**RTL Layout System Design for Kurzora Trading Platform**

**📋 Document Overview**

**Status:** ✅ Production-Ready Technical Specification  
**Version:** 1.0  
**Implementation Priority:** Phase 3 (Arabic Market Entry)  
**Tech Stack:** Next.js + Tailwind CSS + TypeScript

**🏗️ Core RTL Architecture**

**CSS Custom Properties System**

/\* globals.css - RTL CSS Variables \*/

:root {

/\* Base spacing \*/

--spacing-start: 0rem;

--spacing-end: 1rem;

/\* Layout directions \*/

--text-align-start: left;

--text-align-end: right;

--border-start: border-left;

--border-end: border-right;

/\* Transform directions \*/

--rotate-start: 0deg;

--rotate-end: 180deg;

/\* Flexbox directions \*/

--flex-row-direction: row;

--flex-row-reverse-direction: row-reverse;

}

/\* RTL overrides \*/

[dir="rtl"] {

--spacing-start: 1rem;

--spacing-end: 0rem;

--text-align-start: right;

--text-align-end: left;

--border-start: border-right;

--border-end: border-left;

--rotate-start: 180deg;

--rotate-end: 0deg;

--flex-row-direction: row-reverse;

--flex-row-reverse-direction: row;

}

/\* Typography for Arabic \*/

[dir="rtl"] {

font-family: 'Noto Sans Arabic', 'Cairo', 'Tajawal', system-ui, sans-serif;

letter-spacing: 0;

line-height: 1.8; /\* Increased for Arabic readability \*/

}

**Tailwind CSS RTL Configuration**

// tailwind.config.ts - RTL Extensions

import { Config } from 'tailwindcss'

const config: Config = {

content: ['./src/\*\*/\*.{js,ts,jsx,tsx,mdx}'],

darkMode: 'class',

theme: {

extend: {

fontFamily: {

'arabic': ['Noto Sans Arabic', 'Cairo', 'Tajawal', 'sans-serif'],

'english': ['Inter', 'system-ui', 'sans-serif'],

'german': ['Inter', 'system-ui', 'sans-serif'],

},

spacing: {

'rtl-safe': 'var(--spacing-start)',

'rtl-safe-end': 'var(--spacing-end)',

},

textAlign: {

'start': 'var(--text-align-start)',

'end': 'var(--text-align-end)',

},

colors: {

// Arabic market preferred colors

'arabic-primary': '#1B4332',

'arabic-secondary': '#2D6A4F',

'arabic-accent': '#95D5B2',

'arabic-gold': '#FFD60A',

}

},

},

plugins: [

// Custom RTL plugin

function({ addUtilities, theme, variants }: any) {

const newUtilities = {

'.ps-safe': {

'padding-inline-start': 'var(--spacing-start)',

},

'.pe-safe': {

'padding-inline-end': 'var(--spacing-end)',

},

'.ms-safe': {

'margin-inline-start': 'var(--spacing-start)',

},

'.me-safe': {

'margin-inline-end': 'var(--spacing-end)',

},

'.border-s-safe': {

'border-inline-start': '1px solid',

},

'.border-e-safe': {

'border-inline-end': '1px solid',

},

'.text-start-safe': {

'text-align': 'var(--text-align-start)',

},

'.text-end-safe': {

'text-align': 'var(--text-align-end)',

},

'.rotate-icon-rtl': {

'transform': 'rotate(var(--rotate-start))',

},

'[dir="rtl"] .rotate-icon-rtl': {

'transform': 'rotate(var(--rotate-end))',

}

}

addUtilities(newUtilities, variants('margin'))

}

],

}

export default config

**🎨 Component Design Patterns**

**RTL-Aware Layout Components**

// components/layout/RTLProvider.tsx

'use client'

import { createContext, useContext, useEffect, useState } from 'react'

import { useTranslation } from 'react-i18next'

interface RTLContextType {

isRTL: boolean

direction: 'ltr' | 'rtl'

toggleDirection: () => void

}

const RTLContext = createContext<RTLContextType | null>(null)

export function RTLProvider({ children }: { children: React.ReactNode }) {

const { i18n } = useTranslation()

const [direction, setDirection] = useState<'ltr' | 'rtl'>('ltr')

const isRTL = direction === 'rtl'

useEffect(() => {

const currentLang = i18n.language

const newDirection = currentLang === 'ar' ? 'rtl' : 'ltr'

setDirection(newDirection)

// Update document attributes

document.documentElement.setAttribute('dir', newDirection)

document.documentElement.setAttribute('lang', currentLang)

// Update body classes for Tailwind

document.body.className = document.body.className

.replace(/\b(ltr|rtl)\b/g, '')

.concat(` ${newDirection}`)

}, [i18n.language])

const toggleDirection = () => {

const newDirection = direction === 'ltr' ? 'rtl' : 'ltr'

setDirection(newDirection)

}

return (

<RTLContext.Provider value={{ isRTL, direction, toggleDirection }}>

<div className={`font-${i18n.language === 'ar' ? 'arabic' : 'english'}`}>

{children}

</div>

</RTLContext.Provider>

)

}

export const useRTL = () => {

const context = useContext(RTLContext)

if (!context) {

throw new Error('useRTL must be used within RTLProvider')

}

return context

}

**Trading Dashboard RTL Adaptations**

// components/dashboard/SignalHeatmap.tsx

'use client'

import { useRTL } from '@/components/layout/RTLProvider'

import { useTranslation } from 'react-i18next'

interface SignalHeatmapProps {

signals: TradingSignal[]

className?: string

}

export function SignalHeatmap({ signals, className }: SignalHeatmapProps) {

const { isRTL } = useRTL()

const { t } = useTranslation('trading')

return (

<div className={`

w-full bg-white dark:bg-gray-900 rounded-lg shadow-sm border

${isRTL ? 'font-arabic' : 'font-english'}

${className}

`}>

{/\* Header with RTL-safe alignment \*/}

<div className={`

flex items-center justify-between p-6 border-b

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<h2 className={`

text-xl font-semibold text-gray-900 dark:text-white

${isRTL ? 'text-end-safe' : 'text-start-safe'}

`}>

{t('signal\_heatmap.title')}

</h2>

<div className={`

flex items-center gap-2

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<span className="text-sm text-gray-500">

{t('signal\_heatmap.live\_signals')}

</span>

<div className="w-2 h-2 bg-green-500 rounded-full animate-pulse" />

</div>

</div>

{/\* Signal Grid with RTL-aware layout \*/}

<div className="p-6">

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-4">

{signals.map((signal) => (

<SignalCard

key={signal.id}

signal={signal}

isRTL={isRTL}

/>

))}

</div>

</div>

</div>

)

}

// RTL-aware Signal Card Component

function SignalCard({ signal, isRTL }: { signal: TradingSignal, isRTL: boolean }) {

const { t } = useTranslation('trading')

return (

<div className={`

bg-gray-50 dark:bg-gray-800 rounded-lg p-4 border border-gray-200 dark:border-gray-700

hover:shadow-md transition-shadow duration-200

${isRTL ? 'text-end-safe' : 'text-start-safe'}

`}>

{/\* Signal Header \*/}

<div className={`

flex items-center justify-between mb-3

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<div className={`

flex items-center gap-2

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<span className="font-semibold text-gray-900 dark:text-white">

{signal.ticker}

</span>

<div className={`

px-2 py-1 rounded-full text-xs font-medium

${signal.signalType === 'bullish'

? 'bg-green-100 text-green-800 dark:bg-green-900 dark:text-green-200'

: 'bg-red-100 text-red-800 dark:bg-red-900 dark:text-red-200'

}

`}>

{t(`signal\_types.${signal.signalType}`)}

</div>

</div>

<div className="text-lg font-bold text-blue-600 dark:text-blue-400">

{signal.finalScore}

</div>

</div>

{/\* Price Information \*/}

<div className="space-y-2 text-sm">

<div className={`

flex justify-between items-center

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<span className="text-gray-600 dark:text-gray-400">

{t('signal\_card.entry\_price')}

</span>

<span className="font-medium text-gray-900 dark:text-white">

${signal.riskReward.entryPrice.toFixed(2)}

</span>

</div>

<div className={`

flex justify-between items-center

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<span className="text-gray-600 dark:text-gray-400">

{t('signal\_card.target')}

</span>

<span className="font-medium text-green-600 dark:text-green-400">

${signal.riskReward.takeProfit.toFixed(2)}

</span>

</div>

<div className={`

flex justify-between items-center

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}>

<span className="text-gray-600 dark:text-gray-400">

{t('signal\_card.stop\_loss')}

</span>

<span className="font-medium text-red-600 dark:text-red-400">

${signal.riskReward.stopLoss.toFixed(2)}

</span>

</div>

</div>

{/\* Action Button \*/}

<button className={`

w-full mt-4 px-4 py-2 bg-blue-600 hover:bg-blue-700 text-white

rounded-lg font-medium transition-colors duration-200

${isRTL ? 'font-arabic' : 'font-english'}

`}>

{t('signal\_card.view\_details')}

</button>

</div>

)

}

**📊 Trading Interface RTL Adaptations**

**Chart Layout Modifications**

// components/trading/TradingViewChart.tsx

'use client'

import { useRTL } from '@/components/layout/RTLProvider'

import { useEffect, useRef } from 'react'

interface TradingViewChartProps {

symbol: string

interval: string

theme: 'light' | 'dark'

}

export function TradingViewChart({ symbol, interval, theme }: TradingViewChartProps) {

const { isRTL } = useRTL()

const containerRef = useRef<HTMLDivElement>(null)

useEffect(() => {

if (!containerRef.current) return

const script = document.createElement('script')

script.src = 'https://s3.tradingview.com/external-embedding/embed-widget-advanced-chart.js'

script.type = 'text/javascript'

script.async = true

script.innerHTML = JSON.stringify({

autosize: true,

symbol: symbol,

interval: interval,

timezone: 'Etc/UTC',

theme: theme,

style: '1',

locale: isRTL ? 'ar' : 'en',

enable\_publishing: false,

withdateranges: true,

hide\_side\_toolbar: false,

allow\_symbol\_change: true,

studies: [

'RSI@tv-basicstudies',

'MACD@tv-basicstudies',

'Volume@tv-basicstudies'

],

// RTL-specific configurations

drawings\_access: {

type: 'black',

tools: [

{ name: 'Regression Trend', grayed: false },

{ name: 'Trend Line', grayed: false },

{ name: 'Horizontal Line', grayed: false }

]

},

// Toolbar position adjustment for RTL

toolbar\_bg: isRTL ? '#f8f9fa' : '#ffffff',

details: true,

hotlist: true,

calendar: true,

container\_id: 'tradingview\_chart'

})

containerRef.current.appendChild(script)

return () => {

if (containerRef.current) {

containerRef.current.innerHTML = ''

}

}

}, [symbol, interval, theme, isRTL])

return (

<div className={`

w-full h-[600px] bg-white dark:bg-gray-900 rounded-lg overflow-hidden

${isRTL ? 'rtl-chart-container' : 'ltr-chart-container'}

`}>

<div

ref={containerRef}

id="tradingview\_chart"

className="w-full h-full"

/>

</div>

)

}

**Navigation and Menu Systems**

// components/layout/Navigation.tsx

'use client'

import { useRTL } from '@/components/layout/RTLProvider'

import { useTranslation } from 'react-i18next'

import { ChevronLeftIcon, ChevronRightIcon } from '@heroicons/react/24/outline'

export function Navigation() {

const { isRTL } = useRTL()

const { t } = useTranslation('common')

const ChevronIcon = isRTL ? ChevronLeftIcon : ChevronRightIcon

const navigationItems = [

{ key: 'dashboard', href: '/dashboard', icon: '📊' },

{ key: 'signals', href: '/signals', icon: '⚡' },

{ key: 'portfolio', href: '/portfolio', icon: '💼' },

{ key: 'watchlist', href: '/watchlist', icon: '👁️' },

{ key: 'settings', href: '/settings', icon: '⚙️' },

]

return (

<nav className={`

fixed top-0 h-full w-64 bg-white dark:bg-gray-900 border-gray-200 dark:border-gray-800

${isRTL ? 'right-0 border-l' : 'left-0 border-r'}

${isRTL ? 'font-arabic' : 'font-english'}

`}>

{/\* Logo Section \*/}

<div className={`

p-6 border-b border-gray-200 dark:border-gray-800

${isRTL ? 'text-end-safe' : 'text-start-safe'}

`}>

<h1 className="text-2xl font-bold text-blue-600 dark:text-blue-400">

كورزورا {/\* Kurzora in Arabic \*/}

</h1>

<p className="text-sm text-gray-600 dark:text-gray-400 mt-1">

{t('navigation.platform\_subtitle')}

</p>

</div>

{/\* Navigation Items \*/}

<div className="p-4">

<ul className="space-y-2">

{navigationItems.map((item) => (

<li key={item.key}>

<a

href={item.href}

className={`

flex items-center gap-3 px-4 py-3 rounded-lg

text-gray-700 dark:text-gray-300

hover:bg-gray-100 dark:hover:bg-gray-800

transition-colors duration-200

${isRTL ? 'flex-row-reverse' : 'flex-row'}

`}

>

<span className="text-lg">{item.icon}</span>

<span className="font-medium">

{t(`navigation.${item.key}`)}

</span>

<ChevronIcon className={`

w-4 h-4 text-gray-400

${isRTL ? 'me-auto' : 'ms-auto'}

`} />

</a>

</li>

))}

</ul>

</div>

{/\* Islamic Finance Badge (Arabic users only) \*/}

{isRTL && (

<div className="p-4 border-t border-gray-200 dark:border-gray-800 mt-auto">

<div className="bg-arabic-primary text-white p-3 rounded-lg text-center">

<div className="text-sm font-medium">

{t('islamic\_finance.halal\_certified')}

</div>

<div className="text-xs opacity-80 mt-1">

{t('islamic\_finance.shariah\_compliant')}

</div>

</div>

</div>

)}

</nav>

)

}

**🔧 RTL Typography System**

**Arabic Font Configuration**

/\* Arabic font loading and optimization \*/

@import url('https://fonts.googleapis.com/css2?family=Noto+Sans+Arabic:wght@300;400;500;600;700&display=swap');

@import url('https://fonts.googleapis.com/css2?family=Cairo:wght@300;400;500;600;700&display=swap');

/\* Arabic-specific typography rules \*/

[dir="rtl"] {

/\* Base typography \*/

font-family: 'Noto Sans Arabic', 'Cairo', 'Tajawal', system-ui, sans-serif;

font-feature-settings: 'liga' 1, 'calt' 1, 'kern' 1;

/\* Letter spacing adjustments \*/

letter-spacing: 0;

word-spacing: 0.1em;

/\* Line height for Arabic readability \*/

line-height: 1.8;

/\* Text rendering optimization \*/

text-rendering: optimizeLegibility;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

/\* Arabic number formatting \*/

[dir="rtl"] .number-display {

font-variant-numeric: tabular-nums;

direction: ltr;

unicode-bidi: bidi-override;

}

/\* Arabic form inputs \*/

[dir="rtl"] input,

[dir="rtl"] textarea,

[dir="rtl"] select {

text-align: right;

direction: rtl;

}

/\* Arabic table layouts \*/

[dir="rtl"] table {

direction: rtl;

}

[dir="rtl"] th,

[dir="rtl"] td {

text-align: right;

}

**📱 Responsive RTL Behavior**

**Mobile Adaptations for Arabic**

// hooks/useResponsiveRTL.ts

'use client'

import { useRTL } from '@/components/layout/RTLProvider'

import { useEffect, useState } from 'react'

export function useResponsiveRTL() {

const { isRTL } = useRTL()

const [isMobile, setIsMobile] = useState(false)

useEffect(() => {

const checkMobile = () => {

setIsMobile(window.innerWidth < 768)

}

checkMobile()

window.addEventListener('resize', checkMobile)

return () => window.removeEventListener('resize', checkMobile)

}, [])

return {

isRTL,

isMobile,

// Mobile-specific RTL classes

getFlexDirection: () => {

if (isMobile) {

return 'flex-col'

}

return isRTL ? 'flex-row-reverse' : 'flex-row'

},

// Responsive padding

getResponsivePadding: () => {

if (isMobile) {

return 'px-4'

}

return isRTL ? 'pr-6 pl-4' : 'pl-6 pr-4'

},

// Responsive text alignment

getTextAlignment: () => {

if (isMobile) {

return 'text-center'

}

return isRTL ? 'text-right' : 'text-left'

}

}

}

**🧪 RTL Testing Framework**

**Component Testing Utilities**

// utils/rtl-testing.ts

import { render, RenderOptions } from '@testing-library/react'

import { RTLProvider } from '@/components/layout/RTLProvider'

import { I18nextProvider } from 'react-i18next'

import i18n from '@/lib/i18n'

interface RTLRenderOptions extends RenderOptions {

language?: 'en' | 'de' | 'ar'

direction?: 'ltr' | 'rtl'

}

export function renderWithRTL(

ui: React.ReactElement,

{ language = 'en', direction = 'ltr', ...options }: RTLRenderOptions = {}

) {

// Set language for testing

i18n.changeLanguage(language)

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

<I18nextProvider i18n={i18n}>

<RTLProvider>

<div dir={direction}>

{children}

</div>

</RTLProvider>

</I18nextProvider>

)

return render(ui, { wrapper: Wrapper, ...options })

}

// Test utilities for RTL-specific assertions

export const rtlTestUtils = {

expectRTLLayout: (element: HTMLElement) => {

expect(element).toHaveAttribute('dir', 'rtl')

expect(element).toHaveClass('font-arabic')

},

expectLTRLayout: (element: HTMLElement) => {

expect(element).toHaveAttribute('dir', 'ltr')

expect(element).toHaveClass('font-english')

},

expectArabicTypography: (element: HTMLElement) => {

const styles = window.getComputedStyle(element)

expect(styles.fontFamily).toContain('Noto Sans Arabic')

expect(parseFloat(styles.lineHeight)).toBeGreaterThan(1.6)

}

}

**🚀 Implementation Checklist**

**Phase 1: Foundation (Week 1)**

* [ ] Install Arabic font families (Noto Sans Arabic, Cairo)
* [ ] Set up CSS custom properties for RTL
* [ ] Configure Tailwind CSS RTL utilities
* [ ] Create RTLProvider component
* [ ] Test basic RTL/LTR switching

**Phase 2: Core Components (Week 2)**

* [ ] Adapt Navigation component for RTL
* [ ] Modify Dashboard layout for Arabic
* [ ] Update SignalHeatmap component
* [ ] Test TradingView chart RTL integration
* [ ] Validate form components in RTL

**Phase 3: Advanced Features (Week 3)**

* [ ] Implement responsive RTL behavior
* [ ] Add Arabic typography optimizations
* [ ] Create RTL-specific animations
* [ ] Test mobile RTL experience
* [ ] Performance optimization for Arabic fonts

**Phase 4: Testing & Validation (Week 4)**

* [ ] Write comprehensive RTL tests
* [ ] Arabic language QA testing
* [ ] Accessibility testing for RTL
* [ ] Cross-browser RTL validation
* [ ] Performance benchmarking

**📚 Arabic Typography Guidelines**

**Font Hierarchy for Trading Interface**

/\* Trading-specific Arabic typography \*/

.arabic-trading-title {

font-family: 'Cairo', 'Noto Sans Arabic', sans-serif;

font-weight: 600;

font-size: 1.5rem;

line-height: 1.6;

letter-spacing: 0;

}

.arabic-trading-body {

font-family: 'Noto Sans Arabic', 'Cairo', sans-serif;

font-weight: 400;

font-size: 1rem;

line-height: 1.8;

}

.arabic-trading-caption {

font-family: 'Noto Sans Arabic', sans-serif;

font-weight: 300;

font-size: 0.875rem;

line-height: 1.6;

}

/\* Numbers in Arabic context \*/

.arabic-number-display {

font-family: 'Inter', monospace;

direction: ltr;

display: inline-block;

unicode-bidi: bidi-override;

}

This RTL Layout System Design provides a comprehensive foundation for implementing Arabic language support in the Kurzora trading platform, ensuring cultural appropriateness and technical excellence for the Arabic market expansion.