🚀 Social Media Feed Engine

A fully-functional social media platform simulation built with C++ OOP principles. Designed for a **4-person team** completing a **3-day sprint**.

📋 Table of Contents

✨ Features

User Management

- ✅ Sign up / Login system
- ✅ Profile management (name, bio)
- ✅ Follow/Unfollow users
- ✅ View follower/following counts

Post Management

- ✅ Create posts with timestamps
- ✅ Edit post content
- ✅ Like posts
- ✅ View user-specific posts

Feed Engine

- ✅ Personalized feed from followed users
- ✅ Sort by timestamp (newest first)
- ✅ Template-based feed implementation
- ✅ Observer pattern notifications

Persistence

- ✅ File I/O for users and posts
- ✅ Serialization/Deserialization
- ✅ Data integrity checks
- ✅ Logging system

📁 Project Structure

```
SocialMediaFeed/
├── include/
│   ├── User.h            # User class declaration
│   ├── Post.h            # Post class declaration
│   ├── Feed.h            # Feed interface & templates
│   ├── Observer.h        # Observer pattern
│   ├── SystemCore.h      # Singleton core system
│   └── Utils.h           # Utility functions
├── src/
│   ├── User.cpp          # User implementation
│   ├── Post.cpp          # Post implementation
│   ├── SystemCore.cpp    # Core system implementation
│   ├── Utils.cpp         # Utility implementation
│   └── main.cpp          # Main menu & entry point
├── data/
│   ├── users.txt         # User persistence
│   ├── posts.txt         # Post persistence
│   └── logs.txt          # System logs
├── README.md
├── CMakeLists.txt        # CMake build file (optional)
└── Makefile              # Simple makefile
```

🎓 OOP Concepts Covered

Concept	Implementation
Classes & Objects	User, Post, SystemCore
Encapsulation	Private members with public getters/setters
Inheritance	TextFeed<T> inherits from IFeed
Polymorphism	Virtual functions in IFeed interface
Abstract Classes	IFeed, IObserver, ISubject
Templates	TextFeed<T> template class
Exception Handling	Try-catch blocks in deserialization
File I/O	Serialization to users.txt, posts.txt
Design Patterns	Singleton (SystemCore), Observer (notifications)
STL Containers	std::vector, std::unordered_map

👥 Team Division (3-Day Sprint)

Member 1: User Management

Files: `User.h`, `User.cpp`

Day 1:

- Design `User` class with encapsulation
- Implement constructors, getters, setters
- Create basic follow/unfollow logic

Day 2:

- Implement serialization/deserialization
- Add profile update functions
- Test user CRUD operations

Day 3:

- Integration testing with SystemCore
- Bug fixes and documentation

Member 2: Post Management

Files: `Post.h`, `Post.cpp`

Day 1:

- Design `Post` class with timestamp
- Implement CRUD operations
- Add like functionality

Day 2:

- Implement serialization/deserialization
- Link posts to userIDs
- Add exception handling

Day 3:

- Integration with Feed module
- Testing and refinement

Member 3: Feed Engine

Files: `Feed.h`, `Observer.h`

Day 1:

- Design abstract `IFeed` interface
- Implement `TextFeed<T>` template
- Add sorting algorithms

Day 2:

- Implement Observer pattern
- Create notification system
- Hook feed to post events

Day 3:

- Full integration testing
- Performance optimization

Member 4: System Core

Files: `SystemCore.h`, `SystemCore.cpp`, `Utils.h`, `Utils.cpp`, `main.cpp`

Day 1:

- Implement Singleton pattern
- Create file I/O functions
- Design utility helpers

Day 2:

- Implement data loading/saving
- Create user-post mappings
- Add logging system

Day 3:

- Build complete main menu
- Full system integration
- Final testing & deployment

🔨 Build & Run

Method 1: Using g++ (Recommended for Quick Start)


bash

```bash
# Navigate to project directory
cd SocialMediaFeed

# Create data directory
mkdir -p data

# Compile
g++ -std=c++17 -Iinclude src/*.cpp -o social_feed_engine

# Run
./social_feed_engine
```

Method 2: Using Makefile

bash

```
make
./social_feed_engine
```

Method 3: Using CMake

bash

```
mkdir build
cd build
cmake ..
make
./social_feed_engine
```

📖 Usage Guide

1. First Time Setup

1. Sign Up with username
2. Create your profile (name, bio)
3. Start creating posts!

2. Following Users

1. Login
2. Select "Follow User"
3. Choose from available users
4. View their posts in your feed

3. Viewing Feed

1. Login
2. Select "View Feed"
3. See posts from all followed users
4. Posts sorted by newest first

4. Creating Posts

1. Login
2. Select "Create Post"
3. Write your content
4. Post appears in followers' feeds

💾 Data Format

users.txt Format

```
userID|username|name|bio|followers_csv|following_csv
```

Example:

```
u_1000|alice|Alice%20Kumar|Loves%20coding|u_1001,u_1002|u_1003
```

posts.txt Format

```
postID|userID|timestamp|likes|content_encoded
```

Example:

```
p_1000|u_1000|1700000000|5|Hello%20world%21
```

🧪 Testing

Unit Tests

Test User Class:

```cpp
```

```
// Create user
User u1("u_001", "alice");

// Test follow
u1.follow("u_002");
assert(u1.isFollowing("u_002"));

// Test serialization
string serialized = u1.serialize();
User u2 = User::deserialize(serialized);
assert(u2.getUserID() == "u_001");
```

Test Post Class:

cpp

```
// Create post
Post p1("p_001", "u_001", "Hello", currentTimestamp());

// Test like
p1.like();
assert(p1.getLikes() == 1);
```

Integration Tests

Test Feed Generation:

1. Create 3 users: A, B, C
2. A follows B and C
3. B creates post P1
4. C creates post P2
5. Generate feed for A
6. Assert feed contains P1 and P2
7. Assert posts sorted by timestamp

Test Persistence:

1. Create users and posts
2. Save data (call saveAllData())
3. Clear SystemCore
4. Load data (call loadAllData())
5. Verify all users and posts restored

🎯 Sprint Checkpoints

End of Day 1

- [ ] All header files created
- [ ] Basic class implementations done
- [ ] Agreed on serialization format
- [ ] Utils module complete

End of Day 2

- [ ] All modules implement I/O
- [ ] SystemCore loads/saves data
- [ ] Observer pattern functional
- [ ] Basic testing complete

End of Day 3

- [ ] Full system integration
- [ ] Main menu working
- [ ] All features tested
- [ ] Documentation complete

🚨 Common Issues & Solutions

Issue: Compilation errors with templates

Solution: Templates must be defined in header files (`.h`), not `.cpp`

Issue: File not found errors

Solution: Ensure `data/` directory exists before running

Issue: Segmentation fault

Solution: Check for null pointers before accessing SystemCore data

Issue: Data not persisting

Solution: Call `saveAllData()` before program exit

📚 Learning Resources

- **Encapsulation:** See `User` class private members
- **Inheritance:** Check `TextFeed<T> : public IFeed`
- **Polymorphism:** Review `IFeed` virtual functions
- **Singleton:** Study `SystemCore::getInstance()`
- **Observer:** Examine `IObserver` and `ISubject`
- **Templates:** Look at `TextFeed<T>` implementation

🏆 Project Highlights

✅ **Complete OOP coverage** - All major concepts included
✅ **Real-world application** - Actual social media functionality
✅ **Team-friendly** - Clear division of responsibilities
✅ **Production-ready** - Error handling, logging, persistence
✅ **Extensible** - Easy to add features (comments, search, etc.)

📝 Future Enhancements

- 🔍 **Search System** - Find users/posts by keyword
- 💬 **Comments** - Add nested comments to posts
- 📸 **Media Support** - Image/video posts
- 🔔 **Real Notifications** - Display notification list
- 📊 **Analytics** - User/post statistics dashboard
- 🌐 **Network API** - RESTful API for web interface

👨‍💻 Team Credits

Member 1 - User Management System
Member 2 - Post Management System
Member 3 - Feed Engine & Observer Pattern
Member 4 - System Core & Integration

📄 License

Built with ❤️ using Modern C++17