Product Requirements Document

Java Swing Multi-DPI and Multi-Monitor Scaling Support

Version 1.0 | January 2026

Field	Value
Document Owner	[Your Name]
Last Updated	January 28, 2026
Status	Draft
Target Release	[TBD]

Table of Contents

1. Executive Summary

1.1 Purpose

This document defines the requirements for implementing comprehensive DPI-awareness and multi-monitor scaling support in an existing Java Swing application. The application was originally developed for a 16:9 aspect ratio display at 96 DPI and must be enhanced to render correctly across diverse display configurations including high-DPI (HiDPI/Retina) displays, ultra-wide monitors, and multi-monitor setups with varying DPI values.

1.2 Scope

The scope encompasses modifications to the existing application architecture including layout management, icon rendering, font scaling, and component sizing. The solution must maintain backward compatibility with the current 96 DPI baseline while introducing automatic scaling for other configurations.

1.3 Goals

- Enable automatic UI scaling based on system DPI settings without manual configuration
- Maintain visual fidelity and usability across displays ranging from 96 DPI to 288 DPI (300% scaling)
- Support aspect ratios from 4:3 to 32:9 (ultra-wide) without layout degradation
- Preserve existing application functionality and user workflows
- Minimize performance impact from scaling operations

1.4 Success Metrics

Metric	Target	Measurement Method
UI renders correctly at all supported DPI levels	100% of screens	Visual inspection across test configurations
No pixel-level rendering artifacts	Zero defects	Automated screenshot comparison
Icons remain crisp at all scale factors	No visible blur	Manual QA on 4K displays
Font readability maintained	100% legible	User testing with accessibility guidelines
Application startup time impact	< 5% increase	Automated performance benchmarks
Memory footprint increase	< 15%	Profiling tools measurement

2. Current State Analysis

2.1 Existing Architecture

The application currently operates under the following constraints and characteristics:

Component	Current Implementation	Scaling Challenge
Layout Manager	MigLayout with percent-based constraints	Percent constraints scale well; pixel-based gaps/insets do not
Panel Management	CardLayout for component swapping	Cached panel dimensions may become stale on DPI change
Panel Lifecycle	Created at startup, reused throughout	Pre-calculated sizes incompatible with dynamic scaling
Icons	Hardcoded pixel dimensions	Appear too small on HiDPI or blurry when stretched
Fonts	Hardcoded point sizes	May appear too small or large depending on DPI
Component Sizing	Mix of absolute and relative	Absolute sizes break on non-96 DPI displays

2.2 MigLayout Specific Considerations

MigLayout constraints fall into two categories regarding DPI sensitivity:

2.2.1 DPI-Resilient Constraints (No Changes Required)

- Percentage-based widths and heights (e.g., "width 50%", "height 33%")
- Grow and shrink priorities (e.g., "grow", "shrink 0")
- Fill constraints (e.g., "fill", "fillx", "filly")
- Relative positioning (e.g., "push", "pushy")

2.2.2 DPI-Sensitive Constraints (Require Scaling)

- Pixel-based gaps (e.g., "gap 10px", "gapx 5")
- Absolute sizes (e.g., "width 200!", "height 150")
- Minimum/maximum pixel constraints (e.g., "min 100", "max 500")
- Insets specified in pixels (e.g., "insets 10 10 10 10")

2.3 CardLayout Implications

CardLayout containers present unique challenges because all cards are laid out based on the maximum preferred size among all contained components. When panels are pre-created at application startup:

- Initial sizing calculations occur before DPI may be fully determined
- Panels cached in non-visible state may not receive size update notifications
- DPI changes during runtime require explicit invalidation of all cards
- The container must be re-validated after any card is resized

3. Requirements Overview

3.1 Requirement Categories

Category	Priority	Description
Core Scaling Infrastructure	P0 - Critical	Foundation for all scaling operations
Icon Scaling System	P0 - Critical	Multi-resolution icon support
Font Scaling System	P0 - Critical	DPI-aware font rendering
Layout Scaling	P1 - High	MigLayout constraint adaptation
Component Scaling	P1 - High	Individual component size management
Multi-Monitor Support	P2 - Medium	Handle displays with different DPIs
Runtime DPI Changes	P2 - Medium	Respond to DPI changes without restart
Performance Optimization	P3 - Low	Caching and lazy loading strategies

3.2 Supported Configurations

Configuration	Minimum	Maximum	Notes
DPI	72	288	Corresponds to 75% to 300% Windows scaling
Scale Factor	0.75x	3.0x	Relative to 96 DPI baseline
Aspect Ratio	4:3	32:9	Standard to ultra-wide
Resolution Width	1024	7680	Minimum usable to 8K
Resolution Height	768	4320	Minimum usable to 8K

4. Functional Requirements

4.1 FR-001: Scale Factor Detection

Attribute	Specification
ID	FR-001
Priority	P0 - Critical
Description	The system shall automatically detect the current display scale factor on application startup and provide this value to all scaling-dependent components.
Acceptance Criteria	Scale factor is correctly identified within 100ms of application initialization for Windows 10/11, macOS 12+, and Linux (GNOME/KDE) environments.
Dependencies	None

4.1.1 Detection Methods by Platform

The implementation must support multiple detection strategies:

- Windows: Query AffineTransform from GraphicsConfiguration, check GDI scaling via sun.java2d.uiScale system property
- macOS: Use GraphicsDevice.getDefaultConfiguration().getDefaultTransform() for Retina detection
- Linux: Check GDK_SCALE environment variable, query Xrandr for per-monitor DPI, respect GNOME/KDE scaling settings
- Fallback: Use Toolkit.getDefaultToolkit().getScreenResolution() divided by 96 as baseline

4.2 FR-002: Centralized Scale Manager

Attribute	Specification
ID	FR-002
Priority	P0 - Critical
Description	A singleton ScaleManager class shall provide centralized access to scaling calculations, cache scaled values, and notify registered listeners of scale changes.
Acceptance Criteria	All scaling operations route through ScaleManager; no direct DPI calculations elsewhere in codebase.
Dependencies	FR-001

4.2.1 ScaleManager Interface Requirements

The ScaleManager shall expose the following capabilities:

- getScaleFactor(): Returns current scale factor as double (1.0 = 96 DPI baseline)
- scale(int pixels): Returns scaled pixel value for current DPI
- scale(Dimension dim): Returns new Dimension with both width and height scaled
- scale(Insets insets): Returns new Insets with all values scaled
- scaleFont(Font font): Returns font with point size adjusted for current DPI
- addScaleChangeListener(ScaleChangeListener): Register for scale change notifications
- removeScaleChangeListener(ScaleChangeListener): Unregister listener
- invalidateCache(): Force recalculation of all cached scaled values

4.3 FR-003: Multi-Resolution Icon System

Attribute	Specification
ID	FR-003
Priority	P0 - Critical
Description	The system shall support multi-resolution icons that automatically select the appropriate resolution variant based on the current scale factor.
Acceptance Criteria	Icons appear crisp (no visible blur or pixelation) at 100%, 125%, 150%, 175%, 200%, and 300% scaling.
Dependencies	FR-002

4.3.1 Icon Resolution Requirements

Base Size	1x (96 DPI)	1.5x (144 DPI)	2x (192 DPI)	3x (288 DPI)
16x16	16x16	24x24	32x32	48x48
24x24	24x24	36x36	48x48	72x72
32x32	32x32	48x48	64x64	96x96
48x48	48x48	72x72	96x96	144x144
64x64	64x64	96x96	128x128	192x192

4.3.2 Icon Loading Strategy

- Primary: Load exact resolution match if available (e.g., icon_24@2x.png for 2x scale)
- Secondary: Load next higher resolution and downscale with high-quality interpolation
- Tertiary: Load highest available and downscale (may result in some quality loss)
- Emergency: Use base resolution with smoothing (acceptable only for legacy icons)
- Vector Alternative: Support SVG icons rendered at exact required resolution

4.3.3 Icon Naming Convention

Icons shall follow this naming convention for automatic resolution detection:

- Base resolution: iconname.png (e.g., save.png)
- 1.5x resolution: iconname@1.5x.png (e.g., save@1.5x.png)
- 2x resolution: iconname@2x.png (e.g., save@2x.png)
- 3x resolution: iconname@3x.png (e.g., save@3x.png)
- SVG alternative: iconname.svg (e.g., save.svg) - preferred when available

4.4 FR-004: Scalable Font System

Attribute	Specification
ID	FR-004
Priority	P0 - Critical
Description	The system shall provide a centralized font management system that delivers consistently scaled fonts based on logical font sizes and current DPI.
Acceptance Criteria	Text is readable and proportionally correct at all supported DPI levels; no manual font size adjustments required by users.
Dependencies	FR-002

4.4.1 Font Size Strategy

Define logical font sizes that map to physical point sizes based on scale factor:

Logical Size	Purpose	96 DPI (pts)	144 DPI (pts)	192 DPI (pts)
TINY	Tooltips, minor labels	9	9	9
SMALL	Secondary text, captions	10	10	10
NORMAL	Body text, form fields	12	12	12
MEDIUM	Emphasized text	14	14	14
LARGE	Section headers	16	16	16
XLARGE	Dialog titles	18	18	18
HUGE	Main titles	24	24	24

Note: Java's font rendering is inherently DPI-aware when using point sizes. The scaling concern is ensuring that hardcoded pixel-based calculations (line heights, component sizing based on font metrics) are also scaled appropriately.

4.4.2 Font Manager Interface

- getFont(LogicalSize size): Returns Font at specified logical size for current DPI
- getFont(LogicalSize size, int style): Returns Font with style (Font.BOLD, Font.ITALIC)
- getFont(String family, LogicalSize size, int style): Returns Font with specified family
- getBoldFont(LogicalSize size): Convenience method for bold fonts
- getItalicFont(LogicalSize size): Convenience method for italic fonts
- getMonospacedFont(LogicalSize size): Returns monospaced font for code/data display

4.5 FR-005: MigLayout Constraint Scaling

Attribute	Specification
ID	FR-005
Priority	P1 - High
Description	The system shall provide utilities to convert pixel-based MigLayout constraints to scaled equivalents while preserving relative spacing relationships.
Acceptance Criteria	Existing layouts using pixel-based gaps and insets render proportionally at all scale factors.
Dependencies	FR-002

4.5.1 Constraint Transformation Approach

Two implementation strategies shall be supported:

Strategy A - Runtime Constraint Modification:

- Parse existing constraint strings at panel creation time
- Identify pixel-based values using regex patterns
- Replace with scaled equivalents (e.g., "gap 10" becomes "gap 15" at 1.5x)
- Cache transformed constraints to avoid repeated parsing

Strategy B - Logical Unit Abstraction:

- Define custom unit type (e.g., "lu" for logical units)
- Create constraint builder API that accepts logical values
- Generate appropriate pixel values at constraint creation time
- Example: MigConstraints.gap(10) produces "gap 15" at 1.5x scale

4.5.2 Common MigLayout Patterns to Scale

Pattern	Example (96 DPI)	Example (192 DPI)	Notes
Gap	gap 10	gap 20	All gap variants: gapx, gapy, gaptop, etc.

Insets	insets 5 10 5 10	insets 10 20 10 20	Layout-level margins
Min Width	width 100:200:300	width 200:400:600	min:preferred:max
Fixed Size	width 150!	width 300!	Forced size constraints
Padding	pad 5 5 5 5	pad 10 10 10 10	Component padding

4.6 FR-006: CardLayout Panel Management

Attribute	Specification
ID	FR-006
Priority	P1 - High
Description	The system shall ensure CardLayout containers properly handle scale changes for all contained cards, including those not currently visible.
Acceptance Criteria	Switching cards after a DPI change shows correctly sized content without visual artifacts or layout exceptions.
Dependencies	FR-002, FR-005

4.6.1 CardLayout Scale Change Protocol

When a scale change is detected:

- CardLayout container receives scale change notification via ScaleChangeListener
- Container iterates through all contained cards (visible and hidden)
- Each card's constraints are updated via FR-005 scaling utilities
- Each card is invalidated to clear cached layout information
- Container calls revalidate() followed by repaint()
- Currently visible card re-renders at new scale immediately
- Hidden cards will render correctly when next shown

4.6.2 Lazy Scaling Option

For performance optimization, implement optional lazy scaling:

- On scale change, mark all hidden cards as "scale-dirty"
- Only actively scale the currently visible card immediately
- Scale hidden cards just-in-time when they become visible
- Track scale factor at time of last layout to detect staleness
- Configuration option to force immediate scaling for critical panels

4.7 FR-007: Multi-Monitor DPI Handling

Attribute	Specification
ID	FR-007
Priority	P2 - Medium
Description	The system shall detect when the application window moves between monitors with different DPI values and adjust scaling accordingly.
Acceptance Criteria	Dragging application window from 96 DPI monitor to 192 DPI monitor results in correct re-scaling within 500ms of drop.
Dependencies	FR-002, FR-006

4.7.1 Monitor Change Detection

- Register ComponentListener on main JFrame to detect location changes
- Query GraphicsConfiguration when window location changes significantly
- Compare new GraphicsConfiguration DPI to cached value
- Trigger scale change sequence if DPI differs
- Debounce rapid location changes during window dragging

4.7.2 Window Spanning Considerations

When window spans multiple monitors with different DPIs:

- Use DPI of monitor containing majority of window area (>50%)
- Alternative: Use DPI of monitor containing window title bar
- Do not attempt split-DPI rendering (not supported by Swing)
- Document this limitation in user-facing materials

4.8 FR-008: Runtime DPI Change Response

Attribute	Specification
ID	FR-008
Priority	P2 - Medium
Description	The system shall respond to operating system DPI changes without requiring application restart.
Acceptance Criteria	Changing Windows display scaling from 100% to 200% while application is running results in correct re-rendering.
Dependencies	FR-002, FR-003, FR-004, FR-005, FR-006

4.8.1 DPI Change Event Sequence

- OS notifies JVM of display change (platform-specific mechanism)
- PropertyChangeListener on Toolkit detects desktopProperty change
- ScaleManager recalculates and caches new scale factor
- ScaleManager invalidates all cached scaled values
- ScaleManager notifies all registered ScaleChangeListeners
- Each listener updates its associated component(s)
- Main window revalidates and repaints entire component hierarchy

4.8.2 Platform-Specific Detection

Platform	Detection Mechanism	Notes
Windows 10/11	PropertyChangeListener on "win.defaultDPI"	May require JVM restart for some changes
macOS	Automatic via GraphicsConfiguration	Retina switching is seamless
Linux/GNOME	Monitor GSettings changes	May require polling
Linux/KDE	Monitor kwinrc changes	May require polling

5. Technical Specifications

5.1 Architecture Overview

The scaling system follows a centralized observer pattern with the following key components:

Component	Responsibility	Design Pattern
ScaleManager	Central scaling calculations, listener management	Singleton, Observer
ScaleChangeListener	Interface for components needing scale updates	Observer
ScalableIcon	Multi-resolution icon wrapper	Strategy, Lazy Loading
FontManager	Centralized font provisioning	Singleton, Factory
MigConstraintScaler	Constraint string transformation	Utility/Static
ScalablePanel	Base class for DPI-aware panels	Template Method

5.2 Class Specifications

5.2.1 ScaleManager

Purpose: Singleton providing centralized access to all scaling operations.

Key Methods:

- getInstance(): ScaleManager - Returns singleton instance
- getScaleFactor(): double - Returns current scale factor (1.0 = 96 DPI)
- scale(int value): int - Scales integer value by current factor
- scale(double value): double - Scales double value by current factor
- scale(Dimension dim): Dimension - Returns new scaled Dimension
- scale(Insets insets): Insets - Returns new scaled Insets
- scale(Rectangle rect): Rectangle - Returns new scaled Rectangle
- unscale(int value): int - Reverses scaling for coordinate translation
- addScaleChangeListener(ScaleChangeListener l): void - Register listener
- removeScaleChangeListener(ScaleChangeListener l): void - Unregister
- fireScaleChanged(): void - Notify all listeners of scale change

Implementation Notes:

- Thread-safe singleton initialization using holder pattern
- Weak references for listeners to prevent memory leaks
- Cached scaled values for common sizes (invalidated on scale change)
- Configurable base DPI (default 96)

5.2.2 ScaleChangeListener

Purpose: Callback interface for components needing scale change notification.

Interface Definition:

- onScaleChanged(double oldScale, double newScale): void - Called when DPI changes
- Default implementation: component.revalidate(); component.repaint();

5.2.3 ScalableIcon

Purpose: Icon implementation that automatically selects appropriate resolution.

Key Methods:

- ScalableIcon(String basePath): Constructor loads all available resolutions
- ScalableIcon(String basePath, int baseWidth, int baseHeight): With explicit base size
- getIconWidth(): int - Returns scaled width for current DPI
- getIconHeight(): int - Returns scaled height for current DPI
- paintIcon(Component c, Graphics g, int x, int y): void - Renders appropriate resolution

Resolution Selection Algorithm:

- Calculate target size: baseSize * scaleFactor
- Find smallest available resolution >= target size
- If none larger, use largest available
- Apply high-quality scaling if exact match unavailable
- Cache rendered result until scale factor changes

5.2.4 FontManager

Purpose: Singleton providing consistently styled and scaled fonts.

Logical Sizes Enum:

- TINY (9pt base), SMALL (10pt), NORMAL (12pt), MEDIUM (14pt), LARGE (16pt), XLARGE (18pt), HUGE (24pt)

Key Methods:

- getInstance(): FontManager - Returns singleton instance
- getFont(LogicalSize size): Font - Returns plain font at logical size
- getFont(LogicalSize size, int style): Font - With style (BOLD, ITALIC, BOLD|ITALIC)
- getFont(String family, LogicalSize size, int style): Font - Specific family
- getMonospacedFont(LogicalSize size): Font - For code/data display
- deriveScaled(Font baseFont): Font - Scale arbitrary font to current DPI

5.2.5 MigConstraintScaler

Purpose: Utility class for transforming MigLayout constraint strings.

Key Methods:

- scale(String constraint): String - Transforms single constraint
- scaleLayout(String layoutConstraints): String - Scale layout-level constraints
- scaleColumn(String columnConstraints): String - Scale column constraints
- scaleRow(String rowConstraints): String - Scale row constraints
- scaleComponent(String componentConstraints): String - Scale component constraints

Recognized Patterns:

- Numeric values followed by optional unit: 10, 10px, 10pt
- Range specifications: min:pref:max (e.g., 100:150:200)
- Gap specifications: gap, gapx, gapy, gaptop, gapbottom, gapleft, gapright
- Insets: insets followed by 1-4 numeric values
- Size constraints: width, height, wmin, wmax, hmin, hmax
- Padding: pad followed by 1-4 numeric values

5.2.6 ScalablePanel

Purpose: Base JPanel subclass with built-in scaling support.

Key Features:

- Implements ScaleChangeListener automatically
- Stores original MigLayout constraints for re-scaling
- Provides template methods for subclass customization
- Handles proper listener registration/deregistration lifecycle

Template Methods:

- onScaleChanging(double newScale): void - Called before scale update
- onScaleChanged(double newScale): void - Called after scale update
- getOriginalLayoutConstraints(): String - Return constraints for re-scaling
- getOriginalColumnConstraints(): String - Return column constraints
- getOriginalRowConstraints(): String - Return row constraints

5.3 Integration Points

5.3.1 Application Startup Sequence

- JVM starts with -Dsun.java2d.uiScale.enabled=true (if on Java 9+)
- ScaleManager initializes and detects current DPI
- FontManager initializes with current scale factor
- Main window frame created with scaled initial size
- All panels created with scaled constraints
- Icons loaded at appropriate resolutions

- Application becomes visible

5.3.2 JVM Arguments

Argument	Purpose	Recommended Value
-Dsun.java2d.uiScale.enabled	Enable Java's built-in HiDPI support	true
-Dsun.java2d.uiScale	Force specific scale factor (testing)	1.0, 1.5, 2.0, etc.
-Dswing.aatext	Enable antialiased text	true
-Dawt.useSystemAAFontSettings	Use system font smoothing	on
-Dsun.java2d.opengl	Enable OpenGL acceleration (Linux)	true

6. Implementation Strategy

6.1 Phased Approach

Phase 1: Foundation (Weeks 1-2)

Task	Description	Deliverable
1.1	Implement ScaleManager singleton	ScaleManager.java with full API
1.2	Implement ScaleChangeListener interface	Interface + default adapter
1.3	Create unit tests for scaling calculations	Test coverage > 90%
1.4	Document JVM arguments for all platforms	Configuration guide

Phase 2: Core Components (Weeks 3-4)

Task	Description	Deliverable
2.1	Implement FontManager	FontManager.java with logical sizes
2.2	Implement ScalableIcon	ScalableIcon.java with multi-res support
2.3	Create icon asset generation pipeline	Build script + documentation
2.4	Generate multi-resolution icons for existing assets	Complete icon set at all resolutions

Phase 3: Layout Integration (Weeks 5-6)

Task	Description	Deliverable
3.1	Implement MigConstraintScaler	Utility class with pattern matching
3.2	Implement ScalablePanel base class	Base class with lifecycle hooks
3.3	Migrate existing panels to ScalablePanel	All panels extend ScalablePanel
3.4	Update CardLayout containers	Scale-aware card management

Phase 4: Advanced Features (Weeks 7-8)

Task	Description	Deliverable
4.1	Implement multi-monitor detection	Monitor change listener
4.2	Implement runtime DPI change handling	Platform-specific detection
4.3	Performance optimization and caching	Profiling report + optimizations
4.4	Edge case handling and hardening	Comprehensive error handling

Phase 5: Testing and Refinement (Weeks 9-10)

Task	Description	Deliverable
5.1	Cross-platform testing	Test results on Win/Mac/Linux
5.2	Multi-DPI configuration testing	Test matrix completion
5.3	Performance regression testing	Performance benchmark report
5.4	User acceptance testing	UAT sign-off
5.5	Documentation finalization	Complete technical documentation

6.2 Migration Guidelines

6.2.1 Identifying Hardcoded Values

Search the codebase for these patterns indicating hardcoded dimensions:

- new Dimension(\\d+, \\d+) - Hardcoded component sizes
- setPreferredSize, setMinimumSize, setMaximumSize with pixel values
- new Font(.*?, \\d+) - Hardcoded font point sizes
- ImageIcon with pixel dimensions
- MigLayout constraints with numeric pixel values
- setBounds, setLocation with pixel coordinates
- Custom painting with pixel coordinates (getWidth()/getHeight() are okay)

6.2.2 Refactoring Patterns

Before/After examples for common refactoring scenarios:

Dimension Scaling:

Before: new Dimension(200, 150)

After: ScaleManager.getInstance().scale(new Dimension(200, 150))

Font Creation:

Before: new Font("Arial", Font.PLAIN, 12)

After: FontManager.getInstance().getFont(LogicalSize.NORMAL)

Icon Loading:

Before: new ImageIcon(getClass().getResource("/icons/save.png"))

After: new ScalableIcon("/icons/save")

MigLayout Constraints:

Before: new MigLayout("insets 10", "[100][grow]", "[]")

After: new MigLayout(MigConstraintScaler.scaleLayout("insets 10"), MigConstraintScaler.scaleColumn("[100][grow]"), MigConstraintScaler.scaleRow("[]"))

7. Testing Requirements

7.1 Test Environments

Environment	Configuration	Priority
Windows 11	100% scaling (96 DPI)	P0
Windows 11	125% scaling (120 DPI)	P0
Windows 11	150% scaling (144 DPI)	P0
Windows 11	200% scaling (192 DPI)	P0
Windows 11	300% scaling (288 DPI)	P1
Windows 10	Mixed DPI multi-monitor	P1
macOS (Retina)	2x scaling	P0
macOS (Non-Retina)	1x scaling	P1
Ubuntu 22.04 (GNOME)	100%, 200% fractional scaling	P2
Fedora (KDE)	100%, 150%, 200%	P2

7.2 Test Cases

7.2.1 Unit Tests

Test ID	Component	Test Description
UT-001	ScaleManager	Verify scale factor calculation at various DPIs
UT-002	ScaleManager	Verify integer scaling rounds correctly
UT-003	ScaleManager	Verify Dimension scaling preserves aspect ratio
UT-004	ScaleManager	Verify listener notification on scale change
UT-005	FontManager	Verify logical size to point size mapping
UT-006	FontManager	Verify font caching returns same instance
UT-007	MigConstraintScaler	Verify gap constraint transformation
UT-008	MigConstraintScaler	Verify insets constraint transformation
UT-009	MigConstraintScaler	Verify size constraint transformation
UT-010	MigConstraintScaler	Verify percentage constraints unchanged
UT-011	ScalableIcon	Verify correct resolution selection
UT-012	ScalableIcon	Verify fallback to nearest resolution

7.2.2 Integration Tests

Test ID	Scenario	Expected Result
IT-001	Application startup at 96 DPI	All components render at baseline sizes
IT-002	Application startup at 192 DPI	All components render at 2x sizes
IT-003	CardLayout panel switch at 150%	Hidden panel renders correctly when shown
IT-004	Runtime DPI change 100% to 200%	All visible components rescale within 500ms
IT-005	Window move between monitors	Scaling adjusts to new monitor DPI
IT-006	Icon rendering at 1.5x	Icons use @1.5x or scaled @2x variant
IT-007	Font rendering at 200%	Text remains proportional and readable

7.2.3 Visual Regression Tests

- Automated screenshot capture at each supported DPI
- Pixel-level comparison against approved baselines
- Tolerance threshold for antialiasing differences
- Focus on: dialogs, toolbars, menus, main content areas
- Separate baselines per platform due to native rendering differences

7.3 Performance Tests

Metric	Baseline (96 DPI)	Acceptable (192 DPI)	Test Method
Application startup time	< 3 seconds	< 3.5 seconds	Automated timing
Memory at idle	< 200 MB	< 230 MB	JVM heap monitoring
Scale change response	N/A	< 500 ms	Automated timing
Icon load time (cold)	< 50 ms each	< 75 ms each	Profiling
Panel switch (CardLayout)	< 100 ms	< 150 ms	Automated timing

8. Risk Assessment

Risk	Probability	Impact	Mitigation
Performance degradation from constant scale calculations	Medium	High	Implement aggressive caching; profile early and often
Inconsistent behavior across platforms	Medium	Medium	Establish platform-specific test matrix; document known differences
Third-party library incompatibility	Low	High	Audit dependencies for DPI-awareness; isolate in wrapper classes
Memory increase from multi-resolution icons	Medium	Low	Lazy loading; evict unused resolutions from cache
Fractional scaling artifacts	High	Medium	Use Math.round() consistently; test fractional scales extensively
Runtime DPI change not detected	Medium	Medium	Polling fallback; user-accessible refresh button
Breaking existing custom components	Medium	High	Comprehensive audit; provide migration guide
CardLayout sizing issues with cached panels	Medium	High	Explicit invalidation protocol; test all card transitions

9. Acceptance Criteria

9.1 Minimum Viable Product (MVP)

The following criteria must be met for initial release:

- Application renders correctly at 100%, 125%, 150%, and 200% Windows scaling
- All icons appear crisp (no visible blur) at all supported scale factors
- All text is readable and proportionally sized at all supported scale factors
- No layout overflow, clipping, or overlapping components at any scale factor
- Application startup time increase is less than 15% compared to baseline
- Memory footprint increase is less than 20% compared to baseline
- All existing functionality remains operational without regression

9.2 Full Release

Additional criteria for complete implementation:

- Application responds to runtime DPI changes without restart
- Multi-monitor support with different DPIs per monitor
- macOS Retina display support verified
- Linux (GNOME and KDE) scaling support verified
- Support for scale factors up to 300% (288 DPI)
- Complete technical documentation delivered
- All unit and integration tests passing

10. Appendices

10.1 Appendix A: DPI and Scale Factor Reference

Windows Scaling	DPI Value	Scale Factor	Common Use Case
100%	96	1.0	Standard desktop monitors
125%	120	1.25	Laptops, smaller 4K monitors
150%	144	1.5	High-res laptops (Surface, XPS)
175%	168	1.75	Large high-res displays
200%	192	2.0	4K monitors, Retina displays
225%	216	2.25	Very high-res displays
250%	240	2.5	5K monitors
300%	288	3.0	8K monitors, extreme HiDPI

10.2 Appendix B: Aspect Ratio Reference

Aspect Ratio	Common Resolutions	Use Case
4:3	1024x768, 1280x960, 1600x1200	Legacy monitors, some projectors
16:10	1280x800, 1440x900, 1920x1200	Older MacBooks, some Dell monitors
16:9	1920x1080, 2560x1440, 3840x2160	Most modern monitors and TVs
21:9	2560x1080, 3440x1440, 5120x2160	Ultra-wide monitors
32:9	3840x1080, 5120x1440	Super ultra-wide gaming monitors

10.3 Appendix C: Java Version Considerations

Java Version	HiDPI Support	Notes
Java 8	Partial	Requires manual scaling; no automatic support
Java 9	Improved	Added sun.java2d.uiScale property
Java 11	Good	Better multi-monitor support
Java 17	Excellent	Recommended; best cross-platform support
Java 21	Excellent	Latest LTS; full feature support

10.4 Appendix D: Glossary

Term	Definition
DPI	Dots Per Inch - measure of display pixel density
HiDPI	High DPI - displays with greater than 96 DPI
Retina	Apple's marketing term for HiDPI displays (typically 2x)
Scale Factor	Multiplier applied to base dimensions (1.0 = 96 DPI baseline)
Logical Pixel	Abstract pixel unit that scales with DPI
Physical Pixel	Actual screen pixel
Device Pixel Ratio	Ratio of physical to logical pixels
PPI	Pixels Per Inch - often used interchangeably with DPI
Fractional Scaling	Scale factors between whole numbers (e.g., 1.25, 1.5)

10.5 Appendix E: Reference Implementation Locations

Suggested package structure for scaling infrastructure:

- com.yourapp.ui.scale.ScaleManager - Central scaling singleton
- com.yourapp.ui.scale.ScaleChangeListener - Observer interface
- com.yourapp.ui.scale.FontManager - Font provisioning singleton
- com.yourapp.ui.scale.ScalableIcon - Multi-resolution icon class
- com.yourapp.ui.scale.MigConstraintScaler - Constraint transformation utility
- com.yourapp.ui.scale.ScalablePanel - DPI-aware panel base class
- com.yourapp.ui.scale.LogicalSize - Font size enumeration