**Profile**

**🎯 Kurzora Profile Page - Complete UI Analysis**

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

**1. UI Components & Layout**

**Interactive Elements**

**Primary Profile Components:**

* **PersonalInfoCard** (name/email display, change password button)
* **SubscriptionCard** (plan details, status, expiration)
* **ChangePasswordDialog** (modal for password updates)
* **ProfileEditForm** (future: editable profile fields)
* **SubscriptionManagement** (future: upgrade/downgrade options)

**Form Controls & Inputs:**

* Read-only input fields for name/email
* Change password button with modal trigger
* Future: Profile picture upload
* Future: Notification preferences toggles
* Future: Language/timezone selectors

**React + TypeScript Component Structure**

// Complete Profile Page Architecture

<Layout>

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

{/\* Header Section \*/}

<div className="mb-8">

<h1 className="text-3xl font-bold text-white mb-2">Profile</h1>

<p className="text-slate-400">Manage your account information</p>

</div>

{/\* Main Content Grid \*/}

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

{/\* Personal Information Card \*/}

<PersonalInfoCard

user={user}

onEditProfile={handleEditProfile}

onChangePassword={handleChangePassword}

isEditing={isEditing}

onSave={handleSaveProfile}

onCancel={handleCancelEdit}

/>

{/\* Subscription Card \*/}

<SubscriptionCard

subscription={user.subscription}

onUpgrade={handleUpgrade}

onDowngrade={handleDowngrade}

onCancelSubscription={handleCancelSubscription}

/>

</div>

{/\* Additional Settings Grid (Future) \*/}

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

<NotificationPreferencesCard />

<SecuritySettingsCard />

</div>

{/\* Modals \*/}

<ChangePasswordDialog

open={isChangePasswordOpen}

onOpenChange={setIsChangePasswordOpen}

onSuccess={handlePasswordChangeSuccess}

/>

<ProfilePictureModal

open={isProfilePictureOpen}

onOpenChange={setIsProfilePictureOpen}

currentImage={user.profilePicture}

onUpload={handleProfilePictureUpload}

/>

</div>

</Layout>

**Enhanced PersonalInfoCard Component**

interface PersonalInfoCardProps {

user: User;

onEditProfile: () => void;

onChangePassword: () => void;

isEditing: boolean;

onSave: (data: ProfileUpdateData) => Promise<void>;

onCancel: () => void;

}

const PersonalInfoCard: React.FC<PersonalInfoCardProps> = ({

user,

onEditProfile,

onChangePassword,

isEditing,

onSave,

onCancel

}) => {

const [formData, setFormData] = useState({

name: user.name,

phone: user.phone || '',

timezone: user.timezone || 'UTC',

language: user.language || 'en'

});

const [isUploading, setIsUploading] = useState(false);

return (

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700 hover:bg-slate-800/70 transition-all duration-300">

<CardHeader>

<CardTitle className="text-lg text-white flex items-center justify-between">

<div className="flex items-center">

<User className="h-5 w-5 mr-2 text-blue-400" />

Personal Information

</div>

{!isEditing && (

<Button

variant="ghost"

size="sm"

onClick={onEditProfile}

className="text-blue-400 hover:text-blue-300"

>

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

Edit

</Button>

)}

</CardTitle>

</CardHeader>

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

{/\* Profile Picture Section \*/}

<div className="flex items-center space-x-4 mb-6">

<div className="relative">

<Avatar className="h-16 w-16">

<AvatarImage src={user.profilePicture} alt={user.name} />

<AvatarFallback className="bg-blue-600 text-white text-lg">

{user.name?.charAt(0)?.toUpperCase()}

</AvatarFallback>

</Avatar>

{isEditing && (

<Button

size="sm"

className="absolute -bottom-2 -right-2 h-8 w-8 rounded-full p-0 bg-blue-600 hover:bg-blue-700"

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

>

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

</Button>

)}

</div>

<div>

<p className="text-white font-medium">{user.name}</p>

<p className="text-slate-400 text-sm">Member since {formatDate(user.createdAt)}</p>

</div>

</div>

{/\* Form Fields \*/}

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

<div>

<Label className="text-slate-300 text-sm font-medium">Full Name</Label>

{isEditing ? (

<Input

value={formData.name}

onChange={(e) => setFormData({ ...formData, name: e.target.value })}

className="bg-slate-700 border-slate-600 text-white mt-1 focus:ring-2 focus:ring-blue-400"

placeholder="Enter your full name"

/>

) : (

<div className="mt-1 p-3 bg-slate-700 border border-slate-600 rounded-md text-white">

{user.name}

</div>

)}

</div>

<div>

<Label className="text-slate-300 text-sm font-medium">Email Address</Label>

<div className="mt-1 p-3 bg-slate-700 border border-slate-600 rounded-md text-white relative">

{user.email}

<div className="absolute right-3 top-1/2 transform -translate-y-1/2">

{user.emailVerified ? (

<div className="flex items-center text-green-400">

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

<span className="text-xs">Verified</span>

</div>

) : (

<div className="flex items-center text-red-400">

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

<span className="text-xs">Unverified</span>

</div>

)}

</div>

</div>

{!user.emailVerified && (

<Button variant="link" size="sm" className="text-blue-400 p-0 h-auto mt-1">

Resend verification email

</Button>

)}

</div>

{isEditing && (

<>

<div>

<Label className="text-slate-300 text-sm font-medium">Phone Number</Label>

<Input

value={formData.phone}

onChange={(e) => setFormData({ ...formData, phone: e.target.value })}

className="bg-slate-700 border-slate-600 text-white mt-1 focus:ring-2 focus:ring-blue-400"

placeholder="+1 (555) 123-4567"

/>

</div>

<div>

<Label className="text-slate-300 text-sm font-medium">Timezone</Label>

<Select value={formData.timezone} onValueChange={(value) => setFormData({ ...formData, timezone: value })}>

<SelectTrigger className="bg-slate-700 border-slate-600 text-white mt-1">

<SelectValue />

</SelectTrigger>

<SelectContent className="bg-slate-700 border-slate-600">

<SelectItem value="UTC">UTC</SelectItem>

<SelectItem value="America/New\_York">Eastern Time</SelectItem>

<SelectItem value="America/Chicago">Central Time</SelectItem>

<SelectItem value="America/Denver">Mountain Time</SelectItem>

<SelectItem value="America/Los\_Angeles">Pacific Time</SelectItem>

<SelectItem value="Europe/London">London</SelectItem>

<SelectItem value="Asia/Dubai">Dubai</SelectItem>

<SelectItem value="Asia/Riyadh">Riyadh</SelectItem>

</SelectContent>

</Select>

</div>

</>

)}

</div>

{/\* Action Buttons \*/}

<div className="space-y-3 pt-4">

{isEditing ? (

<div className="flex space-x-3">

<Button

onClick={() => onSave(formData)}

className="flex-1 bg-emerald-600 hover:bg-emerald-700"

disabled={isUploading}

>

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

Save Changes

</Button>

<Button

variant="outline"

onClick={onCancel}

className="flex-1 border-slate-600 text-slate-300 hover:bg-slate-700"

>

Cancel

</Button>

</div>

) : (

<Button

onClick={onChangePassword}

className="w-full bg-blue-600 hover:bg-blue-700 transition-all duration-200"

>

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

Change Password

</Button>

)}

</div>

</CardContent>

</Card>

);

};

**Enhanced SubscriptionCard Component**

const SubscriptionCard: React.FC<{

subscription: Subscription | null;

onUpgrade: () => void;

onDowngrade: () => void;

onCancelSubscription: () => void;

}> = ({ subscription, onUpgrade, onDowngrade, onCancelSubscription }) => {

const daysUntilExpiry = subscription

? Math.ceil((new Date(subscription.expiresAt).getTime() - Date.now()) / (1000 \* 60 \* 60 \* 24))

: 0;

const isExpiringSoon = daysUntilExpiry <= 7;

return (

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700 hover:bg-slate-800/70 transition-all duration-300">

<CardHeader>

<CardTitle className="text-lg text-white flex items-center">

<Shield className="h-5 w-5 mr-2 text-green-400" />

Subscription

</CardTitle>

</CardHeader>

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

{subscription ? (

<>

{/\* Plan Information \*/}

<div className="bg-slate-700/50 p-4 rounded-lg space-y-3">

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

<span className="text-slate-400">Plan</span>

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

<Badge className="bg-emerald-600 text-white">

{subscription.tier}

</Badge>

{subscription.tier !== 'Pro Trader' && (

<Button size="sm" variant="outline" onClick={onUpgrade}>

Upgrade

</Button>

)}

</div>

</div>

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

<span className="text-slate-400">Status</span>

<div className="flex items-center">

<div className={`w-2 h-2 rounded-full mr-2 ${

subscription.active ? 'bg-green-400' : 'bg-red-400'

}`} />

<span className={subscription.active ? 'text-green-400' : 'text-red-400'}>

{subscription.active ? 'Active' : 'Inactive'}

</span>

</div>

</div>

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

<span className="text-slate-400">Next Billing</span>

<div className="text-right">

<div className={`text-white ${isExpiringSoon ? 'text-orange-400' : ''}`}>

{formatDate(subscription.expiresAt)}

</div>

{isExpiringSoon && (

<div className="text-xs text-orange-400">

Expires in {daysUntilExpiry} days

</div>

)}

</div>

</div>

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

<span className="text-slate-400">Amount</span>

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

${subscription.price}/month

</span>

</div>

</div>

{/\* Subscription Features \*/}

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

<h4 className="text-white font-medium">Plan Features</h4>

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

{getSubscriptionFeatures(subscription.tier).map((feature, index) => (

<div key={index} className="flex items-center text-sm">

<CheckCircle className="h-4 w-4 text-green-400 mr-2" />

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

</div>

))}

</div>

</div>

{/\* Action Buttons \*/}

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

<Button className="w-full bg-blue-600 hover:bg-blue-700">

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

Manage Billing

</Button>

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

{subscription.tier !== 'Basic' && (

<Button

variant="outline"

size="sm"

onClick={onDowngrade}

className="flex-1 border-slate-600 text-slate-300"

>

Downgrade

</Button>

)}

<Button

variant="outline"

size="sm"

onClick={onCancelSubscription}

className="flex-1 border-red-600 text-red-400 hover:bg-red-600 hover:text-white"

>

Cancel

</Button>

</div>

</div>

</>

) : (

/\* No Subscription State \*/

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

<div className="mb-4">

<Shield className="h-12 w-12 text-slate-500 mx-auto mb-3" />

<h3 className="text-lg font-medium text-white mb-2">No Active Subscription</h3>

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

Subscribe to unlock premium features and advanced trading signals

</p>

</div>

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

<Button className="w-full bg-emerald-600 hover:bg-emerald-700">

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

Subscribe to Pro Trader

</Button>

<Button variant="outline" className="w-full border-slate-600 text-slate-300">

View All Plans

</Button>

</div>

</div>

)}

</CardContent>

</Card>

);

};

**2. State Management (Zustand)**

**Profile Store Structure**

interface ProfileStore {

// User Data State

user: User | null;

originalUser: User | null; // For edit cancellation

// UI State

isEditing: boolean;

isLoading: boolean;

isSaving: boolean;

error: string | null;

// Modal States

isChangePasswordOpen: boolean;

isProfilePictureOpen: boolean;

isSubscriptionModalOpen: boolean;

// Form State

profileForm: {

name: string;

phone: string;

timezone: string;

language: string;

};

// Actions

setUser: (user: User) => void;

updateProfile: (data: ProfileUpdateData) => Promise<void>;

uploadProfilePicture: (file: File) => Promise<void>;

changePassword: (data: PasswordChangeData) => Promise<void>;

startEditing: () => void;

cancelEditing: () => void;

saveProfile: () => Promise<void>;

// Modal Actions

openChangePassword: () => void;

closeChangePassword: () => void;

openProfilePicture: () => void;

closeProfilePicture: () => void;

// Subscription Actions

upgradeSubscription: (planId: string) => Promise<void>;

downgradeSubscription: (planId: string) => Promise<void>;

cancelSubscription: () => Promise<void>;

}

// Zustand Store Implementation

export const useProfileStore = create<ProfileStore>((set, get) => ({

// Initial State

user: null,

originalUser: null,

isEditing: false,

isLoading: false,

isSaving: false,

error: null,

isChangePasswordOpen: false,

isProfilePictureOpen: false,

isSubscriptionModalOpen: false,

profileForm: {

name: '',

phone: '',

timezone: 'UTC',

language: 'en'

},

// User Data Actions

setUser: (user) => {

set({

user,

profileForm: {

name: user.name || '',

phone: user.phone || '',

timezone: user.timezone || 'UTC',

language: user.language || 'en'

}

});

},

// Profile Management

startEditing: () => {

const { user } = get();

set({

isEditing: true,

originalUser: user,

profileForm: {

name: user?.name || '',

phone: user?.phone || '',

timezone: user?.timezone || 'UTC',

language: user?.language || 'en'

}

});

},

cancelEditing: () => {

const { originalUser } = get();

set({

isEditing: false,

user: originalUser,

originalUser: null,

error: null

});

},

updateProfile: async (data) => {

set({ isSaving: true, error: null });

try {

const response = await fetch('/api/profile', {

method: 'PUT',

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

body: JSON.stringify(data)

});

if (!response.ok) {

throw new Error('Failed to update profile');

}

const updatedUser = await response.json();

set({

user: updatedUser,

isEditing: false,

originalUser: null,

isSaving: false

});

// Show success toast

toast.success('Profile updated successfully');

} catch (error) {

set({

error: error.message,

isSaving: false

});

toast.error('Failed to update profile');

}

},

uploadProfilePicture: async (file) => {

set({ isSaving: true, error: null });

try {

const formData = new FormData();

formData.append('profilePicture', file);

const response = await fetch('/api/profile/picture', {

method: 'POST',

body: formData

});

if (!response.ok) {

throw new Error('Failed to upload profile picture');

}

const { profilePictureUrl } = await response.json();

set(state => ({

user: { ...state.user!, profilePicture: profilePictureUrl },

isSaving: false,

isProfilePictureOpen: false

}));

toast.success('Profile picture updated successfully');

} catch (error) {

set({

error: error.message,

isSaving: false

});

toast.error('Failed to upload profile picture');

}

},

changePassword: async (data) => {

set({ isSaving: true, error: null });

try {

const response = await fetch('/api/auth/change-password', {

method: 'POST',

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

body: JSON.stringify(data)

});

if (!response.ok) {

const errorData = await response.json();

throw new Error(errorData.message);

}

set({

isSaving: false,

isChangePasswordOpen: false

});

toast.success('Password changed successfully');

} catch (error) {

set({

error: error.message,

isSaving: false

});

toast.error(error.message);

}

},

// Modal Actions

openChangePassword: () => set({ isChangePasswordOpen: true }),

closeChangePassword: () => set({ isChangePasswordOpen: false, error: null }),

openProfilePicture: () => set({ isProfilePictureOpen: true }),

closeProfilePicture: () => set({ isProfilePictureOpen: false }),

// Subscription Actions

upgradeSubscription: async (planId) => {

set({ isLoading: true, error: null });

try {

const response = await fetch('/api/subscription/upgrade', {

method: 'POST',

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

body: JSON.stringify({ planId })

});

if (!response.ok) {

throw new Error('Failed to upgrade subscription');

}

const updatedUser = await response.json();

set({ user: updatedUser, isLoading: false });

toast.success('Subscription upgraded successfully');

} catch (error) {

set({ error: error.message, isLoading: false });

toast.error('Failed to upgrade subscription');

}

},

cancelSubscription: async () => {

set({ isLoading: true, error: null });

try {

const response = await fetch('/api/subscription/cancel', {

method: 'POST'

});

if (!response.ok) {

throw new Error('Failed to cancel subscription');

}

const updatedUser = await response.json();

set({ user: updatedUser, isLoading: false });

toast.success('Subscription cancelled successfully');

} catch (error) {

set({ error: error.message, isLoading: false });

toast.error('Failed to cancel subscription');

}

}

}));

**3. API Contracts & Integration**

**API Endpoints**

// GET /api/profile - Get user profile

interface ProfileResponse {

user: {

id: string;

name: string;

email: string;

phone?: string;

profilePicture?: string;

timezone: string;

language: string;

emailVerified: boolean;

createdAt: string;

updatedAt: string;

subscription?: {

id: string;

tier: 'Basic' | 'Pro Trader' | 'Elite';

active: boolean;

price: number;

expiresAt: string;

features: string[];

};

};

}

// PUT /api/profile - Update user profile

interface ProfileUpdateRequest {

name?: string;

phone?: string;

timezone?: string;

language?: string;

}

interface ProfileUpdateResponse {

user: User;

message: string;

}

// POST /api/profile/picture - Upload profile picture

interface ProfilePictureResponse {

profilePictureUrl: string;

message: string;

}

// POST /api/auth/change-password - Change password

interface ChangePasswordRequest {

currentPassword: string;

newPassword: string;

confirmPassword: string;

}

interface ChangePasswordResponse {

message: string;

success: boolean;

}

// POST /api/subscription/upgrade - Upgrade subscription

interface SubscriptionUpgradeRequest {

planId: string;

paymentMethodId?: string;

}

interface SubscriptionUpgradeResponse {

user: User;

paymentIntent?: {

clientSecret: string;

status: string;

};

}

// GET /api/subscription/plans - Get available plans

interface SubscriptionPlansResponse {

plans: {

id: string;

name: string;

price: number;

features: string[];

recommended: boolean;

}[];

}

// Error Response Format

interface ErrorResponse {

error: string;

message: string;

code?: string;

details?: Record<string, any>;

}

**4. Performance & Optimization**

**Lazy Loading Strategies**

// Lazy load heavy components

const ChangePasswordDialog = lazy(() => import('../components/profile/ChangePasswordDialog'));

const ProfilePictureModal = lazy(() => import('../components/profile/ProfilePictureModal'));

const SubscriptionManagement = lazy(() => import('../components/profile/SubscriptionManagement'));

const NotificationPreferences = lazy(() => import('../components/profile/NotificationPreferences'));

// Lazy load third-party libraries

const ImageCropper = lazy(() => import('react-image-crop'));

const DatePicker = lazy(() => import('react-datepicker'));

**Memoization Opportunities**

// Memoized PersonalInfoCard

const PersonalInfoCard = React.memo(({

user,

isEditing,

onSave,

onCancel

}) => {

// Component implementation

}, (prevProps, nextProps) => {

return prevProps.user.id === nextProps.user.id &&

prevProps.user.updatedAt === nextProps.user.updatedAt &&

prevProps.isEditing === nextProps.isEditing;

});

// Memoized subscription features calculation

const useSubscriptionFeatures = (tier: string) => {

return useMemo(() => {

const featureMap = {

'Basic': ['Basic signals', 'Email alerts', 'Mobile app'],

'Pro Trader': ['Advanced signals', 'Real-time alerts', 'Paper trading', 'Premium support'],

'Elite': ['All Pro features', 'Custom indicators', 'API access', 'Priority support']

};

return featureMap[tier] || [];

}, [tier]);

};

// Memoized date formatting

const useFormattedDate = (dateString: string) => {

return useMemo(() => {

return new Intl.DateTimeFormat('en-US', {

year: 'numeric',

month: 'long',

day: 'numeric'

}).format(new Date(dateString));

}, [dateString]);

};

**Image Upload Optimization**

const useOptimizedImageUpload = () => {

const compressImage = useCallback(async (file: File): Promise<File> => {

const canvas = document.createElement('canvas');

const ctx = canvas.getContext('2d')!;

const img = new Image();

return new Promise((resolve) => {

img.onload = () => {

// Resize to max 300x300

const maxSize = 300;

const ratio = Math.min(maxSize / img.width, maxSize / img.height);

canvas.width = img.width \* ratio;

canvas.height = img.height \* ratio;

ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

canvas.toBlob((blob) => {

resolve(new File([blob!], file.name, { type: 'image/jpeg' }));

}, 'image/jpeg', 0.8);

};

img.src = URL.createObjectURL(file);

});

}, []);

return { compressImage };

};

**5. Database Schema**

**PostgreSQL Table Structures**

-- Users table (enhanced)

CREATE TABLE users (

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

email VARCHAR(255) UNIQUE NOT NULL,

name VARCHAR(255) NOT NULL,

phone VARCHAR(20),

profile\_picture\_url TEXT,

timezone VARCHAR(50) DEFAULT 'UTC',

language VARCHAR(5) DEFAULT 'en',

email\_verified BOOLEAN DEFAULT false,

email\_verification\_token VARCHAR(255),

password\_hash VARCHAR(255) NOT NULL,

last\_password\_change TIMESTAMP WITH TIME ZONE,

failed\_login\_attempts INTEGER DEFAULT 0,

locked\_until TIMESTAMP WITH TIME ZONE,

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

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

-- Indexes

INDEX idx\_users\_email (email),

INDEX idx\_users\_email\_verified (email\_verified),

INDEX idx\_users\_created\_at (created\_at DESC)

);

-- Subscriptions table

CREATE TABLE subscriptions (

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

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

stripe\_subscription\_id VARCHAR(255) UNIQUE,

stripe\_customer\_id VARCHAR(255),

tier VARCHAR(20) NOT NULL CHECK (tier IN ('Basic', 'Pro Trader', 'Elite')),

status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'canceled', 'past\_due', 'incomplete')),

current\_period\_start TIMESTAMP WITH TIME ZONE NOT NULL,

current\_period\_end TIMESTAMP WITH TIME ZONE NOT NULL,

price\_cents INTEGER NOT NULL,

currency VARCHAR(3) DEFAULT 'USD',

trial\_end TIMESTAMP WITH TIME ZONE,

canceled\_at TIMESTAMP WITH TIME ZONE,

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

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

-- Indexes

INDEX idx\_subscriptions\_user\_id (user\_id),

INDEX idx\_subscriptions\_status (status),

INDEX idx\_subscriptions\_stripe\_id (stripe\_subscription\_id),

INDEX idx\_subscriptions\_period\_end (current\_period\_end)

);

-- User preferences table

CREATE TABLE user\_preferences (

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

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

-- Notification preferences

email\_notifications BOOLEAN DEFAULT true,

telegram\_notifications BOOLEAN DEFAULT false,

sms\_notifications BOOLEAN DEFAULT false,

push\_notifications BOOLEAN DEFAULT true,

-- Signal preferences

signal\_score\_threshold INTEGER DEFAULT 80,

preferred\_markets TEXT[] DEFAULT ARRAY['usa'],

preferred\_sectors TEXT[] DEFAULT ARRAY['tech', 'finance'],

preferred\_timeframes TEXT[] DEFAULT ARRAY['1D', '4H'],

-- UI preferences

dark\_mode BOOLEAN DEFAULT true,

compact\_view BOOLEAN DEFAULT false,

auto\_refresh BOOLEAN DEFAULT true,

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

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

UNIQUE(user\_id)

);

-- Profile picture uploads table

CREATE TABLE profile\_pictures (

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

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

original\_filename VARCHAR(255) NOT NULL,

file\_path TEXT NOT NULL,

file\_size INTEGER NOT NULL,

mime\_type VARCHAR(100) NOT NULL,

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

INDEX idx\_profile\_pictures\_user\_id (user\_id)

);

-- Password reset tokens table

CREATE TABLE password\_reset\_tokens (

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

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

token VARCHAR(255) NOT NULL UNIQUE,

expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,

used\_at TIMESTAMP WITH TIME ZONE,

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

INDEX idx\_password\_reset\_tokens\_token (token),

INDEX idx\_password\_reset\_tokens\_user\_id (user\_id),

INDEX idx\_password\_reset\_tokens\_expires\_at (expires\_at)

);

-- User sessions table (for enhanced security)

CREATE TABLE user\_sessions (

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

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

session\_token VARCHAR(255) NOT NULL UNIQUE,

ip\_address INET,

user\_agent TEXT,

expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,

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

last\_accessed TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

INDEX idx\_user\_sessions\_token (session\_token),

INDEX idx\_user\_sessions\_user\_id (user\_id),

INDEX idx\_user\_sessions\_expires\_at (expires\_at)

);

**6. User Experience**

**Loading States & Skeleton Screens**

const ProfileSkeleton = () => (

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

<div className="mb-8">

<div className="h-8 bg-slate-700 rounded w-32 mb-2 animate-pulse" />

<div className="h-4 bg-slate-700 rounded w-64 animate-pulse" />

</div>

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

{/\* Personal Info Skeleton \*/}

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

<CardHeader>

<div className="h-6 bg-slate-700 rounded w-48 animate-pulse" />

</CardHeader>

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

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

<div className="h-16 w-16 bg-slate-700 rounded-full animate-pulse" />

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

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

<div className="h-3 bg-slate-700 rounded w-32 animate-pulse" />

</div>

</div>

{Array.from({ length: 2 }).map((\_, i) => (

<div key={i} className="space-y-2">

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

<div className="h-10 bg-slate-700 rounded w-full animate-pulse" />

</div>

))}

<div className="h-10 bg-slate-700 rounded w-full animate-pulse" />

</CardContent>

</Card>

{/\* Subscription Skeleton \*/}

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

<CardHeader>

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

</CardHeader>

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

<div className="bg-slate-700/50 p-4 rounded-lg space-y-3">

{Array.from({ length: 4 }).map((\_, i) => (

<div key={i} className="flex justify-between items-center">

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

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

</div>

))}

</div>

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

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

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

{Array.from({ length: 3 }).map((\_, i) => (

<div key={i} className="h-3 bg-slate-700 rounded w-full animate-pulse" />

))}

</div>

</div>

<div className="h-10 bg-slate-700 rounded w-full animate-pulse" />

</CardContent>

</Card>

</div>

</div>

);

const SaveSuccessAnimation = () => (

<motion.div

initial={{ scale: 0, opacity: 0 }}

animate={{ scale: 1, opacity: 1 }}

exit={{ scale: 0, opacity: 0 }}

className="fixed top-4 right-4 bg-emerald-600 text-white p-4 rounded-lg shadow-lg flex items-center space-x-2 z-50"

>

<CheckCircle className="h-5 w-5" />

<span>Profile saved successfully!</span>

</motion.div>

);

**Error Boundaries & Fallback UI**

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

return (

<ErrorBoundary

fallback={

<div className="max-w-4xl mx-auto px-4 py-16 text-center">

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

<h2 className="text-xl font-semibold text-white mb-2">

Unable to load profile

</h2>

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

There was an error loading your profile information. Please try refreshing the page.

</p>

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

Refresh Page

</Button>

</div>

}

>

{children}

</ErrorBoundary>

);

};

const FormErrorDisplay = ({ error }: { error: string | null }) => {

if (!error) return null;

return (

<motion.div

initial={{ opacity: 0, y: -10 }}

animate={{ opacity: 1, y: 0 }}

exit={{ opacity: 0, y: -10 }}

className="bg-red-500/10 border border-red-500/20 rounded-lg p-3 mb-4"

>

<div className="flex items-center">

<AlertCircle className="h-4 w-4 text-red-400 mr-2" />

<span className="text-red-300 text-sm">{error}</span>

</div>

</motion.div>

);

};

**Accessibility Features**

// Enhanced form accessibility

const AccessibleFormField = ({

label,

error,

required,

children,

description

}: {

label: string;

error?: string;

required?: boolean;

children: React.ReactNode;

description?: string;

}) => {

const fieldId = useId();

const errorId = useId();

const descriptionId = useId();

return (

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

<Label

htmlFor={fieldId}

className="text-slate-300 text-sm font-medium"

>

{label}

{required && <span className="text-red-400 ml-1" aria-label="required">\*</span>}

</Label>

{description && (

<p id={descriptionId} className="text-xs text-slate-400">

{description}

</p>

)}

{React.cloneElement(children as React.ReactElement, {

id: fieldId,

'aria-describedby': [

error ? errorId : '',

description ? descriptionId : ''

].filter(Boolean).join(' '),

'aria-invalid': !!error,

'aria-required': required

})}

{error && (

<p id={errorId} className="text-red-400 text-xs" role="alert">

{error}

</p>

)}

</div>

);

};

// Keyboard navigation for profile cards

const useKeyboardNavigation = () => {

useEffect(() => {

const handleKeyDown = (event: KeyboardEvent) => {

if (event.key === 'Tab' && event.shiftKey) {

// Handle backward tab navigation

} else if (event.key === 'Tab') {

// Handle forward tab navigation

} else if (event.key === 'Escape') {

// Close any open modals

useProfileStore.getState().closeChangePassword();

useProfileStore.getState().closeProfilePicture();

}

};

document.addEventListener('keydown', handleKeyDown);

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

}, []);

};

**7. Integration Points**

**Navigation Patterns**

// Profile navigation configuration

const profileRoutes = {

profile: '/profile',

billing: '/profile/billing',

preferences: '/profile/preferences',

security: '/profile/security',

notifications: '/profile/notifications'

};

// Profile navigation hook

const useProfileNavigation = () => {

const navigate = useNavigate();

const location = useLocation();

const navigateToSection = (section: keyof typeof profileRoutes) => {

navigate(profileRoutes[section]);

};

const isActiveSection = (section: keyof typeof profileRoutes) => {

return location.pathname === profileRoutes[section];

};

return { navigateToSection, isActiveSection };

};

// Breadcrumb navigation

const ProfileBreadcrumbs = () => {

const location = useLocation();

const breadcrumbs = useMemo(() => {

const path = location.pathname;

const segments = path.split('/').filter(Boolean);

return segments.map((segment, index) => ({

label: segment.charAt(0).toUpperCase() + segment.slice(1),

path: '/' + segments.slice(0, index + 1).join('/'),

isLast: index === segments.length - 1

}));

}, [location.pathname]);

return (

<nav aria-label="Breadcrumb" className="mb-6">

<ol className="flex items-center space-x-2 text-sm">

{breadcrumbs.map((crumb, index) => (

<li key={crumb.path} className="flex items-center">

{index > 0 && <ChevronRight className="h-4 w-4 text-slate-400 mx-2" />}

{crumb.isLast ? (

<span className="text-white font-medium">{crumb.label}</span>

) : (

<Link to={crumb.path} className="text-slate-400 hover:text-white transition-colors">

{crumb.label}

</Link>

)}

</li>

))}

</ol>

</nav>

);

};

**Cross-Component Communication**

// Profile event system

export const profileEventEmitter = new EventTarget();

export const ProfileEvents = {

PROFILE\_UPDATED: 'profile:updated',

SUBSCRIPTION\_CHANGED: 'subscription:changed',

PASSWORD\_CHANGED: 'password:changed',

PICTURE\_UPLOADED: 'picture:uploaded'

} as const;

// Event emission helpers

export const emitProfileEvent = (eventType: string, data: any) => {

profileEventEmitter.dispatchEvent(

new CustomEvent(eventType, { detail: data })

);

};

// Profile event listener hook

export const useProfileEvents = () => {

useEffect(() => {

const handleProfileUpdate = (event: CustomEvent) => {

const updatedUser = event.detail;

// Update global auth context

useAuthStore.getState().setUser(updatedUser);

// Update dashboard metrics if needed

useDashboardStore.getState().refreshUserData();

};

const handleSubscriptionChange = (event: CustomEvent) => {

const subscription = event.detail;

// Update billing information

// Refresh feature access

// Update navigation permissions

};

profileEventEmitter.addEventListener(ProfileEvents.PROFILE\_UPDATED, handleProfileUpdate);

profileEventEmitter.addEventListener(ProfileEvents.SUBSCRIPTION\_CHANGED, handleSubscriptionChange);

return () => {

profileEventEmitter.removeEventListener(ProfileEvents.PROFILE\_UPDATED, handleProfileUpdate);

profileEventEmitter.removeEventListener(ProfileEvents.SUBSCRIPTION\_CHANGED, handleSubscriptionChange);

};

}, []);

};

**8. Testing Strategy**

**Unit Test Requirements**

// PersonalInfoCard.test.tsx

describe('PersonalInfoCard', () => {

const mockUser = {

id: '1',

name: 'John Doe',

email: 'john@example.com',

phone: '+1234567890',

emailVerified: true,

createdAt: '2024-01-01T00:00:00Z'

};

it('displays user information correctly', () => {

render(<PersonalInfoCard user={mockUser} isEditing={false} />);

expect(screen.getByDisplayValue('John Doe')).toBeInTheDocument();

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

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

});

it('switches to edit mode when edit button is clicked', () => {

const mockOnEdit = jest.fn();

render(<PersonalInfoCard user={mockUser} onEditProfile={mockOnEdit} isEditing={false} />);

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

expect(mockOnEdit).toHaveBeenCalled();

});

it('shows save and cancel buttons in edit mode', () => {

render(<PersonalInfoCard user={mockUser} isEditing={true} />);

expect(screen.getByText('Save Changes')).toBeInTheDocument();

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

});

it('validates required fields before saving', async () => {

const mockOnSave = jest.fn();

render(<PersonalInfoCard user={mockUser} isEditing={true} onSave={mockOnSave} />);

// Clear name field

const nameInput = screen.getByDisplayValue('John Doe');

fireEvent.change(nameInput, { target: { value: '' } });

fireEvent.click(screen.getByText('Save Changes'));

expect(screen.getByText('Name is required')).toBeInTheDocument();

expect(mockOnSave).not.toHaveBeenCalled();

});

});

// SubscriptionCard.test.tsx

describe('SubscriptionCard', () => {

const mockActiveSubscription = {

tier: 'Pro Trader',

active: true,

price: 49,

expiresAt: '2025-12-31T23:59:59Z'

};

it('displays subscription information for active subscription', () => {

render(<SubscriptionCard subscription={mockActiveSubscription} />);

expect(screen.getByText('Pro Trader')).toBeInTheDocument();

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

expect(screen.getByText('$49/month')).toBeInTheDocument();

});

it('shows subscription prompt for users without subscription', () => {

render(<SubscriptionCard subscription={null} />);

expect(screen.getByText('No Active Subscription')).toBeInTheDocument();

expect(screen.getByText('Subscribe to Pro Trader')).toBeInTheDocument();

});

it('shows warning for expiring subscriptions', () => {

const expiringSoon = {

...mockActiveSubscription,

expiresAt: new Date(Date.now() + 5 \* 24 \* 60 \* 60 \* 1000).toISOString() // 5 days from now

};

render(<SubscriptionCard subscription={expiringSoon} />);

expect(screen.getByText(/Expires in 5 days/)).toBeInTheDocument();

});

});

// Profile integration tests

describe('Profile Page Integration', () => {

beforeEach(() => {

// Mock API responses

fetchMock.get('/api/profile', {

user: mockUser

});

});

it('loads and displays profile data on mount', async () => {

render(<Profile />);

await waitFor(() => {

expect(screen.getByText('John Doe')).toBeInTheDocument();

});

});

it('handles profile update flow correctly', async () => {

fetchMock.put('/api/profile', {

user: { ...mockUser, name: 'Jane Doe' }

});

render(<Profile />);

// Start editing

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

// Update name

const nameInput = screen.getByDisplayValue('John Doe');

fireEvent.change(nameInput, { target: { value: 'Jane Doe' } });

// Save changes

fireEvent.click(screen.getByText('Save Changes'));

await waitFor(() => {

expect(screen.getByText('Profile updated successfully')).toBeInTheDocument();

});

});

});

**9. Charts & Data Visualizations**

**Subscription Usage Analytics**

// Subscription usage chart component

const SubscriptionUsageChart = ({ subscription }: { subscription: Subscription }) => {

const usageData = useMemo(() => {

const currentDate = new Date();

const startDate = new Date(subscription.current\_period\_start);

const endDate = new Date(subscription.current\_period\_end);

const totalDays = Math.ceil((endDate.getTime() - startDate.getTime()) / (1000 \* 60 \* 60 \* 24));

const daysUsed = Math.ceil((currentDate.getTime() - startDate.getTime()) / (1000 \* 60 \* 60 \* 24));

const daysRemaining = totalDays - daysUsed;

return {

used: Math.max(0, daysUsed),

remaining: Math.max(0, daysRemaining),

total: totalDays,

percentage: Math.min(100, Math.max(0, (daysUsed / totalDays) \* 100))

};

}, [subscription]);

return (

<div className="bg-slate-700/30 p-4 rounded-lg">

<div className="flex justify-between items-center mb-3">

<h4 className="text-sm font-medium text-slate-300">Billing Period</h4>

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

{usageData.used} of {usageData.total} days used

</span>

</div>

<div className="w-full bg-slate-600 rounded-full h-2 mb-2">

<div

className="bg-gradient-to-r from-blue-400 to-blue-600 h-2 rounded-full transition-all duration-500"

style={{ width: `${usageData.percentage}%` }}

/>

</div>

<div className="flex justify-between text-xs text-slate-400">

<span>{formatDate(subscription.current\_period\_start)}</span>

<span>{formatDate(subscription.current\_period\_end)}</span>

</div>

</div>

);

};

// Feature usage breakdown

const FeatureUsageBreakdown = ({ userId }: { userId: string }) => {

const [usageData, setUsageData] = useState(null);

useEffect(() => {

fetch(`/api/analytics/usage/${userId}`)

.then(res => res.json())

.then(setUsageData);

}, [userId]);

if (!usageData) return <div>Loading usage data...</div>;

return (

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

<h4 className="text-sm font-medium text-slate-300">Feature Usage This Month</h4>

{usageData.features.map((feature, index) => (

<div key={feature.name} className="flex items-center justify-between">

<span className="text-slate-400 text-sm">{feature.name}</span>

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

<div className="w-20 bg-slate-600 rounded-full h-1">

<div

className="bg-emerald-400 h-1 rounded-full"

style={{ width: `${(feature.used / feature.limit) \* 100}%` }}

/>

</div>

<span className="text-xs text-slate-400 w-16 text-right">

{feature.used}/{feature.limit}

</span>

</div>

</div>

))}

</div>

);

};

**10. Visual Data Elements**

**Profile Completion Progress**

const ProfileCompletionProgress = ({ user }: { user: User }) => {

const completionData = useMemo(() => {

const fields = [

{ key: 'name', label: 'Full Name', completed: !!user.name },

{ key: 'email', label: 'Email', completed: !!user.email },

{ key: 'phone', label: 'Phone', completed: !!user.phone },

{ key: 'profilePicture', label: 'Profile Picture', completed: !!user.profilePicture },

{ key: 'emailVerified', label: 'Email Verification', completed: user.emailVerified },

];

const completedCount = fields.filter(field => field.completed).length;

const percentage = (completedCount / fields.length) \* 100;

return { fields, completedCount, percentage, total: fields.length };

}, [user]);

return (

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

<CardHeader>

<CardTitle className="text-lg text-white flex items-center justify-between">

<span>Profile Completion</span>

<Badge className={`${

completionData.percentage === 100 ? 'bg-emerald-600' :

completionData.percentage >= 60 ? 'bg-blue-600' : 'bg-yellow-600'

} text-white`}>

{Math.round(completionData.percentage)}%

</Badge>

</CardTitle>

</CardHeader>

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

<div className="w-full bg-slate-600 rounded-full h-3">

<div

className={`h-3 rounded-full transition-all duration-700 ${

completionData.percentage === 100 ? 'bg-gradient-to-r from-emerald-400 to-emerald-600' :

completionData.percentage >= 60 ? 'bg-gradient-to-r from-blue-400 to-blue-600' :

'bg-gradient-to-r from-yellow-400 to-yellow-600'

}`}

style={{ width: `${completionData.percentage}%` }}

/>

</div>

<div className="text-center text-sm text-slate-400">

{completionData.completedCount} of {completionData.total} sections completed

</div>

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

{completionData.fields.map((field) => (

<div key={field.key} className="flex items-center justify-between">

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

<div className="flex items-center">

{field.completed ? (

<CheckCircle className="h-4 w-4 text-emerald-400" />

) : (

<Circle className="h-4 w-4 text-slate-500" />

)}

</div>

</div>

))}

</div>

{completionData.percentage < 100 && (

<div className="bg-blue-500/10 border border-blue-500/20 rounded-lg p-3 mt-4">

<p className="text-blue-300 text-sm">

Complete your profile to unlock all features and improve your trading experience.

</p>

</div>

)}

</CardContent>

</Card>

);

};

**Account Security Score**

const SecurityScoreIndicator = ({ user }: { user: User }) => {

const securityScore = useMemo(() => {

let score = 0;

const factors = [];

// Email verification (25 points)

if (user.emailVerified) {

score += 25;

factors.push({ name: 'Email verified', completed: true, points: 25 });

} else {

factors.push({ name: 'Email verified', completed: false, points: 25 });

}

// Strong password (25 points) - would need to check password strength

const hasStrongPassword = user.lastPasswordChange &&

new Date(user.lastPasswordChange) > new Date(Date.now() - 90 \* 24 \* 60 \* 60 \* 1000);

if (hasStrongPassword) {

score += 25;

factors.push({ name: 'Recent password change', completed: true, points: 25 });

} else {

factors.push({ name: 'Recent password change', completed: false, points: 25 });

}

// Phone number (20 points)

if (user.phone) {

score += 20;

factors.push({ name: 'Phone number added', completed: true, points: 20 });

} else {

factors.push({ name: 'Phone number added', completed: false, points: 20 });

}

// Profile picture (10 points)

if (user.profilePicture) {

score += 10;

factors.push({ name: 'Profile picture', completed: true, points: 10 });

} else {

factors.push({ name: 'Profile picture', completed: false, points: 10 });

}

// Recent activity (20 points)

const recentActivity = new Date(user.updatedAt) > new Date(Date.now() - 30 \* 24 \* 60 \* 60 \* 1000);

if (recentActivity) {

score += 20;

factors.push({ name: 'Recent account activity', completed: true, points: 20 });

} else {

factors.push({ name: 'Recent account activity', completed: false, points: 20 });

}

return { score, factors, maxScore: 100 };

}, [user]);

const getScoreColor = (score: number) => {

if (score >= 80) return 'text-emerald-400';

if (score >= 60) return 'text-yellow-400';

return 'text-red-400';

};

const getScoreLevel = (score: number) => {

if (score >= 80) return 'Excellent';

if (score >= 60) return 'Good';

if (score >= 40) return 'Fair';

return 'Poor';

};

return (

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

<CardHeader>

<CardTitle className="text-lg text-white flex items-center">

<Shield className="h-5 w-5 mr-2 text-blue-400" />

Security Score

</CardTitle>

</CardHeader>

<CardContent>

<div className="text-center mb-6">

<div className={`text-4xl font-bold ${getScoreColor(securityScore.score)} mb-2`}>

{securityScore.score}/100

</div>

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

{getScoreLevel(securityScore.score)} Security

</div>

</div>

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

{securityScore.factors.map((factor, index) => (

<div key={index} className="flex items-center justify-between">

<div className="flex items-center">

{factor.completed ? (

<CheckCircle className="h-4 w-4 text-emerald-400 mr-2" />

) : (

<XCircle className="h-4 w-4 text-slate-500 mr-2" />

)}

<span className={`text-sm ${factor.completed ? 'text-slate-300' : 'text-slate-400'}`}>

{factor.name}

</span>

</div>

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

+{factor.points}

</span>

</div>

))}

</div>

{securityScore.score < 80 && (

<div className="bg-yellow-500/10 border border-yellow-500/20 rounded-lg p-3 mt-4">

<p className="text-yellow-300 text-sm">

Improve your security score by completing the missing items above.

</p>

</div>

)}

</CardContent>

</Card>

);

};

**11. Security & Validation**

**Input Validation Schemas**

import { z } from 'zod';

// Profile update validation

export const profileUpdateSchema = z.object({

name: z.string()

.min(2, 'Name must be at least 2 characters')

.max(100, 'Name must not exceed 100 characters')

.regex(/^[a-zA-Z\s'-]+$/, 'Name can only contain letters, spaces, hyphens, and apostrophes'),

phone: z.string()

.optional()

.refine(

(val) => !val || /^\+?[1-9]\d{1,14}$/.test(val),

'Invalid phone number format'

),

timezone: z.string()

.refine(

(val) => Intl.supportedValuesOf('timeZone').includes(val),

'Invalid timezone'

),

language: z.enum(['en', 'ar', 'de'], {

errorMap: () => ({ message: 'Invalid language selection' })

})

});

// Password change validation

export const passwordChangeSchema = z.object({

currentPassword: z.string()

.min(1, 'Current password is required'),

newPassword: z.string()

.min(8, 'Password must be at least 8 characters')

.max(128, 'Password must not exceed 128 characters')

.regex(/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]/,

'Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character'),

confirmPassword: z.string()

}).refine(

(data) => data.newPassword === data.confirmPassword,

{

message: "Passwords don't match",

path: ["confirmPassword"]

}

).refine(

(data) => data.currentPassword !== data.newPassword,

{

message: "New password must be different from current password",

path: ["newPassword"]

}

);

// Profile picture validation

export const profilePictureSchema = z.object({

file: z.instanceof(File)

.refine(

(file) => file.size <= 5 \* 1024 \* 1024, // 5MB

'File size must be less than 5MB'

)

.refine(

(file) => ['image/jpeg', 'image/png', 'image/webp'].includes(file.type),

'File must be a JPEG, PNG, or WebP image'

)

.refine(

(file) => {

return new Promise((resolve) => {

const img = new Image();

img.onload = () => {

resolve(img.width >= 100 && img.height >= 100 && img.width <= 2000 && img.height <= 2000);

};

img.src = URL.createObjectURL(file);

});

},

'Image dimensions must be between 100x100 and 2000x2000 pixels'

)

});

// Form validation hook

export const useFormValidation = <T>(schema: z.ZodSchema<T>) => {

const [errors, setErrors] = useState<Record<string, string>>({});

const validate = (data: T): boolean => {

try {

schema.parse(data);

setErrors({});

return true;

} catch (error) {

if (error instanceof z.ZodError) {

const fieldErrors: Record<string, string> = {};

error.errors.forEach((err) => {

if (err.path.length > 0) {

fieldErrors[err.path[0] as string] = err.message;

}

});

setErrors(fieldErrors);

}

return false;

}

};

const clearErrors = () => setErrors({});

const clearFieldError = (field: string) => {

setErrors(prev => {

const newErrors = { ...prev };

delete newErrors[field];

return newErrors;

});

};

return { validate, errors, clearErrors, clearFieldError };

};

**API Security Middleware**

// Rate limiting for profile endpoints

export const profileRateLimit = rateLimit({

windowMs: 15 \* 60 \* 1000, // 15 minutes

max: 10, // Limit each IP to 10 requests per windowMs

message: 'Too many profile update attempts, please try again later',

standardHeaders: true,

legacyHeaders: false,

});

// File upload security

export const secureFileUpload = multer({

storage: multer.memoryStorage(),

limits: {

fileSize: 5 \* 1024 \* 1024, // 5MB

files: 1

},

fileFilter: (req, file, cb) => {

const allowedMimes = ['image/jpeg', 'image/png', 'image/webp'];

if (allowedMimes.includes(file.mimetype)) {

cb(null, true);

} else {

cb(new Error('Invalid file type. Only JPEG, PNG, and WebP images are allowed.'));

}

}

});

// Input sanitization middleware

export const sanitizeProfileInput = (req: Request, res: Response, next: NextFunction) => {

if (req.body.name) {

req.body.name = validator.escape(req.body.name.trim());

}

if (req.body.phone) {

req.body.phone = req.body.phone.replace(/[^\d+\-\s]/g, '');

}

next();

};

// Password strength validation

export const validatePasswordStrength = (password: string): boolean => {

const minLength = 8;

const hasUpperCase = /[A-Z]/.test(password);

const hasLowerCase = /[a-z]/.test(password);

const hasNumbers = /\d/.test(password);

const hasSpecialChar = /[@$!%\*?&]/.test(password);

const hasNoCommonPatterns = !(/(.)\1{2,}/.test(password)); // No repeated characters

return password.length >= minLength &&

hasUpperCase &&

hasLowerCase &&

hasNumbers &&

hasSpecialChar &&

hasNoCommonPatterns;

};

**12. Environment & Configuration**

**Environment Variables**

// .env.local

NEXT\_PUBLIC\_API\_URL=http://localhost:3001

NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

NEXT\_PUBLIC\_STRIPE\_PUBLISHABLE\_KEY=pk\_test\_your\_stripe\_key

# Server-side only

DATABASE\_URL=postgresql://user:password@localhost:5432/kurzora

STRIPE\_SECRET\_KEY=sk\_test\_your\_stripe\_secret

SENDGRID\_API\_KEY=your\_sendgrid\_key

CLOUDINARY\_CLOUD\_NAME=your\_cloudinary\_name

CLOUDINARY\_API\_KEY=your\_cloudinary\_key

CLOUDINARY\_API\_SECRET=your\_cloudinary\_secret

JWT\_SECRET=your\_jwt\_secret\_key

ENCRYPTION\_KEY=your\_encryption\_key\_for\_sensitive\_data

**Feature Flags Configuration**

// Feature flags for profile functionality

export const profileFeatureFlags = {

profilePictureUpload: true,

phoneNumberVerification: process.env.NODE\_ENV === 'production',

twoFactorAuth: false, // Coming soon

advancedNotifications: true,

socialLogin: false, // Future enhancement

accountDeletion: process.env.NODE\_ENV === 'production',

subscriptionManagement: true,

billingHistory: true,

exportData: true, // GDPR compliance

darkModeToggle: true,

languageSwitcher: true

};

// Feature flag hook

export const useProfileFeatureFlag = (flag: keyof typeof profileFeatureFlags) => {

return profileFeatureFlags[flag];

};

// Dynamic feature configuration

export const getProfileConfig = () => ({

maxProfilePictureSize: 5 \* 1024 \* 1024, // 5MB

allowedImageTypes: ['image/jpeg', 'image/png', 'image/webp'],

supportedTimezones: Intl.supportedValuesOf('timeZone'),

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

passwordMinLength: 8,

sessionTimeout: 30 \* 24 \* 60 \* 60 \* 1000, // 30 days

// Subscription tiers

subscriptionTiers: {

basic: { price: 0, features: ['basic\_signals', 'email\_alerts'] },

pro: { price: 49, features: ['advanced\_signals', 'real\_time\_alerts', 'paper\_trading'] },

elite: { price: 99, features: ['all\_pro\_features', 'api\_access', 'priority\_support'] }

}

});

**Monitoring & Analytics**

// Profile analytics tracking

export const trackProfileEvent = (eventName: string, properties: Record<string, any>) => {

if (typeof window !== 'undefined') {

// Google Analytics

if (window.gtag) {

window.gtag('event', eventName, {

event\_category: 'Profile',

event\_label: properties.action,

custom\_parameter\_1: properties.user\_tier,

custom\_parameter\_2: properties.profile\_completion

});

}

// Mixpanel (if implemented)

if (window.mixpanel) {

window.mixpanel.track(`Profile ${eventName}`, properties);

}

}

};

// Error reporting for profile operations

export const reportProfileError = (error: Error, context: Record<string, any>) => {

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

// Sentry error reporting

console.error('Profile Error:', {

error: error.message,

stack: error.stack,

context,

timestamp: new Date().toISOString()

});

// Custom error logging

fetch('/api/errors', {

method: 'POST',

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

body: JSON.stringify({

type: 'profile\_error',

message: error.message,

context,

userAgent: navigator.userAgent

})

}).catch(() => {}); // Fail silently for error reporting

}

};

**13. Cross-Screen Data Flow**

**Global State Synchronization**

// Profile data synchronization with other stores

export const useProfileDataSync = () => {

const profileStore = useProfileStore();

const authStore = useAuthStore();

const dashboardStore = useDashboardStore();

const notificationStore = useNotificationStore();

// Sync profile updates with auth context

useEffect(() => {

if (profileStore.user) {

authStore.updateUser(profileStore.user);

}

}, [profileStore.user]);

// Sync subscription changes with dashboard

useEffect(() => {

if (profileStore.user?.subscription) {

dashboardStore.updateSubscriptionStatus(profileStore.user.subscription);

}

}, [profileStore.user?.subscription]);

// Sync notification preferences

useEffect(() => {

if (profileStore.user?.preferences) {

notificationStore.updatePreferences(profileStore.user.preferences);

}

}, [profileStore.user?.preferences]);

};

// Real-time profile updates

export const useRealtimeProfileUpdates = () => {

const profileStore = useProfileStore();

useEffect(() => {

const channel = supabase

.channel('profile\_updates')

.on('postgres\_changes', {

event: 'UPDATE',

schema: 'public',

table: 'users',

filter: `id=eq.${profileStore.user?.id}`

}, (payload) => {

profileStore.setUser(payload.new);

// Emit profile update event for other components

emitProfileEvent(ProfileEvents.PROFILE\_UPDATED, payload.new);

})

.subscribe();

return () => {

supabase.removeChannel(channel);

};

}, [profileStore.user?.id]);

};

**Data Persistence & Caching**

// Profile data caching strategy

export const useProfileCache = () => {

const queryClient = useQueryClient();

const cacheProfileData = (user: User) => {

// Cache user data for 5 minutes

queryClient.setQueryData(['profile', user.id], user, {

updatedAt: Date.now(),

cacheTime: 5 \* 60 \* 1000

});

// Cache subscription data separately for longer

if (user.subscription) {

queryClient.setQueryData(['subscription', user.id], user.subscription, {

updatedAt: Date.now(),

cacheTime: 15 \* 60 \* 1000

});

}

};

const invalidateProfileCache = (userId: string) => {

queryClient.invalidateQueries({ queryKey: ['profile', userId] });

queryClient.invalidateQueries({ queryKey: ['subscription', userId] });

};

const prefetchRelatedData = (userId: string) => {

// Prefetch billing history

queryClient.prefetchQuery({

queryKey: ['billing-history', userId],

queryFn: () => fetch(`/api/billing/history/${userId}`).then(res => res.json())

});

// Prefetch notification preferences

queryClient.prefetchQuery({

queryKey: ['notifications', userId],

queryFn: () => fetch(`/api/notifications/preferences/${userId}`).then(res => res.json())

});

};

return { cacheProfileData, invalidateProfileCache, prefetchRelatedData };

};

// Optimistic updates for better UX

export const useOptimisticProfileUpdates = () => {

const profileStore = useProfileStore();

const queryClient = useQueryClient();

const optimisticUpdate = async (

updateData: Partial<User>,

apiCall: () => Promise<User>

) => {

// Store current state for rollback

const previousUser = profileStore.user;

// Apply optimistic update

profileStore.setUser({ ...previousUser!, ...updateData });

try {

// Perform actual API call

const updatedUser = await apiCall();

// Update with real data

profileStore.setUser(updatedUser);

// Update cache

queryClient.setQueryData(['profile', updatedUser.id], updatedUser);

} catch (error) {

// Rollback on error

profileStore.setUser(previousUser);

// Show error message

toast.error('Failed to update profile. Please try again.');

throw error;

}

};

return { optimisticUpdate };

};

**🚀 Implementation Checklist**

**Phase 1: Core Profile Components (Week 1)**

* [ ] Create enhanced PersonalInfoCard with edit functionality
* [ ] Implement SubscriptionCard with usage analytics
* [ ] Set up Zustand store for profile state management
* [ ] Add form validation with Zod schemas
* [ ] Implement loading states and error handling

**Phase 2: Advanced Features (Week 2)**

* [ ] Add profile picture upload with compression
* [ ] Implement ChangePasswordDialog with security validation
* [ ] Create ProfileCompletionProgress component
* [ ] Add SecurityScoreIndicator with real-time scoring
* [ ] Implement subscription management features

**Phase 3: Integration & Security (Week 3)**

* [ ] Connect to PostgreSQL database with proper schema
* [ ] Set up secure API endpoints with rate limiting
* [ ] Add comprehensive input validation and sanitization
* [ ] Implement real-time updates with Supabase
* [ ] Add monitoring and error reporting

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

* [ ] Write comprehensive unit and integration tests
* [ ] Implement React Query for optimized caching
* [ ] Add accessibility features and keyboard navigation
* [ ] Performance optimization with memoization
* [ ] Deploy with proper environment configuration

**This comprehensive analysis provides everything needed for immediate implementation in Cursor. Each section includes specific code examples, TypeScript interfaces, and detailed implementation guidance for building a production-ready Profile page.**