FR5 Swing Migration Functional Design Document (FDD)

# Purpose

This document defines the functional behavior, modernization approach, and integration design for migrating a legacy Java Swing application into a JavaFX or Web-based architecture, aligned with the FR5 platform ecosystem (.NET, Vue.js, Node.js). It also outlines DevOps tooling, API interfaces, and usability standards consistent with FR5’s user-centric web applications.

# Scope

In Scope:  
• Migration and modularization of Swing UI into modern web or desktop technologies.  
• Integration with FR5 backend services built on .NET APIs and Node.js microservices.  
• Adoption of Vue.js for responsive and user-centric front-end modules.  
• CI/CD enablement using Azure DevOps pipelines and GitHub Actions.  
• Implementation of usability and accessibility best practices.  
  
Out of Scope:  
• Full FR5 platform redesign.  
• Backend infrastructure overhaul beyond existing service contracts.

# System Overview

The modernization supports two target architectures — JavaFX desktop and Vue.js web — both leveraging FR5’s .NET APIs and Node.js services.  
  
JavaFX Variant:  
• Modular MVVM pattern with REST integration.  
• Gradle + Azure DevOps build and deploy pipelines.  
  
Web Variant:  
• Vue.js SPA consuming FR5’s .NET REST APIs.  
• Node.js microservices for async and integration logic.  
• Containerized via Docker, deployed to Azure App Services.

# Functional Overview

Integration across layers:  
• Presentation: Vue.js (responsive UX with FR5 components)  
• Service: .NET and Node.js backends  
• Tooling: Azure DevOps and GitHub Actions pipelines for CI/CD  
• Infrastructure: Azure-hosted containers, integrated with FR5’s monitoring stack.

# Functional Modules

UI Modernization:  
• Replace Swing forms with Vue.js pages or JavaFX FXML views.  
• Align layout and theme with FR5 Design System.  
  
Service Refactor:  
• Reuse FR5 .NET microservices and add Node.js middleware APIs.  
• Document APIs via Swagger/OpenAPI.  
  
DevOps & Tooling:  
• Source control via Git.  
• Pipelines through Azure DevOps and GitHub Actions.  
• Automated build, test, deploy with semantic versioning.  
  
Lucid Diagram Placeholders:  
[Insert Diagram: Strangler Fig Migration]  
[Insert Diagram: Ports & Adapters View]  
[Insert Diagram: Delivery Pipelines]

# User-Centric Design

FR5 UX patterns ensure accessibility, clarity, and responsiveness.  
Regular feedback loops and heuristic evaluations are integrated in each sprint.

# Non-Functional Requirements

• Performance: <2s render; <500ms API latency.  
• Accessibility: WCAG 2.1 AA.  
• Security: OAuth2/OIDC; HTTPS-only.  
• DevOps: Azure DevOps + GitHub Actions automation.  
• Usability: Responsive and consistent FR5 UX.

# Deployment Overview

Frontend (Vue.js): npm + GitHub Actions → Docker image → Azure App Service.  
Backend (.NET): Azure DevOps + MSBuild → API containers.  
Node.js: GitHub Actions → Docker → Azure Functions or Kubernetes.  
JavaFX: Gradle + Azure Pipeline → MSI/DMG installers.  
  
Lucid Diagram Placeholder: [Insert Delivery Pipeline Diagram]

# Security & Compliance

OIDC authentication via Azure AD with RBAC and token rotation.  
Signed artifacts and dependency scanning integrated in CI/CD.  
Lucid Diagram Placeholder: [Insert Auth Flow Diagram]

# Risk & Mitigation

• API schema drift → Contract testing.  
• UX inconsistency → FR5 component library.  
• Build failures → Multi-branch CI testing.  
• Accessibility regressions → Automated axe-core CI audits.