UNIVERSITY OF WEST LONDON
The Career University

Applied Software Engineering

Coursework

Assignment- 1

Aman Adish
21586075
BSc Computer Science

Table of Contents

Task 1: Use Case Model

Use Case: Advertisement Approval

Use Case Description: The Marketing Staff forwards an advertisement with all the details to the Editor for review. This makes sure the advertisement is ready for publication.

Primary Actor: Marketing Staff

Secondary Actor: Editor

Preconditions:

- The Marketing Staff should be logged into the system.
- The advertisement details (advertiser, size, placement, etc.) must be recorded.

Triggers: The Marketing Staff selects an advertisement and send it to Editor.

Postconditions:

- The advertisement is successfully added to the Editor's queue for review.
- A confirmation is provided to the Marketing Staff.

Basic Flow:

- Marketing Staff logs into the system.
- The system displays the list of recorded advertisements.
- Marketing Staff selects an advertisement to forward to the Editor and verifies all the details.
- The system confirms the Editor's queue availability.
- The system forwards the advertisement to the Editor.
- The system notifies the Marketing Staff that the submission was successful.
- The system can now track the status of advertisement.

Alternative Flow:

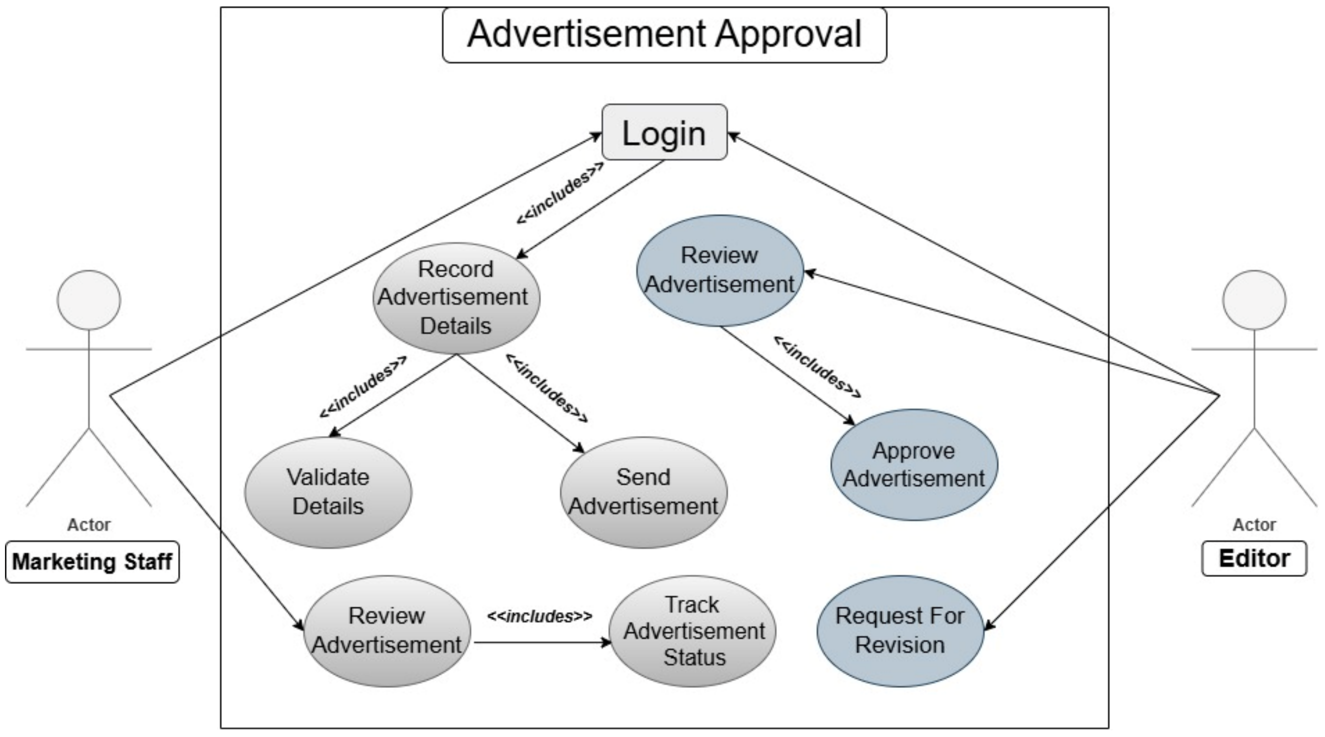
Advertisement Rejected by Editor:

- The Editor rejects the advertisement.
- The system notifies the Marketing Department of the rejection.
- The Marketing Department can recheck the advertisement and submit again.

Duplicate Advertisement Submission

- If the advertisement has already been submitted, the system notifies the Marketing Staff about the duplicate ad.
- Marketing Staff reviews and cancel the submission.

Use Case Model



Task 2: Class Diagram

Based on the "Advertisement Approval" use case, we can identify the following key classes:

Marketing Staff

Attributes

- id
- name
- departmentID

Methods

- recordAdDetails()
- sendAdToEditor()
- trackAdStatus()
- processAd()

Editor

Attributes

- editorID
- name
- departmentID

Methods

- reviewAd()
- approveAd()
- requestAdRevision()
- rejectAd()
- processAd()

Advertisement

Attributes

- advertisementID
- advertiserDetails
- size
- dateOfPublication
- placement
- status

Methods

- updateContent()
- setStatus()

<u>System Class</u>

Methods

- loginUser()
- recordAdDetails()
- getAdvertisementStatus()
- notifyUsers()

<u>Staff Member</u>

Attributes

- id
- name
- departmentID

Methods

- processAd()

The design promotes ***<u>modularity</u>*** by separating **different types of users** (such as Editor and Marketing Staff) and their respective responsibilities. Also, ***<u>reusability</u>*** is very evident in Advertisement Class as they hav operations related to advertisements in a single place so that the Staff can reuse these methods **without needing to redefine them**.

Object Oriented Principles (OOP) are effectively used in this Class Diagram

- **Inheritance**: Editor and Marketing Staff (Child Classes) inheriting attributes (id, name, departmentID) from Staff Member (Parent Class)

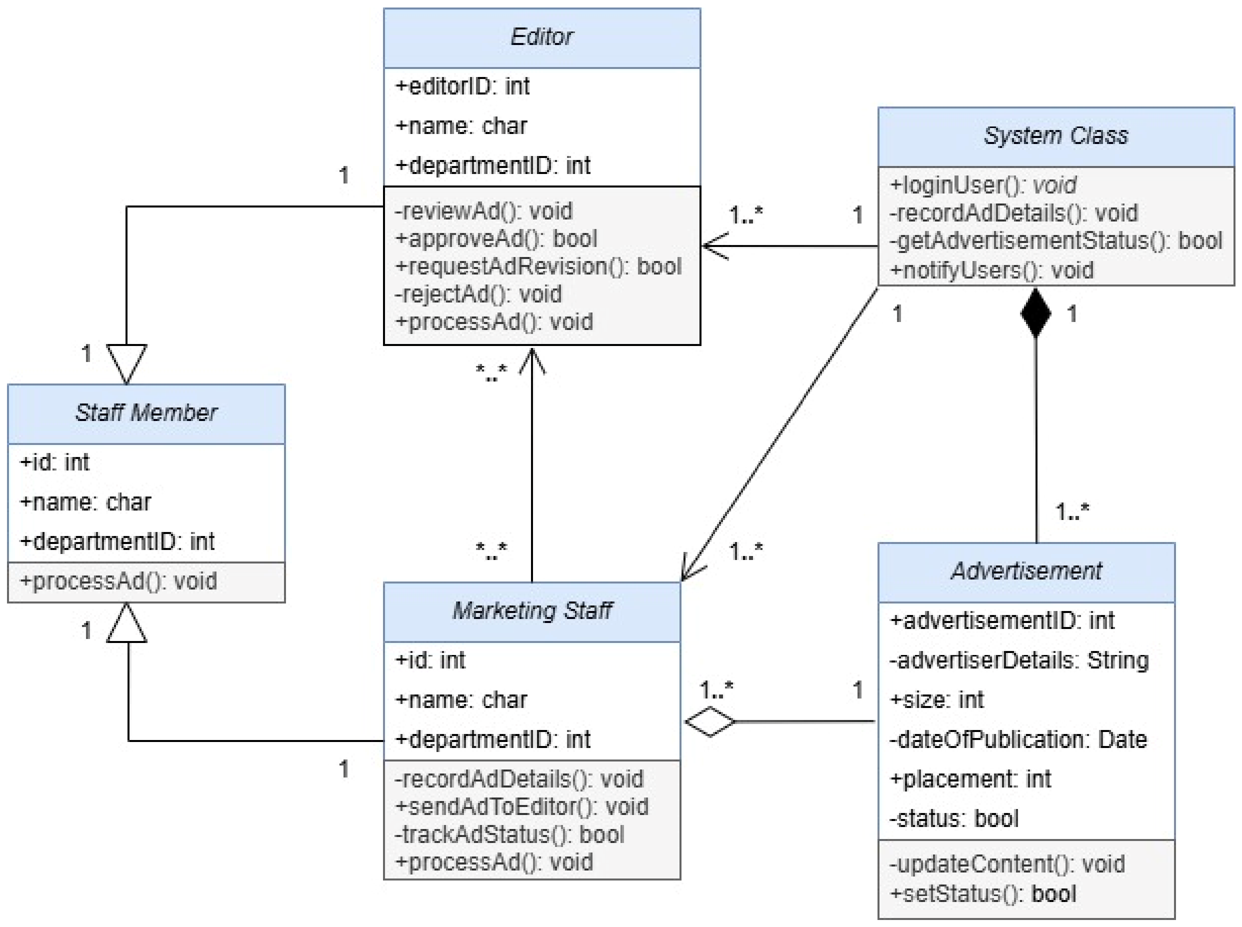
- **Aggregation**: The Marketing Staff (aggregator class) has an aggregation relationship with the Advertisement. This signifies that advertisements are associated with marketing staff but do not depend on them for their existence. If Marketing Staff is removed, Advertisement can still exist.

- **Composition**: System Class has composition relation with Advertisement, meaning, The Advertisement objects are owned by the System Class, meaning they cannot exist independently of it. If the System Class is destroyed, all associated Advertisement objects are also destroyed.

- **Encapsulation**: Encapsulation is used by keeping attributes private, such as updateContent() in Advertisement and trackAdStatus() in Marketing Staff. This limits outiside access to the attributes and provides controlled access through methods, which protects the data integrity.

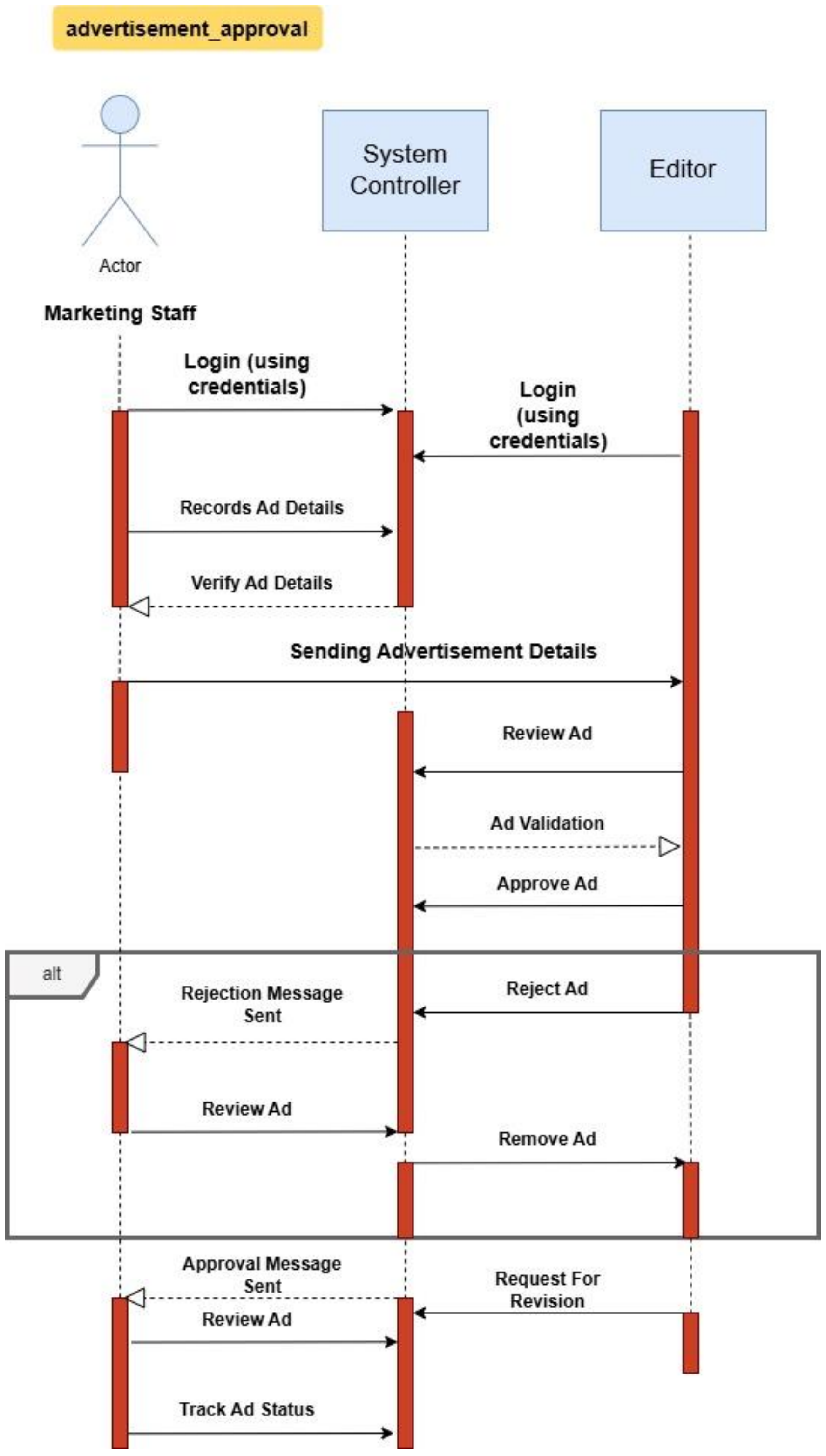
- **Polymorphism**: The Staff Member defines a general method, processAd() and both Editor and Marketing Staff override processAd() with their specific implementations, This allows treating all Staff Member's equally and yet allowing different behaviors.

Class Diagram



Task 3: Sequence Diagram

Sequence Diagram for “Advertisement Approval” Use Case



Explanation

1. The **Marketing Staff logs** into the System and **records** the advertisement details.
2. The System **verifies the login credentials** and validates the advertisement details.
3. If the details are valid, the System checks the **Editor's queue availability**.
4. If the Editor's queue is available, the **System forwards the advertisement** to the Editor.
5. The Editor **reviews the advertisement** and either approves or rejects it.
6. If **approved**, the process is completed.
7. The System updates the advertisement **status to "Approved"** and notifies the Marketing Staff of the approval.
8. If **rejected**, the Editor provides feedback and sends a rejection notification to the System.
9. The System forwards the **rejection notification** to the Marketing Staff.
10. The Marketing Staff **revises the advertisement** based on the feedback and resubmits it to the System.

Guards, Iterations, and Messages Operation

Guards: Applied at decision points ([Valid Credentials], [Queue Available], [Decision = Approve/Reject]) to determine the next step.

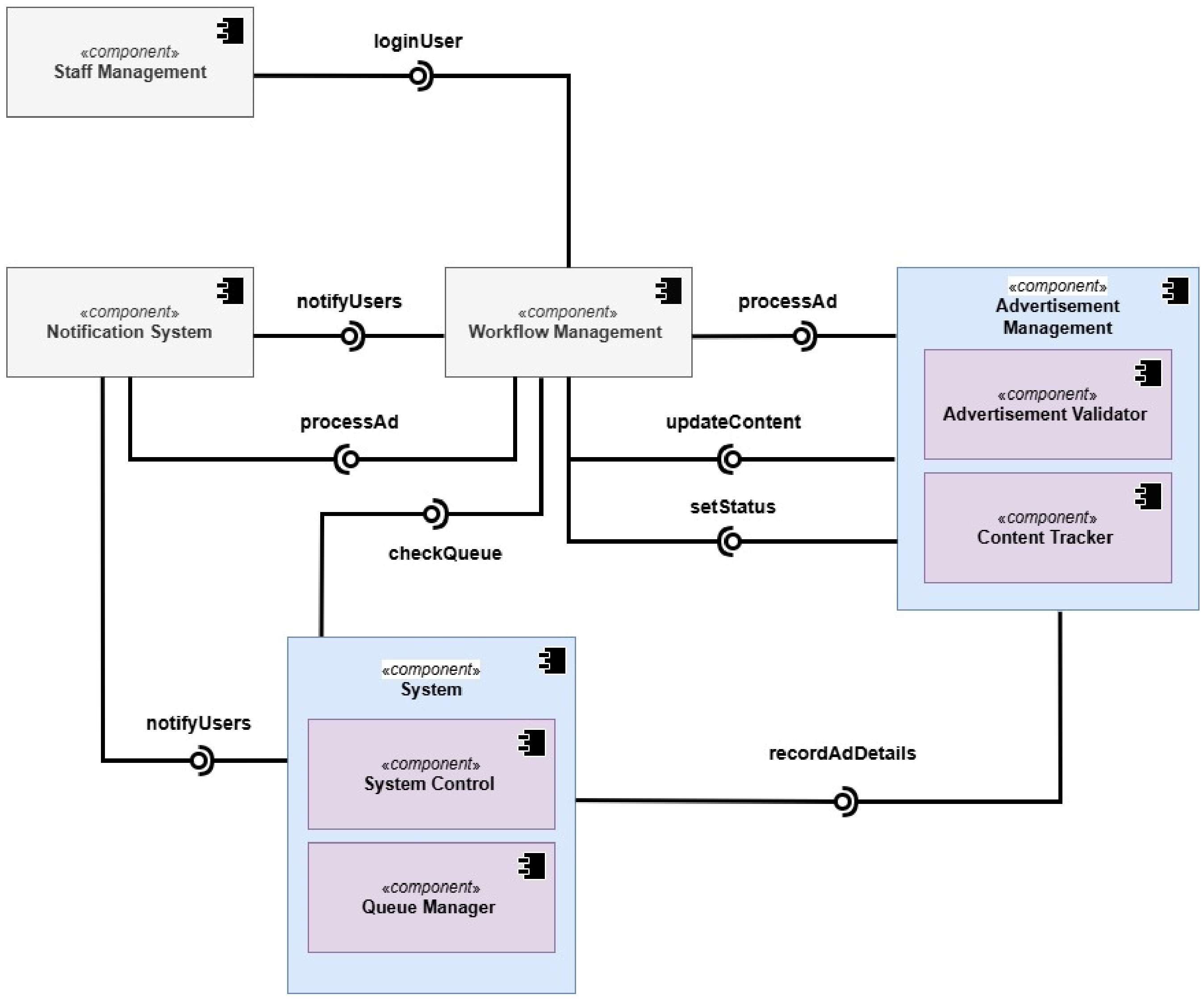
Iterations: Occur in the "Revise and Resubmit" cycle, repeating Steps 3–5 until the advertisement is approved.

Message Operations: Clearly specify the interactions (e.g., login request, validation messages, notification updates).

For Example: At any point, the Marketing Staff can send a status request to the System, which retrieves and displays the current advertisement status (e.g., "Pending," "Approved," or "Rejected").

Task 4: Component Model

Component Model Diagram


«component» Staff Management
loginUser
«component» Notification System
notifyUsers
«component» Workflow Management
processAd
«component» Advertisement Management
«component» Advertisement Validator
«component» Content Tracker
processAd
updateContent
setStatus
checkQueue
«component» System
«component» System Control
«component» Queue Manager
notifyUsers
recordAdDetails

Component

System Component

Purpose

The **System Component** serves as the central orchestrator that coordinates and manages interactions between all other components in the Advertisement Approval System. It ensures that workflows, validations, and notifications occur seamlessly and according to business rules.

Responsibilities

- Workflow Orchestration
- Validation of Advertisement Submissions
- Queue Management
- Feedback Management
- Error Handling and Monitoring

<u>Interface Methods</u>:

- **loginUser()**

Allows staff members(Editor, Marketing) to log in and authenticate their access to the system.

- **recordAdDetails()**

Stores detailed information about advertisements for tracking and auditing purposes.

- **getAdvertisementStatus()**

Retrieves the current status of a specific advertisement within the system. It provides real-time updates to authorized users about whether an advertisement is under review, approved, rejected, or awaiting resubmission.

- **notifyUsers()**

Generate and send notifications to relevant staffs about updates or status changes.

<u>**Component Composition**</u>

The component composition used in System Component is **<u>Hierarchical Composition</u>**.

Top-level: System Component

Sub-components:

<u>*System Control*</u>

Oversees the overall operations, including managing interactions between other components such as the Queue Manager and external interfaces.

<u>*Queue Manager*</u>

Operates under the System Control and specifically focuses on queue-related tasks, such as monitoring pending advertisements and scheduling processing tasks.

System Control acts as the system manager, while Queue Manager is a specialized unit for queue handling, ***<u>making it a clear hierarchy with functional delegation</u>***.

The System Component is composed of the following **internal submodules or functional units**:

Validation Module

- Ensures the advertisement submissions meet predefined standards and rules.
- Verifies data integrity and correctness before passing it to other components.

Queue Management Module

- Coordinates interactions between components like User Management, Advertisement Management, Workflow Management, and Notification System.
- Ensures seamless execution of tasks across the system.

Feedback Processing Module

- Handles feedback from Editors for rejected advertisements.

- Sends processed feedback to Marketing Staff for revision and resubmission.

Error Logging and Monitoring Module

- Monitors all operations within the system.
- Logs errors and exceptions to maintain system reliability and aid in troubleshooting.

Task 5: Object Constraint Language

OCL Code for Business Rules:

- **Marketing Staff must be verified (logged in) before performing any operations** (Precondition)

context System::sendAdvertisementToEditor(advertisement : Advertisement)

pre:

self.currentUser.isLoggedIn() and

advertisement.isValid() and

not self.editorQueue->includes(advertisement)

- **Duplicate advertisements should not be forwarded to the Editor's queue.**

context System

inv noDuplicateSubmissions:

self.editorQueue.advertisements->forAll(a | a <> advertisement)

- **The Editor's queue must have capacity before accepting new advertisements.**

context EditorQueue

inv queueCapacity: self.advertisements->size() < self.maxCapacity

OCL Code for Constraints

- **Each advertisement must have a valid Advertiser Name.** (Precondition)

context Advertisement

inv validAdvertiserName: self.advertiserName <> null and self.advertiserName.size() > 0

- **The Size of the advertisement must be greater than 0.** (Precondition)

context Advertisement

inv positiveSize: self.size > 0

- **The Placement of the advertisement must not be null or undefined.** (Precondition)

context Advertisement

inv validPlacement: self.placement <> null

- **The advertisement must have Payment Terms defined.** (Precondition)

context Advertisement

inv validPaymentTerms: self.paymentTerms <> null

Postconditions

- **The advertisement is successfully added to the Editor's queue for review.**

post:

-- Advertisement is added to the Editor's queue

self.editorQueue->includes(advertisement) and

advertisement.status = 'submitted' and

- **A confirmation is provided to the Marketing Staff**

post:

-- Marketing Staff receives confirmation

self.notifications->includes('Advertisement has been forwarded to the Editor for review')

Multiplicities

--MarketingStaff - Editor (Many-to-Many)

context MarketingStaff

inv manyToManyEditors: self.editors->forAll(e | e.marketingStaffs->includes(self))

--StaffMember – Editor (One-to-One)

context StaffMember

inv oneToOneEditor: self.editor->size() = 1

--StaffMember – Marketing Staff (One-to-One)

context StaffMember

inv oneToOneMarketingStaff: self.marketingStaff->size() = 1

--System - Editor (One-to-Many)

context System

inv oneToManyEditors: self.editors->size() >= 1

--Advertisement - Marketing Staff (One-to-Many)

context Advertisement

inv oneToOneMarketingStaff: self.marketingStaff->size() = 1

--Advertisement - System (Many-to-One)

context Advertisement

inv manyToOneSystem: self.system->size() = 1

--System - Marketing Staff (One-to-Many)

context System

inv oneToManyMarketingStaffs: self.marketingStaffs->forAll(ms | ms.system = self)

Conclusion

The **Advertisement Approval** system is designed to streamline and ensure the accuracy of the advertisement review process by integrating core **Object-Oriented Principles (OOP)**. Through its modular design, the system separates responsibilities across distinct components and classes, promoting clarity and scalability. **Inheritance** is effectively utilized as well as **Aggregation** and **Composition** between components enhancing data management.

The **sequence** and **component diagrams** illustrate the workflow's logical flow and interaction among submodules. **Constraints** defined using **OCL** ensure business rules are enforced, such as preventing duplicate submissions and maintaining valid attributes. These safeguards enhance the reliability and efficiency of operations.

By aligning design with real-world business requirements, the system provides robust capabilities, including status tracking, error logging, and dynamic interactions between actors. The inclusion of functional delegation, error monitoring, and feedback processing further demonstrates the system's scalability and reliability. Overall, this design creates an efficient, user-friendly framework for managing the advertisement approval process while maintaining data integrity and operational precision.

On a personal note, **the slides provided by our instructor** were immensely helpful in understanding the theoretical and practical aspects. They served as an excellent guide for identifying the relationships and constraints essential to this system. Integrating those lessons into this design made the process more intuitive and enjoyable. This project not only strengthened my technical skills but also deepened my appreciation for how academic resources can bridge classroom learning with real-world applications.

References

GeeksforGeeks. (2020). Component Based Diagram. [online] Available at: https://www.geeksforgeeks.org/component-based-diagram/.

Hindi, in (2020). Component Diagram Explained in Hindi l UML Diagram l Software Modeling and Designing Course. [online] YouTube. Available at: https://youtu.be/L0CZ5wM-DVI?si=k2BUb33CZ0VzxwKp [Accessed 19 Nov. 2024].

www.javatpoint.com. (n.d.). UML Component Diagram - Javatpoint. [online] Available at: https://www.javatpoint.com/uml-component-diagram.

Youtu.be. (2024). UML - Component diagram introduction - YouTube. [online] Available at: https://youtu.be/VIqJ4rdKUEY?si=VrCMe93RE-YErbRE [Accessed 19 Nov. 2024].

Visual-paradigm.com. (2019). What is Component Diagram? [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/.

Blackboard Slides for Applied Software Engineering