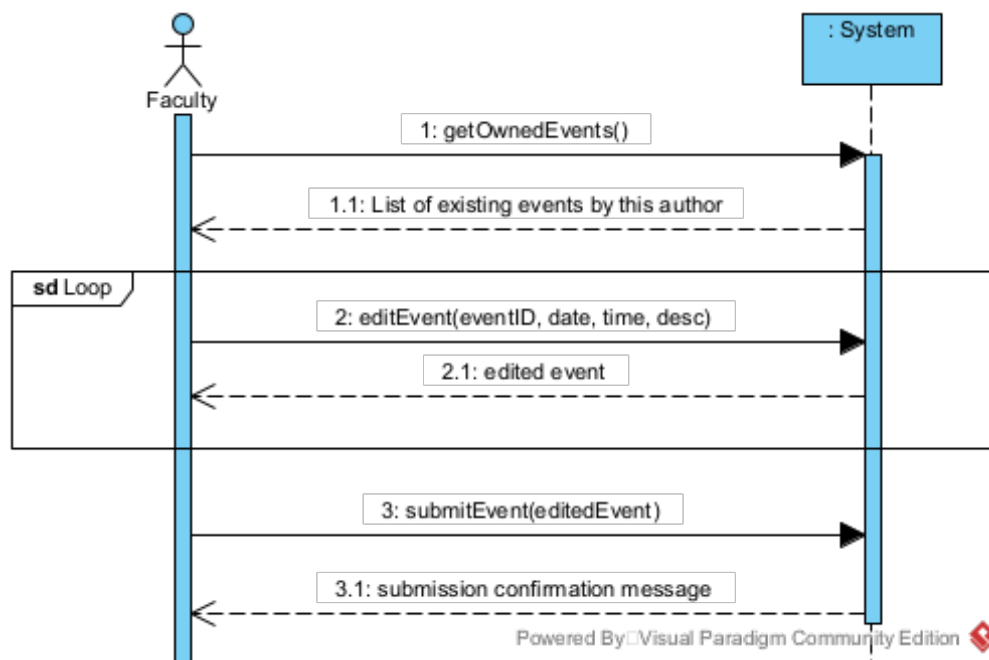


Figure 5: System Sequence Diagram: Edit Event

**Operation:** getOwnedEvents()

**Cross-References:** UC05 (Edit Event), UC06 (Delete Event), UC07 (View Event)

**Pre-conditions:** Event object has been created. Event has been associated with Faculty. Event attribute status is set to approved.

**Post-conditions:** None.

**Operation:** editEvent(eventID, date, time, desc)

**Cross-References:** UC05 (Edit Event)

**Pre-conditions:**

- Event object has been created.
- Event attribute status is set to approved.
- Event has been associated with Faculty.

**Post-conditions:** editedEvent has been created.

- editedEvent object has been created.
- editedEvent attributes (date, time, desc) have been updated.
- editedEvent has been associated with Faculty.

**Operation:** submitEvent(editedEvent)

**Cross-References:** UC05 (Edit Event)

**Pre-conditions:**

- editedEvent object has been created.
- editedEvent attributes (date, time, desc) have been updated.
- editedEvent has been associated with Faculty.

**Post-conditions:**

- Notification object is created.
- Notification object is associated with editedEvent, Faculty, and Admin/Reviewer.

#### 6.2.6 UC06: Delete Event

**ID:** UC06 (Delete Event)

**Scope:** CS Automated Information Timeline

**Level:** User Goal

**Primary Actor:** Faculty

**Stakeholders and Interests:**

- Audience: Wants to view up-to-date information about events in the CS Department
- Faculty: Wants the ability to delete events that they submitted that are no longer happening
- Office Manager: Wants to display accurate event information on the calendar on the Lobby TV



**Preconditions:** Faculty created an event (UC04) and submitted it for review (UC08). Admin/Reviewer reviewed the event (UC09) and approved. Faculty is identified and authenticated in the system and is viewing the event calendar (UC03).

**Postconditions:** Event status is in a deleted state.

**Main Success Scenario:**

1. Faculty clicks the “View My Events” link on the event calendar page.
2. System returns a list of existing events authored by Faculty that have been approved.
3. Faculty selects the event they wish to delete from the returned list of their previously approved events.
4. System returns the event page for the selected event.
5. Faculty clicks the “Delete This Event” button on the event page.
6. System prompts Faculty to confirm the removal of the event from the event calendar.
7. Faculty confirms the removal of the event from the event calendar by clicking “Yes”.
8. System provides Faculty with a confirmation message.

**Extensions (or Alternate Flows):**

3a. Faculty does not see the event they wish to delete in the returned list because the event has not yet been approved by the Admin/Reviewer:

1. Faculty must wait for approval from Admin/Reviewer on the original event before the event is eligible for removal from the event calendar.

7a. Faculty decides they do not want to remove the event from the event calendar:

1. Faculty clicks the “No” button and is redirected to the list of existing events authored by Faculty that have been approved.

**Special Requirements:** None

**Technology and Data Variations List:** None

**Frequency of Occurrence:** Maximum once per submitted and approved event.

**Open Issues:** None

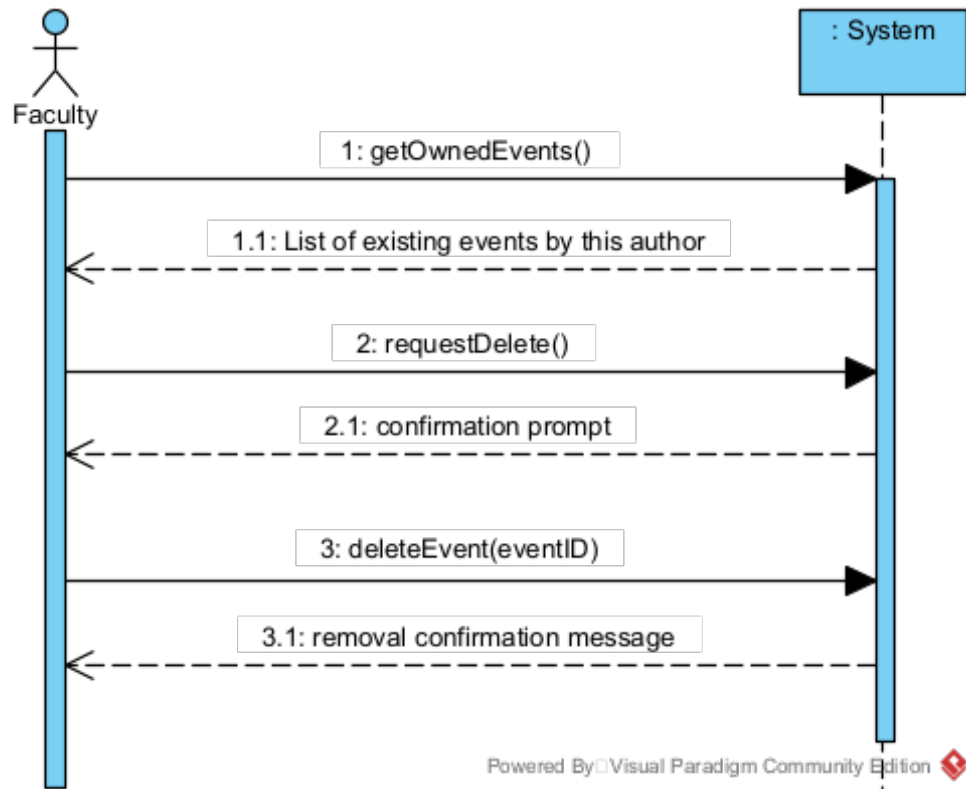


Figure 6: System Sequence Diagram: Delete Event

**Operation:** requestDelete()

**Cross-References:** UC06 (Delete Event)

**Pre-conditions:**

- Event object has been created.
- Event attribute status is set to approved.
- Event has been associated with Faculty.

**Post-conditions:** None

**Operation:** deleteEvent(eventID)

**Cross-References:** UC06 (Delete Event)

**Pre-conditions:**

- Event object has been created.
- Event attribute status is set to approved.
- Event has been associated with Faculty.

**Post-conditions:**

- Event attribute status is set to deleted.
- Event associated with Faculty has been broken.

### 6.2.7 UC07: View Event

**ID:** UC07 (View Event)

**Scope:** CS Automated Information Timeline

**Level:** User Goal

**Primary Actor:** Faculty

**Stakeholders and Interests:**

- Audience: Wants to view up-to-date information about events in the CS Department
- Faculty: Wants the ability to view events they have submitted
- Office Manager: Wants to display calendar events on the Lobby TV
- Admin/Reviewer: Wants to view events submitted by Faculty so they can review them

**Preconditions:** Faculty created an event (UC04) and submitted it for review (UC08). Admin/Reviewer reviewed the event (UC09) and approved. Faculty is identified and authenticated in the system and is viewing the event calendar (UC03).

**Postconditions:** Event status is in an approved state.

**Main Success Scenario:**

1. Faculty clicks the “View My Events” link on the event calendar page.
2. System returns a list of existing events authored by Faculty that have been approved.
3. Faculty selects the event they wish to view from the returned list of their previously approved events.
4. System returns the event page for the selected event.

**Extensions (or Alternate Flows):**

3a. Faculty does not see the event they wish to view in the returned list because the event has not yet been approved by the Admin/Reviewer:

1. Faculty must wait for approval from Admin/Reviewer on the original event before the event is eligible for viewing from the event calendar.

**Special Requirements:** None

**Technology and Data Variations List:** None

**Frequency of Occurrence:** Continuous

**Open Issues:** None

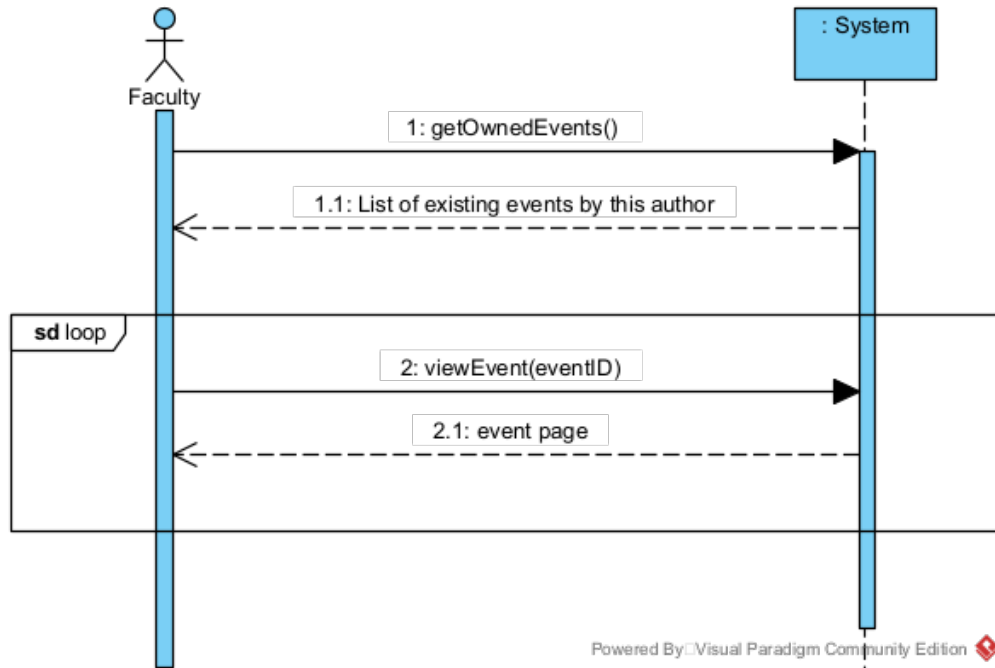


Figure 7: System Sequence Diagram: View Event

**Operation:** viewEvent(eventID)

**Cross-References:** UC07 (View Event)

**Pre-conditions:**

- Event object has been created.
- Event attribute status is set to approved.
- Event has been associated with Faculty.

**Post-conditions:** None

### 6.2.8 UC08: Submit For Review

**Main Success Scenario:** A faculty user submits a post, event, media, or page to the system. The system generates an id and a timestamp that is set to the system time upon submission and associates these attributes with the post, event, media, or page object. The system associates the faculty user that submitted with the post, event, media, or page. The system assigns the status of the post, event, media, or page to the proposed status. The system generates a notification object and attaches the post, event, media, or page object with the notification. The system persists both the created post, event, media, or page object and the created notification object to the database. The created post, event, media, or page is returned to the faculty user.

**Alternative Flows:**

- The post, event, media, page, or notification object is not persisted.

- The system will respond to the faculty user with an error message indicating the reason for persistence failure.
- The faculty member will correct the identified error and attempt to resubmit the post, event, media, or page item.
- If the system does not respond within a given timeframe.
  - The system will return an error message to the faculty user indicating a timeout has occurred.
  - The faculty user will reattempt the submission of the post, event, media, or page item.
    - \* If the system continues to fail to respond within a given timeframe.
      - The faculty user will inform the admin/reviewer of the error.
      - The admin/reviewer will restart the system.
      - The faculty user will reattempt submission of the post, event, media, or page item.

### 6.2.9 UC09: Review For Approval

**ID:** UC09 (Review for Approval)

**Scope:** CS Automated Information Timeline

**Level:** User Goal

**Stakeholders and Interests:**

- Faculty: A person that works for the university and is interested in gaining visibility of their post and/or event.
- Administrator/Reviewer: A person that works for the university and is interested in reviewing and approving requests for system content additions.

**Preconditions:**

- Administrator/Reviewer a has been identified and authenticated.
- Notification n has been created and persisted.
- n.event, n.post, or n.media has been set.

**Postconditions:**

- Notification n.entity was identified as either a Post, Event, or Media.
- n.entity.status was set to either “Approved” or “Rejected”.
- If the review was favorable, n.entity.status was set to “Approved”.
- If the review was unfavorable, n.entity.status was set to “Rejected”.
- n.reviewer was set to a.
- n.entity was persisted.

**Main Success Scenario:**

1. Administrator/Reviewer, a, gets all notifications.

2. For each Notification, n, in notifications, Administrator/Reviewer, a, performs manual review of the Post, Event, or Media attached to Notification, n.
3. Administrator/Reviewer, a, updates the Post, Event, or Media that was attached to Notification, n, to either “Approved” or “Rejected”.
4. If the review is favorable, n.event.status, n.post.status, or n.media.status is set to “Approved”.
5. If the review is not favorable, n.event.status, n.post.status, or n.media.status is set to “Rejected”.
6. The system persists the updated Post, Event, or Media with the status set by Administrator/Reviewer, a.
7. The system associates Administrator/Reviewer, a, with Notification n.
8. Notification, n, is persisted.
9. The Event, Post, or Media is persisted.

**Extensions:**

2a. There are multiple objects, Post, Event, or Media, attached to Notification n.

- Administrator/Reviewer, a, deletes Notification, n.

**Special Requirements:** None

**Technology and Data Variation List:**

- Notification n.post is set and of Post type; n.event and n.media are null.
- Notification n.event is set and of Event type; n.post and n.media are null.
- Notification n.media is set and of Media type; n.post and n.event are null.

**Frequency of Occurrence:** Could be nearly continuous.



Figure 8: System Sequence Diagram: Review For Approval

**Operation:** getNotifications()

**Cross References:** UC09 (Review for Approval)

**Preconditions:**

- Admin/Reviewer, a, has been identified and authenticated.
- Notification, n, has been created and persisted.
- Notification n.entity has been set to Post p, Event e, and/or Media m.

**Postconditions:** None

**Operation:** deleteNotification(notification: Notification)

**Cross References:** UC09 (Review for Approval)

**Preconditions:**

- Admin/Reviewer, a, has been identified and authenticated.
- Notification, n, has been created and persisted.
- Notification n.entity has been set to Post p, Event e, and/or Media m.

**Postconditions:**

- Association n.entity was disassociated.

- All entities, Post p, Event e, and/or Media m, from association n.entity were removed from persistence and destroyed.
- Notification, n, was removed from persistence and destroyed.

**Operation:** setStatus(status: String)

**Cross References:** UC09 (Review for Approval)

**Preconditions:**

- Admin/Reviewer, a, has been identified and authenticated.
- Notification, n, has been created and persisted.
- n.reviewer has been set to Admin/Reviewer a.
- n.entity has been set to one of Post p, Event e, or Media m.

**Postconditions:**

- Notification n.entity.status was set to status.
- Notification n.reviewer was set to a.
- Notification, n, was persisted.

#### 6.2.10 UC10: Create Post

**ID:** UC10 (Create Post)

**Scope:** CS Automated Information Timeline

**Level:** User Goal

**Stakeholders and Interests:**

- Faculty: A person who works for the university and is interested in gaining visibility of this post and/or event.
- Admin/Reviewer: A person who works for the university and approves and/or removes posts and/or event from the system.

**Preconditions:**

- Faculty, f, has been identified and authenticated.

**Postconditions:**

- Post, p, was created.
- p.status was set to "Pending".
- p.createdBy was set to Faculty f.
- Post, p, was persisted.
- Notification, n, was created.



- n.post was set to Post p.
- Notification, n, was persisted.

**Main Success Scenario:**

1. Faculty writes the post using the post creation tool.
2. The post title is written by the faculty user.
3. The post body is written by the faculty user.
4. Faculty reviews the post draft.
5. Faculty submits the post to the system.
6. A post id is generated by the system.
7. A timestamp is generated by the system.
8. The user id of the faculty user is attached to the post object.
9. The system assigns the status of the post to the proposed state.
10. The system generates a notification object.
11. The post object is attached to the notification object.
12. The post object is persisted.
13. The notification object is persisted.
14. The system returns the persisted post object to the faculty user.

**Extensions:**

\*.a. Anytime the system does not respond.

1. Faculty will notify the Admin/Reviewer.
2. Admin/Reviewer will restart the system.
3. Faculty will recreate and submit the post.

5.a. Anytime the system does not respond.

1. The system will return an error message to the faculty user.
2. The faculty user will resubmit the post.

6.a. If the post is not persisted.

1. The system will return an error message to the faculty user.
2. The faculty user will reattempt the submission of the post.

**Special Requirements:** None

**Technology and Data Variations List:**

1. Date will be of the format “yyyy-MM-ddTHH:mm:ss”.
2. Id will be of the format of a universally unique identifier, UUID.

**Frequency of Occurrence:** Could be nearly continuous.



Figure 9: System Sequence Diagram: Create Post

**Operation:** createPost(post: Post)

**Cross References:** UC10 (Create Post)

**Preconditions:**

- Faculty, f, has been identified and authenticated.

**Postconditions:**

- Post, p, was created.
- p.status was set to “Pending”.
- p.createdBy was set to Faculty f.
- Post, p, was persisted.
- Notification, n, was created.
- n.post was set to Post p.
- Notification, n, was persisted.

#### 6.2.11 UC11: Edit Post

**ID:** UC11 (Edit Post)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Faculty or Admin/Reviewer

**Stakeholders and Interests:**

- Audience: A person that is interested in viewing all approved content on the system using their mobile device.

- Faculty: A person that works for the university and is interested in gaining visibility of their post and/or event.
- Office Manager: A person that works for the university and is interested in prioritizing the order of posts and/or events.
- Admin/Reviewer: A person that works for the university and approves and/or removes posts and/or events from the system.

**Preconditions:**

- Faculty has been identified and authenticated.
- A post has been created.

**Postconditions:**

- Post has been updated and saved to database.
- Media library has been updated with new images and videos from the updated post.
- The display board reflects the updated post flagged for display.

**Main Success Scenario:**

1. User selects a post to edit
2. System opens the editable view for the selected post with options to edit the text, delete the photos and videos, if present, on the post, upload photos and videos on the post and update the display flag.
3. User saves the updated post.
4. System presents the confirmation that the post is updated

**Alternative Flows:**

a. At any time, system fails or becomes unresponsive and does not provide an error message

1. User performs a hard refresh on the browser (ctrl + f5 or shift + reload)
2. System reloads the editable view for the post

b. (3.a) Uploaded media is in unsupported format or exceeds the file size limit

1. System prompts the user with the appropriate error message
2. User acknowledges the error and reuploads the media in supported format and size.

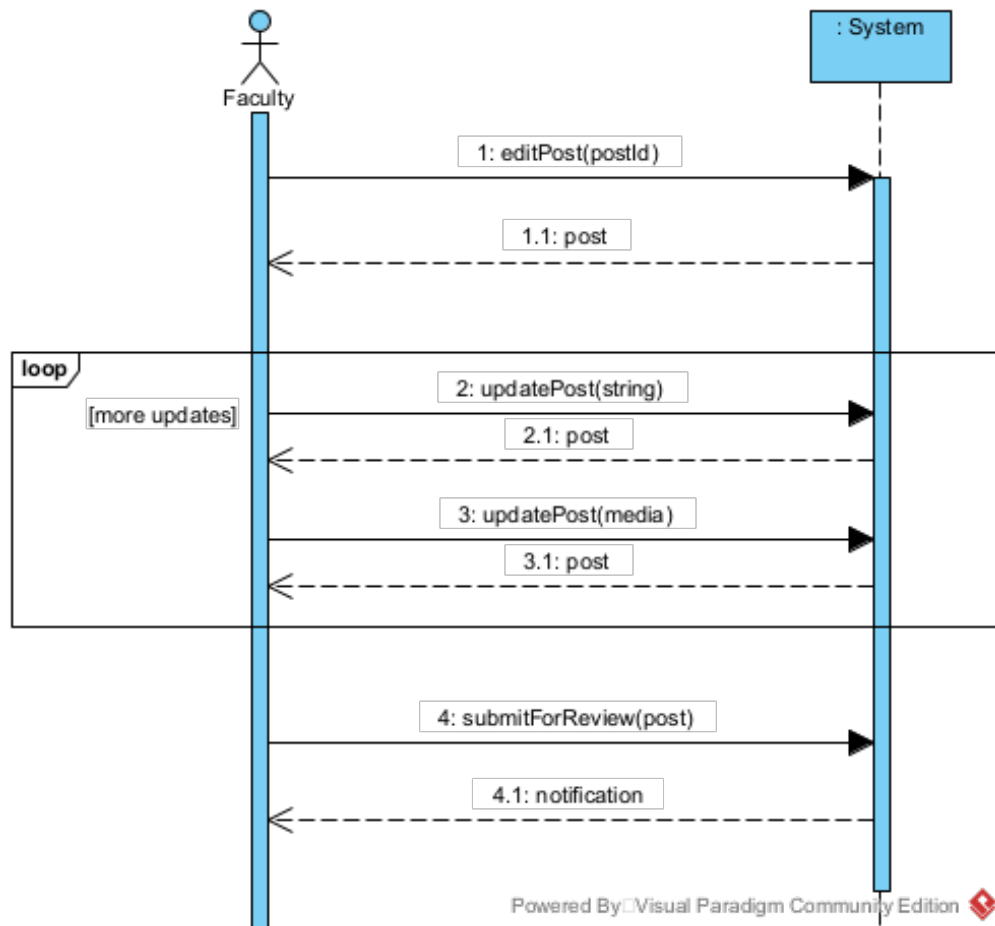


Figure 10: System Sequence Diagram: Edit Post

**Operation:** updatePost(post)

**Cross Reference:** UC11 (Edit Post)

**Preconditions:**

- Faculty has been identified and authenticated.
- A post has been created and persisted in the database.
- Update operation is underway.

**Postconditions:**

- A Post instance, post was created
- post was associated with the Update operation.
- post was modified
- Modified post was persisted to the database

### 6.2.12 UC12: Delete Post

**ID:** UC12 (Delete Post)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Faculty or Admin/Reviewer

**Stakeholders and Interests:**

- Audience: A person that is interested in viewing all approved content on the system using their mobile device.
- Faculty: A person that works for the university and is interested in gaining visibility of their post and/or event.
- Office Manager: A person that works for the university and is interested in prioritizing the order of posts and/or events.
- Admin/Reviewer: A person that works for the university and approves and/or removes posts and/or events from the system.

**Preconditions:**

- Faculty has been identified and authenticated.
- A post to be deleted exists.

**Postconditions:**

- The post has been deleted from the database.
- Media library has been updated.
- The display board no longer displays the deleted post.

**Main Success Scenario:**

1. User navigates to the admin dashboard and accesses the manage post view.
2. User selects the post to be managed and clicks the delete post button.
3. System presents the confirmation window prompting the user if they are sure about deleting the post
4. User confirms deletion.
5. The system deletes the selected post from the database and updates the display board contents.
6. System presents the confirmation that the post is deleted

**Alternative Flows:**

a. At any time, system fails or becomes unresponsive and does not provide an error message

1. User performs a hard refresh on the browser (ctrl + f5 or shift + reload)
2. System reloads the editable view for the post

b. (4.a) User does not confirm post deletion (clicks no)

1. No action is taken, and the post is displayed in manage post view

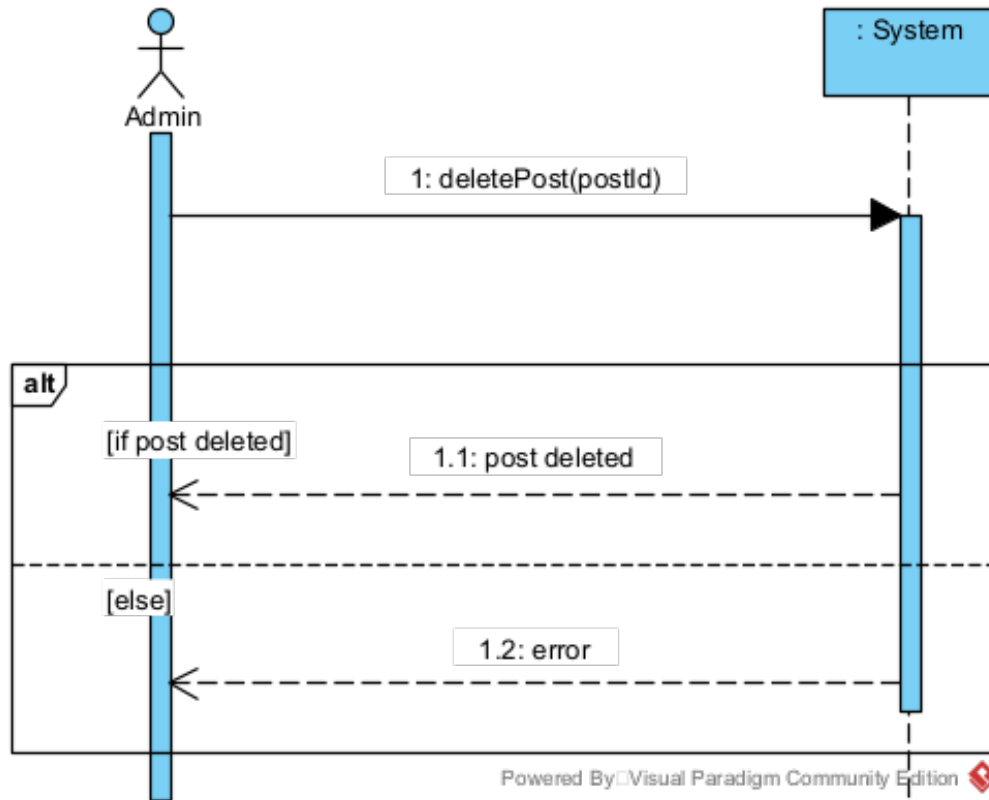


Figure 11: System Sequence Diagram: Delete Post

**Operation:** deletePost(postId)

**Cross Reference:** UC12 (Delete Post)

**Preconditions:**

- Faculty has been identified and authenticated.
- A post has been created and persisted in the database.
- Delete operation is underway.

**Postconditions:**

- A Post instance, post was created
- post was associated with the Delete operation.
- post was deleted

### 6.2.13 UC13: View Post

**ID:** UC13 (View Post)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Audience

**Stakeholders and Interests:**

- Audience: A person that is interested in viewing all approved content on the system using their mobile device.
- Faculty: A person that works for the university and is interested in gaining visibility of their post and/or event.
- Office Manager: A person that works for the university and is interested in prioritizing the order of posts and/or events.
- Admin/Reviewer: A person that works for the university and approves and/or removes posts and/or events from the system.

**Preconditions:**

- A post has been created and staged for display.

**Postconditions:** None

**Main Success Scenario:**

1. User accesses the web application UI by scanning the QR code on the display board.
2. System presents the web view with posts staged for display

**Alternative Flows:**

- a. At any time, system fails or becomes unresponsive and does not provide an error message
  1. User performs a hard refresh on the browser (ctrl + f5 or shift + reload)
  2. System reloads the editable view for the post

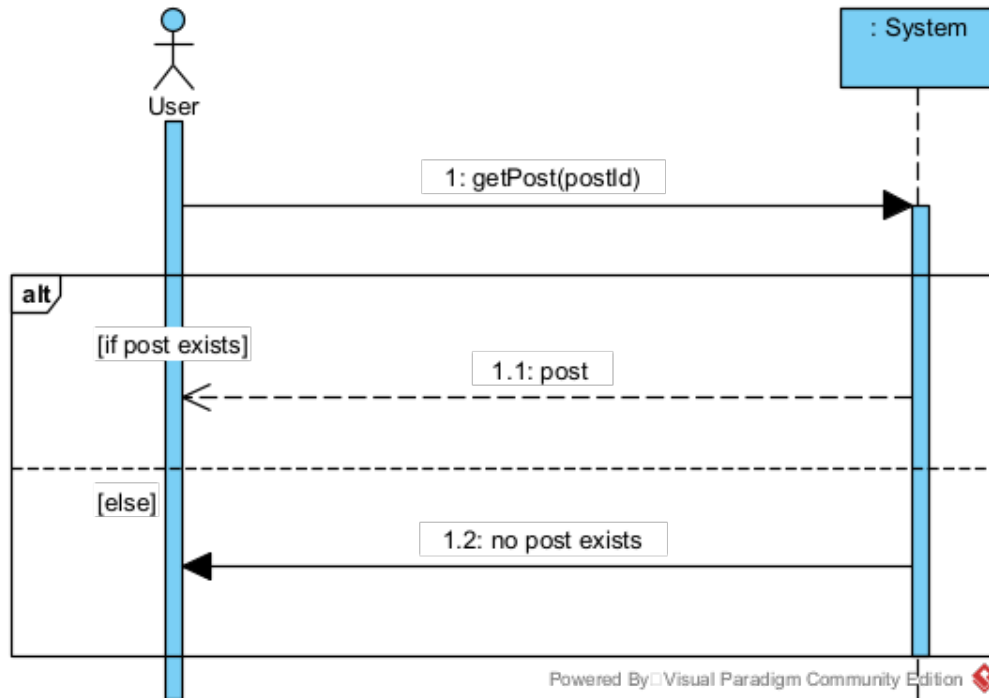


Figure 12: System Sequence Diagram: View Post

**Operation:** getPost(postId)

**Cross Reference:** UC13 (View Post) **Preconditions:**

- Post exists and is staged for display
- User is attempting to view a specific post (request)

**Postconditions:**

- A PostService instance, postService was created
- A PostRepository instance postRepository was created
- A Post instance post was created
- post was associated with the request

#### 6.2.14 UC14: Edit Page of Event

**ID:** UC14 (Edit Page of Event)

**Precondition:** Office Manager has been identified and authenticated.

**Main Success Scenario:** Office Manager can edit HTML page of the event using the WYSIWYG (What You See Is What You Get) editor. Office Manager can select an event and click the edit option, which will allow them to use the WYSIWYG editor to compose and submit the HTML page associated with the event. The changes will take effect immediately and will be reflected on the display board.



### 6.2.15 UC15: Manage Displayed Posts

**ID:** UC15 (Manage Displayed Posts)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Office Manager

**Stakeholders and Interests:**

- Admin/Reviewer: Wants to ensure a curated set of posts are displayed for guests
- Guest/Audience: Wants to be able to see priority posts from the department
- Faculty: Wants to ensure their relevant/priority posts are displayed to guests

**Preconditions:**

- User with Office Manager role has been authenticated
- More than 10 posts exist in the system to allow assigning of priority

**Postconditions:**

- Curated list of posts are displayed on the TV display
- The ordering of displayed posts is correct

**Main Success Scenario:**

1. User navigates to view all posts
2. System displays all posts
3. User selects post to be displayed on the main TV display
4. The system presents the current list of curated posts
5. Repeat step 3 and 4 until all desired posts are selected by the user
6. System presents final list of posts to be displayed on the main display
7. User approves final list of media
8. The system records the final list and the user ID associated with the created list
9. The system updates the main display with the curated list of posts
10. The main display begins to display the new posts
11. The system notifies the user that the list is being displayed

**Alternative Flows:**

3A: User does not deem any posts need to be displayed

1. User decides to not provide a list based on the currently available media

2. The user exits the action of curating the displayed posts
3. The system discards any in work lists

3B: The user wishes to change a previously selected post

1. User selects post to not be displayed
2. The system updates the list of curated posts
3. Renter at beginning of step 3

7A: The user needs to change the order of curated list

1. The user selects the post they wish to change position
2. The user indicates which position the post should be displayed in
3. The system updates the curated list
4. The system re-presents the final list to user for approval
5. Renter at step 7 entry

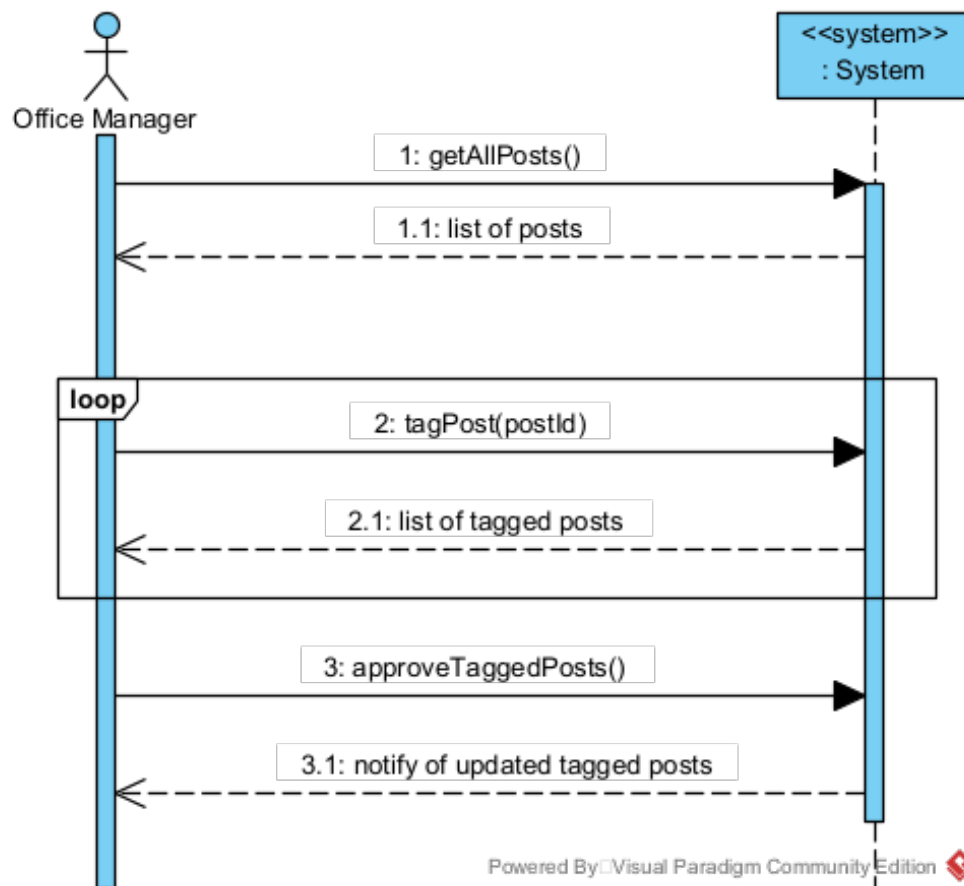


Figure 13: System Sequence Diagram: Manage Displayed Posts

**Operation:** getAllPosts()

**Cross-References:** UC15 (Manage Displayed Posts)

**Pre-conditions:**

- Posts exist in system
- Post count > 10

**Post-conditions:**

- All posts returned to user (create list of posts,  $lp$ )

**Operation:** tagPost(postId)

**Cross-References:** UC15 (Manage Displayed Posts)

**Pre-conditions:**

- Size of  $lp < 10$

Post-conditions:

- Post item,  $p$ , is added to  $lp$
- Size of  $lp \leq 10$

**Operation:** approveTaggedPosts()

**Cross-References:** UC15 (Manage Displayed Posts)

**Pre-conditions:**

- Size of  $lp = 10$
- $lp$  has not been approved

**Post-conditions:**

- $lp$  is approved
- MainDisplay updates with contents of  $lp$

#### 6.2.16 UC16: Manage Displayed Media

**ID:** UC16 (Manage Media For Display)

**Scope:** CS Automated Information Timeline

**Level:** User goal

**Primary Actor:** Office Manager

**Stakeholders and Interests:**

- Admin/Reviewer: Wants to ensure curated list of media items are on display for guests
- Guest: Wants to be able to see images and/or video on the TV display

- Faculty: Wants to ensure relevant media can be shown to guests.

**Preconditions:**

- User with Office Manager role has been authenticated.
- Media library is available.

**Postconditions:**

- Media is displayed on the TV display
- The ordering of the media displayed is correct

**Main Success Scenario:**

1. User navigates to media library
2. The system displays the media library and available files for display
3. User selects media to be displayed on the main display
4. The system displays the list of media that will be displayed on the main display
5. Repeat step 3 and 4 until all required media is selected for display
6. User reviews final list of media to be displayed
7. User approves media display and saves new list of media to be displayed
8. The system records the final list and the user identification associated with the created list
9. The system updates the main display with the list of media to display
10. The system provides local copies of media to the main display
11. The main display begins displaying the listed media
12. The system notifies the user that the listed media has been saved and is displayed on the main display

**Alternative Flows:**

3A: User does not find suitable media for display and has media to upload

1. User selects media they wish to upload from local device
2. User uploads new media to media library
3. The system confirms successful upload of media
4. The system adds newly uploaded media to list of media to display
5. Proceed to step 4 of the main scenario

3B: User wants to remove items from current Media list

1. User proceeds to modify currently returned media list
2. System returns the current list of media marked for display on the main display
3. User updates list to remove media
4. System returns updated list
5. Repeat step 3 and 4 until either all media is removed from the list or user cancels the action.
6. Proceed to step 6 of the main scenario

7A: User needs to modify order of media to be displayed

1. User selects option to change media ordering
2. The system notifies the user the current order of media will be lost
3. User confirms
4. The system returns user to step 3 of the main scenario

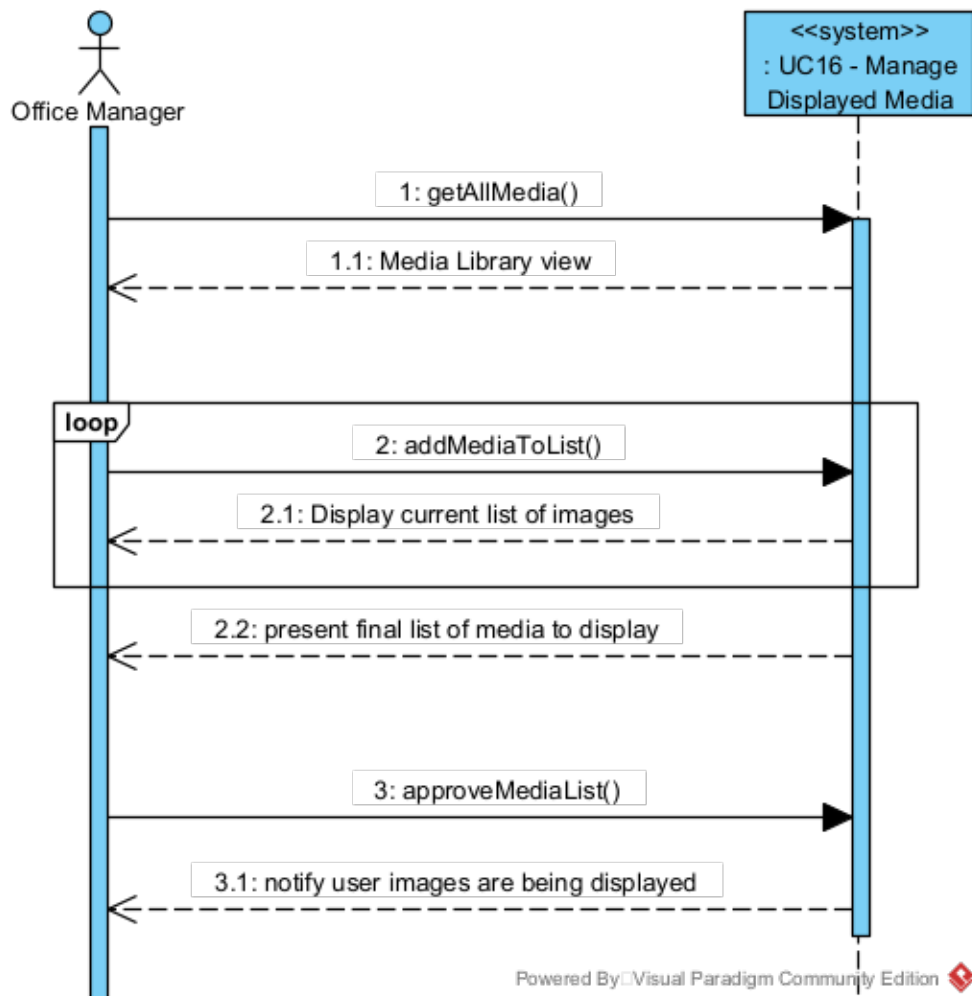


Figure 14: System Sequence Diagram: Manage Displayed Media

**Operation:** getAllMedia()

**Cross-References:** UC16 (Manage Media For Display)

**Pre-conditions:**

- Media objects exist in the media library (database store)
- Media objects have been approved by admin

**Post-conditions:**

- List of media returned to user as List<Media> object, *ml*

**Operation:** addMediaToList()

**Cross-References:** UC16 (Manage Media For Display)

**Pre-conditions:**

- Media List object, *ml*, is not at capacity yet
- *ml* has not been approved

**Post-conditions:**

- *ml* has association with media item, *m*, formed
- *ml* has been updated and requests approval

**Operation:** approveMediaList ()

**Cross-References:** UC16 (Manage Media For Display)

**Pre-conditions:**

- User has marked *ml* as ready for publish
- *ml* is not over capacity
- *ml* has not been approved yet

**Post-conditions:**

- *ml* is approved and set for display
- *MainDisplay* updates with new list, *ml*

## 7 Activity Diagram: Create Event

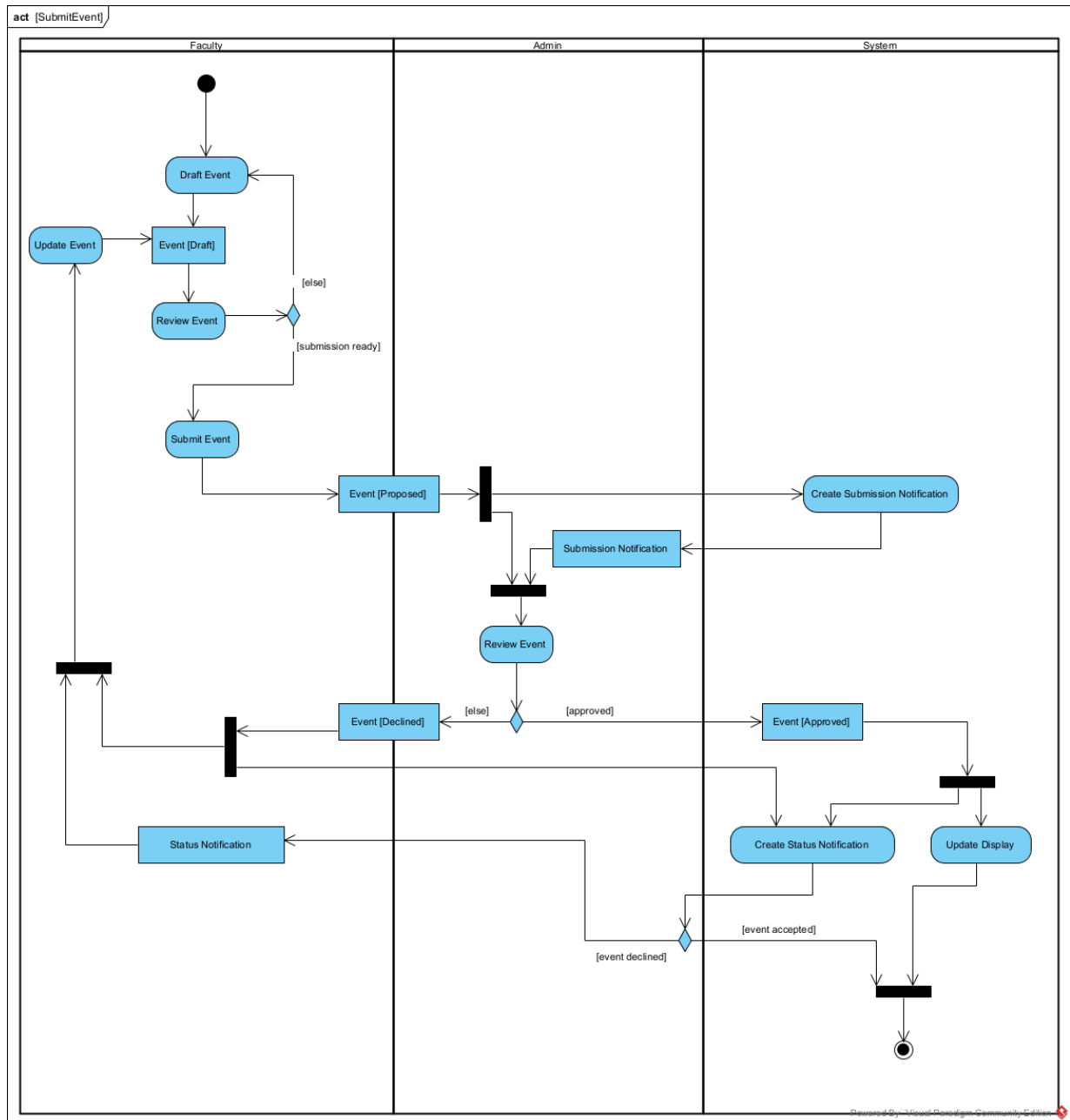


Figure 15: Activity Diagram: Create Event

## 8 Wireframes

### 8.1 Login Screen

The wireframe shows a login screen with a dark green header and a light gray footer. The header contains the 'BU LOGO' on the left, navigation links for 'Posts', 'Media', and 'Events' in the center, and a 'Login' button on the right. The main content area is a white box with the title 'Login'. Below the title are two input fields: 'Email' and 'Password'. A dark green 'Login' button is positioned below the password field, with a 'Reset password' link underneath it. The footer contains links for 'Product', 'Resources', 'About', and 'Support', along with the text 'CS DEPARTMENT BAYLOR UNIVERSITY' and a copyright notice '© 2010 - 2020 Privacy - Terms'.

BU LOGO

Posts Media Events

Login

Login

Email

Password

Login

Reset password

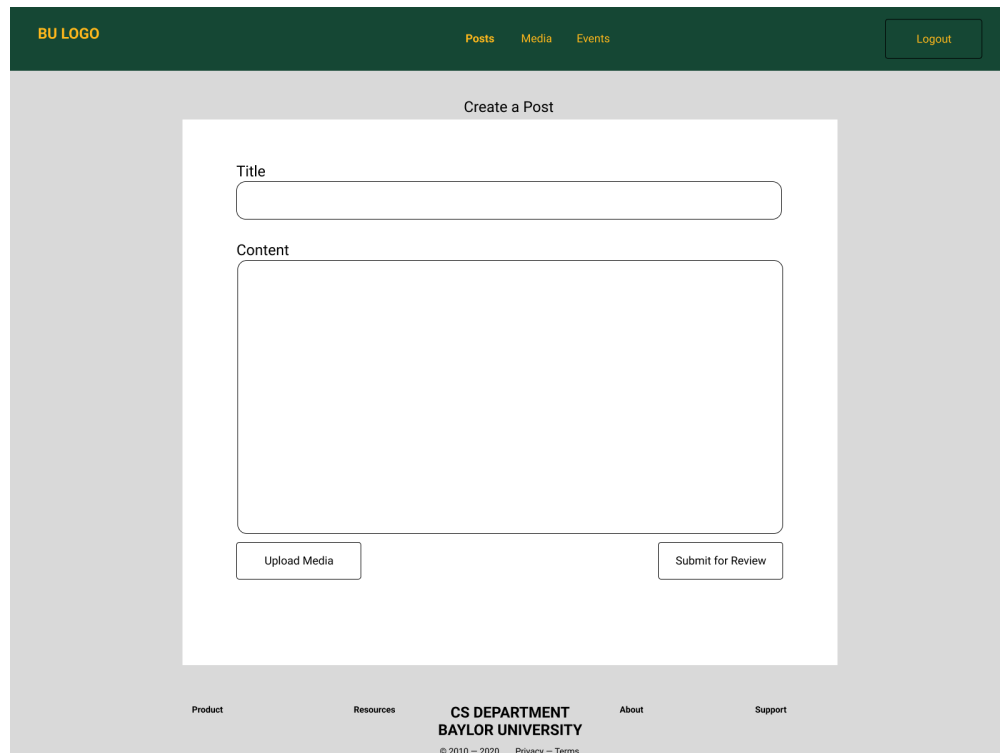
Product Resources CS DEPARTMENT BAYLOR UNIVERSITY About Support

© 2010 - 2020 Privacy - Terms

Figure 16: Wireframe: Login Screen



## 8.2 Create Post Screen



The wireframe illustrates the 'Create a Post' interface. It features a dark green header with the 'BU LOGO' on the left, navigation links for 'Posts', 'Media', and 'Events' in the center, and a 'Logout' button on the right. The main content area is titled 'Create a Post' and contains a form with a 'Title' input field and a larger 'Content' text area. Below the content area are two buttons: 'Upload Media' and 'Submit for Review'. The footer includes links for 'Product', 'Resources', 'About', and 'Support', along with the 'CS DEPARTMENT BAYLOR UNIVERSITY' logo and copyright information for 2010-2020.

BU LOGO

Posts Media Events Logout

Create a Post

Title

Content

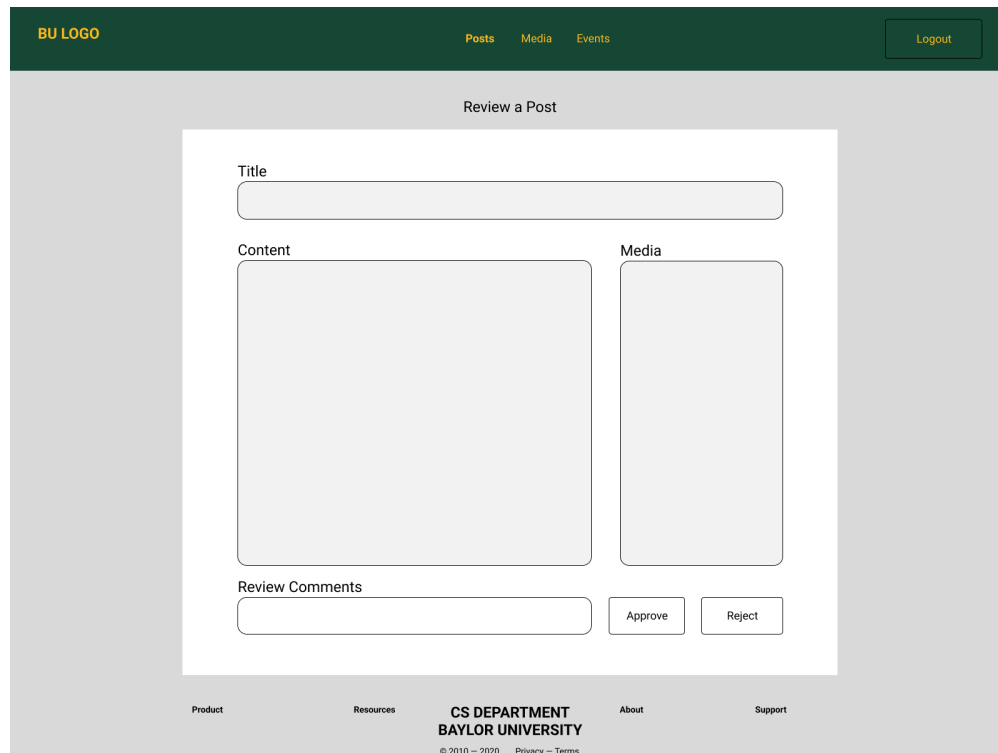
Upload Media Submit for Review

Product Resources CS DEPARTMENT BAYLOR UNIVERSITY About Support

© 2010 – 2020 Privacy – Terms

Figure 17: Wireframe: Create Post Screen

## 8.3 Review Post Screen



The wireframe illustrates the 'Review a Post' interface. It features a dark green header with the 'BU LOGO' on the left, navigation links for 'Posts', 'Media', and 'Events' in the center, and a 'Logout' button on the right. The main content area is titled 'Review a Post' and contains a form with the following elements: a 'Title' text input field; a 'Content' text area; a 'Media' image upload area; a 'Review Comments' text input field; and two buttons labeled 'Approve' and 'Reject'. The footer includes links for 'Product', 'Resources', 'About', and 'Support', along with the 'CS DEPARTMENT BAYLOR UNIVERSITY' logo and copyright information: '© 2010 – 2020 Privacy – Terms'.

Figure 18: Wireframe: Review Post Screen

## 8.4 Faculty Dashboard

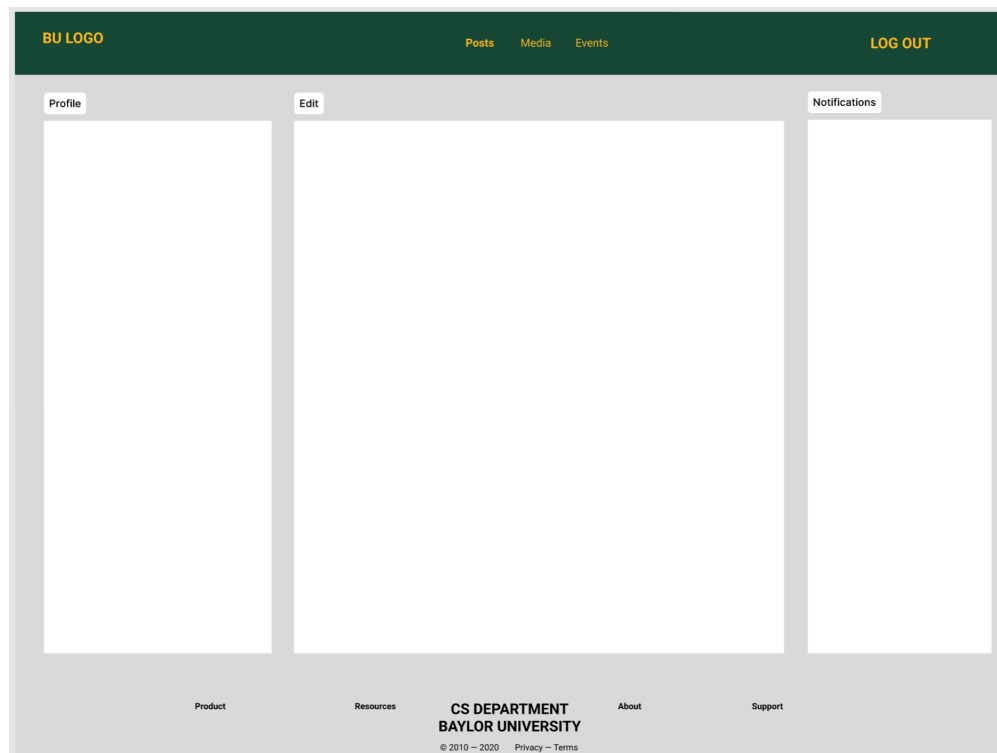


Figure 19: Wireframe: Faculty Dashboard

## 8.5 TV

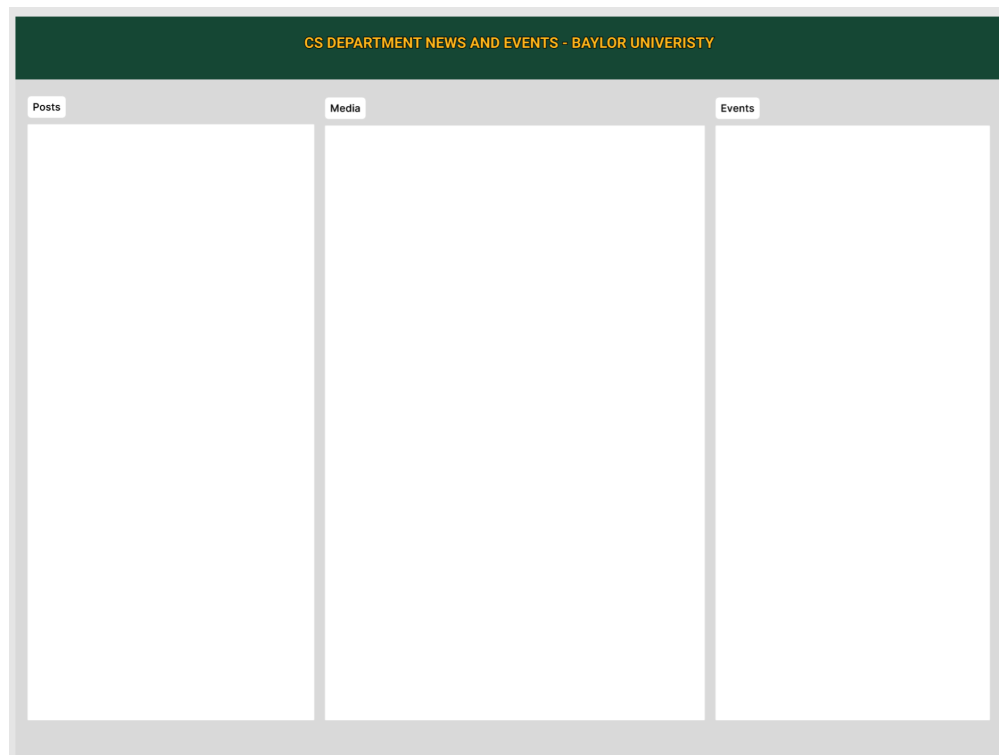


Figure 20: Wireframe: TV

## 8.6 Web UI

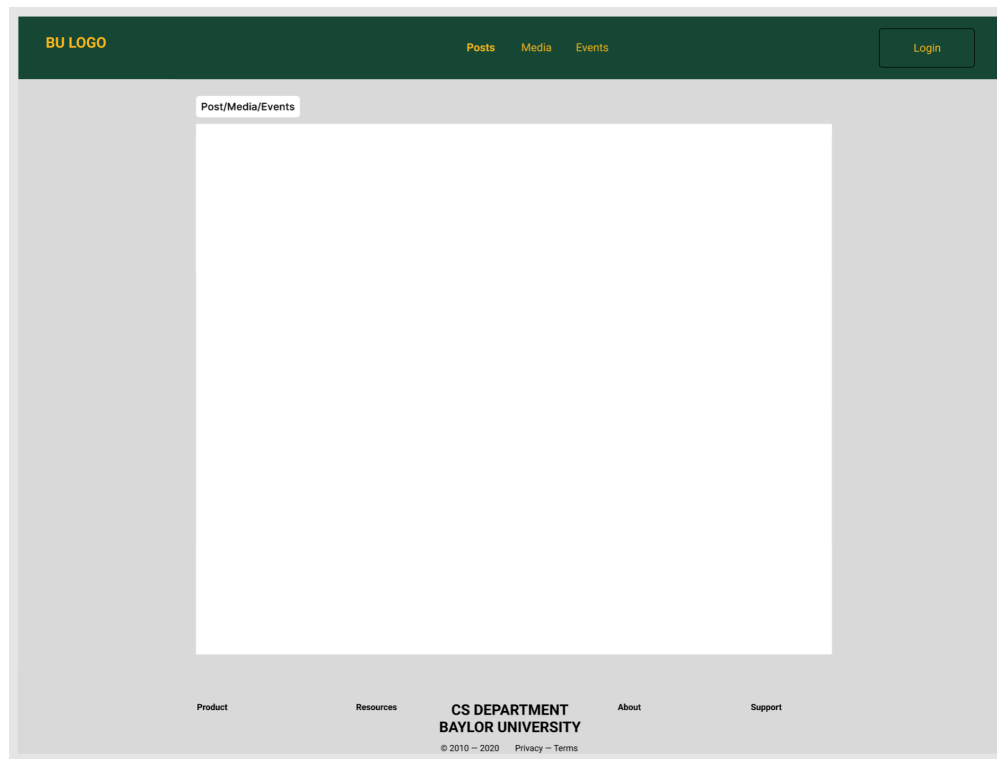
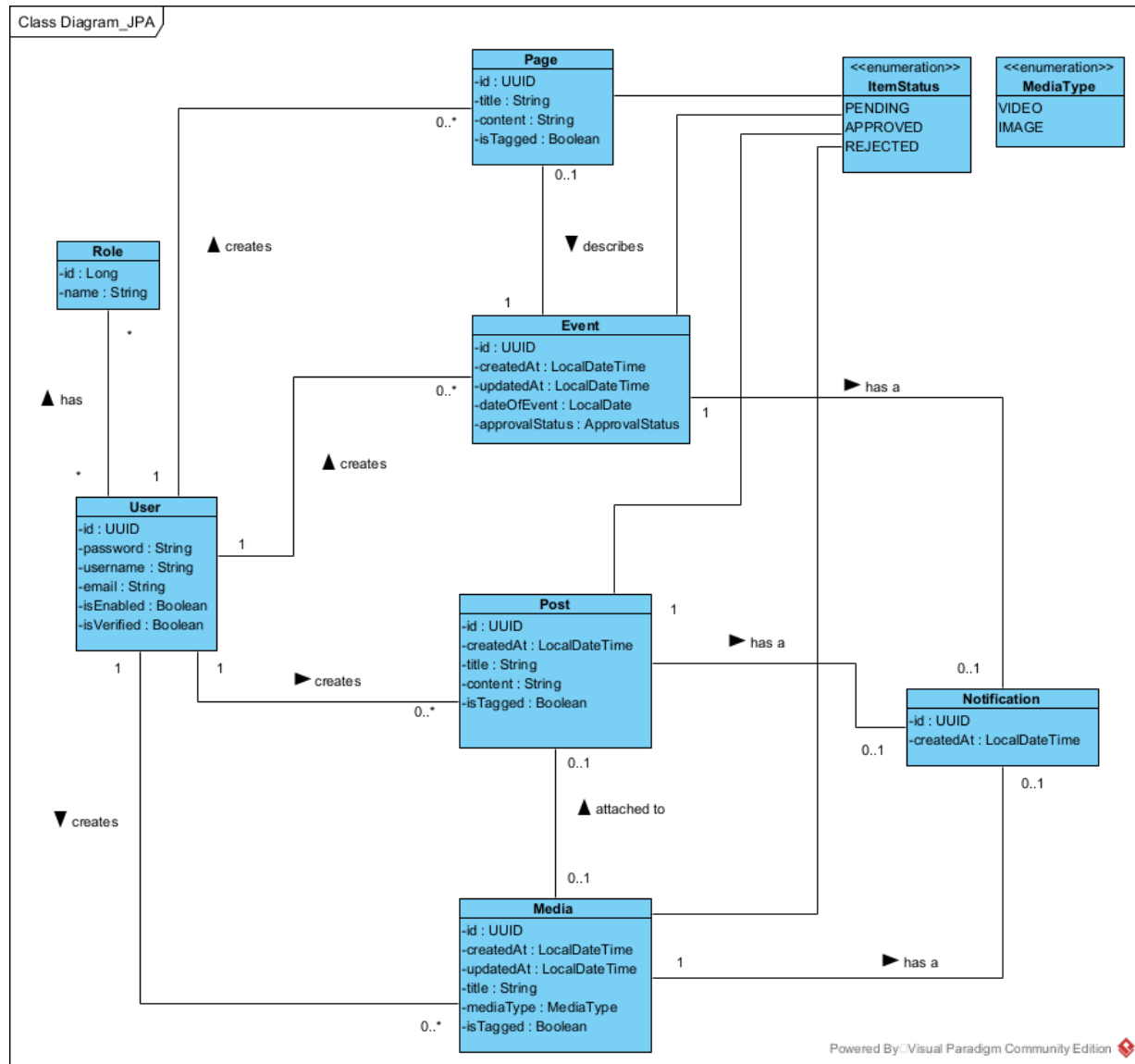


Figure 21: Wireframe: Web UI

## 9 Class Diagram



## 10 GRASP Patterns

Table below categorizes all the classes that appear on the sequence diagrams based on the GRASP concept they implement.

<b>Controller</b>	<b>Creator</b>	<b>Information Expert</b>	<b>Low Coupling/High Cohesion</b>
EventController	EventMapper	EventRepository	EventService
ReviewController	EventRepository	NotificationRepository	NotificationService
PostController	NotificationRepository	PostRepository	ReviewService
MediaController	NotificationMapper	MediaRepository	PostService
	PostRepository	PageRepository	MediaService
	MediaRepository		DisplayService
	PageRepository		PageService
			EventMapper
			NotificationMapper

Table 1: Classes with the identified GRASP concepts

Low coupling and high cohesion is achieved by clearly separating out Controller, Service and Repository actions so that each class is only responsible for its specific layer of the application. This is also a core part of the MVC pattern which is known to be loosely coupled and highly cohesive.

The mapper classes (EventMapper and NotificationMapper) were also identified as a creators because they are meant to be injected in service classes to create objects to be used as entities. Mapper classes support low coupling by removing a need for mapping user inputs directly to domain object needs and high cohesion by dealing only with JSON Serialization/deserialization of Event/EventDTO objects.

## 10.1 Annotated Sequence diagrams

A subset of sequence diagrams are annotated with the identified GRASP concepts and are included below. These sequence diagrams cover all the classes and the GRASP patterns they implement.

## 10.1.1 Sequence Diagram: Create Event

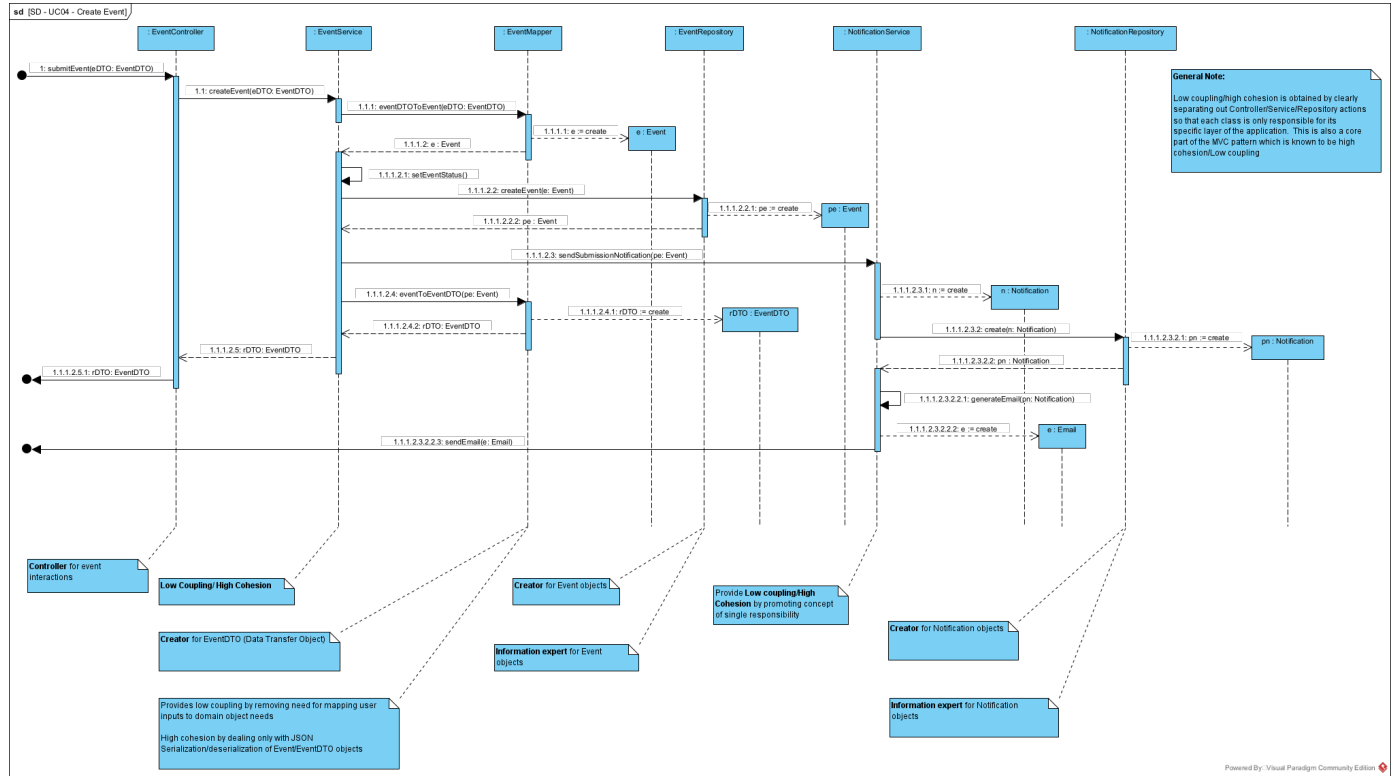


Figure 23: Sequence Diagram: Create Event



## 10.1.2 Sequence Diagram: Review for Approval

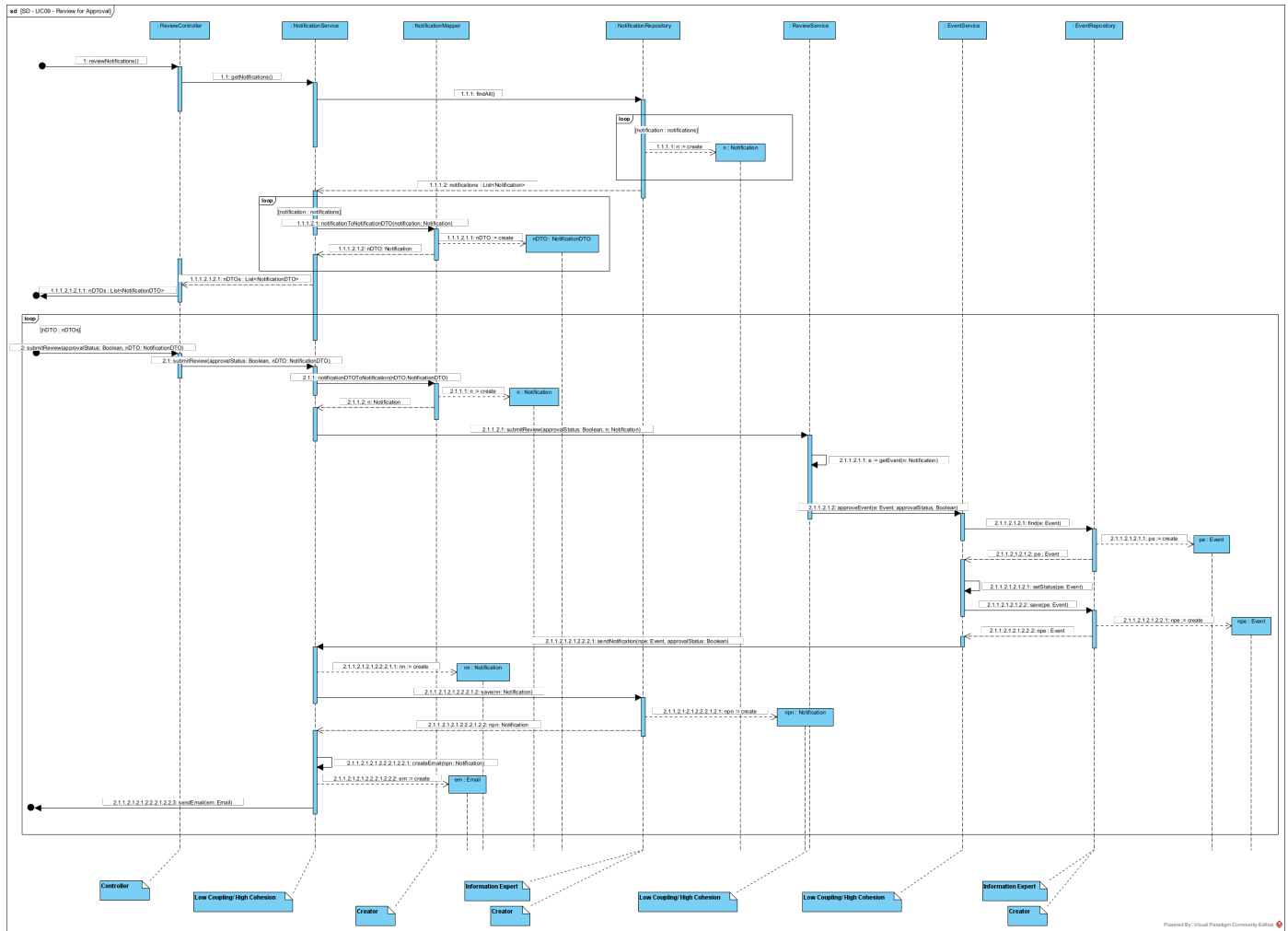


Figure 24: Sequence Diagram: Review for Approval

### 10.1.3 Sequence Diagram: Edit Post

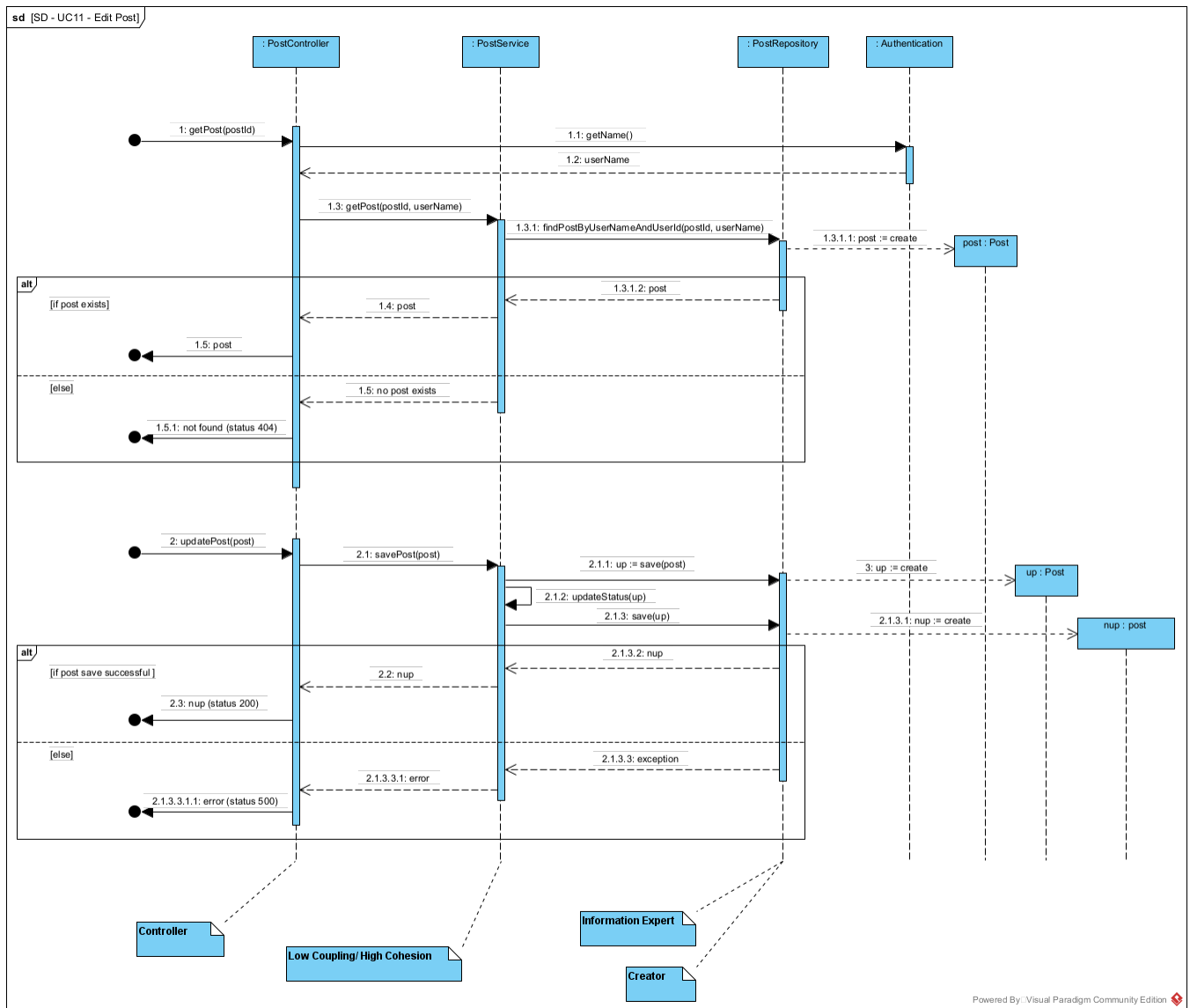


Figure 25: Sequence Diagram: Edit Post

### 10.1.4 Sequence Diagram: Manage Displayed Media

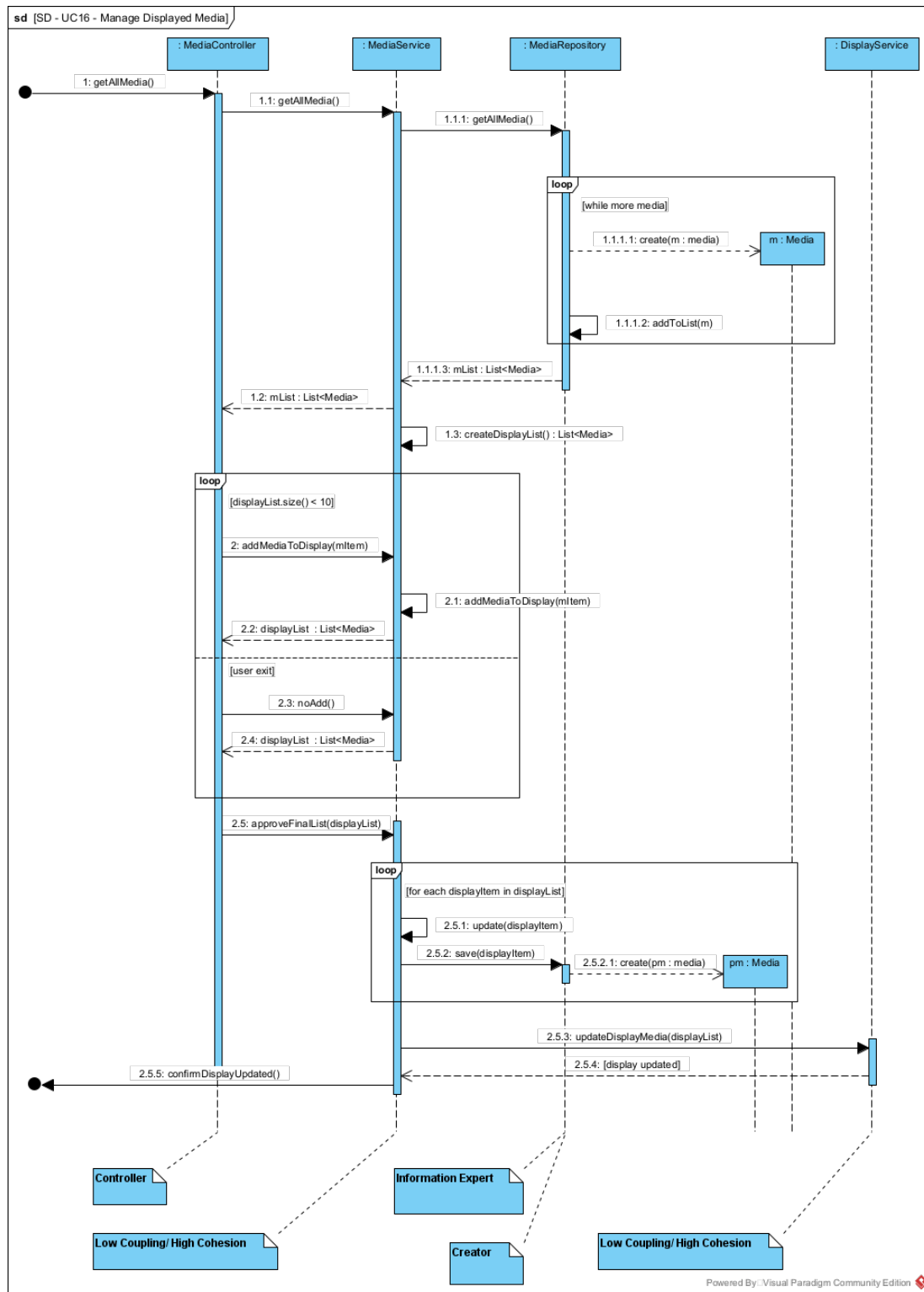


Figure 26: Sequence Diagram: Manage Displayed Media

### 10.1.5 Sequence Diagram: Add Page to Event

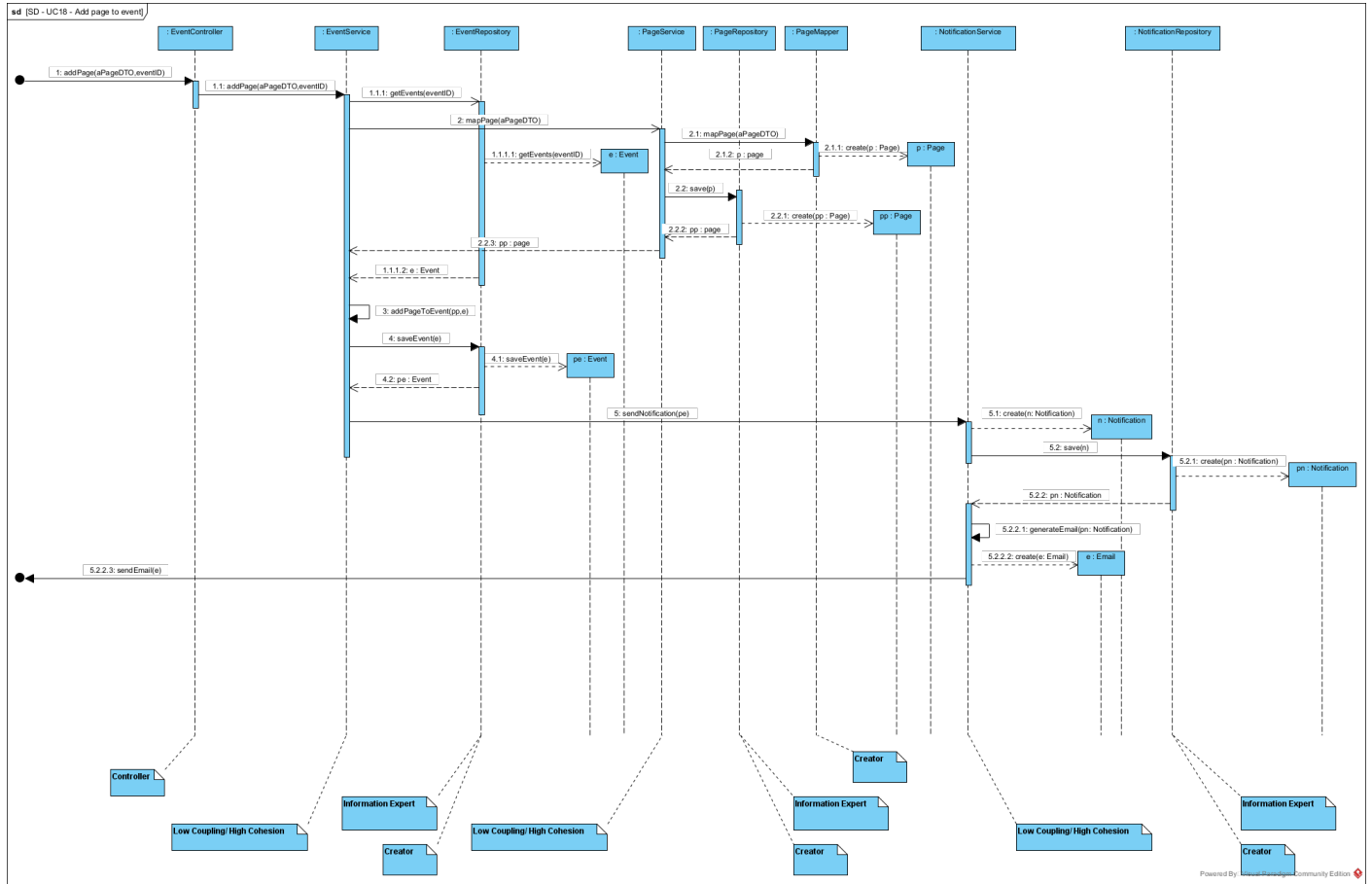


Figure 27: Sequence Diagram: Add page to Event

## 11 Object Constraint Language

Below are the three most challenging constraints in the design of the CS Department Automated Information Timeline project which are described in OCL.

### 11.1 Describing Constraints with OCL

#### 11.1.1 Constraint 1: Staging Approved Posts

The Office Manager can only stage approved posts.

```
{context: Post::tagPost():Post
inv: self.approvalStatus = ItemStatus.APPROVED}
```

#### 11.1.2 Constraint 2: Displaying Posts

The main display cannot display more than 10 posts, and all displayed posts must be approved.

```

{context: PostController::displayPosts(): Set<Post>
inv: let posts = self.getApprovedPosts(): Set<Posts>
inv: posts→size() ≤ 10
inv: posts→forEach(p — p.approvalStatus = ItemStatus.APPROVED)}

```

### 11.1.3 Constraint 3: Notifications

A notification can only be associated with one post, media, or event.

```

{context: Notification
inv: (self.post <> null xor self.media <> null) xor (self.event <> null)}

```

## 12 State Machine Diagram

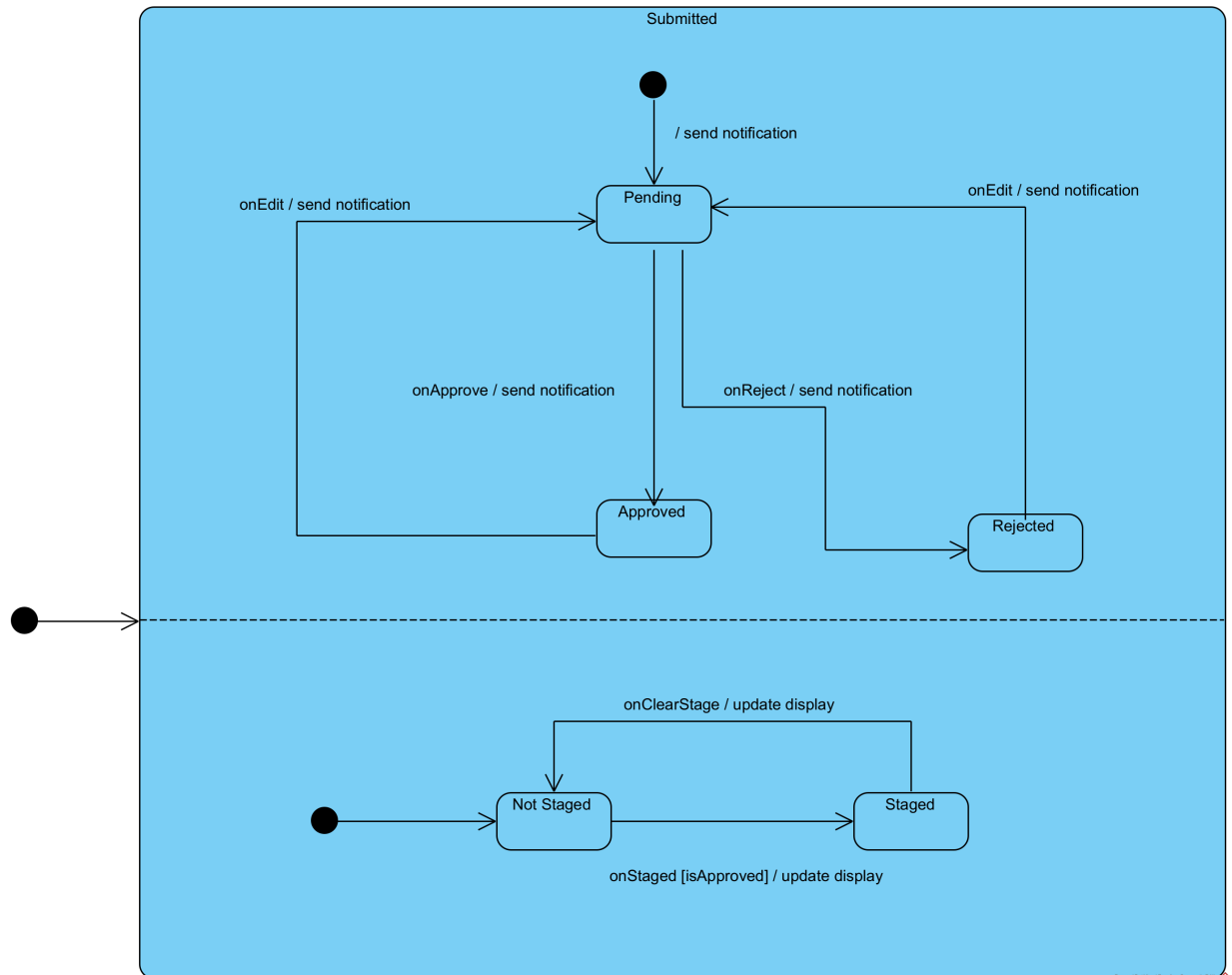


Figure 28: State Machine Diagram for review and staging of submitted posts, events and pages.

## 13 Package Diagram

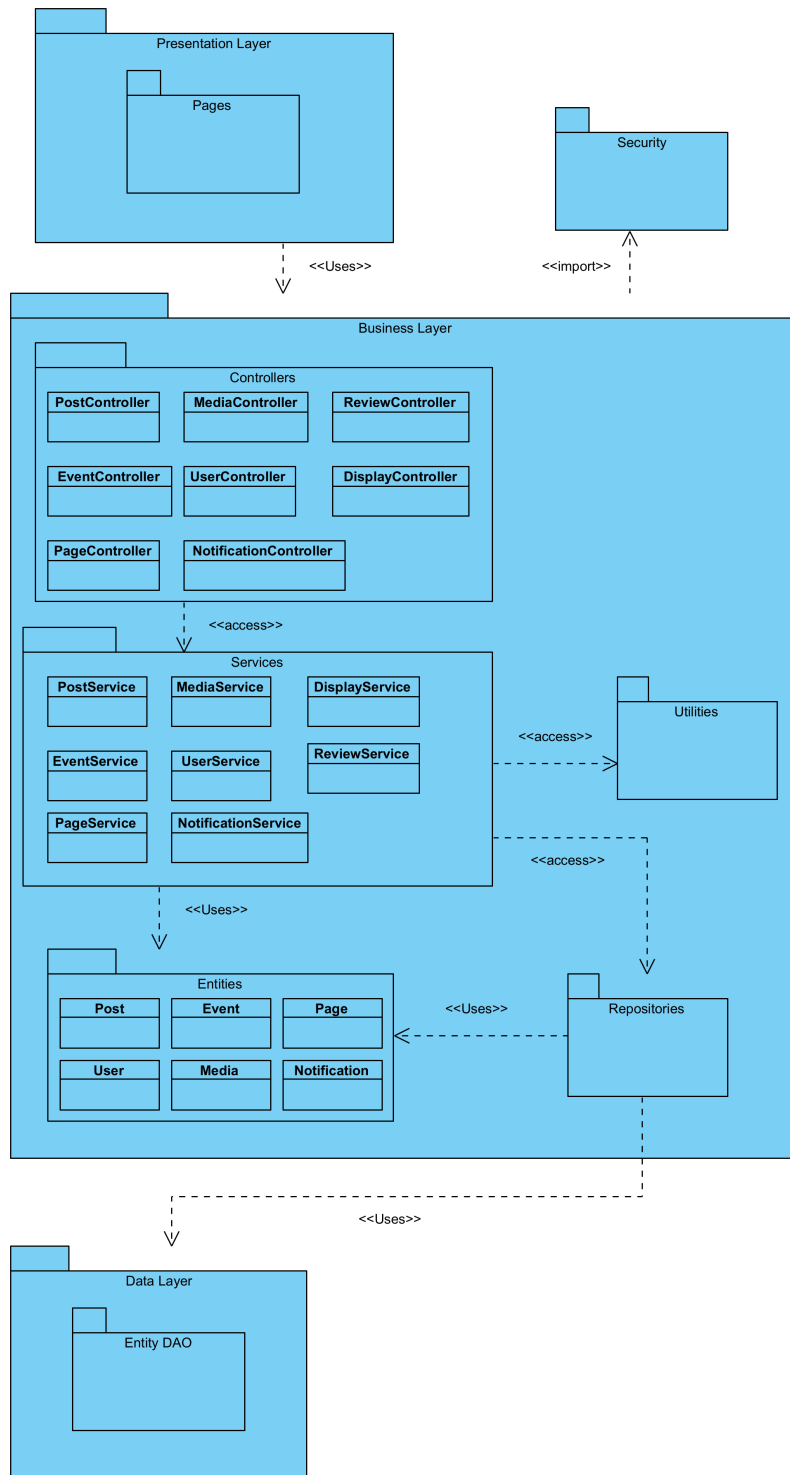


Figure 29: Package Diagram

## 14 Deployment Diagram

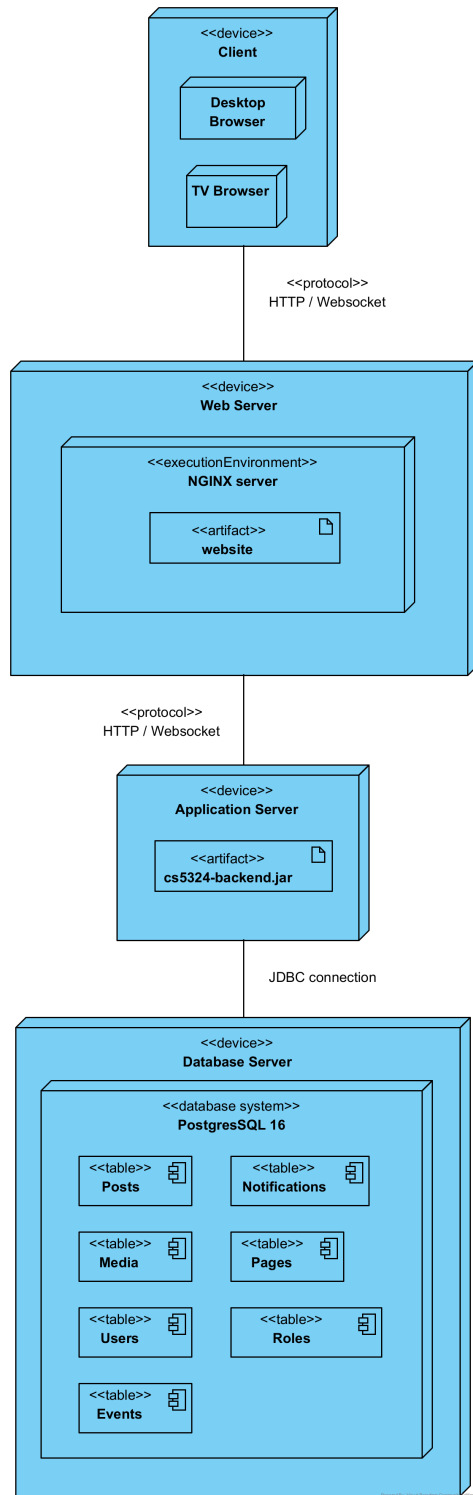


Figure 30: Deployment Diagram

## 15 Component Diagram

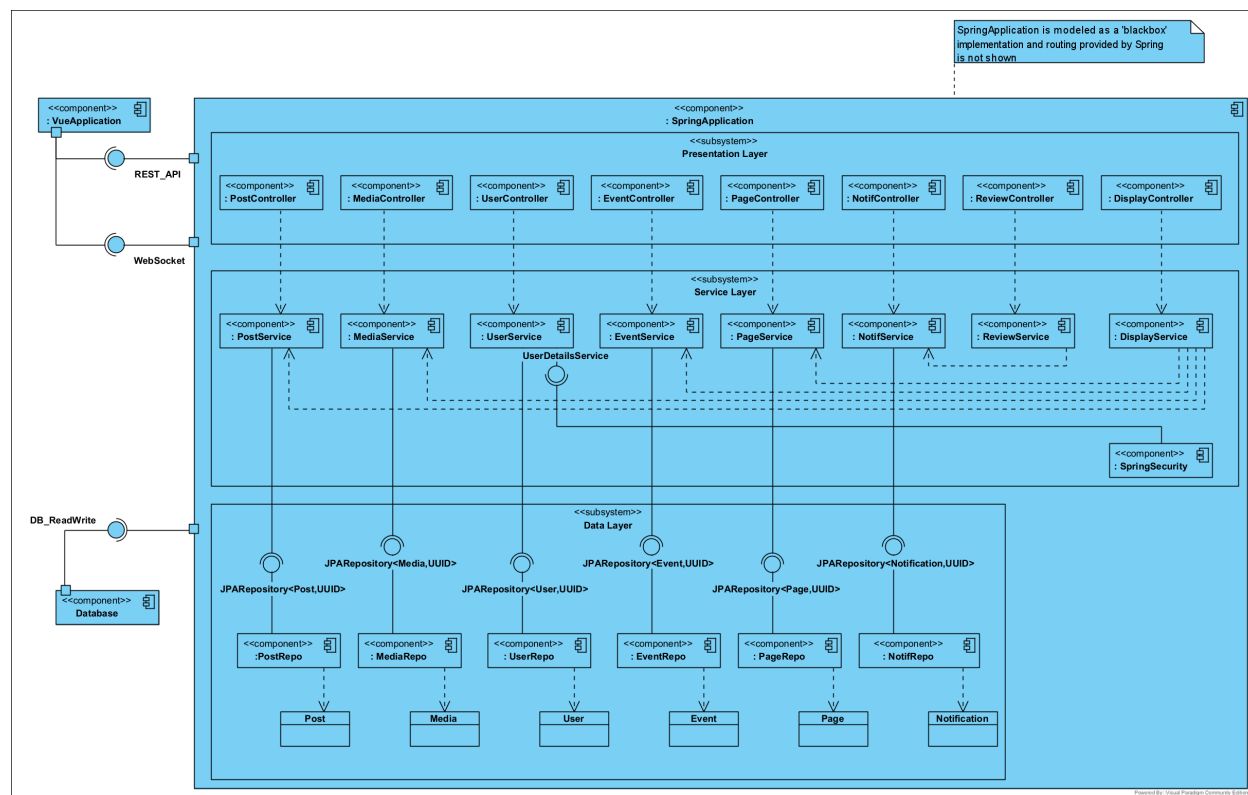


Figure 31: Component Diagram

## 16 Code

For the initial iteration submission, we have provided an implemented domain using the Spring JPA framework. The following items were considered in this iteration. The code base will continue to evolve over subsequent iterations.

- Version allows for optimistic locking during database transactional contexts to ensure that a stale persistence does not occur.
- Created at and updated at values use Spring auditing to add the appropriate system time to our entities upon submission.
- Enumerations were provided as separate classes to promote reusability and extensibility within the code base.
- Care was given in assigning relationship owning entities to allow for succinct operations in persistence and modification.
- At this time, no cascading operation have been determined. This is to be provided later as realization of these operations will become more available while development continues.
- Special consideration was given to the Notification entity to ensure that it may have one and only one relationship with the following differing entity types: Post, Event, Media.



- Custom user implementation of UserDetails and UserDetailsService (from Spring security) was used to create the base user class. This was chosen since it will provide roles tied to logged in users and can be fetched to determine which options are available to a logged in user based on the granted authorities for that role.

The source code with JPA implementation is provided in a zip folder along with this document. The github repository this source code is hosted at is linked [here](#).