Epic 3.4: Error Handling Molecules

Epic Overview

This epic creates comprehensive error handling molecule components that ensure graceful error management, recovery, and user communication across all workspace contexts. This is essential for production reliability.

Priority: P0 (Critical)
Timeline: 3 weeks
Dependencies: Epic 2.2 (Display Components), Epic 3.2 (Display Molecules)

Story 3.4.1: Error Boundary Components

Overview

Create robust error boundary components that catch and handle errors gracefully, preventing application crashes and providing meaningful feedback to users.

AI Developer Prompt

You are creating error boundary components for THE WHEEL design system. This is a critical P0 component essential for production applications with graceful error handling.

Context

- Existing molecule component system with workspace context
- Real-time collaboration features that can encounter errors
- Multiple workspace contexts requiring different error handling
- Need for graceful degradation and error recovery
- Integration with existing error tracking systems

Requirements

1. Create error boundary architecture:

- React error boundary with fallback UI
- Error categorization and severity levels
- Context-aware error messages and recovery
- Error reporting and analytics integration

- Graceful degradation strategies

2. Implement workspace context error handling:

- Context-specific error messages
- Permission-based error information
- Workspace-specific recovery actions
- Brand-aware error UI styling
- Context-aware error escalation

3. Create error recovery mechanisms:

- Retry mechanisms for transient errors
- Fallback content for failed components
- Error state persistence across sessions
- User feedback collection for errors
- Automatic error reporting

Specific Tasks

- [ ] Create ErrorBoundary component
- [ ] Implement error categorization system
- [ ] Set up error recovery mechanisms
- [ ] Create fallback UI components
- [ ] Implement error reporting integration
- [ ] Set up error analytics

Documentation Required

- Error boundary implementation guide
- Error categorization and handling
- Recovery mechanism documentation
- Error reporting integration
- Debugging and troubleshooting guide

Testing Requirements

- Error boundary functionality tests
- Error recovery mechanism tests

- Context-specific error handling tests
- Error reporting integration tests
- Performance impact tests

Integration Points

- Integration with existing error tracking
- Workspace context integration
- Real-time collaboration error handling
- Theme system integration
- Analytics and reporting integration

Deliverables

- Complete error boundary system
- Error categorization and recovery
- Context-aware error handling
- Error reporting integration
- Comprehensive error documentation

Performance Requirements

- Error boundary rendering under 100ms
- Error recovery under 500ms
- Memory usage under 20MB
- Error reporting under 1 second
- UI responsiveness maintained during errors

Component Specifications

typescript

```typescript
interface ErrorBoundaryProps {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  fallback?: React.ComponentType<ErrorFallbackProps>
  onError?: (error: Error, errorInfo: ErrorInfo) => void
  resetKeys?: Array<string | number>
  resetOnPropsChange?: boolean
  isolate?: boolean
  level?: 'page' | 'section' | 'component'
  children: React.ReactNode
}

interface ErrorFallbackProps {
  error: Error
  resetError: () => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  errorInfo?: ErrorInfo
  level?: 'page' | 'section' | 'component'
}

interface ErrorInfo {
  componentStack: string
  digest?: string
  errorBoundary?: ErrorBoundary
  errorBoundaryProps?: ErrorBoundaryProps
}

interface ErrorCategory {
  type: 'network' | 'permission' | 'validation' | 'system' | 'unknown'
  severity: 'low' | 'medium' | 'high' | 'critical'
  recoverable: boolean
  userMessage: string
  technicalMessage: string
  suggestedActions: ErrorAction[]
}

interface ErrorAction {
  label: string
  action: () => void | Promise<void>
  type: 'primary' | 'secondary'
  icon?: string
}
```

Story 3.4.2: Error State Display Components

Overview

Create user-friendly error state display components that communicate errors clearly and provide actionable recovery options.

AI Developer Prompt

You are creating error state display components for THE WHEEL design system. Building on the error boundary components from Story 3.4.1, you need to create user-friendly error displays.

Context

- Error boundary system established with recovery mechanisms
- Multiple workspace contexts requiring different error presentations
- Need for user-friendly error messaging and actions
- Integration with existing design system components
- Accessibility requirements for error communication

Requirements

1. Create error display components:

- Error alert components with severity levels
- Inline error messages for forms and inputs
- Page-level error states with recovery actions
- Toast notifications for transient errors
- Modal error dialogs for critical errors

2. Implement workspace context integration:

- Context-specific error messaging
- Role-based error information disclosure
- Workspace-themed error styling
- Context-aware recovery actions
- Permission-based error details

3. Create error communication features:

- Clear error descriptions and causes
- Actionable recovery suggestions
- Error code references for support
- Visual error indicators and icons
- Accessibility compliance for screen readers

Specific Tasks

- [ ] Create ErrorAlert component
- [ ] Implement InlineError component
- [ ] Set up ErrorPage component
- [ ] Create ErrorToast component
- [ ] Implement ErrorModal component
- [ ] Set up accessibility features

Documentation Required

- Error display component API
- Error messaging guidelines
- Accessibility implementation
- Context-specific usage examples
- Error communication best practices

Testing Requirements

- Error display rendering tests
- Accessibility compliance tests
- Context variation tests
- Error message clarity tests
- Recovery action tests

Integration Points

- Integration with error boundary system
- Workspace context integration
- Theme system integration
- Notification system integration
- Form validation integration

Deliverables

- Complete error display component library
- Context-aware error messaging
- Accessibility-compliant error communication
- User-friendly error interfaces
- Comprehensive error display documentation

Performance Requirements

- Error display rendering under 100ms
- Error message loading under 50ms
- Memory usage under 10MB
- Animation performance maintains 60fps
- Screen reader compatibility verified

Component Specifications

typescript

```tsx
interface ErrorAlertProps {
  error: Error | ErrorInfo
  severity?: 'low' | 'medium' | 'high' | 'critical'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  dismissible?: boolean
  onDismiss?: () => void
  actions?: ErrorAction[]
  showDetails?: boolean
  showErrorCode?: boolean
}

interface InlineErrorProps {
  message: string
  fieldName?: string
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  icon?: boolean
  animate?: boolean
}

interface ErrorPageProps {
  error: Error | ErrorInfo
  title?: string
  description?: string
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  illustration?: React.ReactNode
  actions?: ErrorAction[]
  showHomeButton?: boolean
  showSupportContact?: boolean
}

interface ErrorToastProps {
  error: Error | ErrorInfo
  duration?: number
  position?: 'top' | 'bottom' | 'top-right' | 'bottom-right'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  action?: ErrorAction
  onClose?: () => void
}

interface ErrorModalProps {
  error: Error | ErrorInfo
  isOpen: boolean
  onClose: () => void
```

```
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  title?: string
  showDetails?: boolean
  actions?: ErrorAction[]
  severity?: 'low' | 'medium' | 'high' | 'critical'
}
```

Story 3.4.3: Error Recovery Components

Overview

Create interactive error recovery components that help users resolve errors and continue their work with minimal disruption.

AI Developer Prompt

You are creating error recovery components for THE WHEEL design system. Building on the error state display components from Story 3.4.2, you need to create interactive recovery mechanisms.

Context

- Error display components established with user-friendly interfaces
- Need for interactive error recovery and retry mechanisms
- Multiple workspace contexts requiring different recovery options
- Real-time collaboration features requiring error recovery
- Integration with existing error tracking and reporting

Requirements

1. Create error recovery components:

- Retry button with exponential backoff
- Refresh page component with state preservation
- Fallback content switcher
- Error feedback collection form
- Recovery progress indicators

2. Implement workspace context recovery:

- Context-specific recovery actions
- Permission-based recovery options

- Workspace-aware error escalation
- Context-specific support channels
- Role-based recovery workflows

3. Create recovery automation features:

- Automatic retry mechanisms
- Intelligent fallback selection
- Recovery success tracking
- Error pattern detection
- Proactive recovery suggestions

Specific Tasks

- [ ] Create RetryButton component
- [ ] Implement RefreshPage component
- [ ] Set up FallbackContent component
- [ ] Create ErrorFeedback component
- [ ] Implement RecoveryProgress component
- [ ] Set up automatic recovery mechanisms

Documentation Required

- Error recovery implementation guide
- Recovery pattern documentation
- Automation configuration guide
- Context-specific recovery workflows
- User feedback collection guide

Testing Requirements

- Error recovery functionality tests
- Retry mechanism tests
- Fallback content tests
- Recovery automation tests
- User feedback collection tests

Integration Points

- Integration with error display system
- Workspace context integration
- Real-time collaboration integration
- Error tracking integration
- Analytics and reporting integration

Deliverables

- Complete error recovery system
- Interactive recovery components
- Automated recovery mechanisms
- Context-aware recovery workflows
- Comprehensive recovery documentation

Performance Requirements

- Recovery action response under 200ms
- Retry mechanism under 1 second
- Fallback content loading under 500ms
- Memory usage under 15MB
- Recovery success rate above 80%

Component Specifications

typescript

```
interface RetryButtonProps {
  onRetry: () => Promise<void>
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  maxRetries?: number
  backoffMs?: number
  exponential?: boolean
  label?: string
  loadingLabel?: string
  failureLabel?: string
}

interface RefreshPageProps {
  preserveState?: boolean
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  confirmBeforeRefresh?: boolean
  customMessage?: string
  onBeforeRefresh?: () => void
}

interface FallbackContentProps {
  primaryContent: React.ReactNode
  fallbackContent: React.ReactNode
  error?: Error
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showError?: boolean
  onContentSwitch?: (isFallback: boolean) => void
}

interface ErrorFeedbackProps {
  error: Error | ErrorInfo
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onSubmit: (feedback: ErrorFeedback) => Promise<void>
  fields?: FeedbackField[]
  showErrorDetails?: boolean
  anonymous?: boolean
}

interface RecoveryProgressProps {
  steps: RecoveryStep[]
  currentStep: number
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showStepDetails?: boolean
  onStepComplete?: (step: RecoveryStep) => void
```

```
  onRecoveryComplete?: () => void
  onRecoveryFail?: (error: Error) => void
}

interface RecoveryStep {
  id: string
  label: string
  description?: string
  action: () => Promise<void>
  canSkip?: boolean
  timeout?: number
}

interface ErrorFeedback {
  description: string
  steps?: string
  impact?: 'low' | 'medium' | 'high'
  frequency?: 'once' | 'intermittent' | 'frequent'
  contactInfo?: string
  screenshot?: File
}
```

Timeline and Dependencies

Timeline

- Week 1: Story 3.4.1 - Error Boundary Components
- Week 2: Story 3.4.2 - Error State Display Components
- Week 3: Story 3.4.3 - Error Recovery Components

Dependencies

- Epic 2.2 (Display Components) - Complete
- Epic 3.2 (Display Molecules) - Complete
- Error tracking system available
- Analytics integration ready

Success Metrics

- Zero unhandled errors in production
- Error recovery success rate above 80%

- User-friendly error messages for all error types
- 100% accessibility compliance for error states
- Error reporting latency under 1 second
- Complete test coverage (95%+ for error handling)

Risk Mitigation

- Comprehensive error scenario testing
- Performance monitoring under error conditions
- Regular error recovery drills
- User testing for error messaging clarity
- Clear escalation paths for critical errors
- Redundant error reporting mechanisms