FR5 Swing Migration Technical Design Document (TDD)

Version: 1.0 | Audience: Architecture Review + Developer Handover | Owner: [Your Name / Team]

# 1. Purpose & Scope

This TDD presents a blended view for architecture review and developer handover. It outlines the target solution for modernising a legacy Java Swing application into JavaFX or Web (Vue.js), integrated with FR5 services (.NET APIs and Node.js integrations). It includes simplified pseudocode, Lucid diagram placeholders, and configuration appendices following FR5 naming conventions.

# 2. Architecture Overview

Target runtime offers two delivery modes:  
• JavaFX Desktop Variant (MVVM) consuming FR5 REST APIs  
• Web Variant (Vue.js SPA) consuming FR5 REST/GraphQL APIs; Node.js for async/integration  
  
Key layers (FR5-aligned):  
• FR5.Web.\* → Vue.js SPA modules and UI composition  
• FR5.Services.\* → .NET API surface (controllers, DTOs, domain services)  
• FR5.Integration.\* → Node.js middleware (adapters, schedulers, streams)  
• Shared → Auth (OIDC), Observability, CI/CD, Compliance

[Insert Lucid Diagram: System Architecture Map — import CSV from lucid/web\_target\_edges.csv]

# 3. Component Model

3.1 UI Components

FR5.Web.Customers  
• Pages: CustomerList, CustomerDetail  
• Components: CustomerForm, AddressPanel, BusyIndicator  
• Store: customersStore (state, actions, effects)  
• Navigation: role-based routes

3.2 Service Components (.NET)

FR5.Services.Customers  
• Controllers: CustomersController  
• Services: CustomerUseCases, ValidationService  
• Ports: CustomerRepository (interface)  
• Adapters: CustomerRepositoryEf (EF Core), ReportingClient

3.3 Integration Components (Node.js)

FR5.Integration.Messaging  
• APIs: EventBridgeController (optional)  
• Workers: AddressUpdateWorker  
• Adapters: KafkaAdapter / ServiceBusAdapter

# 4. Key Interaction Flows

4.1 Fetch Customer (Web Variant)

Browser → Customers API → UseCases → Repository → DB  
Pseudocode:  
UI.clickRow(id):  
 api.get('/customers/{id}') -> setState(customer)  
API.get(id):  
 return svc.getCustomer(id)  
UseCases.getCustomer(id):  
 return repo.findById(id)

[Insert Lucid Diagram: API Interaction Flow — import CSV from lucid/ports\_adapters\_edges.csv]

4.2 Update Address (Desktop JavaFX Variant)

ViewModel.saveAddress():  
 if not validator.isValid(addr): showError()  
 else api.put('/customers/{id}/address', addr) -> notify('Saved')

[Insert Lucid Diagram: JavaFX MVVM Binding — import CSV from lucid/javafx\_mvvm\_edges.csv]

# 5. Security & Access Control

• Authentication: OIDC (Azure AD) — PKCE for SPA; device code/embedded browser for desktop.  
• Authorization: RBAC via roles/claims; policy checks on API endpoints.  
• Token Handling: short-lived access tokens; refresh tokens where applicable.  
• Audit: write immutable events for sensitive operations.

[Insert Lucid Diagram: AuthN/AuthZ Flow — import CSV from lucid/auth\_flow\_edges.csv]

# 6. Data & Contracts

6.1 Domain & DTOs (simplified)

Domain:  
 Customer { id, name, address }  
 Address { street }  
  
DTOs:  
 CustomerDto { id, name, street }  
 AddressDto { street }

6.2 API Endpoints (OpenAPI summary)

GET /api/customers/{id} → 200 CustomerDto  
PUT /api/customers/{id}/address { AddressDto } → 200 CustomerDto  
Errors: 400 validation, 401 unauth, 403 forbidden, 409 conflict

# 7. Observability

• Logs: structured JSON (API, Node.js, client errors)  
• Metrics: request latency, error rates, queue depths  
• Traces: API to DB spans; correlation id propagated from UI

# 8. DevOps & CI/CD

8.1 Pipelines and Stages (FR5 Naming)

Azure DevOps (for .NET & JavaFX):  
 Pipeline: FR5.Services.Customers-CI → stages: build, test, package, deploy  
GitHub Actions (for Vue.js & Node.js):  
 Workflow: fr5-web-customers-ci.yml → jobs: build, test, dockerize, deploy

8.2 Azure DevOps (YAML pseudocode)

stages:  
- stage: Build  
 jobs:  
 - job: BuildApi  
 steps:  
 - task: DotNetCoreCLI@2 # build  
 - task: DotNetCoreCLI@2 # test  
- stage: Package  
 jobs:  
 - job: Dockerize  
 steps:  
 - script: docker build -t fr5-services-customers:${{ Build.BuildNumber }} .  
- stage: Deploy  
 jobs:  
 - job: DeployToAzure  
 steps:  
 - task: AzureWebAppContainer@1 # deploy container

8.3 GitHub Actions (YAML pseudocode)

name: fr5-web-customers-ci  
on: [push]  
jobs:  
 build\_test:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - uses: actions/setup-node@v4  
 with:  
 node-version: '20.x'  
 - run: npm ci && npm run build && npm test  
 dockerize\_deploy:  
 needs: build\_test  
 runs-on: ubuntu-latest  
 steps:  
 - run: docker build -t fr5-web-customers:${{ github.run\_number }} .  
 - run: echo 'deploy step (placeholder)'

# 9. Deployment & Environments

• Dev → Test → UAT → Prod with approvals.  
• Blue/Green for web; signed installers for desktop.  
• Feature flags for gradual rollout.

[Insert Lucid Diagram: Delivery Pipelines — import CSV from lucid/delivery\_pipelines\_edges.csv]

# 10. Risks & Mitigations

R1 Component parity gaps → Spike replacements, maintain parity list.  
R2 Accessibility regressions → Automated audits + manual testing.  
R3 Pipeline flakiness → Retry policies, isolated test envs, cache restores.  
R4 API contract drift → Contract tests pre-deploy.

# Appendix A — Repository & Structure (FR5 Conventions)

Repos:  
• FR5.Services.Customers (C#)  
• FR5.Web.Customers (Vue.js)  
• FR5.Integration.Messaging (Node.js)  
  
Folders (example):  
FR5.Services.Customers/  
 src/, tests/, deploy/, docs/  
FR5.Web.Customers/  
 src/components/, src/pages/, src/store/, tests/, deploy/  
FR5.Integration.Messaging/  
 src/adapters/, src/workers/, tests/, deploy/

# Appendix B — Configuration & Secrets (Simplified)

appsettings.json (API):  
{  
 "Jwt": { "Authority": "https://login.microsoftonline.com/...", "Audience": "fr5-services" },  
 "ConnectionStrings": { "Main": "..." }  
}  
  
.env (Node.js):  
OIDC\_AUTHORITY=...  
SERVICEBUS\_CONN=...  
  
Vue env (.env):  
VITE\_API\_BASE=https://api.fr5.local  
VITE\_AUTH\_CLIENT\_ID=...

# Appendix C — Test Strategy (High Level)

• Unit tests: domain and utilities  
• Contract tests: API endpoints vs OpenAPI  
• UI automation: critical flows (Cypress/Playwright)  
• Performance: API latency SLA checks  
• Security: dependency scanning, SAST