**FRONTEND DEVELOPMENT GUIDE**

**📋 DOCUMENT STATUS**

* **Status:** ✅ **MASTER IMPLEMENTATION GUIDE**
* **Version:** 1.0
* **Authority:** Single Source of Truth for Frontend Development
* **Location:** 04\_IMPLEMENTATION/
* **Target:** Complete Next.js frontend with multi-language support

**🎯 FRONTEND OVERVIEW**

**Core Architecture:** Next.js 13+ App Router + TypeScript + Tailwind CSS + Supabase Auth

This guide provides step-by-step instructions for building Kurzora's complete frontend, integrating all existing UI specifications with multi-language support (EN/DE/AR) and RTL layout capabilities.

**Technology Stack:**

* **Framework:** Next.js 13+ with App Router
* **Language:** TypeScript (strict mode)
* **Styling:** Tailwind CSS with RTL support
* **Authentication:** Supabase Auth integration
* **Internationalization:** next-intl for EN/DE/AR
* **Charts:** TradingView Widget integration
* **State Management:** Zustand for client state
* **Forms:** React Hook Form + Zod validation
* **UI Components:** Custom components + Lucide React icons

**📁 PROJECT STRUCTURE SETUP**

**Step 1: Create Next.js Project**

# 1. Create project with all required flags

npx create-next-app@latest kurzora-platform --typescript --tailwind --app --src-dir --import-alias "@/\*"

# 2. Navigate to project

cd kurzora-platform

# 3. Install required dependencies

npm install @supabase/supabase-js @supabase/auth-helpers-nextjs

npm install next-intl zustand react-hook-form @hookform/resolvers zod

npm install lucide-react recharts framer-motion react-hot-toast

npm install date-fns clsx tailwind-merge

npm install @radix-ui/react-dialog @radix-ui/react-dropdown-menu

npm install @radix-ui/react-select @radix-ui/react-switch

npm install @radix-ui/react-toast @radix-ui/react-tabs

# 4. Install development dependencies

npm install -D @types/node @types/react @types/react-dom

npm install -D eslint eslint-config-next prettier

**Step 2: Configure Project Structure**

Create this exact folder structure:

src/

├── app/ # Next.js 13 App Router

│ ├── [locale]/ # Internationalization wrapper

│ │ ├── (auth)/ # Auth route group

│ │ │ ├── login/ # Login page

│ │ │ └── register/ # Registration page

│ │ ├── (dashboard)/ # Protected route group

│ │ │ ├── dashboard/ # Main dashboard

│ │ │ ├── signals/ # Signal pages

│ │ │ ├── portfolio/ # Paper trading

│ │ │ └── settings/ # User settings

│ │ ├── layout.tsx # Root layout with i18n

│ │ ├── page.tsx # Landing page

│ │ └── loading.tsx # Global loading component

│ ├── api/ # API routes

│ │ ├── auth/ # Authentication endpoints

│ │ └── webhooks/ # Webhook handlers

│ ├── globals.css # Global styles with RTL

│ └── layout.tsx # App-level layout

├── components/ # Reusable components

│ ├── ui/ # Base UI components

│ ├── auth/ # Authentication components

│ ├── dashboard/ # Dashboard-specific components

│ ├── signals/ # Signal-related components

│ ├── charts/ # Chart components

│ └── layout/ # Layout components

├── lib/ # Utilities and configurations

│ ├── supabase/ # Supabase configuration

│ ├── auth/ # Authentication utilities

│ ├── i18n/ # Internationalization setup

│ ├── utils/ # General utilities

│ └── validations/ # Zod schemas

├── hooks/ # Custom React hooks

├── stores/ # Zustand stores

├── styles/ # Additional styles

└── types/ # TypeScript type definitions

**🌍 INTERNATIONALIZATION SETUP**

**Step 3: Configure next-intl**

**File: src/lib/i18n/config.ts**

import { notFound } from 'next/navigation'

import { getRequestConfig } from 'next-intl/server'

// Supported locales

export const locales = ['en', 'de', 'ar'] as const

export type Locale = typeof locales[number]

// Default locale

export const defaultLocale: Locale = 'en'

// RTL languages

export const rtlLocales: Locale[] = ['ar']

export default getRequestConfig(async ({ locale }) => {

// Validate that the incoming `locale` parameter is valid

if (!locales.includes(locale as Locale)) notFound()

return {

messages: (await import(`../../messages/${locale}.json`)).default

}

})

**File: next.config.js**

const withNextIntl = require('next-intl/plugin')(

'./src/lib/i18n/config.ts'

)

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

experimental: {

serverComponentsExternalPackages: ['@supabase/auth-helpers-nextjs']

}

}

module.exports = withNextIntl(nextConfig)

**File: middleware.ts**

import createMiddleware from 'next-intl/middleware'

import { locales, defaultLocale } from './src/lib/i18n/config'

export default createMiddleware({

locales,

defaultLocale,

localePrefix: 'always'

})

export const config = {

// Match only internationalized pathnames

matcher: ['/', '/(de|ar|en)/:path\*']

}

**Step 4: Create Translation Files**

**File: src/messages/en.json**

{

"common": {

"loading": "Loading...",

"error": "Error",

"success": "Success",

"cancel": "Cancel",

"save": "Save",

"delete": "Delete",

"edit": "Edit",

"close": "Close"

},

"auth": {

"signIn": "Sign In",

"signUp": "Sign Up",

"signOut": "Sign Out",

"email": "Email",

"password": "Password",

"confirmPassword": "Confirm Password",

"name": "Full Name",

"forgotPassword": "Forgot Password?",

"createAccount": "Create Account",

"alreadyHaveAccount": "Already have an account?",

"dontHaveAccount": "Don't have an account?"

},

"dashboard": {

"title": "Trading Dashboard",

"signals": "Signals",

"portfolio": "Portfolio",

"settings": "Settings",

"signalHeatmap": "Signal Heatmap",

"topSignals": "Top Signals",

"recentActivity": "Recent Activity"

},

"signals": {

"score": "Score",

"type": "Type",

"entryPrice": "Entry Price",

"strength": "Strength",

"bullish": "Bullish",

"bearish": "Bearish",

"confidence": "Confidence",

"timeframe": "Timeframe"

}

}

**File: src/messages/de.json** (German translations) **File: src/messages/ar.json** (Arabic translations)

**🎨 TAILWIND CSS & RTL CONFIGURATION**

**Step 5: Configure Tailwind with RTL Support**

**File: tailwind.config.ts**

import type { Config } from 'tailwindcss'

const config: Config = {

content: [

'./src/pages/\*\*/\*.{js,ts,jsx,tsx,mdx}',

'./src/components/\*\*/\*.{js,ts,jsx,tsx,mdx}',

'./src/app/\*\*/\*.{js,ts,jsx,tsx,mdx}',

],

theme: {

extend: {

colors: {

primary: {

50: '#f0f9ff',

100: '#e0f2fe',

500: '#0ea5e9',

600: '#0284c7',

700: '#0369a1',

900: '#0c4a6e',

},

success: {

50: '#f0fdf4',

500: '#22c55e',

600: '#16a34a',

},

danger: {

50: '#fef2f2',

500: '#ef4444',

600: '#dc2626',

}

},

fontFamily: {

sans: ['Inter', 'sans-serif'],

arabic: ['Cairo', 'sans-serif'],

},

spacing: {

'18': '4.5rem',

'88': '22rem',

}

},

},

plugins: [

require('@tailwindcss/forms'),

require('@tailwindcss/typography'),

// Custom plugin for RTL support

function({ addUtilities }: any) {

const newUtilities = {

'.rtl': {

direction: 'rtl',

},

'.ltr': {

direction: 'ltr',

},

'.start-0': {

'inset-inline-start': '0px',

},

'.end-0': {

'inset-inline-end': '0px',

},

'.ms-auto': {

'margin-inline-start': 'auto',

},

'.me-auto': {

'margin-inline-end': 'auto',

},

'.ps-4': {

'padding-inline-start': '1rem',

},

'.pe-4': {

'padding-inline-end': '1rem',

},

'.border-s': {

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

},

'.border-e': {

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

},

}

addUtilities(newUtilities)

}

],

}

export default config

**File: src/app/globals.css**

@tailwind base;

@tailwind components;

@tailwind utilities;

/\* Import fonts \*/

@import url('https://fonts.googleapis.com/css2?family=Inter: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');

@layer base {

html {

font-family: 'Inter', sans-serif;

}

/\* Arabic font for Arabic locale \*/

html[lang="ar"] {

font-family: 'Cairo', sans-serif;

}

/\* RTL support \*/

[dir="rtl"] {

direction: rtl;

}

/\* Custom scrollbar \*/

::-webkit-scrollbar {

width: 6px;

height: 6px;

}

::-webkit-scrollbar-track {

background: #f1f5f9;

}

::-webkit-scrollbar-thumb {

background: #cbd5e1;

border-radius: 3px;

}

::-webkit-scrollbar-thumb:hover {

background: #94a3b8;

}

}

@layer components {

/\* Custom component styles \*/

.btn-primary {

@apply bg-primary-600 hover:bg-primary-700 text-white font-medium py-2 px-4 rounded-lg transition-colors duration-200;

}

.btn-secondary {

@apply bg-gray-100 hover:bg-gray-200 text-gray-900 font-medium py-2 px-4 rounded-lg transition-colors duration-200;

}

.card {

@apply bg-white rounded-xl shadow-sm border border-gray-200 p-6;

}

.input-field {

@apply w-full px-3 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-primary-500 focus:border-transparent;

}

/\* RTL-aware utilities \*/

.rtl-flip {

transform: scaleX(-1);

}

[dir="rtl"] .rtl-flip {

transform: scaleX(1);

}

}

**🔐 SUPABASE AUTH INTEGRATION**

**Step 6: Configure Supabase Client**

**File: src/lib/supabase/client.ts**

import { createBrowserClient } from '@supabase/ssr'

export function createClient() {

return createBrowserClient(

process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

)

}

**File: src/lib/supabase/server.ts**

import { createServerClient, type CookieOptions } from '@supabase/ssr'

import { cookies } from 'next/headers'

export function createClient() {

const cookieStore = cookies()

return createServerClient(

process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!,

{

cookies: {

get(name: string) {

return cookieStore.get(name)?.value

},

set(name: string, value: string, options: CookieOptions) {

try {

cookieStore.set({ name, value, ...options })

} catch (error) {

// The `set` method was called from a Server Component.

// This can be ignored if you have middleware refreshing

// user sessions.

}

},

remove(name: string, options: CookieOptions) {

try {

cookieStore.set({ name, value: '', ...options })

} catch (error) {

// The `delete` method was called from a Server Component.

// This can be ignored if you have middleware refreshing

// user sessions.

}

},

},

}

)

}

**Step 7: Create Authentication Context**

**File: src/contexts/AuthContext.tsx**

'use client'

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

import { User, Session } from '@supabase/supabase-js'

import { createClient } from '@/lib/supabase/client'

import { useRouter } from 'next/navigation'

interface AuthContextType {

user: User | null

session: Session | null

loading: boolean

signIn: (email: string, password: string) => Promise<{ error?: string }>

signUp: (email: string, password: string, name: string) => Promise<{ error?: string }>

signOut: () => Promise<void>

updateProfile: (updates: { name?: string }) => Promise<{ error?: string }>

}

const AuthContext = createContext<AuthContextType>({} as AuthContextType)

export const useAuth = () => {

const context = useContext(AuthContext)

if (!context) {

throw new Error('useAuth must be used within an AuthProvider')

}

return context

}

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {

const [user, setUser] = useState<User | null>(null)

const [session, setSession] = useState<Session | null>(null)

const [loading, setLoading] = useState(true)

const router = useRouter()

const supabase = createClient()

useEffect(() => {

// Get initial session

const getInitialSession = async () => {

const { data: { session } } = await supabase.auth.getSession()

setSession(session)

setUser(session?.user ?? null)

setLoading(false)

}

getInitialSession()

// Listen for auth changes

const { data: { subscription } } = supabase.auth.onAuthStateChange(

async (event, session) => {

setSession(session)

setUser(session?.user ?? null)

setLoading(false)

// Handle auth events

if (event === 'SIGNED\_IN') {

router.push('/dashboard')

} else if (event === 'SIGNED\_OUT') {

router.push('/')

}

}

)

return () => subscription.unsubscribe()

}, [router])

const signIn = async (email: string, password: string) => {

try {

const { error } = await supabase.auth.signInWithPassword({

email,

password,

})

if (error) {

return { error: error.message }

}

return {}

} catch (error) {

return { error: 'An unexpected error occurred' }

}

}

const signUp = async (email: string, password: string, name: string) => {

try {

const { data, error } = await supabase.auth.signUp({

email,

password,

options: {

data: {

name: name,

},

},

})

if (error) {

return { error: error.message }

}

return {}

} catch (error) {

return { error: 'An unexpected error occurred' }

}

}

const signOut = async () => {

await supabase.auth.signOut()

}

const updateProfile = async (updates: { name?: string }) => {

try {

if (!user) return { error: 'No user logged in' }

const { error } = await supabase.auth.updateUser({

data: updates,

})

if (error) {

return { error: error.message }

}

return {}

} catch (error) {

return { error: 'An unexpected error occurred' }

}

}

const value = {

user,

session,

loading,

signIn,

signUp,

signOut,

updateProfile,

}

return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>

}

**🏗️ CORE COMPONENT ARCHITECTURE**

**Step 8: Create Base Layout Components**

**File: src/app/layout.tsx** (Root layout)

import type { Metadata } from 'next'

import { Inter } from 'next/font/google'

import './globals.css'

const inter = Inter({ subsets: ['latin'] })

export const metadata: Metadata = {

title: 'Kurzora - Trading Signals Platform',

description: 'Professional trading signals with multi-language support',

}

export default function RootLayout({

children,

}: {

children: React.ReactNode

}) {

return (

<html>

<body className={inter.className}>

{children}

</body>

</html>

)

}

**File: src/app/[locale]/layout.tsx** (Internationalized layout)

import { NextIntlClientProvider } from 'next-intl'

import { getMessages } from 'next-intl/server'

import { AuthProvider } from '@/contexts/AuthContext'

import { Toaster } from 'react-hot-toast'

import { rtlLocales, type Locale } from '@/lib/i18n/config'

export default async function LocaleLayout({

children,

params: { locale }

}: {

children: React.ReactNode

params: { locale: Locale }

}) {

const messages = await getMessages()

const isRtl = rtlLocales.includes(locale)

return (

<html lang={locale} dir={isRtl ? 'rtl' : 'ltr'}>

<body className={isRtl ? 'font-arabic' : 'font-sans'}>

<NextIntlClientProvider messages={messages}>

<AuthProvider>

{children}

<Toaster

position={isRtl ? 'top-left' : 'top-right'}

toastOptions={{

className: isRtl ? 'rtl' : 'ltr'

}}

/>

</AuthProvider>

</NextIntlClientProvider>

</body>

</html>

)

}

**Step 9: Create UI Components**

**File: src/components/ui/Button.tsx**

import React from 'react'

import { clsx } from 'clsx'

import { Loader2 } from 'lucide-react'

interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {

variant?: 'primary' | 'secondary' | 'danger' | 'ghost'

size?: 'sm' | 'md' | 'lg'

loading?: boolean

children: React.ReactNode

}

export const Button: React.FC<ButtonProps> = ({

variant = 'primary',

size = 'md',

loading = false,

className,

children,

disabled,

...props

}) => {

const baseClasses = 'inline-flex items-center justify-center font-medium transition-colors duration-200 rounded-lg focus:outline-none focus:ring-2 focus:ring-offset-2'

const variantClasses = {

primary: 'bg-primary-600 hover:bg-primary-700 text-white focus:ring-primary-500',

secondary: 'bg-gray-100 hover:bg-gray-200 text-gray-900 focus:ring-gray-500',

danger: 'bg-danger-600 hover:bg-danger-700 text-white focus:ring-danger-500',

ghost: 'hover:bg-gray-100 text-gray-700 focus:ring-gray-500'

}

const sizeClasses = {

sm: 'text-sm px-3 py-1.5',

md: 'text-sm px-4 py-2',

lg: 'text-base px-6 py-3'

}

return (

<button

className={clsx(

baseClasses,

variantClasses[variant],

sizeClasses[size],

(disabled || loading) && 'opacity-50 cursor-not-allowed',

className

)}

disabled={disabled || loading}

{...props}

>

{loading && <Loader2 className="w-4 h-4 animate-spin me-2" />}

{children}

</button>

)

}

**File: src/components/ui/Input.tsx**

import React from 'react'

import { clsx } from 'clsx'

interface InputProps extends React.InputHTMLAttributes<HTMLInputElement> {

label?: string

error?: string

icon?: React.ReactNode

}

export const Input: React.FC<InputProps> = ({

label,

error,

icon,

className,

...props

}) => {

return (

<div className="space-y-1">

{label && (

<label className="block text-sm font-medium text-gray-700">

{label}

</label>

)}

<div className="relative">

{icon && (

<div className="absolute inset-y-0 start-0 flex items-center ps-3 pointer-events-none">

{icon}

</div>

)}

<input

className={clsx(

'w-full px-3 py-2 border rounded-lg focus:ring-2 focus:ring-primary-500 focus:border-transparent transition-colors duration-200',

icon && 'ps-10',

error

? 'border-danger-300 focus:ring-danger-500'

: 'border-gray-300',

className

)}

{...props}

/>

</div>

{error && (

<p className="text-sm text-danger-600">{error}</p>

)}

</div>

)

}

**📱 PAGE IMPLEMENTATIONS**

**Step 10: Landing Page Implementation**

**File: src/app/[locale]/page.tsx**

import { useTranslations } from 'next-intl'

import { Button } from '@/components/ui/Button'

import Link from 'next/link'

export default function LandingPage() {

const t = useTranslations('landing')

return (

<div className="min-h-screen bg-gradient-to-br from-primary-50 to-primary-100">

{/\* Navigation \*/}

<nav className="border-b border-primary-200 bg-white/80 backdrop-blur-sm">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<div className="flex justify-between items-center h-16">

<div className="flex items-center">

<h1 className="text-2xl font-bold text-primary-900">Kurzora</h1>

</div>

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

<Link href="/login">

<Button variant="ghost">{t('signIn')}</Button>

</Link>

<Link href="/register">

<Button>{t('getStarted')}</Button>

</Link>

</div>

</div>

</div>

</nav>

{/\* Hero Section \*/}

<main className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-20">

<div className="text-center space-y-8">

<h1 className="text-4xl sm:text-6xl font-bold text-gray-900">

{t('heroTitle')}

</h1>

<p className="text-xl text-gray-600 max-w-3xl mx-auto">

{t('heroDescription')}

</p>

<div className="flex flex-col sm:flex-row gap-4 justify-center">

<Link href="/register">

<Button size="lg" className="w-full sm:w-auto">

{t('startFreeTrial')}

</Button>

</Link>

<Button variant="secondary" size="lg" className="w-full sm:w-auto">

{t('watchDemo')}

</Button>

</div>

</div>

{/\* Features Grid \*/}

<div className="mt-20 grid grid-cols-1 md:grid-cols-3 gap-8">

<div className="text-center space-y-4">

<div className="w-16 h-16 bg-primary-100 rounded-full flex items-center justify-center mx-auto">

<span className="text-2xl">📊</span>

</div>

<h3 className="text-xl font-semibold">{t('feature1Title')}</h3>

<p className="text-gray-600">{t('feature1Description')}</p>

</div>

<div className="text-center space-y-4">

<div className="w-16 h-16 bg-primary-100 rounded-full flex items-center justify-center mx-auto">

<span className="text-2xl">🎯</span>

</div>

<h3 className="text-xl font-semibold">{t('feature2Title')}</h3>

<p className="text-gray-600">{t('feature2Description')}</p>

</div>

<div className="text-center space-y-4">

<div className="w-16 h-16 bg-primary-100 rounded-full flex items-center justify-center mx-auto">

<span className="text-2xl">🌍</span>

</div>

<h3 className="text-xl font-semibold">{t('feature3Title')}</h3>

<p className="text-gray-600">{t('feature3Description')}</p>

</div>

</div>

</main>

</div>

)

}

**Step 11: Authentication Pages**

**File: src/app/[locale]/(auth)/login/page.tsx**

'use client'

import { useState } from 'react'

import { useTranslations } from 'next-intl'

import { useAuth } from '@/contexts/AuthContext'

import { Button } from '@/components/ui/Button'

import { Input } from '@/components/ui/Input'

import { Mail, Lock } from 'lucide-react'

import Link from 'next/link'

import { toast } from 'react-hot-toast'

export default function LoginPage() {

const t = useTranslations('auth')

const { signIn } = useAuth()

const [loading, setLoading] = useState(false)

const [formData, setFormData] = useState({

email: '',

password: ''

})

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault()

setLoading(true)

try {

const { error } = await signIn(formData.email, formData.password)

if (error) {

toast.error(error)

} else {

toast.success(t('signInSuccess'))

}

} catch (error) {

toast.error(t('signInError'))

} finally {

setLoading(false)

}

}

return (

<div className="min-h-screen flex items-center justify-center bg-gray-50 py-12 px-4 sm:px-6 lg:px-8">

<div className="max-w-md w-full space-y-8">

<div className="text-center">

<h2 className="text-3xl font-bold text-gray-900">

{t('signIn')}

</h2>

<p className="mt-2 text-gray-600">

{t('signInDescription')}

</p>

</div>

<form className="mt-8 space-y-6" onSubmit={handleSubmit}>

<div className="space-y-4">

<Input

label={t('email')}

type="email"

value={formData.email}

onChange={(e) => setFormData(prev => ({ ...prev, email: e.target.value }))}

icon={<Mail className="w-5 h-5 text-gray-400" />}

required

/>

<Input

label={t('password')}

type="password"

value={formData.password}

onChange={(e) => setFormData(prev => ({ ...prev, password: e.target.value }))}

icon={<Lock className="w-5 h-5 text-gray-400" />}

required

/>

</div>

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

<Link href="/forgot-password" className="text-sm text-primary-600 hover:text-primary-500">

{t('forgotPassword')}

</Link>

</div>

<Button

type="submit"

loading={loading}

className="w-full"

size="lg"

>

{t('signIn')}

</Button>

<div className="text-center">

<span className="text-sm text-gray-600">

{t('dontHaveAccount')}{' '}

<Link href="/register" className="text-primary-600 hover:text-primary-500">

{t('signUp')}

</Link>

</span>

</div>

</form>

</div>

</div>

)

}

**📊 DASHBOARD IMPLEMENTATION**

**Step 12: Dashboard Layout**

**File: src/app/[locale]/(dashboard)/layout.tsx**

import { createClient } from '@/lib/supabase/server'

import { redirect } from 'next/navigation'

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

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

export default async function DashboardLayout({

children,

}: {

children: React.ReactNode

}) {

const supabase = createClient()

const {

data: { user },

} = await supabase.auth.getUser()

if (!user) {

redirect('/login')

}

return (

<div className="min-h-screen bg-gray-50">

<DashboardSidebar />

<div className="lg:ps-64">

<DashboardHeader />

<main className="p-4 lg:p-8">

{children}

</main>

</div>

</div>

)

}

**Step 13: Dashboard Components**

**File: src/components/layout/DashboardSidebar.tsx**

'use client'

import { useTranslations } from 'next-intl'

import { useAuth } from '@/contexts/AuthContext'

import { Button } from '@/components/ui/Button'

import { BarChart3, Target, Settings, LogOut } from 'lucide-react'

import Link from 'next/link'

import { usePathname } from 'next/navigation'

import { clsx } from 'clsx'

export const DashboardSidebar: React.FC = () => {

const t = useTranslations('dashboard')

const { signOut } = useAuth()

const pathname = usePathname()

const navigation = [

{ name: t('dashboard'), href: '/dashboard', icon: BarChart3 },

{ name: t('signals'), href: '/signals', icon: Target },

{ name: t('portfolio'), href: '/portfolio', icon: BarChart3 },

{ name: t('settings'), href: '/settings', icon: Settings },

]

return (

<div className="fixed inset-y-0 start-0 z-50 w-64 bg-white border-e border-gray-200 lg:translate-x-0">

<div className="flex flex-col h-full">

{/\* Logo \*/}

<div className="flex items-center h-16 px-6 border-b border-gray-200">

<h1 className="text-2xl font-bold text-primary-900">Kurzora</h1>

</div>

{/\* Navigation \*/}

<nav className="flex-1 px-6 py-4 space-y-2">

{navigation.map((item) => {

const Icon = item.icon

const isActive = pathname.includes(item.href)

return (

<Link

key={item.name}

href={item.href}

className={clsx(

'flex items-center px-3 py-2 rounded-lg text-sm font-medium transition-colors',

isActive

? 'bg-primary-100 text-primary-700'

: 'text-gray-600 hover:bg-gray-100 hover:text-gray-900'

)}

>

<Icon className="w-5 h-5 me-3" />

{item.name}

</Link>

)

})}

</nav>

{/\* Sign Out \*/}

<div className="p-6 border-t border-gray-200">

<Button

variant="ghost"

onClick={signOut}

className="w-full justify-start"

>

<LogOut className="w-5 h-5 me-3" />

{t('signOut')}

</Button>

</div>

</div>

</div>

)

}

**🚀 IMPLEMENTATION CHECKLIST**

**✅ Phase 1: Foundation (Complete these first)**

* [ ] Create Next.js project with TypeScript and Tailwind
* [ ] Set up internationalization (next-intl) for EN/DE/AR
* [ ] Configure RTL support in Tailwind CSS
* [ ] Set up Supabase client and authentication
* [ ] Create AuthContext and AuthProvider
* [ ] Build base UI components (Button, Input, etc.)
* [ ] Implement root and locale layouts

**✅ Phase 2: Core Pages**

* [ ] Landing page with hero section and features
* [ ] Authentication pages (login, register)
* [ ] Dashboard layout with sidebar and header
* [ ] Dashboard main page with signal heatmap
* [ ] Signal detail pages
* [ ] User settings page

**✅ Phase 3: Advanced Features**

* [ ] Paper trading portfolio components
* [ ] Real-time signal updates
* [ ] Chart integration (TradingView)
* [ ] Notification system
* [ ] Performance optimization
* [ ] Mobile responsiveness testing

**✅ Phase 4: Multi-Language Polish**

* [ ] Complete all translation files
* [ ] Test RTL layout thoroughly
* [ ] Validate Arabic font rendering
* [ ] Test all flows in all languages
* [ ] Optimize for performance

**🔧 DEVELOPMENT TIPS**

**RTL Development Best Practices:**

1. Use logical properties (start/end instead of left/right)
2. Test all layouts in RTL mode regularly
3. Use the rtl-flip utility class for icons that need mirroring
4. Ensure proper text alignment in RTL languages

**Multi-Language Testing:**

1. Test all user flows in all three languages
2. Verify proper font loading for Arabic
3. Check text overflow and layout breaks
4. Validate form validation messages

**Performance Optimization:**

1. Use Next.js Image component for all images
2. Implement proper loading states
3. Optimize bundle size with dynamic imports
4. Use React.memo for expensive components

**TypeScript Best Practices:**

1. Define proper interfaces for all data structures
2. Use strict type checking
3. Implement proper error boundaries
4. Create reusable type definitions

**🎯 This guide provides the complete foundation for building Kurzora's frontend. Follow each step sequentially for best results, and reference the existing UI documentation for specific component implementations.**