

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**ADVANCED OBJECT ORIENTED PROGRAMMING**

**2023–2024 SPRING SEMESTER**

**PROJECT REPORT**

**(Facebook-like social networking software)**

**DELIVERY DATE**

23/05/2024

**PREPARED BY**

05210000297, Ömer Karaduman

05210000194, Melih Kumru

05210000278, İsmail Erkan

05220000981, Burak Taşkın

Contents

UML………………………………………………………………………………………………………………………………………………..

Explanation of Each Class…………………………………………………………………………….………….….…………..3

1) public class AddFriend;.…………………………………………………………………………….……….……………….3

2) public class CreateGroup; .………………………………………………………………………………….………..…….6

3) public class CreatePost; .……………………………………………………….……………………………………..…….6

4) public class CreateUser; .……………………………………………………….…………….……………………………..7

5) public class GenerateTimeline; .………………………………………………………………….………………………8

6) public class JoinGroup; .………………………………………………………………………………………….….........10

7) public class ReadAllGroupMembers; .………………………………………………………………………………11

8) public class ReadAllGroups; .………………………………………………………………….…………………………12

9) public class ReadAllUsers;.……………………………………………………………………..…………………………14

10) public class ReadUser;.……………………………………………………………………………………………………16

11) public class ReadUserByID;……………………………………………………………………………………………..18

12) public class RemoveFriend; .………………………………………………………………………….………………..18

13) public class UpdateUserPrivacy; .…………………………………………………………………………………….19

14) public class Database;……….……………………………………………..……………………………………………..19

15) public class DatabaseFacade;.……………………………………………………..………………………………….20

16) public class Group;…………………………………………………………………………………..……..……………….21

17) public class Post;…………………………………………………………………………...………………………….……..22

18) public class User;…………………………………………………………………………………………….……………....23

Design Patterns……………………………………………………………………………………………………………………..

Test Cases/Screenshots………………………………………………………………………………………………………….26

1) Register Screen………………..………………………………………………………………………………………………..26

2) Login Screen..……………………………………………………………………………………………………………………28

3) Post Screen…………………………..…………………………………………………………………………………………29

4) Add friends / Friends Screen………………………………………………………………………………………….29

5) Group Screen…………………………………………………………………………………………………………………….30

# Explanation of Each Class

# 1. public class AddFriend

# **Explanation**: The AddFriend class is part of the Controller package and is responsible for handling the logic associated with adding a friend for a user in an application. It interacts with the DatabaseFacade to execute the necessary database operations and uses the Alert view to notify the user in case of errors.

### 1.Source Code

package Controller;

import Model.DatabaseFacade;

import Model.User;

import View.Alert;

import java.sql.SQLException;

public class AddFriend {

private User user,f;

private DatabaseFacade databaseFacade;

// This method allows the user to add a friend.

// It returns true if the operation is successful, false otherwise.

public AddFriend(User user,DatabaseFacade databaseFacade,User f){

this.user= user;

this.databaseFacade = databaseFacade;

this.f = f;

}

public boolean isAdded(){

boolean added = false;

String insert ="INSERT INTO friends(`User`, `Friend`) VALUES ('"+user.getID()+"','"+f.getID()+"')";

try {

databaseFacade.execute(insert);

added = true;

} catch (SQLException e){

new Alert(e.getMessage(), null); // In case of an error, show an alert to the user.

added = false;

}

return added;

}

}

## 

## 2. public class CreateGroup;

**Explanation:** The CreateGroup class is part of the Controller package and is responsible for handling the creation of a new group within the application. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user about the success or failure of the group creation process.

### 2.Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Group;

import Model.User;

import View.Alert;

import java.sql.SQLException;

public class CreateGroup {

private User u;

private DatabaseFacade databaseFacade;

private Group g;

// Constructor for CreateGroup class

public CreateGroup(Group g, DatabaseFacade databaseFacade,User mainUser){

this.g = g;

this.databaseFacade = databaseFacade;

this.u = mainUser;

}

// Method to create a new group

public void create(){

// SQL query to insert the group name into the groups table

String insert = "INSERT INTO groups (Name) VALUES ('"

+ g.getName() + "')" ;

try{

databaseFacade.execute(insert);

new Alert("New Group is Created!",null);

}

catch(SQLException e){

// If an SQL exception occurs, display the error message

new Alert(e.getMessage(), null);

System.out.println(e.getMessage());

}

}

}

## 3. public class CreatePost;

# **Explanation**: The CreatePost class is part of the Controller package and is responsible for handling the creation and posting of new posts within the application. It interacts with the DatabaseFacade to execute the necessary database operations and uses the Alert view to notify the user in case of errors.

# 3.Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Post;

import View.Alert;

import java.sql.SQLException;

public class CreatePost {

private Post post;

private DatabaseFacade databaseFacade;

// Constructor for CreatePost class.

public CreatePost(Post post, DatabaseFacade databaseFacade){

this.post = post;

this.databaseFacade = databaseFacade;

}

// Method to create and post a new post.

// Returns true if the post is successfully created and posted, false otherwise.

public boolean posted(){

boolean posted = false;

try {

// SQL query to insert a new post into the database.

String insert = "INSERT INTO `posts`(`Content`, `User`, `DateTime`) VALUES ('"+post.getContent()+"','"+post.getUser().getID()+"','"+post.getDateTimeToString()+"')";

databaseFacade.execute(insert);

posted = true;

} catch(SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

posted = false;

}

return posted; // Return whether the post was successfully created and posted or not.

}

}

## 4. public class CreateUser;

# **Explanation**: The CreateUser class is part of the Controller package and is responsible for handling the creation of new users, checking if an email is already used, and retrieving user details from the database. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

# 4. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.User;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

import java.util.ArrayList;

public class CreateUser {

private User u;

private DatabaseFacade databaseFacade;

// Constructor for CreateUser class.

public CreateUser(User u, DatabaseFacade databaseFacade){

this.u = u;

this.databaseFacade = databaseFacade;

}

// Method to create a new user.

public void create(){

// SQL query to insert a new user into the database.

String insert = "INSERT INTO users (FirstName, LastName, Email, Password, privacyEnabled) VALUES ('"

+ u.getFirstName() + "','"

+ u.getLastName() + "','"

+ u.getEmail() + "','"

+ u.getPassword() + "', false)";

try{

databaseFacade.execute(insert);

}

catch(SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to check if the email is already used by another user.

public boolean isEmailUsed(){

String select = "SELECT \* FROM users WHERE Email = '"+u.getEmail()+"'";

boolean used = false;

try {

// Execute the SQL query and check if there is any result.

ResultSet rs = databaseFacade.executeQuery(select);

used = rs.next();

} catch (SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

return used;

}

// Method to get the user from the database based on email and password.

public User getUser(){

u.setFriends(new ArrayList<>());

u.setPosts(new ArrayList<>());

// SQL query to select the user's ID based on email and password.

String select = "SELECT ID FROM users WHERE Email = '"+u.getEmail()+"' AND Password = '"+u.getPassword()+"'";

try {

ResultSet rs = databaseFacade.executeQuery(select);

rs.next();

u.setID(rs.getInt("ID"));

} catch(SQLException e){

new Alert(e.getMessage(), null);

}

return u;

}

}

## 5. public class GenerateTimeline;

# **Explanation**: The GenerateTimeline class is part of the Controller package and is responsible for generating a timeline of posts for a user. It can generate a timeline based on posts from all friends of the user or posts from a specific user. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 5. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Post;

import Model.User;

import View.Alert;

import java.util.ArrayList;

import java.sql.SQLException;

import java.sql.ResultSet;

public class GenerateTimeline {

private ArrayList<Post> posts;

// Constructor for generating timeline based on all friends' posts.

public GenerateTimeline(User user, DatabaseFacade databaseFacade){

posts = new ArrayList<>();

// If the user has no friends, return without fetching posts.

if (user.getFriendsIDs().isEmpty()) {

return;

}

StringBuilder sb = new StringBuilder();

// Constructing SQL query to fetch posts from all friends.

if (user.getFriendsIDs().size()!=0){

for (int i = 0 ; i<user.getFriendsIDs().size();i++){

sb.append(" User = "+user.getFriendsIDs().get(i));

if(i!= user.getFriendsIDs().size()-1){

sb.append(" OR ");

}

else {

sb.append(";");

}

}

}

String select= "SELECT \* FROM `posts` WHERE "+sb.toString();

try {

ResultSet rs = databaseFacade.executeQuery(select);

ArrayList<Integer> usersIDs = new ArrayList<>();

while (rs.next()){

Post p = new Post();

p.setID(rs.getInt("ID"));

p.setContent(rs.getString("Content"));

usersIDs.add(rs.getInt("User"));

p.setDateTimeFromString(rs.getString("DateTime"));

posts.add(p);

}

for (int i=0; i <usersIDs.size(); i++){

posts.get(i).setUser(new ReadUserByID(usersIDs.get(i),databaseFacade).getUser());

}

}catch(SQLException e){

System.out.printf(e.getMessage());

new Alert(e.getMessage(), null);

}

}

// Constructor for generating timeline based on posts of a specific user.

public GenerateTimeline(User user, DatabaseFacade databaseFacade,User mainUser){

posts = new ArrayList<>();

// Constructing SQL query to fetch posts of the specified user.

String select= "SELECT \* FROM `posts` WHERE User = "+user.getID()+"";

try {

ResultSet rs = databaseFacade.executeQuery(select);

ArrayList<Integer> usersIDs = new ArrayList<>();

while (rs.next()){

Post p = new Post();

p.setID(rs.getInt("ID"));

p.setContent(rs.getString("Content"));

usersIDs.add(rs.getInt("User"));

p.setDateTimeFromString(rs.getString("DateTime"));

posts.add(p);

}

for (int i=0; i <usersIDs.size(); i++){

posts.get(i).setUser(new ReadUserByID(usersIDs.get(i),databaseFacade).getUser());

}

}catch(SQLException e){

System.out.printf(e.getMessage());

new Alert(e.getMessage(), null);

}

}

// Method to get the generated posts.

public ArrayList<Post> getPosts(){

return posts;

}

}

## 

## 6. public class JoinGroup;

# **Explanation**: The JoinGroup class is part of the Controller package and is responsible for handling the logic for a user to join a group. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 6. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Group;

import Model.User;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

public class JoinGroup {

private Group group;

private DatabaseFacade databaseFacade;

private User user;

// Constructor for JoinGroup class.

public JoinGroup(Group g, DatabaseFacade databaseFacade,User user){

this.group = g;

this.databaseFacade = databaseFacade;

this.user = user;

}

// Method to join the group.

public boolean isJoined(){

boolean joined = false;

// SQL query to insert the user into the group's members.

String insert ="INSERT INTO groupsmembers(`GID`, `UID`) VALUES ("+group.getGID()+", "+user.getID()+") ";

try {

databaseFacade.execute(insert);

joined = true;

} catch (SQLException e){

System.out.println(e.getMessage());

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

joined = false;

}

return joined;

}

// Method to check if the user is already a member of the group.

public boolean isMember(){

boolean ismember = false;

// SQL query to check if the user is already a member of the group.

String select ="SELECT \* FROM groupsmembers WHERE GID ="+group.getGID()+" AND UID = "+user.getID()+" ";

try {

// Execute the SQL query and check if there is any result.

ResultSet rs = databaseFacade.executeQuery(select);

ismember = rs.next();

} catch (SQLException e){

System.out.println(e.getMessage());

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

ismember = false;

}

return ismember; // Return whether the user is already a member of the group or not.

}

}

## 7. public class ReadAllGroupMembers;

# **Explanation**: The ReadAllGroupMembers class is part of the Controller package and is responsible for fetching and returning all members of a specified group from the database. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 7. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Group;

import Model.User;

import java.util.ArrayList;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

public class ReadAllGroupMembers {

private ArrayList<User> members;

// Constructor for ReadAllGroupMembers class.

public ReadAllGroupMembers(DatabaseFacade databaseFacade, Group group){

// SQL query to select all members of the group.

String select = "SELECT \* FROM groupsmembers WHERE GID = '"+group.getGID()+"'";

members = new ArrayList<>();

try {

ArrayList<Integer> ids = new ArrayList<>();

ResultSet rs = databaseFacade.executeQuery(select);

while (rs.next()){

int id = rs.getInt("UID");

ids.add(id);

}

for (int id: ids){

User u = new User();

ReadUserByID rd = new ReadUserByID(id,databaseFacade);

u = rd.getUser();

members.add(u);

}

}catch (SQLException e){

System.out.println(e.getMessage());

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to get the list of group members.

public ArrayList<User> getList() {

return members;

}

}

## 8. public class ReadAllGroups;

# **Explanation**: The ReadAllGroups class is part of the Controller package and is responsible for fetching and returning all groups from the database. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 8. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.Group;

import java.util.ArrayList;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

public class ReadAllGroups {

private ArrayList<Group> groups;

// Constructor for ReadAllGroups class.

public ReadAllGroups(DatabaseFacade databaseFacade){

// SQL query to select all groups.

String select = "SELECT \* FROM groups";

groups = new ArrayList<>();

try {

ResultSet rs = databaseFacade.executeQuery(select);

while (rs.next()){

Group g = new Group();

g.setGID(rs.getInt("GID"));

g.setName(rs.getString("Name"));

groups.add(g);

}

}catch (SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to get the list of groups.

public ArrayList<Group> getList() {

return groups;

}

}

## 9. public class ReadAllUsers;

# **Explanation**: The ReadAllUsers class is part of the Controller package and is responsible for fetching and returning all users from the database, excluding the logged-in user and users who have enabled privacy in their settings. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 9. Source Code

package Controller;

import Model.DatabaseFacade;

import java.util.ArrayList;

import java.sql.ResultSet;

import Model.User;

import View.Alert;

import java.sql.SQLException;

public class ReadAllUsers {

private ArrayList<User> users;

// Constructor for ReadAllUsers class.

public ReadAllUsers(DatabaseFacade databaseFacade, User user){

// SQL query to select all users.

String select = "SELECT \* FROM users";

users = new ArrayList<>();

try {

ResultSet rs = databaseFacade.executeQuery(select);

while (rs.next()){

User u = new User();

u.setID(rs.getInt("ID"));

u.setFirstName(rs.getString("FirstName"));

u.setLastName(rs.getString("LastName"));

u.setEmail(rs.getString("Email"));

// Skipping user if privacyEnabled is true or if it's the same user as the logged-in user.

if ("true".equals(rs.getString("privacyEnabled"))){

continue;

}

if (u.getID()!= user.getID()) users.add(u);

}

}catch (SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to get the list of users.

public ArrayList<User> getList() {

return users;

}

}

10) public class ReadUser;

# **Explanation**: The ReadUser class is part of the Controller package and is responsible for fetching a user from the database based on the provided email and password. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 10. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.User;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

import java.util.ArrayList;

public class ReadUser {

private boolean loggedIn;

private User user;

// Constructor for ReadUser class.

public ReadUser(String email, String password, DatabaseFacade databaseFacade){

// SQL query to select user with the provided email and password.

String select = "SELECT \* FROM users WHERE Email = '"+email+"' AND Password = '"+password+"'";

try{

ResultSet rs = databaseFacade.executeQuery(select);

loggedIn = rs.next();

if(loggedIn){

// Create a new User object and set its attributes based on the database result.

user = new User();

user.setID(rs.getInt("ID"));

user.setFirstName(rs.getString("FirstName"));

user.setLastName(rs.getString("LastName"));

user.setEmail(rs.getString("Email"));

user.setPassword(rs.getString("Password"));

if("0".equals(rs.getString("privacyEnabled"))){

user.setprivacyEnabled(false);

}

else {

user.setprivacyEnabled(true);

}

if("false".equals(rs.getString("privacyEnabled"))){

user.setprivacyEnabled(false);

}

else {

user.setprivacyEnabled(true);

}

String findFriends = "SELECT \* FROM friends WHERE User = "+user.getID()+"";

ResultSet rs2 = databaseFacade.executeQuery(findFriends);

ArrayList<Integer> friendsIDs = new ArrayList<>();

while (rs2.next()){

friendsIDs.add(rs2.getInt("Friend"));

}

user.setFriendsIDs(friendsIDs);

}

}catch(SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to check if the user is logged in.

public boolean loggedIn(){

return loggedIn;

}

// Method to get the logged-in user.

public User getUser(){

return user;

}

}

11) public class ReadUserByID;

# **Explanation:** The ReadUserByID class is part of the Controller package and is responsible for fetching a user from the database based on the provided user ID. It interacts with the DatabaseFacade to perform the necessary database operations and uses the Alert view to notify the user in case of errors.

### 11. Source Code

package Controller;

import Model.DatabaseFacade;

import Model.User;

import View.Alert;

import java.sql.SQLException;

import java.sql.ResultSet;

public class ReadUserByID {

private User user;

// Constructor for ReadUserByID class.

public ReadUserByID(int ID, DatabaseFacade databaseFacade){

// SQL query to select user by ID.

String select = "SELECT \* FROM users WHERE ID = "+ID+"";

try {

ResultSet rs = databaseFacade.executeQuery(select);

rs.next();

// Create a new User object and set its attributes based on the database result.

user = new User();

user.setID(ID);

user.setFirstName(rs.getString("FirstName"));

user.setLastName(rs.getString("LastName"));

user.setEmail(rs.getString("Email"));

// Set privacyEnabled based on the database value.

if("false".equals(rs.getString("privacyEnabled"))){

user.setprivacyEnabled(false);

}

else {

user.setprivacyEnabled(true);

}

}catch(SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

// Method to get the user.

public User getUser(){

return user;

}

}

### 12) public class RemoveFriend ;

### **Explanation**: The RemoveFriend class, located in the Controller package, is responsible for removing a friend relationship between two users from the database. It interacts with the DatabaseFacade to execute SQL commands and displays an alert in case of errors using the Alert view.

### 12. Source Code

package Controller;

import View.Alert;

import java.sql.SQLException;

import Model.DatabaseFacade;

import Model.User;

public class RemoveFriend {

private User user,f;

private DatabaseFacade databaseFacade;

// Constructor for RemoveFriend class.

public RemoveFriend(User user, DatabaseFacade databaseFacade, User f){

this.user = user;

this.f = f;

this.databaseFacade = databaseFacade;

}

// Method to remove a friend.

public boolean isRemoved(){

boolean removed = false;

// SQL query to remove the friendship relation from the database.

String delete = "DELETE FROM friends WHERE User = "+user.getID()+" AND Friend = "+f.getID()+" ";

try {

databaseFacade.execute(delete);

removed = true;

}catch (SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

removed = false;

}

return removed;

}

}

13) public class UpdateUserPrivacy;

### **Explanation**: The UpdateUserPrivacy class, residing in the Controller package, is responsible for updating the privacy setting of a user in the database. It interacts with the DatabaseFacade to execute SQL commands and utilizes the Alert view to notify the user in case of errors.

### 13. Source Code

package Controller;

import Model.User;

import Model.DatabaseFacade;

import View.Alert;

import java.sql.SQLException;

public class UpdateUserPrivacy {

// Constructor for UpdateUserPrivacy class.

public UpdateUserPrivacy (User u, DatabaseFacade databaseFacade,String t){

// SQL query to update user privacy setting.

String updateQuery = "UPDATE users SET privacyEnabled = '"+t+"' WHERE ID = '"+u.getID()+"'";

try {

databaseFacade.executeUpdate(updateQuery);

} catch (SQLException e){

// In case of an error, show an alert to the user with the error message.

new Alert(e.getMessage(), null);

}

}

}

14) public class Database;

**Explanation**: The Database class, located in the Model package, is responsible for establishing a connection to the database and providing a statement object to execute SQL queries. It utilizes JDBC (Java Database Connectivity) to interact with the MySQL database.

### 14. Source Code

package Model;

import java.sql.Statement;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.DriverManager;

public class Database {

private String user = "root";

private String pass = "";

private String url = "jdbc:mysql://localhost:3308/socialmedia";

private Statement statement;

// Constructor for Database class.

public Database() {

try {

// Establishing connection to the database.

Connection connection = DriverManager.getConnection(url, user, pass);

// Creating a statement for executing SQL queries.

statement = connection.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY);

} catch (SQLException e) {

// If connection fails, print stack trace.

e.printStackTrace();

}

}

// Method to get the statement object.

public Statement getStatement() {

return statement;

}

}

15) public class DatabaseFacade;

**Explanation**: The DatabaseFacade class, located in the Model package, serves as an intermediary between the application and the database. It encapsulates the functionality for executing SQL queries and handling database operations. This class interacts with the Database class to perform database operations.

### 15. Source Code

package Model;

import java.sql.ResultSet;

import java.sql.SQLException;

public class DatabaseFacade {

private Database database;

// Constructor for DatabaseFacade class.

public DatabaseFacade() {

// Create an instance of the Database class.

this.database = new Database();

}

// Method to execute a query and return a ResultSet.

public ResultSet executeQuery(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

return database.getStatement().executeQuery(query);

}

// Method to execute a query without returning a ResultSet.

public void execute(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

database.getStatement().execute(query);

}

// Method to execute an update query (e.g., INSERT, UPDATE, DELETE).

public void executeUpdate(String query) throws SQLException{

// Execute the update query using the statement obtained from the Database object.

database.getStatement().executeUpdate(query);

}

}

16) public class Group;

**Explanation**: The Group class, located in the Model package, represents a group entity in the social media application. It encapsulates information about a group, such as its unique identifier (ID) and name.

### 16. Source Code

package Model;

public class Group {

private int GID; // Group ID

private String Name; // Group name

// Default constructor

public Group(){}

// Getter for Group ID

public int getGID() {

return GID;

}

// Setter for Group ID

public void setGID(int GID) {

this.GID = GID;

}

// Getter for Group name

public String getName() {

return Name;

}

// Setter for Group name

public void setName(String Name) {

this.Name = Name;

}

}

17) public class Post;

### **Explanation**: The Post class, located in the Model package, represents a post made by a user in the social media application. It encapsulates information about a post, such as its unique identifier (ID), content, creator, creation date and time, users who liked the post, and methods for manipulating date and time.

### 17. Source Code

package Model;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

public class Post {

private int ID; // Post ID

private String content; // Post content

private User user; // User who created the post

private LocalDateTime dateTime; // Date and time when the post was created

private ArrayList<User> likes; // List of users who liked the post

private DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"); // Formatter for date and time

private DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("EEE,dd MMM yyyy "); // Formatter for date only

// Default constructor

public Post() {}

// Constructor with content and user parameters

public Post(String content, User user){

this.content = content;

this.user = user;

dateTime = LocalDateTime.now(); // Set current date and time

}

// Getter for Post ID

public int getID() {

return ID;

}

// Setter for Post ID

public void setID(int ID) {

this.ID = ID;

}

// Getter for content

public String getContent() {

return content;

}

// Setter for content

public void setContent(String content) {

this.content = content;

}

// Getter for user

public User getUser() {

return user;

}

// Setter for user

public void setUser(User user) {

this.user = user;

}

// Getter for dateTime

public LocalDateTime getDateTime() {

return dateTime;

}

// Setter for dateTime

public void setDateTime(LocalDateTime dateTime) {

this.dateTime = dateTime;

}

// Getter for likes

public ArrayList<User> getLikes() {

return likes;

}

// Setter for likes

public void setLikes(ArrayList<User> likes) {

this.likes = likes;

}

// Method to format dateTime to string

public String getDateTimeToString(){

return dateTimeFormatter.format(dateTime);

}

// Method to parse string dateTime to LocalDateTime

public void setDateTimeFromString(String dateTime){

this.dateTime = LocalDateTime.parse(dateTime, dateTimeFormatter);

}

// Method to format date to string

public String getDateToString(){

return dateFormatter.format(dateTime);

}

}

18) public class User;

### **Explanation**: The User class, located in the Model package, represents a user entity in the social media application. It encapsulates information about a user, such as their unique identifier (ID), first name, last name, email, password, list of posts, list of friends, privacy settings, and methods for interacting with friends.

### 18. Source Code

package Model;

import java.util.ArrayList;

public class User {

private int ID; // User ID

private String firstName; // First name

private String lastName; // Last name

private String email; // Email

private String password; // Password

private ArrayList<Post> posts; // List of posts created by the user

private ArrayList<User> friends; // List of user's friends

private ArrayList<Integer> friendsIDs; // List of IDs of user's friends

private boolean privacyEnabled; // Privacy setting

// Default constructor

public User(){

// Initialize ArrayLists

posts = new ArrayList<>();

friends = new ArrayList<>();

friendsIDs = new ArrayList<>();

}

// Getter for ID

public int getID() {

return ID;

}

// Setter for ID

public void setID(int ID) {

this.ID = ID;

}

// Getter for firstName

public String getFirstName() {

return firstName;

}

// Setter for firstName

public void setFirstName(String firstName) {

this.firstName = firstName;

}

// Getter for lastName

public String getLastName() {

return lastName;

}

// Getter for full name

public String getName(){

return firstName + " " + lastName;

}

// Setter for lastName

public void setLastName(String lastName) {

this.lastName = lastName;

}

// Getter for email

public String getEmail() {

return email;

}

// Setter for email

public void setEmail(String email) {

this.email = email;

}

// Getter for password

public String getPassword() {

return password;

}

// Setter for password

public void setPassword(String password) {

this.password = password;

}

// Getter for posts

public ArrayList<Post> getPosts() {

return posts;

}

// Setter for posts

public void setPosts(ArrayList<Post> posts) {

this.posts = posts;

}

// Getter for friends

public ArrayList<Integer> getFriends() {

// Convert list of User objects to a list of their IDs

ArrayList<Integer> ids = new ArrayList<>();

for (User friend: friends){

ids.add(friend.getID());

}

return ids;

}

// Setter for friends

public void setFriends(ArrayList<User> friends) {

this.friends = friends;

// Update friendsIDs list

friendsIDs = new ArrayList<>();

for( User u: friends){

friendsIDs.add(u.getID());

}

}

// Setter for friendsIDs

public void setFriendsIDs(ArrayList<Integer> friendsIDs){

this.friendsIDs = friendsIDs;

}

// Getter for friendsIDs

public ArrayList<Integer> getFriendsIDs(){

return friendsIDs;

}

// Method to check if a user is a friend of this user

public boolean isFriend(User u ){

return friendsIDs.contains(u.getID());

}

// Method to add a friend

public void addFriend(User f) {

friends.add(f);

friendsIDs.add(f.getID());

}

// Method to remove a friend

public void removeFriend(User f) {

friends.remove(f);

friendsIDs.remove((Integer)f.getID());

}

// Getter for privacyEnabled

public boolean getprivacyEnabled(){

return privacyEnabled;

}

// Setter for privacyEnabled

public void setprivacyEnabled(boolean p){

this.privacyEnabled = p;

}

# Design Pattern

# 1)MVP (Model View Presenter) Design Pattern

**Model (Model):** The Model section of the project is generally responsible for data processing and storage operations. For example, classes like **User**, **Group**, and **DatabaseFacade** can be considered as part of this section. These classes access data, implement business logic, and perform database operations.

**View:** The View part is where the user interface is created and presented to the user. Classes like **Login**, **Welcome**, **Home**, and **Post** can be considered as part of this section. These classes contain different components of the user interface (such as **JPanel, JFrame**, **JLabel**, etc.) and are arranged for presentation to the user.

**Presenter:** The Presenter acts as a bridge between the Model and the View. In this section, user interactions (such as button clicks, text inputs, etc.) are listened to, and based on that, operations are performed on the Model or the View is updated. Presenter classes are often referred to as Controllers as well. The button click actions and data processing steps inside the **Login**, **Welcome**, and **Home** classes can be performed in this section.

**Model-View-Presenter Communication:** In the MVP design pattern, there is no direct connection between the Model and the View. Instead, the Presenter receives data processed by the Model, passes it to the View, and forwards user interactions from the View to the Model. This reduces the dependency between the Model and the View, making the code more modular, maintainable, and testable. For example, a user login attempt in the **Login** class is forwarded through the Presenter to the Model, where validation is performed, and the results are shown to the user through the View.

package Model;

import java.util.ArrayList;

public class User {

private int ID; // User ID

private String firstName; // First name

private String lastName; // Last name

private String email; // Email

private String password; // Password

private ArrayList<Post> posts; // List of posts created by the user

private ArrayList<User> friends; // List of user's friends

private ArrayList<Integer> friendsIDs; // List of IDs of user's friends

private boolean privacyEnabled; // Privacy setting

// Default constructor

public User(){

// Initialize ArrayLists

posts = new ArrayList<>();

friends = new ArrayList<>();

friendsIDs = new ArrayList<>();

}

// Getter for ID

public int getID() {

return ID;

}

// Setter for ID

public void setID(int ID) {

this.ID = ID;

}

// Getter for firstName

public String getFirstName() {

return firstName;

}

// Setter for firstName

public void setFirstName(String firstName) {

this.firstName = firstName;

}

// Getter for lastName

public String getLastName() {

return lastName;

}

// Getter for full name

public String getName(){

return firstName + " " + lastName;

}

// Setter for lastName

public void setLastName(String lastName) {

this.lastName = lastName;

}

// Getter for email

public String getEmail() {

return email;

}

// Setter for email

public void setEmail(String email) {

this.email = email;

}

// Getter for password

public String getPassword() {

return password;

}

// Setter for password

public void setPassword(String password) {

this.password = password;

}

// Getter for posts

public ArrayList<Post> getPosts() {

return posts;

}

// Setter for posts

public void setPosts(ArrayList<Post> posts) {

this.posts = posts;

}

// Getter for friends

public ArrayList<Integer> getFriends() {

// Convert list of User objects to a list of their IDs

ArrayList<Integer> ids = new ArrayList<>();

for (User friend: friends){

ids.add(friend.getID());

}

return ids;

}

// Setter for friends

public void setFriends(ArrayList<User> friends) {

this.friends = friends;

// Update friendsIDs list

friendsIDs = new ArrayList<>();

for( User u: friends){

friendsIDs.add(u.getID());

}

}

// Setter for friendsIDs

public void setFriendsIDs(ArrayList<Integer> friendsIDs){

this.friendsIDs = friendsIDs;

}

// Getter for friendsIDs

public ArrayList<Integer> getFriendsIDs(){

return friendsIDs;

}

// Method to check if a user is a friend of this user

public boolean isFriend(User u ){

return friendsIDs.contains(u.getID());

}

// Method to add a friend

public void addFriend(User f) {

friends.add(f);

friendsIDs.add(f.getID());

}

// Method to remove a friend

public void removeFriend(User f) {

friends.remove(f);

friendsIDs.remove((Integer)f.getID());

}

// Getter for privacyEnabled

public boolean getprivacyEnabled(){

return privacyEnabled;

}

// Setter for privacyEnabled

public void setprivacyEnabled(boolean p){

this.privacyEnabled = p;

}

}

package Model;

public class Group {

private int GID; // Group ID

private String Name; // Group name

// Default constructor

public Group(){}

// Getter for Group ID

public int getGID() {

return GID;

}

// Setter for Group ID

public void setGID(int GID) {

this.GID = GID;

}

// Getter for Group name

public String getName() {

return Name;

}

// Setter for Group name

public void setName(String Name) {

this.Name = Name;

}

}

package Model;

import java.sql.ResultSet;

import java.sql.SQLException;

public class DatabaseFacade {

private Database database;

// Constructor for DatabaseFacade class.

public DatabaseFacade() {

// Create an instance of the Database class.

this.database = new Database();

}

// Method to execute a query and return a ResultSet.

public ResultSet executeQuery(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

return database.getStatement().executeQuery(query);

}

// Method to execute a query without returning a ResultSet.

public void execute(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

database.getStatement().execute(query);

}

// Method to execute an update query (e.g., INSERT, UPDATE, DELETE).

public void executeUpdate(String query) throws SQLException{

// Execute the update query using the statement obtained from the Database object.

database.getStatement().executeUpdate(query);

}

}

package View;

import Controller.ReadUser;

import Model.DatabaseFacade;

import Model.User;

import java.awt.BorderLayout;

import java.awt.Cursor;

import java.awt.Font;

import javax.swing.BorderFactory;

import javax.swing.JPanel;

import java.awt.GridLayout;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

public class Login {

public Login(DatabaseFacade databaseFacade){

// Creating a JFrame.

JFrame frame = new JFrame();

// Creating the main panel.

JPanel panel = new JPanel(new BorderLayout());

panel.setBackground(null);

// Setting the margins of the panel.

panel.setBorder(BorderFactory.createEmptyBorder(115,0,182,0));

// Creating a title label and adding it to the north of the panel.

JLabel title = new JLabel("Login", 40, GUIConstants.blue,Font.BOLD);

title.setHorizontalAlignment(JLabel.CENTER);

panel.add(title,BorderLayout.NORTH);

// Creating the center panel using GridLayout.

JPanel center = new JPanel(new GridLayout(3,1,10,10));

center.setBackground(null);

// Setting the margins of the center panel.

center.setBorder(BorderFactory.createEmptyBorder(34,315,17,315));

// Creating an email input field and adding it to the center panel.

JTextField email = new JTextField("Email");

center.add(email);

// Creating a password input field and adding it to the center panel.

JTextField password = new JTextField("Password");

center.add(password);

// Creating a login button and listening for click events.

JButton login = new JButton("Login",42,20);

login.addMouseListener(new MouseListener(){

@Override

public void mouseReleased(MouseEvent e){

}

@Override

public void mouseClicked(MouseEvent e) {

// Showing an alert if email or password is empty.

if(email.isEmpty()){

new Alert("Email cannot be empty",frame);

return;

}

if(password.isEmpty()){

new Alert("Password cannot be empty",frame);

return;

}

// Checking user credentials.

ReadUser read = new ReadUser(email.getText(),password.getText(),databaseFacade);

if (read.loggedIn()){

// Navigating to the home page if login is successful.

User user = read.getUser();

new Home(user,databaseFacade);

frame.dispose();

} else {

// Showing an alert if login fails.

new Alert("Incorrect email or password",frame);

}

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

center.add(login);

panel.add(center,BorderLayout.CENTER);

// Creating a label for creating a new account and listening for click events.

JLabel createAcc = new JLabel("Don't have an account? Create new one!",20,GUIConstants.black,Font.BOLD);

createAcc.addMouseListener(new MouseListener(){

@Override

public void mouseReleased(MouseEvent e){

}

@Override

public void mouseClicked(MouseEvent e) {

// Navigating to the new account creation page.

new Welcome(databaseFacade);

frame.dispose();

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

// Setting the cursor to a hand pointer.

createAcc.setCursor(new Cursor(Cursor.HAND\_CURSOR));

createAcc.setHorizontalAlignment(JLabel.CENTER);

panel.add(createAcc,BorderLayout.SOUTH);

// Adding the main panel to the content pane of the frame and making the frame visible.

frame.getContentPane().add(panel);

frame.setVisible(true);

frame.requestFocus();

}

}

package View;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.ImageIcon;

import javax.swing.JPanel;

@SuppressWarnings("serial")

public class Post extends JPanel{

public Post(Model.Post post){

// Setting up panel layout and appearance

setLayout(new BoxLayout(this,BoxLayout.Y\_AXIS));

setBackground(GUIConstants.white);

setBorder(BorderFactory.createEmptyBorder(15,15,15,25));

// Header panel setup

JPanel header = new JPanel(new BorderLayout());

header.setBackground(null);

// Adding author name to the header panel

JLabel author = new JLabel(post.getUser().getName(), 20,GUIConstants.post,Font.BOLD);

header.add(author,BorderLayout.WEST);

// Adding date to the header panel

JLabel date = new JLabel(post.getDateToString(),15,GUIConstants.post,Font.PLAIN);

header.add(date, BorderLayout.EAST);

add(header);

add(Box.createVerticalStrut(7));// Adding vertical space

// Center panel setup

JPanel center = new JPanel(new FlowLayout(FlowLayout.LEADING));

center.setBackground(null);

// Adding content text area to the center panel

JTextArea content = new JTextArea(post.getContent(),18,GUIConstants.post,Font.PLAIN);

center.add(content);

add(center);

add(Box.createVerticalStrut(7));// Adding vertical space

// Bottom panel setup

JPanel bottom = new JPanel(new BorderLayout());

bottom.setBackground(null);

// Likes panel setup

JPanel likes = new JPanel(new FlowLayout(FlowLayout.LEFT,13,13));

likes.setBackground(null);

// Adding like icon to the likes panel

likes.add(new javax.swing.JLabel(new ImageIcon("icons/like.png")));

likes.add(new JLabel("0 Likes",15,GUIConstants.textFieldHint, Font.BOLD));

bottom.add(likes,BorderLayout.WEST);

// Adding comments label to the bottom panel

JLabel comments = new JLabel("0 Comments",15,GUIConstants.textFieldHint,Font.BOLD);

bottom.add(comments,BorderLayout.EAST);

add(bottom);

// Calculating and setting preferred size of the post panel

int height = (int) (115+ content.getPreferredSize().getHeight());

Dimension dimension = new Dimension(500,height);

setPreferredSize(dimension);

setMaximumSize(dimension);

setMinimumSize(dimension);

}

}

package View;

import Controller.CreateUser;

import Model.DatabaseFacade;

import Model.User;

import javax.swing.BorderFactory;

import java.awt.Font;

import java.awt.GridLayout;

import java.awt.BorderLayout;

import java.awt.Cursor;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import javax.swing.JPanel;

public class Welcome {

public Welcome(DatabaseFacade databaseFacade){

JFrame frame = new JFrame();

JPanel panel = new JPanel(new BorderLayout());

// Setting up the main panel

panel.setBackground(null);

panel.setBorder(BorderFactory.createEmptyBorder(53,84,76,84));

panel.add(new JLabel("Welcome",40,GUIConstants.blue,Font.BOLD),BorderLayout.NORTH);

// Creating center panel for input fields and button

JPanel center = new JPanel(new GridLayout(6,1,10,10));

center.setBackground(null);

center.setBorder(BorderFactory.createEmptyBorder(22,231,17,231));

// Adding input fields

JTextField firstName = new JTextField("First Name");

center.add(firstName);

JTextField lastName = new JTextField("Last Name");

center.add(lastName);

JTextField email = new JTextField("Email");

center.add(email);

JTextField password = new JTextField("password");

center.add(password);

JTextField confirmPassword = new JTextField("Confirm Password");

center.add(confirmPassword);

JButton createAcc = new JButton("Create Account", 45,20);

// Adding action listener to create account button

createAcc.addMouseListener(new MouseListener(){

@Override

public void mouseClicked(MouseEvent e) {

// Validation checks for input fields

if (firstName.isEmpty()){

new Alert("First Name cannot be empty",frame);

return;

}

if (lastName.isEmpty()){

new Alert("Last Name cannot be empty",frame);

return;

}

if (email.isEmpty()){

new Alert("Email cannot be empty",frame);

return;

}

if (password.isEmpty()){

new Alert("Password cannot be empty",frame);

return;

}

if (password.getText().length()<6){

new Alert("Password must be contains at least 6 characters",frame);

return;

}

if (confirmPassword.isEmpty()){

new Alert("Please confirm password",frame);

return;

}

if (!password.getText().equals(confirmPassword.getText())){

new Alert ("Password doesn't match",frame);

}

// Creating user object with input data

User u = new User();

u.setFirstName(firstName.getText());

u.setLastName(lastName.getText());

u.setEmail(email.getText());

u.setPassword(password.getText());

// Creating user in the database

CreateUser create = new CreateUser(u,databaseFacade);

if(!create.isEmailUsed()){

create.create();

u = create.getUser();

// Navigating to home page after successful account creation

new Home(u,databaseFacade);

}

else {

new Alert("This email has been used before",frame);

}

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseReleased(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

// Adding create account button to center panel

center.add(createAcc);

// Adding center panel to main panel

panel.add(center,BorderLayout.CENTER);

// Adding login label

JLabel login = new JLabel("Already have an account? Login", 20, GUIConstants.black, Font.BOLD);

login.addMouseListener(new MouseListener(){

@Override

public void mouseReleased(MouseEvent e){

}

@Override

public void mouseClicked(MouseEvent e) {

new Login(databaseFacade);

frame.dispose();

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

// Configuring login label

login.setCursor(new Cursor(Cursor.HAND\_CURSOR));

login.setHorizontalAlignment(JLabel.CENTER);

panel.add(login,BorderLayout.SOUTH);

// Adding main panel to frame

frame.getContentPane().add(panel);

frame.setVisible(true);

frame.requestFocus();

}

}

package View;

import Controller.CreatePost;

import Controller.GenerateTimeline;

import Model.DatabaseFacade;

import Model.User;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.util.ArrayList;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.JPanel;

public class Home {

// Constructor for the Home view

public Home(User user, DatabaseFacade databaseFacade){

JFrame frame = new JFrame();

frame.getContentPane().setLayout(new BorderLayout());

// Generate the user's timeline

new GenerateTimeline(user,databaseFacade);

// Create and configure the sidebar

JPanel sideBar = new JPanel();

sideBar.setBackground(GUIConstants.white);

Dimension sideBarDim = new Dimension(182,1000);

sideBar.setPreferredSize(sideBarDim);

sideBar.setMaximumSize(sideBarDim);

sideBar.setMinimumSize(sideBarDim);

sideBar.setLayout(new BoxLayout(sideBar,BoxLayout.Y\_AXIS));

sideBar.add(Box.createVerticalStrut(10));

// Add user profile to the sidebar

JPanel profile = new JPanel(new FlowLayout(FlowLayout.CENTER,10,10));

profile.setMaximumSize(new Dimension(182,50));

profile.setBackground(GUIConstants.white);

profile.add(new JLabel(user.getName(),19,GUIConstants.black,Font.BOLD));

sideBar.add(profile);

sideBar.add(Box.createVerticalStrut(3));

// Add navigation buttons to the sidebar

sideBar.add(new SideButton("Posts","more",user,databaseFacade,frame));

sideBar.add(Box.createVerticalStrut(3));

sideBar.add(new SideButton("Group","people",user,databaseFacade,frame));

sideBar.add(Box.createVerticalStrut(3));

sideBar.add(new SideButton("Settings","setting",user,databaseFacade,frame));

sideBar.add(Box.createVerticalStrut(3));

sideBar.add(new SideButton("Friends","friends",user,databaseFacade,frame));

sideBar.add(Box.createVerticalStrut(3));

frame.getContentPane().add(sideBar,BorderLayout.WEST);

// Create and configure the main panel

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel,BoxLayout.Y\_AXIS));

panel.setBackground(null);

// Create and configure the header

JPanel header = new JPanel(new BorderLayout());

header.setBackground(GUIConstants.white);

Dimension dimension = new Dimension(500,159);

header.setPreferredSize(dimension);

header.setMinimumSize(dimension);

header.setMaximumSize(dimension);

header.setBorder(BorderFactory.createEmptyBorder(10,15,10,15));

JPanel north = new JPanel(new BorderLayout());

north.setBackground(null);

north.add(new JLabel("Home",20,GUIConstants.black,Font.BOLD),BorderLayout.WEST);

header.add(north,BorderLayout.NORTH);

// Add the post input text area

JTextArea postIn = new JTextArea("What is happening!?",18,20);

header.add(new JScrollPane(postIn),BorderLayout.CENTER);

// Create and configure the south panel for the post button

JPanel south = new JPanel(new FlowLayout(FlowLayout.RIGHT));

south.setBackground(null);

JButton postBtn = new JButton("Post",35,16);

postBtn.addMouseListener(new MouseListener() {

@Override

public void mouseClicked(MouseEvent e) {

if (postIn.isEmpty()){

new Alert("Cannot publish empty post!!",frame);

return;

}

Model.Post post = new Model.Post(postIn.getText(),user);

if (new CreatePost(post,databaseFacade).posted()){

new Alert("Posted succesfully",frame);

postIn.setText("");

}

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseReleased(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

});

postBtn.setPreferredSize(new Dimension(81,37));

south.add(postBtn);

header.add(south,BorderLayout.SOUTH);

panel.add(header);

// Add posts to the main panel

ArrayList<Model.Post> posts = new GenerateTimeline(user,databaseFacade).getPosts();

for (int i = 0 ; i<posts.size();i++){

panel.add(Box.createVerticalStrut(7));

panel.add(new Post(posts.get(i)));

}

frame.getContentPane().add(new JScrollPane(panel),BorderLayout.CENTER);

frame.getContentPane().add(Box.createHorizontalStrut(182),BorderLayout.EAST);

frame.setVisible(true);

frame.requestFocus();

}

}

# 2)Facade Design Pattern

Facade is a design pattern that encapsulates a more complex subsystem with a user-friendly and understandable interface. In this case, the **DatabaseFacade** class hides the complexity of the underlying **Database** class and provides clients with a simpler and easier-to-use interface.

The Facade pattern reduces code repetition and hides how the subsystem works when using it. This way, the rest of the system is not affected by changes made to the subsystem, minimizing dependencies on the subsystem.

In the given code example, methods like **executeQuery**, **execute**, and **executeUpdate** within the **DatabaseFacade** class hide the functionality of the underlying **Database** class and provide an interface to the user. This prevents clients from being exposed to the details or complexity of the **Database** class.

package Model;

import java.sql.ResultSet;

import java.sql.SQLException;

public class DatabaseFacade {

private Database database;

// Constructor for DatabaseFacade class.

public DatabaseFacade() {

// Create an instance of the Database class.

this.database = new Database();

}

// Method to execute a query and return a ResultSet.

public ResultSet executeQuery(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

return database.getStatement().executeQuery(query);

}

// Method to execute a query without returning a ResultSet.

public void execute(String query) throws SQLException {

// Execute the query using the statement obtained from the Database object.

database.getStatement().execute(query);

}

// Method to execute an update query (e.g., INSERT, UPDATE, DELETE).

public void executeUpdate(String query) throws SQLException{

// Execute the update query using the statement obtained from the Database object.

database.getStatement().executeUpdate(query);

}

}

# Test Cases/Screenshots

## 

## 1) Register Screen

* **When you first launch the application, you will be directed to the register screen.**
* **Fill in the required information such as first name, last name, email, password and confirm password.**
* **Click on the "Create Account" button to create your account.**
* **If you have an account yet, click on the "Already have an account? Login"**

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If you leave the first name blank you will receive a warning message.**

metin, ekran görüntüsü, yazı tipi, marka içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 2) Login Screen

* **Enter your email and password in the designated fields.**
* **Click on the "Login" button to proceed.**
* **If you don’t have an account yet, click on the "Don’t have an account? Create new one!"**

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 3) Post Screen

* **After logging in, you can do everything on the main screen. You can share posts, add friends, create and join groups, and customize your privacy settings in the “Settings” section.**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If you want to share a post, write something and click the post button.**

metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **After clicking the post button, a message saying “Posted successfully” will appear on the screen.**

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **You can also see the posts you've shared in the post section.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If you have friends, their posts will also appear on the main screen.**

metin, ekran görüntüsü, sayı, numara, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 4) Add friends / Friends Screen

* **If there are other users registered in the system, you can become friends with them by pressing the “Add” button.**

metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If other users want, they can make themselves invisible by clicking the “Privacy” button in the settings section.**
* **When users click the “Privacy” button, they become invisible, and no one can become friends with them.**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **After users click the “Privacy” button, it changes to “Unprivacy”. If users click the “Unprivacy” button, they make themselves visible again and can become friends with other users.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 5) Group Screen

* **When users click on 'Group' on the main screen, the group tab opens, revealing the create group button and the names of the existing groups.**

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If the user clicks on the 'Create' button, the 'Create Group' tab opens.**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **When the user enters the name of the group and then clicks the 'Create' button, a message saying 'New group is created!' is displayed on the screen.**

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **If you click on the 'Info' button next to the group names in the group tab, the names of the users in that group will be displayed.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu