# Kiwik Feature Spec: Market Card Image ↔ Graph Rotation

> **Purpose**: Implement an auto-rotating display on market cards that transitions between the market cover image and a dynamically-drawn price history graph. This adds dynamism, reinforces that markets are "live," and provides valuable price trend information at a glance.
>
> **Design Philosophy**: The animation should feel organic and alive—like the market is breathing. The graph line should draw as if being sketched by hand, not rendered by a computer. Timing is critical: too fast feels chaotic, too slow feels sluggish.

---

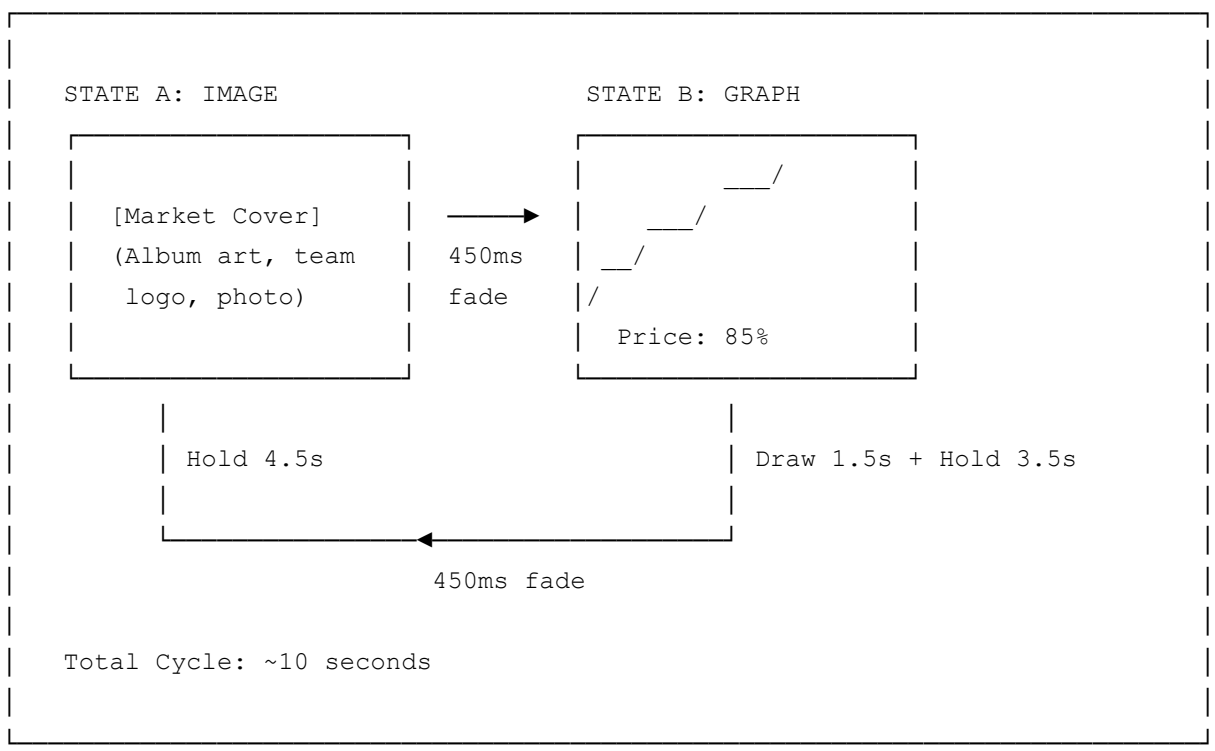
## Table of Contents


1. Visual Concept
2. Timing Specification
3. Graph Drawing Animation
4. Transition Effects
5. Component Architecture
6. Graph Visual Design
7. Interaction Behaviors
8. Performance Optimization
9. Accessibility Considerations
10. Edge Cases
11. Implementation Code
12. Acceptance Criteria


---

## Visual Concept

### State Diagram




### What Users See

1. **Market card displays cover image** (e.g., Bad Bunny for music market, team logo for sports)
2. **Smooth crossfade to graph view** (image fades out, graph fades in simultaneously)
3. **Graph line draws from left to right** (like being sketched in real-time)
4. **Current price point pulses briefly** (draws attention to current probability)
5. **Hold on graph** (user absorbs trend information)
6. **Smooth crossfade back to image**
7. **Cycle repeats**

---

## Timing Specification

### Master Timing Table

| Phase | Duration | Cumulative | Purpose |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Image display | 4500ms | 0-4500ms | Let user see/recognize the market |
| Crossfade to graph | 450ms | 4500-4950ms | Smooth transition |
| Graph line drawing | 1500ms | 4950-6450ms | The “wow” moment |
| Endpoint pulse | 300ms | 6450-6750ms | Highlight current price |
| Graph hold | 3250ms | 6750-10000ms | Absorb trend data |
| Crossfade to image | 450ms | 10000-10450ms | Return to visual |
| **Total cycle** | **~10000ms** | | Full rotation |

## Timing Rationale

### Why 450ms Crossfade?

```
< 300ms  → Feels abrupt, jarring, "flashy"
300-350ms → Acceptable but slightly rushed
400-500ms → Sweet spot: perceptually instant yet smooth
> 600ms  → Feels sluggish, draws attention to the transition itself

Recommendation: 450ms with ease-in-out easing
```

### Why 1500ms Line Drawing?

```
< 800ms  → Mechanical, robotic, loses the "hand-drawn" feel
800-1200ms → Acceptable for short/simple lines
1200-1800ms → Optimal: feels organic, satisfying to watch
> 2500ms → Tests patience, feels slow

Recommendation: 1500ms with ease-out easing (fast start, gentle finish)
```

### Why 4500ms Image Hold?

```
Users need time to:
1. Notice the card (~500ms)
2. Recognize the image content (~1000ms)
3. Read the market question (~2000ms)
4. Prepare for transition (~1000ms)
```

```
Shorter durations create cognitive overload.
Longer durations feel static/boring.
```

#### Why 3250ms Graph Hold?

```
Shorter than image hold because:
- Graph is more information-dense (needs focus, not recognition)
- Users either "get it" quickly or need more time than we can give
- Keeps the overall cycle feeling dynamic

The 1500ms draw time + 3250ms hold ≈ same total as image phase
```

### Timing Constants (for implementation)

```
const TIMING = {
  IMAGE_HOLD: 4500,
  CROSSFADE: 450,
  LINE_DRAW: 1500,
  ENDPOINT_PULSE: 300,
  GRAPH_HOLD: 3250,
  TOTAL_CYCLE: 10000,

  // Stagger offset between cards
  CARD_STAGGER: 2500,
};
```

---

## Graph Drawing Animation

### SVG Path Drawing Technique

The "line drawing" effect uses SVG's `stroke-dasharray` and `stroke-dashoffset` properties. The path is rendered with a dash pattern equal to its total length, then animated from fully hidden to fully visible.

#### How It Works

```
1. Calculate total path length
2. Set stroke-dasharray to path length (creates one "dash" = entire line)
3. Set stroke-dashoffset to path length (hides entire line)
4. Animate stroke-dashoffset to 0 (reveals line progressively)
```

### CSS Implementation

```
.graph-line {
  fill: none;
  stroke: var(--color-primary);
  stroke-width: 2.5px;
  stroke-linecap: round;
  stroke-linejoin: round;

  /* These values set by JS based on actual path length */
  stroke-dasharray: var(--path-length);
  stroke-dashoffset: var(--path-length);

  /* Animation triggered by adding .drawing class */
  transition: stroke-dashoffset 1.5s ease-out;
}

.graph-line.drawing {
  stroke-dashoffset: 0;
}
```

### JavaScript Path Length Calculation

```
// After SVG renders, calculate actual path length
const calculatePathLength = (pathElement) => {
  const length = pathElement.getTotalLength();
  pathElement.style.setProperty('--path-length', length);
  return length;
};

// Usage
const path = document.querySelector('.graph-line');
calculatePathLength(path);
```

## Easing Function Deep Dive

```
Recommended: ease-out
CSS: cubic-bezier(0, 0, 0.58, 1)
Custom: cubic-bezier(0.25, 0.46, 0.45, 0.94) — slightly softer

Behavior:
- Starts at full speed
- Gradually decelerates
```

```
- Mimics hand-drawn stroke (confident start, careful finish)

Avoid:
- linear: Robotic, unnatural
- ease-in: Slow start feels laggy/hesitant
- ease-in-out: Slow start unnecessary for line drawing
```

### Line Drawing with Area Fill

For added visual impact, fill the area under the line after the line completes drawing:

```
.graph-area {
  fill: var(--color-primary);
  opacity: 0;
  transition: opacity 0.4s ease 1.5s; /* Delay until line finishes */
}

.graph-area.visible {
  opacity: 0.1; /* Subtle fill */
}
```

---

## Transition Effects

### Option A: Simple Crossfade (Recommended)

The simplest and cleanest approach. Both layers exist; we animate opacity.

```
.card-visual-container {
  position: relative;
  aspect-ratio: 16 / 9; /* Or your card image ratio */
  overflow: hidden;
  border-radius: 8px;
}

.visual-layer {
  position: absolute;
  inset: 0;
  transition: opacity 450ms ease-in-out;
}

.visual-layer--image {
  z-index: 1;
}
```

```
.visual-layer--graph {
  z-index: 2;
  opacity: 0;
  pointer-events: none;
}

/* When showing graph */
.card-visual-container.show-graph .visual-layer--image {
  opacity: 0;
}

.card-visual-container.show-graph .visual-layer--graph {
  opacity: 1;
  pointer-events: auto;
}
```

### Option B: Clip/Wipe Reveal

Graph “wipes” in from left to right, synchronized with line drawing. More dynamic but more complex.

```
.visual-layer--graph {
  clip-path: inset(0 100% 0 0); /* Hidden: clipped from right */
  transition: clip-path 1.5s ease-out; /* Same duration as line draw */
}

.card-visual-container.show-graph .visual-layer--graph {
  clip-path: inset(0 0 0 0); /* Fully visible */
}
```

### Option C: Scale + Fade

Graph fades in while scaling from 95% to 100%. Adds subtle depth.

```
.visual-layer--graph {
  opacity: 0;
  transform: scale(0.95);
  transition:
    opacity 450ms ease-in-out,
    transform 450ms ease-out;
}

.card-visual-container.show-graph .visual-layer--graph {
  opacity: 1;
```

```
  transform: scale(1);
}
```

### Recommendation

Start with **Option A (Simple Crossfade)**. It's clean, performant, and doesn't compete with the line-drawing animation for attention. The line draw IS the visual interest—the transition should be invisible.

---

## Component Architecture

### React Component Structure

```
MarketCard/
├── MarketCard.jsx           # Main card component
├── MarketCardVisual.jsx     # Handles image ↔ graph rotation
├── PriceGraph.jsx           # SVG graph with drawing animation
├── useRotationCycle.js      # Custom hook for timing logic
└── styles.module.css        # Component styles
```

### Component Hierarchy

```
<MarketCard>
  <MarketCardVisual
    image={market.coverImage}
    priceHistory={market.priceHistory}
    currentPrice={market.currentPrice}
  />
  <MarketCardContent>
    <CategoryTag />
    <MarketQuestion />
    <ProbabilityBar />
    <TimeRemaining />
  </MarketCardContent>
</MarketCard>
```

### MarketCardVisual Component

```
// MarketCardVisual.jsx
import { useState, useEffect, useRef } from 'react';
import PriceGraph from './PriceGraph';
```

```
import styles from './styles.module.css';

const TIMING = {
  IMAGE_HOLD: 4500,
  CROSSFADE: 450,
  LINE_DRAW: 1500,
  GRAPH_HOLD: 3250,
};

const MarketCardVisual = ({
  image,
  priceHistory,
  currentPrice,
  autoRotate = true,
  initialDelay = 0
}) => {
  const [showGraph, setShowGraph] = useState(false);
  const [isDrawing, setIsDrawing] = useState(false);
  const [isPaused, setIsPaused] = useState(false);
  const timeoutRef = useRef(null);

  useEffect(() => {
    if (!autoRotate || isPaused) return;

    const startCycle = () => {
      // Phase 1: Show image (already showing)
      timeoutRef.current = setTimeout(() => {
        // Phase 2: Transition to graph
        setShowGraph(true);
        setIsDrawing(true);

        // Phase 3: Line drawing completes
        setTimeout(() => {
          setIsDrawing(false);
        }, TIMING.LINE_DRAW);

        // Phase 4: Hold graph, then transition back
        setTimeout(() => {
          setShowGraph(false);
        }, TIMING.LINE_DRAW + TIMING.GRAPH_HOLD);

      }, TIMING.IMAGE_HOLD);
    };

    // Initial delay for staggering multiple cards
    const initialTimer = setTimeout(startCycle, initialDelay);
```

```
  // Set up repeating cycle
  const cycleInterval = setInterval(() => {
    startCycle();
  }, TIMING.IMAGE_HOLD + TIMING.CROSSFADE + TIMING.LINE_DRAW + TIMING.GRAPH_HOL

  return () => {
    clearTimeout(initialTimer);
    clearTimeout(timeoutRef.current);
    clearInterval(cycleInterval);
  };
}, [autoRotate, isPaused, initialDelay]);

// Pause on hover
const handleMouseEnter = () => setIsPaused(true);
const handleMouseLeave = () => setIsPaused(false);

return (
  <div
    className={`${styles.visualContainer} ${showGraph ? styles.showGraph : ''}`
    onMouseEnter={handleMouseEnter}
    onMouseLeave={handleMouseLeave}
  >
    {/* Image Layer */}
    <div className={`${styles.layer} ${styles.imageLayer}`}>
      <img
        src={image}
        alt=""
        loading="lazy"
        className={styles.coverImage}
      />
    </div>

    {/* Graph Layer */}
    <div className={`${styles.layer} ${styles.graphLayer}`}>
      <PriceGraph
        data={priceHistory}
        currentPrice={currentPrice}
        isDrawing={isDrawing}
        showArea={!isDrawing && showGraph}
      />
    </div>

    {/* Optional: Visual indicator dots */}
    <div className={styles.indicators}>
      <span className={`${styles.dot} ${!showGraph ? styles.active : ''}`} />
      <span className={`${styles.dot} ${showGraph ? styles.active : ''}`} />
    </div>
```

```
    </div>
  );
};

export default MarketCardVisual;
```

## PriceGraph Component

```
// PriceGraph.jsx
import { useRef, useEffect, useState } from 'react';
import styles from './styles.module.css';

const PriceGraph = ({
  data,
  currentPrice,
  isDrawing,
  showArea
}) => {
  const pathRef = useRef(null);
  const [pathLength, setPathLength] = useState(0);

  // Generate SVG path from price data
  const generatePath = (data, width, height) => {
    if (!data || data.length === 0) return '';

    const padding = 10;
    const graphWidth = width - (padding * 2);
    const graphHeight = height - (padding * 2);

    const xStep = graphWidth / (data.length - 1);

    const points = data.map((value, index) => {
      const x = padding + (index * xStep);
      const y = padding + graphHeight - (value / 100 * graphHeight);
      return `${x},${y}`;
    });

    return `M ${points.join(' L ')}`;
  };

  // Generate area path (same line + bottom edge)
  const generateAreaPath = (data, width, height) => {
    const linePath = generatePath(data, width, height);
    if (!linePath) return '';

    const padding = 10;
```

```
  const lastX = width - padding;
  const firstX = padding;
  const bottom = height - padding;

  return `${linePath} L ${lastX},${bottom} L ${firstX},${bottom} Z`;
};

// Calculate path length after render
useEffect(() => {
  if (pathRef.current) {
    const length = pathRef.current.getTotalLength();
    setPathLength(length);
  }
}, [data]);

const width = 300;
const height = 150;
const linePath = generatePath(data, width, height);
const areaPath = generateAreaPath(data, width, height);

// Current price point position
const lastDataPoint = data[data.length - 1];
const dotX = width - 10;
const dotY = 10 + (height - 20) - (lastDataPoint / 100 * (height - 20));

return (
  <svg
    viewBox={`0 0 ${width} ${height}`}
    className={styles.graphSvg}
    preserveAspectRatio="none"
  >
    {/* Background grid (optional) */}
    <defs>
      <pattern id="grid" width="30" height="30" patternUnits="userSpaceOnUse">
        <path
          d="M 30 0 L 0 0 0 30"
          fill="none"
          stroke="currentColor"
          strokeWidth="0.5"
          opacity="0.1"
        />
      </pattern>
    </defs>
    <rect width="100%" height="100%" fill="url(#grid)" />

    {/* Area fill (appears after line draws) */}
    <path
```

```
        d={areaPath}
        className={`${styles.graphArea} ${showArea ? styles.visible : ''}`}
      />

      {/* Main price line */}
      <path
        ref={pathRef}
        d={linePath}
        className={`${styles.graphLine} ${isDrawing ? styles.drawing : ''}`}
        style={{
          '--path-length': pathLength,
          strokeDasharray: pathLength,
          strokeDashoffset: isDrawing ? 0 : pathLength,
        }}
      />

      {/* Current price dot */}
      <circle
        cx={dotX}
        cy={dotY}
        r="5"
        className={`${styles.priceDot} ${!isDrawing && showArea ? styles.visible
      />

      {/* Price label */}
      <text
        x={dotX - 12}
        y={dotY - 12}
        className={`${styles.priceLabel} ${!isDrawing && showArea ? styles.visibl
      >
        {currentPrice}%
      </text>
    </svg>
  );
};

export default PriceGraph;
```

## CSS Styles

```
/* styles.module.css */

/* Container */
.visualContainer {
  position: relative;
  aspect-ratio: 16 / 9;
```

```
  overflow: hidden;
  border-radius: 8px 8px 0 0;
  background: var(--bg-secondary);
}

/* Layers */
.layer {
  position: absolute;
  inset: 0;
  transition: opacity 450ms ease-in-out;
}

.imageLayer {
  z-index: 1;
}

.graphLayer {
  z-index: 2;
  opacity: 0;
  pointer-events: none;
  background: var(--bg-primary);
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 16px;
}

.showGraph .imageLayer {
  opacity: 0;
}

.showGraph .graphLayer {
  opacity: 1;
  pointer-events: auto;
}

/* Cover Image */
.coverImage {
  width: 100%;
  height: 100%;
  object-fit: cover;
}

/* Graph SVG */
.graphSvg {
  width: 100%;
  height: 100%;
```

```
}

/* Graph Line */
.graphLine {
  fill: none;
  stroke: var(--color-primary);
  stroke-width: 2.5px;
  stroke-linecap: round;
  stroke-linejoin: round;
  transition: stroke-dashoffset 1.5s ease-out;
}

.graphLine:not(.drawing) {
  stroke-dashoffset: var(--path-length) !important;
}

.graphLine.drawing {
  stroke-dashoffset: 0 !important;
}

/* Area Fill */
.graphArea {
  fill: var(--color-primary);
  opacity: 0;
  transition: opacity 0.4s ease;
  transition-delay: 0.2s; /* Slight delay after line completes */
}

.graphArea.visible {
  opacity: 0.1;
}

/* Price Dot */
.priceDot {
  fill: var(--color-primary);
  opacity: 0;
  transform-origin: center;
  transition: opacity 0.3s ease, transform 0.3s ease;
}

.priceDot.visible {
  opacity: 1;
  animation: pulseDot 1s ease-in-out;
}

@keyframes pulseDot {
  0%, 100% { transform: scale(1); }
```

```
  50% { transform: scale(1.4); }
}

/* Price Label */
.priceLabel {
  font-size: 12px;
  font-weight: 600;
  fill: var(--color-primary);
  opacity: 0;
  transition: opacity 0.3s ease;
}

.priceLabel.visible {
  opacity: 1;
}

/* Indicator Dots */
.indicators {
  position: absolute;
  bottom: 8px;
  left: 50%;
  transform: translateX(-50%);
  display: flex;
  gap: 6px;
  z-index: 10;
}

.dot {
  width: 6px;
  height: 6px;
  border-radius: 50%;
  background: white;
  opacity: 0.4;
  transition: opacity 0.3s ease;
}

.dot.active {
  opacity: 1;
}

/* Reduced Motion */
@media (prefers-reduced-motion: reduce) {
  .layer,
  .graphLine,
  .graphArea,
  .priceDot,
  .priceLabel {
```

```
    transition: none !important;
    animation: none !important;
  }

  .graphLine {
    stroke-dashoffset: 0 !important;
  }
}
```

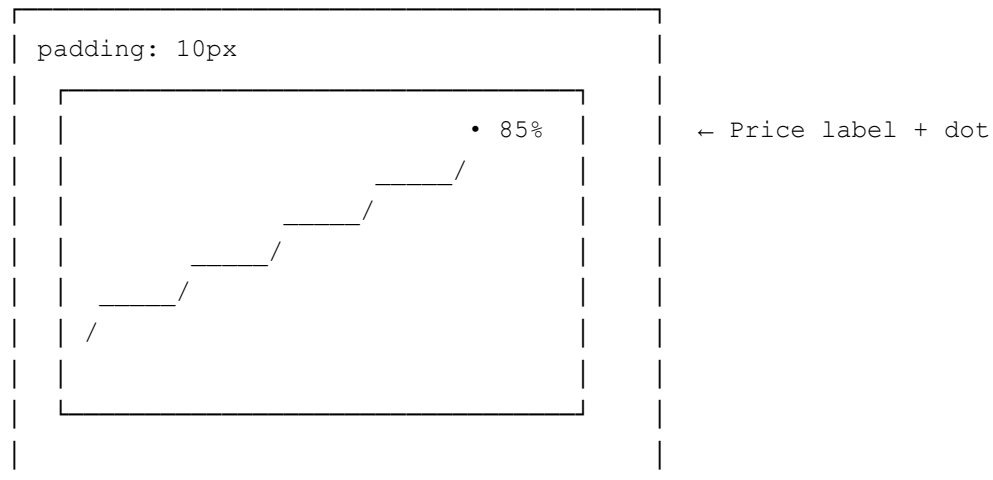
---

# Graph Visual Design

## Color Coding by Outcome

```
// Determine line color based on which outcome is displayed
const getLineColor = (outcome, probability) => {
  if (outcome === 'yes') {
    // Green spectrum based on probability
    if (probability >= 70) return '#16a34a'; // Strong green
    if (probability >= 50) return '#22c55e'; // Medium green
    return '#86efac'; // Light green
  } else {
    // Neutral/red spectrum for "No"
    if (probability >= 70) return '#dc2626'; // Strong red (high No probability)
    if (probability >= 50) return '#94a3b8'; // Neutral gray
    return '#cbd5e1'; // Light gray
  }
};
```

## Graph Dimensions & Layout



```
│  [7d]                          [Now]   │  ← Optional time labels
│                                        │
└────────────────────────────────────────┘
```

```
Aspect ratio: 16:9 (matches typical card images)
Y-axis: 0% to 100% (implicit, not labeled)
X-axis: Last 7 or 30 days of price data
```

### Subtle Grid Background

```
<defs>
  <pattern id="grid" width="30" height="30" patternUnits="userSpaceOnUse">
    <path
      d="M 30 0 L 0 0 0 30"
      fill="none"
      stroke="currentColor"
      strokeWidth="0.5"
      opacity="0.08"
    />
  </pattern>
</defs>
<rect width="100%" height="100%" fill="url(#grid)" />
```

### Line Styling

```
.graphLine {
  stroke-width: 2.5px;      /* Visible but not chunky */
  stroke-linecap: round;    /* Soft line endings */
  stroke-linejoin: round;   /* Smooth corners */
}
```

---

## Interaction Behaviors

### Pause on Hover

When user hovers over a card, pause the rotation cycle. They're engaged and shouldn't be interrupted.

```
const [isPaused, setIsPaused] = useState(false);

// In useEffect, check isPaused before continuing cycle
```

```
useEffect(() => {
  if (isPaused) return;
  // ... rotation logic
}, [isPaused]);

// Event handlers
<div
  onMouseEnter={() => setIsPaused(true)}
  onMouseLeave={() => setIsPaused(false)}
>
```

## Resume Logic

When unpausing, you have two options:

### Option A: Resume from current state

```
// Continue from wherever we paused
// Simpler, but might cut phases short
```

### Option B: Start fresh cycle

```
// Reset to beginning of cycle
// More predictable, cleaner
const handleMouseLeave = () => {
  setIsPaused(false);
  setShowGraph(false);
  setIsDrawing(false);
  // Will start fresh cycle on next interval
};
```

## Manual Toggle (Optional)

Allow users to click to toggle between image and graph:

```
const handleClick = () => {
  setIsPaused(true); // Stop auto-rotation
  setShowGraph(prev => !prev);
  if (!showGraph) {
    setIsDrawing(true);
    setTimeout(() => setIsDrawing(false), TIMING.LINE_DRAW);
  }
};
```

### Touch Devices

On mobile, hover isn't available. Consider:

```
// Tap to pause/play
const handleTap = () => {
  setIsPaused(prev => !prev);
};

// Or: Tap to toggle view
const handleTap = () => {
  setShowGraph(prev => !prev);
};
```

---

## Performance Optimization

### Key Principles

```
1. Use CSS transitions/animations (GPU-accelerated)
2. Avoid animating layout properties (width, height, margin)
3. Use transform and opacity only
4. Limit simultaneous animations
5. Pause off-screen cards
```

### Intersection Observer for Visibility

Only animate cards that are in the viewport:

```
const useVisibility = (ref) => {
  const [isVisible, setIsVisible] = useState(false);

  useEffect(() => {
    const observer = new IntersectionObserver(
      ([entry]) => setIsVisible(entry.isIntersecting),
      { threshold: 0.2 }
    );

    if (ref.current) observer.observe(ref.current);
    return () => observer.disconnect();
  }, [ref]);

  return isVisible;
```

```
};

// In component
const containerRef = useRef(null);
const isVisible = useVisibility(containerRef);

// Only run cycle when visible
useEffect(() => {
  if (!isVisible) return;
  // ... rotation logic
}, [isVisible]);
```

### Stagger Card Animations

Don’t start all cards simultaneously—creates visual chaos and performance spike:

```
// Parent component
const MarketCardList = ({ markets }) => {
  return (
    <div className="market-grid">
      {markets.map((market, index) => (
        <MarketCard
          key={market.id}
          {...market}
          initialDelay={index * 2500} // 2.5s stagger
        />
      ))}
    </div>
  );
};
```

### Will-Change Optimization

Hint to browser about upcoming animations:

```
.layer {
  will-change: opacity;
}

/* Remove after animation settles (optional) */
.layer:not(.transitioning) {
  will-change: auto;
}
```

### Limit Simultaneous Animations

If many cards are visible, limit how many can animate at once:

```
const MAX_SIMULTANEOUS = 4;

// Track which cards are currently animating
const [animatingCards, setAnimatingCards] = useState(new Set());

const canAnimate = (cardId) => {
  return animatingCards.size < MAX_SIMULTANEOUS;
};
```

---

## Accessibility Considerations

### Reduced Motion

Respect user's motion preferences:

```
@media (prefers-reduced-motion: reduce) {
  .layer {
    transition: none !important;
  }

  .graphLine {
    transition: none !important;
    stroke-dashoffset: 0 !important; /* Show immediately */
  }

  .priceDot {
    animation: none !important;
  }
}


// In component
const prefersReducedMotion = window.matchMedia(
  '(prefers-reduced-motion: reduce)'
).matches;

// If reduced motion, show graph without animation
if (prefersReducedMotion) {
  setIsDrawing(false); // Skip drawing phase
}
```

### Screen Reader Considerations

The rotation is purely visual. Ensure the information is available textually:

```
<div className={styles.visualContainer} aria-hidden="true">
  {/* Visual rotation content */}
</div>

{/* Accessible price information */}
<div className="sr-only">
  Current probability: {currentPrice}%
  Trend: {trend === 'up' ? 'increasing' : 'decreasing'} over last 7 days
</div>
```

### Pause Control

Provide a way to stop all animations:

```
<button
  className={styles.pauseAllButton}
  onClick={() => setGlobalPause(prev => !prev)}
  aria-label={globalPause ? 'Resume animations' : 'Pause animations'}
>
  {globalPause ? '▶' : '⏸️'}
</button>
```

---

## Edge Cases

### No Price History Data

If a market is new and has no historical data:

```
if (!priceHistory || priceHistory.length < 2) {
  // Don't show graph rotation
  // Or show a flat line at current price
  return <StaticImage image={image} />;
}
```

### Image Load Failure

```
const [imageError, setImageError] = useState(false);

<img
  src={image}
  onError={() => setImageError(true)}
  alt=""
/>

{imageError && (
  <div className={styles.fallbackImage}>
    <CategoryIcon category={market.category} />
  </div>
)}
```

### Very Short Price History

If only a few data points, the graph looks sparse:

```
// Interpolate additional points for smoother line
const interpolateData = (data, targetPoints = 20) => {
  if (data.length >= targetPoints) return data;

  const result = [];
  const step = (data.length - 1) / (targetPoints - 1);

  for (let i = 0; i < targetPoints; i++) {
    const index = i * step;
    const lower = Math.floor(index);
    const upper = Math.ceil(index);
    const t = index - lower;

    if (upper >= data.length) {
      result.push(data[data.length - 1]);
    } else {
      result.push(data[lower] * (1 - t) + data[upper] * t);
    }
  }

  return result;
};
```

### Extreme Price Swings

If price swings from 5% to 95%, the graph is very vertical. Consider clamping or smoothing:

```
// Optional: Apply smoothing to reduce visual noise
const smoothData = (data, windowSize = 3) => {
  return data.map((_, index, arr) => {
    const start = Math.max(0, index - windowSize);
    const end = Math.min(arr.length, index + windowSize + 1);
    const window = arr.slice(start, end);
    return window.reduce((a, b) => a + b, 0) / window.length;
  });
};
```

### Component Unmount During Animation

Clean up timeouts to prevent memory leaks and state updates on unmounted components:

```
useEffect(() => {
  const timeouts = [];

  const addTimeout = (fn, delay) => {
    const id = setTimeout(fn, delay);
    timeouts.push(id);
    return id;
  };

  // Use addTimeout instead of setTimeout
  addTimeout(() => {/* ... */}, 1000);

  return () => {
    timeouts.forEach(clearTimeout);
  };
}, []);
```

---

## Implementation Code

### Complete Standalone Example

Here's a complete, copy-paste ready implementation:

```
// MarketCardWithRotation.jsx
import { useState, useEffect, useRef, useCallback } from 'react';

// ============================================
// TIMING CONFIGURATION
```

```
// ============================================
const TIMING = {
  IMAGE_HOLD: 4500,
  CROSSFADE: 450,
  LINE_DRAW: 1500,
  GRAPH_HOLD: 3250,
  TOTAL_CYCLE: 10000,
};

// ============================================
// PRICE GRAPH COMPONENT
// ============================================
const PriceGraph = ({ data, currentPrice, isDrawing, showFill }) => {
  const pathRef = useRef(null);
  const [pathLength, setPathLength] = useState(0);

  const width = 280;
  const height = 140;
  const padding = 12;

  // Generate SVG path
  const generatePath = useCallback(() => {
    if (!data || data.length < 2) return '';

    const graphWidth = width - (padding * 2);
    const graphHeight = height - (padding * 2);
    const xStep = graphWidth / (data.length - 1);

    const points = data.map((value, index) => {
      const x = padding + (index * xStep);
      const y = padding + graphHeight - ((value / 100) * graphHeight);
      return `${x},${y}`;
    });

    return `M ${points.join(' L ')}`;
  }, [data]);

  // Area path for fill
  const generateAreaPath = useCallback(() => {
    const linePath = generatePath();
    if (!linePath) return '';

    const graphHeight = height - (padding * 2);
    const bottom = padding + graphHeight;
    const lastX = width - padding;
    const firstX = padding;
```

```
  return `${linePath} L ${lastX},${bottom} L ${firstX},${bottom} Z`;
}, [generatePath]);

useEffect(() => {
  if (pathRef.current) {
    setPathLength(pathRef.current.getTotalLength());
  }
}, [data]);

const linePath = generatePath();
const areaPath = generateAreaPath();

// Calculate dot position
const lastValue = data?.[data.length - 1] || 50;
const dotY = padding + (height - padding * 2) - ((lastValue / 100) * (height -

return (
  <svg viewBox={`0 0 ${width} ${height}`} style={{ width: '100%', height: '100%
    {/* Subtle grid */}
    <defs>
      <pattern id="graphGrid" width="28" height="28" patternUnits="userSpaceOnU
        <path d="M 28 0 L 0 0 0 28" fill="none" stroke="#e5e7eb" strokeWidth="0
      </pattern>
    </defs>
    <rect width="100%" height="100%" fill="url(#graphGrid)" />

    {/* Area fill */}
    <path
      d={areaPath}
      fill="#3b82f6"
      style={{
        opacity: showFill ? 0.1 : 0,
        transition: 'opacity 0.4s ease',
      }}
    />

    {/* Price line */}
    <path
      ref={pathRef}
      d={linePath}
      fill="none"
      stroke="#3b82f6"
      strokeWidth="2.5"
      strokeLinecap="round"
      strokeLinejoin="round"
      style={{
        strokeDasharray: pathLength,
```

```
          strokeDashoffset: isDrawing ? 0 : pathLength,
          transition: isDrawing ? 'stroke-dashoffset 1.5s ease-out' : 'none',
        }}
      />

      {/* Current price dot */}
      <circle
        cx={width - padding}
        cy={dotY}
        r="5"
        fill="#3b82f6"
        style={{
          opacity: showFill ? 1 : 0,
          transform: showFill ? 'scale(1)' : 'scale(0)',
          transformOrigin: 'center',
          transition: 'all 0.3s ease',
        }}
      />

      {/* Price label */}
      <text
        x={width - padding - 8}
        y={dotY - 10}
        textAnchor="end"
        fontSize="11"
        fontWeight="600"
        fill="#3b82f6"
        style={{
          opacity: showFill ? 1 : 0,
          transition: 'opacity 0.3s ease',
        }}
      >
        {currentPrice}%
      </text>
    </svg>
  );
};

// ============================================
// MARKET CARD VISUAL ROTATION
// ============================================
const MarketCardVisual = ({
  image,
  priceHistory,
  currentPrice,
  initialDelay = 0,
}) => {
```

```
const [showGraph, setShowGraph] = useState(false);
const [isDrawing, setIsDrawing] = useState(false);
const [isPaused, setIsPaused] = useState(false);
const containerRef = useRef(null);
const timeoutsRef = useRef([]);

// Clear all timeouts
const clearTimeouts = () => {
  timeoutsRef.current.forEach(clearTimeout);
  timeoutsRef.current = [];
};

// Add timeout with tracking
const addTimeout = (fn, delay) => {
  const id = setTimeout(fn, delay);
  timeoutsRef.current.push(id);
  return id;
};

// Run one rotation cycle
const runCycle = useCallback(() => {
  // Phase 1: Transition to graph
  setShowGraph(true);
  setIsDrawing(true);

  // Phase 2: Line finishes drawing
  addTimeout(() => {
    setIsDrawing(false);
  }, TIMING.LINE_DRAW);

  // Phase 3: Hold graph, then switch back
  addTimeout(() => {
    setShowGraph(false);
  }, TIMING.LINE_DRAW + TIMING.GRAPH_HOLD);
}, []);

useEffect(() => {
  if (isPaused) return;

  // Initial delay for staggering
  addTimeout(() => {
    runCycle();

    // Set up repeating interval
    const interval = setInterval(() => {
      if (!isPaused) runCycle();
    }, TIMING.TOTAL_CYCLE);
```

```
    timeoutsRef.current.push(interval);
  }, initialDelay + TIMING.IMAGE_HOLD);

  return clearTimeouts;
}, [isPaused, initialDelay, runCycle]);

// Check for reduced motion preference
const prefersReducedMotion =
  typeof window !== 'undefined' &&
  window.matchMedia?.('(prefers-reduced-motion: reduce)').matches;

const containerStyle = {
  position: 'relative',
  aspectRatio: '16 / 9',
  overflow: 'hidden',
  borderRadius: '8px 8px 0 0',
  backgroundColor: '#f3f4f6',
};

const layerStyle = {
  position: 'absolute',
  inset: 0,
  transition: prefersReducedMotion ? 'none' : 'opacity 450ms ease-in-out',
};

return (
  <div
    ref={containerRef}
    style={containerStyle}
    onMouseEnter={() => setIsPaused(true)}
    onMouseLeave={() => setIsPaused(false)}
  >
    {/* Image Layer */}
    <div style={{
      ...layerStyle,
      zIndex: 1,
      opacity: showGraph ? 0 : 1,
    }}>
      <img
        src={image}
        alt=""
        style={{ width: '100%', height: '100%', objectFit: 'cover' }}
      />
    </div>

    {/* Graph Layer */}
```

```
      <div style={{
        ...layerStyle,
        zIndex: 2,
        opacity: showGraph ? 1 : 0,
        backgroundColor: '#ffffff',
        padding: '16px',
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center',
      }}>
        <PriceGraph
          data={priceHistory}
          currentPrice={currentPrice}
          isDrawing={isDrawing}
          showFill={!isDrawing && showGraph}
        />
      </div>

      {/* Indicator Dots */}
      <div style={{
        position: 'absolute',
        bottom: '8px',
        left: '50%',
        transform: 'translateX(-50%)',
        display: 'flex',
        gap: '6px',
        zIndex: 10,
      }}>
        {[false, true].map((isGraphDot, i) => (
          <span
            key={i}
            style={{
              width: '6px',
              height: '6px',
              borderRadius: '50%',
              backgroundColor: 'white',
              opacity: (showGraph === isGraphDot) ? 1 : 0.4,
              transition: 'opacity 0.3s ease',
            }}
          />
        ))}
      </div>
    </div>
  );
};

export default MarketCardVisual;
```

## Acceptance Criteria

### Core Functionality

- Cards automatically rotate between image and graph views
- Rotation cycle completes in ~10 seconds
- Graph line draws from left to right over 1.5 seconds
- Crossfade transitions are smooth (450ms)
- Current price point appears after line draws

### Timing

- Image displays for ~4.5 seconds before transitioning
- Graph displays for ~5 seconds total (1.5s draw + 3.5s hold)
- Cards stagger their start times (2.5s offset between cards)

### Interaction

- Rotation pauses when user hovers over card
- Rotation resumes when user moves mouse away
- Touch devices: tap pauses/plays rotation

### Visual Quality

- Line drawing animation feels organic (ease-out)
- Graph has subtle grid background
- Price dot pulses briefly when appearing
- Area fill fades in after line completes
- Indicator dots show current view state

### Performance

- Animations run at 60fps

- Off-screen cards pause their rotation
- No memory leaks (timeouts cleared on unmount)
- No layout shift during transitions

### Accessibility

- Respects `prefers-reduced-motion` (instant transitions)
- Graph data available as text for screen readers
- Pause control available for users who need it

### Edge Cases

- Handles missing price history (shows image only)
- Handles image load failure (shows fallback)
- Works with varying amounts of price data
- Component cleanup prevents state updates after unmount

---

## Testing Checklist

### Manual Testing

- Watch several full cycles on desktop
- Hover to pause, verify it stops
- Move mouse away, verify it resumes
- Test on mobile (touch to pause)
- Enable reduced motion in OS, verify instant transitions
- Scroll cards in/out of view, verify pause/resume

### Performance Testing

- Profile with React DevTools (no excessive re-renders)
- Profile with Chrome DevTools (60fps animations)
- Test with 20+ cards visible simultaneously

- Test on low-end device / throttled CPU

### Cross-Browser

- Chrome (latest)
- Firefox (latest)
- Safari (latest)
- Edge (latest)
- iOS Safari
- Android Chrome

---

*Document version: 1.0*
*Created: 2026-02-14*
*Feature: Market Card Image ↔ Graph Rotation*