Testing Strategy for Instant Messaging System

Automated Testing Strategy

1.Unit Testing

- Objective: Verify the functionality of individual components (e.g., `User`, `Message`, `Conversation`, `ConversationManager`) in isolation.
- Tools: JUnit, Mockito (for mocking dependencies).
- Test Cases:
    - User Class:
        - Test `getUsername()`, `isOnline()`, `sendMessage()`.
        - Mock WebSocket session to test message sending.
    - Message Class:
        - Test `getContent()`, `getSender()`, `getTimestamp()`.
    - Conversation Class:
        - Test adding/removing participants.
        - Verify message addition and retrieval.
    - Conversation-Manager Class:
        - Test creating and retrieving conversations.
        - Ensure conversation termination correctly removes it from active list.
- Expected Outcome: All individual methods perform as expected, with edge cases (e.g., adding the same participant twice) handled gracefully.

2. Integration Testing

- Objective: Test how different components of the system interact with each other.
- Tools: JUnit, Spring Test.
- Test Cases:
    - User-Message Interaction:
        - Send a message from one user to another; verify it is received.
        - Test message persistence and retrieval.
    - Conversation-Message Interaction:
        - Add multiple messages to a conversation; ensure all are correctly retrieved.
    - Conversation-Manager-Conversation Interaction:
        - Create a conversation, add participants, send messages, and terminate the conversation.
- Expected Outcome: Components work together as expected, with messages correctly routed and stored.

3. System Testing

- Objective: Validate the system's overall functionality, including user authentication, message handling, and group management.
- Tools: Selenium (for UI testing), Postman (for API testing), JMeter (for load testing).
- Test Cases:
    - User Authentication:
        - Test registration, login, and logout workflows.
        - Ensure unauthorized actions are blocked.
    - Messaging:
        - Test sending and receiving private and group messages.
        - Validate offline message delivery and historical message retrieval.
    - Group Management:
        - Test creating, joining, and leaving groups.
        - Ensure group messages are correctly delivered to all members.
    - Simultaneous Chats:
        - Participate in multiple chats and verify independent message flow.
- Expected Outcome: The system should handle end-to-end operations smoothly, with proper message delivery, user management, and error handling.

4. Load and Performance Testing

- Objective: Ensure the system can handle multiple users and messages simultaneously without performance degradation.
- Tools: Apache JMeter, Gatling.
- Test Cases:
    - Simulate multiple users sending/receiving messages concurrently.
    - Measure response times for message delivery and historical message retrieval.
- Expected Outcome: The server should maintain acceptable performance levels, with no crashes or significant slowdowns.

Manual Testing Strategy

1.User Acceptance Testing (UAT)

- Objective: Validate the system against user requirements to ensure it meets their needs.
- Test Cases:
    - Conduct end-to-end testing with real users.
    - Gather feedback on usability, performance, and any issues.
- Expected Outcome: The system should satisfy user requirements, with any necessary adjustments made based on feedback.

Conclusion

This testing strategy covers the essential aspects of the IM system, ensuring that it is functional, performant, and user friendly. Regularly executing and updating these tests will help maintain the quality and reliability of your application throughout its development lifecycle.