## InCollege Project: Week 8 Deliverable - Basic Messaging System - Part 1 (Send Message)

**Objective:** This week, your team will implement the foundational capability for connected users to send private messages to each other within the InCollege application. This involves allowing users to initiate a message and ensuring that message content is saved persistently, with all input and output handled via files as per our testing protocol.

**Focus Areas:**

1. **Messaging Option:**
    - A new option will be added to the main post-login menu (e.g., "Messages").
    - When a user selects "Messages," they should be presented with at least two options: "Send a New Message" and "View My Messages" (the latter will remain "under construction" for this week).
2. **Recipient Selection:**
    - When a user chooses "Send a New Message," they should first be prompted to enter the username of the intended recipient.
    - **Constraint:** A user should only be able to send a message to someone they are **already connected with** (based on your Week 5 implementation). Your system must validate this. If the entered username is not a recognized connection, an appropriate message should be displayed (e.g., "You can only message users you are connected with," or "User not found in your network").
3. **Message Content Input:**
    - Once a valid, connected recipient is identified, the system should prompt the sender to enter the content of their message.
    - The message content should be free-form text.
4. **Message Persistence:**
    - All sent messages must be saved persistently. This means they should be stored in a way that allows them to be retrieved by the recipient even after the application is closed and reopened.
    - The stored message needs to include:
        - Sender's Username
        - Recipient's Username
        - Message Content
        - (Optional but recommended for future expansion) A timestamp for when the message was sent.
5. **I/O Requirements:**
    - **Input:** All user input (e.g., menu selections, recipient username, message content) will be read from a predefined input file.
    - **Output Display:** All program output (e.g., prompts, confirmation messages, error messages) must be displayed on the screen (standard output).
    - **Output Preservation:** The exact same output displayed on the screen must also be written to a separate output file for testing and record-keeping purposes.

**COBOL Implementation Details (For Programmers):**

- **Messaging Data Structure:** Define a new COBOL data structure (e.g., a new sequential file) specifically for storing messages. This structure will need fields for sender, recipient, and the message content.
- **File I/O for Messages:** Implement COBOL file operations to write new messages to your persistent storage.
- **Connection Validation:** Leverage your existing established connections data (from Week 5) to validate if the intended recipient is indeed a connection of the sender before allowing the message to be sent.
- **Input File Handling:** Continue to implement COBOL READ statements to read user input from the designated input file for all messaging-related prompts.
- **Output File Handling:** Ensure all program output, including prompts for recipient/message input and confirmation/error messages, is displayed on the screen and *identically* written to your dedicated output file.
- **Menu Integration:** Integrate the new "Messages" option into the main post-login menu, and then the "Send a New Message" option within the "Messages" sub-menu. The "View My Messages" option should still display an "under construction" message.

**Testing Responsibilities (For Testers):**

- **Test Case Development:** Create comprehensive test cases in Jira for all Week 8 functionalities, including:
    - **Positive Test Cases:** Scenarios for successfully sending a message to an existing, connected user.
    - **Negative Test Cases:** Scenarios for attempting to send a message to:
        - A non-existent user.
        - A user who is not a connection.
        - A user who is a pending connection (not yet accepted).
        - Messages with very long content (if applicable).
    - **Persistence:** Verify that sent messages are saved persistently and can be theoretically retrieved (though retrieval is Week 9's focus).
- **Test Execution:** Execute all developed test cases using the specified input file.
- **Bug Reporting:** For every issue or discrepancy found, create a detailed bug ticket in Jira. Include steps to reproduce, actual results, and expected results.
- **Output Verification:** Meticulously compare the program's console output against the generated output file to ensure they are absolutely identical for all message sending scenarios.
- **Collaboration:** Work closely with the programmers to help them understand and reproduce bugs.

**Jira Requirements (For Scrum Master, Programmers, & Testers):**

Your team's Jira board for Week 8 should include:

- **Epic #5: Messaging System:** Define this new epic for all messaging-related features.
- **User Stories for Sending Messages:**
  - "As a logged-in user, I want to send a private message to a user I am connected with."
  - "As a user sending a message, I want to select the recipient by their username."
  - "As a user sending a message, I want to enter free-form text as the message content."
  - "As a user, I want to be informed if I try to send a message to someone I am not connected with."
  - "As a user, I want my sent messages to be saved persistently for the recipient to view later."
  - "As a user, I want the main menu to have a 'Messages' option, which includes 'Send a New Message'."
- **New User Story (for Testing):** "As a tester, I want the program to read all user inputs for sending messages from a file so I can automate testing."
- **New User Story (for Testing):** "As a tester, I want the program to write all screen output related to sending messages to a file so I can easily verify results."
- **Tasks:** Break down each User Story into granular tasks that individual team members can work on. (e.g., *Programmers:* "Define COBOL file structure for messages," "Implement COBOL module for recipient validation against connections," "Develop COBOL routine to capture message content," "Create COBOL routine to write message data to file," "Update I/O for message sending features to use file input/output consistently." *Testers:* "Develop test cases for message sending," "Execute message sending tests," "Log bugs related to message sending," "Verify I/O consistency for message sending features").
- **Bug Tickets:** Log any issues found during development and testing.

**GitHub Requirements:**

- **New Modules/Files:** Commit any new COBOL modules or updated existing files for handling message sending.
- **Branching:** Continue to follow your team's established branching strategy for new features.
- **Regular Commits:** Ensure consistent, descriptive commits throughout the week. Testers should commit their test files.
- **README.md Update:** Update your README.md to reflect the new functionality for sending messages, explicitly detailing how to prepare input for these features and where to find the corresponding output.

**Deliverables for End of Week 8:**

1. **Roles.txt**: List of team members and the roles that they played this week.
2. **InCollege.cob: Working COBOL Program:** A console-based COBOL application that allows logged-in users to:
    a. Navigate to a "Messages" menu.
    b. Select "Send a New Message."
    c. Select a connected user as a recipient.
    d. Enter message content.
    e. Persistently save the sent message.
    f. The "View My Messages" option should display an "under construction" message.
    g. This must seamlessly integrate with all previous weeks' functionality (login, profile, connections, job board). All inputs must be read from a file, all outputs displayed on the screen, and the exact same outputs written to a separate file.
3. **InCollege-Input.txt: Sample Input File:** A sample text file demonstrating the format of input your program expects for Week 8's functionality (e.g., menu choices, recipient username, message content).
4. **InCollege-Output.txt:** A sample text file showing the expected output for a typical run of your program, demonstrating a user sending a message to a connected user.

```
--- SAMPLE_OUTPUT_WEEK8.TXT ---
Welcome to InCollege!
1. Log In
2. Create New Account
Enter your choice:
Please enter your username:
Please enter your password:
You have successfully logged in.
Welcome, SendingUser!
1. View My Profile
2. Search for User
3. Learn a New Skill
4. View My Pending Connection Requests
5. View My Network
6. Messages
Enter your choice:
--- Messages Menu ---
1. Send a New Message
2. View My Messages
3. Back to Main Menu
Enter your choice:
Enter recipient's username (must be a connection):
Enter your message (max 200 chars):
Message sent to ConnectedUser successfully!
---------------------
1. Send a New Message
2. View My Messages
```

```
3. Back to Main Menu
Enter your choice:
View My Messages is under construction.
1. Send a New Message
2. View My Messages
3. Back to Main Menu
Enter your choice:
--- END_OF_PROGRAM_EXECUTION ---
```

5. **Epic8-Storyx-Test-Input.zip: Test Input Files:** A set of test input files used by the testers, covering positive, negative, and edge cases for each of this week's stories.
6. **Epic8-Storyx-Test-Output.zip: Actual Test Output Files:** The exact output generated by running your program with the Epic8-Storyx-Input input Files, submitted for review.
7. **Jira.jpg: Updated Jira Board:** All relevant User Stories, tasks, and bugs (with their status) for Week 3 should be updated in Jira.
8. **Jira:** Two Burndown charts. The first created on Monday and the second created when the Sprint is complete.
9. **GitHub.jpg:** Go to the repository's main page. Click the "Commits" link (next to the green "Code" button). Show a chronological list of all commits with messages, authors, and timestamps.

Your testers will be vital this week in verifying that messages are correctly sent and saved, and most importantly, that messages can *only* be sent to established connections. They will also meticulously compare the console output with the generated output file to ensure perfect consistency. The scrum master will continue to facilitate daily stand-ups and remove any impediments for the team.