**How It Works Page - Comprehensive UI Analysis**

**1. UI Components & Layout**

**Interactive Elements**

* **Navigation Components**
  + Logo/brand link (routes to home)
  + "Back to Home" link
  + Footer navigation links (Home, Pricing, Contact, Telegram, Legal pages)
* **Content Sections**
  + Hero section (static content)
  + Signal process cards (6 interactive hover cards)
  + Signal scoring cards (4 score category cards)
  + Performance metrics (static display)
  + Risk disclosure expandable section

**React + TypeScript Component Structure**

// HowItWorks.tsx

interface HowItWorksProps {}

interface SignalProcessCard {

id: string;

icon: LucideIcon;

title: string;

description: string;

iconColor: string;

bgColor: string;

}

interface SignalScoreCategory {

id: string;

emoji: string;

title: string;

range: string;

description: string;

gradientFrom: string;

gradientTo: string;

borderColor: string;

}

interface PerformanceMetric {

id: string;

value: string;

label: string;

color: string;

}

// Component breakdown:

- HowItWorksPage (main container)

- Navigation (shared component)

- HeroSection

- SignalProcessSection

- ProcessCard[] (6 cards)

- SignalScoringSection

- ScoreCard[] (4 cards)

- PerformanceSection

- MetricCard[] (3 metrics)

- RiskDisclosure

- Footer (shared component)

**Responsive Design Considerations**

/\* Mobile First Approach \*/

.container {

@apply px-4 sm:px-6 lg:px-8;

}

.grid-responsive {

@apply grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6;

}

.text-responsive {

@apply text-sm sm:text-base lg:text-lg;

}

.hero-title {

@apply text-2xl sm:text-3xl lg:text-4xl;

}

**Loading States & Error Handling**

interface LoadingState {

isLoading: boolean;

error: string | null;

retryCount: number;

}

// Loading skeleton for cards

const CardSkeleton = () => (

<div className="animate-pulse bg-slate-800/30 h-48 rounded-lg" />

);

// Error boundary for resilient UI

const ErrorFallback = ({ error, resetErrorBoundary }) => (

<div className="text-center p-8 bg-red-900/20 rounded-lg">

<AlertTriangle className="h-8 w-8 text-red-400 mx-auto mb-4" />

<p className="text-red-400 mb-4">{error.message}</p>

<button onClick={resetErrorBoundary} className="btn-primary">

Try Again

</button>

</div>

);

**2. State Management (Zustand)**

**Store Structure**

// stores/howItWorksStore.ts

interface HowItWorksStore {

// Page data

processCards: SignalProcessCard[];

scoreCategories: SignalScoreCategory[];

performanceMetrics: PerformanceMetric[];

// UI state

isLoading: boolean;

error: string | null;

activeCard: string | null;

// Actions

setActiveCard: (cardId: string | null) => void;

fetchPageData: () => Promise<void>;

resetError: () => void;

}

// Local vs Global State Decisions

// Global: Performance metrics (shared with dashboard)

// Local: Card hover states, UI interactions

// Global: User preferences, language settings

**State Update Patterns**

const useHowItWorksStore = create<HowItWorksStore>((set, get) => ({

// Initial state

processCards: [],

scoreCategories: [],

performanceMetrics: [],

isLoading: false,

error: null,

activeCard: null,

// Actions

setActiveCard: (cardId) => set({ activeCard: cardId }),

fetchPageData: async () => {

set({ isLoading: true, error: null });

try {

const data = await howItWorksAPI.getPageData();

set({

processCards: data.processCards,

scoreCategories: data.scoreCategories,

performanceMetrics: data.performanceMetrics,

isLoading: false

});

} catch (error) {

set({ error: error.message, isLoading: false });

}

},

resetError: () => set({ error: null }),

}));

**3. API Contracts & Integration**

**API Endpoints**

// API endpoints for page data

interface HowItWorksAPI {

// GET /api/how-it-works/content

getPageContent(): Promise<PageContentResponse>;

// GET /api/performance/metrics

getPerformanceMetrics(): Promise<PerformanceMetricsResponse>;

// GET /api/localization/how-it-works

getLocalizedContent(locale: string): Promise<LocalizedContentResponse>;

}

// Request/Response Schemas

interface PageContentResponse {

processCards: SignalProcessCard[];

scoreCategories: SignalScoreCategory[];

lastUpdated: string;

}

interface PerformanceMetricsResponse {

winRate: number;

tradesAnalyzed: number;

averageROI: number;

lastCalculated: string;

backtestPeriod: {

start: string;

end: string;

};

}

interface LocalizedContentResponse {

translations: Record<string, string>;

locale: string;

}

// Error Response Format

interface APIError {

code: string;

message: string;

details?: Record<string, any>;

timestamp: string;

}

**Error Handling**

// API error handling

const handleAPIError = (error: APIError) => {

switch (error.code) {

case 'CONTENT\_NOT\_FOUND':

return 'Content temporarily unavailable';

case 'PERFORMANCE\_DATA\_STALE':

return 'Performance data is being updated';

default:

return 'Something went wrong. Please try again.';

}

};

**4. Performance & Optimization**

**Lazy Loading Strategies**

// Lazy load heavy sections

const PerformanceSection = lazy(() => import('./PerformanceSection'));

const SignalScoringSection = lazy(() => import('./SignalScoringSection'));

// Intersection Observer for scroll-based loading

const useIntersectionObserver = (threshold = 0.1) => {

const [isVisible, setIsVisible] = useState(false);

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const observer = new IntersectionObserver(

([entry]) => setIsVisible(entry.isIntersecting),

{ threshold }

);

if (ref.current) observer.observe(ref.current);

return () => observer.disconnect();

}, [threshold]);

return { ref, isVisible };

};

**Memoization Opportunities**

// Memoize expensive calculations

const ProcessCard = memo(({ card }: { card: SignalProcessCard }) => {

const iconComponent = useMemo(() => {

const IconComponent = card.icon;

return <IconComponent className={`h-6 w-6 ${card.iconColor}`} />;

}, [card.icon, card.iconColor]);

return (

<Card className="transform transition-transform hover:scale-105">

{iconComponent}

{/\* Rest of card content \*/}

</Card>

);

});

// Memoize performance metrics

const performanceData = useMemo(() => {

return metrics.map(metric => ({

...metric,

formattedValue: formatMetricValue(metric.value, metric.type)

}));

}, [metrics]);

**Bundle Splitting**

// Route-based code splitting

const HowItWorks = lazy(() => import('./pages/HowItWorks'));

// Feature-based splitting

const AdvancedCharts = lazy(() => import('./components/AdvancedCharts'));

**5. Database Schema**

**PostgreSQL Tables**

-- Page content management

CREATE TABLE how\_it\_works\_content (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

section\_type VARCHAR(50) NOT NULL,

content\_key VARCHAR(100) NOT NULL,

content\_value JSONB NOT NULL,

locale VARCHAR(10) DEFAULT 'en',

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

version INTEGER DEFAULT 1,

is\_active BOOLEAN DEFAULT true

);

-- Performance metrics storage

CREATE TABLE performance\_metrics (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

metric\_type VARCHAR(50) NOT NULL,

value DECIMAL(10,4) NOT NULL,

calculation\_date DATE NOT NULL,

data\_period\_start DATE NOT NULL,

data\_period\_end DATE NOT NULL,

trades\_count INTEGER,

metadata JSONB,

created\_at TIMESTAMP DEFAULT NOW()

);

-- Signal scoring categories

CREATE TABLE signal\_score\_categories (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

score\_range\_min INTEGER NOT NULL,

score\_range\_max INTEGER NOT NULL,

category\_name VARCHAR(50) NOT NULL,

description TEXT,

display\_emoji VARCHAR(10),

style\_config JSONB,

is\_active BOOLEAN DEFAULT true

);

-- Indexes for performance

CREATE INDEX idx\_content\_section\_locale ON how\_it\_works\_content(section\_type, locale);

CREATE INDEX idx\_performance\_date ON performance\_metrics(calculation\_date);

CREATE INDEX idx\_score\_range ON signal\_score\_categories(score\_range\_min, score\_range\_max);

**Data Constraints**

-- Constraints

ALTER TABLE performance\_metrics

ADD CONSTRAINT chk\_positive\_value CHECK (value >= 0);

ALTER TABLE signal\_score\_categories

ADD CONSTRAINT chk\_valid\_range CHECK (score\_range\_min <= score\_range\_max);

ALTER TABLE how\_it\_works\_content

ADD CONSTRAINT chk\_valid\_locale CHECK (locale ~ '^[a-z]{2}(-[A-Z]{2})?$');

**6. User Experience**

**Loading States**

// Skeleton screens for different sections

const HeroSkeleton = () => (

<div className="text-center animate-pulse">

<div className="h-8 bg-slate-700 rounded w-3/4 mx-auto mb-4" />

<div className="h-4 bg-slate-700 rounded w-1/2 mx-auto" />

</div>

);

const CardsSkeleton = () => (

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">

{Array.from({ length: 6 }).map((\_, i) => (

<div key={i} className="h-48 bg-slate-800/30 rounded-lg animate-pulse" />

))}

</div>

);

**Accessibility Considerations**

// ARIA labels and keyboard navigation

const ProcessCard = ({ card, isActive, onFocus }) => (

<Card

role="article"

aria-labelledby={`card-title-${card.id}`}

aria-describedby={`card-desc-${card.id}`}

tabIndex={0}

onFocus={() => onFocus(card.id)}

onKeyDown={(e) => {

if (e.key === 'Enter' || e.key === ' ') {

onFocus(card.id);

}

}}

className={`focus:ring-2 focus:ring-blue-500 focus:outline-none ${

isActive ? 'ring-2 ring-blue-400' : ''

}`}

>

<CardTitle id={`card-title-${card.id}`}>{card.title}</CardTitle>

<CardContent id={`card-desc-${card.id}`}>{card.description}</CardContent>

</Card>

);

// Screen reader announcements

const announceToScreenReader = (message: string) => {

const announcement = document.createElement('div');

announcement.setAttribute('aria-live', 'polite');

announcement.setAttribute('aria-atomic', 'true');

announcement.className = 'sr-only';

announcement.textContent = message;

document.body.appendChild(announcement);

setTimeout(() => document.body.removeChild(announcement), 1000);

};

**Animation Requirements**

/\* Smooth transitions for interactive elements \*/

.card-hover {

@apply transform transition-all duration-300 ease-in-out;

}

.card-hover:hover {

@apply scale-105 shadow-2xl;

}

/\* Fade-in animations for sections \*/

.fade-in {

animation: fadeIn 0.6s ease-in-out;

}

@keyframes fadeIn {

from { opacity: 0; transform: translateY(20px); }

to { opacity: 1; transform: translateY(0); }

}

/\* Number counter animation \*/

.counter-animation {

animation: countUp 1.5s ease-out;

}

**7. Integration Points**

**Navigation Patterns**

// Navigation integration

const HowItWorksNavigation = () => {

const navigate = useNavigate();

const { user } = useAuthStore();

return (

<nav className="bg-slate-900/50 backdrop-blur-sm">

<div className="flex justify-between items-center">

<Link to="/" className="logo-container">

<TrendingUp className="h-8 w-8" />

<span>Kurzora</span>

</Link>

<div className="flex items-center space-x-4">

{user && (

<Link to="/dashboard" className="btn-secondary">

Dashboard

</Link>

)}

<Link to="/" className="text-slate-300 hover:text-white">

← Back to Home

</Link>

</div>

</div>

</nav>

);

};

**Shared Components**

// Shared components used across screens

import { Navigation } from '@/components/layout/Navigation';

import { Footer } from '@/components/layout/Footer';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { LoadingSpinner } from '@/components/ui/LoadingSpinner';

// Event handling between components

const usePageAnalytics = () => {

const trackPageView = useCallback(() => {

analytics.track('how\_it\_works\_viewed', {

timestamp: new Date().toISOString(),

user\_agent: navigator.userAgent,

});

}, []);

const trackCardInteraction = useCallback((cardId: string) => {

analytics.track('signal\_card\_viewed', {

card\_id: cardId,

section: 'how\_it\_works',

});

}, []);

return { trackPageView, trackCardInteraction };

};

**8. Testing Strategy**

**Unit Test Requirements**

// Component tests

describe('HowItWorks', () => {

it('renders all signal process cards', () => {

render(<HowItWorks />);

expect(screen.getAllByRole('article')).toHaveLength(6);

});

it('displays performance metrics correctly', () => {

const mockMetrics = {

winRate: 68,

tradesAnalyzed: 180000,

averageROI: 6

};

render(<HowItWorks />, { initialState: { metrics: mockMetrics } });

expect(screen.getByText('68%')).toBeInTheDocument();

expect(screen.getByText('180,000+')).toBeInTheDocument();

});

it('handles keyboard navigation', () => {

render(<HowItWorks />);

const firstCard = screen.getAllByRole('article')[0];

fireEvent.keyDown(firstCard, { key: 'Enter' });

expect(firstCard).toHaveClass('ring-2');

});

});

// Store tests

describe('howItWorksStore', () => {

it('fetches page data successfully', async () => {

const mockAPI = jest.spyOn(howItWorksAPI, 'getPageData');

mockAPI.mockResolvedValue(mockPageData);

await useHowItWorksStore.getState().fetchPageData();

expect(useHowItWorksStore.getState().isLoading).toBe(false);

expect(useHowItWorksStore.getState().processCards).toHaveLength(6);

});

});

**Integration Test Scenarios**

// E2E tests with Playwright

test('User can navigate through how it works page', async ({ page }) => {

await page.goto('/how-it-works');

// Test page load

await expect(page.locator('h1')).toContainText('How Our Signals Work');

// Test card interactions

await page.hover('[data-testid="process-card-0"]');

await expect(page.locator('[data-testid="process-card-0"]')).toHaveClass(/scale-105/);

// Test navigation

await page.click('[data-testid="back-to-home"]');

await expect(page).toHaveURL('/');

});

**Mock Data Structures**

// Mock data for development and testing

export const mockProcessCards: SignalProcessCard[] = [

{

id: 'multi-timeframe',

icon: Layers,

title: 'Multi Time Frame',

description: 'Confirm trends, reduce false signals...',

iconColor: 'text-blue-400',

bgColor: 'bg-blue-500/20'

},

// ... rest of cards

];

export const mockPerformanceMetrics: PerformanceMetric[] = [

{ id: 'win-rate', value: '68%', label: 'Average Win Rate', color: 'text-emerald-400' },

{ id: 'trades', value: '180,000+', label: 'Trades Analyzed', color: 'text-emerald-400' },

{ id: 'roi', value: '6%', label: 'Average ROI per Trade', color: 'text-emerald-400' }

];

**9. Charts & Data Visualizations**

**Performance Metrics Display**

// Animated counter component

const AnimatedCounter = ({ value, duration = 1500 }: { value: number; duration?: number }) => {

const [displayValue, setDisplayValue] = useState(0);

useEffect(() => {

let startTimestamp = 0;

const animate = (timestamp: number) => {

if (!startTimestamp) startTimestamp = timestamp;

const progress = Math.min((timestamp - startTimestamp) / duration, 1);

setDisplayValue(Math.floor(progress \* value));

if (progress < 1) {

requestAnimationFrame(animate);

}

};

requestAnimationFrame(animate);

}, [value, duration]);

return <span>{displayValue}</span>;

};

// Win rate circular progress

const WinRateProgress = ({ percentage }: { percentage: number }) => {

const circumference = 2 \* Math.PI \* 45; // radius = 45

const offset = circumference - (percentage / 100) \* circumference;

return (

<div className="relative w-32 h-32">

<svg className="transform -rotate-90 w-32 h-32">

<circle

cx="64"

cy="64"

r="45"

stroke="rgb(30 41 59)" // slate-800

strokeWidth="8"

fill="transparent"

/>

<circle

cx="64"

cy="64"

r="45"

stroke="rgb(16 185 129)" // emerald-500

strokeWidth="8"

strokeDasharray={circumference}

strokeDashoffset={offset}

strokeLinecap="round"

fill="transparent"

className="transition-all duration-1000 ease-out"

/>

</svg>

<div className="absolute inset-0 flex items-center justify-center">

<span className="text-2xl font-bold text-emerald-400">

<AnimatedCounter value={percentage} />%

</span>

</div>

</div>

);

};

**Chart Configurations**

// Recharts configuration for potential data visualization

const BacktestChart = ({ data }: { data: BacktestData[] }) => (

<ResponsiveContainer width="100%" height={300}>

<LineChart data={data}>

<CartesianGrid strokeDasharray="3 3" stroke="rgb(51 65 85)" />

<XAxis

dataKey="date"

stroke="rgb(148 163 184)"

fontSize={12}

/>

<YAxis

stroke="rgb(148 163 184)"

fontSize={12}

/>

<Tooltip

contentStyle={{

backgroundColor: 'rgb(30 41 59)',

border: '1px solid rgb(51 65 85)',

borderRadius: '8px'

}}

/>

<Line

type="monotone"

dataKey="cumulative\_return"

stroke="rgb(16 185 129)"

strokeWidth={2}

dot={false}

activeDot={{ r: 6 }}

/>

</LineChart>

</ResponsiveContainer>

);

**10. Visual Data Elements**

**Progress Indicators & Counters**

// Score category indicator

const ScoreIndicator = ({ score, category }: { score: number; category: string }) => {

const getColorScheme = (score: number) => {

if (score >= 80) return { bg: 'bg-emerald-500', text: 'text-emerald-400' };

if (score >= 60) return { bg: 'bg-blue-500', text: 'text-blue-400' };

if (score >= 40) return { bg: 'bg-yellow-500', text: 'text-yellow-400' };

return { bg: 'bg-red-500', text: 'text-red-400' };

};

const colors = getColorScheme(score);

return (

<div className="flex items-center space-x-2">

<div className={`w-3 h-3 rounded-full ${colors.bg}`} />

<span className={`font-semibold ${colors.text}`}>

{category} ({score})

</span>

</div>

);

};

// Metric display with icon

const MetricDisplay = ({ metric }: { metric: PerformanceMetric }) => (

<div className="bg-slate-700/50 rounded-lg p-6 text-center">

<div className="text-3xl font-bold text-emerald-400 mb-2">

<AnimatedCounter value={parseFloat(metric.value)} />

{metric.value.includes('%') && '%'}

{metric.value.includes('+') && '+'}

</div>

<div className="text-slate-400 text-sm">{metric.label}</div>

</div>

);

**Typography & Visual Hierarchy**

/\* Typography scale \*/

.hero-title {

@apply text-4xl font-bold text-white mb-4;

letter-spacing: -0.025em;

}

.section-title {

@apply text-2xl font-bold text-white mb-6;

}

.card-title {

@apply text-lg font-semibold text-white;

}

.metric-value {

@apply text-3xl font-bold;

font-variant-numeric: tabular-nums;

}

/\* Visual emphasis patterns \*/

.emphasis-gradient {

background: linear-gradient(135deg, rgb(59 130 246), rgb(147 51 234));

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

background-clip: text;

}

.glow-effect {

box-shadow: 0 0 20px rgba(59, 130, 246, 0.3);

}

**11. Security & Validation**

**Input Validation Schemas**

// Zod schemas for API responses

const SignalProcessCardSchema = z.object({

id: z.string().uuid(),

icon: z.string(),

title: z.string().min(1).max(100),

description: z.string().min(1).max(500),

iconColor: z.string().regex(/^text-\w+-\d+$/),

bgColor: z.string().regex(/^bg-\w+-\d+\/\d+$/)

});

const PerformanceMetricSchema = z.object({

id: z.string(),

value: z.string().regex(/^\d+(\.\d+)?[%+]?$/),

label: z.string().min(1).max(50),

color: z.string().regex(/^text-\w+-\d+$/),

lastUpdated: z.string().datetime()

});

// Validate API responses

const validatePageContent = (data: unknown) => {

const schema = z.object({

processCards: z.array(SignalProcessCardSchema),

performanceMetrics: z.array(PerformanceMetricSchema),

lastUpdated: z.string().datetime()

});

return schema.parse(data);

};

**Authentication & Authorization**

// Route protection (though this page is likely public)

const ProtectedRoute = ({ children }: { children: React.ReactNode }) => {

const { user, isLoading } = useAuthStore();

if (isLoading) return <LoadingSpinner />;

// How It Works is public, but track authenticated users

if (user) {

analytics.identify(user.id, {

plan: user.subscription?.plan,

viewed\_how\_it\_works: true

});

}

return <>{children}</>;

};

**Data Sanitization**

// Sanitize dynamic content

const sanitizeContent = (content: string) => {

return DOMPurify.sanitize(content, {

ALLOWED\_TAGS: ['strong', 'em', 'p', 'br'],

ALLOWED\_ATTR: []

});

};

// Safe content rendering

const SafeContent = ({ content }: { content: string }) => (

<div

dangerouslySetInnerHTML={{

\_\_html: sanitizeContent(content)

}}

/>

);

**12. Environment & Configuration**

**Environment Variables**

# .env.local

NEXT\_PUBLIC\_API\_URL=https://api.kurzora.com

NEXT\_PUBLIC\_WS\_URL=wss://ws.kurzora.com

NEXT\_PUBLIC\_ANALYTICS\_KEY=your\_analytics\_key

NEXT\_PUBLIC\_SENTRY\_DSN=your\_sentry\_dsn

# Performance monitoring

NEXT\_PUBLIC\_PERFORMANCE\_API=https://perf.kurzora.com

NEXT\_PUBLIC\_CDN\_URL=https://cdn.kurzora.com

# Feature flags

NEXT\_PUBLIC\_ENABLE\_ANIMATIONS=true

NEXT\_PUBLIC\_ENABLE\_REAL\_TIME\_METRICS=true

NEXT\_PUBLIC\_SHOW\_ADVANCED\_METRICS=false

**Feature Flags**

// Feature flag configuration

interface FeatureFlags {

enableAnimations: boolean;

enableRealTimeMetrics: boolean;

showAdvancedMetrics: boolean;

enableA11yEnhancements: boolean;

}

const useFeatureFlags = (): FeatureFlags => {

return {

enableAnimations: process.env.NEXT\_PUBLIC\_ENABLE\_ANIMATIONS === 'true',

enableRealTimeMetrics: process.env.NEXT\_PUBLIC\_ENABLE\_REAL\_TIME\_METRICS === 'true',

showAdvancedMetrics: process.env.NEXT\_PUBLIC\_SHOW\_ADVANCED\_METRICS === 'true',

enableA11yEnhancements: true // Always enabled for accessibility

};

};

**Error Reporting Setup**

// Sentry configuration

import \* as Sentry from '@sentry/nextjs';

Sentry.init({

dsn: process.env.NEXT\_PUBLIC\_SENTRY\_DSN,

environment: process.env.NODE\_ENV,

tracesSampleRate: 0.1,

beforeSend(event) {

// Filter out non-critical errors for How It Works page

if (event.tags?.page === 'how-it-works' && event.level === 'warning') {

return null;

}

return event;

}

});

// Performance monitoring

const trackPagePerformance = () => {

if (typeof window !== 'undefined') {

const observer = new PerformanceObserver((list) => {

list.getEntries().forEach((entry) => {

if (entry.entryType === 'largest-contentful-paint') {

analytics.track('lcp\_measured', {

value: entry.startTime,

page: 'how-it-works'

});

}

});

});

observer.observe({ entryTypes: ['largest-contentful-paint'] });

}

};

**13. Cross-Screen Data Flow**

**Data Dependencies**

// Shared performance metrics with dashboard

interface SharedPerformanceData {

winRate: number;

totalTrades: number;

averageROI: number;

lastUpdated: string;

}

// Global store for shared metrics

const usePerformanceStore = create<{

metrics: SharedPerformanceData | null;

fetchMetrics: () => Promise<void>;

subscribeToUpdates: () => void;

}>((set) => ({

metrics: null,

fetchMetrics: async () => {

const data = await performanceAPI.getLatestMetrics();

set({ metrics: data });

},

subscribeToUpdates: () => {

const ws = new WebSocket(process.env.NEXT\_PUBLIC\_WS\_URL + '/performance');

ws.onmessage = (event) => {

const updatedMetrics = JSON.parse(event.data);

set({ metrics: updatedMetrics });

};

}

}));

**Real-time Update Propagation**

// WebSocket integration for live updates

const useRealTimeMetrics = () => {

const { metrics, subscribeToUpdates } = usePerformanceStore();

const [isConnected, setIsConnected] = useState(false);

useEffect(() => {

const handleConnection = () => {

setIsConnected(true);

subscribeToUpdates();

};

const handleDisconnection = () => {

setIsConnected(false);

};

// Connection management

handleConnection();

return () => {

handleDisconnection();

};

}, [subscribeToUpdates]);

return { metrics, isConnected };

};

**Cache Invalidation**

// Cache management for page data

const useCacheManagement = () => {

const queryClient = useQueryClient();

const invalidateHowItWorksCache = useCallback(() => {

queryClient.invalidateQueries(['how-it-works']);

queryClient.invalidateQueries(['performance-metrics']);

}, [queryClient]);

const prefetchRelatedPages = useCallback(() => {

queryClient.prefetchQuery(['pricing'], () => pricingAPI.getPlans());

queryClient.prefetchQuery(['dashboard'], () => dashboardAPI.getOverview());

}, [queryClient]);

return { invalidateHowItWorksCache, prefetchRelatedPages };

};

**Event Handling Between Components**

// Event bus for cross-component communication

const eventBus = new EventTarget();

// Custom hooks for event handling

const usePageEvents = () => {

const emitCardView = useCallback((cardId: string) => {

eventBus.dispatchEvent(new CustomEvent('card-viewed', {

detail: { cardId, timestamp: Date.now() }

}));

}, []);

const emitNavigationIntent = useCallback((destination: string) => {

eventBus.dispatchEvent(new CustomEvent('navigation-intent', {

detail: { destination, source: 'how-it-works' }

}));

}, []);

return { emitCardView, emitNavigationIntent };

};

// Listen for events in other components

const useEventListener = (eventType: string, handler: (event: CustomEvent) => void) => {

useEffect(() => {

eventBus.addEventListener(eventType, handler as EventListener);

return () => eventBus.removeEventListener(eventType, handler as EventListener);

}, [eventType, handler]);

};

**Implementation Priority**

**Phase 1 - Core Structure (Week 1)**

1. Basic component structure and layout
2. Static content rendering
3. Navigation integration
4. Basic responsive design

**Phase 2 - Interactivity (Week 2)**

1. Card hover effects and animations
2. Performance metrics display
3. State management setup
4. API integration

**Phase 3 - Enhancement (Week 3)**

1. Advanced animations and transitions
2. Accessibility improvements
3. Performance optimizations
4. Real-time data updates

**Phase 4 - Polish (Week 4)**

1. Testing implementation
2. Error handling refinement
3. Analytics integration
4. Security hardening

This comprehensive analysis provides immediate implementable specifications for the How It Works page with full consideration of performance, accessibility, and maintainability requirements.