15. Facebook

Facebook is an online social networking service where users can connect with other users to post and read messages. Users access Facebook through their website interface or mobile apps.

System Requirements

We will focus on the following set of requirements while designing Facebook:

1. Each member should be able to add information about their basic profile, work experience, education, etc.
2. Any user of our system should be able to search other members, groups or pages by their name.
3. Members should be able to send and accept/reject friend requests from other members.
4. Members should be able to follow other members without becoming their friend.
5. Members should be able to create groups and pages, as well as join already created groups, and follow pages.
6. Members should be able to create new posts to share with their friends.
7. Members should be able to add comments to posts, as well as like or share a post or comment.
8. Members should be able to create privacy lists containing their friends. Members can link any post with a privacy list to make the post visible only to the members of that list.
9. Any member should be able to send messages to other members.
10. Any member should be able to add a recommendation for any page.
11. The system should send a notification to a member whenever there is a new message or friend request or comment on their post.
12. Members should be able to search through posts for a word.

Extended Requirement: Write a function to find a connection suggestion for a member.

Use Case Diagram

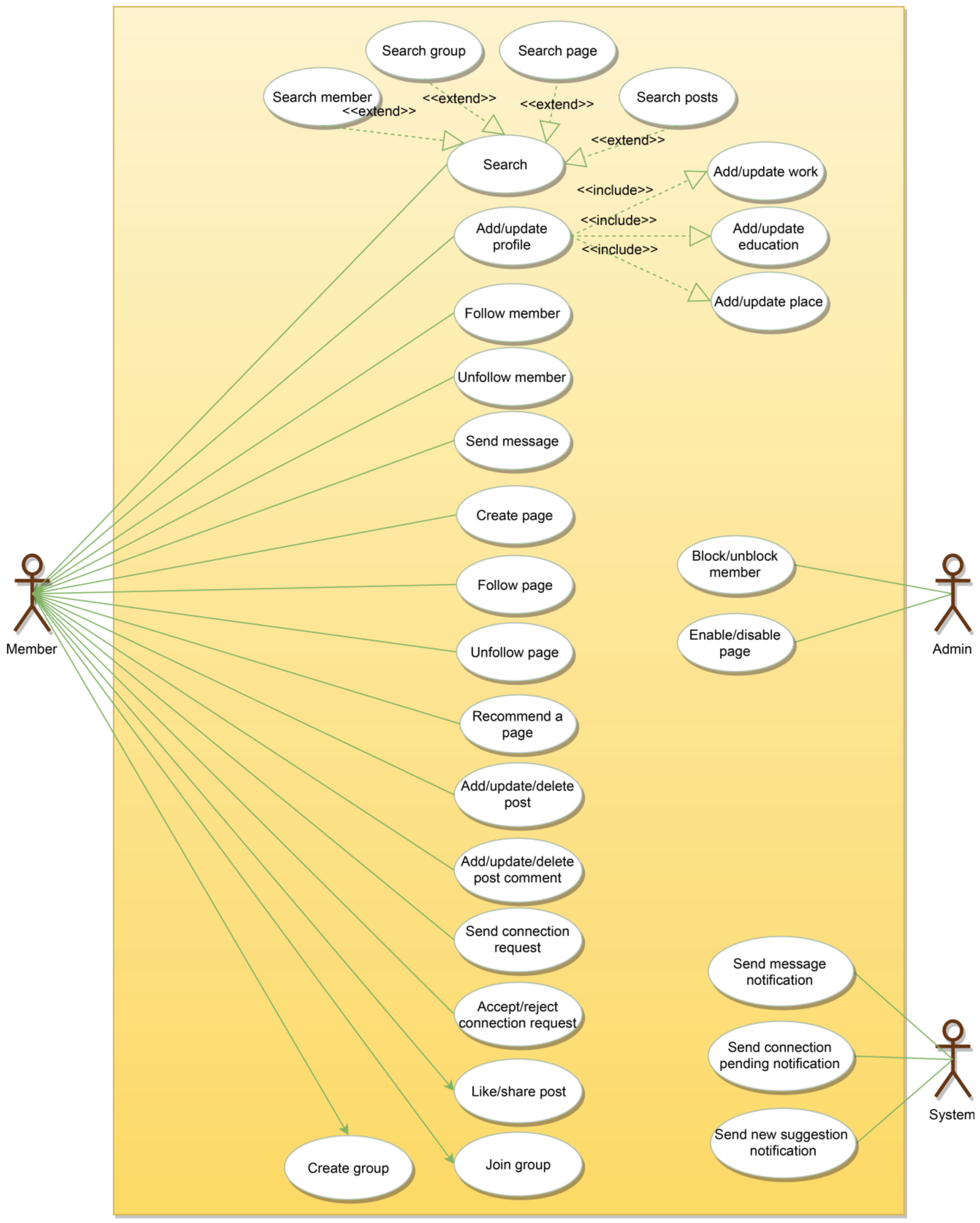
We have three main Actors in our system:

- **Member**: All members can search for other members, groups, pages, or posts, as well as send friend requests, create posts, etc.
- **Admin**: Mainly responsible for admin functions like blocking and unblocking a member, etc.
- **System**: Mainly responsible for sending notifications for new messages, friend requests, etc.

Here are the top use cases of our system:

- **Add/update profile**: Any member should be able to create their profile to reflect their work experiences, education, etc.
- **Search**: Members can search for other members, groups or pages. Members can send a friend request to other members.
- **Follow or Unfollow a member or a page**: Any member can follow or unfollow any other member or page.
- **Send message**: Any member can send a message to any of their friends.
- **Create post**: Any member can create a post to share with their friends, as well as like or add comments to any post visible to them.
- **Send notification**: The system will be able to send notifications for new messages, friend requests, etc.

Here is the use case diagram of Facebook:

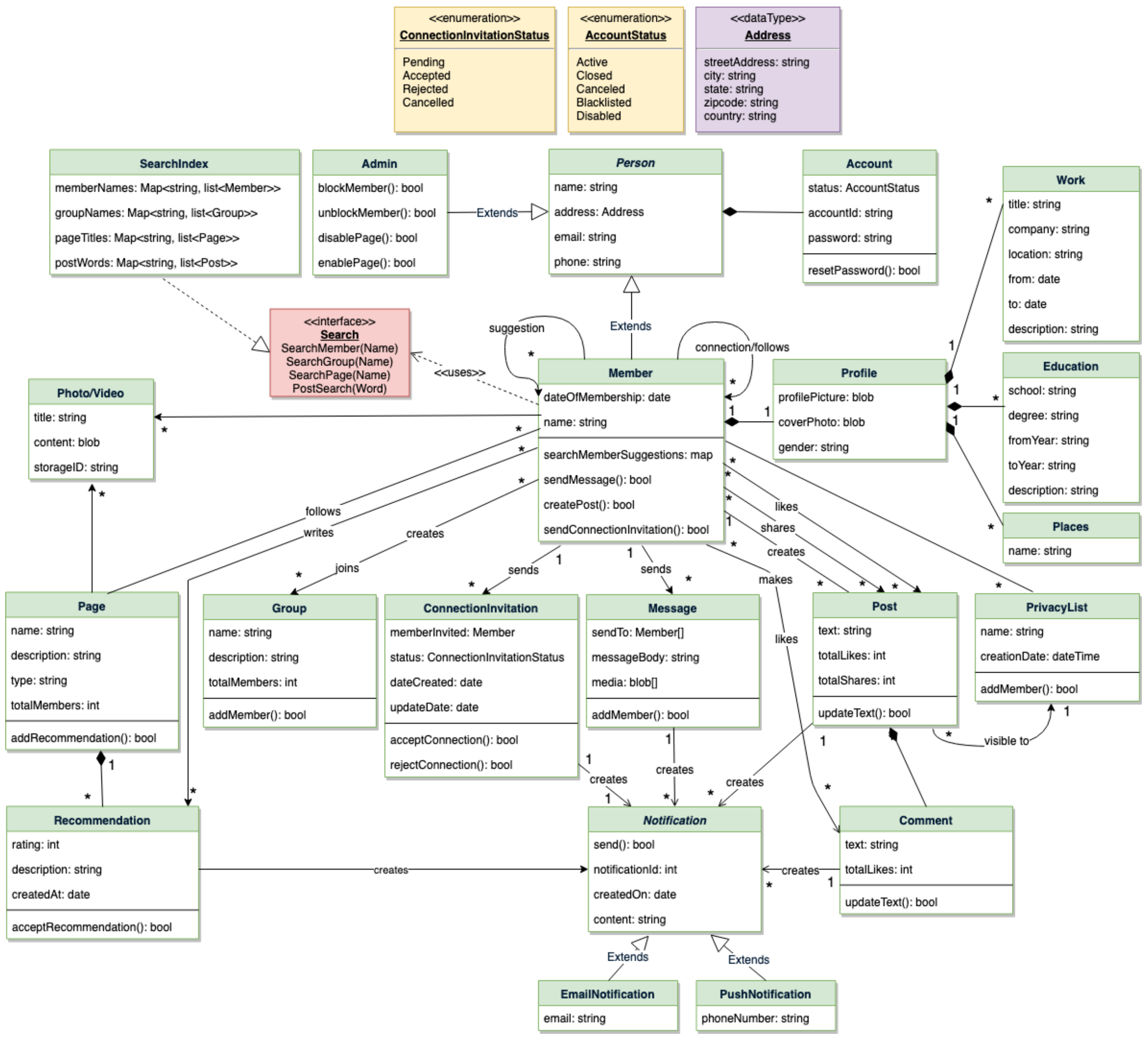


Use Case Diagram for Facebook

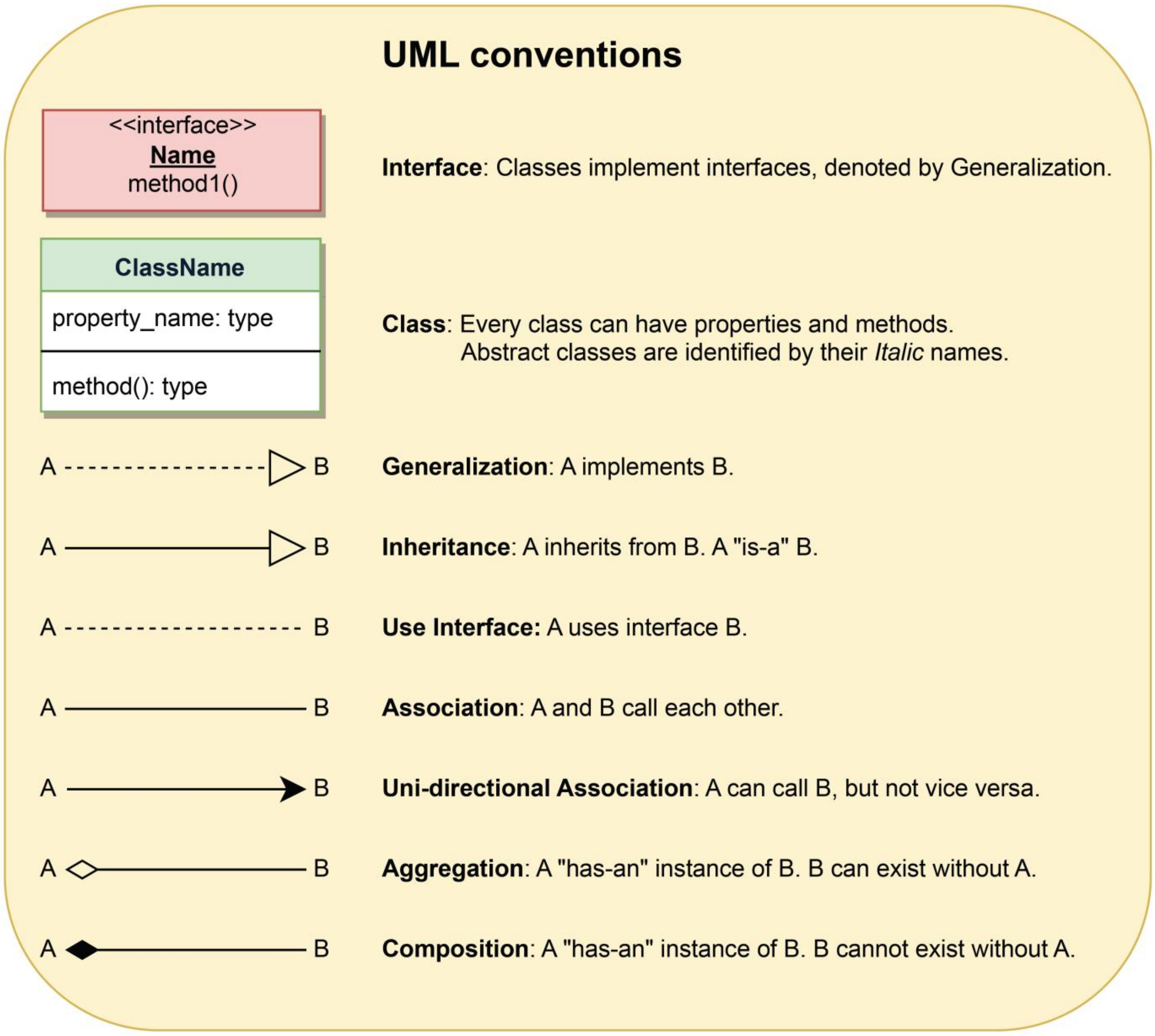
Class Diagram

Here are the main classes of the Facebook system:

- **Member**: This will be the main component of our system. Each member will have a profile which includes their Work Experiences, Education, etc. Members will be connected to other members and they can follow other members and pages. Members will also have suggestions to send friend requests to other members.
- **Search**: Our system will support searching for other members, groups and pages by their names, and through posts for any word.
- **Message**: Members can send messages to other members with text, photos, and videos.
- **Post**: Members can create posts containing text and media, as well as like and share a post.
- **Comment**: Members can add comments to posts as well as like any comment.
- **Group**: Members can create and join groups.
- **PrivacyList**: Members can create privacy lists containing their friends. Members can link any post with a privacy list, to make the post visible only to the members of that list.
- **Page**: Members can create pages that other members can follow, and share messages there.
- **Notification**: This class will take care of sending notifications to members. The system will be able to send a push notification or an email.



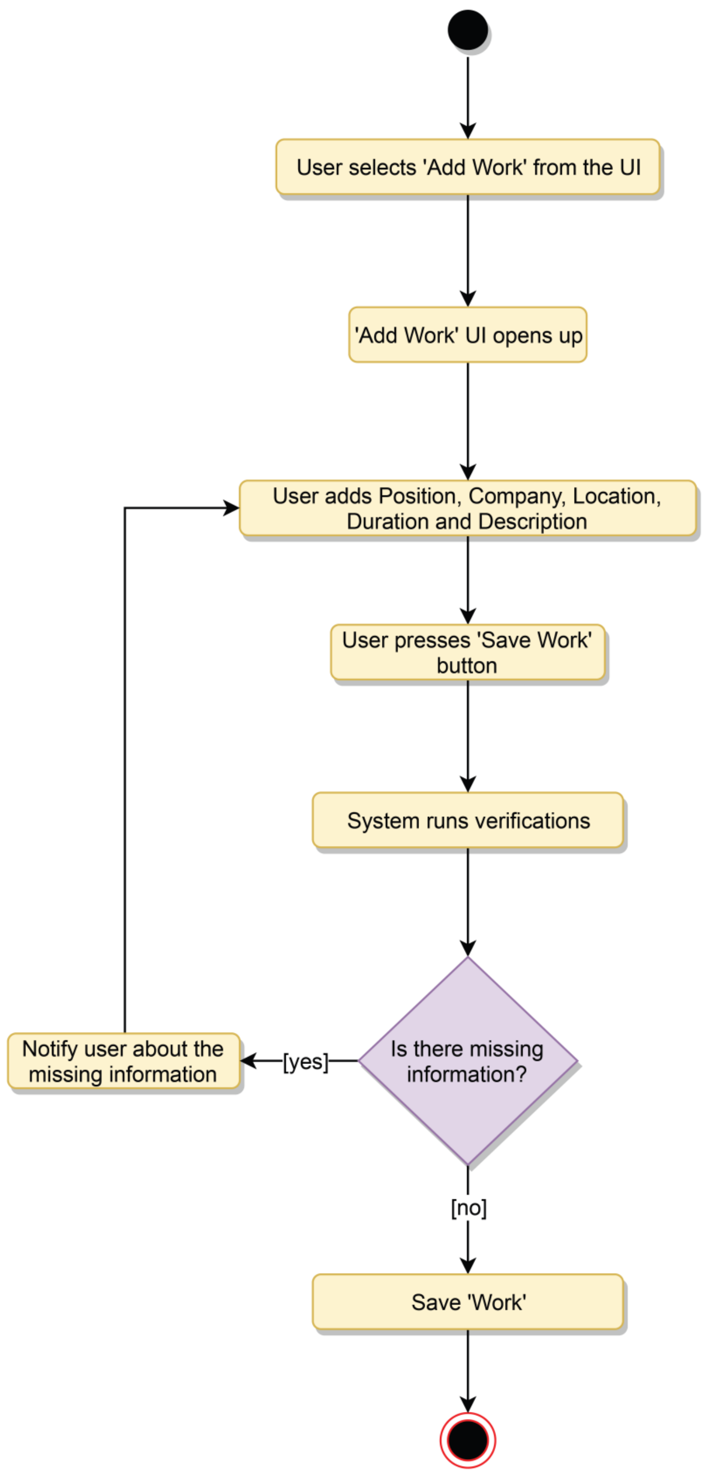
Class Diagram for Facebook


UML conventions
<<interface>>
Name
method1()
Interface: Classes implement interfaces, denoted by Generalization.
ClassName
property_name: type
method(): type
Class: Every class can have properties and methods.
Abstract classes are identified by their Italic names.
A
B
Generalization: A implements B.
Inheritance: A inherits from B. A "is-a" B.
Use Interface: A uses interface B.
Association: A and B call each other.
Uni-directional Association: A can call B, but not vice versa.
Aggregation: A "has-an" instance of B. B can exist without A.
Composition: A "has-an" instance of B. B cannot exist without A.

UML for Facebook

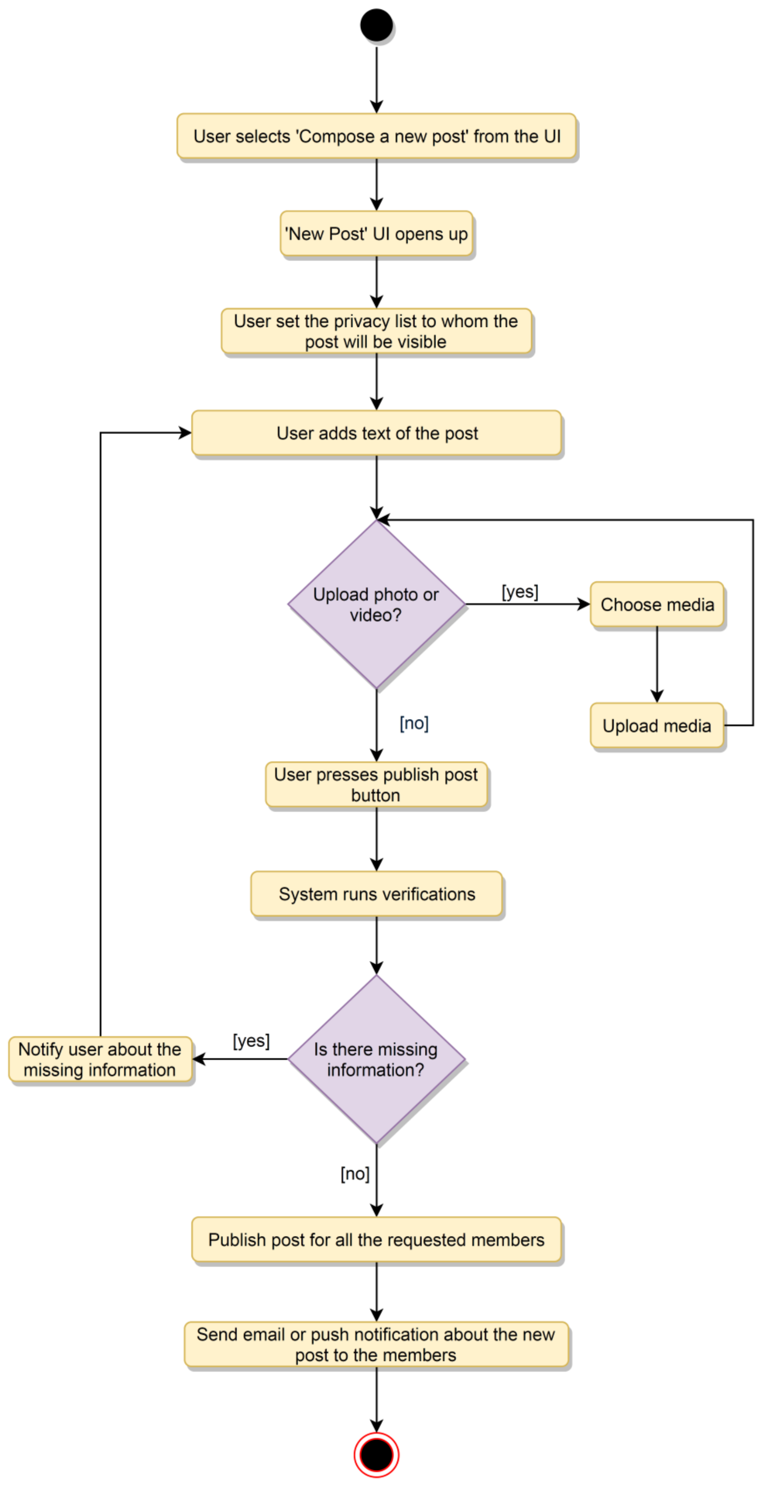
Activity Diagrams

Add work experience to profile: Any Facebook member can perform this activity. Here are the steps to add work experience to a member's profile:


User selects 'Add Work' from the UI
'Add Work' UI opens up
User adds Position, Company, Location, Duration and Description
User presses 'Save Work' button
System runs verifications
Is there missing information?
[yes]
Notify user about the missing information
[no]
Save 'Work'

Activity Diagram for Facebook Add Experience to Profile

Create a new post: Any Member can perform this activity. Here are the steps for creating a post:


User selects 'Compose a new post' from the UI
'New Post' UI opens up
User set the privacy list to whom the post will be visible
User adds text of the post
Upload photo or video?
[yes]
Choose media
Upload media
[no]
User presses publish post button
System runs verifications
Is there missing information?
[yes]
Notify user about the missing information
[no]
Publish post for all the requested members
Send email or push notification about the new post to the members

Activity Diagram for Facebook Create New Post

Code

Here is the high-level definition for the classes described above.

Enums, data types, and constants: Here are the required enums, data types, and constants:

```
import java.util.*;
import java.time.LocalDate;

enum ConnectionInvitationStatus {
   PENDING(1), ACCEPTED(2), REJECTED(3), CANCELED(4);
   private final int value;
   ConnectionInvitationStatus(int value) {
      this.value = value;
   }
}

enum AccountStatus {
   ACTIVE(1), CLOSED(2), CANCELED(3), BLACKLISTED(4), DISABLED(5);
   private final int value;
   AccountStatus(int value) {
      this.value = value;
   }
}

class Address {
   private String streetAddress;
   private String city;
   private String state;
   private String zipCode;
   private String country;

   public Address(String streetAddress, String city, String state, String zipCode, String country) {
      this.streetAddress = streetAddress;
      this.city = city;
      this.state = state;
      this.zipCode = zipCode;
      this.country = country;
   }
}

abstract class Person {
   protected String name;
```

```
    protected Address address;
    protected String email;
    protected String phone;
    protected Account account;

    public Person(String name, Address address, String email, String phone, Account account) {
        this.name = name;
        this.address = address;
        this.email = email;
        this.phone = phone;
        this.account = account;
    }
}

class Account {
    private String id;
    private String password;
    private AccountStatus status;

    public Account(String id, String password, AccountStatus status) {
        this.id = id;
        this.password = password;
        this.status = status;
    }

    public void resetPassword() {
        // Implement reset password logic here
    }
}

class Member extends Person {
    private String memberId;
    private LocalDate dateOfMembership;
    private List<String> memberFollows;
    private List<String> memberConnections;
    private List<String> pageFollows;
    private List<String> memberSuggestions;
    private List<ConnectionInvitation> connectionInvitations;
    private List<String> groupFollows;
    private Profile profile;

    public Member(String memberId, LocalDate dateOfMembership, String name) {
        super(name, null, null, null, null);
        this.memberId = memberId;
        this.dateOfMembership = dateOfMembership;
        this.profile = new Profile();
        this.memberFollows = new ArrayList<>();
        this.memberConnections = new ArrayList<>();
        this.pageFollows = new ArrayList<>();
        this.memberSuggestions = new ArrayList<>();
        this.connectionInvitations = new ArrayList<>();
        this.groupFollows = new ArrayList<>();
    }

    public void sendMessage(String message) {
        // Implement send message logic here
    }

    public void createPost(String post) {
        // Implement create post logic here
    }

    public void sendConnectionInvitation(ConnectionInvitation invitation) {
        // Implement send connection invitation logic here
    }

    public void searchMemberSuggestions() {
        // Implement search member suggestions logic here
    }
}

class Admin extends Person {
    public Admin(String name, Address address, String email, String phone, Account account) {
        super(name, address, email, phone, account);
    }

    public void blockUser(Member customer) {
        // Implement block user logic here
    }

    public void unblockUser(Member customer) {
        // Implement unblock user logic here
    }

    public void enablePage(Page page) {
        // Implement enable page logic here
    }
    public void disablePage(Page page) {
        // Implement disable page logic here
    }
}

class ConnectionInvitation {
    private Member memberInvited;
    private ConnectionInvitationStatus status;
    private LocalDate dateCreated;
    private LocalDate dateUpdated;

    public ConnectionInvitation(Member memberInvited, ConnectionInvitationStatus status) {
        this.memberInvited = memberInvited;
        this.status = status;
        this.dateCreated = LocalDate.now();
        this.dateUpdated = LocalDate.now();
    }

    public void acceptConnection() {
        // Implement accept connection logic here
    }

    public void rejectConnection() {
        // Implement reject connection logic here
    }
}

class Profile {
    private String profilePicture;
    private String coverPhoto;
    private String gender;
    private List<String> workExperiences;
    private List<String> educations;
    private List<String> places;
    private List<String> stats;

    public Profile() {
        this.workExperiences = new ArrayList<>();
        this.educations = new ArrayList<>();
        this.places = new ArrayList<>();
        this.stats = new ArrayList<>();
    }

    public void addWorkExperience(String work) {
        // Implement add work experience logic here
    }

    public void addEducation(String education) {
        // Implement add education logic here
    }

    public void addPlace(String place) {
        // Implement add place logic here
    }
}

class Work {
    private String title;
    private String company;
    private String location;
    private LocalDate from;
    private LocalDate to;
    private String description;

    public Work(String title, String company, String location, LocalDate from, LocalDate to, String
description) {
        this.title = title;
        this.company = company;
        this.location = location;
        this.from = from;
        this.to = to;
        this.description = description;
    }
}

class Page {
    private String pageId;
    private String name;
    private String description;
    private String type;
    private int totalMembers;
    private List<Recommendation> recommendations;

    public Page(String pageId, String name, String description, String type, int totalMembers) {
        this.pageId = pageId;
        this.name = name;
        this.description = description;
        this.type = type;
        this.totalMembers = totalMembers;
        this.recommendations = new ArrayList<>();
    }

    public List<Recommendation> getRecommendations() {
        return recommendations;
    }
}

class Recommendation {
    private String recommendationId;
    private int rating;
    private String description;
    private LocalDate createdAt;

    public Recommendation(String recommendationId, int rating, String description) {
        this.recommendationId = recommendationId;
        this.rating = rating;
        this.description = description;
        this.createdAt = LocalDate.now();
    }
}

class Group {
    private String groupId;
    private String name;
    private String description;
    private int totalMembers;
    private List<Member> members;

    public Group(String groupId, String name, String description, int totalMembers) {
        this.groupId = groupId;
        this.name = name;
        this.description = description;
        this.totalMembers = totalMembers;
        this.members = new ArrayList<>();
    }

    public void addMember(Member member) {
        // Implement add member logic here
    }

    public void updateDescription(String description) {
        // Implement update description logic here
    }
}
```

```
class Post {
   private String postId;
   private String text;
   private int totalLikes;
   private int totalShares;
   private Member owner;

   public Post(String postId, String text, int totalLikes, int totalShares, Member owner) {
      this.postId = postId;
      this.text = text;
      this.totalLikes = totalLikes;
      this.totalShares = totalShares;
      this.owner = owner;
   }
}

class Message {
   private String messageId;
   private Member sentTo;
   private String messageBody;
   private String media;

   public Message(String messageId, Member sentTo, String messageBody, String media) {
      this.messageId = messageId;
      this.sentTo = sentTo;
      this.messageBody = messageBody;
      this.media = media;
   }

   public void addMember(Member member) {
      // Implement add member logic here
   }
}

class Comment {
   private String commentId;
   private String text;
   private int totalLikes;
   private Member owner;

   public Comment(String commentId, String text, int totalLikes, Member owner) {
      this.commentId = commentId;
      this.text = text;
      this.totalLikes = totalLikes;
      this.owner = owner;
```

```
    }
}

interface Search {
    Member searchMember(String name);
    Group searchGroup(String name);
    Page searchPage(String name);
    Post searchPost(String word);
}

class SearchIndex implements Search {
    private Map<String, Set<Member>> memberNames;
    private Map<String, Set<Group>> groupNames;
    private Map<String, Set<Page>> pageTitles;
    private Map<String, Set<Post>> posts;

    public SearchIndex() {
        memberNames = new HashMap<>();
        groupNames = new HashMap<>();
        pageTitles = new HashMap<>();
        posts = new HashMap<>();
    }

    public void addMember(Member member) {
        memberNames.computeIfAbsent(member.name, k -> new HashSet<>()).add(member);
    }

    public void addGroup(Group group) {
        // Implement add group logic here
    }

    public void addPage(Page page) {
        // Implement add page logic here
    }

    public void addPost(Post post) {
        // Implement add post logic here
    }

    public Member searchMember(String name) {
        return memberNames.getOrDefault(name,
Collections.emptySet()).stream().findFirst().orElse(null);
    }

    public Group searchGroup(String name) {
        return groupNames.getOrDefault(name,
Collections.emptySet()).stream().findFirst().orElse(null);
    }

    public Page searchPage(String name) {
        return pageTitles.getOrDefault(name, Collections.emptySet()).stream().findFirst().orElse(null);
    }

    public Post searchPost(String word) {
        return posts.getOrDefault(word, Collections.emptySet()).stream().findFirst().orElse(null);
    }
}
```