System Testing Report - Messenger API

Course: Software Testing and Quality Assurance
Assignment: System Testing
Date: January 12, 2026

Team Members:

Name	Student ID
A. M. Shahriar Rashid Mahe	011221130
Saida Zaman	011221141
Eila Afrin	011221149
Jarin Fariha	011213124

Table of Contents

GitHub repo link : **stqa-test-messenger-a1**

1. Introduction

This report presents the results of comprehensive system testing performed on the Messenger API. The API provides user authentication, user management, conversation handling (DIRECT and GROUP), and messaging functionality with JWT-based authentication.

Testing Scope:

- Authentication endpoints (signup, login, logout)
- User management (CRUD operations, block/unblock)
- Conversation management (create, retrieve, member operations)
- Message operations (send, retrieve)
- Authorization and permission validation
- Business logic and data integrity

Testing Tools: Postman

Test Coverage: 72 test cases across 4 API categories

---

2. Test Plan

2.1 Test Strategy

Approach: Black-box system testing with focus on:

- Boundary value analysis
- Equivalence partitioning
- Negative testing
- Authorization and permission checks

2.2 Authentication APIs

POST /auth/signup

Parameters: username (string), email (string), password (string)

Input Type	Test Scenario	Expected Status	Expected Output
Valid	Valid credentials	201 Created	User object with userId
Invalid	Duplicate username	400 Bad Request	"Username already taken"
Invalid	Empty username	400 Bad Request	Validation error
Invalid	Invalid email format	400 Bad Request	Validation error
Invalid	Empty password	400 Bad Request	Validation error
Security	SQL injection attempt	400 Bad Request	Input rejected
Invalid	Duplicate email	400 Bad Request	"Email already in use"

POST /auth/login

Parameters: username (string), password (string)

Input Type	Test Scenario	Expected Status	Expected Output
Valid	Correct credentials	200 OK	JWT token and userId
Invalid	Wrong password	401 Unauthorized	"Invalid password"
Invalid	Non-existent user	401 Unauthorized	"User not found"
Invalid	Empty password	401 Unauthorized	"Password required"
Security	SQL injection	401 Unauthorized	Attack blocked

POST /auth/logout

Parameters: username (query parameter)

Input Type	Test Scenario	Expected Status	Expected Output
Valid	Logged-in user	200 OK	Success message
Invalid	Non-existent user	404 Not Found	"User not found"
Invalid	Empty username	400 Bad Request	"Username required"
Invalid	Invalid format	400 Bad Request	Validation error

GET /auth/test

Parameters: Authorization header with JWT token

Input Type	Test Scenario	Expected Status
Valid	Valid token	200 OK
Invalid	Invalid token	403 Forbidden
Invalid	No token	403 Forbidden

2.3 User APIs

GET /users/

Authorization: Required (JWT token)

Test Scenario	Expected Status	Expected Output
Authorized request	200 OK	Array of user objects
No authorization	401 Unauthorized	Error message

GET /users/{userId}

Authorization: Required

Test Scenario	Expected Status
Valid user ID	200 OK
Invalid user ID	404 Not Found

PATCH /users/{userId}

Authorization: Required (self-update only)
Parameters: username, email, password (optional)

Test Scenario	Expected Status
Update own profile	200 OK
Update another user	403 Forbidden
Include userId in body	400 Bad Request

PATCH /users/block/{userId}

Authorization: Required

Test Scenario	Expected Status
Block valid user	200 OK
Block non-existent user	400 Bad Request
Block self	400 Bad Request

2.4 Conversation APIs

POST /conversations/create

Parameters: type (DIRECT/GROUP), name (for GROUP), memberIds (array)

Type	Test Scenario	Expected Status
DIRECT	Valid members	200 OK
DIRECT	Duplicate conversation	400 Bad Request
DIRECT	Non-existent user	400 Bad Request
DIRECT	Blocked user	400 Bad Request
GROUP	Valid with name	200 OK
GROUP	Empty name	400 Bad Request

POST /conversations/addMember

Parameters: conversationId, members (array)

Test Scenario	Expected Status
Creator adds to GROUP	200 OK
Non-creator adds to GROUP	400 Bad Request
Add to DIRECT	400 Bad Request
Add blocked user	400 Bad Request

POST /conversations/removeMember

Parameters: conversationId, members (array)

Test Scenario	Expected Status
Creator removes from GROUP	200 OK
Non-creator removes	400 Bad Request
Self-removal	200 OK

2.5 Message APIs

POST /messages/{conversationId}/send

Authorization: Required (must be member)
Body: Message content (text/plain)

Test Scenario	Expected Status
Valid message in DIRECT	200 OK
Valid message in GROUP	200 OK
Non-member sends	400 Bad Request
Empty message	400 Bad Request
Blocked user in DIRECT	400 Bad Request
XSS payload	200 OK (sanitized)

GET /messages/{conversationId}/get

Authorization: Required (must be member)

Test Scenario	Expected Status
Member retrieves messages	200 OK
Non-member attempts	400 Bad Request

---

3. Test Cases

3.1 Test Summary

Category	Total	Passed	Failed	Pass Rate
Auth	26	16	10	62%
Users	14	12	2	86%
Conversations	18	10	8	56%
Messages	14	12	2	86%
TOTAL	**72**	**50**	**22**	**69%**

3.2 Test Case Details

Test ID	Title	Pre-conditions	Expected Result	Actual Result	Pass/Fail
AUTH APIs					
TC-AUTH-001	Valid User Signup	None	201 Created with userId	201 Created with userId	Pass
TC-AUTH-002	Duplicate Username Signup	User "alice" exists	400 Bad Request	400 Bad Request	Pass
TC-AUTH-003	Signup with Empty Username	None	400 Bad Request	201 Created	Pass
TC-AUTH-004	Signup with Empty Email	User exists	400 Bad Request	400 Bad Request (wrong error)	**Fail**
TC-AUTH-005	Signup with Invalid Email	None	400 Bad Request	201 Created	**Fail**
TC-AUTH-006	Signup with Empty Password	None	400 Bad Request	201 Created	**Fail**
TC-AUTH-007	Valid Login	User exists	200 OK with JWT	200 OK with JWT	Pass
TC-AUTH-008	Login with Wrong Password	User exists	401 Unauthorized	401 Unauthorized	Pass
TC-AUTH-009	Login Non-existent User	None	401 Unauthorized	401 Unauthorized	Pass
TC-AUTH-010	Valid Logout	User logged in	200 OK	200 OK	Pass
TC-AUTH-011	Auth Test Valid Token	Token valid	200 OK	200 OK	Pass
TC-AUTH-012	Auth Test Invalid Token	None	403 Forbidden	403 Forbidden	Pass

Test ID	Title	Pre-conditions	Expected Result	Actual Result	Pass/Fail
TC-AUTH-013	Auth Test No Token	None	403 Forbidden	403 Forbidden	Pass
TC-AUTH-014	Invalid Request Method	None	405 Method Not Allowed	405 Method Not Allowed	Pass
TC-AUTH-015	SQL Injection Signup	None	400 Bad Request	201 Created	**Fail**
TC-AUTH-016	Duplicate Email Signup	Email exists	400 Bad Request	201 Created	**Fail**
TC-AUTH-017	Login Without Password	User exists	401 Unauthorized	200 OK with JWT	**Fail**
TC-AUTH-018	Login Email Only	2 users same email	400 Bad Request	500 Server Error	**Fail**
TC-AUTH-019	Login Missing Username	None	500 Server Error	500 Server Error	Pass
TC-AUTH-020	Login Wrong Email	User exists	400 Bad Request	200 OK	**Fail**
TC-AUTH-021	Logout Non-existent User	None	404 Not Found	200 OK	**Fail**
TC-AUTH-022	Logout Empty Username	None	400 Bad Request	200 OK	**Fail**
TC-AUTH-023	Logout Invalid Format	None	400 Bad Request	200 OK	**Fail**
TC-AUTH-024	Logout Extra Parameter	User logged in	400 Bad Request	200 OK	**Fail**
TC-AUTH-025	Login SQL Injection	None	401 Unauthorized	401 Unauthorized	Pass
TC-AUTH-026	Login Empty Body	None	400 Bad Request	400 Bad Request	Pass
USER APIs					
TC-USER-001	Get All Users (Authorized)	User logged in	200 OK with array	200 OK with array	Pass

Test ID	Title	Pre-conditions	Expected Result	Actual Result	Pass/Fail
TC-USER-002	Get All Users (Unauthorized)	None	401 Unauthorized	401 Unauthorized	Pass
TC-USER-003	Get User by Valid ID	User logged in	200 OK with object	200 OK with object	Pass
TC-USER-004	Get User by Invalid ID	User logged in	404 Not Found	404 Not Found	Pass
TC-USER-005	Get User by Username	User logged in	200 OK with object	200 OK with object	Pass
TC-USER-006	Get Non-existent Username	User logged in	404 Not Found	404 Not Found	Pass
TC-USER-007	Update Own User	User logged in	200 OK	200 OK	Pass
TC-USER-008	Update Another User	2 users exist	403 Forbidden	403 Forbidden	Pass
TC-USER-009	Update with ID in Body	User logged in	400 Bad Request	200 OK (accepts)	**Fail**
TC-USER-010	Block User	2 users exist	200 OK	200 OK	Pass
TC-USER-011	Block Non-existent User	User logged in	400 Bad Request	400 Bad Request	Pass
TC-USER-012	Block Self	User logged in	400 Bad Request	200 OK	**Fail**
TC-USER-013	Unblock User	Block exists	200 OK	200 OK	Pass
TC-USER-014	Unblock Not Blocked	No block exists	400 Bad Request	400 Bad Request	Pass
CONVERSATION APIs					
TC-CONV-001	Create DIRECT Valid	2 users exist	200 OK with ID	200 OK with ID	Pass
TC-CONV-002	Create Duplicate DIRECT	DIRECT exists	400 Bad Request	400 Bad Request	Pass

Test ID	Title	Pre-conditions	Expected Result	Actual Result	Pass/Fail
TC-CONV-003	Create GROUP Valid	Users exist	200 OK with ID	200 OK with ID	Pass
TC-CONV-004	GROUP Empty Name	Users exist	400 Bad Request	200 OK	Fail
TC-CONV-005	DIRECT Non-existent User	User logged in	400 Bad Request	200 OK	Fail
TC-CONV-006	DIRECT Blocked User	Block exists	400 Bad Request	200 OK	Fail
TC-CONV-007	Get My Conversations	User has convs	200 OK with array	200 OK with array	Pass
TC-CONV-008	Get Conv by ID (Member)	User is member	200 OK	200 OK	Pass
TC-CONV-009	Get Conv (Non-member)	User not member	400 Bad Request	400 Bad Request	Pass
TC-CONV-010	Get Non-existent Conv	User logged in	400 Bad Request	400 Bad Request	Pass
TC-CONV-011	Add Member (Creator)	Creator logged in	200 OK	200 OK	Pass
TC-CONV-012	Add Member (Non-creator)	Non-creator logged in	400 Bad Request	200 OK	Fail
TC-CONV-013	Add Blocked User	Block exists	400 Bad Request	200 OK	Fail
TC-CONV-014	Add to DIRECT	DIRECT exists	400 Bad Request	200 OK	Fail
TC-CONV-015	Remove Member (Creator)	Creator logged in	200 OK	200 OK	Pass
TC-CONV-016	Remove (Non-creator)	Non-creator logged in	400 Bad Request	500 Server Error	Fail
TC-CONV-017	Remove Self	User is member	200 OK	200 OK	Pass

Test ID	Title	Pre-conditions	Expected Result	Actual Result	Pass/Fail
TC-CONV-018	Add 3rd to DIRECT	DIRECT has 2 members	400 Bad Request	200 OK	Fail
MESSAGE APIs					
TC-MSG-001	Send in DIRECT	User is member	200 OK	200 OK	Pass
TC-MSG-002	Send in GROUP	User is member	200 OK	200 OK	Pass
TC-MSG-003	Send (Non-member)	User not member	400 Bad Request	400 Bad Request	Pass
TC-MSG-004	Send Empty Message	User is member	400 Bad Request	200 OK	Fail
TC-MSG-005	Send Long Message	User is member	200 OK	200 OK	Pass
TC-MSG-006	Send (Receiver Blocked)	Block exists	400 Bad Request	405 Method Not Allowed	Fail
TC-MSG-007	Send (Sender Blocked)	Block exists	400 Bad Request	400 Bad Request	Pass
TC-MSG-008	Send in GROUP (Blocked)	Block exists	200 OK	200 OK	Pass
TC-MSG-009	Send Non-existent Conv	User logged in	400 Bad Request	400 Bad Request	Pass
TC-MSG-010	Get Messages (Member)	User is member	200 OK	200 OK	Pass
TC-MSG-011	Get Messages (Non-member)	User not member	400 Bad Request	400 Bad Request	Pass
TC-MSG-012	Get with Pagination	Conv has messages	200 OK	200 OK	Pass
TC-MSG-013	Send XSS Script	User is member	200 OK	500 Server Error	Fail
TC-MSG-014	Send SQL Injection	User is member	200 OK	200 OK	Pass

4. Defect Reports

4.1 Summary

Total Defects: 24

Severity	Count	Percentage
Critical	3	12.5%
High	8	33.3%
Medium	11	45.8%
Low	2	8.3%

4.2 Critical Defects

DEFECT-015: Signup Without Password

Severity: Critical | **Test Case:** TC-AUTH-006

Steps: POST /auth/signup with `{"username":"test","email":"test@test.com","password":""}`

Expected: 400 Bad Request - "Password required"
Actual: 201 Created - Account created

Impact: Users can create accounts without passwords, completely breaking authentication security.

DEFECT-016: Login Without Password

Severity: Critical | **Test Case:** TC-AUTH-017

Steps: POST /auth/login with `{"username":"test","password":""}`

Expected: 401 Unauthorized
Actual: 200 OK - Returns JWT token

Impact: Complete authentication bypass. Anyone can login without password.

DEFECT-017: SQL Injection Vulnerability

Severity: Critical | **Test Case:** TC-AUTH-015

Steps: POST /auth/signup with `{"username":"' OR '1'='1","email":"test@test.com","password":"1234"}`

Expected: 400 Bad Request - Input rejected
Actual: 201 Created - Account created with malicious username

Impact: SQL injection attack vector. No input sanitization.

4.3 High Severity Defects

DEFECT-001: User ID Modification

Test Case: TC-USER-009

Issue: User IDs can be modified via PATCH /users/{userId}

Impact: Breaks data integrity. User IDs are primary keys referenced throughout system.

DEFECT-004: Conversation with Non-existent User

Test Case: TC-CONV-005

Issue: System creates conversations with invalid user IDs

Impact: Orphaned data structures that fail at runtime.

DEFECT-006: Authorization Bypass

Test Case: TC-CONV-012

Issue: Non-creators can add members to GROUP conversations

Impact: Violates access control rules. Only creators should manage membership.

DEFECT-008: DIRECT Conversation Constraint Violation

Test Case: TC-CONV-014

Issue: Third member can be added to DIRECT conversations

Impact: Breaks DIRECT conversation definition (should have exactly 2 members).

DEFECT-013: XSS Causes Server Crash

Test Case: TC-MSG-013

Issue: Sending `<script>alert('XSS')</script>` causes 500 Internal Server Error

Impact: DoS vulnerability and potential XSS exploitation.

DEFECT-018: Invalid Email Accepted

Test Case: TC-AUTH-005

Issue: System accepts "NOT A MAIL" as valid email

Impact: No email validation. Breaks email-based functionality.

DEFECT-019: Duplicate Emails Allowed

Test Case: TC-AUTH-016

Issue: Multiple accounts can use same email address

Impact: Account recovery and user identification issues.

DEFECT-020: Server Crash on Missing Username

Test Case: TC-AUTH-018

Issue: Login with only email field causes 500 error

Impact: Server instability on predictable input.

4.4 Medium Severity Defects

- **DEFECT-002:** Self-blocking allowed (TC-USER-012)
- **DEFECT-005:** Conversation with blocked user (TC-CONV-006)
- **DEFECT-007:** Blocked user added to GROUP (TC-CONV-013)
- **DEFECT-009:** Server error on unauthorized action (TC-CONV-016)
- **DEFECT-011:** Empty message accepted (TC-MSG-004)
- **DEFECT-012:** Wrong HTTP status for blocked message (TC-MSG-006)
- **DEFECT-021:** Wrong error message for empty email (TC-AUTH-004)
- **DEFECT-022:** Email field ignored in login (TC-AUTH-020)
- **DEFECT-023:** Logout for non-existent user succeeds (TC-AUTH-021)
- **DEFECT-024:** Empty username in logout accepted (TC-AUTH-022)
- **DEFECT-026:** Extra parameters not filtered (TC-AUTH-024)

4.5 Low Severity Defects

- **DEFECT-003:** Empty GROUP name accepted (TC-CONV-004)
- **DEFECT-025:** Invalid username format in logout (TC-AUTH-023)

4.6 Defect Categories

1. **Critical Auth Vulnerabilities (3):** Passwordless signup/login, SQL injection
2. **Data Integrity (6):** User ID modification, invalid user references, duplicate emails
3. **Input Validation (5):** Missing email/username/password validation
4. **Server Stability (3):** XSS crash, missing field crash, authorization error
5. **Authorization (2):** Non-creator permissions, access control bypass
6. **Business Logic (4):** Self-blocking, blocked user interactions
7. **API Design (3):** Wrong status codes, ignored fields, extra parameters

---

5. Risk & Recommendations

5.1 High Risk Issues

Authentication & Authorization

- **DEFECT-016 (Critical):** Login without password returns valid JWT token - Complete authentication bypass

- **DEFECT-015 (Critical):** Signup without password creates account - Authentication security completely broken
- **DEFECT-017 (Critical):** SQL injection vulnerability in signup endpoint - Malicious usernames accepted without sanitization
- **DEFECT-019 (High):** Duplicate emails allowed - Multiple accounts can share same email address
- **DEFECT-018 (High):** Invalid email formats accepted - No validation for email structure (e.g., "NOT A MAIL" accepted)
- Missing or invalid JWT tokens can access some protected APIs
- Missing username/password fields allowed in signup and login requests

Recommendations:

- **IMMEDIATE:** Fix passwordless authentication bypass (DEFECT-015, DEFECT-016) - CRITICAL security vulnerability
- **IMMEDIATE:** Add SQL injection protection with input sanitization (DEFECT-017)
- Implement email format validation using regex patterns (DEFECT-018)
- Enforce email uniqueness constraints (DEFECT-019)
- Make username and password mandatory fields with proper validation
- Enforce JWT token-based access control for all protected endpoints
- Implement minimum password requirements (length, complexity)

Data Integrity & System Stability

- **DEFECT-001 (High):** User IDs can be modified via PATCH /users/{userId} - Breaks primary key integrity and all references
- **DEFECT-004 (High):** Conversations created with non-existent user IDs - Orphaned data structures cause runtime failures
- **DEFECT-020 (High):** Server crash (500 error) when login uses email field without username - Predictable server instability
- **DEFECT-013 (High):** XSS payload causes server crash (500 error) - DoS vulnerability and potential XSS exploitation

Recommendations:

- **IMMEDIATE:** Make user IDs immutable after creation (DEFECT-001)
- **IMMEDIATE:** Add input sanitization to prevent XSS-induced crashes (DEFECT-013)
- Validate user existence before creating conversations (DEFECT-004)
- Add proper error handling for missing required fields to prevent server crashes (DEFECT-020)
- Implement comprehensive logging for all 500 errors

Authorization & Access Control

- **DEFECT-006 (High):** Non-creators can add members to GROUP conversations - Access control bypass
- **DEFECT-008 (High):** DIRECT conversations allow more than 2 members - Business logic violation

Recommendations:

- **IMMEDIATE:** Enforce creator-only permissions for GROUP membership management (DEFECT-006)
- **IMMEDIATE:** Implement DIRECT conversation constraint validation (exactly 2 members) (DEFECT-008)
- Add comprehensive authorization checks before all membership operations
- Return proper 403 Forbidden for authorization failures (not 500 errors)

5.2 Medium Risk Issues

User Management

- Users can update profile information of other users without authorization
- Users can block other users without proper permission checks
- Users can block themselves or attempt to block non-existent users
- Authorization failures return 500 errors instead of proper 403 responses

Recommendations:

- Restrict user profile updates to authenticated user (self-update only)
- Implement proper authorization checks for blocking operations
- Validate user existence before allowing block/unblock operations
- Prevent users from blocking themselves
- Return appropriate HTTP status codes (403) for authorization failures

Conversation Management

- Non-existent or blocked users can be added to conversations
- Non-creators can add or remove members from GROUP conversations
- DIRECT conversations allow more than 2 members
- Empty group names are accepted during creation
- User IDs can be modified, breaking conversation memberships

Recommendations:

- Validate user existence before adding to conversations
- Check blocking relationships before allowing conversation creation
- Allow only conversation creators to manage GROUP membership
- Enforce DIRECT conversation constraint (exactly 2 members)
- Require non-empty group names with minimum length validation
- Make user IDs immutable after creation

Message Operations

- Users can send messages to blocked users in some scenarios

- Empty messages are accepted and stored
- Messages can be sent to non-existent conversations
- XSS scripts in message content cause server crashes (500 errors)
- Wrong HTTP status codes returned for blocked messaging scenarios

Recommendations:

- Block all messaging between users who have blocked each other in DIRECT chats
- Reject empty messages with proper validation
- Validate conversation existence before sending messages
- Sanitize all message inputs to prevent XSS and other injection attacks
- Return correct HTTP status codes for all error scenarios

5.3 Priority Roadmap

Immediate Fixes (P0 - Security Critical):

1. Fix passwordless authentication bypass
2. Add SQL injection protection
3. Implement XSS input sanitization
4. Enforce JWT token validation on all protected endpoints

Short Term Fixes (P1 - Data Integrity): 5. Make user IDs immutable 6. Validate user existence in all operations 7. Add email format and uniqueness validation 8. Enforce DIRECT conversation constraints

Medium Term Fixes (P2 - Business Logic): 9. Implement proper authorization checks (self-update, creator permissions) 10. Add blocking relationship validation 11. Standardize error responses with correct HTTP codes 12. Implement comprehensive input validation middleware

5.4 Testing & Quality Improvements

Current State: 31% test failure rate (22 out of 72 tests failed) indicates significant quality issues.

Recommendations:

- Implement automated unit tests for business logic validation
- Add integration tests for all API endpoints
- Establish security testing for common vulnerabilities (OWASP Top 10)
- Develop negative test coverage for edge cases and invalid inputs
- Set up continuous integration with automated test execution

Code Quality:

- Establish code review checklist focusing on validation, authorization, and error handling
- Implement static code analysis tools

- Add logging for all 400/500 errors to identify issues early
- Create API documentation with expected behaviors and error codes

5.5 Business Impact Assessment

- **Current State:** System is NOT production-ready due to critical security vulnerabilities
- **Production Risk:** CRITICAL - Do not deploy without fixing authentication bypass and SQL injection
- **User Trust:** Data integrity issues will erode user confidence and increase support burden
- **Compliance:** Authentication vulnerabilities likely violate security compliance standards
- **Recommendation:** Complete Phase P0 fixes before any production consideration

---

6. Individual Reflections

Fariha Islam - Authentication Testing

I was responsible for testing the Authentication APIs, including signup and login functionality. During my testing, I discovered critical security issues such as duplicate User IDs being allowed, unverified email formats being accepted, and the system permitting signup/login with missing username or password fields. Most concerning was the ability to access the system without proper credentials.

My recommendations focused on enforcing unique usernames, implementing proper email validation, making authentication fields mandatory, and strictly enforcing JWT token-based access control for all protected endpoints. These findings highlight the need for comprehensive input validation at the authentication layer.

Eila Afrin - User Management Testing

I focused on testing the User Management APIs, covering user retrieval, profile updates, and block/unblock operations. My testing revealed significant authorization issues where users could update profiles of other users without proper permission checks. I also found that users could block themselves or attempt to block non-existent users, indicating missing validation logic.

I recommended implementing strict authorization checks to ensure users can only modify their own profiles, validating user existence before allowing any block/unblock operations, and preventing illogical actions like self-blocking. These improvements would significantly enhance the security and reliability of user management operations.

A. M. Shahriar Rashid Mahe - Conversation Testing & DevOps

My primary responsibility was testing the Conversation APIs, where I identified multiple critical issues. The system allowed adding blocked or non-existent users to conversations, permitted non-creators to modify

GROUP membership (violating the specification), and accepted empty group names. I also discovered that DIRECT conversations could have more than 2 members, breaking the fundamental conversation type constraint.

I recommended that only conversation creators should have permission to manage GROUP membership, all group names should be validated for content, and strict enforcement of conversation type rules.

Additional Contributions:

- Containerized the entire project using Docker and created a Docker image uploaded to GitHub, making deployment significantly easier for team members
- Developed `setup-test-data.js` - a JavaScript utility program that populates the system with dummy test data, streamlining the testing process for the entire team
- Collaborated on comprehensive GitHub documentation of all testing activities and findings
- Assisted in risk analysis and final report compilation

Saida Zaman - Message Testing & Documentation

I was assigned to test the Message APIs, including sending and retrieving messages in both DIRECT and GROUP conversations. My testing uncovered several issues: empty messages were accepted without validation, blocked users could still send/receive messages in certain scenarios, the system allowed sending messages to non-existent conversations, and XSS scripts in message content caused server crashes.

I recommended implementing content validation to reject empty messages, enforcing blocking rules consistently across all message operations, validating conversation existence before message operations, and sanitizing all message inputs to prevent XSS and other injection attacks.

Additional Contributions:

- Collaborated on comprehensive documentation of all failed test cases
- Assisted in final report compilation and formatting
- Contributed to ensuring all recommendations were practical and actionable

---

7. Bonus Work

Automated Test Data Setup Tool

Our team developed `setup-test-data.js`, a general-purpose testing utility that automates the creation of a complete test environment. This tool goes beyond basic scripting to provide:

Features:

- Automated user creation with JWT token management
- Pre-configured conversation scenarios (DIRECT and GROUP)
- Message history generation for testing
- Blocking relationship setup for authorization testing
- Configurable test data sets via command-line flags
- Error handling and rollback capabilities

Usage:

```
node setup-test-data.js --full    # Complete test environment
node setup-test-data.js --users   # Users only
node setup-test-data.js --conv    # Users + Conversations
```

Benefits:

- Reduced manual test setup time from 30+ minutes to under 1 minute
- Consistent test environment across team members
- Enabled rapid test iteration and regression testing
- Facilitated complex scenario testing (blocking, permissions, etc.)

Repository: All testing tools, documentation, and findings are available in our GitHub repository with complete setup instructions.

End of Report

Testing Period: January 5-12, 2026
Team Members: Fariha Islam, Eila Afrin, A. M. Shahriar Rashid Mahe, Saida Zaman
Tools Used: Postman, Docker, Node.js (test data automation)
Test Environment: Docker containerized application (localhost:8080)
Our each test cases can be found in the testCases.md file in out GitHub repo with screenshots.