Quality Assurance Activity Report

SkillSwap Learning Network Project

Project: SkillSwap - Peer-to-Peer Skill Exchange Platform
Team: SkillSwap Development Team
Date: December 2024
Report Type: Quality Assurance Activity Report

Executive Summary

This report documents the comprehensive quality assurance activities conducted for the SkillSwap Learning Network project. The project is a full-stack web application built with Node.js/Express backend and React frontend, implementing a peer-to-peer skill exchange platform with real-time messaging, notifications, and a points-based economy system.

---

1. Unit Testing Analysis

1.1 Number of Unit Tests Written

Total Unit Tests: 20 unit tests across 3 test suites

Test Distribution:

- **Authentication Routes (auth.test.js):** 6 tests
- **Skills Routes (skills.test.js):** 7 tests
- **Database Service (database.test.js):** 7 tests

1.2 Code Units Tested

Authentication Module (routes/auth.js):

- User registration with validation
- User login with credential verification
- JWT token generation and validation
- Password validation (minimum 6 characters)
- Email uniqueness validation
- Error handling for invalid credentials

Skills Module (routes/skills.js):

- Get all available skills
- Get skill categories
- Search teachers with filters
- Error handling for database failures
- API response formatting

Database Service (models/Database.js):

- User creation and management
- User lookup by email, ID, and userID
- Exchange creation and management
- Transaction recording
- Database connection handling

1.3 Sample Unit Test Code

Test 1: User Registration Validation

```
it('should register a new user with valid data', async () => {
  const userData = {
    name: 'Test User',
    email: 'test@example.com',
    password: 'password123',
    location: 'Stockholm, Sweden',
    bio: 'Test bio'
  };

  database.findUserByEmail.mockResolvedValue(null);
  database.createUser.mockResolvedValue({
    id: 'test-id',
    userID: 'SSL123456',
    ...userData,
    pointsBalance: 100
  });

  const response = await request(app)
    .post('/api/auth/register')
    .send(userData)
    .expect(201);

  expect(response.body.success).toBe(true);
  expect(response.body.data.user.name).toBe(userData.name);
  expect(response.body.data.user.email).toBe(userData.email);
  expect(response.body.data.user.pointsBalance).toBe(100);
  expect(response.body.data.token).toBeDefined();
  expect(database.createUser).toHaveBeenCalledWith(userData);
});
```

Test 2: Skills Search Functionality

```
it('should search skills with valid parameters', async () => {
  const mockSearchResults = [
    {
      id: 1,
      skillName: 'JavaScript',
      category: 'Programming',
      userName: 'John Doe',
      userLocation: 'Stockholm, Sweden',
      pointsCost: 50
    }
  ];

  database.searchTeachers.mockResolvedValue(mockSearchResults);

  const response = await request(app)
    .get('/api/skills/search')
    .query({
      skill: 'JavaScript',
      category: 'Programming',
      location: 'Stockholm'
    })
    .expect(200);

  expect(response.body.success).toBe(true);
  expect(response.body.data).toEqual(mockSearchResults);
  expect(database.searchTeachers).toHaveBeenCalledWith(
    expect.objectContaining({
      skill: 'JavaScript',
      category: 'Programming',
      location: 'Stockholm'
    })
  );
});
```

---

2. Code Coverage Analysis

2.1 Overall Coverage Statistics

Current Code Coverage:

- **Statements:** 12.58% (164/1,304 statements)
- **Branches:** 8.17% (80/979 branches)
- **Functions:** 11.57% (15/130 functions)
- **Lines:** 12.62% (163/1,291 lines)

2.2 Coverage by Module

Authentication Routes (routes/auth.js):

- **Statements:** 52.05%
- **Branches:** 59.25%

- **Functions:** 25%
- **Lines:** 52.05%

Skills Routes (routes/skills.js):

- **Statements:** 48.21%
- **Branches:** 25%
- **Functions:** 50%
- **Lines:** 48.21%

Database Service (models/Database.js):

- **Statements:** 17.02%
- **Branches:** 3.77%
- **Functions:** 18.6%
- **Lines:** 17.14%

Untested Modules:

- Server.js (0% coverage)
- Exchanges routes (0% coverage)
- Notifications routes (0% coverage)
- Users routes (0% coverage)
- NotificationService (0% coverage)

2.3 Coverage Analysis

The current coverage is relatively low at 12.58%, primarily due to:

1. **Limited test scope:** Tests focus on core authentication and skills functionality
2. **Untested modules:** Several major components (exchanges, notifications, users) have no unit tests
3. **Integration complexity:** Real-time features (WebSocket, notifications) require integration testing
4. **Database interactions:** Many database operations are not fully tested due to mocking limitations

---

3. Integration Testing

3.1 Integration Tests Status

Number of Integration Tests: 0 formal integration tests

Reason for No Integration Tests:

1. **Time Constraints:** Development timeline focused on core functionality delivery
2. **Manual Testing Approach:** Comprehensive manual testing was conducted instead
3. **Real-time Complexity:** WebSocket and notification systems require complex integration setup
4. **Database Dependencies:** Integration tests would require full database setup and teardown

3.2 Manual Integration Testing Conducted

Areas Manually Tested:

- User registration and login flow
- Skill search and filtering
- Exchange request and notification system

- Real-time messaging between users
- Points system and transactions
- Rating and review system
- Cross-browser compatibility
- Network access from multiple devices

---

4. Acceptance Testing

4.1 Acceptance Tests Status

Number of Acceptance Tests: 0 formal acceptance tests

Reason for No Acceptance Tests:

1. **Manual Testing Preference:** Team conducted extensive manual acceptance testing
2. **User Story Validation:** All user stories were manually validated through comprehensive testing
3. **End-to-End Complexity:** Full user journeys involve multiple systems (auth, messaging, notifications, payments)
4. **Time Investment:** Automated acceptance tests would require significant additional development time

4.2 Manual Acceptance Testing

User Stories Validated:

- User can register and create account
- User can search for skills and teachers
- User can request skill exchanges
- Teachers receive notifications for exchange requests
- Users can communicate via real-time messaging
- Exchange completion triggers points transfer
- Users can rate completed exchanges
- System supports Swedish localization
- Application works across network devices

---

5. Bug Statistics and Analysis

5.1 Bugs Found During Testing

Total Bugs Identified: 8 major bugs

Bug Categories:

1. **Privacy Issues:** 1 bug (points balance visible to all users)
2. **Functionality Issues:** 3 bugs (exchange requests, search filters, messages page)
3. **Localization Issues:** 1 bug (Swedish character support)
4. **Rating System Issues:** 1 bug (bidirectional rating instead of one-way)
5. **Message Duplication:** 1 bug (messages appearing twice)
6. **Points Display:** 1 bug (points not visible in navbar)

5.2 Detailed Bug Analysis

Bug #1: Profile Points Privacy

- **Severity:** Medium
- **Description:** User points balance was visible when viewing other users' profiles
- **Resolution:** Added conditional rendering to hide points from other users
- **Files Modified:** client/src/pages/Profile.js

Bug #2: Request Exchange Functionality

- **Severity:** High
- **Description:** Exchange request button not working, no notifications sent
- **Resolution:** Fixed API integration and notification system
- **Files Modified:** client/src/pages/SkillSearch.js, routes/exchanges.js

Bug #3: Advanced Search Filters

- **Severity:** Medium
- **Description:** Search filters not combining correctly
- **Resolution:** Fixed filter parameter handling and state management
- **Files Modified:** client/src/pages/SkillSearch.js

Bug #4: Messages Page Loading

- **Severity:** High
- **Description:** Messages page failing to load due to API configuration
- **Resolution:** Created centralized axios configuration
- **Files Modified:** client/src/config/axios.js, multiple components

Bug #5: Swedish Localization

- **Severity:** Low
- **Description:** Special characters (Ö, Å, Ä) not displaying correctly
- **Resolution:** Fixed character encoding and database handling
- **Files Modified:** Database schema, frontend components

Bug #6: One-Way Rating System

- **Severity:** Medium
- **Description:** Teachers could rate students back (unwanted behavior)
- **Resolution:** Implemented one-way rating (students rate teachers only)
- **Files Modified:** client/src/pages/ExchangeDetails.js, routes/exchanges.js

Bug #7: Message Duplication

- **Severity:** Medium
- **Description:** Messages appearing twice in conversation panel
- **Resolution:** Fixed double-add issue in socket handling
- **Files Modified:** Socket context and message handling

Bug #8: Points Display in Navbar

- **Severity:** Low
- **Description:** User points not visible in navigation bar
- **Resolution:** Added points display to navbar component
- **Files Modified:** client/src/components/Navbar.js

5.3 Time Spent on Bug Removal

Total Time Invested: Approximately 24-30 hours across development sprints

Time Breakdown:

- **Bug Investigation:** 8-10 hours
- **Code Analysis:** 6-8 hours
- **Implementation of Fixes:** 8-10 hours
- **Testing and Validation:** 2-4 hours

Sprint Distribution:

- **Sprint 1:** 4 hours (Bugs #1, #2, #3)
- **Sprint 2:** 8 hours (Bug #4 - major API configuration issue)
- **Sprint 3:** 6 hours (Bugs #5, #6)
- **Sprint 4:** 4 hours (Bugs #7, #8)
- **Final Testing:** 2-4 hours (comprehensive validation)

---

6. Confidence Assessment

6.1 Product Readiness for Demonstration

Overall Confidence Level: 85% confident that the product is ready for demonstration

6.2 Confidence Breakdown

High Confidence Areas (90-95%):

- User authentication and registration
- Skill search and filtering functionality
- Basic exchange request system
- Real-time messaging
- Points system and transactions
- Swedish localization
- Network accessibility

Medium Confidence Areas (75-85%):

- ⚠ Notification system reliability
- ⚠ Complex search filter combinations
- ⚠ Cross-browser compatibility
- ⚠ Performance under load

Areas Requiring Attention (60-75%):

- ⚠ Error handling in edge cases
- ⚠ Database performance with large datasets
- ⚠ Security validation for all inputs
- ⚠ Mobile responsiveness

6.3 Risk Assessment

Low Risk:

- Core functionality is stable and tested
- Manual testing has been comprehensive
- All major bugs have been resolved

Medium Risk:

- Limited automated test coverage (12.58%)
- No formal integration or acceptance tests
- Some edge cases may not be covered

Mitigation Strategies:

- Comprehensive manual testing documentation
- Detailed troubleshooting guides
- Fallback error handling implemented
- User feedback collection system ready

6.4 Demonstration Readiness

Ready for Demonstration: Yes, with the following considerations:

Strengths:

- All core features working as expected
- Comprehensive manual testing completed
- Bug-free core user journeys
- Professional UI/UX implementation
- Real-time features functioning correctly

Recommendations:

- Have backup demo data prepared
- Test on multiple devices before presentation
- Prepare troubleshooting steps for common issues
- Document known limitations for transparency

---

7. Recommendations for Future Development

7.1 Testing Improvements

1. **Increase Unit Test Coverage:**
    - Target 70%+ statement coverage
    - Add tests for exchanges, notifications, and users modules
    - Implement database integration tests
2. **Implement Integration Testing:**
    - Set up test database for integration tests
    - Test complete user journeys
    - Validate API endpoint interactions

3. **Add Acceptance Testing:**
    - Implement end-to-end testing framework
    - Automate critical user scenarios
    - Set up continuous integration testing

7.2 Quality Assurance Process

1. **Code Review Process:**
    - Implement mandatory code reviews
    - Add automated code quality checks
    - Establish coding standards documentation
2. **Testing Strategy:**
    - Develop comprehensive testing strategy
    - Implement test-driven development practices
    - Add performance testing
3. **Documentation:**
    - Maintain up-to-date testing documentation
    - Document all known issues and limitations
    - Create user acceptance criteria

Conclusion

The SkillSwap Learning Network project has undergone comprehensive quality assurance activities, including unit testing, manual integration testing, and extensive bug resolution. While the automated test coverage is currently at 12.58%, the project demonstrates high quality through thorough manual testing and successful resolution of all identified bugs.

The team is 85% confident in the product's readiness for demonstration, with all core functionality working correctly and major issues resolved. The comprehensive manual testing approach has validated all critical user journeys, and the system is ready for presentation with appropriate preparation and backup plans.

Key Achievements:

- 20 unit tests implemented and passing
- 8 major bugs identified and resolved
- 24-30 hours invested in quality assurance
- Comprehensive manual testing completed
- All core features validated and working

Next Steps:

- Implement additional unit tests for untested modules
- Develop integration testing framework
- Establish continuous integration pipeline
- Create automated acceptance testing suite

Report Prepared By: SkillSwap Development Team
Date: December 2024
Version: 1.0