Admin Panel

**Kurzora Admin Panel - Complete UI Analysis**

**13-Point Framework for Immediate Cursor Implementation**

**1. UI Components & Layout**

**Interactive Elements**

**Primary Admin Components:**

* **AdminLayout** (sidebar navigation with route protection)
* **AdminDashboard** (metrics overview and activity feed)
* **AdminUsers** (user management CRUD operations)
* **AdminSignals** (signal management and audit trails)
* **AdminSettings** (system configuration panels)
* **CreateSignalModal** (signal creation dialog)
* **SignalAuditModal** (audit trail viewer)
* **ActivityFeed** (real-time system events)
* **MetricCards** (KPI dashboard widgets)
* **DataTables** (users/signals with sorting and filtering)

**Navigation & Controls:**

* **AdminSidebar** (4-section navigation: Dashboard, Users, Signals, Settings)
* **SearchAndFilter** (global search across all admin sections)
* **ExportButtons** (CSV/PDF export functionality)
* **BulkActions** (multi-select operations)
* **RoleManagement** (inline role editing)
* **StatusToggles** (enable/disable system features)

**React + TypeScript Component Structure**

// Complete Admin Panel Architecture

<AdminProtectedRoute>

<AdminLayout>

<div className="min-h-screen bg-gradient-to-br from-slate-950 via-blue-950 to-slate-950">

{/\* Top Navigation Bar \*/}

<nav className="bg-slate-900/50 backdrop-blur-sm border-b border-red-500/30">

<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 space-x-4">

<Button variant="ghost" size="sm" asChild>

<Link to="/dashboard">

<ArrowLeft className="h-4 w-4 mr-2" />

Back to Platform

</Link>

</Button>

<div className="flex items-center space-x-1">

<Shield className="h-5 w-5 text-red-400" />

<span className="text-white font-semibold">Admin Panel</span>

</div>

</div>

<AdminUserInfo user={user} />

</div>

</div>

</nav>

<div className="flex">

{/\* Sidebar Navigation \*/}

<AdminSidebar currentPath={location.pathname} />

{/\* Main Content Area \*/}

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

<AdminErrorBoundary>

<Routes>

<Route path="/admin" element={<AdminDashboard />} />

<Route path="/admin/users" element={<AdminUsers />} />

<Route path="/admin/signals" element={<AdminSignals />} />

<Route path="/admin/settings" element={<AdminSettings />} />

</Routes>

</AdminErrorBoundary>

</main>

</div>

</div>

</AdminLayout>

</AdminProtectedRoute>

// AdminDashboard Component Structure

<div className="space-y-6">

{/\* Header \*/}

<AdminPageHeader

title="Admin Dashboard"

subtitle="System overview and recent activity"

actions={<SystemRefreshButton />}

/>

{/\* KPI Metrics Grid \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

<MetricCard

title="Total Users"

value={1247}

change="+12% from last month"

icon={Users}

color="blue"

trend="up"

/>

<MetricCard

title="Active Signals"

value={42}

change="+3 today"

icon={TrendingUp}

color="green"

trend="up"

/>

<MetricCard

title="Pending Issues"

value={7}

change="Requires attention"

icon={AlertTriangle}

color="orange"

trend="warning"

/>

<MetricCard

title="System Health"

value="98.9%"

change="All systems operational"

icon={CheckCircle}

color="emerald"

trend="stable"

/>

</div>

{/\* Activity Feed \*/}

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700">

<CardHeader>

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

<CardTitle>Recent Activity</CardTitle>

<ActivityFilters

searchValue={activitySearch}

onSearchChange={setActivitySearch}

filterValue={activityFilter}

onFilterChange={setActivityFilter}

/>

</div>

</CardHeader>

<CardContent>

<ActivityFeed

activities={filteredActivity}

loading={loading}

onActivityClick={handleActivityClick}

/>

</CardContent>

</Card>

</div>

**Responsive Design Considerations**

// Mobile-responsive admin layout

const ResponsiveAdminLayout = () => {

const [sidebarOpen, setSidebarOpen] = useState(false);

const [isMobile] = useMediaQuery('(max-width: 768px)');

return (

<div className="flex h-screen">

{/\* Mobile sidebar overlay \*/}

{isMobile && sidebarOpen && (

<div

className="fixed inset-0 bg-black/50 z-40"

onClick={() => setSidebarOpen(false)}

/>

)}

{/\* Sidebar \*/}

<aside className={`

${isMobile ? 'fixed' : 'relative'}

${isMobile && !sidebarOpen ? '-translate-x-full' : 'translate-x-0'}

w-64 h-full bg-slate-900/30 border-r border-red-500/20

transition-transform duration-300 z-50

`}>

<AdminSidebar onNavigate={() => isMobile && setSidebarOpen(false)} />

</aside>

{/\* Main content \*/}

<main className="flex-1 overflow-auto">

{isMobile && (

<Button

variant="ghost"

size="sm"

onClick={() => setSidebarOpen(true)}

className="md:hidden fixed top-4 left-4 z-30"

>

<Menu className="h-4 w-4" />

</Button>

)}

{children}

</main>

</div>

);

};

// Responsive data tables

const ResponsiveDataTable = ({ data, columns, type }) => {

const [isMobile] = useMediaQuery('(max-width: 768px)');

return isMobile ? (

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

{data.map(item => (

<MobileDataCard key={item.id} item={item} type={type} />

))}

</div>

) : (

<Table>

<TableHeader>

{columns.map(col => (

<TableHead key={col.key}>{col.label}</TableHead>

))}

</TableHeader>

<TableBody>

{data.map(item => (

<TableRow key={item.id}>

{columns.map(col => (

<TableCell key={col.key}>

{col.render ? col.render(item) : item[col.key]}

</TableCell>

))}

</TableRow>

))}

</TableBody>

</Table>

);

};

**2. State Management (Zustand)**

**Admin Store Structure**

interface AdminStore {

// Dashboard state

dashboardMetrics: DashboardMetrics;

activityFeed: ActivityItem[];

systemHealth: SystemHealth;

// User management state

users: AdminUser[];

filteredUsers: AdminUser[];

selectedUsers: string[];

userFilters: UserFilters;

// Signal management state

signals: AdminSignal[];

filteredSignals: AdminSignal[];

selectedSignals: string[];

signalFilters: SignalFilters;

// Settings state

systemSettings: SystemSettings;

notificationSettings: NotificationSettings;

securitySettings: SecuritySettings;

tradingParameters: TradingParameters;

// UI state

loading: {

dashboard: boolean;

users: boolean;

signals: boolean;

settings: boolean;

};

error: string | null;

lastUpdated: Record<string, string>;

// Modal state

modals: {

createSignal: boolean;

editUser: boolean;

auditTrail: boolean;

bulkActions: boolean;

};

selectedItems: {

userId: string | null;

signalId: string | null;

auditData: any | null;

};

// Actions

loadDashboardData: () => Promise<void>;

loadUsers: () => Promise<void>;

loadSignals: () => Promise<void>;

loadSettings: () => Promise<void>;

// User actions

createUser: (userData: CreateUserRequest) => Promise<void>;

updateUser: (userId: string, updates: UpdateUserRequest) => Promise<void>;

deleteUser: (userId: string) => Promise<void>;

updateUserRole: (userId: string, role: UserRole) => Promise<void>;

bulkUpdateUsers: (userIds: string[], updates: BulkUpdateRequest) => Promise<void>;

// Signal actions

createSignal: (signalData: CreateSignalRequest) => Promise<void>;

updateSignal: (signalId: string, updates: UpdateSignalRequest) => Promise<void>;

closeSignal: (signalId: string, reason: string) => Promise<void>;

bulkCloseSignals: (signalIds: string[], reason: string) => Promise<void>;

// Settings actions

updateSystemSettings: (settings: Partial<SystemSettings>) => Promise<void>;

updateNotificationSettings: (settings: Partial<NotificationSettings>) => Promise<void>;

updateSecuritySettings: (settings: Partial<SecuritySettings>) => Promise<void>;

updateTradingParameters: (params: Partial<TradingParameters>) => Promise<void>;

// Filter actions

updateUserFilters: (filters: Partial<UserFilters>) => void;

updateSignalFilters: (filters: Partial<SignalFilters>) => void;

resetFilters: (section: 'users' | 'signals') => void;

// Export actions

exportUsers: (format: 'csv' | 'pdf') => Promise<void>;

exportSignals: (format: 'csv' | 'pdf') => Promise<void>;

exportActivityLog: (format: 'csv' | 'pdf') => Promise<void>;

// Real-time updates

subscribeToUpdates: () => void;

unsubscribeFromUpdates: () => void;

handleRealTimeUpdate: (update: RealTimeUpdate) => void;

}

export const useAdminStore = create<AdminStore>((set, get) => ({

// Initial state

dashboardMetrics: {

totalUsers: 0,

activeSignals: 0,

pendingIssues: 0,

systemHealth: 0

},

activityFeed: [],

systemHealth: { uptime: 0, status: 'unknown' },

users: [],

filteredUsers: [],

selectedUsers: [],

userFilters: {

search: '',

role: 'all',

status: 'all',

subscription: 'all'

},

signals: [],

filteredSignals: [],

selectedSignals: [],

signalFilters: {

search: '',

type: 'all',

status: 'all',

creator: 'all'

},

// Actions implementation

loadDashboardData: async () => {

set(state => ({

loading: { ...state.loading, dashboard: true },

error: null

}));

try {

const [metrics, activity, health] = await Promise.all([

fetch('/api/admin/metrics').then(r => r.json()),

fetch('/api/admin/activity').then(r => r.json()),

fetch('/api/admin/health').then(r => r.json())

]);

set({

dashboardMetrics: metrics,

activityFeed: activity.items,

systemHealth: health,

loading: { ...get().loading, dashboard: false },

lastUpdated: { ...get().lastUpdated, dashboard: new Date().toISOString() }

});

} catch (error) {

set({

error: error.message,

loading: { ...get().loading, dashboard: false }

});

}

},

createUser: async (userData) => {

try {

const response = await fetch('/api/admin/users', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${getAuthToken()}`

},

body: JSON.stringify(userData)

});

if (!response.ok) throw new Error('Failed to create user');

const newUser = await response.json();

// Optimistic update

set(state => ({

users: [newUser, ...state.users],

filteredUsers: [newUser, ...state.filteredUsers]

}));

// Refresh dashboard metrics

get().loadDashboardData();

} catch (error) {

set({ error: error.message });

throw error;

}

},

updateUserRole: async (userId, role) => {

// Optimistic update

const oldUsers = get().users;

set(state => ({

users: state.users.map(user =>

user.id === userId ? { ...user, role } : user

),

filteredUsers: state.filteredUsers.map(user =>

user.id === userId ? { ...user, role } : user

)

}));

try {

const response = await fetch(`/api/admin/users/${userId}/role`, {

method: 'PATCH',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${getAuthToken()}`

},

body: JSON.stringify({ role })

});

if (!response.ok) throw new Error('Failed to update user role');

} catch (error) {

// Revert optimistic update

set({ users: oldUsers });

set({ error: error.message });

throw error;

}

},

// Real-time update handling

handleRealTimeUpdate: (update) => {

const { type, data } = update;

switch (type) {

case 'USER\_REGISTERED':

set(state => ({

users: [data.user, ...state.users],

dashboardMetrics: {

...state.dashboardMetrics,

totalUsers: state.dashboardMetrics.totalUsers + 1

},

activityFeed: [data.activity, ...state.activityFeed.slice(0, 19)]

}));

break;

case 'SIGNAL\_CREATED':

set(state => ({

signals: [data.signal, ...state.signals],

dashboardMetrics: {

...state.dashboardMetrics,

activeSignals: state.dashboardMetrics.activeSignals + 1

},

activityFeed: [data.activity, ...state.activityFeed.slice(0, 19)]

}));

break;

case 'SYSTEM\_ALERT':

set(state => ({

dashboardMetrics: {

...state.dashboardMetrics,

pendingIssues: state.dashboardMetrics.pendingIssues + 1

},

activityFeed: [data.activity, ...state.activityFeed.slice(0, 19)]

}));

break;

}

}

}));

**3. API Contracts & Integration**

**API Endpoints**

// GET /api/admin/metrics - Dashboard metrics

interface DashboardMetricsResponse {

totalUsers: number;

activeSignals: number;

pendingIssues: number;

systemHealth: number;

trends: {

usersGrowth: string;

signalsToday: number;

issuesResolved: number;

systemUptime: string;

};

}

// GET /api/admin/activity - Activity feed

interface ActivityFeedRequest {

page?: number;

limit?: number;

type?: 'all' | 'user' | 'signal' | 'system';

search?: string;

startDate?: string;

endDate?: string;

}

interface ActivityFeedResponse {

items: ActivityItem[];

pagination: PaginationInfo;

filters: {

types: string[];

dateRange: { min: string; max: string };

};

}

// GET /api/admin/users - User management

interface AdminUsersRequest {

page?: number;

limit?: number;

search?: string;

role?: UserRole | 'all';

status?: UserStatus | 'all';

subscription?: string | 'all';

sortBy?: 'name' | 'email' | 'joinDate' | 'lastActive';

sortOrder?: 'asc' | 'desc';

}

interface AdminUsersResponse {

users: AdminUser[];

pagination: PaginationInfo;

aggregations: {

totalByRole: Record<UserRole, number>;

totalByStatus: Record<UserStatus, number>;

totalBySubscription: Record<string, number>;

};

}

// POST /api/admin/users - Create user

interface CreateUserRequest {

name: string;

email: string;

role: UserRole;

subscription?: string;

sendWelcomeEmail?: boolean;

tempPassword?: string;

}

// PATCH /api/admin/users/:id - Update user

interface UpdateUserRequest {

name?: string;

email?: string;

role?: UserRole;

status?: UserStatus;

subscription?: string;

notes?: string;

}

// GET /api/admin/signals - Signal management

interface AdminSignalsRequest {

page?: number;

limit?: number;

search?: string;

type?: 'Long' | 'Short' | 'all';

status?: SignalStatus | 'all';

creator?: string | 'all';

isShariahCompliant?: boolean;

sortBy?: 'symbol' | 'createdAt' | 'pnl' | 'status';

sortOrder?: 'asc' | 'desc';

}

interface AdminSignalsResponse {

signals: AdminSignal[];

pagination: PaginationInfo;

analytics: {

totalActive: number;

successRate: number;

totalPnL: number;

averageHoldTime: number;

};

}

// POST /api/admin/signals - Create signal

interface CreateSignalRequest {

symbol: string;

type: 'Long' | 'Short';

entryPrice: number;

stopLoss: number;

takeProfit: number;

status: SignalStatus;

notes?: string;

isShariahCompliant: boolean;

autoClose?: boolean;

autoCloseAfterDays?: number;

}

// GET /api/admin/settings - System settings

interface SystemSettingsResponse {

system: {

platformName: string;

maxUsers: number;

signalRefreshRate: number;

maintenanceMode: boolean;

};

notifications: {

emailEnabled: boolean;

smsEnabled: boolean;

pushEnabled: boolean;

adminEmail: string;

};

security: {

twoFactorRequired: boolean;

loginAttemptsLimit: number;

sessionTimeout: number;

autoLogoutEnabled: boolean;

};

trading: {

defaultRiskLevel: number;

maxSignalsPerDay: number;

autoCloseEnabled: boolean;

autoCloseDays: number;

demoModeEnabled: boolean;

};

}

// WebSocket events for real-time updates

interface RealTimeUpdate {

type: 'USER\_REGISTERED' | 'SIGNAL\_CREATED' | 'SIGNAL\_CLOSED' | 'SYSTEM\_ALERT' | 'USER\_ACTION';

data: {

user?: AdminUser;

signal?: AdminSignal;

activity?: ActivityItem;

alert?: SystemAlert;

};

timestamp: string;

}

**TypeScript Interfaces**

interface AdminUser {

id: string;

name: string;

email: string;

role: UserRole;

status: UserStatus;

subscription: string;

joinDate: string;

lastActive: string;

totalTrades: number;

totalPnL: number;

isVerified: boolean;

notes?: string;

}

interface AdminSignal {

id: string;

symbol: string;

type: 'Long' | 'Short';

status: SignalStatus;

entryPrice: number;

currentPrice: number;

stopLoss: number;

takeProfit: number;

pnl: number;

pnlPercent: number;

createdAt: string;

createdBy: string;

lastModified?: string;

lastModifiedBy?: string;

isShariahCompliant: boolean;

notes: string;

subscriberCount: number;

executionRate: number;

}

interface ActivityItem {

id: string;

type: 'user' | 'signal' | 'system';

description: string;

timestamp: string;

userId?: string;

signalId?: string;

metadata?: Record<string, any>;

severity: 'info' | 'warning' | 'error' | 'success';

}

interface DashboardMetrics {

totalUsers: number;

activeSignals: number;

pendingIssues: number;

systemHealth: number;

revenue: {

monthly: number;

growth: number;

};

engagement: {

activeUsers: number;

avgSessionTime: number;

};

}

type UserRole = 'admin' | 'analyst' | 'support' | 'user';

type UserStatus = 'active' | 'inactive' | 'suspended' | 'pending';

type SignalStatus = 'active' | 'pending' | 'closed' | 'cancelled';

**4. Performance & Optimization**

**Lazy Loading Implementation**

// Route-based code splitting

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

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

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

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

// Component lazy loading

const CreateSignalModal = lazy(() => import('./modals/CreateSignalModal'));

const BulkActionsModal = lazy(() => import('./modals/BulkActionsModal'));

const UserEditModal = lazy(() => import('./modals/UserEditModal'));

// Large data table virtualization

import { FixedSizeList as List } from 'react-window';

const VirtualizedUserTable = ({ users, onUserClick }) => {

const Row = ({ index, style }) => {

const user = users[index];

return (

<div style={style} className="flex items-center p-4 border-b border-slate-700">

<UserRow user={user} onClick={() => onUserClick(user)} />

</div>

);

};

return (

<List

height={600}

itemCount={users.length}

itemSize={70}

overscanCount={10}

>

{Row}

</List>

);

};

**Memoization Strategies**

// Heavy computation memoization

const AdminDashboard = () => {

const { dashboardMetrics, activityFeed } = useAdminStore();

const processedMetrics = useMemo(() => ({

userGrowthRate: calculateGrowthRate(dashboardMetrics.totalUsers),

signalSuccessRate: calculateSuccessRate(dashboardMetrics.activeSignals),

systemHealthScore: calculateHealthScore(dashboardMetrics.systemHealth),

trendAnalysis: analyzeTrends(activityFeed)

}), [dashboardMetrics, activityFeed]);

return <DashboardContent metrics={processedMetrics} />;

};

// Component memoization

const MetricCard = React.memo(({

title,

value,

change,

icon: Icon,

color,

trend

}) => {

return (

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700">

<CardHeader className="pb-2">

<CardTitle className="text-sm font-medium text-slate-300 flex items-center">

<Icon className={`h-4 w-4 mr-2 text-${color}-400`} />

{title}

</CardTitle>

</CardHeader>

<CardContent>

<div className="text-3xl font-bold text-white">{value}</div>

<TrendIndicator change={change} trend={trend} />

</CardContent>

</Card>

);

});

// Expensive filter operations

const useFilteredData = (data, filters) => {

return useMemo(() => {

return data.filter(item => {

const matchesSearch = !filters.search ||

item.name.toLowerCase().includes(filters.search.toLowerCase()) ||

item.email.toLowerCase().includes(filters.search.toLowerCase());

const matchesRole = filters.role === 'all' || item.role === filters.role;

const matchesStatus = filters.status === 'all' || item.status === filters.status;

return matchesSearch && matchesRole && matchesStatus;

}).sort((a, b) => {

if (filters.sortBy === 'name') {

return filters.sortOrder === 'asc'

? a.name.localeCompare(b.name)

: b.name.localeCompare(a.name);

}

// Add other sorting logic

return 0;

});

}, [data, filters]);

};

**Caching Strategy**

// React Query for admin data

const useAdminDashboard = () => {

return useQuery({

queryKey: ['admin', 'dashboard'],

queryFn: () => fetch('/api/admin/metrics').then(r => r.json()),

staleTime: 2 \* 60 \* 1000, // 2 minutes

cacheTime: 10 \* 60 \* 1000, // 10 minutes

refetchInterval: 5 \* 60 \* 1000, // 5 minutes

refetchOnWindowFocus: true

});

};

const useAdminUsers = (filters) => {

return useQuery({

queryKey: ['admin', 'users', filters],

queryFn: () => fetchAdminUsers(filters),

staleTime: 1 \* 60 \* 1000, // 1 minute

keepPreviousData: true,

onSuccess: (data) => {

// Prefetch user details for first 10 users

data.users.slice(0, 10).forEach(user => {

queryClient.prefetchQuery({

queryKey: ['admin', 'user', user.id],

queryFn: () => fetchUserDetails(user.id)

});

});

}

});

};

// Service worker for offline caching

const cachingStrategy = {

dashboard: 'cache-first', // Load from cache, fallback to network

users: 'network-first', // Load from network, fallback to cache

settings: 'cache-only', // Only use cached data

activity: 'network-only' // Always fetch fresh data

};

**5. Database Schema**

**PostgreSQL Tables**

-- Admin users table (extends main users table)

CREATE TABLE admin\_users (

id UUID PRIMARY KEY REFERENCES users(id),

admin\_role VARCHAR(20) NOT NULL CHECK (admin\_role IN ('super\_admin', 'admin', 'analyst', 'support')),

permissions JSONB NOT NULL DEFAULT '{}',

last\_admin\_action\_at TIMESTAMP WITH TIME ZONE,

admin\_notes TEXT,

created\_by UUID REFERENCES admin\_users(id),

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Admin activity log

CREATE TABLE admin\_activity\_log (

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

admin\_user\_id UUID NOT NULL REFERENCES admin\_users(id),

action\_type VARCHAR(50) NOT NULL,

target\_type VARCHAR(50), -- 'user', 'signal', 'system', etc.

target\_id UUID,

description TEXT NOT NULL,

metadata JSONB,

ip\_address INET,

user\_agent TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- System settings

CREATE TABLE system\_settings (

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

category VARCHAR(50) NOT NULL, -- 'system', 'notifications', 'security', 'trading'

setting\_key VARCHAR(100) NOT NULL,

setting\_value JSONB NOT NULL,

description TEXT,

is\_sensitive BOOLEAN DEFAULT false,

updated\_by UUID REFERENCES admin\_users(id),

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

UNIQUE(category, setting\_key)

);

-- Admin signals (extends main signals table)

CREATE TABLE admin\_signals (

id UUID PRIMARY KEY REFERENCES trading\_signals(id),

created\_by UUID REFERENCES admin\_users(id),

approved\_by UUID REFERENCES admin\_users(id),

approval\_status VARCHAR(20) DEFAULT 'pending' CHECK (approval\_status IN ('pending', 'approved', 'rejected')),

approval\_notes TEXT,

audit\_trail JSONB DEFAULT '[]',

subscriber\_count INTEGER DEFAULT 0,

execution\_rate DECIMAL(5,2) DEFAULT 0.0,

performance\_metrics JSONB,

last\_modified\_by UUID REFERENCES admin\_users(id),

last\_modified\_at TIMESTAMP WITH TIME ZONE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- User management audit

CREATE TABLE user\_management\_audit (

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

admin\_user\_id UUID NOT NULL REFERENCES admin\_users(id),

target\_user\_id UUID NOT NULL REFERENCES users(id),

action VARCHAR(50) NOT NULL, -- 'created', 'updated', 'deleted', 'role\_changed', 'suspended'

old\_values JSONB,

new\_values JSONB,

reason TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- System health metrics

CREATE TABLE system\_health\_metrics (

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

metric\_name VARCHAR(100) NOT NULL,

metric\_value DECIMAL(10,2) NOT NULL,

metric\_unit VARCHAR(20),

threshold\_warning DECIMAL(10,2),

threshold\_critical DECIMAL(10,2),

status VARCHAR(20) DEFAULT 'normal' CHECK (status IN ('normal', 'warning', 'critical')),

recorded\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Indexes for performance

CREATE INDEX idx\_admin\_activity\_log\_admin\_user ON admin\_activity\_log(admin\_user\_id, created\_at DESC);

CREATE INDEX idx\_admin\_activity\_log\_action\_type ON admin\_activity\_log(action\_type, created\_at DESC);

CREATE INDEX idx\_admin\_activity\_log\_target ON admin\_activity\_log(target\_type, target\_id);

CREATE INDEX idx\_system\_settings\_category ON system\_settings(category, setting\_key);

CREATE INDEX idx\_admin\_signals\_created\_by ON admin\_signals(created\_by, created\_at DESC);

CREATE INDEX idx\_admin\_signals\_status ON admin\_signals(approval\_status);

CREATE INDEX idx\_user\_management\_audit\_admin ON user\_management\_audit(admin\_user\_id, created\_at DESC);

CREATE INDEX idx\_user\_management\_audit\_target ON user\_management\_audit(target\_user\_id, created\_at DESC);

CREATE INDEX idx\_system\_health\_metrics\_name\_time ON system\_health\_metrics(metric\_name, recorded\_at DESC);

-- Materialized view for admin dashboard metrics

CREATE MATERIALIZED VIEW admin\_dashboard\_metrics AS

SELECT

(SELECT COUNT(\*) FROM users WHERE created\_at >= CURRENT\_DATE - INTERVAL '30 days') as new\_users\_30d,

(SELECT COUNT(\*) FROM users WHERE status = 'active') as active\_users,

(SELECT COUNT(\*) FROM trading\_signals WHERE status = 'active') as active\_signals,

(SELECT COUNT(\*) FROM admin\_activity\_log WHERE action\_type = 'system\_alert' AND created\_at >= CURRENT\_DATE) as pending\_issues,

(SELECT AVG(metric\_value) FROM system\_health\_metrics WHERE metric\_name = 'system\_uptime' AND recorded\_at >= CURRENT\_DATE) as system\_health,

CURRENT\_TIMESTAMP as last\_updated;

-- Function to refresh dashboard metrics

CREATE OR REPLACE FUNCTION refresh\_admin\_dashboard\_metrics()

RETURNS void AS $$

BEGIN

REFRESH MATERIALIZED VIEW admin\_dashboard\_metrics;

END;

$$ LANGUAGE plpgsql;

-- Trigger to log admin actions

CREATE OR REPLACE FUNCTION log\_admin\_action()

RETURNS trigger AS $$

BEGIN

INSERT INTO admin\_activity\_log (

admin\_user\_id,

action\_type,

target\_type,

target\_id,

description,

metadata

) VALUES (

COALESCE(NEW.updated\_by, NEW.created\_by),

TG\_OP,

TG\_TABLE\_NAME,

NEW.id,

format('%s %s', TG\_OP, TG\_TABLE\_NAME),

to\_jsonb(NEW)

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**6. User Experience**

**Loading States & Skeleton Screens**

const AdminDashboardSkeleton = () => (

<div className="space-y-6">

{/\* Header skeleton \*/}

<div className="animate-pulse">

<div className="h-8 bg-slate-700 rounded w-64 mb-2"></div>

<div className="h-4 bg-slate-700 rounded w-48"></div>

</div>

{/\* Metrics cards skeleton \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

{[...Array(4)].map((\_, i) => (

<Card key={i} className="bg-slate-800/50">

<CardContent className="p-6">

<div className="animate-pulse space-y-3">

<div className="h-4 bg-slate-700 rounded w-20"></div>

<div className="h-8 bg-slate-700 rounded w-16"></div>

<div className="h-3 bg-slate-700 rounded w-24"></div>

</div>

</CardContent>

</Card>

))}

</div>

{/\* Activity feed skeleton \*/}

<Card className="bg-slate-800/50">

<CardHeader>

<div className="animate-pulse">

<div className="h-6 bg-slate-700 rounded w-32"></div>

</div>

</CardHeader>

<CardContent>

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

{[...Array(8)].map((\_, i) => (

<div key={i} className="animate-pulse flex items-center space-x-3">

<div className="h-5 w-5 bg-slate-700 rounded"></div>

<div className="flex-1">

<div className="h-4 bg-slate-700 rounded w-3/4 mb-1"></div>

<div className="h-3 bg-slate-700 rounded w-1/2"></div>

</div>

<div className="h-6 bg-slate-700 rounded w-16"></div>

</div>

))}

</div>

</CardContent>

</Card>

</div>

);

const DataTableSkeleton = ({ rows = 10 }) => (

<div className="space-y-3">

{/\* Header \*/}

<div className="animate-pulse flex space-x-4 p-4 border-b border-slate-700">

<div className="h-4 bg-slate-700 rounded w-24"></div>

<div className="h-4 bg-slate-700 rounded w-16"></div>

<div className="h-4 bg-slate-700 rounded w-20"></div>

<div className="h-4 bg-slate-700 rounded w-32"></div>

<div className="h-4 bg-slate-700 rounded w-24"></div>

<div className="h-4 bg-slate-700 rounded w-20"></div>

</div>

{/\* Rows \*/}

{[...Array(rows)].map((\_, i) => (

<div key={i} className="animate-pulse flex space-x-4 p-4">

<div className="h-4 bg-slate-700 rounded w-24"></div>

<div className="h-4 bg-slate-700 rounded w-16"></div>

<div className="h-4 bg-slate-700 rounded w-20"></div>

<div className="h-4 bg-slate-700 rounded w-32"></div>

<div className="h-4 bg-slate-700 rounded w-24"></div>

<div className="h-4 bg-slate-700 rounded w-20"></div>

</div>

))}

</div>

);

**Error Boundaries & Fallbacks**

class AdminErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false, error: null, errorInfo: null };

}

static getDerivedStateFromError(error) {

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

this.setState({ error, errorInfo });

// Log admin errors with high priority

Sentry.withScope((scope) => {

scope.setTag('component', 'AdminPanel');

scope.setLevel('error');

scope.setContext('errorInfo', errorInfo);

scope.setUser({ id: this.props.userId, role: 'admin' });

Sentry.captureException(error);

});

}

render() {

if (this.state.hasError) {

return (

<div className="min-h-[400px] flex items-center justify-center">

<Card className="bg-slate-800/50 border-red-500/30 max-w-md">

<CardContent className="p-8 text-center">

<AlertTriangle className="h-12 w-12 text-red-400 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">

Admin Panel Error

</h3>

<p className="text-slate-400 mb-6">

There was an error loading the admin interface. This has been reported to the development team.

</p>

<div className="space-y-3">

<Button

onClick={() => window.location.reload()}

className="w-full bg-red-600 hover:bg-red-700"

>

Reload Page

</Button>

<Button

onClick={() => window.location.href = '/dashboard'}

variant="outline"

className="w-full border-slate-600"

>

Return to Dashboard

</Button>

</div>

{process.env.NODE\_ENV === 'development' && (

<details className="mt-4 text-left">

<summary className="text-red-400 cursor-pointer">

Error Details (Dev Mode)

</summary>

<pre className="text-xs text-slate-400 mt-2 overflow-auto">

{this.state.error && this.state.error.toString()}

</pre>

</details>

)}

</CardContent>

</Card>

</div>

);

}

return this.props.children;

}

}

// Network error fallback

const NetworkErrorFallback = ({ onRetry }) => (

<div className="text-center py-8">

<Wifi className="h-8 w-8 text-slate-600 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">Connection Lost</h3>

<p className="text-slate-400 mb-4">

Unable to connect to admin services. Check your connection and try again.

</p>

<Button onClick={onRetry} className="bg-blue-600 hover:bg-blue-700">

Retry Connection

</Button>

</div>

);

// Empty state components

const EmptyUsersState = () => (

<div className="text-center py-12">

<Users className="h-12 w-12 text-slate-600 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">No Users Found</h3>

<p className="text-slate-400 mb-6">

No users match your current filter criteria.

</p>

<Button variant="outline" onClick={() => resetFilters()}>

Clear Filters

</Button>

</div>

);

**Accessibility Implementation**

// Keyboard navigation for admin interface

const useAdminKeyboardShortcuts = () => {

useEffect(() => {

const handleKeyDown = (e) => {

// Only handle shortcuts when not in input fields

if (e.target.tagName === 'INPUT' || e.target.tagName === 'TEXTAREA') return;

switch (e.key) {

case 'D':

if (e.ctrlKey || e.metaKey) {

e.preventDefault();

navigate('/admin');

}

break;

case 'U':

if (e.ctrlKey || e.metaKey) {

e.preventDefault();

navigate('/admin/users');

}

break;

case 'S':

if (e.ctrlKey || e.metaKey) {

e.preventDefault();

navigate('/admin/signals');

}

break;

case 'Escape':

closeAllModals();

break;

}

};

document.addEventListener('keydown', handleKeyDown);

return () => document.removeEventListener('keydown', handleKeyDown);

}, []);

};

// Screen reader optimized components

const AccessibleDataTable = ({ data, columns, caption }) => (

<Table role="table" aria-label={caption}>

<caption className="sr-only">{caption}</caption>

<TableHeader>

<TableRow>

{columns.map((col, index) => (

<TableHead

key={col.key}

scope="col"

aria-sort={getSortDirection(col.key)}

tabIndex={0}

onKeyDown={(e) => {

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

e.preventDefault();

handleSort(col.key);

}

}}

>

{col.label}

{col.sortable && <SortIcon direction={getSortDirection(col.key)} />}

</TableHead>

))}

</TableRow>

</TableHeader>

<TableBody>

{data.map((item, rowIndex) => (

<TableRow

key={item.id}

aria-rowindex={rowIndex + 1}

aria-selected={selectedItems.includes(item.id)}

>

{columns.map((col) => (

<TableCell key={col.key} role="gridcell">

{col.render ? col.render(item) : item[col.key]}

</TableCell>

))}

</TableRow>

))}

</TableBody>

</Table>

);

// Voice announcements for admin actions

const useAdminAnnouncements = () => {

const announce = useCallback((message: string, priority: 'polite' | 'assertive' = 'polite') => {

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

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

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

announcement.className = 'sr-only';

announcement.textContent = message;

document.body.appendChild(announcement);

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

}, []);

return announce;

};

// Usage in admin actions

const handleUserRoleUpdate = async (userId: string, newRole: string) => {

try {

await updateUserRole(userId, newRole);

announce(`User role updated to ${newRole}`, 'assertive');

} catch (error) {

announce(`Failed to update user role: ${error.message}`, 'assertive');

}

};

**7. Integration Points**

**Navigation & Routing**

// Admin route configuration

const adminRoutes = createBrowserRouter([

{

path: '/admin',

element: <AdminProtectedRoute><AdminLayout /></AdminProtectedRoute>,

children: [

{

index: true,

element: <AdminDashboard />,

loader: adminDashboardLoader

},

{

path: 'users',

element: <AdminUsers />,

loader: adminUsersLoader

},

{

path: 'users/:id',

element: <UserDetailView />,

loader: userDetailLoader

},

{

path: 'signals',

element: <AdminSignals />,

loader: adminSignalsLoader

},

{

path: 'signals/:id',

element: <SignalDetailView />,

loader: signalDetailLoader

},

{

path: 'settings',

element: <AdminSettings />,

loader: adminSettingsLoader

}

]

}

]);

// Route loaders for data prefetching

const adminDashboardLoader = async () => {

const [metrics, activity] = await Promise.all([

fetch('/api/admin/metrics'),

fetch('/api/admin/activity?limit=20')

]);

return {

metrics: await metrics.json(),

activity: await activity.json()

};

};

// Admin navigation context

const AdminNavigationContext = createContext(null);

const useAdminNavigation = () => {

const navigate = useNavigate();

const location = useLocation();

const navigateWithState = useCallback((to: string, state?: any) => {

navigate(to, {

state: {

...state,

previousPath: location.pathname,

timestamp: Date.now()

}

});

}, [navigate, location]);

const navigateToUserDetail = useCallback((userId: string) => {

navigateWithState(`/admin/users/${userId}`, { source: 'user-table' });

}, [navigateWithState]);

const navigateToSignalDetail = useCallback((signalId: string) => {

navigateWithState(`/admin/signals/${signalId}`, { source: 'signal-table' });

}, [navigateWithState]);

return {

navigateToUserDetail,

navigateToSignalDetail,

goBack: () => navigate(-1),

currentPath: location.pathname

};

};

**Cross-Component State Sharing**

// Global admin state synchronization

const useAdminStateSync = () => {

const adminStore = useAdminStore();

const queryClient = useQueryClient();

// Listen for real-time updates via WebSocket

useEffect(() => {

const ws = new WebSocket(`${process.env.NEXT\_PUBLIC\_WS\_URL}/admin`);

ws.onmessage = (event) => {

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

// Update Zustand store

adminStore.handleRealTimeUpdate(update);

// Invalidate React Query cache

switch (update.type) {

case 'USER\_REGISTERED':

case 'USER\_UPDATED':

queryClient.invalidateQueries(['admin', 'users']);

queryClient.invalidateQueries(['admin', 'dashboard']);

break;

case 'SIGNAL\_CREATED':

case 'SIGNAL\_UPDATED':

queryClient.invalidateQueries(['admin', 'signals']);

queryClient.invalidateQueries(['admin', 'dashboard']);

break;

case 'SYSTEM\_ALERT':

queryClient.invalidateQueries(['admin', 'dashboard']);

break;

}

};

return () => ws.close();

}, [adminStore, queryClient]);

// Broadcast admin actions to other tabs

const broadcastAdminAction = useCallback((action: AdminAction) => {

const channel = new BroadcastChannel('admin-actions');

channel.postMessage(action);

}, []);

return { broadcastAdminAction };

};

// Admin action coordination

const useAdminActions = () => {

const adminStore = useAdminStore();

const { toast } = useToast();

const announce = useAdminAnnouncements();

const executeAdminAction = useCallback(async (

action: AdminActionType,

data: any,

options: { optimistic?: boolean; silent?: boolean } = {}

) => {

try {

// Optimistic update if requested

if (options.optimistic) {

adminStore.applyOptimisticUpdate(action, data);

}

// Execute the action

const result = await adminStore[action](data);

// Show success feedback

if (!options.silent) {

toast({

title: getActionSuccessMessage(action),

description: getActionDescription(action, data),

variant: 'success'

});

announce(getActionSuccessMessage(action), 'assertive');

}

return result;

} catch (error) {

// Revert optimistic update on error

if (options.optimistic) {

adminStore.revertOptimisticUpdate(action, data);

}

// Show error feedback

toast({

title: 'Action Failed',

description: error.message,

variant: 'destructive'

});

announce(`Action failed: ${error.message}`, 'assertive');

throw error;

}

}, [adminStore, toast, announce]);

return { executeAdminAction };

};

**8. Testing Strategy**

**Unit Tests**

// Admin store tests

describe('AdminStore', () => {

beforeEach(() => {

useAdminStore.getState().reset();

});

test('loads dashboard metrics successfully', async () => {

const mockMetrics = {

totalUsers: 1247,

activeSignals: 42,

pendingIssues: 7,

systemHealth: 98.9

};

global.fetch = jest.fn().mockResolvedValue({

ok: true,

json: async () => mockMetrics

});

const { loadDashboardData } = useAdminStore.getState();

await loadDashboardData();

const { dashboardMetrics } = useAdminStore.getState();

expect(dashboardMetrics).toEqual(mockMetrics);

});

test('handles user role update optimistically', async () => {

const initialUsers = [

{ id: '1', name: 'John', role: 'user' },

{ id: '2', name: 'Jane', role: 'admin' }

];

useAdminStore.setState({ users: initialUsers });

// Mock successful API call

global.fetch = jest.fn().mockResolvedValue({ ok: true });

const { updateUserRole } = useAdminStore.getState();

await updateUserRole('1', 'admin');

const { users } = useAdminStore.getState();

expect(users[0].role).toBe('admin');

});

test('reverts optimistic update on API failure', async () => {

const initialUsers = [{ id: '1', name: 'John', role: 'user' }];

useAdminStore.setState({ users: initialUsers });

// Mock failed API call

global.fetch = jest.fn().mockRejectedValue(new Error('API Error'));

const { updateUserRole } = useAdminStore.getState();

await expect(updateUserRole('1', 'admin')).rejects.toThrow();

const { users } = useAdminStore.getState();

expect(users[0].role).toBe('user'); // Should revert

});

});

// Component tests

describe('AdminDashboard', () => {

test('renders metrics correctly', () => {

const mockMetrics = {

totalUsers: 1247,

activeSignals: 42,

pendingIssues: 7,

systemHealth: 98.9

};

render(<AdminDashboard />, {

wrapper: ({ children }) => (

<QueryClient>

<MemoryRouter>

{children}

</MemoryRouter>

</QueryClient>

)

});

expect(screen.getByText('1,247')).toBeInTheDocument();

expect(screen.getByText('42')).toBeInTheDocument();

expect(screen.getByText('7')).toBeInTheDocument();

expect(screen.getByText('98.9%')).toBeInTheDocument();

});

test('filters activity feed correctly', async () => {

render(<AdminDashboard />);

const filterSelect = screen.getByRole('combobox', { name: /filter/i });

fireEvent.click(filterSelect);

fireEvent.click(screen.getByText('User Events'));

await waitFor(() => {

expect(screen.queryByText('Signal created')).not.toBeInTheDocument();

expect(screen.getByText('New user registration')).toBeInTheDocument();

});

});

});

describe('AdminUsers', () => {

test('updates user role via dropdown', async () => {

const mockUpdateUserRole = jest.fn().mockResolvedValue({});

render(<AdminUsers />);

const roleSelect = screen.getAllByRole('combobox')[1]; // First user's role

fireEvent.click(roleSelect);

fireEvent.click(screen.getByText('Admin'));

await waitFor(() => {

expect(mockUpdateUserRole).toHaveBeenCalledWith('user-1', 'admin');

});

});

test('exports users to CSV', async () => {

const mockCreateObjectURL = jest.fn();

global.URL.createObjectURL = mockCreateObjectURL;

render(<AdminUsers />);

const exportButton = screen.getByText('Export Users CSV');

fireEvent.click(exportButton);

await waitFor(() => {

expect(mockCreateObjectURL).toHaveBeenCalled();

});

});

});

**Integration Tests**

// Admin workflow tests

describe('Admin User Management Workflow', () => {

beforeEach(() => {

// Setup mock API responses

server.use(

rest.get('/api/admin/users', (req, res, ctx) => {

return res(ctx.json({ users: mockUsers }));

}),

rest.patch('/api/admin/users/:id/role', (req, res, ctx) => {

return res(ctx.json({ success: true }));

})

);

});

test('complete user role update flow', async () => {

render(<AdminApp />);

// Navigate to users page

fireEvent.click(screen.getByText('Users'));

// Wait for users to load

await waitFor(() => {

expect(screen.getByText('john@example.com')).toBeInTheDocument();

});

// Update user role

const roleSelect = screen.getByDisplayValue('user');

fireEvent.click(roleSelect);

fireEvent.click(screen.getByText('Admin'));

// Verify success message

await waitFor(() => {

expect(screen.getByText('Role Updated')).toBeInTheDocument();

});

// Verify dashboard metrics updated

fireEvent.click(screen.getByText('Dashboard'));

await waitFor(() => {

expect(screen.getByText('Recent Activity')).toBeInTheDocument();

});

});

});

// Real-time update tests

describe('Real-time Admin Updates', () => {

test('receives and displays new user registration', async () => {

const mockWebSocket = new MockWebSocket();

global.WebSocket = jest.fn().mockImplementation(() => mockWebSocket);

render(<AdminDashboard />);

// Simulate WebSocket message

const newUserUpdate = {

type: 'USER\_REGISTERED',

data: {

user: { id: 'new-user', name: 'New User', email: 'new@example.com' },

activity: {

id: 'activity-1',

type: 'user',

description: 'New user registration: new@example.com',

timestamp: new Date().toISOString()

}

}

};

mockWebSocket.simulate('message', { data: JSON.stringify(newUserUpdate) });

await waitFor(() => {

expect(screen.getByText('New user registration: new@example.com')).toBeInTheDocument();

});

});

});

**Mock Data**

export const mockAdminUsers: AdminUser[] = [

{

id: 'user-1',

name: 'John Doe',

email: 'john@example.com',

role: 'user',

status: 'active',

subscription: 'Pro Trader',

joinDate: '2024-01-15',

lastActive: '2024-06-10T15:30:00Z',

totalTrades: 45,

totalPnL: 1250.50,

isVerified: true

},

{

id: 'user-2',

name: 'Jane Smith',

email: 'jane@example.com',

role: 'analyst',

status: 'active',

subscription: 'Basic',

joinDate: '2024-02-20',

lastActive: '2024-06-10T14:15:00Z',

totalTrades: 23,

totalPnL: -150.25,

isVerified: true

}

];

export const mockAdminSignals: AdminSignal[] = [

{

id: 'signal-1',

symbol: 'EUR/USD',

type: 'Long',

status: 'active',

entryPrice: 1.0950,

currentPrice: 1.0975,

stopLoss: 1.0900,

takeProfit: 1.1050,

pnl: 25,

pnlPercent: 2.3,

createdAt: '2024-06-10T14:30:00Z',

createdBy: 'admin@kurzora.com',

isShariahCompliant: true,

notes: 'Strong bullish momentum expected',

subscriberCount: 124,

executionRate: 89.5

}

];

export const mockActivityFeed: ActivityItem[] = [

{

id: 'activity-1',

type: 'user',

description: 'New user registration: john@example.com',

timestamp: '2024-06-10T16:45:00Z',

userId: 'user-1',

severity: 'info'

},

{

id: 'activity-2',

type: 'signal',

description: 'Signal created for EUR/USD by admin@kurzora.com',

timestamp: '2024-06-10T16:30:00Z',

signalId: 'signal-1',

severity: 'success'

}

];

**9. Charts & Data Visualizations**

**Admin Dashboard Charts**

// System Health Gauge Chart

const SystemHealthGauge = ({ healthScore }) => {

const gaugeData = [

{ name: 'Health', value: healthScore, fill: getHealthColor(healthScore) },

{ name: 'Issues', value: 100 - healthScore, fill: '#374151' }

];

return (

<ResponsiveContainer width="100%" height={200}>

<PieChart>

<Pie

data={gaugeData}

cx="50%"

cy="50%"

startAngle={180}

endAngle={0}

innerRadius={60}

outerRadius={80}

dataKey="value"

stroke="none"

/>

<text

x="50%"

y="50%"

textAnchor="middle"

dominantBaseline="middle"

fontSize="24"

fontWeight="bold"

fill="#FFFFFF"

>

{healthScore.toFixed(1)}%

</text>

<text

x="50%"

y="60%"

textAnchor="middle"

dominantBaseline="middle"

fontSize="12"

fill="#94A3B8"

>

System Health

</text>

</PieChart>

</ResponsiveContainer>

);

};

// User Growth Chart

const UserGrowthChart = ({ data }) => {

return (

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

<AreaChart data={data}>

<defs>

<linearGradient id="userGrowth" x1="0" y1="0" x2="0" y2="1">

<stop offset="5%" stopColor="#3B82F6" stopOpacity={0.8}/>

<stop offset="95%" stopColor="#3B82F6" stopOpacity={0.1}/>

</linearGradient>

</defs>

<CartesianGrid strokeDasharray="3 3" stroke="#374151" />

<XAxis

dataKey="date"

stroke="#9CA3AF"

tick={{ fill: '#9CA3AF' }}

tickFormatter={(value) => format(new Date(value), 'MMM dd')}

/>

<YAxis

stroke="#9CA3AF"

tick={{ fill: '#9CA3AF' }}

/>

<Tooltip

contentStyle={{

backgroundColor: '#1F2937',

border: '1px solid #374151',

borderRadius: '8px',

color: '#FFFFFF'

}}

labelFormatter={(value) => format(new Date(value), 'MMM dd, yyyy')}

formatter={(value) => [value, 'New Users']}

/>

<Area

type="monotone"

dataKey="newUsers"

stroke="#3B82F6"

fillOpacity={1}

fill="url(#userGrowth)"

strokeWidth={2}

/>

</AreaChart>

</ResponsiveContainer>

);

};

// Signal Performance Heatmap

const SignalPerformanceHeatmap = ({ data }) => {

const maxValue = Math.max(...data.map(d => Math.abs(d.pnl)));

return (

<ResponsiveContainer width="100%" height={400}>

<Treemap

data={data}

dataKey="pnl"

ratio={4/3}

stroke="#374151"

strokeWidth={2}

>

<Tooltip

contentStyle={{

backgroundColor: '#1F2937',

border: '1px solid #374151',

borderRadius: '8px'

}}

formatter={(value, name, props) => [

`${value > 0 ? '+' : ''}${value.toFixed(2)}%`,

'P&L'

]}

labelFormatter={(label, payload) => {

if (payload && payload[0]) {

return payload[0].payload.symbol;

}

return label;

}}

/>

</Treemap>

</ResponsiveContainer>

);

};

// Activity Timeline Chart

const ActivityTimelineChart = ({ activities }) => {

const timelineData = useMemo(() => {

const grouped = activities.reduce((acc, activity) => {

const hour = format(new Date(activity.timestamp), 'HH:00');

acc[hour] = (acc[hour] || 0) + 1;

return acc;

}, {});

return Object.entries(grouped).map(([hour, count]) => ({

hour,

count,

timestamp: hour

}));

}, [activities]);

return (

<ResponsiveContainer width="100%" height={200}>

<BarChart data={timelineData}>

<CartesianGrid strokeDasharray="3 3" stroke="#374151" />

<XAxis

dataKey="hour"

stroke="#9CA3AF"

tick={{ fill: '#9CA3AF', fontSize: 12 }}

/>

<YAxis

stroke="#9CA3AF"

tick={{ fill: '#9CA3AF', fontSize: 12 }}

/>

<Tooltip

contentStyle={{

backgroundColor: '#1F2937',

border: '1px solid #374151',

borderRadius: '8px'

}}

formatter={(value) => [value, 'Activities']}

/>

<Bar

dataKey="count"

fill="#8B5CF6"

radius={[2, 2, 0, 0]}

/>

</BarChart>

</ResponsiveContainer>

);

};

**Real-time Chart Updates**

// Real-time metric updating

const RealTimeMetricChart = ({ metricName, refreshInterval = 30000 }) => {

const [data, setData] = useState([]);

const [isLive, setIsLive] = useState(true);

useEffect(() => {

if (!isLive) return;

const interval = setInterval(async () => {

try {

const response = await fetch(`/api/admin/metrics/${metricName}/live`);

const newDataPoint = await response.json();

setData(prevData => {

const newData = [...prevData, newDataPoint];

// Keep only last 50 data points

return newData.slice(-50);

});

} catch (error) {

console.error('Failed to fetch live metric:', error);

}

}, refreshInterval);

return () => clearInterval(interval);

}, [metricName, refreshInterval, isLive]);

return (

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

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

<h3 className="text-lg font-semibold text-white">{metricName}</h3>

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

<div className={`h-2 w-2 rounded-full ${isLive ? 'bg-green-400' : 'bg-gray-400'}`} />

<span className="text-sm text-slate-400">

{isLive ? 'Live' : 'Paused'}

</span>

<Button

size="sm"

variant="outline"

onClick={() => setIsLive(!isLive)}

className="ml-2"

>

{isLive ? 'Pause' : 'Resume'}

</Button>

</div>

</div>

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

<LineChart data={data}>

<CartesianGrid strokeDasharray="3 3" stroke="#374151" />

<XAxis

dataKey="timestamp"

stroke="#9CA3AF"

tickFormatter={(value) => format(new Date(value), 'HH:mm')}

/>

<YAxis stroke="#9CA3AF" />

<Tooltip

contentStyle={{

backgroundColor: '#1F2937',

border: '1px solid #374151',

borderRadius: '8px'

}}

labelFormatter={(value) => format(new Date(value), 'HH:mm:ss')}

/>

<Line

type="monotone"

dataKey="value"

stroke="#10B981"

strokeWidth={2}

dot={false}

isAnimationActive={true}

animationDuration={300}

/>

</LineChart>

</ResponsiveContainer>

</div>

);

};

// Chart export functionality

const useChartExport = () => {

const exportChart = useCallback(async (chartRef, filename, format = 'png') => {

if (!chartRef.current) return;

try {

const canvas = await html2canvas(chartRef.current, {

backgroundColor: '#0f172a',

scale: 2

});

if (format === 'png') {

const link = document.createElement('a');

link.download = `${filename}.png`;

link.href = canvas.toDataURL();

link.click();

} else if (format === 'pdf') {

const imgData = canvas.toDataURL('image/png');

const pdf = new jsPDF();

pdf.addImage(imgData, 'PNG', 10, 10, 190, 100);

pdf.save(`${filename}.pdf`);

}

} catch (error) {

console.error('Failed to export chart:', error);

}

}, []);

return { exportChart };

};

**10. Visual Data Elements**

**Status Indicators & Badges**

// Admin role badge system

const AdminRoleBadge = ({ role }) => {

const roleConfig = {

super\_admin: {

color: 'bg-red-600 text-white',

icon: Crown,

label: 'Super Admin'

},

admin: {

color: 'bg-orange-600 text-white',

icon: Shield,

label: 'Admin'

},

analyst: {

color: 'bg-blue-600 text-white',

icon: TrendingUp,

label: 'Analyst'

},

support: {

color: 'bg-purple-600 text-white',

icon: Headphones,

label: 'Support'

},

user: {

color: 'bg-gray-600 text-white',

icon: User,

label: 'User'

}

};

const config = roleConfig[role] || roleConfig.user;

const Icon = config.icon;

return (

<Badge className={`${config.color} flex items-center gap-1`}>

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

{config.label}

</Badge>

);

};

// System status indicator

const SystemStatusIndicator = ({ status, metric }) => {

const getStatusConfig = (status) => {

switch (status) {

case 'operational':

return { color: 'text-green-400', bg: 'bg-green-400/10', icon: CheckCircle };

case 'warning':

return { color: 'text-yellow-400', bg: 'bg-yellow-400/10', icon: AlertTriangle };

case 'critical':

return { color: 'text-red-400', bg: 'bg-red-400/10', icon: AlertCircle };

default:

return { color: 'text-gray-400', bg: 'bg-gray-400/10', icon: Circle };

}

};

const config = getStatusConfig(status);

const Icon = config.icon;

return (

<div className={`flex items-center space-x-2 px-3 py-2 rounded-lg ${config.bg}`}>

<Icon className={`h-4 w-4 ${config.color}`} />

<span className={`text-sm font-medium ${config.color}`}>

{metric || status.charAt(0).toUpperCase() + status.slice(1)}

</span>

</div>

);

};

// Activity severity indicators

const ActivitySeverityBadge = ({ severity, type }) => {

const severityConfig = {

info: { color: 'bg-blue-500/20 text-blue-400 border-blue-500/30' },

success: { color: 'bg-green-500/20 text-green-400 border-green-500/30' },

warning: { color: 'bg-yellow-500/20 text-yellow-400 border-yellow-500/30' },

error: { color: 'bg-red-500/20 text-red-400 border-red-500/30' }

};

const config = severityConfig[severity] || severityConfig.info;

return (

<Badge variant="outline" className={`${config.color} text-xs`}>

{type}

</Badge>

);

};

// Progress indicators for admin tasks

const AdminProgressIndicator = ({ progress, status, label }) => {

return (

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

<div className="flex justify-between text-sm">

<span className="text-slate-300">{label}</span>

<span className="text-slate-400">{progress}%</span>

</div>

<div className="w-full bg-slate-700 rounded-full h-2">

<div

className={`h-2 rounded-full transition-all duration-500 ${

status === 'completed' ? 'bg-green-500' :

status === 'error' ? 'bg-red-500' :

'bg-blue-500'

}`}

style={{ width: `${progress}%` }}

/>

</div>

</div>

);

};

**Animated Counters & Metrics**

// Animated admin metric counter

const AnimatedAdminMetric = ({

value,

previousValue,

format = (v) => v.toLocaleString(),

duration = 1500

}) => {

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

const [isIncreasing, setIsIncreasing] = useState(value > (previousValue || 0));

useEffect(() => {

setIsIncreasing(value > displayValue);

let startTime;

let animationFrame;

const animate = (timestamp) => {

if (!startTime) startTime = timestamp;

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

// Ease-out cubic animation

const easedProgress = 1 - Math.pow(1 - progress, 3);

const currentValue = displayValue + (value - displayValue) \* easedProgress;

setDisplayValue(currentValue);

if (progress < 1) {

animationFrame = requestAnimationFrame(animate);

} else {

setDisplayValue(value);

}

};

if (Math.abs(value - displayValue) > 0.1) {

animationFrame = requestAnimationFrame(animate);

}

return () => {

if (animationFrame) {

cancelAnimationFrame(animationFrame);

}

};

}, [value, displayValue, duration]);

return (

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

<span className="text-3xl font-bold text-white">

{format(displayValue)}

</span>

{Math.abs(value - (previousValue || 0)) > 0 && (

<div className={`flex items-center text-sm ${

isIncreasing ? 'text-green-400' : 'text-red-400'

}`}>

{isIncreasing ? (

<TrendingUp className="h-4 w-4" />

) : (

<TrendingDown className="h-4 w-4" />

)}

<span className="ml-1">

{Math.abs(value - (previousValue || 0))}

</span>

</div>

)}

</div>

);

};

// Real-time activity pulse

const ActivityPulse = ({ isActive, count }) => {

return (

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

<div className={`relative ${isActive ? 'animate-pulse' : ''}`}>

<div className={`h-3 w-3 rounded-full ${

isActive ? 'bg-green-400' : 'bg-gray-400'

}`} />

{isActive && (

<div className="absolute inset-0 h-3 w-3 rounded-full bg-green-400 animate-ping opacity-75" />

)}

</div>

<span className="text-sm text-slate-300">

{count} active {count === 1 ? 'session' : 'sessions'}

</span>

</div>

);

};

// Visual data comparison

const MetricComparison = ({ current, previous, label, format }) => {

const difference = current - previous;

const percentChange = previous ? (difference / previous) \* 100 : 0;

const isPositive = difference > 0;

return (

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

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

<span className="text-slate-400 text-sm">{label}</span>

<div className={`flex items-center text-sm ${

isPositive ? 'text-green-400' : difference < 0 ? 'text-red-400' : 'text-slate-400'

}`}>

{difference !== 0 && (

<>

{isPositive ? <ArrowUp className="h-3 w-3" /> : <ArrowDown className="h-3 w-3" />}

<span className="ml-1">

{Math.abs(percentChange).toFixed(1)}%

</span>

</>

)}

</div>

</div>

<div className="text-2xl font-bold text-white">

{format ? format(current) : current}

</div>

<div className="text-xs text-slate-500">

vs {format ? format(previous) : previous} last period

</div>

</div>

);

};

**11. Security & Validation**

**Input Validation Schemas**

import { z } from 'zod';

// Admin user validation

const AdminUserSchema = z.object({

name: z.string().min(2, 'Name must be at least 2 characters').max(50),

email: z.string().email('Invalid email address'),

role: z.enum(['super\_admin', 'admin', 'analyst', 'support', 'user']),

status: z.enum(['active', 'inactive', 'suspended', 'pending']),

subscription: z.string().optional(),

notes: z.string().max(500, 'Notes cannot exceed 500 characters').optional()

});

// Signal creation validation

const CreateSignalSchema = z.object({

symbol: z.string().min(1, 'Symbol is required').max(10, 'Symbol too long'),

type: z.enum(['Long', 'Short']),

entryPrice: z.number().positive('Entry price must be positive'),

stopLoss: z.number().positive('Stop loss must be positive'),

takeProfit: z.number().positive('Take profit must be positive'),

status: z.enum(['active', 'pending', 'closed']),

notes: z.string().max(1000, 'Notes cannot exceed 1000 characters').optional(),

isShariahCompliant: z.boolean()

}).refine((data) => {

if (data.type === 'Long') {

return data.takeProfit > data.entryPrice && data.stopLoss < data.entryPrice;

} else {

return data.takeProfit < data.entryPrice && data.stopLoss > data.entryPrice;

}

}, {

message: 'Stop loss and take profit levels must be logically consistent with signal type'

});

// System settings validation

const SystemSettingsSchema = z.object({

platformName: z.string().min(1, 'Platform name is required').max(100),

maxUsers: z.number().int().min(1, 'Max users must be at least 1').max(1000000),

signalRefreshRate: z.number().int().min(5, 'Refresh rate must be at least 5 seconds').max(300),

maintenanceMode: z.boolean()

});

const SecuritySettingsSchema = z.object({

twoFactorRequired: z.boolean(),

loginAttemptsLimit: z.number().int().min(3).max(10),

sessionTimeout: z.number().int().min(15, 'Minimum 15 minutes').max(480, 'Maximum 8 hours'),

autoLogoutEnabled: z.boolean()

});

// Bulk action validation

const BulkActionSchema = z.object({

action: z.enum(['update\_role', 'update\_status', 'delete', 'export']),

targetIds: z.array(z.string().uuid()).min(1, 'At least one item must be selected').max(100, 'Too many items selected'),

params: z.record(z.any()).optional()

});

// Filter validation

const AdminFilterSchema = z.object({

search: z.string().max(100).optional(),

role: z.enum(['all', 'super\_admin', 'admin', 'analyst', 'support', 'user']).optional(),

status: z.enum(['all', 'active', 'inactive', 'suspended', 'pending']).optional(),

dateRange: z.object({

start: z.string().datetime().optional(),

end: z.string().datetime().optional()

}).optional(),

sortBy: z.string().optional(),

sortOrder: z.enum(['asc', 'desc']).optional(),

page: z.number().int().min(1).optional(),

limit: z.number().int().min(1).max(100).optional()

});

**Authentication & Authorization**

// Admin role hierarchy and permissions

const ADMIN\_PERMISSIONS = {

super\_admin: [

'user:create', 'user:read', 'user:update', 'user:delete',

'signal:create', 'signal:read', 'signal:update', 'signal:delete',

'system:read', 'system:update', 'system:maintenance',

'admin:create', 'admin:read', 'admin:update', 'admin:delete'

],

admin: [

'user:read', 'user:update',

'signal:create', 'signal:read', 'signal:update', 'signal:delete',

'system:read', 'system:update'

],

analyst: [

'user:read',

'signal:create', 'signal:read', 'signal:update',

'system:read'

],

support: [

'user:read', 'user:update',

'signal:read',

'system:read'

]

};

// Permission checking hook

const useAdminPermissions = () => {

const { user } = useAuth();

const hasPermission = useCallback((permission: string) => {

if (!user?.adminRole) return false;

const userPermissions = ADMIN\_PERMISSIONS[user.adminRole] || [];

return userPermissions.includes(permission);

}, [user]);

const requirePermission = useCallback((permission: string) => {

if (!hasPermission(permission)) {

throw new Error(`Insufficient permissions: ${permission} required`);

}

}, [hasPermission]);

return { hasPermission, requirePermission };

};

// Protected admin action wrapper

const withAdminPermission = (permission: string) => {

return function (target: any, propertyName: string, descriptor: PropertyDescriptor) {

const method = descriptor.value;

descriptor.value = async function (...args: any[]) {

const { requirePermission } = useAdminPermissions();

requirePermission(permission);

return method.apply(this, args);

};

};

};

// Admin session management

const useAdminSession = () => {

const [adminSession, setAdminSession] = useState(null);

useEffect(() => {

// Enhanced session validation for admin users

const validateAdminSession = async () => {

try {

const response = await fetch('/api/admin/session/validate', {

headers: {

'Authorization': `Bearer ${getAuthToken()}`

}

});

if (!response.ok) {

throw new Error('Invalid admin session');

}

const sessionData = await response.json();

setAdminSession(sessionData);

// Setup auto-logout for admin inactivity

setupAdminInactivityTimer(sessionData.timeoutMinutes);

} catch (error) {

console.error('Admin session validation failed:', error);

// Redirect to login

window.location.href = '/';

}

};

validateAdminSession();

}, []);

const extendAdminSession = useCallback(async () => {

try {

await fetch('/api/admin/session/extend', {

method: 'POST',

headers: {

'Authorization': `Bearer ${getAuthToken()}`

}

});

} catch (error) {

console.error('Failed to extend admin session:', error);

}

}, []);

return { adminSession, extendAdminSession };

};

**Data Sanitization & XSS Prevention**

import DOMPurify from 'dompurify';

// Sanitize admin input data

const sanitizeAdminInput = (data: any): any => {

if (typeof data === 'string') {

return DOMPurify.sanitize(data, {

ALLOWED\_TAGS: [], // Strip all HTML tags

ALLOWED\_ATTR: []

});

}

if (Array.isArray(data)) {

return data.map(sanitizeAdminInput);

}

if (typeof data === 'object' && data !== null) {

const sanitized: any = {};

for (const [key, value] of Object.entries(data)) {

sanitized[DOMPurify.sanitize(key)] = sanitizeAdminInput(value);

}

return sanitized;

}

return data;

};

// Secure admin data display

const SecureDataDisplay = ({ data, type }) => {

const sanitizedData = useMemo(() => {

if (type === 'html') {

return DOMPurify.sanitize(data, {

ALLOWED\_TAGS: ['b', 'i', 'em', 'strong', 'span'],

ALLOWED\_ATTR: ['class']

});

}

return sanitizeAdminInput(data);

}, [data, type]);

if (type === 'html') {

return <div dangerouslySetInnerHTML={{ \_\_html: sanitizedData }} />;

}

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

};

// API request sanitization middleware

const sanitizeAdminRequest = (request: any) => {

return {

...request,

body: sanitizeAdminInput(request.body),

query: sanitizeAdminInput(request.query),

params: sanitizeAdminInput(request.params)

};

};

// Rate limiting for admin endpoints

const useAdminRateLimit = () => {

const [rateLimitStatus, setRateLimitStatus] = useState({

remaining: 100,

resetTime: Date.now() + 3600000

});

const checkRateLimit = useCallback(async (endpoint: string) => {

try {

const response = await fetch(`/api/admin/rate-limit/${endpoint}`, {

method: 'HEAD',

headers: {

'Authorization': `Bearer ${getAuthToken()}`

}

});

setRateLimitStatus({

remaining: parseInt(response.headers.get('X-RateLimit-Remaining') || '0'),

resetTime: parseInt(response.headers.get('X-RateLimit-Reset') || '0')

});

return response.ok;

} catch (error) {

console.error('Rate limit check failed:', error);

return false;

}

}, []);

return { rateLimitStatus, checkRateLimit };

};

**12. Environment & Configuration**

**Environment Variables**

// .env.local for Admin Panel

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

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

NEXT\_PUBLIC\_SENTRY\_DSN=https://...@sentry.io/...

NEXT\_PUBLIC\_AMPLITUDE\_API\_KEY=your\_amplitude\_key

// Admin-specific configuration

ADMIN\_SESSION\_TIMEOUT=3600000 // 1 hour in milliseconds

ADMIN\_MAX\_CONCURRENT\_SESSIONS=3

ADMIN\_RATE\_LIMIT\_REQUESTS=1000

ADMIN\_RATE\_LIMIT\_WINDOW=3600000

ADMIN\_AUDIT\_LOG\_RETENTION\_DAYS=365

// Security settings

ADMIN\_JWT\_SECRET=your\_super\_secure\_jwt\_secret

ADMIN\_ENCRYPTION\_KEY=your\_encryption\_key

ADMIN\_2FA\_ISSUER=Kurzora Admin

ADMIN\_SESSION\_COOKIE\_SECURE=true

ADMIN\_SESSION\_COOKIE\_HTTPONLY=true

// Feature flags for admin panel

FEATURE\_BULK\_OPERATIONS=true

FEATURE\_ADVANCED\_ANALYTICS=false

FEATURE\_REAL\_TIME\_MONITORING=true

FEATURE\_AUDIT\_EXPORT=true

FEATURE\_MAINTENANCE\_MODE=false

// Monitoring and alerts

ADMIN\_ERROR\_WEBHOOK\_URL=https://hooks.slack.com/...

ADMIN\_METRICS\_ENDPOINT=https://metrics.kurzora.com

ADMIN\_LOG\_LEVEL=info

**Feature Flags Configuration**

// Feature flag management

interface AdminFeatureFlags {

bulkOperations: boolean;

advancedAnalytics: boolean;

realTimeMonitoring: boolean;

auditExport: boolean;

maintenanceMode: boolean;

userImpersonation: boolean;

systemDiagnostics: boolean;

autoBackup: boolean;

}

const defaultFeatureFlags: AdminFeatureFlags = {

bulkOperations: process.env.FEATURE\_BULK\_OPERATIONS === 'true',

advancedAnalytics: process.env.FEATURE\_ADVANCED\_ANALYTICS === 'true',

realTimeMonitoring: process.env.FEATURE\_REAL\_TIME\_MONITORING === 'true',

auditExport: process.env.FEATURE\_AUDIT\_EXPORT === 'true',

maintenanceMode: process.env.FEATURE\_MAINTENANCE\_MODE === 'true',

userImpersonation: false, // High-security feature

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

autoBackup: true

};

// Feature flag hook

const useAdminFeatureFlags = () => {

const [featureFlags, setFeatureFlags] = useState<AdminFeatureFlags>(defaultFeatureFlags);

const isFeatureEnabled = useCallback((feature: keyof AdminFeatureFlags) => {

return featureFlags[feature];

}, [featureFlags]);

const updateFeatureFlag = useCallback(async (feature: keyof AdminFeatureFlags, enabled: boolean) => {

// Update feature flag via API

try {

await fetch('/api/admin/feature-flags', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${getAuthToken()}`

},

body: JSON.stringify({ feature, enabled })

});

setFeatureFlags(prev => ({ ...prev, [feature]: enabled }));

} catch (error) {

console.error('Failed to update feature flag:', error);

}

}, []);

return { featureFlags, isFeatureEnabled, updateFeatureFlag };

};

// Conditional component rendering

const ConditionalAdminFeature: React.FC<{

feature: keyof AdminFeatureFlags;

children: React.ReactNode;

fallback?: React.ReactNode;

}> = ({ feature, children, fallback = null }) => {

const { isFeatureEnabled } = useAdminFeatureFlags();

return isFeatureEnabled(feature) ? <>{children}</> : <>{fallback}</>;

};

**Performance Monitoring**

// Admin panel performance monitoring

const useAdminPerformanceMonitoring = () => {

useEffect(() => {

// Monitor Core Web Vitals for admin panel

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

for (const entry of list.getEntries()) {

if (entry.entryType === 'navigation') {

const navigationEntry = entry as PerformanceNavigationTiming;

// Track admin page load performance

const metrics = {

page: 'admin-panel',

loadTime: navigationEntry.loadEventEnd - navigationEntry.loadEventStart,

domContentLoaded: navigationEntry.domContentLoadedEventEnd - navigationEntry.loadEventStart,

firstPaint: performance.getEntriesByType('paint')[0]?.startTime || 0,

largestContentfulPaint: 0 // Will be updated by LCP observer

};

// Send to analytics

amplitude.track('Admin Panel Performance', metrics);

}

}

});

observer.observe({ entryTypes: ['navigation'] });

// LCP observer

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

for (const entry of list.getEntries()) {

amplitude.track('Admin LCP', {

lcp: entry.startTime,

element: entry.element?.tagName

});

}

});

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

return () => {

observer.disconnect();

lcpObserver.disconnect();

};

}, []);

};

// Error tracking and reporting

const useAdminErrorTracking = () => {

useEffect(() => {

// Enhanced error tracking for admin operations

const handleError = (error: Error, context: any) => {

Sentry.withScope((scope) => {

scope.setTag('area', 'admin-panel');

scope.setLevel('error');

scope.setContext('admin-context', context);

scope.setUser({

id: context.userId,

role: context.userRole,

adminPermissions: context.permissions

});

Sentry.captureException(error);

});

// Also send to admin monitoring webhook

if (process.env.ADMIN\_ERROR\_WEBHOOK\_URL) {

fetch(process.env.ADMIN\_ERROR\_WEBHOOK\_URL, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

error: error.message,

stack: error.stack,

context,

timestamp: new Date().toISOString()

})

}).catch(console.error);

}

};

window.addEventListener('error', (event) => {

handleError(event.error, { source: 'window-error' });

});

window.addEventListener('unhandledrejection', (event) => {

handleError(new Error(event.reason), { source: 'unhandled-promise' });

});

}, []);

};

// Admin action analytics

const useAdminAnalytics = () => {

const trackAdminAction = useCallback((action: string, properties: any = {}) => {

// Track admin actions for analytics

amplitude.track('Admin Action', {

action,

...properties,

timestamp: Date.now(),

userAgent: navigator.userAgent,

url: window.location.href

});

// Also log to admin audit trail

fetch('/api/admin/audit/track', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${getAuthToken()}`

},

body: JSON.stringify({

action,

properties,

timestamp: new Date().toISOString()

})

}).catch(console.error);

}, []);

return { trackAdminAction };

};

**13. Cross-Screen Data Flow**

**Real-time Data Synchronization**

// Admin WebSocket manager

const useAdminWebSocket = () => {

const adminStore = useAdminStore();

const [wsStatus, setWsStatus] = useState<'connecting' | 'connected' | 'disconnected'>('disconnected');

useEffect(() => {

const ws = new WebSocket(`${process.env.NEXT\_PUBLIC\_ADMIN\_WS\_URL}/admin`);

let heartbeatInterval: NodeJS.Timeout;

ws.onopen = () => {

setWsStatus('connected');

// Send authentication

ws.send(JSON.stringify({

type: 'auth',

token: getAuthToken()

}));

// Setup heartbeat

heartbeatInterval = setInterval(() => {

if (ws.readyState === WebSocket.OPEN) {

ws.send(JSON.stringify({ type: 'ping' }));

}

}, 30000);

};

ws.onmessage = (event) => {

try {

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

switch (data.type) {

case 'user\_registered':

adminStore.handleUserUpdate(data.payload);

break;

case 'signal\_created':

adminStore.handleSignalUpdate(data.payload);

break;

case 'system\_alert':

adminStore.handleSystemAlert(data.payload);

break;

case 'bulk\_update\_progress':

adminStore.updateBulkProgress(data.payload);

break;

case 'admin\_action':

adminStore.handleAdminAction(data.payload);

break;

}

} catch (error) {

console.error('Failed to parse WebSocket message:', error);

}

};

ws.onclose = () => {

setWsStatus('disconnected');

clearInterval(heartbeatInterval);

// Attempt reconnection after delay

setTimeout(() => {

if (document.visibilityState === 'visible') {

// Recursive call to reconnect

useAdminWebSocket();

}

}, 5000);

};

ws.onerror = (error) => {

console.error('Admin WebSocket error:', error);

setWsStatus('disconnected');

};

return () => {

clearInterval(heartbeatInterval);

ws.close();

};

}, [adminStore]);

return { wsStatus };

};

// Cross-tab admin synchronization

const useAdminTabSync = () => {

const adminStore = useAdminStore();

useEffect(() => {

const channel = new BroadcastChannel('admin-sync');

channel.onmessage = (event) => {

const { type, data } = event.data;

switch (type) {

case 'admin-action-performed':

// Refresh affected data in other tabs

adminStore.refreshData(data.affectedSections);

break;

case 'user-role-updated':

adminStore.updateUserInCache(data.userId, data.updates);

break;

case 'signal-status-changed':

adminStore.updateSignalInCache(data.signalId, data.updates);

break;

case 'settings-updated':

adminStore.refreshSettings();

break;

}

};

// Broadcast admin actions to other tabs

const originalActions = {

updateUserRole: adminStore.updateUserRole,

updateSignal: adminStore.updateSignal,

updateSettings: adminStore.updateSettings

};

// Wrap actions to broadcast changes

adminStore.updateUserRole = async (...args) => {

const result = await originalActions.updateUserRole(...args);

channel.postMessage({

type: 'user-role-updated',

data: { userId: args[0], updates: args[1] }

});

return result;

};

return () => {

channel.close();

// Restore original actions

Object.assign(adminStore, originalActions);

};

}, [adminStore]);

};

**Cache Management & Invalidation**

// Intelligent admin cache management

const useAdminCacheManager = () => {

const queryClient = useQueryClient();

const invalidateAdminCache = useCallback((sections: string[] = []) => {

if (sections.length === 0) {

// Invalidate all admin data

queryClient.invalidateQueries(['admin']);

} else {

// Invalidate specific sections

sections.forEach(section => {

queryClient.invalidateQueries(['admin', section]);

});

}

}, [queryClient]);

const updateAdminCache = useCallback((section: string, key: string, updater: (old: any) => any) => {

queryClient.setQueryData(['admin', section, key], updater);

}, [queryClient]);

const prefetchAdminData = useCallback(async (sections: string[]) => {

const prefetchPromises = sections.map(section => {

switch (section) {

case 'users':

return queryClient.prefetchQuery({

queryKey: ['admin', 'users'],

queryFn: () => fetch('/api/admin/users').then(r => r.json())

});

case 'signals':

return queryClient.prefetchQuery({

queryKey: ['admin', 'signals'],

queryFn: () => fetch('/api/admin/signals').then(r => r.json())

});

default:

return Promise.resolve();

}

});

await Promise.all(prefetchPromises);

}, [queryClient]);

return {

invalidateAdminCache,

updateAdminCache,

prefetchAdminData

};

};

// Smart cache invalidation based on admin actions

const useSmartCacheInvalidation = () => {

const { invalidateAdminCache } = useAdminCacheManager();

const handleAdminAction = useCallback((action: string, data: any) => {

switch (action) {

case 'user:create':

case 'user:update':

case 'user:delete':

invalidateAdminCache(['users', 'dashboard']);

break;

case 'signal:create':

case 'signal:update':

case 'signal:close':

invalidateAdminCache(['signals', 'dashboard']);

break;

case 'settings:update':

invalidateAdminCache(['settings']);

break;

case 'bulk:operation':

// Invalidate all data for bulk operations

invalidateAdminCache();

break;

default:

// For unknown actions, invalidate dashboard

invalidateAdminCache(['dashboard']);

}

}, [invalidateAdminCache]);

return { handleAdminAction };

};

// Optimistic updates with rollback

const useOptimisticAdminUpdates = () => {

const queryClient = useQueryClient();

const [rollbackActions] = useState(new Map());

const performOptimisticUpdate = useCallback(async (

queryKey: string[],

updater: (old: any) => any,

apiCall: () => Promise<any>

) => {

// Store the previous value for rollback

const previousValue = queryClient.getQueryData(queryKey);

rollbackActions.set(queryKey.join(':'), () => {

queryClient.setQueryData(queryKey, previousValue);

});

// Apply optimistic update

queryClient.setQueryData(queryKey, updater);

try {

// Perform the actual API call

const result = await apiCall();

// Remove rollback action on success

rollbackActions.delete(queryKey.join(':'));

return result;

} catch (error) {

// Rollback on error

const rollback = rollbackActions.get(queryKey.join(':'));

if (rollback) {

rollback();

rollbackActions.delete(queryKey.join(':'));

}

throw error;

}

}, [queryClient, rollbackActions]);

return { performOptimisticUpdate };

};

// Event-driven admin state management

const useAdminEventBus = () => {

const eventBus = useRef(new EventTarget());

const emit = useCallback((eventType: string, data: any) => {

eventBus.current.dispatchEvent(new CustomEvent(eventType, { detail: data }));

}, []);

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

eventBus.current.addEventListener(eventType, handler as EventListener);

return () => {

eventBus.current.removeEventListener(eventType, handler as EventListener);

};

}, []);

return { emit, subscribe };

};

**Implementation Priority for Admin Panel:**

1. **Core Authentication & Authorization** - Admin role protection and permissions
2. **Dashboard Metrics & Activity Feed** - Real-time system overview
3. **User Management CRUD** - Full user lifecycle management
4. **Signal Management** - Trading signal administration
5. **System Settings** - Configuration management
6. **Real-time Updates** - WebSocket integration for live data
7. **Advanced Analytics** - Charts and performance monitoring
8. **Audit Trail & Security** - Comprehensive logging and monitoring