**Testing Plan for userRequestSchema Validation**

**🎯 Testing Objectives**

Validate that the userRequestSchema properly:

* Accepts valid user registration data
* Rejects invalid data with appropriate error messages
* Enforces data type constraints
* Enforces format/pattern constraints
* Enforces length constraints
* Validates required vs optional fields
* Validates enum values

**📋 Test Structure**

userRequestSchema Validation Tests

├── Positive Path Tests (Happy Path)

├── Negative Path Tests (Field-by-Field)

├── Edge Cases Tests

└── Integration Tests (Multiple Errors)

**✅ Positive Path Tests (Valid Data)**

**1. Complete Valid User Data**

* Test with ALL fields populated with valid data
* Verify schema accepts and returns validated data
* **Goal:** Confirm schema works with perfect input

**2. Minimal Valid User Data**

* Test with only REQUIRED fields
* Omit all optional fields
* **Goal:** Verify optional fields are truly optional

**3. Boundary Values (Valid)**

* Test minimum valid lengths
* Test maximum valid lengths
* Test minimum/maximum numeric values
* **Goal:** Confirm boundaries are inclusive

**❌ Negative Path Tests (Invalid Data)**

**Field-by-Field Validation**

**A. Email Field**

* ❌ Missing email (if required)
* ❌ Invalid email format (no @)
* ❌ Invalid email format (no domain)
* ❌ Email too short (below minimum)
* ❌ Email too long (above maximum)
* ❌ Email with special characters
* ❌ Wrong data type (number, object, array)

**B. Password Field**

* ❌ Missing password (if required)
* ❌ Password too short
* ❌ Password too long
* ❌ No uppercase letter
* ❌ No lowercase letter
* ❌ No digit
* ❌ No special character
* ❌ Invalid special characters
* ❌ Wrong data type

**C. Phone Number Field**

* ❌ Missing phone\_number (if required)
* ❌ Too short
* ❌ Too long
* ❌ Invalid format (letters included)
* ❌ Invalid country code
* ❌ Wrong data type

**D. First Name Field**

* ❌ Missing first\_name (if required)
* ❌ Empty string
* ❌ Too long
* ❌ Invalid characters (numbers, special chars)
* ❌ Wrong data type

**E. Last Name Field**

* ❌ Missing last\_name (if optional, should pass)
* ❌ Empty string (if provided)
* ❌ Too long
* ❌ Invalid characters
* ❌ Wrong data type

**F. User Role Field**

* ❌ Missing user\_role (if required)
* ❌ Invalid enum value (e.g., "superadmin")
* ❌ Wrong data type
* ❌ Case sensitivity test (e.g., "USER" vs "user")

**G. Size Fields (upper\_size\_number, upper\_size\_letter, etc.)**

* ❌ Invalid number values (out of range)
* ❌ Invalid letter values (not in enum)
* ❌ Wrong data type (string for number field)
* ❌ Missing if required

**H. Email Communications Type**

* ❌ Missing (if required)
* ❌ Invalid enum value
* ❌ Wrong data type

**I. SMS Communications**

* ❌ Missing (if required)
* ❌ Wrong data type (string instead of boolean)
* ❌ Invalid boolean-like values ("true" as string)

**J. ShippingAddresses Array**

* ❌ Not an array (object, string, number)
* ❌ Non-empty array (if should be empty on registration)
* ❌ Array with invalid objects

**K. CartItems Array**

* ❌ Not an array
* ❌ Non-empty array (if should be empty on registration)
* ❌ Array with invalid objects

**🔍 Edge Cases Tests**

**1. Boundary Testing**

* Exact minimum length strings
* Exact maximum length strings
* Minimum numeric values
* Maximum numeric values

**2. Special Characters**

* Unicode characters in names
* Accented letters
* Hyphens and apostrophes in names
* Special characters in allowed fields

**3. Whitespace Handling**

* Leading/trailing spaces
* Multiple spaces between words
* Empty strings vs null vs undefined

**4. Data Type Coercion**

* Numeric strings ("123" for numbers)
* Boolean strings ("true" for booleans)
* Verify schema doesn't auto-convert

**🔗 Integration Tests (Multiple Errors)**

**1. Multiple Invalid Fields**

* Test with 2-3 invalid fields simultaneously
* Verify all errors are returned (not just first one)

**2. Completely Empty Request**

* Send empty object {}
* Verify all required field errors are reported

**3. All Fields Wrong Type**

* Send valid structure but all wrong types
* Verify type errors for each field

**📊 Test Organization Strategy**

describe('userRequestSchema Validation', () => {

describe('Positive Path - Valid Data', () => {

test('should accept complete valid user data');

test('should accept minimal required fields only');

test('should accept valid boundary values');

});

describe('Negative Path - Email Validation', () => {

test('should reject missing email');

test('should reject invalid email format');

test('should reject email too short');

// ... etc

});

describe('Negative Path - Password Validation', () => {

test('should reject weak password (no uppercase)');

test('should reject password too short');

// ... etc

});

describe('Negative Path - Phone Number Validation', () => {

// ... tests

});

describe('Negative Path - Name Validation', () => {

// ... first\_name and last\_name tests

});

describe('Negative Path - User Role Validation', () => {

// ... enum tests

});

describe('Negative Path - Size Fields Validation', () => {

// ... number and letter size tests

});

describe('Negative Path - Communication Preferences', () => {

// ... email\_comms\_type and sms\_comms tests

});

describe('Negative Path - Array Fields Validation', () => {

// ... ShippingAddresses and CartItems tests

});

describe('Edge Cases', () => {

// ... boundary and special character tests

});

describe('Integration - Multiple Errors', () => {

// ... multiple invalid fields tests

});

});

**🎯 Test Assertions Strategy**

For each test, verify:

**Positive Tests:**

✓ Schema validation succeeds (.success === true)

✓ Validated data is returned

✓ No errors are present

**Negative Tests:**

✓ Schema validation fails (.success === false)

✓ Appropriate error message is returned

✓ Error specifies which field failed

✓ Error describes why validation failed

**📈 Coverage Goals**

* **100% of fields tested** (all 14-16 fields)
* **All validation rules tested** (type, format, length, enum)
* **All error paths covered**
* **Minimum 3-5 negative tests per field**
* **Total estimated tests: 50-70 tests**

**🔧 Test Data Strategy**

**Valid Test Data Template:**

const validUserData = {

email: 'valid@example.com',

password: 'ValidPass123!',

phone\_number: '14155552671',

first\_name: 'John',

last\_name: 'Doe',

user\_role: 'user',

upper\_size\_number: 42,

upper\_size\_letter: 'L',

others\_size\_letter: 'M',

email\_comms\_type: 'I want all emails',

sms\_comms: true,

ShippingAddresses: [],

CartItems: []

};

**Invalid Test Data Variations:**

* Create specific mutations for each negative test
* Change ONE field at a time for clarity
* Document why each variation is invalid

**📝 Test Priorities**

**Priority 1 (Critical - Must Test):**

* Required field validation
* Data type validation
* Email format validation
* Password strength validation
* Enum value validation

**Priority 2 (Important - Should Test):**

* Length constraints
* Numeric range validation
* Array structure validation
* Optional field handling

**Priority 3 (Nice to Have):**

* Edge cases
* Special character handling
* Multiple simultaneous errors
* Whitespace handling

**🚀 Expected Outcomes**

After implementing this plan:

* ✅ Confidence that schema properly validates all user input
* ✅ Documentation of all validation rules (via tests)
* ✅ Protection against invalid data reaching the database
* ✅ Clear error messages for API consumers
* ✅ Regression protection when schema changes

**Ready for schema details!** Please provide the exact field definitions, data types, constraints, and validation rules from userRequestSchema, and I'll implement the tests based on this plan. 📋