REPORT OF PROJECT

NEWS PORTAL SYSTEM

SUBJECT

SOFTWARE DESIGN AND ARCHITECTURE

TEACHER

SIR MUKHTIAR ZAMIN

SUBMITTED BY

AIMAN SHABBIR

FA22-BSE-104

DEADLINE

11. 7. 2024

Department of Software Engineering

COMSATS University Islamabad

Abbottabad campus

SCENARIO:

Make use case diagram, full dressed use case, system sequence diagram, communication diagram, class diagram. Implement principles in communication diagram in the form of code. Also make package diagram and it's layered architecture. Implement code in layered architecture by making user interface for it.

REPORT:

News Portal System

1. Use Case Diagram

The use case diagram depicts the interactions between users and the system of a news portal.

Actors:

1. **Admin**: Represents the administrator of the news portal.
2. **User:** Represents any user who interacts with the news portal.

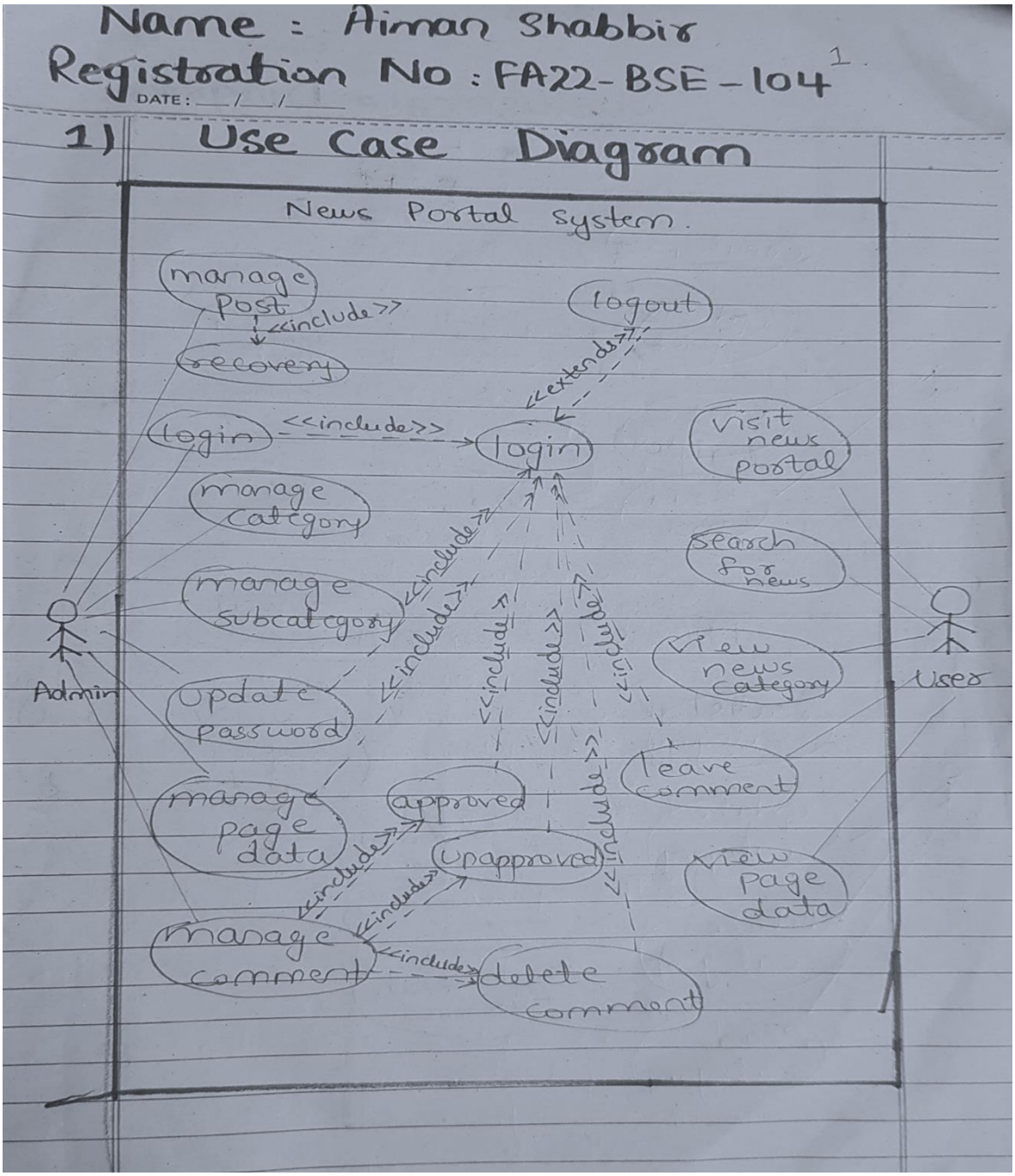
Use Cases:

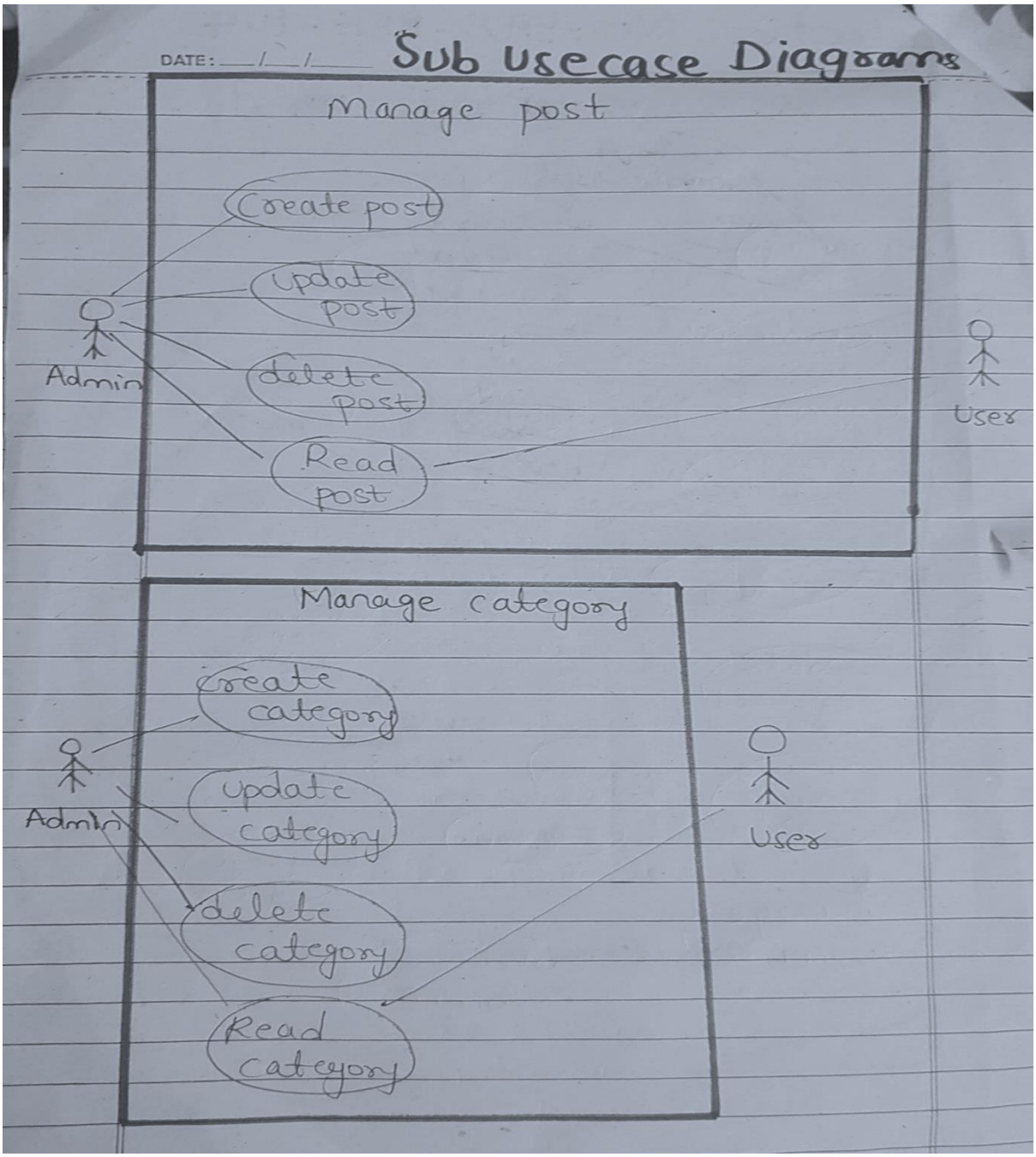
1. **Login**: Both the admin and users need to login to access the system.
2. **Manage Category**: The admin manages categories of news.
3. **Manage Subcategory:** The admin manages subcategories under each category.
4. **Manage Post:** The admin creates, edits, and manages news posts.
5. **Manage Page Data:** The admin manages the content of various pages in the news portal.
6. **Manage Comment:** The admin manages comments on the news posts.
7. **Update**: Allows the admin to update their password.
8. **Visit News Portal:** Users can browse and read the news content.
9. **Search for News:** Users can search for specific news using keywords or filters.
10. **View News Category:** Users can view news based on categories.
11. **Leave Comment:** Users can leave comments on news posts.
12. **View Page Data:** Users can view details about specific pages within the news portal.

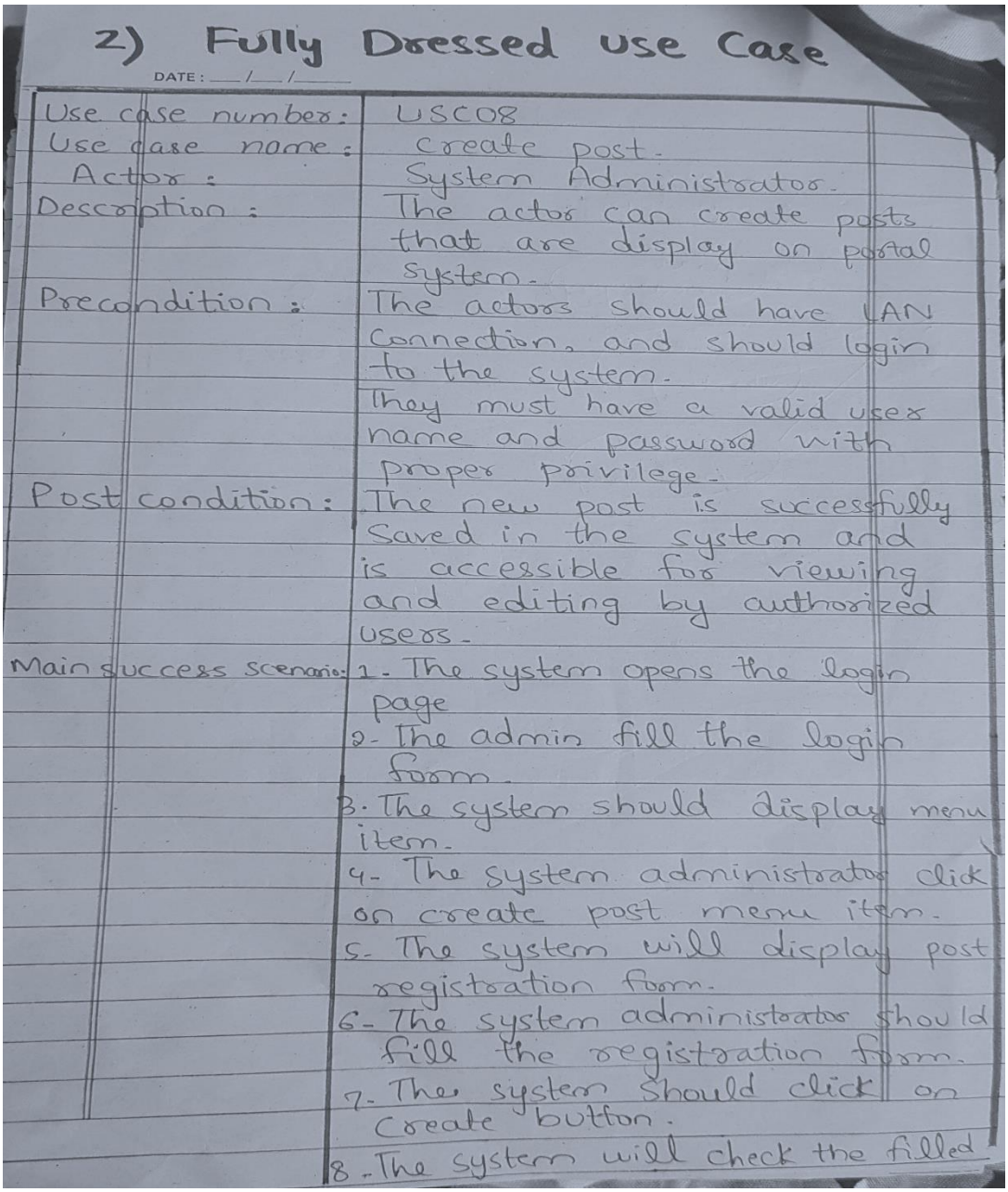
Relationships:

1. **Includes:** Indicates that one use case contains the functionality of another. For instance, "Manage Category" includes "Manage Subcategory" as a subcategory is managed within a larger category.

2. **Extends:** Indicates that one use case provides an optional or alternative functionality to another. For example, "Logout" extends "Login" as the user can log out of the system after a login.



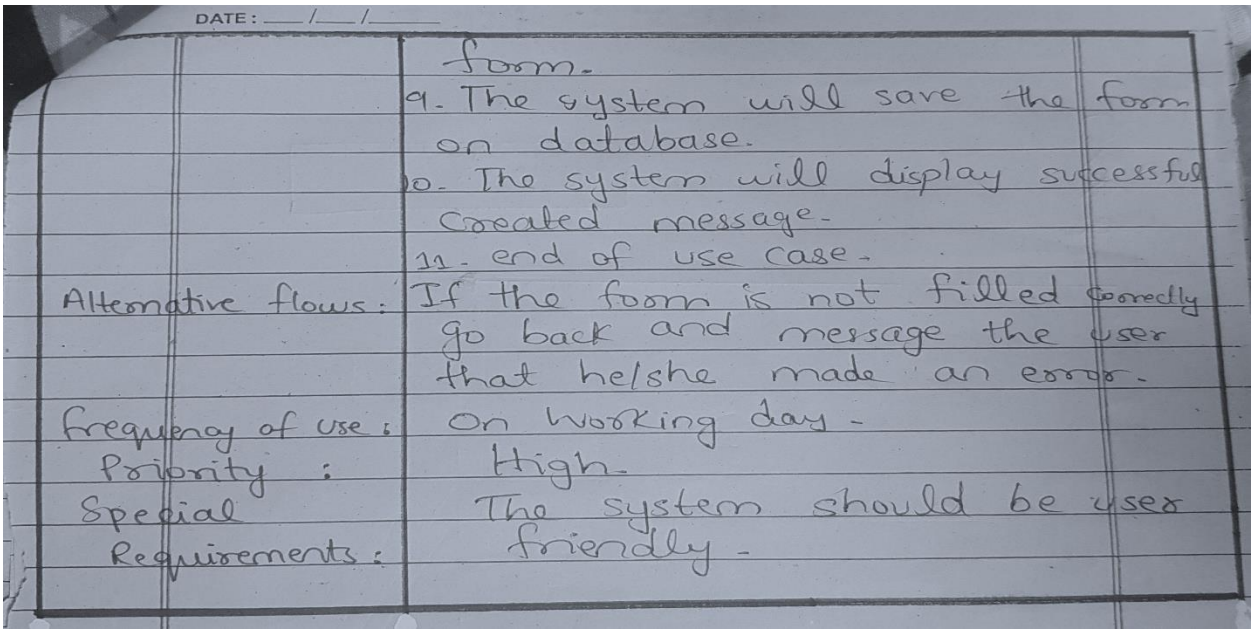

DATE: __/__/__
Sub usecase Diagrams
Manage post
Create post
Update post
delete post
Read post
Admin
User
Manage category
create category
update category
delete category
Read category
Admin
User



2) Fully Dressed use Case

DATE: ___/___/_____

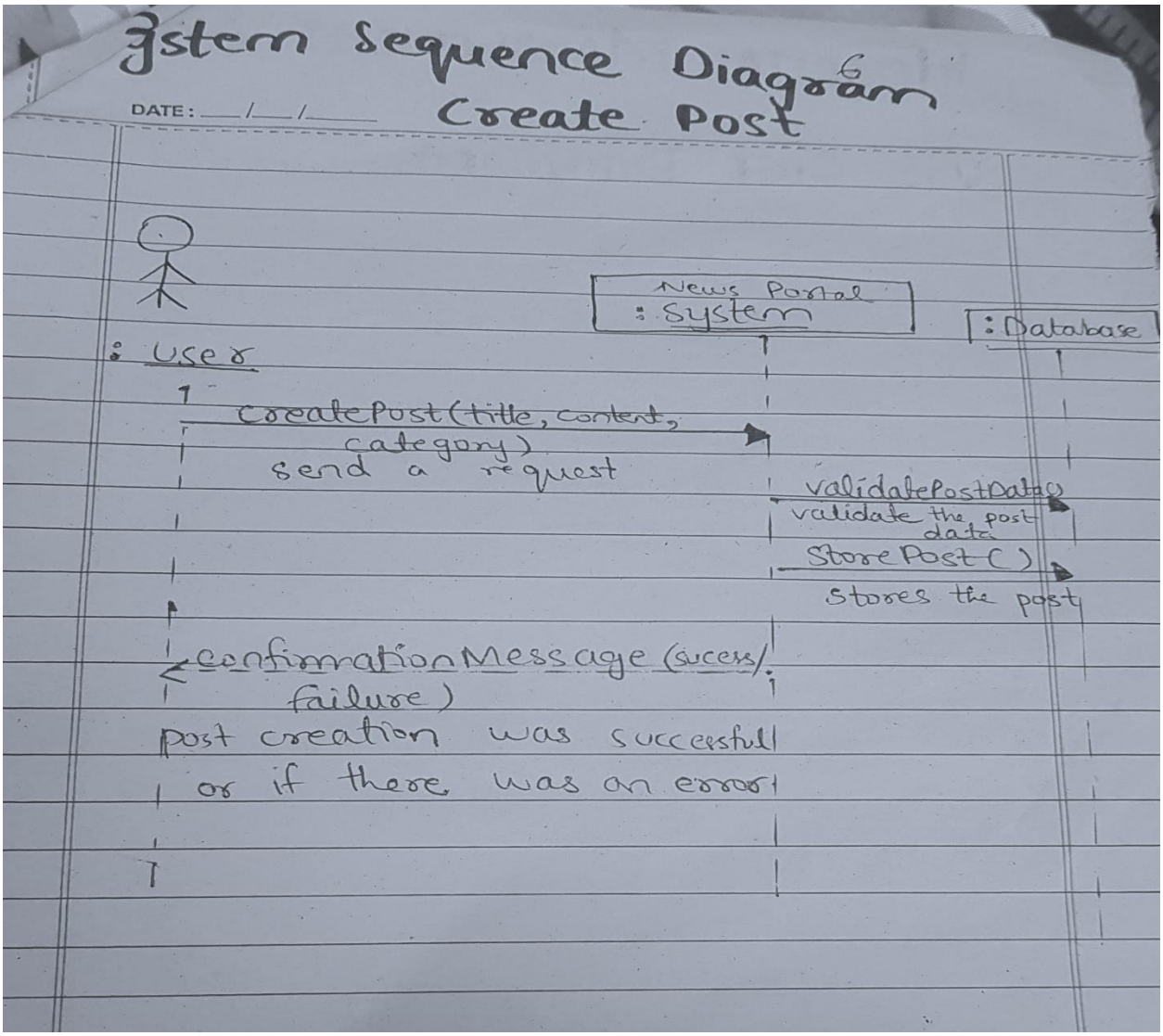
Use case number:	USC08
Use case name:	Create post.
Actor:	System Administrator.
Description:	The actor can create posts that are display on portal system.
Precondition:	The actors should have LAN Connection, and should login to the system. They must have a valid user name and password with proper privilege.
Post condition:	The new post is successfully Saved in the system and is accessible for viewing and editing by authorized users.
Main success scenario:	1. The system opens the login page 2. The admin fill the login foorm. 3. The system should display menu item. 4. The system administrator click on create post menu item. 5. The system will display post registration foorm. 6. The system administrator should fill the registration foorm. 7. The system should click on Create button. 8. The system will check the filled



2. System Sequence Diagram

The system sequence diagram depicts the interaction between a User and the News Portal system when the User attempts to create a new post.

1. **User Initiates Post Creation:** The user interacts with the News Portal system, intending to create a new post.
2. **User Sends Request:** The user provides the required information, including the title, content, and category of the post. This information is sent as a request to the system.
3. **System Validates Data:** The News Portal system receives the request and starts validating the provided data. This step ensures that the user-supplied data is valid and conforms to the system's requirements.
4. **System Stores the Post:** If the data validation is successful, the system stores the post in the database, thereby creating the new post.
5. **System Sends Confirmation:** The system sends a confirmation message back to the user indicating the success or failure of the post creation process. If the post was created successfully, the user receives a success message. If there was an error during validation or storage, the user receives a failure message, possibly with error details.



3. Communication Diagram

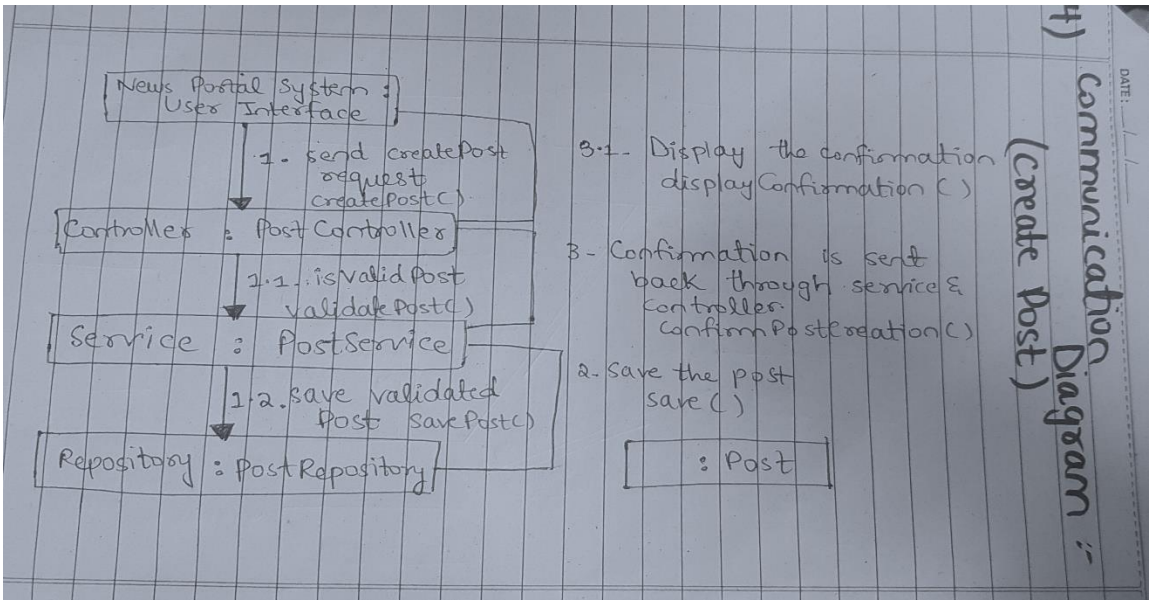
This diagram illustrates the flow of events in a news portal system when a user creates a new post.

1. **User Interface:** The user starts by initiating the creation of a new post through the user interface. This could be done via a button, form, or other UI element.
2. **Controller:** The user interface sends a "create post" request to the controller. The controller is the entry point for handling this request.
3. **Service:** The controller delegates the task of creating the post to the service layer. The service is responsible for business logic and data validation.
4. **Validation:** The service checks if the provided post data is valid. This may include things like ensuring all required fields are filled, validating formatting, and potentially checking for duplicate content.

5. **Repository:** If the post data is valid, the service saves it to the database through the repository. The repository interacts with the database to persist the post.
6. **Confirmation:** Once the post is saved, the service generates a confirmation and sends it back to the controller. This confirmation might include a message indicating success and potentially some information about the newly created post.
7. **Display Confirmation:** The controller receives the confirmation from the service and displays it to the user through the user interface. This provides feedback to the user that their action was successful.

Important Notes:

1. **Layers:** Communication diagram illustrates a common layered architecture (UI, Controller, Service, Repository) in software development.
2. **Separation of Concerns:** Each layer focuses on a specific set of responsibilities. This promotes maintainability and scalability.
3. **Error Handling:** Real-world implementations would include error handling mechanisms to gracefully deal with failures at any point in the process.



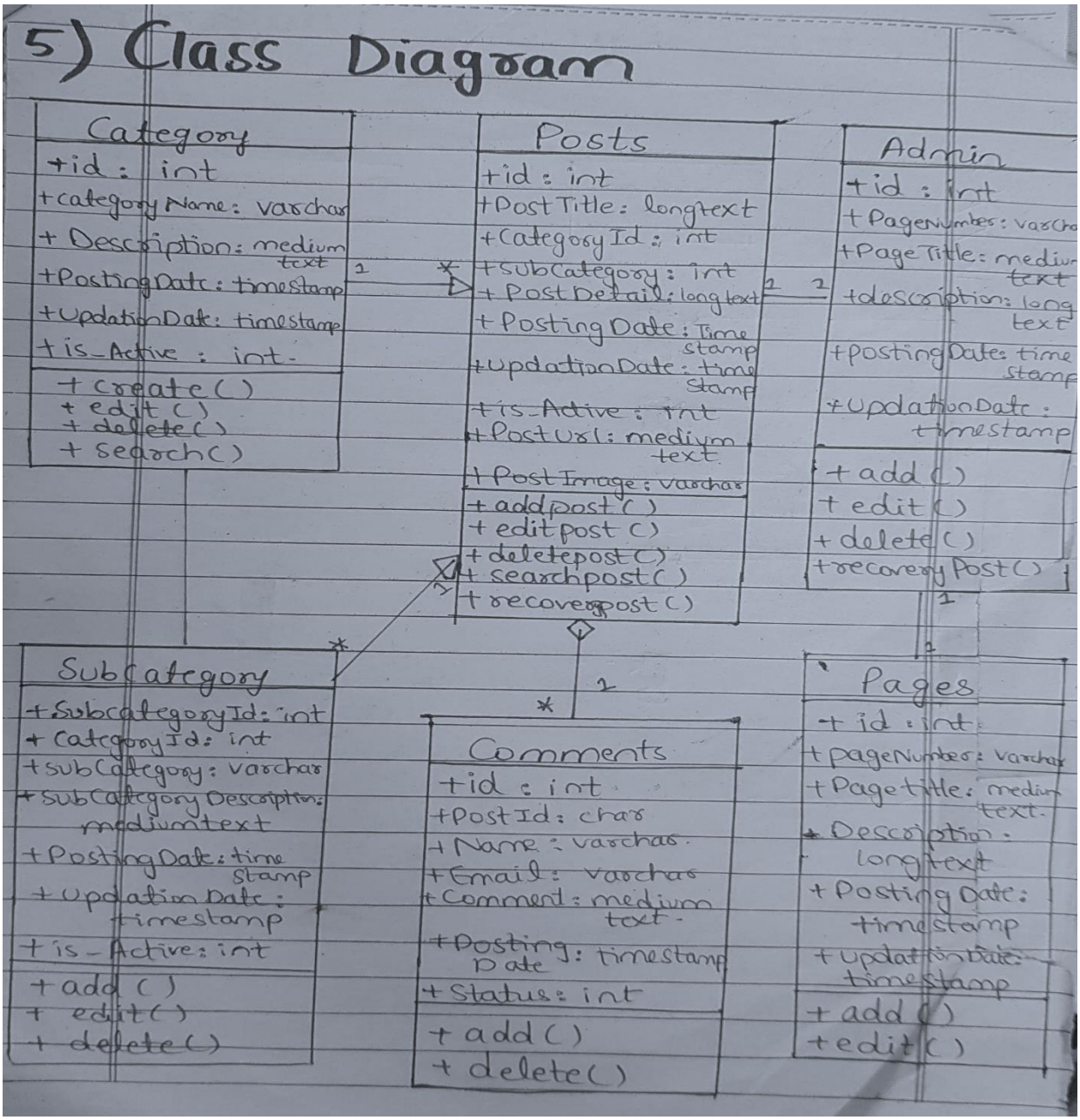
4. Class Diagram

The class diagram represents a news portal system with different entities like Category, Subcategory, Posts, Comments, Pages, and Admin.

Entities:

1. **Category:** This entity defines different categories like News, Sports, Entertainment, etc.

2. **SubCategory:** A subcategory further refines the category, for example, "Sports" can have subcategories like "Cricket", "Football", etc.
3. **Posts:** This is the core entity representing individual news articles or blog posts. It stores information like the post title, content (Post Detail), category and subcategory information, author details, posting date, update date, whether the post is active or inactive.
4. **Comments:** This entity holds comments associated with a particular post.
5. **Pages:** This entity likely refers to static pages like 'About Us', 'Contact Us', 'Terms & Conditions', etc.
6. **Admin:** This entity represents an administrator who has the privilege to manage the website content, including creating, editing, and deleting posts.



Communication Diagram incorporates several GRASP principles:

The "Create Post" use case of news portal system, in its communication diagram incorporates several GRASP (General Responsibility Assignment Software Patterns) principles;

1. **Controller:** The PostController handles the user's input and coordinates the creation of a post. It's responsible for delegating the validation and saving processes to other objects.
2. **Information Expert:** The PostService has the expertise needed to validate and save the post. It contains the logic required for these operations, making it the right choice for handling the validation.

3. **Low Coupling:** The interactions between the PostController, PostService, and PostRepository aim to keep dependencies between objects minimal. Each class handles specific tasks, reducing direct dependencies on other classes.
4. **High Cohesion:** Each object (e.g., PostController, PostService, PostRepository) has a well-defined role. This specialization helps maintain focus, making each object highly cohesive. For example, PostRepository is solely responsible for saving the data to the database.
5. **Creator:** The PostRepository creates and stores the Post object. Since it manages access to the database, it makes sense for it to handle the instantiation and storage of post data.

Code implementation:

```
import java.util.HashMap;
import java.util.Map;
class Post {
    private Long id;
    private String title;
    private String content;
    private String author;

    public Post(String title, String content, String author) {
        this.title = title;
        this.content = content;
        this.author = author;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
```

```
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public String getContent() {
        return content;
    }

    public String getAuthor() {
        return author;
    }
}

class PostController {
    private final PostService postService;

    public PostController(PostService postService) {
        this.postService = postService;
    }

    public void createPost(String title, String content, String author) {
        Post post = new Post(title, content, author);
        boolean isCreated = postService.createPost(post);
        if (isCreated) {
            System.out.println("Post created successfully.");
```

```
        } else {
            System.out.println("Failed to create post.");
        }
    }
}

class PostService {
    private final PostRepository postRepository;

    public PostService(PostRepository postRepository) {
        this.postRepository = postRepository;
    }

    public boolean createPost(Post post) {
        if (isValidPost(post)) {
            return postRepository.save(post);
        }
        return false;
    }

    private boolean isValidPost(Post post) {
        // Basic validation logic
        return post.getTitle() != null && !post.getTitle().isEmpty()
                && post.getContent() != null && !post.getContent().isEmpty()
                && post.getAuthor() != null && !post.getAuthor().isEmpty();
    }
}
```

```
class PostRepository {
    private final Map<Long, Post> database = new HashMap<>();
    private Long idCounter = 1L;

    public boolean save(Post post) {
        post.setId(idCounter++);
        database.put(post.getId(), post);
        System.out.println("Post saved with ID: " + post.getId());
        return true;
    }

    public Post findById(Long id) {
        return database.get(id);
    }
}

public class Main {
    public static void main(String[] args) {
        PostRepository postRepository = new PostRepository();
        PostService postService = new PostService(postRepository);
        PostController postController = new PostController(postService);

        postController.createPost("Sample Title", "This is the content of the post.", "Author
Name");
    }
}
```

5. Package Diagram:

Understanding the Layers in the Diagram

The diagram divides the system into several layers, each with specific responsibilities, which align well with a layered architecture design.

- **Presentation Layer**: This layer handles the user interfaces, such as UserInterface for general users and AdminInterface for administrators. This layer interacts with the Users package, which represents the users of the system (both regular users and administrators).
- **Service Layer**: This layer contains business services, including UserService and AdminService, which provide core functionalities accessible by the Presentation Layer. For example, when creating a post, the AdminInterface might call the AdminService to initiate post creation actions.
- **Business Layer**: This layer contains the main business logic, with modules such as:
    - NewsManagement: Manages operations related to creating, updating, and displaying news posts.
    - CommentManagement: Manages comments on posts.
    - CategoryManagement: Organizes news posts into different categories.
    - UserManagement: Manages user information and access.

For the "Create Post" use case, the NewsManagement component is the primary actor in managing the post-creation process, applying business rules, and validating the post's content before it is saved.

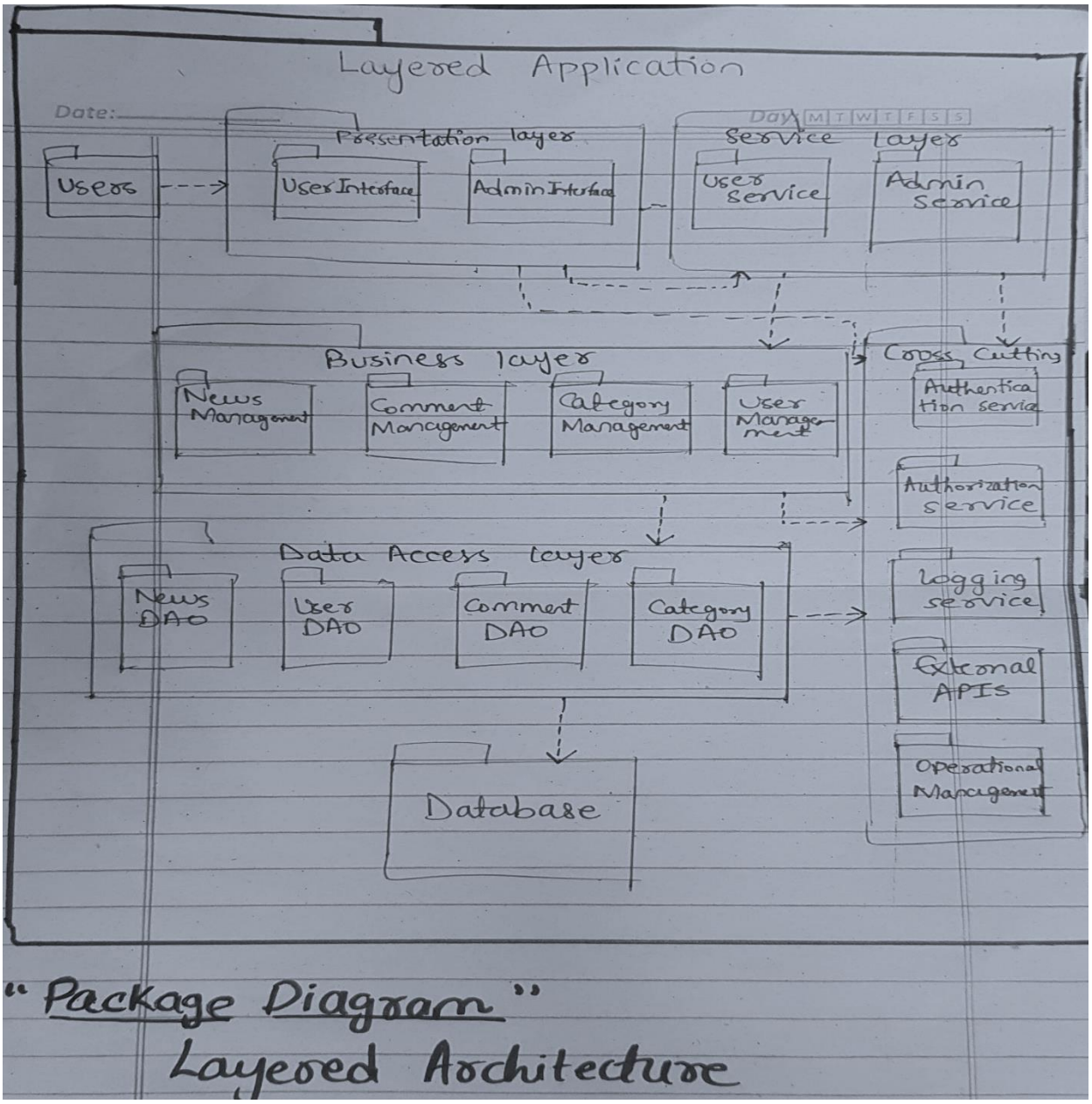
- **Data Access Layer**: This layer is responsible for accessing and modifying data in the database. Each business module has a corresponding Data Access Object (DAO):
    - NewsDAO handles data operations for posts.
    - UsersDAO manages user data.
    - CommentDAO handles comment data.
    - CategoryDAO deals with category data.

For "Create Post," the NewsDAO would interact with the database to store the newly created post.

- **Database**: The database holds all the persistent data related to the news portal, including user information, posts, comments, and categories.
- **Cross-Cutting Concerns**: These components provide services that are needed across the entire system, such as:
    - AuthenticationService and AuthorizationService for securing the system.
    - LoggingService for logging actions and errors.
    - ExternalAPIs for interacting with third-party services.

- OperationalManagement for managing operational aspects.

During the "Create Post" process, AuthenticationService and AuthorizationService would ensure the user has permission to create a post, and LoggingService would log the action.



Code implementation:

- **Presentation Layer (UI)**: The NewsPortalUI class handles user interaction. It gathers input and displays messages to the user, calling the createPost method in NewsPortalApp when the "Create Post" button is clicked.

- **Service Layer (PostService)**: The createPost method in NewsPortalApp calls the PostService to orchestrate business logic and data access. The PostService validates data and then uses the PostRepository to save it.
- **Business Logic Layer (PostBusinessLogic)**: The PostService delegates to PostBusinessLogic to validate the post data, enforcing rules like mandatory fields and content length.
- **Data Access Layer (PostRepository)**: The PostRepository simulates data storage by assigning a unique ID to the post and saving it.

```
import javax.swing.*;
import java.awt.*;

public class NewsPortalUI extends JFrame {
    private JTextField titleField;
    private JTextArea contentArea;
    private JTextField authorField;
    private JButton createPostButton;
    private JLabel messageLabel;

    public NewsPortalUI() {
        setTitle("News Portal - Create Post");
        setSize(500, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 1, 10, 10));

        JLabel titleLabel = new JLabel("Title:");
        titleField = new JTextField();
        panel.add(titleLabel);
```

```
        panel.add(titleField);

        JLabel contentLabel = new JLabel("Content:");
        contentArea = new JTextArea(5, 20);
        panel.add(contentLabel);
        panel.add(new JScrollPane(contentArea));

        JLabel authorLabel = new JLabel("Author:");
        authorField = new JTextField();
        panel.add(authorLabel);
        panel.add(authorField);

        createPostButton = new JButton("Create Post");
        messageLabel = new JLabel("");
        messageLabel.setForeground(Color.RED);

        add(panel, BorderLayout.CENTER);
        add(createPostButton, BorderLayout.SOUTH);
        add(messageLabel, BorderLayout.NORTH);

        createPostButton.addActionListener(e -> handleCreatePost());

        setVisible(true);
    }

    private void handleCreatePost() {
        String title = titleField.getText();
        String content = contentArea.getText();
```

```
        String author = authorField.getText();


        NewsPortalApp app = new NewsPortalApp();
        try {
            app.createPost(title, content, author);
            messageLabel.setText("Post created successfully!");
        } catch (IllegalArgumentException ex) {
            messageLabel.setText("Failed to create post: " + ex.getMessage());
        }
    }


    public static void main(String[] args) {
        SwingUtilities.invokeLater(NewsPortalUI::new);
    }
}
```

Output: