Test Strategy

In real-life test strategy the first step would be to understand the requirements. Since I don't have detailed technical or business requirements for this assignment, I'm using example numbers and targets based on common fintech industry standards and reasonable assumptions.

1. Quality Vision & Objectives

Quality goals for a fintech company:

1. **Accuracy** (every transaction is correct, no miscalculation, no data loss, no duplication)
2. **Requirement alignment** (meets product, business, and regulatory requirements)
3. **Reliability** (aim for zero downtime, possible failures are isolated and critical flows keep running)
4. **Security** (no critical vulnerabilities, user data protection, full auditability)
5. **User experience** (intuitive and stress-free user experience)
6. **Scalability and Performance** (ability to handle massive volumes and user growth without compromising on speed or stability)

Key Quality Metrics & KPIs:

1. Accuracy
- Transaction success rate > 99.99%
- Ledger reconciliation errors: 0
- Duplicate/lost events: 0

2. Requirement Alignment
- 100% requirements coverage for business-critical stories
- Regulatory compliance defects = 0 in production

3. Reliability
- API Gateway uptime 99.99%, core services 99.95%
- 5xx error rate <= 0.1%
- MTTR < 30 min for P1 incidents

4. Security
- 0 critical/high vulnerabilities in prod
- High-severity patching <= 14 days

5. User Experience
- Crash-free sessions >= 99.8%
- Support tickets <= 0.3 per 1000 transactions

6. Scalability & Performance
- p95 response times: Login <= 1 s, Transfers <= 2 s
- Peak x1.5 load sustained for >= 30 min without SLO breach

2. Test Pyramid Strategy

E2E tests (including UI and API) - 10%
- The slowest to create and the most expensive to maintain
- Prone to flakiness
- Still an important part of the pyramid since they provide confidence that the entire system works together
- Typically run in late stages to ensure readiness for deployment

Integration - 30%
- Crucial for microservices
- Generally faster than E2E tests but more complex than unit tests

Unit tests - 60%
- Catch most regressions early and give fast CI feedback
- Isolate failures
- Fastest and cheapest to write and maintain

3. CI/CD Quality Gates

Lint & Unit Tests (PR checks)

Pass Criteria:

- 100 % of Unit tests pass
- Coverage >= 80%
- No critical/high vulnerabilities

Integration & API Contracts

Pass Criteria:

- All Integration tests pass
- API compatibility verified

Security Checks

Pass Criteria:

- No critical/high vulnerabilities
- Basic PCI hygiene for payment paths

Smoke / E2E Tests

Pass Criteria:

- All smoke/E2E test suites pass

Performance Check

Pass Criteria:

- Latency: p95 < 500 ms (critical services), p95 < 1 s (supporting).
- Capacity: Handles expected peak load + 50% for >= 30 min.
- Stability: 5xx error rate <= 0.1% during the run

Any stage failure blocks deployment and triggers alerts. Manual override only for critical fixes with approval.

4. Non-Functional Testing

Performance Benchmarks:

System Availability: close to 99.9% uptime per service. If one microservice goes down, it should not affect the entire system.
Response Time: 95% of login and balance requests should complete in about 1 second or less, and transfers should take under 2 seconds.
Error Rate: Less than 0.5% of all requests should result in errors. Errors should trigger alerts.
Load Handling: System is expected to handle regular peak traffic with at least a 50% buffer for unexpected surges
Recovery: should take no more than 10-20 minutes to recover service without loss of any progressing transactions.

Types of performance testing:

Load testing: verify performance under normal and peak conditions against defined benchmarks.
Stress Testing: Determine the point at which performance starts to degrade.
Soak Testing: Identify memory leaks or slow performance over extended usage.