Social Media Application - Moderate Level Plan

Alright! Let's make this a **proper, feature-rich social media app** while still meeting all the CSE215L requirements. This will be impressive but doable.

1. Application Features (Moderate Complexity)

Core Features:

- **User System:** Registration, Login, Logout, Profile management
- **Post System:** Create posts (text/image reference), view feed, delete own posts
- **Like System:** Like/unlike posts, see like count
- **Comment System:** Add comments to posts, view all comments
- **Follow System:** Follow/unfollow users, see following list
- **Search System:** Search users and posts
- **Persistent Storage:** All data saved in text files

Additional Cool Features:

- **Feed Algorithm:** Show posts from followed users first
- **Post Sorting:** Sort by most liked or most recent
- **User Statistics:** Post count, follower count, following count
- **Notifications Count:** Track new likes/comments (simple counter)
- **Edit Profile:** Update bio and email

2. Complete Class Structure (12-15 classes)

Core Entity Classes:

User Class

- **Fields:** `userId`, `username`, `password`, `email`, `bio`, `ArrayList<String> followerIds`, `ArrayList<String> followingIds`, `dateJoined`
- **Methods:** `addFollower()`, `removeFollower()`, `follow()`, `unfollow()`, `isFollowing()`, `getFollowerCount()`, `updateProfile()`
- **Encapsulation:** All private fields with getters/setters
- **Purpose:** Represents a user account

Post Class (Abstract Parent)

- **Fields:** `postId`, `authorId`, `content`, `timestamp`, `ArrayList<Like> likes`, `ArrayList<Comment> comments`

- **Methods:** `abstract String getPostType()`, `abstract void display()`, `addLike()`, `removeLike()`, `addComment()`, `getLikeCount()`, `getCommentCount()`
- **Implements:** `Likeable` interface, `Commentable` interface
- **Purpose:** Base class for all post types

TextPost Class (Child)

- **Extends:** Post
- **Additional Fields:** `int wordCount`
- **Override:** `getPostType()`, `display()`
- **Methods:** `calculateWordCount()`
- **Purpose:** Text-only posts

MediaPost Class (Child)

- **Extends:** Post
- **Additional Fields:** `String mediaPath`, `String mediaType` (image/video)
- **Override:** `getPostType()`, `display()`
- **Purpose:** Posts with media reference

Like Class

- **Fields:** `likeId`, `userId`, `postId`, `timestamp`
- **Methods:** Getters/setters
- **Purpose:** Represents a like on a post

Comment Class

- **Fields:** `commentId`, `postId`, `userId`, `username`, `content`, `timestamp`
- **Methods:** Getters/setters, `display()`
- **Purpose:** Represents a comment on a post

Manager/Service Classes:

AuthenticationManager Class

- **Fields:** `UserManager userManager`, `User currentUser`
- **Methods:** `register()`, `login()`, `logout()`, `isLoggedIn()`, `getCurrentUser()`
- **Exceptions:** Throws `InvalidCredentialsException`, `DuplicateUserException`
- **Purpose:** Handles user authentication
- **Composition:** Strong relationship with UserManager

UserManager Class

- **Fields:** `ArrayList<User> users`

- **Methods:** `addUser()`, `getUserById()`, `getUserByUsername()`, `updateUser()`, `searchUsers()`, `getAllUsers()`
- **Implements:** `Searchable` interface
- **Purpose:** Manages all user operations
- **Aggregation:** Weak relationship with User (users can exist independently)

PostManager Class

- **Fields:** `ArrayList<Post> posts`
- **Methods:** `createPost()` (overloaded), `deletePost()`, `getPostById()`, `getAllPosts()`, `getPostsByUser()`, `getPostsByFollowing()`, `searchPosts()`, `sortPostsByLikes()`, `sortPostsByDate()`
- **Implements:** `Searchable` interface
- **Purpose:** Manages all post operations
- **Aggregation:** Weak relationship with Post

FeedManager Class

- **Fields:** `PostManager postManager`, `UserManager userManager`
- **Methods:** `generateFeed()`, `getFollowingFeed()`, `getExploreFeed()`
- **Purpose:** Generates personalized feeds for users
- **Composition:** Depends on PostManager and UserManager

SocialMediaSystem Class

- **Fields:** `AuthenticationManager authManager`, `UserManager userManager`, `PostManager postManager`, `FeedManager feedManager`, `FileHandler fileHandler`
- **Methods:** `initialize()`, `saveAllData()`, `loadAllData()`
- **Purpose:** Main system coordinator - ties everything together
- **Composition:** Strong relationships with all managers

FileHandler Class

- **Methods:**
  - `saveUsers()`, `loadUsers()`
  - `savePosts()`, `loadPosts()`
  - `saveLikes()`, `loadLikes()`
  - `saveComments()`, `loadComments()`
  - `saveFollowRelations()`, `loadFollowRelations()`
- **Purpose:** Handles all file I/O operations
- **Static methods or singleton pattern**

Interfaces:

Searchable Interface

- **Methods:** `ArrayList<Object> searchByKeyword(String keyword)`, `Object searchById(String id)`
- **Implemented by:** UserManager, PostManager
- **Purpose:** Abstraction for search functionality
- **Justification:** Different entities need different search logic

Likeable Interface

- **Methods:** `void addLike(Like like)`, `void removeLike(String likeId)`, `int getLikeCount()`, `boolean isLikedBy(String userId)`
- **Implemented by:** Post
- **Purpose:** Any content that can be liked

Commentable Interface

- **Methods:** `void addComment(Comment comment)`, `ArrayList<Comment> getComments()`, `int getCommentCount()`
- **Implemented by:** Post
- **Purpose:** Any content that can have comments

Custom Exception Classes:

InvalidCredentialsException

- **Extends:** Exception
- **Purpose:** Thrown when login credentials are wrong
- **Message:** "Invalid username or password"

DuplicateUserException

- **Extends:** Exception
- **Purpose:** Thrown when username already exists during registration
- **Message:** "Username already exists"

PostNotFoundException

- **Extends:** Exception
- **Purpose:** Thrown when post doesn't exist
- **Message:** "Post not found"

3. OOP Requirements - How They're Met

Encapsulation ✓

- All entity classes (User, Post, Like, Comment) use private fields

- Public getters/setters for controlled access
- Validation in setters (e.g., password minimum length, username format)
- Example: `setPassword()` validates length before setting

Inheritance ✓

- **Post (abstract)** → TextPost, MediaPost
- Shared fields: `postId`, `authorId`, `content`, `timestamp`, `likes`, `comments`
- Shared methods: `addLike()`, `addComment()`, `getLikeCount()`
- Children override: `getPostType()`, `display()`
- **Why?** Different post types share behavior but display differently

Polymorphism ✓

Method Overriding:

- `Post.display()` → overridden in TextPost (shows word count) and MediaPost (shows media type)
- `Post.getPostType()` → returns "Text" or "Media"
- `Searchable.searchByKeyword()` → different logic in UserManager (searches usernames) vs PostManager (searches post content)

Method Overloading:

- `PostManager.createPost(String content, String authorId)` → creates TextPost
- `PostManager.createPost(String content, String authorId, String mediaPath, String mediaType)` → creates MediaPost
- `UserManager.searchUsers(String username)` → search by username
- `UserManager.searchUsers(String firstName, String lastName)` → search by full name
- `Comment.Comment(String postId, String userId, String content)` → basic constructor
- `Comment.Comment(String commentId, String postId, String userId, String content, String timestamp)` → for loading from file

Abstraction ✓

- **Abstract Class:** Post with abstract methods `getPostType()`, `display()`
- **Interfaces:** Searchable, Likeable, Commentable
- **Justification:**
  - Post is a concept; concrete types must define their representation
  - Searchable allows different search implementations without exposing logic
  - Likeable/Commentable ensure consistent behavior across likeable content

Composition (Strong "has-a") ✓

- `SocialMediaSystem` **HAS-A** `AuthenticationManager` (can't function without it)
- `SocialMediaSystem` **HAS-A** `PostManager` (can't function without it)
- `FeedManager` **HAS-A** `PostManager` (needs it to generate feeds)
- **Explanation:** These components are essential and tightly coupled

Aggregation (Weak "has-a") ✓

- `UserManager` **HAS-A** `ArrayList<User>` (users exist independently)
- `PostManager` **HAS-A** `ArrayList<Post>` (posts exist independently)
- `Post` **HAS-A** `ArrayList<Comment>` (comments can exist without the post object)
- **Explanation:** Collections hold references; objects can exist separately

Exception Handling ✓

Built-in Exceptions:

1. **IOException** - During file read/write operations in FileHandler
2. **NumberFormatException** - When parsing IDs from text files
3. **NullPointerException** - Can be caught when accessing objects that don't exist

Custom Exceptions:

1. **InvalidCredentialsException** - Wrong login
2. **DuplicateUserException** - Username taken
3. **PostNotFoundException** - Post doesn't exist

Implementation Examples:

```
// Login
try {
    authManager.login(username, password);
} catch (InvalidCredentialsException e) {
    showError("Wrong username or password");
} catch (IOException e) {
    showError("Cannot access user data");
} finally {
    logLoginAttempt(username);
}

// Registration
try {
    authManager.register(username, password, email);
} catch (DuplicateUserException e) {
    showError("Username already taken");
} catch (IOException e) {
    showError("Cannot save user data");
}

// File Loading
```

```
try {
    fileHandler.loadUsers();
} catch (IOException e) {
    System.err.println("Cannot load users");
} catch (NumberFormatException e) {
    System.err.println("Corrupted user data");
} finally {
    System.out.println("Load attempt completed");
}
```

File Handling ✓

Five Text Files:

users.txt

```
userId|username|password|email|bio|dateJoined
1|john_doe|pass123|john@email.com|Hello!|2025-11-20
2|jane_smith|pass456|jane@email.com|Love coding|2025-11-21
```

posts.txt

```
postId|postType|authorId|content|timestamp|wordCount|mediaPath|mediaType
1|TextPost|1|Hello World!|2025-11-20 10:30|2|||
2|MediaPost|2|Check this|2025-11-20 11:00||photo.jpg|image
```

likes.txt

```
likeId|userId|postId|timestamp
1|2|1|2025-11-20 10:35
2|1|2|2025-11-20 11:05
```

comments.txt

```
commentId|postId|userId|username|content|timestamp
1|1|2|jane_smith|Nice post!|2025-11-20 10:40
2|1|1|john_doe|Thanks!|2025-11-20 10:45
```

follows.txt

```
followerId|followingId|timestamp
1|2|2025-11-20 09:00
2|1|2025-11-20 09:30
```

File Operations:

- **On App Start:** Load all 5 files, populate ArrayLists in managers
- **On Actions:** Update ArrayLists in memory
- **On Exit / Save:** Write all ArrayLists back to files
- **Persistence:** Data survives between program runs

4. GUI Structure (JavaFX + Scene Builder)

Scene/FXML Files (7 screens):

1. LoginScreen.fxml

- **Components:**
  - TextField (username)
  - PasswordField (password)
  - Button (Login)
  - Button (Go to Register)
  - Label (error message)
- **Controller:** `LoginController.java`
- **Actions:**
  - Login → validate → open Dashboard
  - Register → open RegisterScreen

2. RegisterScreen.fxml

- **Components:**
  - TextField (username, email)
  - PasswordField (password, confirm password)
  - TextArea (bio)
  - Button (Register)
  - Button (Back to Login)
  - Label (error/success message)
- **Controller:** `RegisterController.java`
- **Actions:**
  - Register → validate → save user → back to Login

3. DashboardScreen.fxml

- **Components:**
  - MenuBar (Home, Profile, Search, Logout)
  - ListView (posts feed)
  - Button (Create Post)
  - Button (Refresh Feed)
  - ComboBox (Sort: Most Recent / Most Liked / Following)
  - Label (welcome message with username)
- **Controller:** `DashboardController.java`
- **Actions:**
  - Display posts from PostManager
  - Click post → open PostDetailScreen
  - Create Post → open CreatePostScreen

4. CreatePostScreen.fxml

- **Components:**
  - TextArea (post content)
  - RadioButton (Text Post / Media Post)
  - TextField (media path - enabled if Media Post selected)
  - ComboBox (media type: Image/Video - if Media Post)
  - Button (Post)
  - Button (Cancel)
  - Label (character/word count)
- **Controller:** `CreatePostController.java`
- **Actions:**
  - Post → create Post object → save → refresh feed → back to Dashboard

5. PostDetailScreen.fxml

- **Components:**
  - Label (author, timestamp, content)
  - Label (like count)
  - Button (Like/Unlike)
  - Button (Delete - only if user is author)
  - ListView (comments)
  - TextArea (add comment)
  - Button (Post Comment)
  - Button (Back)
- **Controller:** `PostDetailController.java`
- **Actions:**
  - Like → update like count → save
  - Comment → add to post → refresh comments → save
  - Delete → remove post → back to Dashboard

6. ProfileScreen.fxml

- **Components:**
  - Label (username, email, bio, dateJoined)
  - Label (Posts: X, Followers: Y, Following: Z)
  - ListView (user's posts)
  - Button (Edit Profile - if own profile)
  - Button (Follow/Unfollow - if other user's profile)
  - Button (Back)
- **Controller:** `ProfileController.java`
- **Actions:**
  - Load user data
  - Edit Profile → open EditProfileScreen
  - Follow → update followers → save

7. SearchScreen.fxml

- **Components:**
  - TextField (search query)
  - RadioButton (Search Users / Search Posts)
  - Button (Search)
  - ListView (search results)
  - Label (no results message)
  - Button (Back)
- **Controller:** `SearchController.java`
- **Actions:**
  - Search → display results in ListView
  - Click result → open Profile or PostDetail

8. EditProfileScreen.fxml

- **Components:**
  - TextField (email - pre-filled)
  - TextArea (bio - pre-filled)
  - PasswordField (new password - optional)
  - PasswordField (confirm new password)
  - Button (Save Changes)
  - Button (Cancel)
  - Label (success/error message)
- **Controller:** `EditProfileController.java`
- **Actions:**
  - Save → update user → save to file → back to Profile

GUI Components Used (10+):

1. TextField
2. PasswordField
3. TextArea
4. Button
5. Label
6. ListView
7. MenuBar
8. RadioButton
9. ComboBox
10. TabPane (optional for Dashboard)

Navigation Flow:

```
Login → Dashboard → Create Post / Post Detail / Profile / Search
                ↓
            Logout → Login
```

5. Development Timeline (Realistic)

Week 1: Backend (Classes + Logic)

Day 1-2: Core Entity Classes

- Create User, Post (abstract), TextPost, MediaPost, Like, Comment
- Implement encapsulation (private fields, getters/setters)
- Add validation in setters
- Test by creating objects in main method

Day 3-4: Interfaces and Manager Classes

- Create Searchable, Likeable, Commentable interfaces
- Create UserManager, PostManager
- Implement basic CRUD operations
- Test with sample data

Day 5-6: Authentication and System Integration

- Create AuthenticationManager
- Create FeedManager
- Create SocialMediaSystem (ties everything together)
- Implement polymorphic methods (overloading, overriding)
- Test complete backend without GUI

Day 7: Exception Handling

- Create custom exception classes
- Add try-catch blocks in managers
- Test error scenarios (wrong login, duplicate user, etc.)

Week 2: File Handling + GUI

Day 8-9: File Handling

- Create FileHandler class
- Implement save methods for all data
- Implement load methods for all data
- Test: add data → save → close → reopen → verify data persists
- Handle IOExceptions, NumberFormatExceptions

Day 10-12: GUI Design (Scene Builder)

- Design all 8 FXML files in Scene Builder
- Create controller classes for each screen
- Set up basic navigation between screens
- No backend integration yet - just UI design

Day 13-15: GUI-Backend Integration

- Connect controllers to SocialMediaSystem
- Implement event handlers (button clicks, list selections)
- Load data into GUI components (ListViews, Labels)
- Test all features end-to-end
- Handle GUI exceptions (show error messages in Labels)

Week 3: Testing, UML, Report

Day 16-17: Testing & Bug Fixes

- Test all features thoroughly
- Test edge cases (empty fields, long text, special characters)
- Test file persistence (close and reopen multiple times)
- Fix any bugs

Day 18-19: UML Diagram

- Create class diagram with all 15 classes
- Show inheritance arrows
- Show interface implementations
- Show composition (filled diamonds)
- Show aggregation (hollow diamonds)
- Use draw.io or Lucidchart

Day 20-21: Write Report

- Follow the report structure from PDF
- Explain each OOP concept with code snippets
- Add UML diagram
- Add screenshots of all 8 screens
- Proofread and finalize

Day 22: Final Review

- Review code
- Review report
- Prepare for demonstration
- Practice explaining your project

6. Report Structure (4-6 pages)

Page 1: Title Page + Introduction

- Group member names, IDs, section

- Project title: "Social Media Application"
- Brief introduction (3-4 sentences about what the app does)

Page 2: System Overview + OOP Concepts (Part 1)

- Domain description (social media features)
- **Encapsulation:** Explain with User class example + code snippet
- **Inheritance:** Explain Post hierarchy with UML snippet + code
- **Polymorphism:** Show overriding and overloading examples with code

Page 3: OOP Concepts (Part 2)

- **Abstraction:** Explain Post abstract class and interfaces + justification
- **Composition:** SocialMediaSystem with managers + UML snippet
- **Aggregation:** UserManager with Users + UML snippet
- **Exception Handling:** Show try-catch examples + custom exceptions

Page 4: File Handling + Class Descriptions

- **File Handling:** Explain 5 text files, format, load/save process + code snippet
- **Class Descriptions:** 2-3 sentences per major class

Page 5: UML Diagram

- Full class diagram with all relationships

Page 6: GUI Screenshots + Conclusion

- 8 screenshots (one per screen) with captions
- Brief conclusion (3-4 sentences)

7. Key Implementation Tips

Polymorphism Example:

```
// In PostManager - Method Overloading
public Post createPost(String content, String authorId) {
    return new TextPost(content, authorId);
}

public Post createPost(String content, String authorId, String mediaPath,
String type) {
    return new MediaPost(content, authorId, mediaPath, type);
}

// In Post subclasses - Method Overriding
// TextPost
```

```
@Override
public void display() {
    System.out.println("[TEXT POST] " + content + " (Words: " + wordCount +
")");
}

// MediaPost
@Override
public void display() {
    System.out.println("[MEDIA POST] " + content + " [" + mediaType + ": " +
mediaPath + "]");
}
```

Composition vs Aggregation:

```
// Composition - SocialMediaSystem OWNS managers
class SocialMediaSystem {
    private PostManager postManager = new PostManager(); // Created here
    private UserManager userManager = new UserManager(); // Created here
    // If System dies, managers die too
}

// Aggregation - UserManager REFERENCES users
class UserManager {
    private ArrayList<User> users; // Users exist independently
    // Users can exist without UserManager
}
```

File Handling Example:

```
// In FileHandler
public void savePosts(ArrayList<Post> posts) throws IOException {
    BufferedWriter writer = new BufferedWriter(new FileWriter("posts.txt"));
    try {
        for (Post post : posts) {
            String line = post.getPostId() + "|" +
                          post.getPostType() + "|" +
                          post.getAuthorId() + "|" +
                          post.getContent() + "|" +
                          post.getTimestamp();
            writer.write(line);
            writer.newLine();
        }
    } finally {
        writer.close();
    }
}
```

Exception Handling in GUI:

```
// In LoginController
@FXML
private void handleLogin() {
    try {
```

```
            authManager.login(usernameField.getText(), passwordField.getText());
            // Navigate to Dashboard
            loadDashboard();
        } catch (InvalidCredentialsException e) {
            errorLabel.setText("Invalid username or password!");
            errorLabel.setVisible(true);
        } catch (IOException e) {
            errorLabel.setText("Cannot access user data. Please try again.");
            errorLabel.setVisible(true);
        } finally {
            // Log the attempt
            System.out.println("Login attempt: " + usernameField.getText());
        }
}
```

8. Cool Features to Add Time Permitting

- **Post timestamp formatting:** "2 hours ago", "1 day ago"
- **Follow suggestions:** Show users you don't follow yet
- **Post edit history:** Track edits (just in memory, don't need to save)
- **Username validation:** No spaces, minimum length
- **Bio character limit:** Max 150 characters
- **Post character limit:** Max 280 characters (like Twitter)
- **Sort comments:** Oldest first / Newest first
- **Double-click to like:** In ListView

9. Final Checklist Before Submission

Code:

- [ ] All 12-15 classes created
- [ ] Encapsulation implemented (private fields)
- [ ] Inheritance hierarchy works (Post → TextPost/MediaPost)
- [ ] Polymorphism demonstrated (overriding + overloading)
- [ ] Abstraction used (abstract class + interfaces)
- [ ] Composition relationship exists
- [ ] Aggregation relationship exists
- [ ] 3 exception classes created and used
- [ ] Try-catch-finally blocks implemented
- [ ] File I/O works (save and load all data)
- [ ] Data persists between runs
- [ ] GUI has all 8 screens
- [ ] GUI shows loaded data
- [ ] All buttons work
- [ ] Navigation between screens works

Report:

- [ ] All OOP concepts explained
- [ ] Code snippets included
- [ ] UML diagram complete
- [ ] Composition vs Aggregation explained with UML
- [ ] Exception handling explained
- [ ] File handling explained
- [ ] 8 GUI screenshots included
- [ ] 4-6 pages total
- [ ] Proofread for errors

Demo Preparation:

- [ ] App runs without errors
- [ ] Can demonstrate: register → login → create post → like → comment → search → logout
- [ ] Can explain any piece of code
- [ ] Can explain OOP concepts used

This is a **moderate, feature-rich project** that will impress your instructor while still being achievable. It has all the requirements plus practical features like a real social media app. Good luck! 🚀