**DatabaseProxy Testing:**

DatabaseProxy class, which handles user and group chat data management in our server application. We use JUnit4 as a test framework to ensure the correctness and reliability of the implementation method.

We have prepared test methods for various functions of DatabaseProxy, such as putUser, putGroupChat, putMessage, getGroupChats, getCoordinator, getMessage, deleteUserFromGroupChat. These tests are designed to ensure that DatabaseProxy methods behave as expected under various conditions.

Testing has shown that the implemented methods can successfully add and retrieve user data, create and manage group chats, send and receive messages, and update and delete user information as needed. For example, the putUser method is tested to verify its ability to add a new user and return its information. The test case shows that the method correctly identifies new and existing users and returns the appropriate information.

Likewise, the ability of the putGroupChat method to create a new group chat and assign a moderator is tested. The test results confirmed the successful creation of group chats and moderator assignments. Other methods, such as getMessage, were also tested to ensure that users could send and receive messages in the group chat, and update the message status accordingly.

Test results show that the DatabaseProxy class works as expected, providing robust and reliable data management for our server application.

**API Testing:**

In this API testing, we focus on verifying the functionality and behavior of the server's API to handle various operations related to group chat and user management. The main purpose of these tests is to ensure that the API behaves as expected in different scenarios, covering both positive and negative cases. We selectively selected test cases covering a wide range of functionality to ensure a thorough evaluation of the API.

The test suite includes tests for creating and modifying group chats, obtaining user and group information, and managing user interactions in group chats. We also included tests to verify error handling and appropriate responses to invalid input.

For example, we test the "drop" feature to ensure that only authorized moderators can add users to group chats. Also, we check the "getMessage" method to verify that the message was sent correctly in the group chat. We also tested the "getID", "getUsers", "getConnectedUsers" and "getCoordinator" methods to confirm that they returned the expected data.

Additionally, we tested the "getUserInGroups" and "getGroupMembers" methods to ensure they returned accurate information about usergroups and group chat members, respectively. We also verified the "getName" and "getMessages" methods to confirm that they retrieved the correct username and messages from the database.

Finally, we test the "kickUser" method to verify that only authorized coordinators can remove users from the group chat, and that the removal process executes successfully.

Overall, these tests provide a solid basis for ensuring the reliability and robustness of the server API. By covering various scenarios, we aim to identify and resolve any potential issues or inconsistencies in API behavior, ultimately leading to a more stable and efficient group chat management system.

**Coordinator Testing:**

To test the Coordinator class, we chose JUnit 4, a widely used and reliable Java application testing framework. JUnit provides various annotations, assertions, and other utilities to simplify the process of writing and running test cases. By using JUnit 4, we can effectively isolate the functionality of the Coordinator class and ensure that its methods work as expected.

The "testLookForMembers" test case focuses on verifying the functionality of the "lookForMembers" method in the Coordinator class. This method accepts an array of group chat IDs with inactive coordinators and returns a map containing the group chat IDs as keys and the connected user IDs as values.

In this test case, we first define a sample input, an array of group chat IDs, and pass it to the "lookForMembers" method. We then create a map of the expected connected users for each group based on the test data. Finally, we compare the result of the method with the expected mapping using the assertEquals method provided by JUnit.

The "testSelectRandomCoordinator" test case focuses on the functionality of the "selectRandomCoordinator" method in the Coordinator class. This method accepts a map containing group chat IDs as keys and a list of connected user IDs as values, then randomly selects a user from the list and designates them as the new moderator of the group chat.

In this test case, we first create a sample input map that contains the group chat ID and the connected user ID. Next, we call the "selectRandomCoordinator" method with the input map. After executing this method, we retrieve the updated coordinator for each group chat from the database and use the assertTrue method provided by JUnit to ensure that the new coordinator is a valid connected user.

By implementing and running the "testLookForMembers" and "testSelectRandomCoordinator" test cases using the JUnit 4 testing framework, we can ensure that the Coordinator class behaves as expected when managing and coordinating group chats in the chat server application. Proper testing of this class helps to identify potential issues early in the development process, reducing the risk of bugs and improving the overall quality of the application.