**i18n TECHNICAL ARCHITECTURE**

**📋 DOCUMENT STATUS**

**Status:** ✅ **MASTER TECHNICAL REFERENCE**  
**Version:** 1.0  
**Authority:** Single Source of Truth for Internationalization Implementation  
**Implementation Priority:** Phase 1 Foundation (Months 1-6)

**🏗️ ARCHITECTURE OVERVIEW**

**Core Technology Stack:**

* **Framework:** React i18next + Next.js App Router
* **Language Detection:** Browser/URL/Cookie based
* **Bundle Management:** Lazy loading with namespace splitting
* **Storage:** Database-backed translations + static JSON files
* **SEO:** Internationalized routing with proper hreflang tags

**📁 PROJECT STRUCTURE**

kurzora-platform/frontend/

├── 📄 next.config.js # i18n configuration

├── 📄 middleware.ts # Language routing middleware

├── 📄 i18n.config.ts # i18next configuration

│

├── 📁 src/

│ ├── 📁 app/

│ │ ├── 📁 [locale]/ # Internationalized app router

│ │ │ ├── 📄 layout.tsx # Locale-aware layout

│ │ │ ├── 📁 dashboard/ # All pages under locale

│ │ │ ├── 📁 signals/

│ │ │ └── 📁 profile/

│ │ └── 📄 globals.css # RTL-aware global styles

│ │

│ ├── 📁 lib/

│ │ ├── 📄 i18n.ts # i18next setup

│ │ ├── 📄 translations.ts # Translation utilities

│ │ └── 📄 rtl-utils.ts # RTL helper functions

│ │

│ ├── 📁 components/

│ │ ├── 📁 ui/

│ │ │ ├── 📄 LanguageSwitch.tsx

│ │ │ ├── 📄 RTLProvider.tsx

│ │ │ └── 📄 DirectionAwareComponent.tsx

│ │ └── 📁 layout/

│ │ └── 📄 InternationalizedLayout.tsx

│ │

│ ├── 📁 hooks/

│ │ ├── 📄 useTranslation.ts # Enhanced translation hook

│ │ ├── 📄 useDirection.ts # RTL/LTR direction hook

│ │ └── 📄 useLocale.ts # Locale management hook

│ │

│ └── 📁 locales/ # Translation files

│ ├── 📁 en/

│ │ ├── 📄 common.json # Common UI elements

│ │ ├── 📄 trading.json # Trading-specific terms

│ │ ├── 📄 auth.json # Authentication

│ │ ├── 📄 dashboard.json # Dashboard content

│ │ ├── 📄 signals.json # Signal-related content

│ │ ├── 📄 islamic.json # Islamic finance terms

│ │ └── 📄 errors.json # Error messages

│ ├── 📁 de/

│ │ ├── 📄 common.json

│ │ ├── 📄 trading.json

│ │ ├── 📄 auth.json

│ │ ├── 📄 dashboard.json

│ │ ├── 📄 signals.json

│ │ ├── 📄 islamic.json

│ │ └── 📄 errors.json

│ └── 📁 ar/

│ ├── 📄 common.json

│ ├── 📄 trading.json

│ ├── 📄 auth.json

│ ├── 📄 dashboard.json

│ ├── 📄 signals.json

│ ├── 📄 islamic.json

│ └── 📄 errors.json

**⚙️ CORE CONFIGURATION**

**Next.js Configuration (next.config.js)**

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

const nextConfig = {

// Internationalization configuration

i18n: {

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

defaultLocale: 'en',

localeDetection: true,

domains: [

{

domain: 'kurzora.com',

defaultLocale: 'en',

locales: ['en']

},

{

domain: 'kurzora.de',

defaultLocale: 'de',

locales: ['de']

},

{

domain: 'kurzora.ae',

defaultLocale: 'ar',

locales: ['ar']

}

]

},

// Webpack configuration for i18n optimization

webpack: (config, { isServer }) => {

// Optimize translation bundle loading

config.optimization.splitChunks.cacheGroups.translations = {

test: /[\\/]locales[\\/]/,

name: 'translations',

chunks: 'all',

priority: 10

};

return config;

},

// Headers for proper language handling

async headers() {

return [

{

source: '/:path\*',

headers: [

{

key: 'Accept-Language',

value: 'en,de,ar'

}

]

}

];

}

};

module.exports = nextConfig;

**i18next Configuration (lib/i18n.ts)**

import i18n from 'i18next';

import { initReactI18next } from 'react-i18next';

import Backend from 'i18next-http-backend';

import LanguageDetector from 'i18next-browser-languagedetector';

// Supported languages configuration

export const SUPPORTED\_LANGUAGES = [

{ code: 'en', name: 'English', dir: 'ltr', flag: '🇺🇸' },

{ code: 'de', name: 'Deutsch', dir: 'ltr', flag: '🇩🇪' },

{ code: 'ar', name: 'العربية', dir: 'rtl', flag: '🇸🇦' }

] as const;

export type SupportedLanguage = typeof SUPPORTED\_LANGUAGES[number]['code'];

// Namespace configuration for better organization

export const NAMESPACES = [

'common', // Common UI elements

'trading', // Trading-specific terminology

'auth', // Authentication pages

'dashboard', // Dashboard content

'signals', // Signal-related content

'islamic', // Islamic finance terminology

'errors' // Error messages

] as const;

export type Namespace = typeof NAMESPACES[number];

// i18next initialization

i18n

.use(Backend)

.use(LanguageDetector)

.use(initReactI18next)

.init({

// Language configuration

lng: 'en',

fallbackLng: 'en',

supportedLngs: SUPPORTED\_LANGUAGES.map(lang => lang.code),

// Namespace configuration

defaultNS: 'common',

ns: NAMESPACES,

// Backend configuration for lazy loading

backend: {

loadPath: '/locales/{{lng}}/{{ns}}.json',

requestOptions: {

cache: 'default',

credentials: 'same-origin'

}

},

// Language detection configuration

detection: {

order: ['path', 'cookie', 'header', 'navigator'],

caches: ['cookie'],

lookupFromPathIndex: 0,

checkWhitelist: true

},

// Interpolation configuration

interpolation: {

escapeValue: false, // React already does escaping

format: (value, format, lng) => {

// Custom formatting for numbers, dates, currencies

if (format === 'currency') {

const currency = lng === 'de' ? 'EUR' : 'USD';

return new Intl.NumberFormat(lng, {

style: 'currency',

currency

}).format(value);

}

if (format === 'number') {

return new Intl.NumberFormat(lng).format(value);

}

if (format === 'date') {

return new Intl.DateTimeFormat(lng).format(new Date(value));

}

return value;

}

},

// React configuration

react: {

useSuspense: false, // Handle loading states manually

bindI18n: 'languageChanged',

bindI18nStore: false,

transEmptyNodeValue: '',

transSupportBasicHtmlNodes: true,

transKeepBasicHtmlNodesFor: ['br', 'strong', 'i', 'em']

},

// Development configuration

debug: process.env.NODE\_ENV === 'development',

// Performance configuration

load: 'currentOnly', // Only load current language

preload: false, // Don't preload other languages

// Missing key configuration

saveMissing: process.env.NODE\_ENV === 'development',

missingKeyHandler: (lng, ns, key) => {

if (process.env.NODE\_ENV === 'development') {

console.warn(`Missing translation key: ${ns}:${key} for language: ${lng}`);

}

}

});

export default i18n;

**Language Routing Middleware (middleware.ts)**

import { NextRequest, NextResponse } from 'next/server';

import { SUPPORTED\_LANGUAGES } from './lib/i18n';

// Language detection priority order

const DETECTION\_ORDER = ['path', 'cookie', 'header'] as const;

export function middleware(request: NextRequest) {

const pathname = request.nextUrl.pathname;

const searchParams = request.nextUrl.searchParams;

// Check if pathname already has a locale

const pathnameHasLocale = SUPPORTED\_LANGUAGES.some(

(lang) => pathname.startsWith(`/${lang.code}/`) || pathname === `/${lang.code}`

);

if (pathnameHasLocale) {

// Extract locale from pathname for direction handling

const locale = pathname.split('/')[1];

const language = SUPPORTED\_LANGUAGES.find(lang => lang.code === locale);

if (language) {

const response = NextResponse.next();

// Set direction header for RTL support

response.headers.set('x-language-direction', language.dir);

response.headers.set('x-current-locale', locale);

return response;

}

}

// Detect user's preferred language

const detectedLocale = detectUserLanguage(request);

// Redirect to localized path

const redirectUrl = new URL(`/${detectedLocale}${pathname}`, request.url);

// Preserve search parameters

searchParams.forEach((value, key) => {

redirectUrl.searchParams.set(key, value);

});

return NextResponse.redirect(redirectUrl);

}

function detectUserLanguage(request: NextRequest): string {

// 1. Check URL parameter override (?lang=de)

const urlLang = request.nextUrl.searchParams.get('lang');

if (urlLang && SUPPORTED\_LANGUAGES.some(lang => lang.code === urlLang)) {

return urlLang;

}

// 2. Check cookie preference

const cookieLang = request.cookies.get('i18next')?.value;

if (cookieLang && SUPPORTED\_LANGUAGES.some(lang => lang.code === cookieLang)) {

return cookieLang;

}

// 3. Check Accept-Language header

const acceptLanguage = request.headers.get('accept-language');

if (acceptLanguage) {

const preferredLanguages = acceptLanguage

.split(',')

.map(lang => lang.split(';')[0].trim().toLowerCase());

for (const prefLang of preferredLanguages) {

// Exact match

const exactMatch = SUPPORTED\_LANGUAGES.find(lang => lang.code === prefLang);

if (exactMatch) return exactMatch.code;

// Partial match (e.g., 'en-US' matches 'en')

const partialMatch = SUPPORTED\_LANGUAGES.find(lang =>

prefLang.startsWith(lang.code)

);

if (partialMatch) return partialMatch.code;

}

}

// 4. Default to English

return 'en';

}

export const config = {

matcher: [

/\*

\* Match all request paths except for the ones starting with:

\* - api (API routes)

\* - \_next/static (static files)

\* - \_next/image (image optimization files)

\* - favicon.ico (favicon file)

\* - locales (translation files)

\*/

'/((?!api|\_next/static|\_next/image|favicon.ico|locales).\*)',

],

};

**🎣 ENHANCED HOOKS**

**Enhanced Translation Hook (hooks/useTranslation.ts)**

import { useTranslation as useI18nTranslation, TFunction } from 'react-i18next';

import { useCallback, useMemo } from 'react';

import { Namespace, SupportedLanguage } from '@/lib/i18n';

interface TranslationOptions {

returnObjects?: boolean;

replace?: Record<string, any>;

lng?: SupportedLanguage;

fallbackKey?: string;

}

interface EnhancedTranslation {

t: TFunction;

currentLanguage: SupportedLanguage;

isLoading: boolean;

changeLanguage: (lng: SupportedLanguage) => Promise<void>;

formatCurrency: (amount: number) => string;

formatNumber: (num: number) => string;

formatDate: (date: Date | string) => string;

formatPercentage: (value: number) => string;

translateArray: (key: string, options?: TranslationOptions) => string[];

hasTranslation: (key: string, ns?: Namespace) => boolean;

}

export function useTranslation(ns?: Namespace): EnhancedTranslation {

const { t, i18n, ready } = useI18nTranslation(ns);

const currentLanguage = i18n.language as SupportedLanguage;

const changeLanguage = useCallback(async (lng: SupportedLanguage) => {

await i18n.changeLanguage(lng);

// Store preference in cookie

document.cookie = `i18next=${lng}; path=/; max-age=31536000; SameSite=Strict`;

// Trigger page reload for complete language switch

window.location.reload();

}, [i18n]);

const formatCurrency = useCallback((amount: number) => {

const currency = currentLanguage === 'de' ? 'EUR' : 'USD';

return new Intl.NumberFormat(currentLanguage, {

style: 'currency',

currency

}).format(amount);

}, [currentLanguage]);

const formatNumber = useCallback((num: number) => {

return new Intl.NumberFormat(currentLanguage).format(num);

}, [currentLanguage]);

const formatDate = useCallback((date: Date | string) => {

const dateObj = typeof date === 'string' ? new Date(date) : date;

return new Intl.DateTimeFormat(currentLanguage, {

year: 'numeric',

month: 'long',

day: 'numeric'

}).format(dateObj);

}, [currentLanguage]);

const formatPercentage = useCallback((value: number) => {

return new Intl.NumberFormat(currentLanguage, {

style: 'percent',

minimumFractionDigits: 2,

maximumFractionDigits: 2

}).format(value / 100);

}, [currentLanguage]);

const translateArray = useCallback((key: string, options?: TranslationOptions) => {

const result = t(key, { ...options, returnObjects: true });

return Array.isArray(result) ? result : [];

}, [t]);

const hasTranslation = useCallback((key: string, namespace?: Namespace) => {

return i18n.exists(key, { ns: namespace });

}, [i18n]);

return useMemo(() => ({

t,

currentLanguage,

isLoading: !ready,

changeLanguage,

formatCurrency,

formatNumber,

formatDate,

formatPercentage,

translateArray,

hasTranslation

}), [

t,

currentLanguage,

ready,

changeLanguage,

formatCurrency,

formatNumber,

formatDate,

formatPercentage,

translateArray,

hasTranslation

]);

}

**Direction Hook (hooks/useDirection.ts)**

import { useMemo } from 'react';

import { useTranslation } from './useTranslation';

import { SUPPORTED\_LANGUAGES } from '@/lib/i18n';

interface DirectionInfo {

direction: 'ltr' | 'rtl';

isRTL: boolean;

isLTR: boolean;

className: string;

opposite: 'ltr' | 'rtl';

}

export function useDirection(): DirectionInfo {

const { currentLanguage } = useTranslation();

return useMemo(() => {

const language = SUPPORTED\_LANGUAGES.find(lang => lang.code === currentLanguage);

const direction = language?.dir || 'ltr';

return {

direction,

isRTL: direction === 'rtl',

isLTR: direction === 'ltr',

className: direction === 'rtl' ? 'rtl' : 'ltr',

opposite: direction === 'rtl' ? 'ltr' : 'rtl'

};

}, [currentLanguage]);

}

**🎨 COMPONENTS**

**Language Switcher (components/ui/LanguageSwitch.tsx)**

'use client';

import React, { useState } from 'react';

import { ChevronDown, Globe } from 'lucide-react';

import { useTranslation } from '@/hooks/useTranslation';

import { SUPPORTED\_LANGUAGES, SupportedLanguage } from '@/lib/i18n';

import {

DropdownMenu,

DropdownMenuContent,

DropdownMenuItem,

DropdownMenuTrigger,

} from '@/components/ui/dropdown-menu';

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

interface LanguageSwitchProps {

variant?: 'default' | 'minimal' | 'flag-only';

showFlag?: boolean;

showText?: boolean;

className?: string;

}

export function LanguageSwitch({

variant = 'default',

showFlag = true,

showText = true,

className = ''

}: LanguageSwitchProps) {

const { currentLanguage, changeLanguage, isLoading } = useTranslation();

const [isChanging, setIsChanging] = useState(false);

const currentLang = SUPPORTED\_LANGUAGES.find(lang => lang.code === currentLanguage);

const handleLanguageChange = async (newLanguage: SupportedLanguage) => {

if (newLanguage === currentLanguage) return;

setIsChanging(true);

try {

await changeLanguage(newLanguage);

} catch (error) {

console.error('Failed to change language:', error);

setIsChanging(false);

}

};

if (isLoading || !currentLang) {

return (

<div className={`animate-pulse bg-slate-700 rounded-md h-10 w-24 ${className}`} />

);

}

return (

<DropdownMenu>

<DropdownMenuTrigger asChild>

<Button

variant="outline"

size="sm"

disabled={isChanging}

className={`${className} ${isChanging ? 'opacity-50' : ''}`}

>

<Globe className="h-4 w-4 mr-1" />

{showFlag && <span className="mr-1">{currentLang.flag}</span>}

{showText && <span>{currentLang.name}</span>}

<ChevronDown className="h-4 w-4 ml-1" />

</Button>

</DropdownMenuTrigger>

<DropdownMenuContent align="end" className="w-48">

{SUPPORTED\_LANGUAGES.map((language) => (

<DropdownMenuItem

key={language.code}

onClick={() => handleLanguageChange(language.code)}

className={`flex items-center space-x-2 ${

language.code === currentLanguage ? 'bg-blue-50 dark:bg-blue-900' : ''

}`}

>

<span className="text-lg">{language.flag}</span>

<span className="flex-1">{language.name}</span>

{language.code === currentLanguage && (

<span className="text-blue-600 dark:text-blue-400">✓</span>

)}

</DropdownMenuItem>

))}

</DropdownMenuContent>

</DropdownMenu>

);

}

**RTL Provider (components/ui/RTLProvider.tsx)**

'use client';

import React, { useEffect } from 'react';

import { useDirection } from '@/hooks/useDirection';

interface RTLProviderProps {

children: React.ReactNode;

}

export function RTLProvider({ children }: RTLProviderProps) {

const { direction, isRTL } = useDirection();

useEffect(() => {

// Set document direction

document.documentElement.dir = direction;

document.documentElement.lang = direction === 'rtl' ? 'ar' : 'en';

// Add/remove RTL class to body

if (isRTL) {

document.body.classList.add('rtl');

document.body.classList.remove('ltr');

} else {

document.body.classList.add('ltr');

document.body.classList.remove('rtl');

}

// Clean up on unmount

return () => {

document.body.classList.remove('rtl', 'ltr');

};

}, [direction, isRTL]);

return (

<div className={`w-full h-full ${direction}`} dir={direction}>

{children}

</div>

);

}

**🗄️ DATABASE SCHEMA EXTENSIONS**

**Translation Tables**

-- Translation management tables

CREATE TABLE translation\_namespaces (

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

name VARCHAR(50) UNIQUE NOT NULL,

description TEXT,

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

CREATE TABLE translation\_keys (

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

namespace\_id UUID NOT NULL REFERENCES translation\_namespaces(id),

key\_name VARCHAR(255) NOT NULL,

description TEXT,

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

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

UNIQUE(namespace\_id, key\_name)

);

CREATE TABLE translations (

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

key\_id UUID NOT NULL REFERENCES translation\_keys(id),

language\_code VARCHAR(5) NOT NULL,

content TEXT NOT NULL,

is\_approved BOOLEAN DEFAULT false,

translator\_id UUID REFERENCES users(id),

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

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

UNIQUE(key\_id, language\_code)

);

-- 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 date\_format VARCHAR(20) DEFAULT 'MM/DD/YYYY';

ALTER TABLE users ADD COLUMN number\_format VARCHAR(20) DEFAULT 'US';

-- Indexes for performance

CREATE INDEX idx\_translations\_language ON translations(language\_code);

CREATE INDEX idx\_translations\_key ON translations(key\_id);

CREATE INDEX idx\_users\_language ON users(preferred\_language);

**🚀 API INTERNATIONALIZATION**

**API Response Localization**

// API middleware for response localization

export async function localizationMiddleware(request: Request, response: any) {

const acceptLanguage = request.headers.get('accept-language');

const userLanguage = detectLanguageFromHeader(acceptLanguage) || 'en';

// Localize error messages

if (response.error) {

response.error.message = await translateErrorMessage(

response.error.code,

userLanguage

);

}

// Localize data labels

if (response.data) {

response.data = await localizeDataLabels(response.data, userLanguage);

}

return response;

}

// Error message translation

async function translateErrorMessage(errorCode: string, language: string): Promise<string> {

const translations = await loadTranslations(language, 'errors');

return translations[errorCode] || translations['generic\_error'] || 'An error occurred';

}

**📊 PERFORMANCE OPTIMIZATION**

**Translation Bundle Management**

// Dynamic import for translation loading

export async function loadTranslationBundle(language: string, namespace: string) {

try {

const translation = await import(`@/locales/${language}/${namespace}.json`);

return translation.default;

} catch (error) {

console.warn(`Failed to load translation bundle: ${language}/${namespace}`);

// Fallback to English

const fallback = await import(`@/locales/en/${namespace}.json`);

return fallback.default;

}

}

// Translation preloading strategy

export function preloadCriticalTranslations(language: string) {

const criticalNamespaces = ['common', 'auth', 'errors'];

return Promise.all(

criticalNamespaces.map(ns => loadTranslationBundle(language, ns))

);

}

**🔧 IMPLEMENTATION CHECKLIST**

**Phase 1: Foundation (Week 1-2)**

* [ ] Install and configure react-i18next
* [ ] Set up Next.js internationalized routing
* [ ] Create base translation file structure
* [ ] Implement language detection middleware
* [ ] Build LanguageSwitch component

**Phase 2: Core Features (Week 3-4)**

* [ ] Implement enhanced translation hooks
* [ ] Create RTL support system
* [ ] Set up namespace-based translation loading
* [ ] Add database schema for translations
* [ ] Build translation management utilities

**Phase 3: Advanced Features (Week 5-6)**

* [ ] Implement API response localization
* [ ] Add currency and number formatting
* [ ] Create translation validation system
* [ ] Set up performance monitoring
* [ ] Add SEO optimization for multiple languages

**Phase 4: Testing & Optimization (Week 7-8)**

* [ ] Write comprehensive tests for i18n features
* [ ] Optimize bundle loading and caching
* [ ] Add accessibility features for RTL
* [ ] Performance testing and optimization
* [ ] Documentation and deployment

**📞 CURSOR IMPLEMENTATION GUIDE**

**Immediate Next Steps:**

1. **Install Dependencies:** npm install react-i18next i18next i18next-http-backend i18next-browser-languagedetector
2. **Copy Configuration Files:** Use the provided config files as-is
3. **Create Translation Files:** Start with English base translations
4. **Test Language Switching:** Implement LanguageSwitch component first
5. **Add RTL Support:** Follow the RTL architecture document for Arabic support

**Critical Implementation Notes:**

* Build English foundation completely before adding other languages
* Test translation loading performance extensively
* Ensure all user-facing text goes through translation system
* Plan for scalable translation management from day one
* Consider translation workflow for ongoing content updates

**Next Document:** Review RTL Layout System Design for Arabic interface implementation.