Test Case Template

Test Case Template

> **Project:** Bilko **Version:** 1.0 **Date:** 2026-02-25 **Author:** Ops Architect **Status:** Final (Template) **Reviewers:** Tech Lead, Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft
1.0	2026-02-25	ALAI Documentation Team	Finalized as reusable template

---

How to Use This Template

Copy the relevant section below for each test case. Financial and accounting test cases require explicit precision validation (NUMERIC(19,4)) and country-specific VAT rates.

Unit Test Case Template

```
// File: apps/api/src/utils/__tests__/<module>.test.ts

import { describe, it, expect } from 'vitest';
import { <functionName> } from '../<module>';

/**
 * TC-ID: UNIT-XXX
 * Module: <module name>
```

```
 * Priority: P0 / P1 / P2
 * Author: <developer name>
 * Date: YYYY-MM-DD
 */
describe('<functionName>', () => {
  // STANDARD CASE
  it('<description of what is tested>', () => {
    // Arrange
    const input = <input value>;

    // Act
    const result = <functionName>(input);

    // Assert
    expect(result).toBe(<expected value>);
  });

  // EDGE CASE
  it('handles <edge case description>', () => {
    // Arrange
    const input = <edge case input>;

    // Act + Assert
    expect(() => <functionName>(input)).toThrow('<error message>');
    // OR
    expect(<functionName>(input)).toBeNull();
  });
});
```

Integration Test Case Template

```
// File: apps/api/src/routes/__tests__/<route>.test.ts

import { describe, it, expect, beforeEach } from 'vitest';
import request from 'supertest';
import { app } from '../../app';
import { prisma } from '../../lib/prisma';
import { createTestOrg, createTestUser, loginTestUser } from '../helpers';
```

```
/**
 * TC-ID: INT-XXX
 * Endpoint: <METHOD> <path>
 * Priority: P0 / P1 / P2
 * Author: <developer name>
 * Date: YYYY-MM-DD
 */
describe('<METHOD> <path>', () => {
  let authToken: string;
  let organizationId: string;

  beforeEach(async () => {
    // Fresh organization + user per test
    const org = await createTestOrg();
    organizationId = org.id;
    authToken = await loginTestUser(org.id);
  });

  it('<success scenario description>', async () => {
    // Arrange
    const payload = { /* request body */ };

    // Act
    const res = await request(app)
      .post('/api/v1/<path>')
      .set('Authorization', `Bearer ${authToken}`)
      .send(payload);

    // Assert
    expect(res.status).toBe(201);
    expect(res.body.<field>).toBe(<expected>);
  });

  it('returns 401 without auth', async () => {
    const res = await request(app).post('/api/v1/<path>').send({});
    expect(res.status).toBe(401);
  });

  it('returns 403 for cross-org access', async () => {
```

```
    // Create resource in a different org
    const otherOrg = await createTestOrg();
    const otherResource = await prisma.<model>.create({
      data: { organizationId: otherOrg.id, /* ... */ }
    });

    const res = await request(app)
      .get(`/api/v1/<path>/${otherResource.id}`)
      .set('Authorization', `Bearer ${authToken}`);

    expect(res.status).toBe(404); // 404, not 403 (no enumeration)
  });
});
```

Accounting-Specific Test Cases (Bilko)

Double-Entry Balance Verification

```
// TC-ID: ACCT-001
// Priority: P0
// Description: Every financial transaction must have balanced debit and credit entries

it('validates double-entry balance — debits must equal credits', () => {
  const entry = {
    debitAccountId: 'acc_receivables',
    creditAccountId: 'acc_revenue',
    amount: new Decimal('50000.0000'), // NUMERIC(19,4)
    currencyCode: 'RSD',
  }

  expect(() => validateDoubleEntry(entry)).not.toThrow()

  const debitTotal = new Decimal('1000.0000')
  const creditTotal = new Decimal('999.9999') // Unbalanced
  expect(() => validateDoubleEntry({ ...entry, creditAmount: creditTotal })).toThrow(
    'Double-entry imbalance: debit 1000.0000 ≠ credit 999.9999',
  )
```

```
})
```

VAT Calculation Accuracy by Country

```
// TC-ID: ACCT-002
// Priority: P0
// Description: VAT must be calculated per country-specific rules with NUMERIC precision

describe('VAT calculation accuracy', () => {
  it('Serbia RS — standard rate 20% on 1000.0000 RSD', () => {
    const result = calculateVAT(new Decimal('1000.0000'), 'RS', 'standard')
    expect(result.vatAmount.toString()).toBe('200.0000')
    expect(result.total.toString()).toBe('1200.0000')
    expect(result.rate).toBe(20)
  })

  it('Bosnia BiH — standard rate 17% on 100.0000 BAM', () => {
    const result = calculateVAT(new Decimal('100.0000'), 'BA', 'standard')
    expect(result.vatAmount.toString()).toBe('17.0000')
    expect(result.total.toString()).toBe('117.0000')
    expect(result.rate).toBe(17)
  })

  it('Croatia HR — standard rate 25% on 100.0000 EUR', () => {
    const result = calculateVAT(new Decimal('100.0000'), 'HR', 'standard')
    expect(result.vatAmount.toString()).toBe('25.0000')
    expect(result.total.toString()).toBe('125.0000')
    expect(result.rate).toBe(25)
  })

  it('exports zero-rated — all countries', () => {
    for (const country of ['RS', 'BA', 'HR']) {
      const result = calculateVAT(new Decimal('5000.0000'), country, 'zero')
      expect(result.vatAmount.toString()).toBe('0.0000')
      expect(result.rate).toBe(0)
    }
  })

  it('NUMERIC precision — no floating point drift', () => {
```

```
    // 0.1 + 0.2 ≠ 0.3 in IEEE 754 — must use Decimal
    const result = calculateVAT(new Decimal('0.1000'), 'RS', 'standard')
    expect(result.vatAmount.toString()).toBe('0.0200') // Not 0.020000000000000004
  })
})
```

Currency Conversion with Rate Locking

```
// TC-ID: ACCT-003
// Priority: P0
// Description: Exchange rate must be locked at transaction date, not current rate

it('locks exchange rate at transaction date — not current rate', async () => {
  const transactionDate = new Date('2026-01-15')
  const historicalRate = new Decimal('117.5000') // EUR/RSD on that date

  // Seed historical rate
  await prisma.exchangeRate.create({
    data: {
      fromCurrency: 'EUR',
      toCurrency: 'RSD',
      rate: historicalRate,
      effectiveDate: transactionDate,
    },
  })

  const result = await lockExchangeRate('EUR', 'RSD', transactionDate)
  expect(result.toString()).toBe('117.5000')

  // Current rate is different — should not matter
  await prisma.exchangeRate.create({
    data: {
      fromCurrency: 'EUR',
      toCurrency: 'RSD',
      rate: new Decimal('118.2000'),
      effectiveDate: new Date(),
    },
  })
```

```
  const lockedResult = await lockExchangeRate('EUR', 'RSD', transactionDate)
  expect(lockedResult.toString()).toBe('117.5000') // Same historical rate
})
```

Invoice Total Calculation with Mixed VAT Rates

```
// TC-ID: ACCT-004
// Priority: P0
// Description: Multi-item invoices with different VAT rates must calculate correctly

it('invoice with mixed VAT rates — NUMERIC precision throughout', () => {
  const items = [
    { description: 'Consulting', quantity: 10, unitPrice: new Decimal('5000.0000'), taxRate:
20 },
    { description: 'Books', quantity: 1, unitPrice: new Decimal('2500.0000'), taxRate: 0 }, //
zero-rated
  ]

  const totals = calculateInvoiceTotals(items)

  expect(totals.subtotal.toString()).toBe('52500.0000')
  expect(totals.taxAmount.toString()).toBe('10000.0000') // Only first item taxed
  expect(totals.totalAmount.toString()).toBe('62500.0000')
  expect(totals.items[0].taxAmount.toString()).toBe('10000.0000')
  expect(totals.items[1].taxAmount.toString()).toBe('0.0000')
})
```

E2E Test Case Template

```
// File: apps/e2e/tests/<flow>.spec.ts

import { test, expect } from '@playwright/test'

/**
 * TC-ID: E2E-XXX
 * Flow: <flow name>
 * Priority: P0 / P1
```

```
 * Author: <developer name>
 * Date: YYYY-MM-DD
 */
test.describe('<Flow Name>', () => {
  test.beforeEach(async ({ page }) => {
    // Login with test user
    await page.goto('/login')
    await page.fill('input[name="email"]', 'demo@bilko.io')
    await page.fill('input[name="password"]', 'Demo123!')
    await page.click('button[type="submit"]')
    await expect(page).toHaveURL('/dashboard')
  })

  test('<scenario description>', async ({ page }) => {
    // Navigate
    await page.goto('/<route>')

    // Interact
    await page.click('<selector>')
    await page.fill('<selector>', '<value>')

    // Assert
    await expect(page.locator('<selector>')).toContainText('<expected>')
    await expect(page).toHaveURL(/<url-pattern>/)
  })
})
```

Test Case Register

Track all test cases in `TEST-INVENTORY.md`. Each test case needs:

Field	Description
ID	`UNIT-XXX`, `INT-XXX`, `ACCT-XXX`, `E2E-XXX`
Title	Brief description
Priority	P0 / P1 / P2
Type	Unit / Integration / E2E
File	Relative path to test file

Field	Description
Status	Not implemented / Pass / Fail / Skipped
Financial logic	Yes / No (if Yes, requires NUMERIC precision assertion)

Related Documents

- Test Strategy
- Test Plan
- TEST-INVENTORY.md
- TESTING-GUIDE.md

Approval

Role	Name	Date	Signature
Author	Ops Architect	2026-02-23	
Reviewer	Tech Lead		
Approver	Alem Bašić		

Revision #14
Created 2026-02-24 22:50:56 UTC by John
Updated 2026-06-07 19:43:43 UTC by John