# Testing Documentation

Rikugan Project Management System

**Document Version:** 1.0
**Printing Date:** December 22, 2025

**Group ID:** Group 9


**Group Members:**

1155211784 Fok Chun Yin
1155211541 HO Sum Ming
1155211779 Chi Ho KWOK
1155211479 NG Ka Long
1155214295 SIN Lok Man


**Revision History:**

| Version | Date | Author | Description |
|---|---|---|---|
| 1.0 | December 22, 2025 | Group 9 | Initial release |

## Contents

# 1 Test Plan Overview

This document outlines the testing strategy for DSCPMS, covering backend API testing (Jest + Supertest) and frontend user requirement validation. The test suite ensures system reliability, security, and compliance with SRS functional requirements.

## 1.1 Testing Scope

- RESTful API endpoints and authentication
- Database operations and transaction integrity
- Role-based access control (RBAC)
- License validation and team management
- Bounty system and business logic

# 2 Testing Strategy

## 2.1 Backend Testing

### 2.1.1 Unit Testing

- **Framework:** Jest
- **Scope:** Individual components, services, utilities, and models in isolation
- **Approach:** Mock external dependencies, test business logic, validate data transformations
- **Coverage:** Service layer functions, utility functions, model methods

### 2.1.2 Integration Testing

- **Framework:** Jest + Supertest
- **Scope:** API endpoints, database operations, component interactions
- **Approach:** Test complete request-response cycles, validate database transactions, verify middleware chains
- **Coverage:** RESTful API endpoints, authentication flows, database integrity, inter-service communication

## 2.2 Frontend Testing

### 2.2.1 Validation Testing

- **Approach:** Manual testing and user acceptance testing (UAT)
- **Scope:** User interface workflows, form validations, SRS requirement compliance
- **Coverage:** User requirement validation, end-to-end scenarios, role-based feature access
- **Focus:** Input validation, error messages, user feedback, navigation flows

## 2.3 Overall Metrics

- **Coverage Target:** 70%+ for critical paths (currently 86%)
- **Test Types:** Functional, security, negative testing, data integrity validation, UAT

# 3 Test Coverage Summary

| Component | Tests | Coverage |
|---|---|---|
| Authentication (auth.test.js) | 41 | ¿90% |
| Task Management (tasks.test.js) | 24 | ¿85% |
| License System (license.test.js) | 47 | ¿90% |
| User Management (user.test.js) | 43 | ¿70% |
| Bounty System (bounty.test.js) | 14 | ¿90% |
| Notifications (notifications.test.js) | 10 | ¿90% |
| Team Management (team.test.js) | 20 | ˜71% |
| Database (database.test.js) | 49 | ˜100% |
| **Total** | **248** | **˜86%** |

# 4 Core Test Cases

**Note:** Complete test suite (248 tests) available in `backend/__tests__/` directory.

## 4.1 Authentication (auth.test.js)

### 4.1.1 TEST-AUTH-001: User Registration

**Endpoint:** `POST /api/v1/auth/register`

**Validates:** New users register with team_id=NULL, password hashing, role assignment

```
const res = await request(app)
  .post('/api/v1/auth/register')
  .send({
    username: 'testuser',
    email: 'test@example.com',
    password: 'Test123!',
    role: 'GOON'
  });

expect(res.statusCode).toBe(201);
expect(res.body.success).toBe(true);
```

### 4.1.2 TEST-AUTH-002: Login with No Team Flag

**Endpoint:** `POST /api/v1/auth/login`

**Validates:** JWT token generation, no_team flag when teamId is null

```
const res = await request(app)
  .post('/api/v1/auth/login')
  .send({
    username: 'logintest',
    password: 'Test123!'
  });

expect(res.statusCode).toBe(200);
expect(res.body.data.no_team).toBe(true);
```

### 4.1.3 TEST-AUTH-003: Expired License Rejection

**Endpoint:** `POST /api/v1/auth/login`

**Validates:** Login blocked for expired team licenses (HTTP 403)

```
const res = await request(app)
  .post('/api/v1/auth/login')
  .send({
    username: 'expireduser',
    password: 'Test123!'
  });

expect(res.statusCode).toBe(403);
expect(res.body.message).toMatch(/license|expired/i);
```

## 4.2 Task Management (tasks.test.js)

### 4.2.1 TEST-TASK-001: Task Creation by HASHIRA

**Endpoint:** `POST /api/v1/tasks`

**Validates:** HASHIRA/OYAKATASAMA can create tasks with bounty

```
const res = await request(app)
  .post('/api/v1/tasks')
  .set('Authorization', `Bearer ${hashiraToken}`)
  .send({
    title: 'Test Task',
    description: 'Task description',
    bountyAmount: 100.00,
    priority: 'HIGH'
  });

expect(res.statusCode).toBe(201);
expect(res.body.success).toBe(true);
```

### 4.2.2 TEST-TASK-002: GOON Role Restriction

**Endpoint:** `POST /api/v1/tasks`

**Validates:** GOON role cannot create tasks (HTTP 403)

```
const res = await request(app)
  .post('/api/v1/tasks')
  .set('Authorization', `Bearer ${goonToken}`)
  .send({
    title: 'Test Task',
    description: 'Task description'
  });

expect(res.statusCode).toBe(403);
```

### 4.2.3 TEST-TASK-003: Task Assignment

**Endpoint:** `POST /api/v1/tasks/:id/assign`

**Validates:** Task assignment, status update, notification creation

```
const res = await request(app)
  .post(`/api/v1/tasks/${taskId}/assign`)
  .set('Authorization', `Bearer ${goonToken}`);

expect(res.statusCode).toBe(200);
expect(res.body.data.status).toBe('IN_PROGRESS');
```

## 4.3 License System (license.test.js)

### 4.3.1 TEST-LICENSE-001: Valid License Validation

**Validates:** Environment-configured license keys accepted

```
const result = await LicenseService.validateForTeamCreation(
  'RIKUGAN-2025-VALID-KEY-A'
);

expect(result.valid).toBe(true);
expect(result.config.max_users).toBe(50);
```

### 4.3.2 TEST-LICENSE-002: Invalid License Rejection

**Validates:** Unknown license keys rejected

```
const result = await LicenseService.validateForTeamCreation(
  'INVALID-KEY-12345'
);

expect(result.valid).toBe(false);
expect(result.error).toMatch(/invalid|not found/i);
```

### 4.3.3 TEST-LICENSE-003: Expiration Enforcement

**Validates:** Expired licenses blocked (HTTP 403)

```
const result = await LicenseService.validateForTeamCreation(
  'RIKUGAN-TEST-2025-EXPIRED'
);

expect(result.valid).toBe(false);
expect(result.error).toMatch(/expired/i);
```

### 4.3.4 TEST-LICENSE-004: User Limit Enforcement

**Validates:** Teams cannot exceed max_users limit

```
const res = await request(app)
  .post(`/api/v1/teams/${teamId}/members`)
  .set('Authorization', `Bearer ${adminToken}`)
  .send({ userId: newUserId });

expect(res.statusCode).toBe(403);
expect(res.body.message).toMatch(/user limit/i);
```

### 4.3.5 TEST-LICENSE-005: Task Limit (Unlicensed)

**Validates:** 3-task limit for teams without valid license

```
const res = await request(app)
  .post('/api/v1/tasks')
  .set('Authorization', `Bearer ${hashiraToken}`)
  .send({ title: 'Task 4', description: 'Fourth task' });

expect(res.statusCode).toBe(403);
expect(res.body.message).toMatch(/task limit/i);
```

## 4.4 Bounty System (bounty.test.js)

### 4.4.1 TEST-BOUNTY-001: Automatic Reward Distribution

**Validates:** Bounty awarded on task completion, balance update

```
await request(app)
  .put(`/api/v1/tasks/${taskId}/status`)
  .set('Authorization', `Bearer ${goonToken}`)
  .send({ status: 'COMPLETED' });

const transaction = await Transaction.findOne({
  where: { taskId, type: 'BOUNTY' }
});

expect(transaction).toBeDefined();
expect(transaction.amount).toBe(bountyAmount);
```

### 4.4.2 TEST-BOUNTY-002: Penalty Application

**Validates:** Penalties for missed deadlines, negative transactions

```
const penalty = await Transaction.findOne({
  where: { taskId, type: 'PENALTY' }
});

expect(penalty).toBeDefined();
expect(penalty.amount).toBeLessThan(0);
expect(user.balance).toBeLessThan(initialBalance);
```

## 4.5 Team Management (team.test.js)

### 4.5.1 TEST-TEAM-001: Team Creation

**Endpoint:** `POST /api/v1/teams/create`

**Validates:** Team created with license, creator as OYAKATASAMA

```
const response = await request(app)
  .post('/api/v1/teams/create')
  .set('Authorization', `Bearer ${adminToken}`)
  .send({
    teamName: 'Alpha Team',
    licenseKey: 'RIKUGAN-2025-TEAM-A'
  });

```

```
expect(response.status).toBe(200);
expect(response.body.success).toBe(true);
```

#### 4.5.2 TEST-TEAM-002: Resource Isolation

**Validates:** Users can only access their team's resources

```
const res = await request(app)
  .get('/api/v1/tasks')
  .set('Authorization', `Bearer ${team1Token}`);

const tasks = res.body.data;
expect(tasks.every(t => t.teamId === team1Id)).toBe(true);
```

### 4.6 Notifications (notifications.test.js)

#### 4.6.1 TEST-NOTIF-001: Multi-Type Creation

**Validates:** All notification types (TASK_ASSIGNED, BOUNTY_RECEIVED, etc.)

```
const taskNotif = await NotificationService.createNotification(
  testUser1.id, 'TASK_ASSIGNED',
  { taskTitle: 'New Task', assignerName: 'Manager',
    bountyAmount: 100, taskId: testTask.id }
);

expect(taskNotif.type).toBe('TASK_ASSIGNED');
expect(taskNotif.title).toBe('New Task Assigned');
expect(taskNotif.readStatus).toBe(false);
```

#### 4.6.2 TEST-NOTIF-002: User Isolation

**Validates:** Users only see their own notifications

```
const notifs = await NotificationService.getUserNotifications(
  testUser1.id, { unreadOnly: true }
);

expect(Array.isArray(notifs)).toBe(true);
expect(notifs.every(n => !n.readStatus)).toBe(true);
```

### 4.7 Database (database.test.js)

#### 4.7.1 TEST-DB-001: Connection Health

**Validates:** MySQL connection established

```
await sequelize.authenticate();

expect(sequelize).toBeDefined();
expect(sequelize.options.database).toBe('dscpms_test');
```

#### 4.7.2 TEST-DB-002: Transaction Rollback

**Validates:** Failed transactions rollback completely

```
try {
  await sequelize.transaction(async (t) => {
    await User.create({ username: 'test' }, { transaction: t });
    throw new Error('Rollback test');
  });
} catch (err) {
  const user = await User.findOne({ where: { username: 'test' } });
  expect(user).toBeNull();
}
```

#### 4.7.3 TEST-DB-003: Model Validations

**Validates:** Email format, required fields, enum values

```
try {
  await User.create({ username: 'test', email: 'invalid' });
} catch (err) {
  expect(err.name).toBe('SequelizeValidationError');
}
```

# 5 Testing Environment

## 5.1 Docker Setup

All tests run inside Docker containers for database connectivity and environment consistency.

**Start environment:**

```
docker-compose up -d database backend
docker exec dscpms-database mysql -u root -proot -e "CREATE DATABASE IF NOT EXISTS dscpms_test;"
```

**Run tests:**

```
# All tests
docker exec dscpms-backend bash -c "NODE_ENV=test npm test -- --forceExit"

# Individual suite
docker exec dscpms-backend bash -c "NODE_ENV=test npm test -- auth.test.js --forceExit"

# With coverage
docker exec dscpms-backend bash -c "NODE_ENV=test npm test -- --coverage --forceExit"
```

**Test Database:** dscpms_test (MySQL, host: database, port: 3306, user: root)

# 6 Test Results

## 6.1 Statistics

- **Total:** 248 tests (100% passing)
- **Coverage:** ˜86% overall
- **Execution Time:** 8-18 seconds

## 6.2 Quality Gates

All tests pass

No coverage decrease

No new ESLint errors

API documentation updated

### 6.3 Validation Testing

We have professional users that can confirm the following requirements in the SRS had been met for the frontend:

**REQ-1: User Registration** Oyakatasama can create accounts with username (3-50 chars), email, password (8+ chars), and role.

**REQ-2: User Login** Authenticate via username/email and password; 8-hour session timeout.

**REQ-3: Role-Based Access** **Goons**: Task selection, status updates, profile viewing. **Hashira**: All Goon functions plus task creation and team monitoring. **Oyakatasama**: All functions plus user/system administration.

**REQ-4: License Management** Hard-coded team license key via environment config; invalid license blocks all access.

**REQ-5: Task Creation** Hashira create tasks with name (max 100 chars), description (max 1000 chars), bounty amount (positive), deadline (future date), priority, and skill tags.

**REQ-6: Task Assignment** Goons select available tasks; automatic assignment to first qualified user.

**REQ-7: Kanban-like status** Display task: Available, In Progress, Review, Completed.

**REQ-8: Task Status Updates** Assigned users update status and add progress comments.

**REQ-9: Deadline Management** Track deadlines; apply monetary penalties for missed deadlines.

**REQ-10: Task Deletion** Hashira/Oyakatasama delete tasks only when Available or unassigned.

**REQ-11: User Profile** Maintain username, email, role, balance, task history, and performance statistics.

**REQ-12: Bounty Rewards** Auto-credit user accounts upon task completion.

**REQ-13: Penalty System** Apply configurable penalties for missed deadlines.

**REQ-14: Performance Tracking** Display tasks completed, avg completion time, success rate, total earnings.

**REQ-15: Task Notifications** Notify on: task assignment, 24hr deadline reminders, status updates, completion/cancellation.

# 7 Conclusion

The Rikugan test suite provides 86% coverage across 248 tests, validating authentication, authorization, license management, bounty system, and database integrity. All major SRS requirements are tested with comprehensive security and business logic validation.