**🚀 KURZORA DEVELOPMENT STEP-BY-STEP GUIDE**

**📋 DOCUMENT STATUS**

* **Status:** ✅ **MASTER IMPLEMENTATION ROADMAP**
* **Version:** 1.0
* **Authority:** Single Source of Truth for Development Execution
* **Target:** Phase 1 Launch (500 S&P stocks, Multi-language support)

**🎯 DEVELOPMENT OVERVIEW**

**Core Principle:** Start Small → Prove Concept → Scale with Revenue

This guide transforms your 60+ planning documents into actionable development steps for building Kurzora's international trading platform with English, German, and Arabic support.

**Total Development Timeline:** 10-12 weeks to Phase 1 launch **Budget:** $149/month operational costs **Success Metric:** 50 paying users by month 6

**📋 PRE-DEVELOPMENT CHECKLIST**

**✅ FOUNDATION REQUIREMENTS**

* [ ] **Supabase Account:** Database operational with complete schema
* [ ] **Environment Variables:** .env.local configured with all API keys
* [ ] **Cursor IDE:** Installed with access to all 60+ project documents
* [ ] **Node.js:** Version 18+ installed
* [ ] **Git Repository:** Initialized for version control
* [ ] **Development Machine:** 8GB+ RAM for smooth development

**✅ ACCOUNT PREREQUISITES**

* [ ] **Polygon.io:** Basic plan ($99/month) for market data
* [ ] **Firebase:** Project created for Cloud Functions
* [ ] **Vercel:** Account for frontend deployment
* [ ] **Make.com:** Account for automation workflows
* [ ] **SendGrid:** Account for email notifications
* [ ] **Stripe:** Account for payment processing

**🏗️ PHASE 1: FOUNDATION SETUP (Weeks 1-2)**

**Week 1: Project Initialization**

**Day 1-2: Environment Setup**

# 1. Create Next.js project

npx create-next-app@latest kurzora-platform --typescript --tailwind --app

# 2. Navigate and install dependencies

cd kurzora-platform

npm install @supabase/supabase-js @stripe/stripe-js @stripe/react-stripe-js

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

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

npm install date-fns clsx lodash @types/lodash

npm install next-intl # For internationalization

**Day 3-4: Database Configuration**

**Reference Documents:** Master DB Schema.docx, i18n Architecture.docx

1. **Update Supabase Schema for Multi-language:**

-- Add translation tables

CREATE TABLE content\_translations (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

table\_name VARCHAR(50) NOT NULL,

record\_id UUID NOT NULL,

field\_name VARCHAR(50) NOT NULL,

language\_code VARCHAR(5) NOT NULL,

translated\_content TEXT NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(table\_name, record\_id, field\_name, language\_code)

);

-- Add user language preferences

ALTER TABLE users ADD COLUMN preferred\_language VARCHAR(5) DEFAULT 'en';

ALTER TABLE users ADD COLUMN timezone VARCHAR(50) DEFAULT 'UTC';

ALTER TABLE users ADD COLUMN region VARCHAR(10) DEFAULT 'US';

-- Add Islamic finance compliance

ALTER TABLE stock\_universe ADD COLUMN is\_islamic\_compliant BOOLEAN DEFAULT false;

ALTER TABLE stock\_universe ADD COLUMN compliance\_last\_checked TIMESTAMP WITH TIME ZONE;

1. **Configure Row Level Security for i18n:**

-- Content translations policies

CREATE POLICY content\_translations\_public\_read ON content\_translations

FOR SELECT USING (true);

**Day 5-7: Authentication Implementation**

**Reference Documents:** Complete Authentication System.docx, Sign In Page.docx

1. **Create Authentication Provider:**

// lib/supabase.ts

import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.NEXT\_PUBLIC\_SUPABASE\_URL!

const supabaseAnonKey = process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

export const supabase = createClient(supabaseUrl, supabaseAnonKey)

1. **Implement Auth Context:**

// contexts/AuthContext.tsx

'use client'

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

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

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

interface AuthContextType {

user: User | null

loading: boolean

signIn: (email: string, password: string) => Promise<any>

signUp: (email: string, password: string, name: string) => Promise<any>

signOut: () => Promise<any>

}

const AuthContext = createContext<AuthContextType | undefined>(undefined)

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

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

const [loading, setLoading] = useState(true)

useEffect(() => {

// Get initial session

supabase.auth.getSession().then(({ data: { session } }) => {

setUser(session?.user ?? null)

setLoading(false)

})

// Listen for auth changes

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

(event, session) => {

setUser(session?.user ?? null)

setLoading(false)

}

)

return () => subscription.unsubscribe()

}, [])

const value = {

user,

loading,

signIn: (email: string, password: string) =>

supabase.auth.signInWithPassword({ email, password }),

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

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

email,

password,

options: {

data: { name }

}

})

return { data, error }

},

signOut: () => supabase.auth.signOut(),

}

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

}

export const useAuth = () => {

const context = useContext(AuthContext)

if (context === undefined) {

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

}

return context

}

**Week 2: Multi-language Foundation**

**Day 8-10: Internationalization Setup**

**Reference Documents:** i18n Architecture.docx, RTL Layout System.docx

1. **Configure Next.js i18n:**

// next.config.js

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

const nextConfig = {

experimental: {

appDir: true,

},

i18n: {

locales: ['en', 'de', 'ar'],

defaultLocale: 'en',

localeDetection: true,

},

async redirects() {

return [

{

source: '/',

destination: '/en',

permanent: false,

},

]

},

}

module.exports = nextConfig

1. **Create Translation Structure:**

// lib/i18n.ts

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

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

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

}))

1. **Create Translation Files:**

// messages/en.json

{

"common": {

"login": "Sign In",

"signup": "Sign Up",

"dashboard": "Dashboard",

"signals": "Trading Signals",

"portfolio": "Portfolio"

},

"trading": {

"signal\_score": "Signal Score",

"entry\_price": "Entry Price",

"stop\_loss": "Stop Loss",

"take\_profit": "Take Profit"

}

}

// messages/de.json

{

"common": {

"login": "Anmelden",

"signup": "Registrieren",

"dashboard": "Dashboard",

"signals": "Handelssignale",

"portfolio": "Portfolio"

},

"trading": {

"signal\_score": "Signal-Bewertung",

"entry\_price": "Einstiegspreis",

"stop\_loss": "Stop-Loss",

"take\_profit": "Gewinnmitnahme"

}

}

// messages/ar.json

{

"common": {

"login": "تسجيل الدخول",

"signup": "إنشاء حساب",

"dashboard": "لوحة التحكم",

"signals": "إشارات التداول",

"portfolio": "المحفظة"

},

"trading": {

"signal\_score": "نقاط الإشارة",

"entry\_price": "سعر الدخول",

"stop\_loss": "وقف الخسارة",

"take\_profit": "جني الأرباح"

}

}

**Day 11-14: RTL Layout System**

**Reference Documents:** RTL Layout System.docx, Islamic Finance Compliance.docx

1. **Create Direction-Aware Components:**

// components/ui/RTLProvider.tsx

'use client'

import { createContext, useContext } from 'react'

import { useLocale } from 'next-intl'

interface RTLContextType {

dir: 'ltr' | 'rtl'

isRTL: boolean

}

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

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

const locale = useLocale()

const isRTL = locale === 'ar'

const dir = isRTL ? 'rtl' : 'ltr'

return (

<RTLContext.Provider value={{ dir, isRTL }}>

<div dir={dir} className={isRTL ? 'rtl' : 'ltr'}>

{children}

</div>

</RTLContext.Provider>

)

}

export const useRTL = () => {

const context = useContext(RTLContext)

if (context === undefined) {

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

}

return context

}

1. **Update Tailwind for RTL:**

// tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: [

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

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

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

],

theme: {

extend: {

fontFamily: {

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

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

},

},

},

plugins: [

require('@tailwindcss/forms'),

require('@tailwindcss/typography'),

// RTL support

function({ addUtilities }) {

const newUtilities = {

'.rtl': {

direction: 'rtl',

},

'.ltr': {

direction: 'ltr',

},

'.rtl .mr-auto': {

'margin-right': 'auto',

'margin-left': '0',

},

'.rtl .ml-auto': {

'margin-left': 'auto',

'margin-right': '0',

},

'.rtl .text-left': {

'text-align': 'right',

},

'.rtl .text-right': {

'text-align': 'left',

},

}

addUtilities(newUtilities)

},

],

}

**🎨 PHASE 2: FRONTEND DEVELOPMENT (Weeks 3-5)**

**Week 3: Landing Page & Authentication UI**

**Day 15-17: Landing Page Implementation**

**Reference Documents:** Landing Page.docx, Multi-Language Strategy.docx

1. **Create App Router Structure:**

app/

├── [locale]/

│ ├── layout.tsx

│ ├── page.tsx

│ ├── auth/

│ │ ├── signin/page.tsx

│ │ └── signup/page.tsx

│ ├── dashboard/page.tsx

│ └── signals/page.tsx

├── globals.css

└── layout.tsx

1. **Implement Internationalized Layout:**

// app/[locale]/layout.tsx

import { NextIntlClientProvider } from 'next-intl'

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

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

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

export default async function LocaleLayout({

children,

params: { locale }

}: {

children: React.ReactNode

params: { locale: string }

}) {

const messages = await getMessages()

return (

<html lang={locale}>

<body>

<NextIntlClientProvider messages={messages}>

<RTLProvider>

<AuthProvider>

{children}

</AuthProvider>

</RTLProvider>

</NextIntlClientProvider>

</body>

</html>

)

}

1. **Build Hero Section with Multi-language:**

// components/landing/HeroSection.tsx

'use client'

import { useTranslations } from 'next-intl'

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

export function HeroSection() {

const t = useTranslations('landing')

const { isRTL } = useRTL()

return (

<section className={`py-20 ${isRTL ? 'text-right' : 'text-left'}`}>

<div className="container mx-auto px-4">

<h1 className="text-5xl font-bold mb-6">

{t('hero\_title')}

</h1>

<p className="text-xl mb-8 text-gray-600">

{t('hero\_subtitle')}

</p>

<div className={`flex gap-4 ${isRTL ? 'flex-row-reverse' : ''}`}>

<button className="bg-blue-600 text-white px-8 py-3 rounded-lg">

{t('get\_started')}

</button>

<button className="border border-gray-300 px-8 py-3 rounded-lg">

{t('learn\_more')}

</button>

</div>

</div>

</section>

)

}

**Day 18-21: Authentication Pages**

**Reference Documents:** Sign In Page.docx, Complete Authentication System.docx

1. **Create Sign In Page:**

// app/[locale]/auth/signin/page.tsx

'use client'

import { useState } from 'react'

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

import { useTranslations } from 'next-intl'

import { useRouter } from 'next/navigation'

export default function SignInPage() {

const [email, setEmail] = useState('')

const [password, setPassword] = useState('')

const [loading, setLoading] = useState(false)

const { signIn } = useAuth()

const t = useTranslations('auth')

const router = useRouter()

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

e.preventDefault()

setLoading(true)

try {

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

if (error) throw error

router.push('/dashboard')

} catch (error) {

console.error('Sign in error:', error)

} finally {

setLoading(false)

}

}

return (

<div className="min-h-screen flex items-center justify-center">

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

<div>

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

{t('sign\_in\_title')}

</h2>

</div>

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

<div>

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

{t('email')}

</label>

<input

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

className="mt-1 block w-full border rounded-lg px-3 py-2"

required

/>

</div>

<div>

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

{t('password')}

</label>

<input

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

className="mt-1 block w-full border rounded-lg px-3 py-2"

required

/>

</div>

<button

type="submit"

disabled={loading}

className="w-full bg-blue-600 text-white py-2 rounded-lg disabled:opacity-50"

>

{loading ? t('signing\_in') : t('sign\_in')}

</button>

</form>

</div>

</div>

)

}

**Week 4: Dashboard Foundation**

**Day 22-24: Dashboard Layout**

**Reference Documents:** Dashboard.docx, Signals.docx

1. **Create Dashboard Layout:**

// components/dashboard/DashboardLayout.tsx

'use client'

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

import { useTranslations } from 'next-intl'

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

import { Navigation } from './Navigation'

import { Header } from './Header'

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

const { user } = useAuth()

const t = useTranslations('dashboard')

const { isRTL } = useRTL()

if (!user) {

return <div>Loading...</div>

}

return (

<div className={`min-h-screen bg-gray-50 ${isRTL ? 'rtl' : 'ltr'}`}>

<Header />

<div className="flex">

<Navigation />

<main className="flex-1 p-6">

{children}

</main>

</div>

</div>

)

}

1. **Create Signal Heatmap Component:**

// components/signals/SignalHeatmap.tsx

'use client'

import { useState, useEffect } from 'react'

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

import { useTranslations } from 'next-intl'

interface Signal {

ticker: string

finalScore: number

signalType: 'bullish' | 'bearish'

strength: string

entryPrice: number

}

export function SignalHeatmap() {

const [signals, setSignals] = useState<Signal[]>([])

const [loading, setLoading] = useState(true)

const t = useTranslations('signals')

useEffect(() => {

fetchSignals()

}, [])

const fetchSignals = async () => {

try {

const { data, error } = await supabase

.from('trading\_signals')

.select('\*')

.gte('final\_score', 80)

.order('final\_score', { ascending: false })

.limit(50)

if (error) throw error

setSignals(data || [])

} catch (error) {

console.error('Error fetching signals:', error)

} finally {

setLoading(false)

}

}

if (loading) return <div>Loading signals...</div>

return (

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

{signals.map((signal) => (

<div

key={signal.ticker}

className={`p-4 rounded-lg border ${

signal.signalType === 'bullish'

? 'bg-green-50 border-green-200'

: 'bg-red-50 border-red-200'

}`}

>

<div className="flex justify-between items-start mb-2">

<h3 className="font-bold text-lg">{signal.ticker}</h3>

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

{signal.finalScore}/100

</span>

</div>

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

<div className="flex justify-between">

<span>{t('type')}:</span>

<span className={signal.signalType === 'bullish' ? 'text-green-600' : 'text-red-600'}>

{t(signal.signalType)}

</span>

</div>

<div className="flex justify-between">

<span>{t('entry\_price')}:</span>

<span>${signal.entryPrice}</span>

</div>

<div className="flex justify-between">

<span>{t('strength')}:</span>

<span>{signal.strength}</span>

</div>

</div>

</div>

))}

</div>

)

}

**Day 25-28: Signal Detail Pages**

**Reference Documents:** Signal Detail Page.docx, Master API Endpoints.docx

**Week 5: Paper Trading & Portfolio**

**Day 29-31: Paper Trading Implementation**

**Reference Documents:** Paper Trading.docx, Portfolio Management.docx

**Day 32-35: User Settings & Preferences**

**Reference Documents:** Settings.docx, User Management.docx

**⚡ PHASE 3: BACKEND DEVELOPMENT (Weeks 6-7)**

**Week 6: API Development**

**Day 36-38: Core API Implementation**

**Reference Documents:** Master API Endpoints.docx, Backend Architecture Analysis.docx

**Day 39-42: Signal Processing Engine**

**Reference Documents:** Financial Data & Signal Processing.docx

**Week 7: External Integrations**

**Day 43-45: Market Data Integration**

**Reference Documents:** Market Data Integration.docx

**Day 46-49: Alert Systems Setup**

**Reference Documents:** Integrations & Deployment.docx

**🔔 PHASE 4: NOTIFICATIONS & AUTOMATION (Week 8)**

**Week 8: Alert Systems**

**Day 50-52: Make.com Workflow Setup**

**Reference Documents:** Make.com Integration Guide.docx

**Day 53-56: Telegram & Email Alerts**

**Reference Documents:** Telegram Bot Setup.docx, Email Templates.docx

**💳 PHASE 5: PAYMENTS & MONETIZATION (Week 9)**

**Week 9: Payment Integration**

**Day 57-59: Stripe Setup**

**Reference Documents:** Payment Integration.docx, Subscription Management.docx

**Day 60-63: Subscription Management**

**🚀 PHASE 6: TESTING & DEPLOYMENT (Weeks 10-12)**

**Week 10: Testing & Quality Assurance**

**Day 64-66: Multi-language Testing**

**Reference Documents:** Testing Strategy.docx

**Day 67-70: Performance Optimization**

**Week 11: Beta Testing**

**Day 71-73: Beta User Onboarding**

**Day 74-77: Feedback Integration**

**Week 12: Production Deployment**

**Day 78-80: Production Setup**

**Reference Documents:** Deployment Guide.docx

**Day 81-84: Launch Preparation**

**✅ SUCCESS CRITERIA & QUALITY GATES**

**Phase 1 Go-Live Requirements:**

* [ ] **Authentication:** Working sign-up/sign-in for all 3 languages
* [ ] **Dashboard:** Signal heatmap displaying 500 S&P stocks
* [ ] **Signals:** Individual signal pages with scoring ≥80
* [ ] **Paper Trading:** Virtual portfolio tracking functionality
* [ ] **Alerts:** Telegram and email notifications working
* [ ] **Payments:** Stripe integration for $29/month subscription
* [ ] **Multi-language:** Full EN/DE/AR support with RTL
* [ ] **Performance:** <3 second page loads, 99%+ uptime
* [ ] **Mobile:** Responsive design working on all devices

**Success Metrics for Phase 1:**

* **Signal Win Rate:** ≥60% by month 6
* **User Base:** 50 paying subscribers
* **Revenue:** $1,450/month recurring
* **User Retention:** ≥40% monthly retention
* **Cost Control:** ≤$149/month operational costs

**🆘 TROUBLESHOOTING & SUPPORT**

**Common Development Issues:**

1. **Translation Loading:** Check locale configuration in next.config.js
2. **RTL Layout:** Verify Tailwind RTL utilities are working
3. **Authentication:** Confirm Supabase environment variables
4. **Database Queries:** Check RLS policies for user permissions
5. **API Calls:** Verify endpoint configurations in Master API Endpoints

**Development Resources:**

* **Technical Support:** Reference specific document for detailed implementation
* **UI Components:** Use existing Lovable components where possible
* **API Integration:** Follow Master API Endpoints.docx exactly
* **Multi-language:** Reference i18n Architecture.docx for all internationalization

**🎯 POST-LAUNCH ROADMAP**

**Phase 2 Scaling (Months 7-12):**

* Expand to Russell 1000 stocks (1,000 total)
* Advanced signal filtering and analytics
* Mobile app development
* Premium tier ($49/month)

**Phase 3 Full Scale (Months 13-18):**

* Complete 6,000+ stock universe
* Institutional features
* API access for developers
* White-label solutions

**🚀 READY FOR SEAMLESS DEVELOPMENT EXECUTION**

**This guide transforms your comprehensive documentation into actionable development steps. Follow each phase sequentially for successful Phase 1 launch of your international trading platform!**