# Social Media Platform

# Project Objective Statement:

To develop a desktop-based social media application using Java Swing, JDBC, and MySQL that enables users to create accounts, share posts, interact through comments and likes, and build connections through a friendship system. The application follows MVC architecture, providing secure user authentication, personalized timelines, and an intuitive graphical interface for seamless social networking experience.

# Algorithm:

## *Controller Package Algorithm Flow*

## Overview

The Controller package handles business logic and database operations for a social media platform with users, posts, comments, likes, and friendships.

### 1. AddFriend Algorithm

**Purpose:** Add a friendship relationship between two users

**Input:** Current user, friend user, database connection

**Steps:**

1. Initialize the user, friend, and database objects
2. Create SQL INSERT statement for the friends table
3. Insert record with User ID and Friend ID
4. Execute the SQL statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

INSERT INTO friends (User, Friend) VALUES (userID, friendID)

### 2. ChangePassword Algorithm

**Purpose:** Update user's password in the database

**Input:** New password, user ID, database connection

**Steps:**

1. Initialize password, user ID, and database connection
2. Create SQL UPDATE statement for the users table
3. Set Password field to new password value
4. Filter by user ID
5. Execute the update statement
6. If successful, return true
7. If SQLException occurs, display error alert and return false

**SQL Query:**

UPDATE users SET Password = 'newPassword' WHERE ID = userID

### 3. CreateComment Algorithm

**Purpose:** Add a new comment to a post

**Input:** Comment object, Post object, User object, database connection

**Steps:**

1. Initialize comment, post, user, and database objects
2. Create SQL INSERT statement for the comments table
3. Insert Content, Post ID, User ID, and DateTime
4. Execute the insert statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

INSERT INTO comments (Content, Post, User, DateTime)

VALUES (content, postID, userID, dateTime)

### 4. CreatePost Algorithm

**Purpose:** Create a new post in the system

**Input:** Post object (with content, user, datetime), database connection

**Steps:**

1. Initialize post object and database connection
2. Create SQL INSERT statement for the posts table
3. Insert Content, User ID, and DateTime
4. Execute the insert statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

INSERT INTO posts (Content, User, DateTime)

VALUES (content, userID, dateTime)

### 5. CreateUser Algorithm

**Purpose:** Register a new user and retrieve user information

**Input:** User object (with firstName, lastName, email, password), database connection

**Steps:**

#### 5.1 Create User

1. Create SQL INSERT statement for the users table
2. Insert FirstName, LastName, Email, and Password
3. Execute the insert statement
4. If SQLException occurs, display error alert

#### 5.2 Check Email Availability

1. Create SQL SELECT statement to find user by email
2. Execute the query
3. If result exists, return true (email is used)
4. Otherwise, return false (email is available)

#### 5.3 Get User After Creation

1. Initialize empty friends list for user
2. Create SQL SELECT statement to get user ID by email and password
3. Execute the query
4. Retrieve the ID from result set
5. Set the ID in user object
6. Return the user object

**SQL Queries:**

INSERT INTO users (FirstName, LastName, Email, Password)

VALUES (firstName, lastName, email, password)

SELECT \* FROM users WHERE Email = 'email'

SELECT ID FROM users WHERE Email = 'email' AND Password = 'password'

### 6. DislikePost Algorithm

**Purpose:** Remove a like from a post (unlike)

**Input:** User object, Post object, database connection

**Steps:**

1. Initialize user, post, and database objects
2. Create SQL DELETE statement for the likes table
3. Filter by User ID and Post ID
4. Execute the delete statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

DELETE FROM likes WHERE User = userID AND Post = postID

### 7. GenerateTimeline Algorithm

**Purpose:** Retrieve all posts from user's friends for timeline display

**Input:** Current user object, database connection

**Steps:**

1. Initialize empty posts list
2. Get list of friend IDs from user object
3. If user has no friends, return empty list
4. Build dynamic SQL WHERE clause:
   * For each friend ID, add condition: User = friendID
   * Join conditions with OR operator
5. Create SQL SELECT statement for posts table with dynamic WHERE clause
6. Execute the query
7. For each result:
   * Create new Post object
   * Set ID, Content, DateTime from result set
   * Store User ID temporarily
   * Add post to list
8. For each post, retrieve and set the User object using ReadUserByID
9. Return the posts list

**SQL Query:**

SELECT \* FROM posts

WHERE User = friendID1 OR User = friendID2 OR User = friendID3...

### 8. LikePost Algorithm

**Purpose:** Add a like to a post

**Input:** User object, Post object, database connection

**Steps:**

1. Initialize user, post, and database objects
2. Create SQL INSERT statement for the likes table
3. Insert User ID and Post ID
4. Execute the insert statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

INSERT INTO likes (User, Post) VALUES (userID, postID)

### 9. ReadAllUsers Algorithm

**Purpose:** Retrieve all users except the current user

**Input:** Database connection, current user object

**Steps:**

1. Initialize empty users list
2. Create SQL SELECT statement to retrieve all users
3. Execute the query
4. For each result:
   * Create new User object
   * Set ID, FirstName, LastName, Email from result set
   * If user ID is not equal to current user ID, add to list
5. Return the users list

**SQL Query:**

SELECT \* FROM users

### 10. ReadPostByID Algorithm

**Purpose:** Retrieve a specific post by its ID

**Input:** Post ID, database connection

**Steps:**

1. Create SQL SELECT statement for posts table filtered by ID
2. Execute the query
3. Move to first result
4. Create new Post object
5. Set ID, Content, DateTime from result set
6. Retrieve User ID from result set
7. Use ReadUserByID to get and set the User object
8. Return the post object

**SQL Query:**

SELECT \* FROM posts WHERE ID = postID

### 11. ReadPostComments Algorithm

**Purpose:** Retrieve all comments for a specific post

**Input:** Post object, database connection

**Steps:**

1. Initialize comments counter to 0
2. Initialize empty comments list and user IDs list
3. Create SQL SELECT statement for comments table filtered by Post ID
4. Execute the query
5. For each result:
   * Increment comments counter
   * Create new Comment object
   * Set ID, Content, DateTime from result set
   * Store User ID temporarily
   * Add comment to list
6. For each comment, retrieve and set User object using ReadUserByID
7. Return comments count and comments list

**SQL Query:**

SELECT \* FROM comments WHERE Post = postID

### 12. ReadPostLikes Algorithm

**Purpose:** Count the number of likes for a specific post

**Input:** Post object, database connection

**Steps:**

1. Initialize likes counter to 0
2. Create SQL SELECT statement for likes table filtered by Post ID
3. Execute the query
4. For each result, increment likes counter
5. Return the likes count

**SQL Query:**

SELECT \* FROM likes WHERE Post = postID

### 13. ReadUser Algorithm (Login)

**Purpose:** Authenticate user and retrieve complete user profile with relationships

**Input:** Email, password, database connection

**Steps:**

#### 13.1 Authenticate User

1. Create SQL SELECT statement for users table filtered by email and password
2. Execute the query
3. Check if result exists (user found)
4. If user found, set loggedIn flag to true

#### 13.2 Retrieve User Details

1. Create User object
2. Set ID, FirstName, LastName, Email, Password from result set

#### 13.3 Retrieve Friends

1. Create SQL SELECT for friends table filtered by User ID
2. Execute the query
3. For each result, add Friend ID to friendsIDs list
4. Set friendsIDs in user object

#### 13.4 Retrieve Liked Posts

1. Create SQL SELECT for likes table filtered by User ID
2. Execute the query
3. For each result, add Post ID to likedPostsIDs list
4. Set likesIDs in user object

#### 13.5 Return Results

1. Return loggedIn status (boolean)
2. Return user object with all relationships

**SQL Queries:**

SELECT \* FROM users WHERE Email = 'email' AND Password = 'password'

SELECT \* FROM friends WHERE User = userID

SELECT \* FROM likes WHERE User = userID

### 14. ReadUserByID Algorithm

**Purpose:** Retrieve basic user information by user ID

**Input:** User ID, database connection

**Steps:**

1. Create SQL SELECT statement for users table filtered by ID
2. Execute the query
3. Move to first result
4. Create new User object
5. Set ID, FirstName, LastName, Email from result set
6. Return the user object

**SQL Query:**

SELECT \* FROM users WHERE ID = userID

### 15. ReadUserComments Algorithm

**Purpose:** Retrieve all comments made by a user along with their associated posts

**Input:** User object, database connection, JFrame object

**Steps:**

1. Initialize empty panels list
2. Create SQL SELECT for comments table filtered by User ID
3. Execute the query
4. Initialize empty comments list and post IDs list
5. For each result:
   * Create Comment object
   * Set ID, Content, User, DateTime from result set
   * Store Post ID temporarily
   * Add comment to list
6. For each comment:
   * Retrieve Post object using ReadPostByID with stored Post ID
   * Create Post panel and add to panels list
   * Create Comment panel and add to panels list
7. Return the panels list

**SQL Query:**

SELECT \* FROM comments WHERE User = userID

### 16. ReadUserLikes Algorithm

**Purpose:** Retrieve all posts liked by a specific user

**Input:** User object, database connection

**Steps:**

1. Initialize empty posts list
2. Create SQL SELECT for likes table filtered by User ID
3. Execute the query
4. Store all Post IDs in temporary list
5. For each Post ID:
   * Use ReadPostByID to retrieve complete Post object
   * Add post to posts list
6. Return the posts list

**SQL Query:**

SELECT \* FROM likes WHERE User = userID

### 17. ReadUserPosts Algorithm

**Purpose:** Retrieve all posts created by a specific user

**Input:** User object, database connection

**Steps:**

1. Initialize empty posts list
2. Create SQL SELECT for posts table filtered by User ID
3. Execute the query
4. For each result:
   * Create Post object
   * Set ID, Content, DateTime from result set
   * Set User to the input user object
   * Add post to list
5. Return the posts list

**SQL Query:**

SELECT \* FROM posts WHERE User = userID

### 18. RemoveFriend Algorithm

**Purpose:** Remove a friendship relationship between two users

**Input:** Current user, friend user, database connection

**Steps:**

1. Initialize user, friend, and database objects
2. Create SQL DELETE statement for the friends table
3. Filter by User ID and Friend ID
4. Execute the delete statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

DELETE FROM friends WHERE User = userID AND Friend = friendID

### 19. UpdateUser Algorithm

**Purpose:** Update user profile information

**Input:** User object (with updated info), database connection

**Steps:**

#### 19.1 Check Email Availability

1. Use CreateUser's isEmailUsed() method
2. Return true if email is already used by another user

#### 19.2 Update User Information

1. Create SQL UPDATE statement for users table
2. Set FirstName, LastName, Email to new values
3. Filter by User ID
4. Execute the update statement
5. If successful, return true
6. If SQLException occurs, display error alert and return false

**SQL Query:**

UPDATE users

SET FirstName = 'firstName', LastName = 'lastName', Email = 'email'

WHERE ID = userID

# *Model Package Algorithm Flow*

## Overview

The Model package contains data entity classes representing core domain objects: User, Post, Comment, and Database connection. These classes encapsulate data and provide methods for data manipulation.

### 1. Comment Class Algorithm

**Purpose:** Represent a comment entity with content, user, and timestamp

**Attributes:**

* ID (int)
* content (String)
* user (User)
* dateTime (LocalDateTime)
* dateTimeFormatter (DateTimeFormatter - "yyyy-MM-dd HH:mm:ss")
* dateFormatter (DateTimeFormatter - "EEE, dd MMM yyyy")

**Steps:**

#### 1.1 Constructor (Empty)

1. Create empty Comment object
2. All attributes initialized to default values

#### 1.2 Constructor (Parameterized)

1. Accept content (String) and user (User) as parameters
2. Set content to provided value
3. Set user to provided User object
4. Set dateTime to current system time using LocalDateTime.now()

#### 1.3 Getter and Setter Methods

1. **getID()** - Return comment ID
2. **setID(int ID)** - Set comment ID
3. **getContent()** - Return comment content
4. **setContent(String content)** - Set comment content
5. **getUser()** - Return User object who created comment
6. **setUser(User user)** - Set User object
7. **getDateTime()** - Return LocalDateTime object
8. **setDateTime(LocalDateTime dateTime)** - Set LocalDateTime object

#### 1.4 Date/Time Formatting Methods

1. **getDateTimeToString()**
   * Format dateTime using dateTimeFormatter pattern
   * Return formatted string: "yyyy-MM-dd HH:mm:ss"
2. **setDateTimeFromString(String dateTime)**
   * Parse string using dateTimeFormatter
   * Convert to LocalDateTime object
   * Set dateTime attribute
3. **getDateToString()**
   * Format dateTime using dateFormatter pattern
   * Return formatted string: "EEE, dd MMM yyyy"

### 2. Post Class Algorithm

**Purpose:** Represent a post entity with content, user, and timestamp

**Attributes:**

* ID (int)
* content (String)
* user (User)
* dateTime (LocalDateTime)
* dateTimeFormatter (DateTimeFormatter - "yyyy-MM-dd HH:mm:ss")
* dateFormatter (DateTimeFormatter - "EEE, dd MMM yyyy")

**Steps:**

#### 2.1 Constructor (Empty)

1. Create empty Post object
2. All attributes initialized to default values

#### 2.2 Constructor (Parameterized)

1. Accept content (String) and user (User) as parameters
2. Set content to provided value
3. Set user to provided User object
4. Set dateTime to current system time using LocalDateTime.now()

#### 2.3 Getter and Setter Methods

1. **getID()** - Return post ID
2. **setID(int ID)** - Set post ID
3. **getContent()** - Return post content
4. **setContent(String content)** - Set post content
5. **getUser()** - Return User object who created post
6. **setUser(User user)** - Set User object
7. **getDateTime()** - Return LocalDateTime object
8. **setDateTime(LocalDateTime dateTime)** - Set LocalDateTime object

#### 2.4 Date/Time Formatting Methods

1. **getDateTimeToString()**
   * Format dateTime using dateTimeFormatter pattern
   * Return formatted string: "yyyy-MM-dd HH:mm:ss"
2. **setDateTimeFromString(String dateTime)**
   * Parse string using dateTimeFormatter
   * Convert to LocalDateTime object
   * Set dateTime attribute
3. **getDateToString()**
   * Format dateTime using dateFormatter pattern
   * Return formatted string: "EEE, dd MMM yyyy"

### 3. User Class Algorithm

**Purpose:** Represent a user entity with profile information and relationship tracking

**Attributes:**

* ID (int)
* firstName (String)
* lastName (String)
* email (String)
* password (String)
* likesIDs (ArrayList<Integer>) - List of liked post IDs
* friendsIDs (ArrayList<Integer>) - List of friend user IDs

**Steps:**

#### 3.1 Constructor

1. Create new User object
2. Initialize likesIDs as empty ArrayList<Integer>
3. Initialize friendsIDs as empty ArrayList<Integer>

#### 3.2 Basic Getter and Setter Methods

1. **getID()** - Return user ID
2. **setID(int ID)** - Set user ID
3. **getFirstName()** - Return first name
4. **setFirstName(String firstName)** - Set first name
5. **getLastName()** - Return last name
6. **setLastName(String lastName)** - Set last name
7. **getName()** - Concatenate and return firstName + " " + lastName
8. **getEmail()** - Return email address
9. **setEmail(String email)** - Set email address
10. **getPassword()** - Return password
11. **setPassword(String password)** - Set password

#### 3.3 Likes Management Methods

1. **getLikesIDs()** - Return ArrayList of liked post IDs
2. **setLikesIDs(ArrayList<Integer> likesIDs)** - Set liked post IDs list
3. **liked(Post p)**
   * Check if likesIDs contains post ID
   * Return true if post is liked, false otherwise
4. **like(Post p)**
   * Add post ID to likesIDs ArrayList
5. **dislike(Post p)**
   * Remove post ID from likesIDs ArrayList
   * Cast to Integer object before removal

#### 3.4 Friends Management Methods

1. **setFriends(ArrayList<User> friends)**
   * Initialize empty friendsIDs ArrayList
   * For each User in friends list:
     + Extract user ID
     + Add to friendsIDs ArrayList
2. **setFriendsIDs(ArrayList<Integer> friendsIDs)**
   * Set friendsIDs ArrayList directly
3. **getFriendsIDs()** - Return ArrayList of friend user IDs
4. **isFriend(User u)**
   * Check if friendsIDs contains user ID
   * Return true if user is friend, false otherwise
5. **addFriend(User f)**
   * Add friend's user ID to friendsIDs ArrayList
6. **removeFriend(User f)**
   * Remove friend's user ID from friendsIDs ArrayList
   * Cast to Integer object before removal

### 4. Database Class Algorithm

**Purpose:** Establish and manage MySQL database connection

**Attributes:**

* user (String) - Database username: "user"
* pass (String) - Database password: "#1#2#3%1%2%3"
* url (String) - Database URL: "jdbc:mysql://localhost/socialmedia"
* statement (Statement) - SQL Statement object for executing queries

**Steps:**

#### 4.1 Constructor

1. Initialize database credentials:
   * Set user = "user"
   * Set pass = "#1#2#3%1%2%3"
   * Set url = "jdbc:mysql://localhost/socialmedia"
2. Establish database connection:
   * Call DriverManager.getConnection(url, user, pass)
   * Store Connection object
3. Create Statement object:
   * Call connection.createStatement() with parameters:
     + ResultSet.TYPE\_SCROLL\_INSENSITIVE (allows cursor movement)
     + ResultSet.CONCUR\_READ\_ONLY (read-only result sets)
   * Store Statement object
4. Handle SQLException:
   * If connection fails, catch SQLException
   * Create Alert with error message
   * Pass null as parent component

#### 4.2 Getter Method

1. **getStatement()** - Return Statement object for query execution

# *SocialMediaPlatform Package Algorithm Flow*

## Overview

The SocialMediaPlatform package contains the main entry point of the application. It initializes the database connection and launches the user interface.

### 1. Main Class Algorithm

**Purpose:** Application entry point that initializes and starts the social media platform

**Package:** SocialMediaPlatform

**Class:** Main

**Dependencies:**

* Model.Database
* View.Welcome

**Steps:**

#### 1.1 main() Method

1. **Start:** Program execution begins
2. **Initialize Database:**
   * Create new Database object
   * Database constructor establishes MySQL connection
   * Statement object created for query execution
3. **Launch User Interface:**
   * Create new Welcome object
   * Pass Database object as parameter to Welcome constructor
   * Welcome window displays to user
4. **Wait:** Application runs and waits for user interaction
5. **End:** Program terminates when user closes application

**Detailed Flow:**

BEGIN PROGRAM

|

├─> Create Database Instance

| ├─> Connect to MySQL database

| ├─> Initialize Statement object

| └─> Handle connection errors (if any)

|

├─> Create Welcome View Instance

| ├─> Pass Database object to Welcome

| ├─> Initialize Welcome UI components

| └─> Display Welcome window

|

└─> Application Running

└─> Wait for user events

END PROGRAM (when window closed)

## Application Startup Sequence

### Phase 1: Database Initialization

1. Main method invoked by JVM
2. Database constructor called via new Database()
3. Database class attempts connection:
   * URL: "jdbc:mysql://localhost/socialmedia"
   * User: "user"
   * Password: "#1#2#3%1%2%3"
4. If connection successful:
   * Statement object created
   * Database ready for queries
5. If connection fails:
   * SQLException caught
   * Alert displayed to user
   * Database object created but statement is null

### Phase 2: UI Initialization

1. Welcome constructor called with Database parameter
2. Welcome window components initialized
3. Welcome window displayed to user
4. Event listeners registered for user actions
5. Application enters event-driven state

### Phase 3: Application Running

1. Event loop starts
2. Application waits for user input
3. User interactions trigger Controller methods
4. Controller methods use Database object for queries
5. Results displayed in View components
6. Loop continues until application exit

## Application Architecture Flow

Main.main()

|

|-- Creates Database object

| |

| |-- Establishes MySQL connection

| |-- Creates Statement object

| |-- Returns Database instance

|

|-- Creates Welcome view

|

|-- Receives Database object

|-- Displays login/signup interface

|-- User authenticates

|

|-- On successful login:

|

|-- Creates main application views

|-- Passes Database + User objects

|-- Application fully initialized

## Package Organization Summary

**SocialMediaPlatform Package:**

* Contains: Main.java only
* Purpose: Application entry point
* Dependencies: Model (Database), View (Welcome)

**Interaction with Other Packages:**

1. **Model Package:** Creates Database instance
2. **View Package:** Creates Welcome window
3. **Controller Package:** Used by View components (indirect)

**Execution Flow:**

SocialMediaPlatform.Main

↓

Model.Database (created)

↓

View.Welcome (created with Database)

↓

User Interaction

↓

Controller Classes (invoked by Views)

↓

Model Classes (data manipulation)

↓

Database (queries executed)

View Package Algorithm Flow

**Overview**

The View package contains all GUI components and user interface logic for the social media platform. It includes custom Swing components, screens for user interaction, and visual presentation of data.

**View Package Algorithm Flow**

**1. Alert Algorithm**

**Purpose:** Display error or informational messages in popup dialog

**Input:** Message content, parent JFrame

**Steps:**

1. Create JFrame (430×170 pixels, DISPOSE\_ON\_CLOSE)
2. Create panel with BorderLayout, white background, 15px padding
3. Add title "Alert" (size 24, blue, bold, CENTER, NORTH)
4. Add message content (size 18, black, bold, CENTER, CENTER)
5. Position relative to parent and set visible

**2. ChangePassword Algorithm**

**Purpose:** Interface for password change

**Input:** User, Database

**Steps:**

1. Create JFrame with BorderLayout panel
2. Add title "Change Password" (size 40, blue, bold, NORTH)
3. Create GridLayout (4×1) with fields:
   * Old Password text field
   * New Password text field
   * Confirm Password text field
   * Submit button
4. Validate on submit:
   * Check old password not empty and matches
   * Check new password not empty and length ≥ 6
   * Check confirm password matches new password
5. Call Controller.ChangePassword
6. If successful: Create Home view, show success alert, close frame

**3. Comment Algorithm (View Component)**

**Purpose:** Display single comment

**Input:** Model.Comment

**Steps:**

1. Create panel with BoxLayout (Y-axis), white background
2. Add header with author name (size 20, post color, bold, WEST)
3. Add content text area (size 18, post color, plain)
4. Add footer with date (size 15, post color, plain, EAST)
5. Calculate height: 90 + content height
6. Set size to 500 × calculated height

**4. Comments Algorithm**

**Purpose:** Display post with all comments and comment input

**Input:** User, Post, Database

**Steps:**

1. Create JFrame with BoxLayout panel
2. Add Post component and 7px spacing
3. Create comment input section (500×58 pixels):
   * Text area with hint "Type a comment"
   * Post button (35 radius, 81×37 pixels)
4. Post button validates input and calls CreateComment
5. Load existing comments using ReadPostComments
6. Add Comment component for each comment with 7px spacing
7. Add WindowListener: on close, create Home view

**5. CustomView Algorithm**

**Purpose:** Display different views (Friends, Posts, Comments, Likes)

**Input:** View name, User, Database

**Steps:**

1. Create JFrame with BoxLayout panel
2. Create header (500×50 pixels):
   * View title (size 20, black, bold, WEST)
   * Home icon with click listener (EAST)
3. Switch on view name:
   * **Friends:** Load ReadAllUsers, add Friend components
   * **Posts:** Load ReadUserPosts, add Post components
   * **Comments:** Load ReadUserComments, add panels
   * **Likes:** Load ReadUserLikes, add Post components
4. Add 7px spacing between each item

**6. Friend Algorithm (View Component)**

**Purpose:** Display user with Follow/Unfollow functionality

**Input:** Main user, Database, Friend user

**Steps:**

1. Create panel (500×67 pixels), BorderLayout, white background
2. Add friend name (size 20, post color, bold, WEST)
3. Create Follow button (35 radius, 81×37 pixels)
4. Create Unfollow label (size 16, blue, bold)
5. Set initial visibility:
   * If already friend: show Unfollow, hide Follow
   * If not friend: show Follow, hide Unfollow
6. Follow button: Call AddFriend, update user's friends list, toggle buttons
7. Unfollow label: Call RemoveFriend, update user's friends list, toggle buttons

**7. GUIConstants Algorithm**

**Purpose:** Define application color constants

**Constants:**

* white = #FFFFFF
* black = #000000
* blue = #02A1FC
* textFieldHint = #959595
* background = #F5F5F5
* textAreaHint = #858585
* hover = #f0f0f0
* post = #606060

**8. Home Algorithm**

**Purpose:** Main dashboard with timeline and navigation

**Input:** User, Database

**Steps:**

1. Create JFrame with BorderLayout
2. Create sidebar (182×1000 pixels, white, BoxLayout Y-axis):
   * Profile panel with user name (click → Modify view)
   * SideButton: Posts, Comments, Likes, Friends
3. Create main panel (BoxLayout Y-axis)
4. Create header (500×159 pixels):
   * Title "Home" (size 20, black, bold)
   * Post input text area with hint "Share your thoughts..."
   * Post button: validate, call CreatePost, show alert
5. Load timeline using GenerateTimeline
6. Add Post component for each timeline post with 7px spacing
7. Add sidebar (WEST), main panel (CENTER), spacing (EAST)

**9. JButton Algorithm (Custom Component)**

**Purpose:** Rounded button with blue background

**Input:** Text, radius, text size

**Steps:**

1. Extend JLabel with text and font (Segoe UI, bold)
2. Set opaque false, white foreground, CENTER alignment, HAND\_CURSOR
3. paintComponent: Fill rounded rectangle with blue color
4. paintBorder: Draw rounded rectangle border
5. contains: Check if point is within rounded shape

**10. JFrame Algorithm (Custom Component)**

**Purpose:** Standard application frame

**Steps:**

1. Extend javax.swing.JFrame with title "Social Media Platform"
2. Set background to GUIConstants.background
3. Set size 900×625 pixels
4. Set default close operation EXIT\_ON\_CLOSE
5. Center on screen (setLocationRelativeTo(null))

**11. JLabel Algorithm (Custom Component)**

**Purpose:** Styled label with custom font and color

**Input:** Text, text size, color, font style

**Steps:**

1. Extend javax.swing.JLabel
2. Set font to Segoe UI with specified size and style
3. Set text and foreground color

**12. JScrollPane Algorithm (Custom Component)**

**Purpose:** Styled scroll pane with custom scrollbar

**Input:** Component to scroll

**Steps:**

1. Extend javax.swing.JScrollPane
2. Remove borders, set background transparent
3. Create custom JScrollBar:
   * Set thumb color to textAreaHint
   * Hide increase/decrease arrows (0×0 buttons)
   * Hide horizontal scrollbar
4. Set vertical scrollbar width to 5px

**13. JTextArea Algorithm (Custom Component)**

**Purpose:** Text area with hint text and focus handling

**Input:** Hint text, text size, padding OR Text, size, color, style

**Steps:**

**Constructor 1 (with hint):**

1. Set font (Segoe UI, bold, specified size)
2. Set hint text with textAreaHint color
3. Add FocusListener:
   * focusGained: Clear hint if text equals hint, set black color
   * focusLost: Restore hint if text empty, set hint color
4. isEmpty: Return true if text equals hint or empty

**Constructor 2 (display only):**

1. Set font with specified style and size
2. Set text and color
3. Set non-editable
4. Set preferred width to 1000px

**14. JTextField Algorithm (Custom Component)**

**Purpose:** Rounded text field with hint text

**Input:** Hint text

**Steps:**

1. Extend javax.swing.JTextField
2. Set font (Segoe UI, bold, size 20), opaque false
3. Set hint text with textFieldHint color
4. Add 20px horizontal padding
5. Add FocusListener:
   * focusGained: Clear hint if text equals hint, set black color
   * focusLost: Restore hint if text empty, set hint color
6. paintComponent: Fill rounded rectangle (radius 45) with white
7. paintBorder: Draw rounded rectangle border
8. contains: Check if point within rounded shape
9. isEmpty: Return true if text equals hint or empty

**15. Login Algorithm**

**Purpose:** User authentication interface

**Input:** Database

**Steps:**

1. Create JFrame with BorderLayout panel
2. Add title "Login" (size 40, blue, bold, CENTER, NORTH)
3. Create GridLayout (3×1) with fields:
   * Email text field
   * Password text field
   * Login button
4. Login button validates:
   * Check email and password not empty
   * Call ReadUser controller
   * If successful: Create Home view, close frame
   * If failed: Show alert "Incorrect email or password"
5. Add "Create new account" link (click → Welcome view)

**16. Modify Algorithm**

**Purpose:** User profile editing interface

**Input:** User, Database

**Steps:**

1. Create JFrame with BorderLayout panel
2. Add title "Modify Your Profile" (size 40, blue, bold, NORTH)
3. Create GridLayout (4×1) with fields:
   * First Name (pre-filled with current value)
   * Last Name (pre-filled with current value)
   * Email (pre-filled with current value)
   * Submit button
4. Submit button validates:
   * Check all fields not empty
   * Check if email changed and already used
   * Call UpdateUser controller
   * If successful: Create Home view, show success alert
5. Add "Change Password" link (click → ChangePassword view)

**17. Post Algorithm (View Component)**

**Purpose:** Display single post with like/comment functionality

**Input:** User, Post, Database, JFrame

**Steps:**

1. Create panel (BoxLayout Y-axis), white background, 15-25px padding
2. Add header:
   * Author name (size 20, post color, bold, WEST)
   * Date (size 15, post color, plain, EAST)
3. Add content text area (size 18, post color, plain)
4. Add footer:
   * Like icon (toggle between like.png/liked.png)
   * Likes counter label
   * Comments counter label (click → Comments view)
5. Like icon click:
   * If not liked: Call LikePost, update icon, update user
   * If liked: Call DislikePost, update icon, update user
   * Refresh likes counter using ReadPostLikes
6. Comments label click: Create Comments view, close frame
7. Calculate height: 115 + content height
8. Set size to 500 × calculated height

**18. SideButton Algorithm (View Component)**

**Purpose:** Sidebar navigation button with icon and text

**Input:** Text, icon name, User, Database, JFrame

**Steps:**

1. Create panel (182×50 pixels), FlowLayout LEFT, white background
2. Add icon image from "pics/[iconname].png"
3. Add text label (size 17, textAreaHint color, bold)
4. Set cursor to HAND\_CURSOR
5. Add MouseListener:
   * mouseEntered: Set background to hover color
   * mouseExited: Set background to white
   * mouseClicked: Create CustomView with text, close frame

**19. Welcome Algorithm**

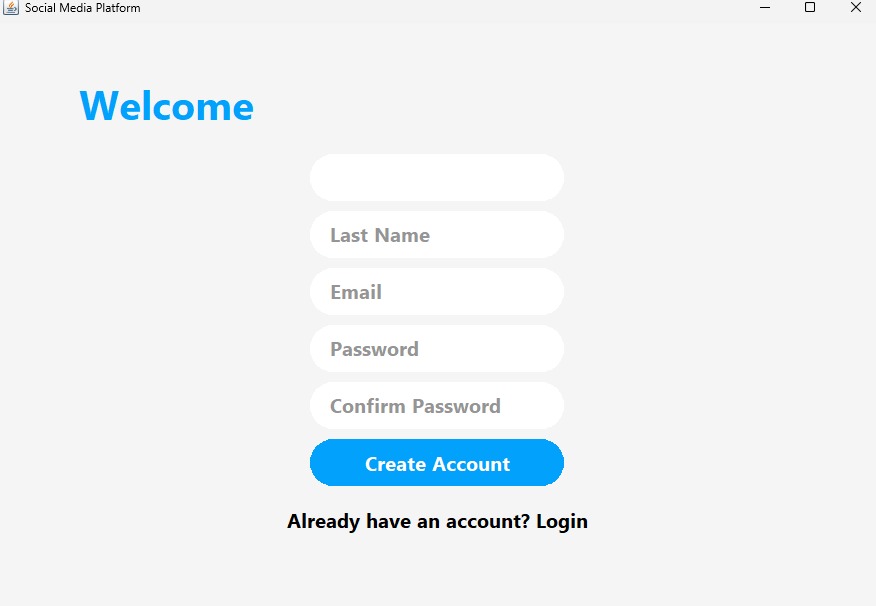
**Purpose:** User registration interface

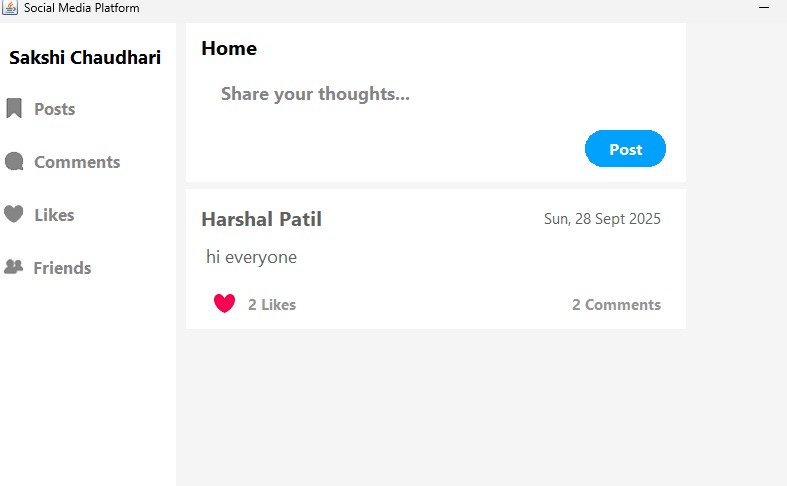
**Input:** Database

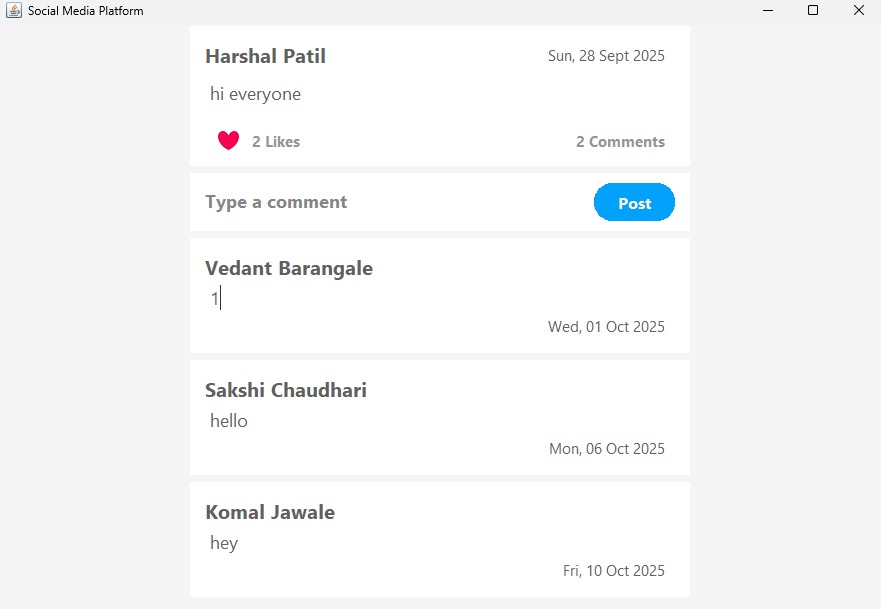
**Steps:**

1. Create JFrame with BorderLayout panel
2. Add title "Welcome" (size 40, blue, bold, NORTH)
3. Create GridLayout (6×1) with fields:
   * First Name text field
   * Last Name text field
   * Email text field
   * Password text field
   * Confirm Password text field
   * Create Account button
4. Create Account button validates:
   * Check all fields not empty
   * Check password length ≥ 6
   * Check passwords match
   * Create User object with form data
   * Call CreateUser controller
   * Check if email already used
   * If successful: Create Home view, close frame
   * If failed: Show appropriate alert
5. Add "Already have account? Login" link (click → Login view)

***OUTPUT :***

****

****

****