Test Planning Document for Apartment Complex Account Management System

Introduction

Purpose

This document ensures all functional and non-functional requirements are validated across all architecture layers (PostgreSQL/Python Flask/ReactJS) of the Apartment Complex Account Management System, with alignment to business requirements, testing strategy, and RTM.

Scope

In-scope:

- API contract testing for all Flask endpoints
- PostgreSQL schema validation and data integrity tests
- UI component testing and user journey validation
- Cross-layer integration (DB→API→UI workflow validation)
- Performance testing up to 10k concurrent users

Out-of-scope:

- Third-party API reliability testing (bank integrations)
- Load testing beyond 10k concurrent users
- Penetration testing (covered separately in security audit)

References

- LLD Document [Version 1.2] (Chapter5-LLD.pdf)
- RTM v2.1 with traceability IDs (Chapter5-User Story and RTM-Gemini.pdf)
- Infrastructure Topology Diagram (AWS 3-tier architecture)
- Test Strategy Document (Chapter-6 Test Strategy Document.pdf)

Test Strategy by Layer

Database (PostgreSQL)

Coverage:

- Schema validation against LLD specifications
- Constraint testing (CHECK, NOT NULL, UNIQUE)
- Transaction isolation level verification
- Balance history table integrity
- Index performance validation

Tools:

- pgTAP (unit testing with PostgreSQL-specific assertions)

- SQLAlchemy for ORM validation
- EXPLAIN ANALYZE for query optimization
- Factory Boy for test data generation

Test Cases:

1. Verify account type ENUM constraint ('INCOME','EXPENSE','ASSET','LIABILITY')
2. Validate transaction amount CHECK (amount > 0)
3. Test account-contra_account inequality constraint
4. Verify balance_history unique constraint (account_id + balance_date)

Middleware (Python Flask)

Coverage:

- API contract compliance (FastAPI endpoints)
- Business logic validation (transaction processing)
- Error handling (400/402/403/404 responses)
- Authentication/authorization flows
- Concurrent transaction handling

Tools:

- PyTest + Flask-Testing
- Postman/Newman (contract validation)
- Requests library for API calls
- Faker for test data generation

Test Cases:

1. POST /accounts - Validate account creation with all required fields
2. POST /transactions - Verify insufficient funds error (402)
3. GET /accounts/{id} - Test JWT authentication requirement
4. Concurrent transaction stress test (10 threads)

UI (ReactJS)

Coverage:

- Component rendering (Account forms, Transaction tables)
- User journeys (Create Account → Record Transaction → View Report)
- Accessibility (WCAG 2.1 AA compliance)
- State management (Redux store validation)
- Cross-browser compatibility

Tools:

- Playwright (cross-browser E2E testing)
- Jest (unit testing)

- React Testing Library (component tests)
- Axe-core (accessibility audits)

Test Cases:

1. Account creation form validation messages
2. Transaction list sorting (date descending default)
3. Report generation loading indicator (<2s)
4. Mobile responsive layout checks

Test Objectives

Success Metrics

- 100% RTM coverage (all BR-00x requirements validated)
- <2% critical defect leakage to production
- 85%+ automation coverage for regression suite
- 95% pass rate for P0/P1 test cases

Priority Framework

Level	Criteria	Test Allocation	Example Test Cases
P0	Core transaction flows	40% resources	US-004 (Record Transaction), US-005 (Prevent Overdrafts)
P1	Secondary features	30% resources	US-003 (View Accounts), US-006 (Account Status Report)
P2	Edge cases	20% resources	Future-dated transactions, optional field handling
P3	Non-functional	10% resources	GDPR logging, 2s report loading

Tool Selection Matrix

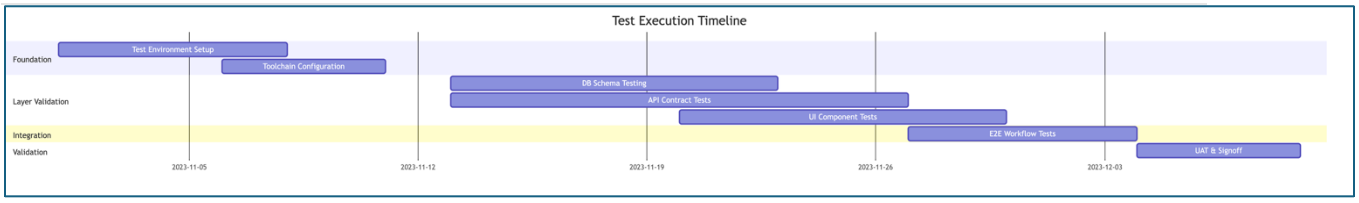
Layer	Recommended Tool	Cost	Rationale
UI E2E	Playwright	Free	Superior React support with component testing capability

Layer	Recommended Tool	Cost	Rationale
API Testing	PyTest + Requests	Free	Native Python stack integration
DB Unit	pgTAP	Free	PostgreSQL-specific assertions
Test Data	Factory Boy	Free	Python-compatible data generation
CI/CD	GitHub Actions	Free	Tight integration with code repositories

Total estimated tooling cost: $0 (fully open-source stack)

Risk Management

Risk	Probability	Impact	Mitigation Strategy
Flask API version drift	Medium	High	Swagger contract tests in CI pipeline
React state management issues	High	Medium	Redux middleware unit tests
DB performance bottlenecks	Low	Critical	EXPLAIN ANALYZE in all test cases
Test data inconsistencies	High	Medium	Factory Boy data factories with seed control



Parallel tracks:

- Unit test development (Weeks 1-4)
- Integration test creation (Weeks 3-6)
- Manual exploratory testing (Week 7)

Traceability Matrix Snippet

ReqID	User Story	Test Case ID	Test Type	Target Layer	Automation
BR-001	US-001	TC-301	Unit, E2E	DB, UI	Yes (Playwright)
BR-004	US-004	TC-304	Integration	API	Yes (PyTest)
BR-005	US-005	TC-305	Unit	DB	Yes (pgTAP)
BR-007	US-007	TC-307	System	UI	No (Manual)

Full RTM attached as Appendix A

Playwright Implementation Notes

For React component testing:
javascript
```javascript
// Example Playwright component test for AccountForm
test('renders account type dropdown', async ({ mount }) => {
  const component = await mount(<AccountForm />);
  await expect(component.getByLabel('Account Type')).toBeVisible();
  await expect(component.getByText('Income')).toBeVisible();
});

// Cross-browser E2E test
test('create account flow', async ({ page }) => {
  await page.goto('/accounts/new');
  await page.fill('#accountName', 'Test Account');
  await page.selectOption('#accountType', 'EXPENSE');
  await page.click('button[type="submit"]');
  await expect(page).toHaveURL(/\/accounts\/[a-f0-9-]+$/);
});
```

Budget Compliance

All recommended tools are open-source with no licensing costs. Budget allocation:

- $300 for test data generation services
- $200 for cloud testing infrastructure
  Total: $500/year

Approval:
Test Lead: ___________________
Date: ___________________