

EGE UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

ADVANCED OBJECT ORIENTED PROGRAMMING
2023–2024

PROJECT REPORT
SOCIAL MEDIA APP
DESIGN PATTERNS

DELIVERY DATE

23.05.2024

PREPARED BY

05210000220, Bengi İlhan

05220000276, Mahmut Karabulut

05220000281, Umut Öztürk

05200000031, Ece Menemencioğlu

Contents

1. Introduction

1.a Overview

1.a.1 Explanation

1.b Design Patterns

1.b.1 Singleton Pattern

1.b.1.1 Purpose

1.b.1.2 Usage in Code

1.b.2 Adapter Pattern

1.b.2.1 Purpose

1.b.2.2 Usage in Code

1.b.3 Observer Pattern

1.b.3.1 Purpose

1.b.3.2 Usage in Code

1.b.4 Memento Pattern

1.b.4.1 Purpose

1.b.4.2 Usage in Code

1.b.5 Decorator Pattern

1.b.5.1 Purpose

1.b.5.2 Usage in Code

2. Code Documentation

2.a FollowManager Class

2.a.1 Code

2.a.2 Explanation

2.b FriendManager Class

2.b.1 Code

2.b.2 Explanation

2.c PostManager Class

2.c.1 Code

2.c.2 Explanation

2.d HomePanel Class

2.d.1 Code

2.d.2 Explanation

3. User Guide

3.a Login

3.a.1 Steps

3.a.2 Screenshot of the app

3.b Home Page

3.b.1 Steps

3.b.2 Screenshot of the app

3.c Profile Page

3.c.1 Steps

3.c.2 Screenshot of the app

3.d Search and Add Friends

3.d.1 Steps

3.d.2 Screenshot of the app

3.e Groups

3.e.1 Steps

3.e.2 Screenshot of the app

4. UML Diagram

Project Report for Social Media Application

Introduction

This report documents the design and implementation of a social media application developed in Java. The application utilizes several design patterns, including Singleton, Adapter, Observer, and Memento, to enhance its functionality, maintainability, and scalability. The report provides a comprehensive overview of the design patterns used, the source code with comments, and a simple user guide with placeholders for screenshots.

Design Patterns Used

Singleton Pattern

The Singleton pattern is used to ensure that a class has only one instance and provides a global point of access to it. This is particularly useful for managing shared resources, such as configuration settings or data management classes.

Purpose:

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it.

Usage in Code:

The UserManager class is implemented as a Singleton to manage user data and operations. This ensures that there is only one instance of UserManager throughout the application, preventing data inconsistency and synchronization issues.

```
package aoopproject;

import java.io.*;
import java.util.*;

public class UserManager {
    private Map<String, User> users; // Map of username to user object
    private User loggedInUser;
    public static final String USERS_DIR = "users/"; // Users data path
    public static final String FRIENDS_FILE = USERS_DIR + "friends.txt"; // Friends data path

    // Singleton instance
    private static UserManager instance;
```

```
// Private constructor to prevent instantiation
private UserManager() {
    users = new HashMap<>();
    loadAllUsers(); // Önce kullanıcıları yükle
    loadAllFriends(); // Daha sonra arkadaşları yükle
}
```

```
// Provides access to the singleton instance of UserManager
public static UserManager getInstance() {
    if (instance == null) {
        instance = new UserManager();
    }
    return instance;
}
```

```
// Adds a user to the user manager and saves the user data.
public void addUser(User user) {
    users.put(user.getUsername(), user);
    saveUser(user);
}
```

```
public User getUser(String username) {
    return users.get(username);
}
```

```
public boolean validateUser(String username, String password) {
    User user = getUser(username);
    if (user != null && user.getPassword().equals(password)) {
        loggedInUser = user;
        return true;
    }
    return false;
}
```

```
public User getLoggedInUser() {
    return loggedInUser;
}
```

```
public List<User> getSearchableUsers() {
    List<User> searchableUsers = new ArrayList<>();
    for (User user : users.values()) {
        if (user.isSearchable()) {
            searchableUsers.add(user);
        }
    }
    return searchableUsers;
}
```

```

private void loadAllUsers() {
    File usersDir = new File(USERS_DIR);
    if (!usersDir.exists()) {
        usersDir.mkdirs();
    }

    File[] userFiles = usersDir.listFiles((dir, name) -> name.endsWith(".txt") &&
!name.endsWith("_friends.txt"));
    if (userFiles != null) {
        for (File userFile : userFiles) {
            loadUser(userFile);
        }
    }
}

private void loadAllFriends() {
    File friendsFile = new File(FRIENDS_FILE);
    if (!friendsFile.exists()) {
        try {
            friendsFile.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    try (BufferedReader reader = new BufferedReader(new FileReader(friendsFile))) {
        String line;
        User currentUser = null;
        while ((line = reader.readLine()) != null) {
            if (line.startsWith("User:")) {
                String username = line.substring(5).trim();
                currentUser = getUser(username);
            } else if (currentUser != null) {
                User friend = getUser(line.trim());
                if (friend != null) {
                    currentUser.getFriendManager().addFriend(friend, false); // Arkadaş eklerken
kaydetme işlemi yapılmıyor
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void loadUser(File userFile) {
    try (BufferedReader reader = new BufferedReader(new FileReader(userFile))) {
        String line = reader.readLine();
    }
}

```

```

        if (line != null) {
            String[] parts = line.split(",");
            if (parts.length >= 4) {
                String username = parts[0];
                String password = parts[1];
                String fullName = parts[2];
                boolean isSearchable = Boolean.parseBoolean(parts[3]);
                User user = new User(username, password, fullName, isSearchable);
                users.put(username, user);
            }
        }

        while ((line = reader.readLine()) != null) {
            if (line.equals("POST")) {
                String postContent = reader.readLine();
                String timestampString = reader.readLine();
                if (postContent == null || timestampString == null ||
timestampString.isEmpty()) {
                    continue;
                }
                long timestamp = Long.parseLong(timestampString);
                //long timestamp = Long.parseLong(reader.readLine());
                Post post = new Post(postContent, user);
                user.getPostManager().addPost(post);
            } else if (line.equals("FRIEND_REQUEST_SENT")) {
                String receiverUsername = reader.readLine();
                User receiver = getUser(receiverUsername);
                if (receiver != null) {
                    user.getFriendManager().addSentFriendRequest(new
FriendRequest(user, receiver));
                }
            } else if (line.equals("FRIEND_REQUEST_RECEIVED")) {
                String senderUsername = reader.readLine();
                User sender = getUser(senderUsername);
                if (sender != null) {
                    user.getFriendManager().addReceivedFriendRequest(new
FriendRequest(sender, user));
                }
            } else if (line.equals("FOLLOW")) {
                String followeeUsername = reader.readLine();
                User followee = getUser(followeeUsername);
                if (followee != null) {
                    user.getFollowManager().follow(user, followee);
                }
            } else if (line.equals("GROUP")) {
                String groupName = reader.readLine();
                Group group = new Group(groupName);
                int memberCount = Integer.parseInt(reader.readLine());
                for (int i = 0; i < memberCount; i++) {
                    String memberUsername = reader.readLine();

```

```

        User member = getUser(memberUsername);
        if (member != null) {
            group.addMember(member);
        }
    }

    int postCount = Integer.parseInt(reader.readLine());
    for (int i = 0; postCount > 0 && i < postCount; i++) {
        String postContent = reader.readLine();
        long timestamp = Long.parseLong(reader.readLine());
        Post post = new Post(postContent, user);
        group.getPostManager().addPost(post);
    }
    user.getGroupManager().addGroup(group);
}
}
}
}
} catch (IOException e) {
    System.out.println("Error loading user from file: " + userFile.getName());
}
}
}

```

```

public void saveUser(User user) {
    File userFile = new File(USERS_DIR + user.getUsername() + ".txt");
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(userFile))) {
        writer.write(user.getUsername() + "," + user.getPassword() + "," +
user.getFullName() + "," + user.isSearchable());
        writer.newLine();
        for (Post post : user.getPostManager().getPosts()) {
            writer.write("POST");
            writer.newLine();
            writer.write(post.getContent());
            writer.newLine();
            writer.write(String.valueOf(post.getTimestamp().getTime()));
            writer.newLine();
        }
        for (FriendRequest request : user.getFriendManager().getSentFriendRequests()) {
            writer.write("FRIEND_REQUEST_SENT");
            writer.newLine();
            writer.write(request.getReceiver().getUsername());
            writer.newLine();
        }
        for (FriendRequest request :
user.getFriendManager().getReceivedFriendRequests()) {
            writer.write("FRIEND_REQUEST_RECEIVED");
            writer.newLine();
            writer.write(request.getSender().getUsername());
            writer.newLine();
        }
    }
}

```



```

    }
    for (User followee : user.getFollowManager().getFollowing()) {
        writer.write("FOLLOW");
        writer.newLine();
        writer.write(followee.getUsername());
        writer.newLine();
    }
    for (Group group : user.getGroupManager().getGroups()) {
        writer.write("GROUP");
        writer.newLine();
        writer.write(group.getName());
        writer.newLine();
        writer.write(String.valueOf(group.getMembers().size()));
        writer.newLine();
        for (User member : group.getMembers()) {
            writer.write(member.getUsername());
            writer.newLine();
        }
        List<Post> groupPosts = group.getPostManager().getPosts();
        writer.write(String.valueOf(groupPosts.size()));
        writer.newLine();
        for (Post post : groupPosts) {
            writer.write(post.getContent());
            writer.newLine();
            writer.write(String.valueOf(post.getTimestamp().getTime()));
            writer.newLine();
        }
    }
} catch (IOException e) {
    System.out.println("Error saving user: " + e.getMessage());
}
}

```

```

// New helper method to save both users
public void saveBothUsers(User user1, User user2) {
    saveUser(user1);
    saveUser(user2);
}

```

```

// New method to load a user by username
public User loadUserByUsername(String username) {
    if (users.containsKey(username)) {
        return users.get(username);
    }
}

```

```

File userFile = new File(USERS_DIR + username + ".txt");
if (userFile.exists()) {
    try (BufferedReader reader = new BufferedReader(new FileReader(userFile))) {

```

```

        String line = reader.readLine();
        if (line != null) {
            String[] parts = line.split(",");
            if (parts.length >= 4) {
                String password = parts[1];
                String fullName = parts[2];
                boolean isSearchable = Boolean.parseBoolean(parts[3]);
                User user = new User(username, password, fullName, isSearchable);
                users.put(username, user);
            }
        }

        while ((line = reader.readLine()) != null) {
            if (line.equals("POST")) {
                String postContent = reader.readLine();
                long timestamp = Long.parseLong(reader.readLine());
                Post post = new Post(postContent, user);
                user.getPostManager().addPost(post);
            } else if (line.equals("FRIEND")) {
                String friendUsername = reader.readLine();
                User friend = getUser(friendUsername);
                if (friend != null) {
                    user.getFriendManager().addFriend(friend, false);
                }
            } else if (line.equals("FRIEND_REQUEST_SENT")) {
                String receiverUsername = reader.readLine();
                User receiver = getUser(receiverUsername);
                if (receiver != null) {
                    user.getFriendManager().addSentFriendRequest(new
FriendRequest(user, receiver));
                }
            } else if (line.equals("FRIEND_REQUEST_RECEIVED")) {
                String senderUsername = reader.readLine();
                User sender = getUser(senderUsername);
                if (sender != null) {
                    user.getFriendManager().addReceivedFriendRequest(new
FriendRequest(sender, user));
                }
            } else if (line.equals("FOLLOW")) {
                String followeeUsername = reader.readLine();
                User followee = getUser(followeeUsername);
                if (followee != null) {
                    user.getFollowManager().follow(user, followee);
                }
            } else if (line.equals("GROUP")) {
                String groupName = reader.readLine();
                Group group = new Group(groupName);
                int memberCount = Integer.parseInt(reader.readLine());
                for (int i = 0; i < memberCount; i++) {
                    String memberUsername = reader.readLine();

```

```

        User member = getUser(memberUsername);
        if (member != null) {
            group.addMember(member);
        }
    }

    int postCount = Integer.parseInt(reader.readLine());
    for (int i = 0; postCount > 0 && i < postCount; i++) {
        String postContent = reader.readLine();
        long timestamp = Long.parseLong(reader.readLine());
        Post post = new Post(postContent, user);
        group.getPostManager().addPost(post);
    }
    user.getGroupManager().addGroup(group);
}

return user;
}
} catch (IOException e) {
    System.out.println("Error loading user from file: " + userFile.getName());
}
}
return null;
}

```

```

public void saveAllUsers() {
    for (User user : users.values()) {
        saveUser(user);
    }
    saveAllFriends(); // Tüm arkadaşlık ilişkilerini kaydet
}

```

```

private void saveAllFriends() {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(FRIENDS_FILE))) {
        for (User user : users.values()) {
            writer.write("User:" + user.getUsername());
            writer.newLine();
            for (User friend : user.getFriendManager().getFriends()) {
                writer.write(friend.getUsername());
                writer.newLine();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Adapter Pattern

The Adapter pattern converts the interface of a class into another interface that a client expects. This allows classes with incompatible interfaces to work together.

Purpose:

The Adapter pattern is used to convert the interface of a class into another interface that clients expect. It allows classes with incompatible interfaces to work together.

Usage in Code:

The UserAdapter class is used to adapt the User class to provide a simplified view of the user's data. This is particularly useful when displaying user information in different contexts, such as in a profile view or search results.

```
package aoopproject;

/**
 * Adapter class to adapt the User class to a different interface.
 * This class provides a simplified view of the User data.
 */
public class UserAdapter {
    private User user;

    public UserAdapter(User user) { // Constructor to initialize the adapter with a User object
        this.user = user;
    }

    /**
     * Gets the adapted user data as a formatted string.
     * This method adapts the User object to a new interface.
     */
    public String getUserData() {
        return "Username: " + user.getUsername() + "\nFull Name: " + user.getFullName() +
            "\nFollowers: " + user.getFollowManager().getFollowers().size() + "\nFollowing: " +
            user.getFollowManager().getFollowing().size();
    }
}
```

Observer Pattern

The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This pattern is useful for implementing distributed event-handling systems.

Purpose:

The Observer pattern is used to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Usage in Code:

The Observable class and Observer interface are used to implement this pattern for the PostManager class to notify observers of changes to posts. This ensures that any changes to the posts, such as new posts or likes, are reflected in all views that display posts.

```

package aoopproject;

/**
 * Observable class for the Observer design pattern.
 * Manages a list of observers and notifies them of changes.
 */
import java.util.ArrayList;
import java.util.List;

public class Observable {
    // List of observers that are watching for changes.
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) { // Adds an observer to the list.
        observers.add(observer);
    }

    public void removeObserver(Observer observer) { // Removes an observer from the list.
        observers.remove(observer);
    }

    public void notifyObservers() { // Notifies all observers of a change.
        for (Observer observer : observers) {
            observer.update();
        }
    }
}

/**
 * Observer interface for the Observer design pattern.
 * Classes that want to be notified of changes should implement this interface.
 */
public interface Observer {
    void update();
}

```

Implementation in PostManager:

```

package aoopproject;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

```

```

/**
 * Manages posts for a user or group.
 * Extends Observable to notify observers of changes to the post list.
 * Supports saving and restoring post history using the Memento design pattern.
 */
public class PostManager extends Observable {
    private List<Post> posts;
    private List<PostMemento> postHistory; // List of mementos for posts

    public PostManager() {
        posts = new ArrayList<>();
        postHistory = new ArrayList<>();
    }

    public void addPost(Post post) { // Adds a post and notifies observers of the change
        posts.add(post);
        postHistory.add(post.saveToMemento());
        notifyObservers();
    }

    public List<Post> getPosts() {
        return new ArrayList<>(posts);
    }

    public List<PostMemento> getPostHistory() { // Gets a list of post mementos.
        return new ArrayList<>(postHistory);
    }

    public void likePost(Post post) { // Likes a post and notifies observers of the change.
        post.incrementLikes();
        postHistory.add(post.saveToMemento());
        notifyObservers();
    }

    // Gets a list of posts sorted by timestamp in descending order.
    public List<Post> getSortedPosts() {
        List<Post> sortedPosts = new ArrayList<>(posts);
        sortedPosts.sort(Comparator.comparing(Post::getTimestamp).reversed());
        return sortedPosts;
    }
}

```

Memento Pattern

The Memento pattern captures and externalizes an object's internal state so that the object can be restored to this state later. This pattern is useful for implementing undo mechanisms or saving state for future use.

Purpose:

The Memento pattern is used to capture and externalize an object's internal state so that the object can be restored to this state later.

Usage in Code:

The PostMemento class is used to store the state of a Post object. This allows the PostManager to save and restore the state of posts, enabling features such as undoing changes or maintaining post history.

```
package aoopproject;

import java.util.Date;

/**
 * Memento class to store the state of a Post object.
 * This class is used to save and restore the state of a Post.
 */
public class PostMemento {
    private final String content;
    private final User user;
    private final int likeCount;
    private final Date timestamp;

    public PostMemento(String content, User user, int likeCount, Date timestamp) {
        this.content = content;
        this.user = user;
        this.likeCount = likeCount;
        this.timestamp = new Date(timestamp.getTime()); // Immutable date
    }

    // Getters to retrieve the state of the Post object
    public String getContent() {
        return content;
    }

    public User getUser() {
        return user;
    }

    public int getLikeCount() {
        return likeCount;
    }

    public Date getTimestamp() {
        return new Date(timestamp.getTime()); // Immutable date
    }
}
```

Implementation in Post:

```
package aoopproject;

import java.util.Date;

/**
 * Represents a post made by a user.
 * Supports saving and restoring its state using the Memento design pattern.
 */
public class Post {
    private String content;
    private User user;
    private int likeCount;
    private Date timestamp;

    public Post(String content, User user) {
        this.content = content;
        this.user = user;
        this.likeCount = 0;
        this.timestamp = new Date();
    }

    public String getContent() {
        return content;
    }

    public User getUser() {
        return user;
    }

    public int getLikeCount() {
        return likeCount;
    }

    public void incrementLikes() {
        likeCount++;
    }

    public Date getTimestamp() {
        return new Date(timestamp.getTime());
    }

    // Saves the current state of the post to a memento.
    public PostMemento saveToMemento() {
        return new PostMemento(content, user, likeCount, timestamp);
    }
}
```



```

// Restores the state of the post from a memento.
public void restoreFromMemento(PostMemento memento) {
    this.content = memento.getContent();
    this.user = memento.getUser();
    this.likeCount = memento.getLikeCount();
    this.timestamp = memento.getTimestamp();
}
}

```

Source Code

FollowManager Class

```

package aoopproject;

import java.util.ArrayList;
import java.util.List;

public class FollowManager {
    private List<User> followers;
    private List<User> following;

    public FollowManager() {
        this.followers = new ArrayList<>();
        this.following = new ArrayList<>();
    }

    public void follow(User follower, User followee) {
        if (!following.contains(followee)) {
            following.add(followee);
            followee.getFollowManager().addFollower(follower);
            System.out.println(follower.getUsername() + " is now following " +
followee.getUsername());
        }
    }

    public void addFollower(User follower) {
        if (!followers.contains(follower)) {
            followers.add(follower);
        }
    }

    public List<User> getFollowers() {
        return followers;
    }

    public List<User> getFollowing() {
        return following;
    }
}

```

```
}  
}
```

FriendManager Class

```
package aoopproject;  
  
import java.io.*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class FriendManager {  
    private User owner;  
    private List<FriendRequest> sentFriendRequests;  
    private List<FriendRequest> receivedFriendRequests;  
    private List<User> friends;  
  
    public FriendManager(User owner) {  
        this.owner = owner;  
        sentFriendRequests = new ArrayList<>();  
        receivedFriendRequests = new ArrayList<>();  
        friends = new ArrayList<>();  
    }  
  
    public List<User> getFriends() {  
        return friends;  
    }  
  
    public List<FriendRequest> getSentFriendRequests() {  
        return new ArrayList<>(sentFriendRequests);  
    }  
  
    public List<FriendRequest> getReceivedFriendRequests() {  
        return new ArrayList<>(receivedFriendRequests);  
    }  
  
    public void sendFriendRequest(User sender, User receiver) {  
        FriendRequest request = new FriendRequest(sender, receiver);  
        sentFriendRequests.add(request);  
        receiver.getFriendManager().addReceivedFriendRequest(request);  
    }  
  
    public void addSentFriendRequest(FriendRequest request) {  
        sentFriendRequests.add(request);  
    }  
  
    public void addReceivedFriendRequest(FriendRequest request) {  
        receivedFriendRequests.add(request);  
    }  
}
```

```

    }

    public void acceptFriendRequest(FriendRequest request, UserManager userManager) {
        User sender = request.getSender();
        User receiver = request.getReceiver();
        addFriend(sender);
        sender.getFriendManager().addFriend(receiver);
        receivedFriendRequests.remove(request);
        sender.getFriendManager().getSentFriendRequests().remove(request);

        // Save both users
        userManager.saveBothUsers(sender, receiver);
    }

    public void removeFriendRequest(FriendRequest request) {
        receivedFriendRequests.remove(request);
    }

    public void addFriend(User friend) {
        addFriend(friend, true);
    }

    public void addFriend(User friend, boolean saveToFile) {
        if (!friends.contains(friend)) {
            friends.add(friend);
            if (saveToFile) {
                saveFriendToFile(friend);
            }
        }
    }

    private void saveFriendToFile(User friend) {
        try (BufferedWriter writer = new BufferedWriter(new
        FileWriter(UserManager.FRIENDS_FILE, true))) {
            writer.write("User:" + owner.getUsername());
            writer.newLine();
            writer.write(friend.getUsername());
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void loadFriendsFromFile() {
        // Loading friends will be handled by UserManager, this method can be left empty
    }

```

PostManager Class

```
package aoopproject;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * Manages posts for a user or group.
 * Extends Observable to notify observers of changes to the post list.
 * Supports saving and restoring post history using the Memento design pattern.
 */
public class PostManager extends Observable {
    private List<Post> posts;
    private List<PostMemento> postHistory; // List of mementos for posts

    public PostManager() {
        posts = new ArrayList<>();
        postHistory = new ArrayList<>();
    }

    public void addPost(Post post) { // Adds a post and notifies observers of the change
        posts.add(post);
        postHistory.add(post.saveToMemento());
        notifyObservers();
    }

    public List<Post> getPosts() {
        return new ArrayList<>(posts);
    }

    public List<PostMemento> getPostHistory() { // Gets a list of post mementos.
        return new ArrayList<>(postHistory);
    }

    public void likePost(Post post) { // Likes a post and notifies observers of the change.
        post.incrementLikes();
        postHistory.add(post.saveToMemento());
        notifyObservers();
    }

    // Gets a list of posts sorted by timestamp in descending order.
    public List<Post> getSortedPosts() {
        List<Post> sortedPosts = new ArrayList<>(posts);
        sortedPosts.sort(Comparator.comparing(Post::getTimestamp).reversed());
        return sortedPosts;
    }
}
```

```
}
```

HomePanel Class

```
package aoopproject;

import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Panel for displaying the home screen, including posts from friends and the user.
 * Implements Observer to update the feed when the PostManager notifies of changes.
 */
public class HomePanel extends JPanel implements Observer {
    private MainFrame mainFrame;
    private UserManager userManager;
    private JTextArea postArea;
    private JButton postButton;
    private JButton logoutButton;
    private JPanel feedPanel; // feedArea yerine JPanel kullanacağız
    private JButton profileButton;
    private JButton searchButton;
    private JButton friendRequestsButton;
    private JLabel homeLabel;

    public HomePanel(MainFrame mainFrame, UserManager userManager) {
        this.mainFrame = mainFrame;
        this.userManager = userManager;

        setLayout(new BorderLayout());

        setBackground(new Color(173, 216, 230)); // Light blue

        homeLabel = new JLabel("Home");
        homeLabel.setFont(new Font("Arial", Font.BOLD, 24));
        homeLabel.setHorizontalAlignment(SwingConstants.CENTER);
        homeLabel.setForeground(new Color(0, 102, 204)); // Dark blue text

        postArea = new JTextArea(3, 50);
        postArea.setBackground(Color.WHITE);
    }
}
```

```
        postArea.setBorder(BorderFactory.createLineBorder(new Color(0, 102, 204))); // Dark blue border
```

```
        postButton = new JButton("Post");  
        postButton.setBackground(new Color(0, 102, 204)); // Dark blue  
        postButton.setForeground(Color.WHITE);
```

```
        logoutButton = new JButton("Logout");  
        logoutButton.setBackground(new Color(0, 102, 204)); // Dark blue  
        logoutButton.setForeground(Color.WHITE);
```

```
        profileButton = new JButton("Profile");  
        profileButton.setBackground(new Color(0, 102, 204)); // Dark blue  
        profileButton.setForeground(Color.WHITE);
```

```
        searchButton = new JButton("Search");  
        searchButton.setBackground(new Color(0, 102, 204)); // Dark blue  
        searchButton.setForeground(Color.WHITE);
```

```
        friendRequestsButton = new JButton("Friend Requests");  
        friendRequestsButton.setBackground(new Color(0, 102, 204)); // Dark blue  
        friendRequestsButton.setForeground(Color.WHITE);
```

```
        JPanel postPanel = new JPanel(new BorderLayout());  
        postPanel.setBackground(new Color(173, 216, 230)); // Light blue  
        postPanel.add(new JScrollPane(postArea), BorderLayout.CENTER);  
        postPanel.add(postButton, BorderLayout.EAST);
```

```
        JPanel buttonPanel = new JPanel();  
        buttonPanel.setBackground(new Color(173, 216, 230)); // Light blue  
        buttonPanel.add(profileButton);  
        buttonPanel.add(searchButton);  
        buttonPanel.add(friendRequestsButton);  
        buttonPanel.add(logoutButton);
```

```
        JPanel topPanel = new JPanel(new BorderLayout());  
        topPanel.setBackground(new Color(173, 216, 230)); // Light blue  
        topPanel.add(homeLabel, BorderLayout.NORTH);  
        topPanel.add(postPanel, BorderLayout.CENTER);
```

```
        feedPanel = new JPanel();  
        feedPanel.setLayout(new BoxLayout(feedPanel, BoxLayout.Y_AXIS));  
        feedPanel.setBackground(Color.WHITE);  
        feedPanel.setBorder(BorderFactory.createLineBorder(new Color(0, 102, 204))); // Dark blue  
        JScrollPane feedScrollPane = new JScrollPane(feedPanel);  
        feedScrollPane.setPreferredSize(new Dimension(800, 600));
```

```
topPanel.add(feedScrollPane, BorderLayout.SOUTH); // feedArea yerine feedPanel  
ekledik
```

```
JPanel mainContent = new JPanel(new BorderLayout());  
mainContent.setBackground(new Color(173, 216, 230)); // Light blue  
mainContent.add(topPanel, BorderLayout.CENTER);  
mainContent.add(buttonPanel, BorderLayout.SOUTH);
```

```
JScrollPane scrollPane = new JScrollPane(mainContent);
```

```
//scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS); //  
delete for extra scrollbar
```

```
scrollPane.getViewport().setBackground(new Color(173, 216, 230)); // Light blue  
background
```

```
add(scrollPane, BorderLayout.CENTER);
```

```
postButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        String content = postArea.getText();  
        if (!content.isEmpty()) {  
            User loggedInUser = userManager.getLoggedInUser();  
            Post newPost = new Post(content, loggedInUser);  
            loggedInUser.getPostManager().addPost(newPost);  
            postArea.setText("");  
            userManager.saveUser(loggedInUser); // Save user after adding a post  
            updateFeed(); // Update feed after adding a post  
        }  
    }  
});
```

```
logoutButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        mainFrame.switchTo("Login");  
    }  
});
```

```
profileButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        mainFrame.getProfilePanel().displayProfile(userManager.getLoggedInUser());  
        mainFrame.switchTo("Profile");  
    }  
});
```

```
searchButton.addActionListener(new ActionListener() {
```

```

        @Override
        public void actionPerformed(ActionEvent e) {
            mainFrame.switchTo("Search");
        }
    });

    friendRequestsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            displayFriendRequests();
        }
    });

    if (userManager.getLoggedInUser() != null) {
        // Add HomePanel as an observer to the logged-in user's PostManager.
        userManager.getLoggedInUser().getPostManager().addObserver(this);
        updateFeed(); // Download post
    }
}

@Override
public void update() { // Updates the feed to display latest post when notifies observer
change
    updateFeed();
}

public void updateFeed() {
    feedPanel.removeAll(); // clear feedPanel
    User loggedInUser = userManager.getLoggedInUser();
    if (loggedInUser != null) {
        List<Post> allPosts = new ArrayList<>();
        for (User friend : loggedInUser.getFriendManager().getFriends()) {
            allPosts.addAll(friend.getPostManager().getPosts());
        }
        allPosts.addAll(loggedInUser.getPostManager().getPosts());
        Collections.sort(allPosts, Comparator.comparing(Post::getTimestamp).reversed());

        for (Post post : allPosts) {
            addPostToFeed(post);
        }

        feedPanel.revalidate();
        feedPanel.repaint();
    }
}

private void addPostToFeed(Post post) {
    JPanel postPanel = new JPanel(new BorderLayout());

```



```
        JLabel postLabel = new JLabel(post.getUser().getFullName() + ": " + post.getContent()
+ " (" + post.getLikeCount() + " likes)");
        JButton likeButton = new JButton("Like");
```

```
        likeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                post.incrementLikes();
                userManager.saveUser(post.getUser());
                postLabel.setText(post.getUser().getFullName() + ": " + post.getContent() + " (" +
post.getLikeCount() + " likes)");
                feedPanel.revalidate();
                feedPanel.repaint();
            }
        });
```

```
        postPanel.add(postLabel, BorderLayout.CENTER);
        postPanel.add(likeButton, BorderLayout.EAST);
        postPanel.setBorder(BorderFactory.createLineBorder(Color.GRAY));
        feedPanel.add(postPanel);
    }
```

```
    public void displayFriendRequests() {
        User loggedInUser = userManager.getLoggedInUser();
        if (loggedInUser != null) {
            List<FriendRequest> friendRequests =
loggedInUser.getFriendManager().getReceivedFriendRequests();
            if (friendRequests.isEmpty()) {
                JOptionPane.showMessageDialog(mainFrame, "No friend requests.");
            } else {
                for (FriendRequest request : new ArrayList<>(friendRequests)) {
                    int result = JOptionPane.showConfirmDialog(mainFrame, "Accept friend request
from " + request.getSender().getUsername() + "?", "Friend Request",
JOptionPane.YES_NO_OPTION);
                    if (result == JOptionPane.YES_OPTION) {
                        loggedInUser.getFriendManager().acceptFriendRequest(request,
userManager);
                        loggedInUser.getFriendManager().removeFriendRequest(request);
                        updateFriendsList(loggedInUser, request.getSender());
                    }
                }
            }
        }
    }
```

```
    private void updateFriendsList(User user1, User user2) {
        // Update the user objects in memory
        user1.getFriendManager().addFriend(user2);
    }
```

```

        user2.getFriendManager().addFriend(user1);

        // Save the updated user objects
        userManager.saveBothUsers(user1, user2);

        // If either user is the logged-in user, update the UI
        if (userManager.getLoggedInUser().equals(user1) ||
            userManager.getLoggedInUser().equals(user2)) {
            updateFeed();
        }
    }
}

```

Decorator Pattern

Purpose:

The Decorator pattern allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class. This pattern is useful for adhering to the Single Responsibility Principle as it allows functionality to be divided between classes with unique areas of concern.

Usage in Code:

The ScrollDecorator class is used to add scrolling capability to various panels in the application without modifying the existing panel classes. This ensures that the panels can be decorated with scrolling functionality as needed.

```

package aoopproject;

import javax.swing.*;

/**
 * This class uses the Decorator design pattern to enhance the functionality of a
 * JComponent.
 */
public class ScrollDecorator {
    /**
     * Adds a JScrollPane around the given JComponent.
     * This method decorates the given component with scrolling capability.
     */
    public static JScrollPane decorate(JComponent component) {
        JScrollPane scrollPane = new JScrollPane(component);

        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        return scrollPane;
    }
}

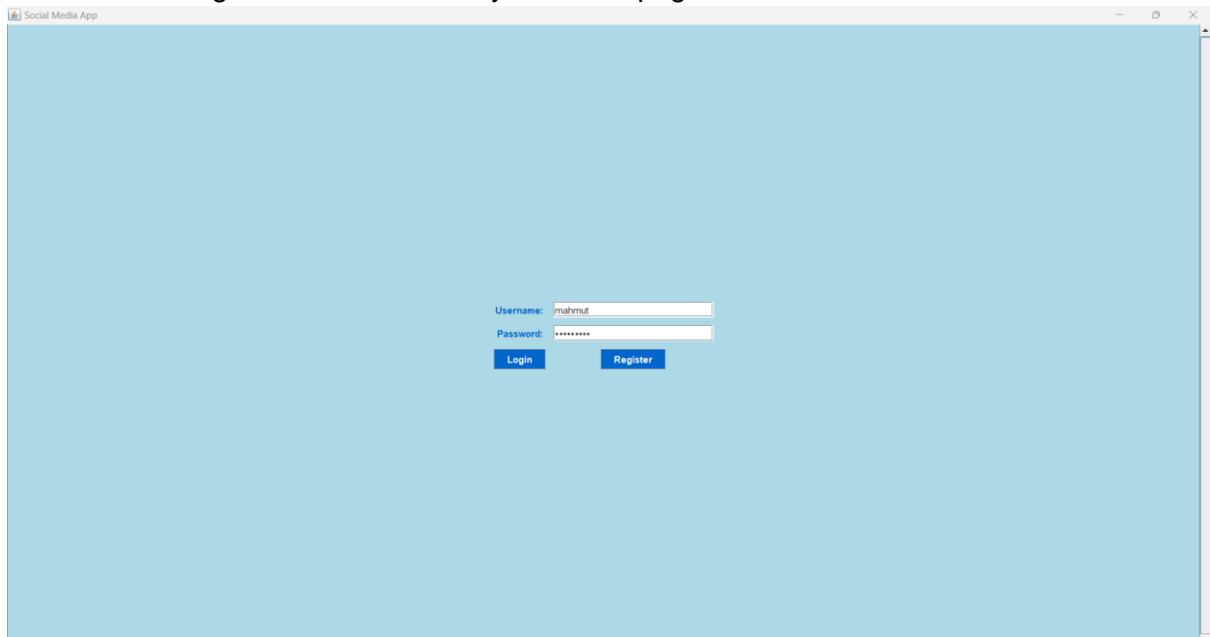
```

User Guide

Login

Open the application and enter your username and password.

Click on the "Login" button to access your home page.



A screenshot of a web browser window titled "Social Media App". The page has a light blue background. In the center, there is a login form with two input fields: "Username:" containing the text "mahmut" and "Password:" containing seven asterisks. Below these fields are two blue buttons: "Login" and "Register".

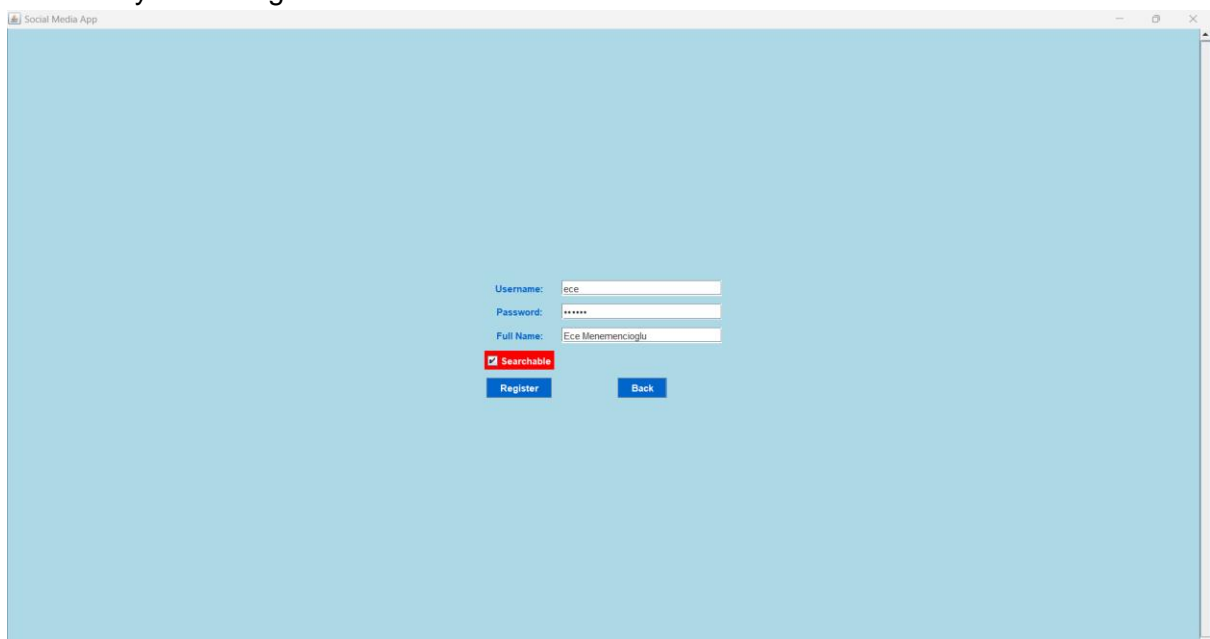
Resgister

Click on the "Register" button to register.

Enter your username, password and your full name.

If you click the searchable button, your friend can send you a friend request. Otherwise they can see your profile.

After that you can login.



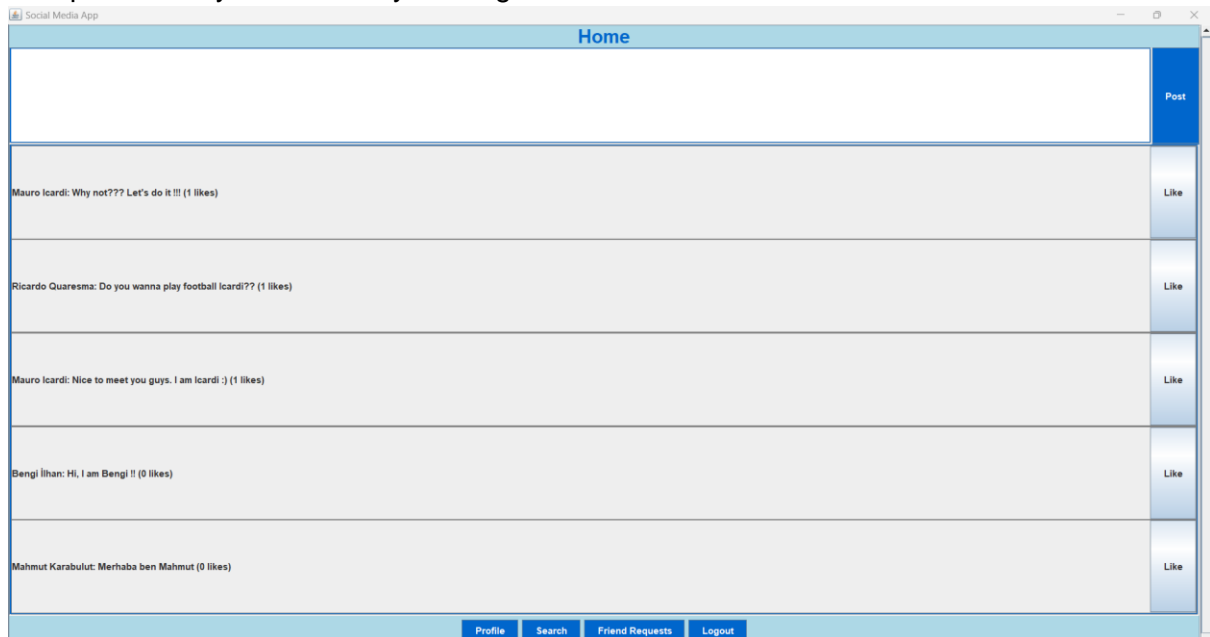
A screenshot of a web browser window titled "Social Media App". The page has a light blue background. In the center, there is a registration form with three input fields: "Username:" containing the text "ece", "Password:" containing seven asterisks, and "Full Name:" containing the text "Ece Menemencioglu". Below these fields is a red checkbox labeled "Searchable" which is checked. At the bottom of the form are two blue buttons: "Register" and "Back".

Home Page

View posts from your friends.

Create a new post using the text area and "Post" button.

Like posts from your friends by clicking the "Like" button.

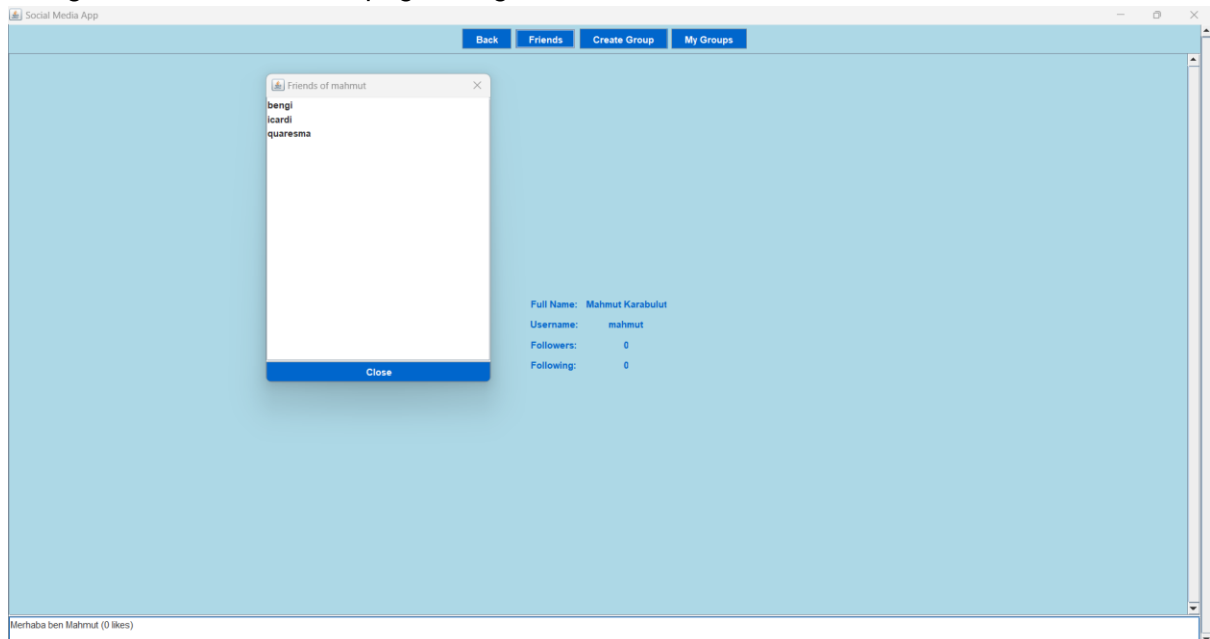


Profile Page

View your profile information, including your full name, username, followers, and following.

See all your posts.

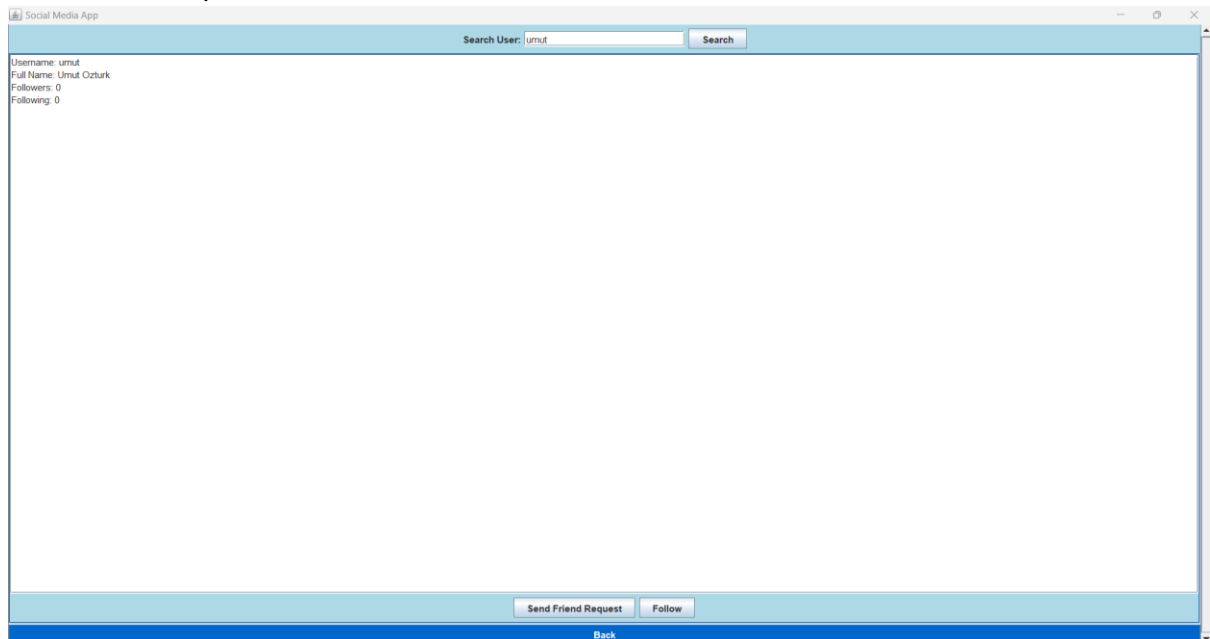
Navigate back to the home page using the "Back" button.



Search and Add Friends

Use the search feature to find users by username or full name.

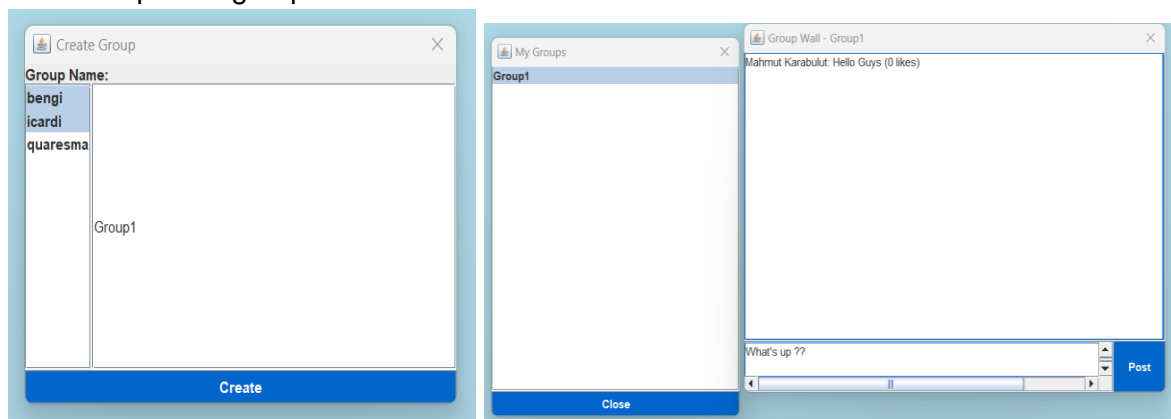
Send friend requests or follow users from the search results.



Groups

Create and manage groups.

View and post in group walls.



UML Diagram