CONTENT MANAGEMENT SYSTEM

Software Construction And Development

BSSE 5TH SEMESTER

REPORT (Assignment 02)

Course Instructor: Sir FAROOQ IQBAL

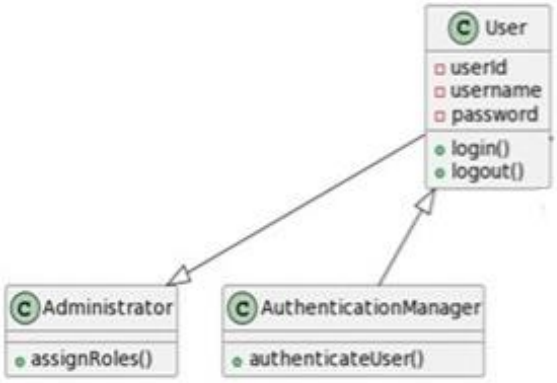
Group Members:

1. **Syeda Anshrah Gillani (1337-2021) - (Group Leader)**
2. **Umema Mujeeb (2396-2021)**
3. **Maheen Ali (1589-2021)**
4. **Areej Asif (2276-2021)**

18/12/2023

Classes into Data, Routines & Internal Routine Design:

(1) UserManagement Subsystem Class



Data:

1. **User Class*:***

   Attributes: UserID, UserName, Email

   Methods: createUser(), editUserDetails(), authenticateUser()

2. **Administrator Class*:***

   Attributes: AdminID, AdminName, Email

   Methods: configureSystem(), manageUserRoles(), authenticateAdministrator()

3. **AuthenticationManager Class:**

   Attributes: None

   Methods: authenticate(User user), validateCredentials(String username, String password)

Routines:

1. **User Class Routines:**

   createUser()

   editUserDetails()

   authenticateUser()

2. **Administrator Class Routines:**

configureSystem()

manageUserRoles()

authenticateAdministrator()

3. **AuthenticationManager Class Routines:**

authenticate(User user)

validateCredentials(String username, String password)

Internal Routine Design:

1. User Class Routines:

```
class User:
    # ----------------------01-----------------------
    def createUser(self):
        # Step 1: Gather user details
        user_details = self.gatherUserDetails()

        # Step 2: Validate the gathered information
        self.validateUserDetails(user_details)

        # Step 3: Generate a unique UserID
        user_details['UserID'] = self.generateUniqueUserID()

        try:
            # Step 4: Save user details in the database
            self.saveUserDetailsToDatabase(user_details)
        except DatabaseErrors as e:
            # Handle database errors
            raise e
```

```
        # Step 5: Return the created user object
        return self.createUserObject(user_details)

    def gatherUserDetails(self):
        # Implementation to gather user details from the user
            interface.
        pass

    def validateUserDetails(self, user_details):
        # Implementation to validate user details.
        pass

    def generateUniqueUserID(self):
        # Implementation to generate a unique user ID.
        pass

    def saveUserDetailsToDatabase(self, user_details):
        # Implementation to save user details in the database.
        pass
```

```
  def createUserObject(self, user_details):
      # Implementation to create a user object with the given
          details.
      pass
-----------------------02-----------------------
  def editUserDetails(self):

  # Step 1: Authenticate the user
  user = self.authenticateUser()

  # Step 2: Gather updated user details
  updated_user_details = self.gatherUpdatedUserDetails()

  # Step 3: Validate updated user details
  self.validateUpdatedUserDetails(updated_user_details)

  try:
      # Step 4: Update user details in the database
      self.updateUserDetailsInDatabase(user, updated_user_details)
  except DatabaseErrors as e:
```

```
    except DatabaseErrors as e:
        # Handle database errors
        raise e
# ----------------------03-----------------------
    def authenticateUser(self):

    # Step 1: Gather user credentials
    credentials = self.gatherUserCredentials()

    # Step 2: Validate user credentials
    self.validateUserCredentials(credentials)

    # Step 3: Authenticate the user
    authenticated_user = self.authenticate(credentials)

    return authenticated_user

    def gatherUserCredentials(self):
        # Implementation to gather user credentials from the user
            interface.
```

```
    def gatherUserCredentials(self):
        # Implementation to gather user credentials from the user
            interface.
        pass

    def validateUserCredentials(self, credentials):
        # Implementation to validate user credentials.
        pass

    def authenticate(self, credentials):
        # Implementation to authenticate the user.
        pass
    # ----------------------END-----------------------
```

2. Administrator Class Routines:

```
class Administrator:
    #_____________________________01_____________________________
    def configureSystem(self):

        # Step 1: Authenticate the administrator
        admin = self.authenticateAdministrator()

        # Step 2: Gather system configuration details
        system_config_details = self.gatherSystemConfigDetails()

        # Step 3: Validate system configuration details
        self.validateSystemConfigDetails(system_config_details)

        try:
            # Step 4: Update system configuration
            self.updateSystemConfig(admin, system_config_details)
        except DatabaseErrors as e:
            # Handle database errors
            raise e
```

```
    def gatherSystemConfigDetails(self):

        # Implementation to gather system configuration details.
        pass

    def validateSystemConfigDetails(self, system_config_details):
        # Implementation to validate system configuration details.
        pass

    def updateSystemConfig(self, admin, system_config_details):
        # Implementation to update system configuration.
        pass
#_____________________________02_____________________________
    def manageUserRoles(self):

    # Step 1: Authenticate the administrator
    admin = self.authenticateAdministrator()
```

```
    # Step 2: Select a user
    selected_user = self.selectUser()

    try:
        # Step 3: Manage roles for the selected user
        self.manageRoles(selected_user)
    except RoleManagementErrors as e:
        # Handle role management errors
        raise e

    def selectUser(self):
        # Implementation to select a user.
        pass

    def manageRoles(self, user):
        # Implementation to manage roles for the selected user.
        pass
```

```
#_____________________________03_____________________________
    def authenticateAdministrator(self):

    # Step 1: Gather administrator credentials
    credentials = self.gatherAdminCredentials()

    # Step 2: Validate administrator credentials
    self.validateAdminCredentials(credentials)

    # Step 3: Authenticate the administrator
    authenticated_admin = self.authenticateAdmin(credentials)

    return authenticated_admin

    def gatherAdminCredentials(self):
        # Implementation to gather administrator credentials.
        pass
```

```
    def validateAdminCredentials(self, credentials):
        # Implementation to validate administrator credentials.
        pass

    def authenticateAdmin(self, credentials):
        # Implementation to authenticate the administrator.
        pass
#_____________________________________END_____________________________________
```

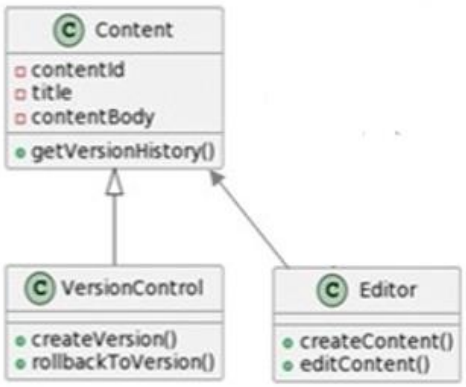
3. AuthenticationManager Class Routines:

```
class AuthenticationManager:
    #_____________________________01_____________________________
    def authenticate(self, user):

        # Step 1: Gather user credentials
        credentials = self.gatherUserCredentials()

        # Step 2: Validate user credentials
        self.validateUserCredentials(credentials)

        # Step 3: Authenticate the user
        authenticated_user = self.authenticate(credentials)

        return authenticated_user

    def gatherUserCredentials(self):
        # Implementation to gather user credentials from the user
            interface.
        pass
```

```
    def validateUserCredentials(self, credentials):
        # Implementation to validate user credentials.
        pass

    def authenticate(self, credentials):
        # Implementation to authenticate the user.
        pass
#_____________________________02_____________________________
    def validateCredentials(self, username, password):

    # Step 1: Check if the username and password meet validation
        criteria
    if not self.isValidUsername(username) or not self
        .isValidPassword(password):
        raise ValidationErrors("Invalid username or password.")

    def isValidUsername(self, username):
        # Implementation to check if the username is valid.
        pass
```

```
    def isValidPassword(self, password):
        # Implementation to check if the password is valid.
        pass
#_____________________________END_____________________________
```

(2) Content Creation and Editing Subsystem Class Diagram



Data:

1. **Content Class:**

   Attributes: ContentID, Title, DateCreated

   Methods: createContent(User creator, String title), editContent(User editor, String newTitle), applyVersionControl()

2. **Editor Class:**

   Attributes: EditorID, EditorName

   Methods: editContent(Content content, String newTitle), reviewContent(Content content), notifyAuthor()

3. **VersionControl Class:**

   Attributes: VersionID, Changes, DateModified

   Methods: trackChanges(Content content), saveVersion()

Routines:

1. **Content Class Routines:**

createContent(User creator, String title)

editContent(User editor, String newTitle)

applyVersionControl()

2. Editor Class Routines:

editContent(Content content, String newTitle)

reviewContent(Content content)

notifyAuthor()

3. VersionControl Class Routines:

trackChanges(Content content)

saveVersion()

Internal Routine Design:

1. Content Class Routines:

```
class Content:
    #_____________________________01_____________________
    def createContent(self, creator, title):

        # Step 1: Check user permissions for content creation
        self.checkContentCreationPermission(creator)

        # Step 2: Create a new content object with the provided
            details
        content_object = self.createNewContentObject(creator, title
            )

        # Step 3: Apply version control to the content
        self.applyVersionControl(content_object)

        try:
            # Step 4: Save content details in the database
            self.saveContentDetailsToDatabase(content_object)
        except DatabaseErrors as e:
            # Handle database errors
```

```
        except DatabaseErrors as e:
            # Handle database errors
            raise e

        # Step 5: Return the created content object
        return content_object

    def checkContentCreationPermission(self, user):
        # Implementation to check if the user has permission to
            create content.
        pass

    def createNewContentObject(self, creator, title):
        # Implementation to create a new content object.
        pass

    def applyVersionControl(self, content_object):
        # Implementation to apply version control to the content.
        pass
```

```
    def saveContentDetailsToDatabase(self, content_object):
        # Implementation to save content details in the database.
        pass
#_____________________________02_____________________________
    def editContent(self, editor, new_title):

    # Step 1: Check user permissions for content editing
    self.checkContentEditingPermission()

    # Step 2: Edit the content with the new title
    edited_content = self.editContentDetails(editor, new_title)

    # Step 3: Notify the author of the changes
    try:
        self.notifyAuthor(edited_content)
    except NotificationErrors as e:
        # Handle notification errors
        raise e
```

```
    try:
        # Step 4: Save the edited content details in the database
        self.saveEditedContentDetailsToDatabase(edited_content)
    except DatabaseErrors as e:
        # Handle database errors
        raise e

    # Step 5: Return the edited content object
    return edited_content

def checkContentEditingPermission(self):
    # Implementation to check if the user has permission to edit
        content.
    pass

def editContentDetails(self, editor, new_title):
    # Implementation to edit the content details.
    pass
```

```
def notifyAuthor(self, edited_content):
    # Implementation to notify the author of content changes.
    pass

def saveEditedContentDetailsToDatabase(self, edited_content):
    # Implementation to save edited content details in the database
        .
    pass
 #_____________________________03_____________________________
    class VersionControl:
    def applyVersionControl(self, content_object):

        # Step 1: Track changes to the content
        self.trackChanges(content_object)

        try:
            # Step 2: Save the versioned content
            self.saveVersion(content_object)
        except VersionControlErrors as e:
            # Handle version control errors
```

```
    def trackChanges(self, content_object):
        # Implementation to track changes to the content.
        pass

    def saveVersion(self, content_object):
        # Implementation to save the versioned content.
        pass
#_____________________________________END_____________________
```

2. Editor Class Routines:

```
class Editor:
    #_____________________________01_____________________
    def editContent(self, content, new_title):

        # Step 1: Check user permissions for content editing
        self.checkContentEditingPermission()

        # Step 2: Edit the content with the new title
        edited_content = self.editContentDetails(content, new_title)

        # Step 3: Notify the author of the changes
        try:
            self.notifyAuthor(content, edited_content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

        try:
            # Step 4: Save the edited content details in the
                database
```

```
            self.saveEditedContentDetailsToDatabase(edited_content)
        except DatabaseErrors as e:
            # Handle database errors
            raise e

        # Step 5: Return the edited content object
        return edited_content

    def checkContentEditingPermission(self):
        # Implementation to check if the user has permission to edit
            content.
        pass

    def editContentDetails(self, content, new_title):
        # Implementation to edit the content details.
        pass

    def notifyAuthor(self, old_content, edited_content):
        # Implementation to notify the author of content changes.
        pass
```

```
#_____________________________________02_____________________________________
    def reviewContent(self, content):

    # Step 1: Check user permissions for content review
    self.checkContentReviewPermission()

    # Step 2: Perform the content review
    review_outcome = self.performContentReview(content)

    # Step 3: Notify the author of the review outcome
    try:
        self.notifyAuthor(content, review_outcome)
    except NotificationErrors as e:
        # Handle notification errors
        raise e

def checkContentReviewPermission(self):
    # Implementation to check if the user has permission to review
        content.
    pass
```

```
def performContentReview(self, content):
    # Implementation to perform the content review.
    pass

def notifyAuthor(self, content, review_outcome):
    # Implementation to notify the author of the review outcome.
    pass
#_____________________________03_____________________________
    def notifyAuthor(self, old_content, new_content):

    # Step 1: Check if the author has opted-in for notifications
    if self.isAuthorSubscribed(old_content.creator):
        # Step 2: Send a notification to the author about the
            content changes
        try:
            self.sendNotification(old_content.creator, new_content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e
```

```
def isAuthorSubscribed(self, author):
    # Implementation to check if the author has opted-in for
        notifications.
    pass

def sendNotification(self, author, content):
    # Implementation to send a notification to the author about the
        content changes.
    pass
#_____________________________END_____________________
```

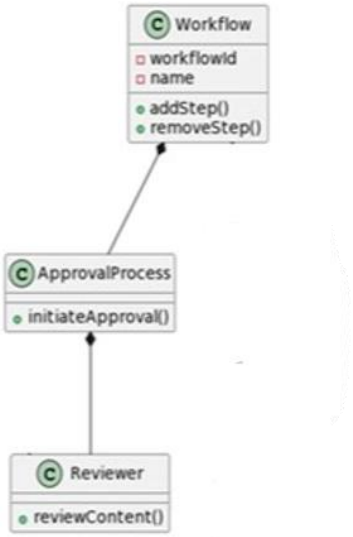
3. VersionControl Class Routines:

```
class VersionControl:
    #_____________________01_____________________
    def trackChanges(self, content_object):

        # Step 1: Determine the changes made to the content
        changes = self.determineContentChanges(content_object)

        try:
            # Step 2: Log the changes for version control
            self.logChanges(content_object, changes)
        except VersionControlErrors as e:
            # Handle version control errors
            raise e

    def determineContentChanges(self, content_object):
        # Implementation to determine the changes made to the
            content.
        pass

    def logChanges(self, content_object, changes):
```

```
    def logChanges(self, content_object, changes):
        # Implementation to log the changes for version control.
        pass


_____________________________________02_____________________
    def saveVersion(self, content_object):

    # Step 1: Create a new version for the content
    new_version = self.createVersion(content_object)

    try:
        # Step 2: Save the versioned content in the database
        self.saveVersionToDatabase(new_version)
    except DatabaseErrors as e:
        # Handle database errors
        raise e

    return new_version
```

```
def createVersion(self, content_object):
    # Implementation to create a new version for the content.
    pass

def saveVersionToDatabase(self, version):
    # Implementation to save the versioned content in the database.
    pass
#_____________________________END_____________________
```

(3) Workflow and Approval Subsystem Class Diagram



Data:

1. **Workflow Class:**

Attributes: WorkflowID, Steps, Status

Methods: initiateWorkflow(Content content), progressWorkflow(), notifyReviewers()

2. **ApprovalProcess Class:**

Attributes: ApprovalProcessID, Approvers, Status

Methods: approveContent(Content content), rejectContent(Content content), notifyAuthor()

3. **Reviewer Class:**

Attributes: ReviewerID, ReviewerName

Methods: reviewContent(Content content), notifyOutcome()

Routines:

1. **Workflow Class Routines:**

initiateWorkflow(Content content)

progressWorkflow()

notifyReviewers()

2. **ApprovalProcess Class Routines:**

approveContent(Content content)

rejectContent(Content content)

notifyAuthor()

3. <u>Reviewer Class Routines:</u>

reviewContent(Content content)

notifyOutcome()

<u>Internal Routine Design:</u>

1. Workflow Class Routines:

```
class Workflow:
    #_____________________________01_____________________
    def initiateWorkflow(self, content):

        # Step 1: Check if a workflow is already in progress for the
            content
        self.checkExistingWorkflow(content)

        # Step 2: Set the initial status and steps for the workflow
        workflow_object = self.setInitialWorkflowDetails(content)

        # Step 3: Notify reviewers about the new workflow
        try:
            self.notifyReviewers(content, workflow_object)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

        try:
            # Step 4: Save workflow details in the database
```

```
        try:
            # Step 4: Save workflow details in the database
            self.saveWorkflowDetailsToDatabase(workflow_object)
        except DatabaseErrors as e:
            # Handle database errors
            raise e

        # Step 5: Return the initiated workflow object
        return workflow_object

    def checkExistingWorkflow(self, content):
        # Implementation to check if a workflow is already in
            progress for the content.
        pass

    def setInitialWorkflowDetails(self, content):
        # Implementation to set initial status and steps for the
            workflow.
        pass
```

```
    def notifyReviewers(self, content, workflow_object):
        # Implementation to notify reviewers about the new workflow.
        pass

    def saveWorkflowDetailsToDatabase(self, workflow_object):
        # Implementation to save workflow details in the database.
        pass
#_____________________________02_____________________________
        def progressWorkflow(self, workflow_object):

    # Step 1: Check if the workflow is in progress
    if workflow_object.status == 'In Progress':
        # Step 2: Move to the next step in the workflow
        self.moveToNextWorkflowStep(workflow_object)

        # Step 3: Update the workflow status
        self.updateWorkflowStatus(workflow_object)

        # Step 4: Notify reviewers about the progress
        try:
```

```
        try:
            self.notifyReviewers(workflow_object)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

def moveToNextWorkflowStep(self, workflow_object):
    # Implementation to move to the next step in the workflow.
    pass

def updateWorkflowStatus(self, workflow_object):
    # Implementation to update the workflow status.
    pass

def notifyReviewers(self, workflow_object):
    # Implementation to notify reviewers about the progress.
    pass
```

```
#_____________________________03_____________________________
    def notifyReviewers(self, workflow_object):

    # Step 1: Check if there are reviewers assigned to the workflow
    if workflow_object.reviewers:
        # Step 2: Send notifications to the assigned reviewers
        try:
            self.sendNotificationsToReviewers(workflow_object)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

def sendNotificationsToReviewers(self, workflow_object):
    # Implementation to send notifications to the assigned reviewers
        .
    pass
#_____________________________END_____________________
```

2. ApprovalProcess Class Routines:

```
class ApprovalProcess:
    #_____________________01_____________________
    def approveContent(self, content):

        # Step 1: Check if the content is in the approval process
        self.checkContentInApprovalProcess(content)

        # Step 2: Approve the content
        self.performContentApproval(content)

        # Step 3: Update the approval status
        self.updateApprovalStatus(content)

        # Step 4: Notify the author of the approval
        try:
            self.notifyAuthor(content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e
```

```
    def checkContentInApprovalProcess(self, content):
        # Implementation to check if the content is in the approval
            process.
        pass

    def performContentApproval(self, content):
        # Implementation to perform content approval.
        pass

    def updateApprovalStatus(self, content):
        # Implementation to update the approval status.
        pass

    def notifyAuthor(self, content):
        # Implementation to notify the author of the approval.
        pass
```

```
#_____________________________________02_____________________________
    def rejectContent(self, content):

    # Step 1: Check if the content is in the approval process
    self.checkContentInApprovalProcess(content)

    # Step 2: Reject the content
    self.performContentRejection(content)

    # Step 3: Update the rejection status
    self.updateRejectionStatus(content)

    # Step 4: Notify the author of the rejection
    try:
        self.notifyAuthor(content)
    except NotificationErrors as e:
        # Handle notification errors
        raise e
```

```
def performContentRejection(self, content):
    # Implementation to perform content rejection.
    pass

def updateRejectionStatus(self, content):
    # Implementation to update the rejection status.
    pass
#_____________________________03_____________________________
    def notifyAuthor(self, content):

    # Step 1: Check if the author has opted-in for notifications
    if self.isAuthorSubscribed(content.creator):
        # Step 2: Send a notification to the author about the
            approval or rejection
        try:
            self.sendNotification(content.creator, content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e
```

```
def isAuthorSubscribed(self, author):
    # Implementation to check if the author has opted-in for
        notifications.
    pass

def sendNotification(self, author, content):
    # Implementation to send a notification to the author about the
        approval or rejection.
    pass
#_____________________________END_____________________________
```

3. Reviewer Class Routines:

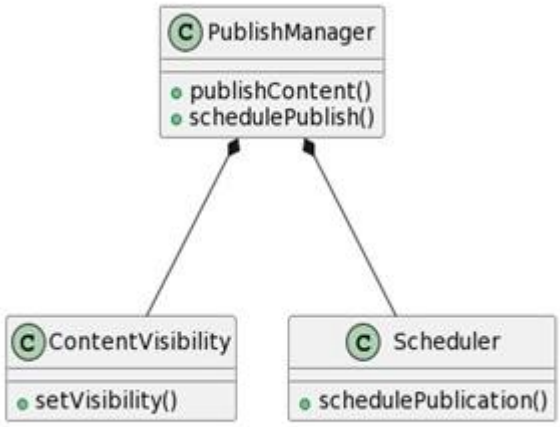
```
class Reviewer:
    #_____________________________01_____________________________
    def reviewContent(self, content):

        # Step 1: Check if the reviewer has permission to review
            content
        self.checkReviewPermission()

        # Step 2: Review the content
        review_outcome = self.performContentReview(content)

        # Step 3: Provide feedback or comments
        self.provideReviewFeedback(content, review_outcome)

        # Step 4: Notify the outcome of the review
        try:
            self.notifyOutcome(content, review_outcome)
        except NotificationErrors as e:
            # Handle notification errors
            raise e
```

```
    def checkReviewPermission(self):
        # Implementation to check if the reviewer has permission to
            review content.
        pass

    def performContentReview(self, content):
        # Implementation to perform the content review and return
            the review outcome.
        pass

    def provideReviewFeedback(self, content, review_outcome):
        # Implementation to provide feedback or comments based on
            the review outcome.
        pass

    def notifyOutcome(self, content, review_outcome):
        # Implementation to notify the outcome of the review.
        pass
```

```
#_____________________________02_____________________________
    def notifyOutcome(self, content, review_outcome):

    # Step 1: Check if there are subscribers interested in review
        outcomes
    if self.hasSubscribersForReviewOutcomes():
        # Step 2: Send notifications to the subscribers
        try:
            self.sendReviewOutcomeNotifications(content,
                review_outcome)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

def hasSubscribersForReviewOutcomes(self):
    # Implementation to check if there are subscribers interested in
        review outcomes.
    pass
```

```
def sendReviewOutcomeNotifications(self, content, review_outcome):
    # Implementation to send notifications to the subscribers about
        the review outcome.
    pass
#_____________________END_____________________________
```

(4) Publishing Subsystem Class Diagram



Data:

1. **PublishManager Class:**

Attributes: PublishManagerID, PublishManagerName

Methods: publishContent(Content content), schedulePublication(Content content, Date publishDate), notifySubscribers()

2. **ContentVisibility Class:**

Attributes: ContentVisibilityID, VisibilitySettings

Methods: setVisibility(Content content, User user), applyVisibilitySettings()

3. **Scheduler Class:**

Attributes: SchedulerID, ScheduleSettings

Methods: scheduleTask(Date date, Task task), executeTask(Task task)

Routines:

1. **PublishManager Class Routines:**

publishContent(Content content)

schedulePublication(Content content, Date publishDate)

notifySubscribers()

2. **ContentVisibility Class Routines:**

setVisibility(Content content, User user)

applyVisibilitySettings()

3. **Scheduler Class Routines:**

scheduleTask(Date date, Task task)

executeTask(Task task)

Internal Design Routines:

1. Publish Manager Class Routines:

```
class PublishManager:
    #_____________________01_____________________________
    def publishContent(self, content):

        # Step 1: Check if the user has permission to publish
            content
        self.checkContentPublicationPermission()

        # Step 2: Publish the content
        published_content = self.performContentPublication(content)

        # Step 3: Notify subscribers about the new publication
        try:
            self.notifySubscribers(content, published_content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e

        try:
            # Step 4: Save publication details in the database
```

```
            # Step 4: Save publication details in the database
            self.savePublicationDetailsToDatabase(published_content)
        except DatabaseErrors as e:
            # Handle database errors
            raise e

        # Step 5: Return the published content object
        return published_content

    def checkContentPublicationPermission(self):
        # Implementation to check if the user has permission to
            publish content.
        pass

    def performContentPublication(self, content):
        # Implementation to perform the content publication and
            return the published content.
        pass
```

```
def notifySubscribers(self, content, published_content):
    # Implementation to notify subscribers about the new
        publication.
    pass

def savePublicationDetailsToDatabase(self, published_content):
    # Implementation to save publication details in the database
        .
    pass
#_____________________________02_____________________________-
    def schedulePublication(self, content, publish_date):

# Step 1: Check if the user has permission to schedule
    publication
self.checkPublicationSchedulingPermission()

# Step 2: Schedule the publication for the specified date
scheduled_publication = self.scheduleContentPublication(content,
    publish_date)
```

```
    try:
        # Step 3: Save the publication schedule details in the
            database
        self.savePublicationScheduleToDatabase(scheduled_publication
            )
    except DatabaseErrors as e:
        # Handle database errors
        raise e

    return scheduled_publication

def checkPublicationSchedulingPermission(self):
    # Implementation to check if the user has permission to schedule
        publication.
    pass

def scheduleContentPublication(self, content, publish_date):
    # Implementation to schedule the publication of the content and
        return the scheduled publication.
    pass
```

```
def savePublicationScheduleToDatabase(self, scheduled_publication):
    # Implementation to save the publication schedule details in the
        database.
    pass
#_____________________________03_____________________________
def notifySubscribers(self, content, published_content):

    # Step 1: Check if there are subscribers interested in new
        publications
    if self.hasSubscribersForNewPublications():
        # Step 2: Send notifications to the subscribers
        try:
            self.sendPublicationNotifications(content,
                published_content)
        except NotificationErrors as e:
            # Handle notification errors
            raise e
```

```
def hasSubscribersForNewPublications(self):
    # Implementation to check if there are subscribers interested in
        new publications.
    pass

def sendPublicationNotifications(self, content, published_content):
    # Implementation to send notifications to the subscribers about
        the new publication.
    pass
#_____________________________END_____________________________
```

2. ContentVisibility Class Routines:

```
class ContentVisibility:
    def setVisibility(self, content, user):

        # Step 1: Check if the user has permission to set content
            visibility
        self.checkContentVisibilityPermission()

        # Step 2: Set the visibility of content for the user
        visibility_settings = self.setVisibilityForUser(content,
            user)

        try:
            # Step 3: Save the visibility settings in the database
            self.saveVisibilitySettingsToDatabase
                (visibility_settings)
        except DatabaseErrors as e:
            # Handle database errors
            raise e
```

```
  def checkContentVisibilityPermission(self):
      # Implementation to check if the user has permission to set
          content visibility.
      pass

  def setVisibilityForUser(self, content, user):
      # Implementation to set the visibility of content for the
          user and return visibility settings.
      pass

  def saveVisibilitySettingsToDatabase(self, visibility_settings):
      # Implementation to save the visibility settings in the
          database.
      pass
```

```
#_____________________________02_____________________________
    def applyVisibilitySettings(self):

    # Step 1: Retrieve visibility settings for all content
    visibility_settings = self.retrieveVisibilitySettings()

    # Step 2: Apply the visibility settings
    self.performVisibilitySettingsApplication(visibility_settings)

def retrieveVisibilitySettings(self):
    # Implementation to retrieve visibility settings for all content
        .
    pass

def performVisibilitySettingsApplication(self, visibility_settings):
    # Implementation to apply the visibility settings for all
        content.
    pass
#_____________________________END_____________________________
```

3. Scheduler Class Routines:

```
class Scheduler:
    #_____________________________01_____________________________
    def scheduleTask(self, date, task):

        # Step 1: Check if the user has permission to schedule tasks
        self.checkTaskSchedulingPermission()

        # Step 2: Schedule the task for the specified date
        scheduled_task = self.scheduleTaskForDate(date, task)

        try:
            # Step 3: Save the task schedule details in the database
            self.saveTaskScheduleToDatabase(scheduled_task)
        except DatabaseErrors as e:
            # Handle database errors
            raise e

        return scheduled_task
```

```
    def checkTaskSchedulingPermission(self):
        # Implementation to check if the user has permission to
            schedule tasks.
        pass

    def scheduleTaskForDate(self, date, task):
        # Implementation to schedule the task for the specified date
            and return the scheduled task.
        pass

    def saveTaskScheduleToDatabase(self, scheduled_task):
        # Implementation to save the task schedule details in the
            database.
        pass
```

```
#_____________________________02_____________________________
    def executeTask(self, task):

    # Step 1: Check if the user has permission to execute tasks
    self.checkTaskExecutionPermission()

    # Step 2: Execute the task
    executed_task = self.performTaskExecution(task)

    # Step 3: Update the task status
    self.updateTaskStatus(executed_task)

    try:
        # Step 4: Save the task execution details in the database
        self.saveTaskExecutionDetailsToDatabase(executed_task)
    except DatabaseErrors as e:
        # Handle database errors
        raise e

    return executed_task
```

```
def checkTaskExecutionPermission(self):
    # Implementation to check if the user has permission to execute
        tasks.
    pass

def performTaskExecution(self, task):
    # Implementation to perform the execution of the task and return
        the executed task.
    pass

def updateTaskStatus(self, executed_task):
    # Implementation to update the task status after execution.
    pass

def saveTaskExecutionDetailsToDatabase(self, executed_task):
    # Implementation to save the task execution details in the
        database.
    pass
#_____________________________END_____________________________
```

THE END