Test Cases for Collaborative Pomodoro Timer

1. Complex Unit Test: `createInvitation_validInputs_success` from InvitationServiceTest

Use Case Description:
This test verifies the complex invitation creation functionality in the InvitationService. It tests the ability to create a group invitation, which is a core feature of the collaborative Pomodoro timer application. The test ensures that when a user invites another user to a group, the invitation is properly created with the correct status, inviter, and group information.

Test Implementation:

```java
//src/test/java/ch/uzh/ifi/hase/soprafs24/service/InvitationServiceTest.java
@Test
void createInvitation_validInputs_success() {
    // then
    GroupMembership createdMembership = invitationService.createInvitation(testGroup.getId(), testToken,
testInvitee.getId());

    // verify
    Mockito.verify(membershipService).addUserToGroup(Mockito.any(), Mockito.any(),
Mockito.eq(MembershipStatus.PENDING), Mockito.eq(testInviter.getId()));

    assertEquals(MembershipStatus.PENDING, createdMembership.getStatus());
    assertEquals(testInviter.getId(), createdMembership.getInvitedBy());
    assertEquals(testInvitee.getId(), createdMembership.getUser().getId());
    assertEquals(testGroup.getId(), createdMembership.getGroup().getId());
}
```

Code Being Tested:

```java
// src/main/java/ch/uzh/ifi/hase/soprafs24/service/InvitationService.java
public GroupMembership createInvitation(Long groupId, String token, Long inviteeId) {
    Group group = groupService.getGroupById(groupId, token);
    User inviter = userService.findByToken(token);

    // The group should contain the inviter
    if (!group.getActiveUsers().contains(inviter)) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, String.format(FORBIDDEN,
inviter.getId()));
    }

    // The invitee should not be a member of the group
    User invitee = userRepository.findById(inviteeId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND,
String.format(NOT_FOUND, "User", inviteeId)));
    if (group.getActiveUsers().contains(invitee)) {
        throw new ResponseStatusException(HttpStatus.CONFLICT, String.format(CONFLICT, inviteeId));
    }

    // Check if an invitation already exists
    GroupMembership existingMembership = membershipService.findByUserAndGroup(invitee, group);
    if (existingMembership != null) {
        if (existingMembership.getStatus() == MembershipStatus.PENDING) {
            throw new ResponseStatusException(HttpStatus.CONFLICT, "An invitation for this user and group
already exists and is pending");
        }
    }

    // Create the invitation using the membership service
    return membershipService.addUserToGroup(invitee, group, MembershipStatus.PENDING, inviter.getId());
}
```

Analysis:
This test is an excellent example of a complex unit test because:

1. It properly isolates the service layer by mocking all dependencies (repositories and other services)
2. It verifies multiple aspects of the invitation creation process (status, inviter, invitee, group)
3. It uses Mockito to verify that the correct service methods are called with the right parameters
4. It tests complex business logic around group membership and invitation states
5. It would catch regressions in:
    - Invitation status setting
    - Inviter/invitee relationship
    - Group membership creation
    - Service method interactions

2. Integration Test: `getGroupsForUser_shouldReturnActiveGroups_whenUserExists` from UserServiceIntegrationTest

Use Case Description:
This integration test verifies the interaction between the UserService and GroupService when retrieving a user's active groups. It tests the complete flow from user creation to group membership management, ensuring that users can see their active groups in the collaborative Pomodoro timer application. This is crucial for the collaborative aspect of the application, as users need to be able to see and join their groups.

Test Implementation:

```java
//src/test/java/ch/uzh/ifi/hase/soprafs24/service/UserServiceIntegrationTest.java
@Test
void getGroupsForUser_shouldReturnActiveGroups_whenUserExists() {
    // Create test user
    User testUser = new User();
    testUser.setUsername("testUsername");
    testUser.setPassword("testPassword");
    testUser.setStatus(UserStatus.ONLINE);
    userService.createUser(testUser);

    // Create and add groups
    Group group1 = new Group();
    group1.setName("Group 1");
    group1.setAdminId(testUser.getId());
    groupRepository.save(group1);

    Group group2 = new Group();
    group2.setName("Group 2");
    group2.setAdminId(testUser.getId());
    groupRepository.save(group2);

    // Add memberships
    GroupMembership membership1 = new GroupMembership();
    membership1.setGroup(group1);
    membership1.setUser(testUser);
    membership1.setStatus(MembershipStatus.ACTIVE);

    GroupMembership membership2 = new GroupMembership();
    membership2.setGroup(group2);
    membership2.setUser(testUser);
    membership2.setStatus(MembershipStatus.ACTIVE);

    testUser.getMemberships().add(membership1);
    testUser.getMemberships().add(membership2);
    userRepository.save(testUser);

    // Get active groups
    var activeGroups = userService.getGroupsForUser(testUser.getId(), testUser.getToken());
    // Verify results
    assertNotNull(activeGroups);
    assertEquals(2, activeGroups.size());
    assertTrue(activeGroups.contains(group1));
    assertTrue(activeGroups.contains(group2));
}
```

Code Being Tested:

```
// src/main/java/ch/uzh/ifi/hase/soprafs24/service/UserService.java
public List<Group> getGroupsForUser(Long userId, String token) {
    User temp_user = findByToken(token);
    if (!temp_user.getId().equals(userId)) { //id we got via token does not match id from the url
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, FORBIDDEN);
    }

    User user = userRepository.findById(userId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND,
String.format(NOT_FOUND, "User", userId)));

    return membershipService.getActiveGroupsForUser(user);
}

// src/main/java/ch/uzh/ifi/hase/soprafs24/service/MembershipServiceImpl.java
@Override
public List<Group> getActiveGroupsForUser(User user) {
    return membershipRepository.findByUserAndStatus(user, MembershipStatus.ACTIVE)
            .stream()
            .map(GroupMembership::getGroup)
            .toList();
}
```

Analysis:
This test is an excellent example of an integration test because:

1. It tests the interaction between multiple services (UserService and MembershipService)
2. It uses real repositories and database operations
3. It verifies the complete flow from user creation to group membership retrieval
4. It would catch regressions in:
   - Database operations
   - Service interactions
   - Entity relationships
   - Transaction management
   - Data consistency
5. It tests both the happy path and ensures proper data relationships are maintained

3. REST Interface Test: `getUserGroups_validUserId_groupsReturned` from UserControllerTest

Use Case Description:
This REST interface test verifies the HTTP endpoint for retrieving a user's groups. It tests the complete HTTP request/response cycle, including proper JSON serialization and deserialization, status codes, and response body structure. This test ensures that the frontend can properly communicate with the backend to display a user's groups in the Pomodoro timer application.

Test Implementation:

```
//src/test/java/ch/uzh/ifi/hase/soprafs24/controller/UserControllerTest.java
@Test
void getUserGroups_validUserId_groupsReturned() throws Exception {
    // Setup test data
    Long userId = 1L;
    User user = new User();
    user.setId(userId);
    user.setUsername("testUsername");
    user.setStatus(UserStatus.ONLINE);

    Group group1 = new Group();
    group1.setName("Group 1");

    Group group2 = new Group();
    group2.setName("Group 2");

    List<Group> activeGroups = Arrays.asList(group1, group2);

    String validToken = "valid-token";
    given(userService.getGroupsForUser(userId, validToken)).willReturn(activeGroups);

    // Perform HTTP request
    MockHttpServletRequestBuilder getRequest = get("/users/{userId}/groups", userId)
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", validToken);

    // Verify response
    mockMvc.perform(getRequest)
            .andExpect(status().isOk())
            .andExpect(jsonPath("$", hasSize(2)))
            .andExpect(jsonPath("$[0].name", is(group1.getName())))
            .andExpect(jsonPath("$[1].name", is(group2.getName())));
}
```

Code Being Tested:

```
// src/main/java/ch/uzh/ifi/hase/soprafs24/controller/UserController.java
@GetMapping("/users/{id}/groups")
@ResponseStatus(HttpStatus.OK)
public List<GroupGetDTO> getGroupsForUser(@PathVariable("id") Long id, @RequestHeader("Authorization")
String token) {
    List<Group> groups = userService.getGroupsForUser(id, token);
    return groups.stream()
            .map(DTOMapper.INSTANCE::convertEntityToGroupGetDTO)
            .toList();
}
```

Analysis:
This test is an excellent example of a REST interface test because:

1. It uses Spring's MockMvc to simulate HTTP requests
2. It verifies both the request handling and response formatting
3. It tests the complete API contract including:
   - HTTP method
   - URL structure
   - Request headers
   - Response status
   - Response body structure
4. It would catch regressions in:
   - API endpoint changes
   - Response format changes
   - Authorization header handling
   - JSON serialization/deserialization
5. It properly mocks the service layer while testing the controller layer in isolation