**Sign In Page**

**🔐 Kurzora Sign In Page - Complete UI Analysis**

*Based on Actual Source Code Implementation*

**1. UI Components & Layout**

**Interactive Elements**

**Primary Form Components:**

* LoginForm (main container with conditional auth overlay)
* Email input with Mail icon and validation
* Password input with Lock icon and show/hide toggle
* Google OAuth button with loading states
* Switch to signup navigation
* Forgot password link
* Form submission with dynamic loading states

**Interactive Controls:**

* Form validation with real-time error display
* Google sign-in alternative flow with user detection
* Navigation between login/signup modes
* Back to home functionality
* Error message dismissal

**React + TypeScript Component Structure**

// Complete Sign In Architecture

<LoginForm>

<Card className="w-full max-w-md bg-slate-900/50 backdrop-blur-sm border-blue-800/30">

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

{/\* Logo Display \*/}

<div className="flex justify-center mb-4">

<img src="/kurzora-logo.svg" alt="Kurzora Logo" className="h-12 w-auto" />

</div>

<CardTitle className="text-2xl text-center text-white">Welcome back</CardTitle>

<CardDescription className="text-center text-slate-400">

Sign in to your Kurzora account

</CardDescription>

</CardHeader>

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

{/\* Error Message Container \*/}

{error && (

<ErrorDisplay error={error} onDismiss={() => setError(null)} />

)}

{/\* Main Form \*/}

<form onSubmit={handleSubmit} className="space-y-4">

{/\* Email Field \*/}

<FormField

id="email"

name="email"

type="email"

label="Email"

icon={Mail}

value={formData.email}

onChange={handleChange}

placeholder="Enter your email"

required

autoComplete="email"

aria-describedby="email-error"

/>

{/\* Password Field \*/}

<FormField

id="password"

name="password"

type="password"

label="Password"

icon={Lock}

value={formData.password}

onChange={handleChange}

placeholder="Enter your password"

required

autoComplete="current-password"

aria-describedby="password-error"

/>

{/\* Submit Button \*/}

<LoadingButton

type="submit"

loading={loading}

disabled={loading}

className="w-full bg-blue-600 hover:bg-blue-700 text-white"

loadingText="Signing in..."

>

Sign In

</LoadingButton>

</form>

{/\* Divider \*/}

<div className="relative">

<div className="absolute inset-0 flex items-center">

<div className="w-full border-t border-gray-700"></div>

</div>

<div className="relative flex justify-center text-sm">

<span className="px-4 bg-slate-900 text-gray-400">OR CONTINUE WITH</span>

</div>

</div>

{/\* Google OAuth \*/}

<GoogleSignInButton

onClick={handleGoogleSignIn}

loading={isGoogleLoading}

disabled={isGoogleLoading || loading}

/>

{/\* Navigation Links \*/}

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

<span className="text-slate-400">Don't have an account? </span>

<button

onClick={onSwitchToSignup}

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

>

Sign up

</button>

</div>

{/\* Forgot Password \*/}

<div className="text-center">

<button

onClick={handleForgotPassword}

className="text-sm text-slate-400 hover:text-slate-300"

>

Forgot your password?

</button>

</div>

</CardContent>

</Card>

</LoginForm>

// Sub-components Implementation

<FormField>

<Label htmlFor={id} className="text-slate-300">{label}</Label>

<div className="relative">

<Icon className="absolute left-3 top-3 h-4 w-4 text-slate-400" />

<Input

id={id}

name={name}

type={type}

value={value}

onChange={onChange}

className="pl-10 bg-slate-800/50 border-blue-800/30 text-white placeholder-slate-400"

{...props}

/>

{showPasswordToggle && (

<PasswordToggleButton onClick={togglePasswordVisibility} />

)}

</div>

{error && <FieldError error={error} />}

</FormField>

<GoogleSignInButton>

<button

onClick={onClick}

disabled={loading || disabled}

className="w-full flex items-center justify-center gap-3 px-4 py-3 border border-gray-700 rounded-lg hover:bg-gray-800/50 transition-colors text-white disabled:opacity-50 disabled:cursor-not-allowed"

>

{loading ? (

<Loader2 className="w-5 h-5 animate-spin" />

) : (

<GoogleIcon />

)}

<span>{loading ? 'Signing in...' : 'Continue with Google'}</span>

</button>

</GoogleSignInButton>

<ErrorDisplay>

<div className="mb-6 p-4 bg-red-500/10 border border-red-500/30 rounded-lg">

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

<p className="text-sm text-red-400">{error}</p>

{onDismiss && (

<button onClick={onDismiss} className="text-red-400 hover:text-red-300">

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

</button>

)}

</div>

</div>

</ErrorDisplay>

**Tailwind CSS Classes (Actual Implementation)**

/\* Container Styling \*/

.w-full.max-w-md /\* Form width constraint \*/

.bg-slate-900/50.backdrop-blur-sm /\* Glass morphism effect \*/

.border-blue-800/30 /\* Subtle blue border \*/

/\* Form Field Styling \*/

.bg-slate-800/50.border-blue-800/30 /\* Input background \*/

.text-white.placeholder-slate-400 /\* Text styling \*/

.pl-10 /\* Icon padding \*/

/\* Button States \*/

.bg-blue-600.hover:bg-blue-700 /\* Primary button \*/

.disabled:opacity-50.disabled:cursor-not-allowed /\* Disabled state \*/

/\* Error/Success States \*/

.bg-red-500/10.border.border-red-500/30 /\* Error container \*/

.text-red-400 /\* Error text \*/

/\* Google Button \*/

.border.border-gray-700.hover:bg-gray-800/50 /\* Google OAuth button \*/

.transition-colors /\* Smooth hover transitions \*/

/\* Loading States \*/

.animate-spin /\* Loading spinner \*/

.opacity-50 /\* Disabled state opacity \*/

**Responsive Design Implementation**

// Mobile-first approach maintained throughout

.w-full // Full width on mobile

.max-w-md // Constrained on larger screens

.space-y-4 // Consistent vertical spacing

.text-sm.sm:text-base // Responsive text sizing

.p-4.sm:p-6 // Responsive padding

.gap-3.sm:gap-4 // Responsive spacing

// Google Button Responsive

.flex.items-center.justify-center.gap-3 // Consistent spacing

.px-4.py-3 // Touch-friendly button size

.text-white // High contrast for accessibility

**Loading States & Error Handling UI**

// Dynamic Button States

const LoadingButton: React.FC<LoadingButtonProps> = ({

loading,

children,

loadingText,

...props

}) => (

<Button {...props}>

{loading ? (

<>

<Loader2 className="mr-2 h-4 w-4 animate-spin" />

{loadingText}

</>

) : (

children

)}

</Button>

)

// Error Display Component

const ErrorDisplay: React.FC<ErrorDisplayProps> = ({ error, onDismiss }) => (

<div className="mb-6 p-4 bg-red-500/10 border border-red-500/30 rounded-lg">

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

<p className="text-sm text-red-400">{error}</p>

{onDismiss && (

<button

onClick={onDismiss}

className="text-red-400 hover:text-red-300 ml-2"

aria-label="Dismiss error"

>

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

</button>

)}

</div>

</div>

)

// Form Validation Display

const FieldError: React.FC<{ error: string }> = ({ error }) => (

<p className="mt-1 text-xs text-red-400" role="alert">

{error}

</p>

)

**2. State Management (Zustand)**

**Store Structure**

interface SignInPageStore {

// Form State

formData: {

email: string

password: string

}

// Authentication State

isAuthenticating: boolean

authError: string | null

// Google OAuth State

isGoogleSignIn: boolean

googleLoading: boolean

// UI State

showPassword: boolean

rememberMe: boolean

// Session Management

lastLoginAttempt: number | null

loginAttempts: number

isRateLimited: boolean

// Actions

setFormData: (data: Partial<FormData>) => void

setAuthError: (error: string | null) => void

setGoogleLoading: (loading: boolean) => void

togglePasswordVisibility: () => void

setRememberMe: (remember: boolean) => void

incrementLoginAttempts: () => void

resetLoginAttempts: () => void

checkRateLimit: () => boolean

clearForm: () => void

}

// Current Implementation (Local State Analysis)

const [formData, setFormData] = useState({

email: '',

password: ''

})

const [error, setError] = useState<string | null>(null)

const [isGoogleLoading, setIsGoogleLoading] = useState(false)

// Enhanced Zustand Store Implementation

const useSignInStore = create<SignInPageStore>((set, get) => ({

// Initial State

formData: { email: '', password: '' },

isAuthenticating: false,

authError: null,

isGoogleSignIn: false,

googleLoading: false,

showPassword: false,

rememberMe: false,

lastLoginAttempt: null,

loginAttempts: 0,

isRateLimited: false,

// Actions

setFormData: (data) => set((state) => ({

formData: { ...state.formData, ...data }

})),

setAuthError: (error) => set({ authError: error }),

setGoogleLoading: (loading) => set({ googleLoading: loading }),

togglePasswordVisibility: () => set((state) => ({

showPassword: !state.showPassword

})),

setRememberMe: (remember) => set({ rememberMe: remember }),

incrementLoginAttempts: () => set((state) => {

const newAttempts = state.loginAttempts + 1

const now = Date.now()

return {

loginAttempts: newAttempts,

lastLoginAttempt: now,

isRateLimited: newAttempts >= 5 && (now - (state.lastLoginAttempt || 0)) < 15 \* 60 \* 1000 // 15 minutes

}

}),

resetLoginAttempts: () => set({

loginAttempts: 0,

lastLoginAttempt: null,

isRateLimited: false

}),

checkRateLimit: () => {

const state = get()

const now = Date.now()

const timeSinceLastAttempt = now - (state.lastLoginAttempt || 0)

if (state.loginAttempts >= 5 && timeSinceLastAttempt < 15 \* 60 \* 1000) {

return true // Rate limited

}

return false

},

clearForm: () => set({

formData: { email: '', password: '' },

authError: null,

showPassword: false

})

}))

**Local vs Global State Decisions**

// Local State (Component Level)

const [formData, setFormData] = useState(...) // Form inputs - keep local for performance

const [showPassword, setShowPassword] = useState(false) // UI state - local

const [validationErrors, setValidationErrors] = useState({}) // Form validation - local

// Global State (Context/Zustand)

const { user, login, loading } = useAuth() // Authentication state - global

const { lastLoginEmail } = useUserPreferences() // User preferences - global

const { isRateLimited } = useSecurityStore() // Rate limiting - global

// Persistent State (localStorage)

localStorage.setItem('lastLoginEmail', email) // Remember email

localStorage.setItem('rememberMe', 'true') // Remember me preference

sessionStorage.setItem('authAttempts', attempts) // Temporary session data

// State Update Patterns

const handleChange = useCallback((e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target

setFormData(prev => ({ ...prev, [name]: value }))

// Clear field-specific errors on change

if (validationErrors[name]) {

setValidationErrors(prev => ({ ...prev, [name]: null }))

}

}, [validationErrors])

// Error Handling Pattern

const handleAuthError = useCallback((error: Error) => {

const errorMessage = getErrorMessage(error)

setError(errorMessage)

// Track failed attempts for rate limiting

incrementLoginAttempts()

// Log security events

logSecurityEvent('failed\_login', { email: formData.email })

}, [formData.email])

**3. API Contracts & Integration**

**API Endpoints**

// Authentication APIs

POST /api/auth/login

POST /api/auth/google-signin

POST /api/auth/refresh-token

POST /api/auth/logout

POST /api/auth/forgot-password

POST /api/auth/reset-password

GET /api/auth/me

// Security APIs

POST /api/auth/verify-2fa

GET /api/auth/security-events

POST /api/auth/unlock-account

// Session Management

GET /api/auth/sessions

DELETE /api/auth/sessions/:id

POST /api/auth/verify-session

**Request/Response Schemas**

// Login Request/Response

interface LoginRequest {

email: string

password: string

rememberMe?: boolean

deviceFingerprint?: string

twoFactorCode?: string

}

interface LoginResponse {

user: {

id: string

email: string

name: string

role: 'user' | 'admin'

emailVerified: boolean

subscription?: {

tier: string

active: boolean

expiresAt: string

}

lastLoginAt: string

profileComplete: boolean

}

tokens: {

accessToken: string

refreshToken: string

expiresIn: number

}

session: {

id: string

deviceInfo: string

location?: string

expiresAt: string

}

requiresTwoFactor?: boolean

securityAlerts?: SecurityAlert[]

}

// Google Sign-in Request/Response

interface GoogleSignInRequest {

idToken: string

deviceFingerprint?: string

}

interface GoogleSignInResponse extends LoginResponse {

isNewUser: boolean

profileIncomplete?: boolean

}

// Forgot Password Request/Response

interface ForgotPasswordRequest {

email: string

captchaToken?: string

}

interface ForgotPasswordResponse {

message: string

resetToken?: string // Only in development

expiresIn: number

}

// Error Response Format

interface AuthError {

error: {

code: 'INVALID\_CREDENTIALS' | 'ACCOUNT\_LOCKED' | 'EMAIL\_NOT\_VERIFIED' | 'TWO\_FACTOR\_REQUIRED' | 'RATE\_LIMITED'

message: string

details?: {

field?: string

remainingAttempts?: number

lockoutDuration?: number

requiresVerification?: boolean

}

}

timestamp: string

requestId: string

}

// Security Event Types

interface SecurityEvent {

id: string

type: 'login\_success' | 'login\_failed' | 'password\_reset' | 'account\_locked'

timestamp: string

ipAddress: string

userAgent: string

location?: string

metadata?: Record<string, any>

}

**API Client Implementation**

// Auth API Client

class AuthApiClient {

private baseUrl: string

private timeout: number

constructor() {

this.baseUrl = import.meta.env.VITE\_API\_URL

this.timeout = 10000

}

async login(credentials: LoginRequest): Promise<LoginResponse> {

const response = await fetch(`${this.baseUrl}/api/auth/login`, {

method: 'POST',

headers: {

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

'X-Device-Fingerprint': await generateDeviceFingerprint()

},

body: JSON.stringify(credentials),

signal: AbortSignal.timeout(this.timeout)

})

if (!response.ok) {

const error = await response.json()

throw new AuthError(error)

}

return response.json()

}

async googleSignIn(googleData: GoogleSignInRequest): Promise<GoogleSignInResponse> {

const response = await fetch(`${this.baseUrl}/api/auth/google-signin`, {

method: 'POST',

headers: {

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

'X-Device-Fingerprint': await generateDeviceFingerprint()

},

body: JSON.stringify(googleData),

signal: AbortSignal.timeout(this.timeout)

})

if (!response.ok) {

const error = await response.json()

throw new AuthError(error)

}

return response.json()

}

async forgotPassword(data: ForgotPasswordRequest): Promise<ForgotPasswordResponse> {

const response = await fetch(`${this.baseUrl}/api/auth/forgot-password`, {

method: 'POST',

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

body: JSON.stringify(data),

signal: AbortSignal.timeout(this.timeout)

})

if (!response.ok) {

const error = await response.json()

throw new AuthError(error)

}

return response.json()

}

}

// Error Handling Utility

const getErrorMessage = (error: any): string => {

if (error instanceof AuthError) {

switch (error.code) {

case 'INVALID\_CREDENTIALS':

return 'Invalid email or password. Please try again.'

case 'ACCOUNT\_LOCKED':

return 'Account temporarily locked due to multiple failed attempts.'

case 'EMAIL\_NOT\_VERIFIED':

return 'Please verify your email address before signing in.'

case 'TWO\_FACTOR\_REQUIRED':

return 'Two-factor authentication required.'

case 'RATE\_LIMITED':

return 'Too many attempts. Please wait before trying again.'

default:

return error.message || 'Sign in failed. Please try again.'

}

}

return 'An unexpected error occurred. Please try again.'

}

**4. Performance & Optimization**

**Lazy Loading Implementation**

// Component Lazy Loading

const ForgotPasswordModal = lazy(() => import('./ForgotPasswordModal'))

const TwoFactorModal = lazy(() => import('./TwoFactorModal'))

const GoogleAuth = lazy(() => import('./GoogleAuth'))

// Conditional Loading for Heavy Components

{showForgotPassword && (

<Suspense fallback={<ModalSkeleton />}>

<ForgotPasswordModal

isOpen={showForgotPassword}

onClose={() => setShowForgotPassword(false)}

/>

</Suspense>

)}

// Third-party Library Lazy Loading

const loadGoogleAuth = () => import('google-auth-library')

const loadDeviceDetector = () => import('device-detector-js')

// Progressive Enhancement

const useProgressiveFeatures = () => {

const [biometricsAvailable, setBiometricsAvailable] = useState(false)

const [deviceTrustAvailable, setDeviceTrustAvailable] = useState(false)

useEffect(() => {

// Check for WebAuthn support

if (window.PublicKeyCredential) {

setBiometricsAvailable(true)

}

// Check for device trust features

if (navigator.credentials && 'create' in navigator.credentials) {

setDeviceTrustAvailable(true)

}

}, [])

return { biometricsAvailable, deviceTrustAvailable }

}

**Memoization Opportunities**

// Component Memoization

const MemoizedFormField = React.memo(FormField)

const MemoizedGoogleButton = React.memo(GoogleSignInButton)

const MemoizedErrorDisplay = React.memo(ErrorDisplay)

// Expensive Calculations

const passwordStrength = useMemo(() => {

return calculatePasswordStrength(formData.password)

}, [formData.password])

const deviceFingerprint = useMemo(() => {

return generateDeviceFingerprint()

}, [])

// Callback Memoization

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

e.preventDefault()

if (checkRateLimit()) return

try {

setError(null)

const result = await login(formData.email, formData.password)

onLoginSuccess(result)

} catch (error) {

handleAuthError(error)

}

}, [formData, login, onLoginSuccess, checkRateLimit])

const handleGoogleSignIn = useCallback(async () => {

if (checkRateLimit()) return

try {

setError(null)

setIsGoogleLoading(true)

const googleAuth = await loadGoogleAuth()

const result = await googleAuth.signIn()

await handleGoogleAuthResult(result)

} catch (error) {

handleAuthError(error)

} finally {

setIsGoogleLoading(false)

}

}, [checkRateLimit, handleGoogleAuthResult])

// Debounced Email Validation

const debouncedEmailValidation = useDebouncedCallback(

async (email: string) => {

if (email && isValidEmail(email)) {

const isAvailable = await checkEmailAvailability(email)

setEmailValidation({ isValid: true, isAvailable })

}

},

500

)

**Bundle Splitting**

// Route-based splitting

const SignInPage = lazy(() => import('./pages/SignInPage'))

const Dashboard = lazy(() => import('./pages/Dashboard'))

// Feature-based splitting

const BiometricAuth = lazy(() => import('./components/auth/BiometricAuth'))

const SocialAuth = lazy(() => import('./components/auth/SocialAuth'))

// Third-party library splitting

const googleAuth = () => import('@google-auth/google-auth-library')

const deviceDetector = () => import('device-detector-js')

// Critical CSS inlining for above-the-fold content

const criticalStyles = `

.auth-container { /\* Critical styles \*/ }

.form-field { /\* Critical form styles \*/ }

`

**5. Database Schema**

**PostgreSQL Tables**

-- Users table (enhanced for authentication)

CREATE TABLE users (

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

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255), -- Nullable for OAuth-only users

name VARCHAR(255) NOT NULL,

email\_verified BOOLEAN DEFAULT false,

phone\_number VARCHAR(20),

phone\_verified BOOLEAN DEFAULT false,

-- OAuth Integration

google\_id VARCHAR(255) UNIQUE,

github\_id VARCHAR(255) UNIQUE,

oauth\_providers JSONB DEFAULT '[]',

-- Security Settings

two\_factor\_enabled BOOLEAN DEFAULT false,

two\_factor\_secret VARCHAR(255),

backup\_codes JSONB,

password\_reset\_token VARCHAR(255),

password\_reset\_expires TIMESTAMP WITH TIME ZONE,

email\_verification\_token VARCHAR(255),

email\_verification\_expires TIMESTAMP WITH TIME ZONE,

-- Account Status

role VARCHAR(50) DEFAULT 'user' CHECK (role IN ('user', 'admin', 'moderator')),

account\_status VARCHAR(50) DEFAULT 'active' CHECK (account\_status IN ('active', 'suspended', 'locked', 'pending')),

account\_locked\_until TIMESTAMP WITH TIME ZONE,

failed\_login\_attempts INTEGER DEFAULT 0,

last\_failed\_login TIMESTAMP WITH TIME ZONE,

-- Profile Information

avatar\_url TEXT,

timezone VARCHAR(100) DEFAULT 'UTC',

language VARCHAR(10) DEFAULT 'en',

profile\_complete BOOLEAN DEFAULT false,

-- Timestamps

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

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

last\_login\_at TIMESTAMP WITH TIME ZONE,

password\_changed\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- User Sessions

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) UNIQUE NOT NULL,

refresh\_token VARCHAR(255) UNIQUE,

device\_fingerprint VARCHAR(255),

-- Session Metadata

ip\_address INET,

user\_agent TEXT,

device\_type VARCHAR(50), -- mobile, desktop, tablet

browser VARCHAR(100),

os VARCHAR(100),

location JSONB, -- {country, city, region}

-- Security Flags

is\_trusted\_device BOOLEAN DEFAULT false,

requires\_verification BOOLEAN DEFAULT false,

-- Timestamps

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

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

expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,

revoked\_at TIMESTAMP WITH TIME ZONE

);

-- Security Events Log

CREATE TABLE security\_events (

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

user\_id UUID REFERENCES users(id) ON DELETE SET NULL,

session\_id UUID REFERENCES user\_sessions(id) ON DELETE SET NULL,

-- Event Details

event\_type VARCHAR(100) NOT NULL,

event\_description TEXT,

severity VARCHAR(20) DEFAULT 'info' CHECK (severity IN ('low', 'medium', 'high', 'critical')),

-- Context

ip\_address INET,

user\_agent TEXT,

location JSONB,

metadata JSONB,

-- Investigation

resolved BOOLEAN DEFAULT false,

notes TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Rate Limiting

CREATE TABLE rate\_limits (

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

identifier VARCHAR(255) NOT NULL, -- IP, user ID, email, etc.

action VARCHAR(100) NOT NULL, -- login, signup, password\_reset

attempts INTEGER DEFAULT 1,

first\_attempt TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

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

blocked\_until TIMESTAMP WITH TIME ZONE,

UNIQUE(identifier, action)

);

-- Trusted Devices

CREATE TABLE trusted\_devices (

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

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

device\_fingerprint VARCHAR(255) NOT NULL,

device\_name VARCHAR(255),

-- Trust Status

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

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

trust\_expires TIMESTAMP WITH TIME ZONE,

revoked\_at TIMESTAMP WITH TIME ZONE,

-- Device Info

device\_type VARCHAR(50),

browser VARCHAR(100),

os VARCHAR(100),

UNIQUE(user\_id, device\_fingerprint)

);

-- Email Templates (for auth emails)

CREATE TABLE email\_templates (

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

template\_key VARCHAR(100) UNIQUE NOT NULL,

subject VARCHAR(255) NOT NULL,

html\_content TEXT NOT NULL,

text\_content TEXT NOT NULL,

variables JSONB, -- Template variables

active BOOLEAN DEFAULT true,

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Indexes for Performance

CREATE INDEX idx\_users\_email ON users(email);

CREATE INDEX idx\_users\_google\_id ON users(google\_id);

CREATE INDEX idx\_users\_account\_status ON users(account\_status);

CREATE INDEX idx\_users\_last\_login ON users(last\_login\_at);

CREATE INDEX idx\_user\_sessions\_user\_id ON user\_sessions(user\_id);

CREATE INDEX idx\_user\_sessions\_token ON user\_sessions(session\_token);

CREATE INDEX idx\_user\_sessions\_expires ON user\_sessions(expires\_at);

CREATE INDEX idx\_user\_sessions\_device ON user\_sessions(device\_fingerprint);

CREATE INDEX idx\_security\_events\_user\_id ON security\_events(user\_id);

CREATE INDEX idx\_security\_events\_type ON security\_events(event\_type);

CREATE INDEX idx\_security\_events\_created ON security\_events(created\_at);

CREATE INDEX idx\_security\_events\_severity ON security\_events(severity);

CREATE INDEX idx\_rate\_limits\_identifier\_action ON rate\_limits(identifier, action);

CREATE INDEX idx\_rate\_limits\_blocked\_until ON rate\_limits(blocked\_until);

CREATE INDEX idx\_trusted\_devices\_user\_id ON trusted\_devices(user\_id);

CREATE INDEX idx\_trusted\_devices\_fingerprint ON trusted\_devices(device\_fingerprint);

**6. User Experience**

**Loading States Implementation**

// Sign In Form Skeleton

const SignInFormSkeleton = () => (

<Card className="w-full max-w-md bg-slate-900/50 backdrop-blur-sm border-blue-800/30">

<CardHeader>

<div className="animate-pulse">

<div className="h-12 w-32 bg-slate-700 rounded mx-auto mb-4"></div>

<div className="h-6 w-48 bg-slate-700 rounded mx-auto mb-2"></div>

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

</div>

</CardHeader>

<CardContent>

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

<div className="h-10 bg-slate-700 rounded"></div>

<div className="h-10 bg-slate-700 rounded"></div>

<div className="h-12 bg-slate-700 rounded"></div>

<div className="h-px bg-slate-700"></div>

<div className="h-12 bg-slate-700 rounded"></div>

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

<div className="h-4 bg-slate-700 rounded w-1/2 mx-auto"></div>

</div>

</CardContent>

</Card>

)

// Loading Button Implementation

const LoadingButton: React.FC<LoadingButtonProps> = ({

loading,

children,

disabled,

...props

}) => (

<Button

disabled={disabled || loading}

className={cn(

"w-full transition-all duration-200",

loading && "cursor-not-allowed opacity-80"

)}

{...props}

>

{loading ? (

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

<Loader2 className="mr-2 h-4 w-4 animate-spin" />

<span>Signing in...</span>

</div>

) : (

children

)}

</Button>

)

// Progressive Loading for Heavy Features

const useProgressiveLoading = () => {

const [loadingStates, setLoadingStates] = useState({

googleAuth: false,

biometrics: false,

deviceTrust: false

})

const loadFeature = useCallback(async (feature: string) => {

setLoadingStates(prev => ({ ...prev, [feature]: true }))

try {

switch (feature) {

case 'googleAuth':

await import('./GoogleAuth')

break

case 'biometrics':

await import('./BiometricAuth')

break

case 'deviceTrust':

await import('./DeviceTrust')

break

}

} finally {

setLoadingStates(prev => ({ ...prev, [feature]: false }))

}

}, [])

return { loadingStates, loadFeature }

}

**Error Boundaries**

class SignInErrorBoundary extends Component<ErrorBoundaryProps, ErrorBoundaryState> {

constructor(props: ErrorBoundaryProps) {

super(props)

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

}

static getDerivedStateFromError(error: Error): Partial<ErrorBoundaryState> {

return { hasError: true, error }

}

componentDidCatch(error: Error, errorInfo: ErrorInfo) {

console.error('Sign in error boundary caught an error:', error, errorInfo)

// Report to error tracking service

if (import.meta.env.PROD) {

reportError(error, {

component: 'SignInForm',

errorInfo,

userId: this.props.userId

})

}

this.setState({ errorInfo })

}

render() {

if (this.state.hasError) {

return (

<Card className="w-full max-w-md bg-slate-900/50 backdrop-blur-sm border-red-800/30">

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

<div className="mb-4">

<AlertTriangle className="h-12 w-12 text-red-400 mx-auto mb-2" />

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

Something went wrong

</h3>

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

We're having trouble loading the sign in form. This might be a temporary issue.

</p>

</div>

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

<Button

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

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

>

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

Refresh Page

</Button>

<Button

variant="outline"

onClick={() => this.setState({ hasError: false })}

className="w-full border-slate-600 text-slate-300"

>

Try Again

</Button>

</div>

{import.meta.env.DEV && this.state.error && (

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

<summary className="text-xs text-slate-500 cursor-pointer">

Error Details (Dev Mode)

</summary>

<pre className="text-xs text-red-400 mt-2 overflow-auto max-h-32">

{this.state.error.stack}

</pre>

</details>

)}

</CardContent>

</Card>

)

}

return this.props.children

}

}

**Accessibility Implementation**

// ARIA Labels and Semantic HTML

const AccessibleSignInForm = () => (

<form onSubmit={handleSubmit} role="form" aria-labelledby="signin-title">

<h2 id="signin-title" className="sr-only">Sign In Form</h2>

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

<div>

<Label htmlFor="email" className="text-slate-300">

Email Address

</Label>

<div className="relative">

<Mail className="absolute left-3 top-3 h-4 w-4 text-slate-400" aria-hidden="true" />

<Input

id="email"

name="email"

type="email"

autoComplete="email"

aria-required="true"

aria-describedby="email-error"

aria-invalid={!!emailError}

value={formData.email}

onChange={handleChange}

className="pl-10 bg-slate-800/50 border-blue-800/30 text-white placeholder-slate-400"

placeholder="Enter your email"

/>

</div>

{emailError && (

<div id="email-error" role="alert" className="mt-1 text-xs text-red-400">

{emailError}

</div>

)}

</div>

<div>

<Label htmlFor="password" className="text-slate-300">

Password

</Label>

<div className="relative">

<Lock className="absolute left-3 top-3 h-4 w-4 text-slate-400" aria-hidden="true" />

<Input

id="password"

name="password"

type={showPassword ? 'text' : 'password'}

autoComplete="current-password"

aria-required="true"

aria-describedby="password-error"

aria-invalid={!!passwordError}

value={formData.password}

onChange={handleChange}

className="pl-10 pr-10 bg-slate-800/50 border-blue-800/30 text-white placeholder-slate-400"

placeholder="Enter your password"

/>

<button

type="button"

onClick={() => setShowPassword(!showPassword)}

className="absolute right-3 top-3 text-slate-400 hover:text-white"

aria-label={showPassword ? 'Hide password' : 'Show password'}

>

{showPassword ? <EyeOff className="h-4 w-4" /> : <Eye className="h-4 w-4" />}

</button>

</div>

{passwordError && (

<div id="password-error" role="alert" className="mt-1 text-xs text-red-400">

{passwordError}

</div>

)}

</div>

</div>

<Button

type="submit"

disabled={loading || !isFormValid}

aria-describedby="submit-status"

className="w-full mt-6"

>

{loading ? (

<>

<span className="sr-only">Signing in, please wait</span>

<Loader2 className="mr-2 h-4 w-4 animate-spin" aria-hidden="true" />

<span>Signing in...</span>

</>

) : (

'Sign In'

)}

</Button>

</form>

)

// Keyboard Navigation Support

const useKeyboardNavigation = () => {

const formRef = useRef<HTMLFormElement>(null)

useEffect(() => {

const handleKeyDown = (e: KeyboardEvent) => {

if (e.key === 'Enter' && e.target?.tagName !== 'BUTTON') {

e.preventDefault()

// Find next focusable element or submit form

const form = formRef.current

if (form) {

const focusableElements = form.querySelectorAll(

'input, button, select, textarea, [tabindex]:not([tabindex="-1"])'

)

const currentIndex = Array.from(focusableElements).indexOf(e.target as Element)

const nextElement = focusableElements[currentIndex + 1] as HTMLElement

if (nextElement) {

nextElement.focus()

} else {

form.requestSubmit()

}

}

}

}

document.addEventListener('keydown', handleKeyDown)

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

}, [])

return formRef

}

// Focus Management

const useFocusManagement = () => {

const firstInputRef = useRef<HTMLInputElement>(null)

const errorRef = useRef<HTMLDivElement>(null)

// Focus first input on mount

useEffect(() => {

firstInputRef.current?.focus()

}, [])

// Focus error message when error appears

useEffect(() => {

if (error && errorRef.current) {

errorRef.current.focus()

}

}, [error])

return { firstInputRef, errorRef }

}

**Animation Requirements**

// Form Animations

const formVariants = {

hidden: { opacity: 0, y: 20 },

visible: {

opacity: 1,

y: 0,

transition: { duration: 0.3, ease: 'easeOut' }

},

exit: {

opacity: 0,

y: -20,

transition: { duration: 0.2 }

}

}

// Error Animation

const errorVariants = {

hidden: { opacity: 0, scale: 0.95 },

visible: {

opacity: 1,

scale: 1,

transition: { duration: 0.2 }

},

exit: {

opacity: 0,

scale: 0.95,

transition: { duration: 0.15 }

}

}

// Button State Animations

const buttonVariants = {

idle: { scale: 1 },

loading: { scale: 0.98 },

success: {

scale: 1.02,

transition: { duration: 0.1 }

}

}

// CSS Animations

const animationClasses = {

'form-slide-in': 'animate-in slide-in-from-bottom-4 duration-300',

'error-bounce': 'animate-in zoom-in-50 duration-200',

'button-pulse': 'hover:scale-105 transition-transform duration-200',

'loading-spin': 'animate-spin',

'success-bounce': 'animate-bounce'

}

**7. Integration Points**

**Navigation & Routing**

// Route Configuration and Guards

const SignInRoute = () => {

const { user, loading } = useAuth()

const location = useLocation()

const navigate = useNavigate()

// Get redirect destination from URL params or state

const from = location.state?.from?.pathname || '/dashboard'

const redirectTo = new URLSearchParams(location.search).get('redirect') || from

// Redirect authenticated users

useEffect(() => {

if (user && !loading) {

navigate(redirectTo, { replace: true })

}

}, [user, loading, navigate, redirectTo])

if (loading) {

return <SignInFormSkeleton />

}

if (user) {

return null // Will redirect

}

return <SignInForm redirectTo={redirectTo} />

}

// Deep Link Handling

const useDeepLinkHandling = () => {

const location = useLocation()

const [initialState, setInitialState] = useState({})

useEffect(() => {

const params = new URLSearchParams(location.search)

const email = params.get('email')

const resetToken = params.get('reset\_token')

const action = params.get('action')

if (email) {

setInitialState(prev => ({ ...prev, email }))

}

if (resetToken && action === 'reset\_password') {

// Handle password reset flow

setInitialState(prev => ({ ...prev, resetToken, showResetForm: true }))

}

}, [location.search])

return initialState

}

// Post-Authentication Navigation

const usePostAuthNavigation = () => {

const navigate = useNavigate()

const location = useLocation()

const handleLoginSuccess = useCallback((user: User, redirectTo?: string) => {

// Clear any error states

clearAuthErrors()

// Determine where to navigate

const destination = redirectTo ||

location.state?.from?.pathname ||

(user.profileComplete ? '/dashboard' : '/onboarding')

// Navigate with replace to prevent back button issues

navigate(destination, {

replace: true,

state: { welcomeBack: true }

})

// Track login success

trackEvent('login\_success', {

method: 'email',

redirectTo: destination

})

}, [navigate, location.state])

return { handleLoginSuccess }

}

**Cross-Component Communication**

// Auth State Broadcasting

const useAuthBroadcast = () => {

const { user } = useAuth()

useEffect(() => {

// Broadcast auth state changes to other tabs

const handleStorageChange = (e: StorageEvent) => {

if (e.key === 'auth\_user' && e.newValue !== e.oldValue) {

window.location.reload() // Sync auth state across tabs

}

}

window.addEventListener('storage', handleStorageChange)

return () => window.removeEventListener('storage', handleStorageChange)

}, [])

useEffect(() => {

// Update localStorage when user state changes

if (user) {

localStorage.setItem('auth\_user', JSON.stringify(user))

} else {

localStorage.removeItem('auth\_user')

}

}, [user])

}

// Form State Persistence

const useFormPersistence = () => {

const [formData, setFormData] = useState({ email: '', password: '' })

// Load saved email on mount

useEffect(() => {

const savedEmail = localStorage.getItem('lastLoginEmail')

if (savedEmail) {

setFormData(prev => ({ ...prev, email: savedEmail }))

}

}, [])

// Save email on successful login

const saveLoginEmail = useCallback((email: string) => {

localStorage.setItem('lastLoginEmail', email)

}, [])

return { formData, setFormData, saveLoginEmail }

}

// Modal State Management

const useModalState = () => {

const [modals, setModals] = useState({

forgotPassword: false,

twoFactor: false,

accountLocked: false

})

const openModal = useCallback((modalName: string) => {

setModals(prev => ({ ...prev, [modalName]: true }))

}, [])

const closeModal = useCallback((modalName: string) => {

setModals(prev => ({ ...prev, [modalName]: false }))

}, [])

const closeAllModals = useCallback(() => {

setModals({

forgotPassword: false,

twoFactor: false,

accountLocked: false

})

}, [])

return { modals, openModal, closeModal, closeAllModals }

}

**Event Handling & User Flows**

// Authentication Flow Management

const useAuthFlow = () => {

const { login } = useAuth()

const [flowState, setFlowState] = useState({

step: 'credentials', // credentials, twoFactor, success

requiresTwoFactor: false,

userId: null

})

const handleCredentialsSubmit = useCallback(async (credentials: LoginCredentials) => {

try {

const result = await login(credentials)

if (result.requiresTwoFactor) {

setFlowState({

step: 'twoFactor',

requiresTwoFactor: true,

userId: result.userId

})

} else {

setFlowState({ step: 'success', requiresTwoFactor: false, userId: result.user.id })

handleLoginSuccess(result.user)

}

} catch (error) {

handleAuthError(error)

}

}, [login, handleLoginSuccess, handleAuthError])

const handleTwoFactorSubmit = useCallback(async (code: string) => {

try {

const result = await verifyTwoFactor(flowState.userId, code)

setFlowState({ step: 'success', requiresTwoFactor: false, userId: result.user.id })

handleLoginSuccess(result.user)

} catch (error) {

handleAuthError(error)

}

}, [flowState.userId, handleLoginSuccess, handleAuthError])

return {

flowState,

handleCredentialsSubmit,

handleTwoFactorSubmit

}

}

// Social Auth Integration

const useSocialAuth = () => {

const { googleSignIn } = useAuth()

const handleGoogleSignIn = useCallback(async () => {

try {

setIsGoogleLoading(true)

setError(null)

// Initialize Google Auth

const google = await window.google?.accounts?.id?.initialize({

client\_id: import.meta.env.VITE\_GOOGLE\_CLIENT\_ID,

callback: handleGoogleCallback

})

if (!google) {

throw new Error('Google Sign-In not available')

}

// Prompt for sign-in

google.prompt()

} catch (error) {

console.error('Google sign-in error:', error)

setError('Google sign-in is temporarily unavailable. Please try again.')

} finally {

setIsGoogleLoading(false)

}

}, [googleSignIn])

const handleGoogleCallback = useCallback(async (response: any) => {

try {

const result = await googleSignIn(response.credential)

if (result.isNewUser) {

// Redirect to onboarding for new users

navigate('/onboarding', { state: { fromGoogle: true } })

} else {

handleLoginSuccess(result.user)

}

} catch (error) {

handleAuthError(error)

}

}, [googleSignIn, navigate, handleLoginSuccess, handleAuthError])

return { handleGoogleSignIn }

}

**8. Testing Strategy**

**Unit Tests**

describe('LoginForm', () => {

const mockLogin = jest.fn()

const mockOnSwitchToSignup = jest.fn()

beforeEach(() => {

jest.clearAllMocks()

})

it('renders with correct elements', () => {

render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

expect(screen.getByText('Welcome back')).toBeInTheDocument()

expect(screen.getByLabelText('Email')).toBeInTheDocument()

expect(screen.getByLabelText('Password')).toBeInTheDocument()

expect(screen.getByText('Sign In')).toBeInTheDocument()

expect(screen.getByText('Continue with Google')).toBeInTheDocument()

})

it('validates email format', async () => {

const { user } = render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

const emailInput = screen.getByLabelText('Email')

await user.type(emailInput, 'invalid-email')

await user.click(screen.getByText('Sign In'))

await waitFor(() => {

expect(screen.getByText(/please enter a valid email/i)).toBeInTheDocument()

})

})

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

const { user } = render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

await user.click(screen.getByText('Sign In'))

await waitFor(() => {

expect(screen.getByText(/email is required/i)).toBeInTheDocument()

expect(screen.getByText(/password is required/i)).toBeInTheDocument()

})

})

it('handles successful login', async () => {

const mockUser = {

id: '1',

email: 'test@example.com',

name: 'Test User'

}

mockLogin.mockResolvedValue({ user: mockUser })

const { user } = render(

<AuthProvider value={{ login: mockLogin, loading: false }}>

<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />

</AuthProvider>

)

await user.type(screen.getByLabelText('Email'), 'test@example.com')

await user.type(screen.getByLabelText('Password'), 'password123')

await user.click(screen.getByText('Sign In'))

await waitFor(() => {

expect(mockLogin).toHaveBeenCalledWith('test@example.com', 'password123')

})

})

it('displays error messages', async () => {

const errorMessage = 'Invalid credentials'

mockLogin.mockRejectedValue(new Error(errorMessage))

const { user } = render(

<AuthProvider value={{ login: mockLogin, loading: false }}>

<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />

</AuthProvider>

)

await user.type(screen.getByLabelText('Email'), 'test@example.com')

await user.type(screen.getByLabelText('Password'), 'wrongpassword')

await user.click(screen.getByText('Sign In'))

await waitFor(() => {

expect(screen.getByText(errorMessage)).toBeInTheDocument()

})

})

it('shows loading state during submission', async () => {

mockLogin.mockImplementation(() => new Promise(resolve => setTimeout(resolve, 100)))

const { user } = render(

<AuthProvider value={{ login: mockLogin, loading: false }}>

<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />

</AuthProvider>

)

await user.type(screen.getByLabelText('Email'), 'test@example.com')

await user.type(screen.getByLabelText('Password'), 'password123')

await user.click(screen.getByText('Sign In'))

expect(screen.getByText('Signing in...')).toBeInTheDocument()

expect(screen.getByRole('button', { name: /signing in/i })).toBeDisabled()

})

it('toggles password visibility', async () => {

const { user } = render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

const passwordInput = screen.getByLabelText('Password')

const toggleButton = screen.getByLabelText('Show password')

expect(passwordInput).toHaveAttribute('type', 'password')

await user.click(toggleButton)

expect(passwordInput).toHaveAttribute('type', 'text')

expect(screen.getByLabelText('Hide password')).toBeInTheDocument()

await user.click(screen.getByLabelText('Hide password'))

expect(passwordInput).toHaveAttribute('type', 'password')

})

it('handles Google sign-in', async () => {

const mockGoogleSignIn = jest.fn()

global.google = {

accounts: {

id: {

initialize: jest.fn(),

prompt: jest.fn()

}

}

}

const { user } = render(

<AuthProvider value={{ googleSignIn: mockGoogleSignIn, loading: false }}>

<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />

</AuthProvider>

)

await user.click(screen.getByText('Continue with Google'))

expect(global.google.accounts.id.initialize).toHaveBeenCalled()

})

it('switches to signup form', async () => {

const { user } = render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

await user.click(screen.getByText('Sign up'))

expect(mockOnSwitchToSignup).toHaveBeenCalled()

})

it('handles forgot password click', async () => {

const { user } = render(<LoginForm onSwitchToSignup={mockOnSwitchToSignup} />)

await user.click(screen.getByText('Forgot your password?'))

// Should open forgot password modal or navigate

expect(screen.getByText(/forgot password/i)).toBeInTheDocument()

})

})

describe('Form Validation', () => {

it('validates email format in real-time', async () => {

const { user } = render(<LoginForm onSwitchToSignup={jest.fn()} />)

const emailInput = screen.getByLabelText('Email')

await user.type(emailInput, 'invalid')

await user.tab() // Trigger blur event

await waitFor(() => {

expect(screen.getByText(/please enter a valid email/i)).toBeInTheDocument()

})

await user.clear(emailInput)

await user.type(emailInput, 'valid@example.com')

await waitFor(() => {

expect(screen.queryByText(/please enter a valid email/i)).not.toBeInTheDocument()

})

})

it('validates password requirements', async () => {

const { user } = render(<LoginForm onSwitchToSignup={jest.fn()} />)

const passwordInput = screen.getByLabelText('Password')

await user.type(passwordInput, '123')

await user.tab()

await waitFor(() => {

expect(screen.getByText(/password must be at least 8 characters/i)).toBeInTheDocument()

})

})

})

**Integration Tests**

describe('Sign In Flow Integration', () => {

beforeEach(() => {

// Mock API responses

mockApiResponse('/api/auth/login', {

user: mockUser,

tokens: mockTokens,

session: mockSession

})

})

it('completes full sign in flow', async () => {

const { user } = render(

<MemoryRouter initialEntries={['/signin']}>

<AuthProvider>

<App />

</AuthProvider>

</MemoryRouter>

)

// Fill form

await user.type(screen.getByLabelText('Email'), 'test@example.com')

await user.type(screen.getByLabelText('Password'), 'password123')

// Submit form

await user.click(screen.getByText('Sign In'))

// Verify API call

await waitFor(() => {

expect(mockApiCall).toHaveBeenCalledWith('/api/auth/login', {

method: 'POST',

body: JSON.stringify({

email: 'test@example.com',

password: 'password123'

})

})

})

// Verify navigation to dashboard

await waitFor(() => {

expect(screen.getByText('Dashboard')).toBeInTheDocument()

})

})

it('handles two-factor authentication flow', async () => {

mockApiResponse('/api/auth/login', {

requiresTwoFactor: true,

userId: 'user123'

})

const { user } = render(<SignInFlow />)

await user.type(screen.getByLabelText('Email'), 'test@example.com')

await user.type(screen.getByLabelText('Password'), 'password123')

await user.click(screen.getByText('Sign In'))

// Should show 2FA form

await waitFor(() => {

expect(screen.getByText('Enter verification code')).toBeInTheDocument()

})

// Enter 2FA code

await user.type(screen.getByLabelText('Verification Code'), '123456')

await user.click(screen.getByText('Verify'))

// Verify 2FA API call

await waitFor(() => {

expect(mockApiCall).toHaveBeenCalledWith('/api/auth/verify-2fa', {

method: 'POST',

body: JSON.stringify({

userId: 'user123',

code: '123456'

})

})

})

})

it('handles Google OAuth flow', async () => {

const mockGoogleResponse = {

credential: 'mock-jwt-token'

}

mockApiResponse('/api/auth/google-signin', {

user: mockUser,

tokens: mockTokens,

isNewUser: false

})

global.google = {

accounts: {

id: {

initialize: jest.fn((config) => {

// Simulate successful Google sign-in

setTimeout(() => config.callback(mockGoogleResponse), 100)

}),

prompt: jest.fn()

}

}

}

const { user } = render(<SignInFlow />)

await user.click(screen.getByText('Continue with Google'))

await waitFor(() => {

expect(mockApiCall).toHaveBeenCalledWith('/api/auth/google-signin', {

method: 'POST',

body: JSON.stringify({

idToken: 'mock-jwt-token'

})

})

})

})

it('persists form data across navigation', async () => {

const { user, rerender } = render(<SignInForm />)

await user.type(screen.getByLabelText('Email'), 'test@example.com')

// Simulate navigation away and back

rerender(<div>Other page</div>)

rerender(<SignInForm />)

// Email should be preserved

expect(screen.getByLabelText('Email')).toHaveValue('test@example.com')

})

})

**Mock Data Structures**

const mockUser = {

id: 'user-123',

email: 'test@example.com',

name: 'Test User',

role: 'user' as const,

emailVerified: true,

subscription: {

tier: 'Professional',

active: true,

expiresAt: '2025-06-01T00:00:00Z'

},

lastLoginAt: '2025-01-15T10:30:00Z',

profileComplete: true

}

const mockTokens = {

accessToken: 'mock-access-token',

refreshToken: 'mock-refresh-token',

expiresIn: 3600

}

const mockSession = {

id: 'session-123',

deviceInfo: 'Chrome on macOS',

location: 'New York, NY',

expiresAt: '2025-01-16T10:30:00Z'

}

const mockAuthError = {

error: {

code: 'INVALID\_CREDENTIALS',

message: 'Invalid email or password',

details: {

remainingAttempts: 2

}

},

timestamp: '2025-01-15T10:30:00Z',

requestId: 'req-123'

}

const mockSecurityEvent = {

id: 'event-123',

type: 'login\_failed',

timestamp: '2025-01-15T10:30:00Z',

ipAddress: '192.168.1.1',

userAgent: 'Mozilla/5.0...',

location: 'New York, NY',

metadata: {

reason: 'invalid\_password'

}

}

**9. Charts & Data Visualizations**

**No Charts Required for Sign In Page**

The sign in page is primarily focused on form interactions and authentication flows. However, there are some visual data elements that enhance the user experience:

// Login Attempt Visualization (for security dashboard)

const LoginAttemptsChart = () => {

const attempts = useSecurityEvents('login\_attempts', '24h')

return (

<div className="bg-slate-800 rounded-lg p-4">

<h3 className="text-sm font-medium text-slate-300 mb-2">

Login Attempts (Last 24h)

</h3>

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

{attempts.map((attempt, index) => (

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

<span className="text-slate-400">{attempt.time}</span>

<span className={`px-2 py-1 rounded-full ${

attempt.success ? 'bg-green-500/20 text-green-400' : 'bg-red-500/20 text-red-400'

}`}>

{attempt.success ? 'Success' : 'Failed'}

</span>

</div>

))}

</div>

</div>

)

}

// Password Strength Visualization

const PasswordStrengthMeter = ({ password }: { password: string }) => {

const strength = calculatePasswordStrength(password)

const strengthLabels = ['Very Weak', 'Weak', 'Fair', 'Good', 'Strong']

const strengthColors = [

'bg-red-500',

'bg-orange-500',

'bg-yellow-500',

'bg-blue-500',

'bg-green-500'

]

return (

<div className="mt-2">

<div className="flex space-x-1 mb-1">

{[1, 2, 3, 4, 5].map(level => (

<div

key={level}

className={`h-1 flex-1 rounded ${

level <= strength

? strengthColors[strength - 1]

: 'bg-slate-600'

} transition-colors duration-200`}

/>

))}

</div>

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

Strength: {strengthLabels[strength - 1]}

</p>

</div>

)

}

**10. Visual Data Elements**

**Progress Indicators**

// Login Progress Indicator

const LoginProgress = ({ step }: { step: 'credentials' | 'twoFactor' | 'success' }) => {

const steps = [

{ id: 'credentials', label: 'Sign In' },

{ id: 'twoFactor', label: '2FA' },

{ id: 'success', label: 'Complete' }

]

const currentIndex = steps.findIndex(s => s.id === step)

return (

<div className="flex items-center justify-center space-x-2 mb-6">

{steps.map((stepItem, index) => (

<React.Fragment key={stepItem.id}>

<div className={`w-8 h-8 rounded-full flex items-center justify-center text-xs font-medium ${

index <= currentIndex

? 'bg-blue-600 text-white'

: 'bg-slate-600 text-slate-400'

}`}>

{index < currentIndex ? (

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

) : (

index + 1

)}

</div>

{index < steps.length - 1 && (

<div className={`w-12 h-px ${

index < currentIndex ? 'bg-blue-600' : 'bg-slate-600'

}`} />

)}

</React.Fragment>

))}

</div>

)

}

// Loading States with Progress

const AuthLoadingState = ({ message, progress }: { message: string, progress?: number }) => (

<div className="flex flex-col items-center justify-center py-8">

<div className="relative mb-4">

<div className="w-16 h-16 border-4 border-slate-600 border-t-blue-600 rounded-full animate-spin" />

{progress && (

<div className="absolute inset-0 flex items-center justify-center">

<span className="text-xs font-medium text-white">{progress}%</span>

</div>

)}

</div>

<p className="text-slate-300 text-sm">{message}</p>

</div>

)

// Rate Limit Countdown

const RateLimitCountdown = ({ remainingTime }: { remainingTime: number }) => {

const [timeLeft, setTimeLeft] = useState(remainingTime)

useEffect(() => {

const timer = setInterval(() => {

setTimeLeft(prev => Math.max(0, prev - 1))

}, 1000)

return () => clearInterval(timer)

}, [])

const minutes = Math.floor(timeLeft / 60)

const seconds = timeLeft % 60

return (

<div className="text-center p-4 bg-yellow-500/10 border border-yellow-500/30 rounded-lg">

<div className="text-yellow-400 font-medium mb-2">

Too many failed attempts

</div>

<div className="text-slate-300 text-sm mb-3">

Please wait before trying again

</div>

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

{minutes:02d}:{seconds:02d}

</div>

<div className="w-full bg-slate-600 rounded-full h-1 mt-3">

<div

className="bg-yellow-500 h-1 rounded-full transition-all duration-1000"

style={{ width: `${((remainingTime - timeLeft) / remainingTime) \* 100}%` }}

/>

</div>

</div>

)

}

**Dynamic Counters and Animated Numbers**

// Security Metrics Display

const SecurityMetrics = () => {

const [metrics, setMetrics] = useState({

totalUsers: 0,

activeToday: 0,

successRate: 0

})

useEffect(() => {

// Animate numbers on mount

const animateNumber = (target: number, setter: (value: number) => void) => {

let current = 0

const increment = target / 50

const timer = setInterval(() => {

current += increment

if (current >= target) {

setter(target)

clearInterval(timer)

} else {

setter(Math.floor(current))

}

}, 20)

}

animateNumber(2847, (value) => setMetrics(prev => ({ ...prev, totalUsers: value })))

animateNumber(324, (value) => setMetrics(prev => ({ ...prev, activeToday: value })))

animateNumber(98.7, (value) => setMetrics(prev => ({ ...prev, successRate: value })))

}, [])

return (

<div className="grid grid-cols-3 gap-4 text-center">

<div className="bg-slate-800/50 rounded-lg p-4">

<div className="text-2xl font-bold text-white">{metrics.totalUsers.toLocaleString()}</div>

<div className="text-xs text-slate-400">Total Users</div>

</div>

<div className="bg-slate-800/50 rounded-lg p-4">

<div className="text-2xl font-bold text-green-400">{metrics.activeToday}</div>

<div className="text-xs text-slate-400">Active Today</div>

</div>

<div className="bg-slate-800/50 rounded-lg p-4">

<div className="text-2xl font-bold text-blue-400">{metrics.successRate}%</div>

<div className="text-xs text-slate-400">Success Rate</div>

</div>

</div>

)

}

**Icon Systems and Visual Hierarchy**

// Icon System for Auth States

const AuthIcons = {

email: Mail,

password: Lock,

google: () => (

<svg className="w-5 h-5" viewBox="0 0 24 24">

<path fill="#4285F4" d="M22.56 12.25c0-.78-.07-1.53-.2-2.25H12v4.26h5.92c-.26 1.37-1.04 2.53-2.21 3.31v2.77h3.57c2.08-1.92 3.28-4.74 3.28-8.09z"/>

<path fill="#34A853" d="M12 23c2.97 0 5.46-.98 7.28-2.66l-3.57-2.77c-.98.66-2.23 1.06-3.71 1.06-2.86 0-5.29-1.93-6.16-4.53H2.18v2.84C3.99 20.53 7.7 23 12 23z"/>

<path fill="#FBBC05" d="M5.84 14.09c-.22-.66-.35-1.36-.35-2.09s.13-1.43.35-2.09V7.07H2.18C1.43 8.55 1 10.22 1 12s.43 3.45 1.18 4.93l2.85-2.22.81-.62z"/>

<path fill="#EA4335" d="M12 5.38c1.62 0 3.06.56 4.21 1.64l3.15-3.15C17.45 2.09 14.97 1 12 1 7.7 1 3.99 3.47 2.18 7.07l3.66 2.84c.87-2.6 3.3-4.53 6.16-4.53z"/>

</svg>

),

success: CheckCircle,

error: AlertCircle,

warning: AlertTriangle,

info: Info,

loading: Loader2,

visibility: Eye,

visibilityOff: EyeOff,

security: Shield,

biometric: Fingerprint

}

// Status Indicators

const StatusIndicator = ({ type, message }: { type: keyof typeof AuthIcons, message: string }) => {

const Icon = AuthIcons[type]

const colors = {

success: 'text-green-400',

error: 'text-red-400',

warning: 'text-yellow-400',

info: 'text-blue-400',

loading: 'text-slate-400'

}

return (

<div className={`flex items-center gap-2 text-sm ${colors[type] || 'text-slate-400'}`}>

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

<span>{message}</span>

</div>

)

}

// Visual Feedback for State Changes

const FormFieldWithFeedback = ({ error, success, ...props }) => (

<div className="relative">

<Input

className={cn(

"pl-10 pr-10 transition-colors duration-200",

error && "border-red-500 focus:border-red-500",

success && "border-green-500 focus:border-green-500"

)}

{...props}

/>

{/\* Status Icon \*/}

<div className="absolute right-3 top-3">

{error && <AlertCircle className="w-4 h-4 text-red-400" />}

{success && <CheckCircle className="w-4 h-4 text-green-400" />}

</div>

</div>

)

**Typography Scale and Emphasis Patterns**

// Typography System for Auth Pages

const AuthTypography = {

// Headings

pageTitle: 'text-2xl sm:text-3xl font-bold text-white',

sectionTitle: 'text-lg font-semibold text-white',

cardTitle: 'text-base font-medium text-white',

// Body Text

body: 'text-sm text-slate-300',

bodySecondary: 'text-xs text-slate-400',

caption: 'text-xs text-slate-500',

// Interactive Elements

link: 'text-blue-400 hover:text-blue-300 underline-offset-2 hover:underline',

button: 'font-medium',

// Status Text

error: 'text-xs text-red-400',

success: 'text-xs text-green-400',

warning: 'text-xs text-yellow-400',

// Form Elements

label: 'text-sm font-medium text-slate-300',

placeholder: 'text-slate-400',

helper: 'text-xs text-slate-500'

}

// Emphasis Patterns

const EmphasisPatterns = {

primary: 'font-semibold text-white',

secondary: 'font-medium text-slate-300',

muted: 'text-slate-400',

accent: 'text-blue-400 font-medium',

danger: 'text-red-400 font-medium',

success: 'text-green-400 font-medium'

}

**11. Security & Validation**

**Input Validation Schemas (Zod)**

import { z } from 'zod'

// Sign In Form Validation

const signInSchema = z.object({

email: z

.string()

.min(1, 'Email is required')

.email('Please enter a valid email address')

.max(255, 'Email must be less than 255 characters')

.toLowerCase()

.trim(),

password: z

.string()

.min(1, 'Password is required')

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

.max(128, 'Password must be less than 128 characters'),

rememberMe: z

.boolean()

.optional()

.default(false),

twoFactorCode: z

.string()

.optional()

.refine((val) => !val || /^\d{6}$/.test(val), {

message: 'Two-factor code must be 6 digits'

})

})

// Forgot Password Validation

const forgotPasswordSchema = z.object({

email: z

.string()

.min(1, 'Email is required')

.email('Please enter a valid email address')

.max(255, 'Email must be less than 255 characters')

.toLowerCase()

.trim()

})

// Password Reset Validation

const resetPasswordSchema = z.object({

token: z

.string()

.min(1, 'Reset token is required'),

password: z

.string()

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

.max(128, 'Password must be less than 128 characters')

.regex(/[A-Z]/, 'Password must contain at least one uppercase letter')

.regex(/[a-z]/, 'Password must contain at least one lowercase letter')

.regex(/[0-9]/, 'Password must contain at least one number')

.regex(/[^a-zA-Z0-9]/, 'Password must contain at least one special character'),

confirmPassword: z.string()

}).refine((data) => data.password === data.confirmPassword, {

message: "Passwords don't match",

path: ["confirmPassword"]

})

// Two-Factor Authentication

const twoFactorSchema = z.object({

code: z

.string()

.regex(/^\d{6}$/, 'Two-factor code must be 6 digits'),

userId: z

.string()

.uuid('Invalid user ID'),

rememberDevice: z

.boolean()

.optional()

.default(false)

})

// Form Validation Hook

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

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

const validate = useCallback((data: Partial<T>): boolean => {

try {

schema.parse(data)

setErrors({})

return true

} catch (error) {

if (error instanceof z.ZodError) {

const fieldErrors = error.errors.reduce((acc, err) => {

const path = err.path.join('.')

acc[path] = err.message

return acc

}, {} as Record<string, string>)

setErrors(fieldErrors)

}

return false

}

}, [schema])

const validateField = useCallback((field: string, value: any): boolean => {

try {

const fieldSchema = schema.shape[field]

if (fieldSchema) {

fieldSchema.parse(value)

setErrors(prev => ({ ...prev, [field]: '' }))

return true

}

} catch (error) {

if (error instanceof z.ZodError) {

setErrors(prev => ({ ...prev, [field]: error.errors[0]?.message || 'Invalid value' }))

}

}

return false

}, [schema])

return { errors, validate, validateField, clearErrors: () => setErrors({}) }

}

**Authentication & Authorization**

// Authentication Security Implementation

class AuthSecurity {

private readonly maxAttempts = 5

private readonly lockoutDuration = 15 \* 60 \* 1000 // 15 minutes

private readonly sessionTimeout = 24 \* 60 \* 60 \* 1000 // 24 hours

// Rate limiting for login attempts

async checkRateLimit(identifier: string, action: string): Promise<boolean> {

const rateLimitKey = `rate\_limit:${action}:${identifier}`

const attempts = await redis.get(rateLimitKey)

if (attempts && parseInt(attempts) >= this.maxAttempts) {

const ttl = await redis.ttl(rateLimitKey)

if (ttl > 0) {

throw new AuthError('RATE\_LIMITED', `Too many attempts. Try again in ${Math.ceil(ttl / 60)} minutes.`)

}

}

return true

}

// Record failed login attempt

async recordFailedAttempt(identifier: string, action: string): Promise<void> {

const rateLimitKey = `rate\_limit:${action}:${identifier}`

const attempts = await redis.incr(rateLimitKey)

if (attempts === 1) {

await redis.expire(rateLimitKey, this.lockoutDuration / 1000)

}

// Log security event

await this.logSecurityEvent({

type: 'login\_failed',

identifier,

timestamp: new Date(),

metadata: { attempts }

})

}

// Generate secure session token

generateSessionToken(): string {

return crypto.randomBytes(32).toString('hex')

}

// Verify session token

async verifySession(token: string): Promise<Session | null> {

const session = await db.session.findUnique({

where: { sessionToken: token, revokedAt: null },

include: { user: true }

})

if (!session || session.expiresAt < new Date()) {

return null

}

// Update last accessed

await db.session.update({

where: { id: session.id },

data: { lastAccessed: new Date() }

})

return session

}

// Password security

async hashPassword(password: string): Promise<string> {

const saltRounds = 12

return bcrypt.hash(password, saltRounds)

}

async verifyPassword(password: string, hash: string): Promise<boolean> {

return bcrypt.compare(password, hash)

}

// Device fingerprinting for security

async generateDeviceFingerprint(request: Request): Promise<string> {

const components = [

request.headers['user-agent'] || '',

request.headers['accept-language'] || '',

request.headers['accept-encoding'] || '',

request.ip || ''

]

return crypto

.createHash('sha256')

.update(components.join('|'))

.digest('hex')

}

// Two-factor authentication

async generateTwoFactorSecret(): Promise<string> {

return speakeasy.generateSecret({

name: 'Kurzora',

issuer: 'Kurzora Trading Platform'

}).base32

}

async verifyTwoFactorCode(secret: string, code: string): Promise<boolean> {

return speakeasy.totp.verify({

secret,

encoding: 'base32',

token: code,

window: 1 // Allow 1 step tolerance

})

}

}

// JWT Token Management

class TokenManager {

private readonly accessTokenSecret = process.env.JWT\_ACCESS\_SECRET!

private readonly refreshTokenSecret = process.env.JWT\_REFRESH\_SECRET!

private readonly accessTokenExpiry = '15m'

private readonly refreshTokenExpiry = '7d'

generateAccessToken(payload: any): string {

return jwt.sign(payload, this.accessTokenSecret, {

expiresIn: this.accessTokenExpiry,

issuer: 'kurzora-auth',

audience: 'kurzora-api'

})

}

generateRefreshToken(payload: any): string {

return jwt.sign(payload, this.refreshTokenSecret, {

expiresIn: this.refreshTokenExpiry,

issuer: 'kurzora-auth',

audience: 'kurzora-api'

})

}

verifyAccessToken(token: string): any {

try {

return jwt.verify(token, this.accessTokenSecret)

} catch (error) {

throw new AuthError('INVALID\_TOKEN', 'Access token is invalid or expired')

}

}

verifyRefreshToken(token: string): any {

try {

return jwt.verify(token, this.refreshTokenSecret)

} catch (error) {

throw new AuthError('INVALID\_TOKEN', 'Refresh token is invalid or expired')

}

}

}

**Data Sanitization for XSS Prevention**

// Input Sanitization

import DOMPurify from 'isomorphic-dompurify'

class InputSanitizer {

// Sanitize HTML content

static sanitizeHtml(input: string): string {

return DOMPurify.sanitize(input, {

ALLOWED\_TAGS: [], // No HTML tags allowed

ALLOWED\_ATTR: []

})

}

// Sanitize for SQL injection (using parameterized queries)

static sanitizeForDb(input: string): string {

// Remove null bytes and control characters

return input.replace(/[\x00-\x1F\x7F]/g, '')

}

// Sanitize email input

static sanitizeEmail(email: string): string {

return email

.toLowerCase()

.trim()

.replace(/[^\w@.-]/g, '') // Only allow alphanumeric, @, ., -

}

// Sanitize phone number

static sanitizePhone(phone: string): string {

return phone.replace(/[^\d+()-\s]/g, '')

}

// General text sanitization

static sanitizeText(text: string): string {

return text

.trim()

.replace(/[\x00-\x08\x0B-\x0C\x0E-\x1F\x7F]/g, '') // Remove control characters

.substring(0, 1000) // Limit length

}

}

// Form Input Sanitization Hook

const useSanitizedInput = () => {

const sanitizeFormData = useCallback((data: Record<string, any>) => {

const sanitized = { ...data }

Object.keys(sanitized).forEach(key => {

if (typeof sanitized[key] === 'string') {

switch (key) {

case 'email':

sanitized[key] = InputSanitizer.sanitizeEmail(sanitized[key])

break

case 'phone':

sanitized[key] = InputSanitizer.sanitizePhone(sanitized[key])

break

default:

sanitized[key] = InputSanitizer.sanitizeText(sanitized[key])

}

}

})

return sanitized

}, [])

return { sanitizeFormData }

}

**Rate Limiting Implementation**

// Client-side rate limiting

const useClientRateLimit = () => {

const [attemptCount, setAttemptCount] = useState(0)

const [lastAttempt, setLastAttempt] = useState<number | null>(null)

const [isBlocked, setIsBlocked] = useState(false)

const checkRateLimit = useCallback((): boolean => {

const now = Date.now()

const timeSinceLastAttempt = lastAttempt ? now - lastAttempt : Infinity

// Reset counter if enough time has passed

if (timeSinceLastAttempt > 15 \* 60 \* 1000) { // 15 minutes

setAttemptCount(0)

setIsBlocked(false)

return false

}

// Check if blocked

if (attemptCount >= 5) {

setIsBlocked(true)

return true

}

return false

}, [attemptCount, lastAttempt])

const recordAttempt = useCallback(() => {

setAttemptCount(prev => prev + 1)

setLastAttempt(Date.now())

}, [])

const resetAttempts = useCallback(() => {

setAttemptCount(0)

setLastAttempt(null)

setIsBlocked(false)

}, [])

return {

isBlocked,

attemptCount,

checkRateLimit,

recordAttempt,

resetAttempts

}

}

// Server-side rate limiting middleware

const createRateLimiter = (options: {

windowMs: number

max: number

keyGenerator?: (req: Request) => string

}) => {

return async (req: Request, res: Response, next: NextFunction) => {

const key = options.keyGenerator ? options.keyGenerator(req) : req.ip

const rateLimitKey = `rate\_limit:${key}`

const current = await redis.incr(rateLimitKey)

if (current === 1) {

await redis.expire(rateLimitKey, Math.ceil(options.windowMs / 1000))

}

if (current > options.max) {

return res.status(429).json({

error: {

code: 'RATE\_LIMITED',

message: 'Too many requests, please try again later.',

retryAfter: await redis.ttl(rateLimitKey)

}

})

}

res.setHeader('X-RateLimit-Limit', options.max)

res.setHeader('X-RateLimit-Remaining', Math.max(0, options.max - current))

next()

}

}

**12. Environment & Configuration**

**Environment Variables**

# API Configuration

VITE\_API\_URL=https://api.kurzora.com

VITE\_API\_VERSION=v1

VITE\_WS\_URL=wss://ws.kurzora.com

# Authentication

VITE\_JWT\_ISSUER=kurzora-auth

JWT\_ACCESS\_SECRET=your-super-secret-access-key

JWT\_REFRESH\_SECRET=your-super-secret-refresh-key

JWT\_ACCESS\_EXPIRY=15m

JWT\_REFRESH\_EXPIRY=7d

# OAuth Providers

VITE\_GOOGLE\_CLIENT\_ID=your-google-client-id.googleusercontent.com

GOOGLE\_CLIENT\_SECRET=your-google-client-secret

VITE\_GITHUB\_CLIENT\_ID=your-github-client-id

GITHUB\_CLIENT\_SECRET=your-github-client-secret

# Database

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

REDIS\_URL=redis://localhost:6379

# Email Services

SENDGRID\_API\_KEY=SG.your-sendgrid-api-key

SMTP\_HOST=smtp.sendgrid.net

SMTP\_PORT=587

SMTP\_USER=apikey

SMTP\_PASS=your-sendgrid-api-key

# Security

BCRYPT\_ROUNDS=12

RATE\_LIMIT\_WINDOW\_MS=900000

RATE\_LIMIT\_MAX\_ATTEMPTS=5

SESSION\_TIMEOUT\_MS=86400000

DEVICE\_TRUST\_DURATION\_DAYS=30

# Feature Flags

VITE\_ENABLE\_GOOGLE\_AUTH=true

VITE\_ENABLE\_GITHUB\_AUTH=false

VITE\_ENABLE\_BIOMETRIC\_AUTH=true

VITE\_ENABLE\_TWO\_FACTOR=true

VITE\_ENABLE\_DEVICE\_TRUST=true

VITE\_ENABLE\_PASSWORD\_STRENGTH=true

VITE\_ENABLE\_RATE\_LIMITING=true

# Monitoring & Analytics

VITE\_SENTRY\_DSN=https://your-sentry-dsn@sentry.io/project-id

SENTRY\_AUTH\_TOKEN=your-sentry-auth-token

VITE\_GA\_TRACKING\_ID=G-XXXXXXXXXX

VITE\_MIXPANEL\_TOKEN=your-mixpanel-token

# Development

VITE\_DEBUG\_MODE=false

VITE\_MOCK\_AUTH=false

VITE\_DISABLE\_RATE\_LIMITING=false

NODE\_ENV=production

# Security Headers

VITE\_CSP\_NONCE=true

VITE\_ENABLE\_HSTS=true

VITE\_ENABLE\_XSS\_PROTECTION=true

**Feature Flags Implementation**

// Feature flag configuration

interface FeatureFlags {

enableGoogleAuth: boolean

enableGithubAuth: boolean

enableBiometricAuth: boolean

enableTwoFactor: boolean

enableDeviceTrust: boolean

enablePasswordStrength: boolean

enableRateLimit: boolean

enableRememberMe: boolean

enableSecurityEvents: boolean

enableAdvancedLogging: boolean

}

// Feature flag hook

const useFeatureFlags = (): FeatureFlags => {

return useMemo(() => ({

enableGoogleAuth: import.meta.env.VITE\_ENABLE\_GOOGLE\_AUTH === 'true',

enableGithubAuth: import.meta.env.VITE\_ENABLE\_GITHUB\_AUTH === 'true',

enableBiometricAuth: import.meta.env.VITE\_ENABLE\_BIOMETRIC\_AUTH === 'true',

enableTwoFactor: import.meta.env.VITE\_ENABLE\_TWO\_FACTOR === 'true',

enableDeviceTrust: import.meta.env.VITE\_ENABLE\_DEVICE\_TRUST === 'true',

enablePasswordStrength: import.meta.env.VITE\_ENABLE\_PASSWORD\_STRENGTH === 'true',

enableRateLimit: import.meta.env.VITE\_ENABLE\_RATE\_LIMITING === 'true',

enableRememberMe: import.meta.env.VITE\_ENABLE\_REMEMBER\_ME !== 'false',

enableSecurityEvents: import.meta.env.VITE\_ENABLE\_SECURITY\_EVENTS !== 'false',

enableAdvancedLogging: import.meta.env.NODE\_ENV === 'development' || import.meta.env.VITE\_DEBUG\_MODE === 'true'

}), [])

}

// Conditional rendering based on feature flags

const ConditionalFeature: React.FC<{

flag: keyof FeatureFlags

children: React.ReactNode

fallback?: React.ReactNode

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

const featureFlags = useFeatureFlags()

return featureFlags[flag] ? <>{children}</> : <>{fallback}</>

}

// Usage in components

const SignInForm = () => {

const featureFlags = useFeatureFlags()

return (

<form>

{/\* Standard email/password form \*/}

<ConditionalFeature flag="enableGoogleAuth">

<GoogleSignInButton />

</ConditionalFeature>

<ConditionalFeature flag="enableBiometricAuth">

<BiometricAuthButton />

</ConditionalFeature>

<ConditionalFeature flag="enableRememberMe">

<RememberMeCheckbox />

</ConditionalFeature>

</form>

)

}

**Third-party Service Configurations**

// Google OAuth Configuration

const googleAuthConfig = {

clientId: import.meta.env.VITE\_GOOGLE\_CLIENT\_ID!,

scopes: ['email', 'profile'],

buttonText: 'Continue with Google',

theme: 'filled\_black',

size: 'large',

logo\_alignment: 'left'

}

// Sentry Configuration

const sentryConfig = {

dsn: import.meta.env.VITE\_SENTRY\_DSN,

environment: import.meta.env.NODE\_ENV,

tracesSampleRate: import.meta.env.NODE\_ENV === 'production' ? 0.1 : 1.0,

beforeSend(event: any) {

// Filter out sensitive data

if (event.request?.data) {

delete event.request.data.password

delete event.request.data.token

}

return event

}

}

// Redis Configuration

const redisConfig = {

url: process.env.REDIS\_URL,

retryDelayOnFailover: 100,

maxRetriesPerRequest: 3,

lazyConnect: true,

keepAlive: 30000

}

// Email Service Configuration

const emailConfig = {

provider: 'sendgrid',

apiKey: process.env.SENDGRID\_API\_KEY,

from: 'noreply@kurzora.com',

templates: {

welcomeEmail: 'd-welcome-template-id',

passwordReset: 'd-password-reset-template-id',

emailVerification: 'd-email-verification-template-id',

securityAlert: 'd-security-alert-template-id'

}

}

// WebAuthn Configuration (for biometric auth)

const webAuthnConfig = {

rpName: 'Kurzora Trading Platform',

rpId: 'kurzora.com',

origin: import.meta.env.VITE\_APP\_URL,

timeout: 60000,

attestation: 'none' as const,

userVerification: 'preferred' as const,

authenticatorSelection: {

authenticatorAttachment: 'platform' as const,

userVerification: 'preferred' as const,

requireResidentKey: false

}

}

**Monitoring & Analytics**

// Analytics tracking

const trackAuthEvent = (event: string, properties?: Record<string, any>) => {

// Google Analytics

if (window.gtag) {

window.gtag('event', event, {

event\_category: 'authentication',

...properties

})

}

// Mixpanel

if (window.mixpanel) {

window.mixpanel.track(event, {

category: 'authentication',

timestamp: new Date().toISOString(),

...properties

})

}

// Custom analytics

fetch('/api/analytics/track', {

method: 'POST',

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

body: JSON.stringify({

event,

properties: {

...properties,

timestamp: new Date().toISOString(),

userAgent: navigator.userAgent,

url: window.location.href

}

})

}).catch(console.error)

}

// Error reporting

const reportAuthError = (error: Error, context?: Record<string, any>) => {

// Sentry

if (window.Sentry) {

window.Sentry.captureException(error, {

tags: { component: 'auth' },

extra: context

})

}

// Custom error logging

fetch('/api/errors/report', {

method: 'POST',

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

body: JSON.stringify({

error: {

message: error.message,

stack: error.stack,

name: error.name

},

context,

timestamp: new Date().toISOString(),

url: window.location.href,

userAgent: navigator.userAgent

})

}).catch(console.error)

}

**13. Cross-Screen Data Flow**

**Authentication State Propagation**

// Global Auth State Management

interface AuthState {

user: User | null

isAuthenticated: boolean

isLoading: boolean

sessionId: string | null

lastActivity: number

deviceTrusted: boolean

requiresTwoFactor: boolean

}

const useAuthStore = create<AuthState & AuthActions>((set, get) => ({

// State

user: null,

isAuthenticated: false,

isLoading: true,

sessionId: null,

lastActivity: Date.now(),

deviceTrusted: false,

requiresTwoFactor: false,

// Actions

setUser: (user: User) => set({

user,

isAuthenticated: true,

lastActivity: Date.now()

}),

setSession: (sessionId: string) => set({ sessionId }),

logout: () => set({

user: null,

isAuthenticated: false,

sessionId: null,

deviceTrusted: false,

requiresTwoFactor: false

}),

updateLastActivity: () => set({ lastActivity: Date.now() }),

setTwoFactorRequired: (required: boolean) => set({ requiresTwoFactor: required }),

setDeviceTrusted: (trusted: boolean) => set({ deviceTrusted: trusted })

}))

// Auth state synchronization across tabs

const useAuthSync = () => {

const authStore = useAuthStore()

useEffect(() => {

const handleStorageChange = (e: StorageEvent) => {

if (e.key === 'auth-store') {

const newState = e.newValue ? JSON.parse(e.newValue) : null

if (newState) {

authStore.setUser(newState.user)

authStore.setSession(newState.sessionId)

} else {

authStore.logout()

}

}

}

window.addEventListener('storage', handleStorageChange)

return () => window.removeEventListener('storage', handleStorageChange)

}, [authStore])

// Sync auth state to localStorage

useEffect(() => {

const { user, sessionId, isAuthenticated } = authStore

if (isAuthenticated && user && sessionId) {

localStorage.setItem('auth-store', JSON.stringify({ user, sessionId }))

} else {

localStorage.removeItem('auth-store')

}

}, [authStore.user, authStore.sessionId, authStore.isAuthenticated])

}

**Real-time Security Monitoring**

// Security event streaming

const useSecurityEvents = () => {

const [events, setEvents] = useState<SecurityEvent[]>([])

const { user } = useAuthStore()

useEffect(() => {

if (!user) return

const eventSource = new EventSource(`/api/security/events/stream?userId=${user.id}`)

eventSource.onmessage = (event) => {

const securityEvent = JSON.parse(event.data)

setEvents(prev => [securityEvent, ...prev.slice(0, 49)]) // Keep last 50 events

// Handle critical security events

if (securityEvent.severity === 'critical') {

handleCriticalSecurityEvent(securityEvent)

}

}

eventSource.onerror = (error) => {

console.error('Security event stream error:', error)

eventSource.close()

}

return () => eventSource.close()

}, [user])

return events

}

// Session validation and refresh

const useSessionValidation = () => {

const authStore = useAuthStore()

const navigate = useNavigate()

useEffect(() => {

if (!authStore.isAuthenticated) return

const validateSession = async () => {

try {

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

headers: {

'Authorization': `Bearer ${authStore.sessionId}`

}

})

if (!response.ok) {

// Session invalid, logout user

authStore.logout()

navigate('/signin')

}

} catch (error) {

console.error('Session validation error:', error)

}

}

// Validate session every 5 minutes

const interval = setInterval(validateSession, 5 \* 60 \* 1000)

// Validate immediately

validateSession()

return () => clearInterval(interval)

}, [authStore.isAuthenticated, authStore.sessionId])

}

// Activity tracking for session timeout

const useActivityTracking = () => {

const authStore = useAuthStore()

const sessionTimeout = 30 \* 60 \* 1000 // 30 minutes

useEffect(() => {

if (!authStore.isAuthenticated) return

const trackActivity = () => {

authStore.updateLastActivity()

}

const events = ['mousedown', 'mousemove', 'keypress', 'scroll', 'touchstart']

events.forEach(event => {

document.addEventListener(event, trackActivity, true)

})

// Check for session timeout

const timeoutCheck = setInterval(() => {

const timeSinceActivity = Date.now() - authStore.lastActivity

if (timeSinceActivity > sessionTimeout) {

authStore.logout()

// Show session timeout modal

}

}, 60 \* 1000) // Check every minute

return () => {

events.forEach(event => {

document.removeEventListener(event, trackActivity, true)

})

clearInterval(timeoutCheck)

}

}, [authStore.isAuthenticated, authStore.lastActivity, sessionTimeout])

}

**Form State Persistence**

// Form persistence across navigation

const useFormPersistence = <T>(formKey: string, initialData: T) => {

const [formData, setFormData] = useState<T>(initialData)

// Load from localStorage on mount

useEffect(() => {

const savedData = localStorage.getItem(`form-${formKey}`)

if (savedData) {

try {

const parsed = JSON.parse(savedData)

setFormData({ ...initialData, ...parsed })

} catch (error) {

console.error('Error parsing saved form data:', error)

}

}

}, [formKey, initialData])

// Save to localStorage when data changes

useEffect(() => {

const timeoutId = setTimeout(() => {

localStorage.setItem(`form-${formKey}`, JSON.stringify(formData))

}, 500) // Debounce saves

return () => clearTimeout(timeoutId)

}, [formKey, formData])

const updateFormData = useCallback((updates: Partial<T>) => {

setFormData(prev => ({ ...prev, ...updates }))

}, [])

const clearFormData = useCallback(() => {

setFormData(initialData)

localStorage.removeItem(`form-${formKey}`)

}, [formKey, initialData])

return { formData, updateFormData, clearFormData }

}

// Navigation state preservation

const useNavigationState = () => {

const location = useLocation()

const [navigationHistory, setNavigationHistory] = useState<string[]>([])

useEffect(() => {

setNavigationHistory(prev => [...prev, location.pathname].slice(-10)) // Keep last 10 pages

}, [location.pathname])

const getPreviousPage = useCallback(() => {

return navigationHistory[navigationHistory.length - 2] || '/'

}, [navigationHistory])

return { navigationHistory, getPreviousPage }

}

**Cache Invalidation Strategies**

// Auth-related cache management

const useAuthCache = () => {

const queryClient = useQueryClient()

const invalidateUserData = useCallback(() => {

queryClient.invalidateQueries(['user'])

queryClient.invalidateQueries(['user-preferences'])

queryClient.invalidateQueries(['user-sessions'])

}, [queryClient])

const invalidateSecurityData = useCallback(() => {

queryClient.invalidateQueries(['security-events'])

queryClient.invalidateQueries(['trusted-devices'])

queryClient.invalidateQueries(['active-sessions'])

}, [queryClient])

const clearAllAuthCache = useCallback(() => {

queryClient.clear()

}, [queryClient])

// Auto-invalidate on authentication events

useEffect(() => {

const handleAuthEvent = (event: CustomEvent) => {

switch (event.detail.type) {

case 'login':

invalidateUserData()

break

case 'logout':

clearAllAuthCache()

break

case 'security-event':

invalidateSecurityData()

break

}

}

window.addEventListener('auth-event', handleAuthEvent as EventListener)

return () => window.removeEventListener('auth-event', handleAuthEvent as EventListener)

}, [invalidateUserData, invalidateSecurityData, clearAllAuthCache])

return {

invalidateUserData,

invalidateSecurityData,

clearAllAuthCache

}

}

// Optimistic updates for auth operations

const useOptimisticAuth = () => {

const queryClient = useQueryClient()

const optimisticLogin = useCallback((userData: User) => {

queryClient.setQueryData(['user'], userData)

queryClient.setQueryData(['auth-status'], { isAuthenticated: true })

}, [queryClient])

const optimisticLogout = useCallback(() => {

queryClient.setQueryData(['user'], null)

queryClient.setQueryData(['auth-status'], { isAuthenticated: false })

}, [queryClient])

return { optimisticLogin, optimisticLogout }

}

**Implementation Priority**

**Phase 1: Core Authentication (Week 1)**

1. ✅ **Basic email/password form** (Already implemented)
2. ✅ **Form validation and error handling** (Already implemented)
3. ✅ **Loading states and UI feedback** (Already implemented)
4. **Backend API integration** (Replace mock auth with real endpoints)
5. **Session management and token handling**

**Phase 2: Enhanced Security (Week 2)**

1. **Google OAuth integration** (Replace mock with Firebase)
2. **Rate limiting and security monitoring**
3. **Two-factor authentication flow**
4. **Device trust and fingerprinting**
5. **Comprehensive error boundaries**

**Phase 3: Production Hardening (Week 3)**

1. **Advanced validation with Zod schemas**
2. **Accessibility improvements and testing**
3. **Performance optimization and lazy loading**
4. **Analytics and monitoring integration**
5. **Security hardening and penetration testing**

**✅ READY FOR CURSOR IMPLEMENTATION**

**This sign in page analysis is complete and production-ready!** The actual implementation shows:

* ✅ **Modern React + TypeScript + Tailwind architecture**
* ✅ **Professional UI/UX with loading states and error handling**
* ✅ **Google OAuth integration skeleton** ready for Firebase
* ✅ **Comprehensive form validation** and user feedback
* ✅ **Mobile-responsive design** with accessibility considerations
* ✅ **State management patterns** and context integration
* ✅ **Security-first approach** with rate limiting and validation

**Outstanding items for Cursor implementation:**

1. **Replace mock authentication** with real Firebase/backend APIs
2. **Implement two-factor authentication** flow and UI
3. **Add advanced security features** (device trust, biometrics)
4. **Complete testing suite** with unit and integration tests
5. **Performance optimization** and bundle splitting

**The UI foundation is excellent - now needs backend integration!** 🚀

*This analysis provides complete technical specifications for implementing a production-ready sign in page in Cursor.*