InCollege Project: Week 4 Deliverable - Connection Requests

Objective: This week, your team will implement the functionality for users to send and store connection requests to other users within the InCollege platform. This lays the groundwork for users to build their professional networks. All program input will continue to be read from a file, all output will be displayed on the screen, and that same output will also be written to a file.

Focus Areas:

1. **Sending Connection Requests:**
    - After successfully searching for and viewing another user's profile (from Week 3), the system should present the option to "Send Connection Request" to that user.
    - When a user sends a request, the system must record this pending connection.
    - The sending user should be informed that their request has been sent.
    - **Constraint:** A user should only be able to send a connection request to someone they are *not already connected with* and *who has not already sent them a pending request*. Your system should validate this. If an invalid request is made (e.g., trying to connect with someone already connected, or someone who has already sent them a request), an appropriate message should be displayed (e.g., "You are already connected with this user," or "This user has already sent you a connection request").
2. **Storing Pending Requests:**
    - All pending connection requests must be saved persistently. This means they should be stored in a way that allows them to be retrieved even after the application is closed and reopened.
    - You'll need to define a data structure to store who sent the request and to whom it was sent.
3. **Viewing Pending Requests:**
    - A new option should be available from the main post-login menu (e.g., "View My Pending Connection Requests").
    - When a user selects this option, the system should display a list of all users who have sent them a connection request that is still pending.
4. **I/O Requirements:**
    - **Input:** All user input (e.g., menu selections, confirmation of sending requests) will be read from a predefined input file.
    - **Output Display:** All program output (e.g., prompts, confirmation messages, displayed lists of pending requests, 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):

- **Connection Data Structure:** Define a COBOL data structure (e.g., a new sequential file or an addition to an existing one) to store pending connection requests. This structure will likely need fields for the sender's username and the recipient's username.
- **File I/O for Connections:** Implement COBOL file operations to write new connection requests to your persistent storage and to read existing pending requests.
- **Request Logic:** Develop COBOL modules to handle the "Send Connection Request" action. This will involve:
    - Validating the request (checking if already connected or if a request is already pending from the recipient).
    - Writing the new request to file.
- **Display Logic:** Implement COBOL routines to retrieve and display the list of pending connection requests for a logged-in user.
- **User Interaction & I/O:** Design clear prompts for the user, ensuring all input is read from the specified input file and all screen output (including prompts and messages) is *also* written to the designated output file.
- **Menu Integration:** Integrate the "Send Connection Request" option when viewing another user's profile and the "View My Pending Connection Requests" option into the main menu.

Testing Responsibilities (For Testers):

- **Test Case Development:** Create comprehensive test cases in Jira for all Week 4 functionalities, including:
    - **Positive Test Cases:** Scenarios for successfully sending a connection request to a new, unconnected user.
    - **Negative Test Cases:** Scenarios for attempting to send a request to a non-existent user, a user already connected, or a user who has already sent a pending request to the current user.
    - **Viewing Pending Requests:** Test scenarios where a user has no pending requests, one pending request, and multiple pending requests.
- **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 connection request 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 4 should update **Epic #3: User Search & Connections (Part 1)** or create a **new Epic: Connection Management**. Let's continue with Epic #3 for now, focusing on the initiation of connections.

- **User Stories for Sending Requests:**
  - "As a logged-in user, I want to send a connection request to another user whose profile I am viewing."
  - "As a user sending a request, I want to be informed if my request was sent successfully."
  - "As a user, I should not be able to send a connection request to someone I am already connected with."
  - "As a user, I should not be able to send a connection request to someone who has already sent me a request."
- **User Stories for Storing & Viewing Requests:**
  - "As a user, I want my sent connection requests to be saved persistently."
  - "As a user, I want to view a list of all pending connection requests I have received."
  - "As a user, I want an option in the main menu to see my pending connection requests."
- **New User Story (for Testing):** "As a tester, I want the program to read all user inputs for connection requests 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 connection requests 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 pending connections," "Implement COBOL validation for existing connections," "Develop COBOL module to write pending requests," "Create COBOL display routine for pending requests list," "Update I/O for connection request features to use file input/output." *Testers:* "Develop test cases for sending requests," "Execute pending request viewing tests," "Log bugs related to connection requests," "Verify I/O consistency for connection request 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 connection requests.
- **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 and viewing connection requests, explicitly detailing how to prepare input for these features and where to find the corresponding output.

Deliverables for End of Week 4:

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 users to:
    a. Send connection requests to other users.
    b. Persistently store these requests.
    c. View a list of pending connection requests they have received.
    d. This must seamlessly integrate with the login, profile, and search functionality from previous weeks. 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 4's functionality (e.g., menu choices, usernames for sending requests).
4. **InCollege-Output.txt:** A sample text file showing the expected output for a typical run of your program, demonstrating sending a request and viewing pending requests.

```
--- SAMPLE_OUTPUT_WEEK4.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, TestUser!
1. View My Profile
2. Search for User
3. Learn a New Skill
4. View My Pending Connection Requests
Enter your choice:
--- Pending Connection Requests ---
You have no pending connection requests at this time.
-----------------------------------
1. View My Profile
2. Search for User
3. Learn a New Skill
4. View My Pending Connection Requests
Enter your choice:
Enter the full name of the person you are looking for:
--- Found User Profile ---
Name: New Student
University: West Coast Uni
Major: Business
Graduation Year: 2027
```

```
-------------------------
1. Send Connection Request
2. Back to Main Menu
Enter your choice:
Connection request sent to New Student.
1. View My Profile
2. Search for User
3. Learn a New Skill
4. View My Pending Connection Requests
Enter your choice:
--- END_OF_PROGRAM_EXECUTION ---
```

5. **Epic4-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. **Epic4-Storyx-Test-Output.zip: Actual Test Output Files:** The exact output generated by running your program with the Epic4-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. **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 have a busy week ensuring that requests are sent correctly, stored persistently, and displayed accurately. They'll also be crucial in testing the validation rules to prevent duplicate or invalid requests, all while verifying the consistency of console and file output. The scrum master will continue to facilitate daily stand-ups and remove any impediments for the team.