

Epic 4.3: Form Organisms

Epic Overview

Form Organisms are sophisticated, multi-component forms that handle complex business workflows in THE WHEEL design system. These organisms combine multiple molecules and atoms to create comprehensive form experiences that adapt to different workspace contexts and user roles.

Epic Goals:

- Create dynamic form builders for complex business processes
 - Implement workspace-specific form templates
 - Support multi-step workflows with validation
 - Enable collaborative form editing
 - Provide form analytics and tracking
-

Story 4.3.1: Form Builders

Overview

Enhance form building organisms to create sophisticated form building components that handle complex business workflows across different workspace contexts.

Context

- Complete data display system with workspace context
- Need dynamic form generation for various business processes
- Must support multi-step workflows and validation
- Form builders are critical for business process automation
- Need workspace-specific form templates and validation

Requirements

1. Enhance FormBuilder with workspace forms:

- Workspace-specific field types and validation
- Dynamic form generation from schemas
- Form template system
- Conditional field display

- Advanced validation rules

2. Enhance FormWizard with workspace wizards:

- Multi-step form workflows
- Step validation and navigation
- Progress tracking and persistence
- Conditional step branching
- Workspace-specific wizard templates

3. Add advanced form features:

- Form state persistence
- Auto-save functionality
- Form versioning
- Collaborative form editing
- Form analytics and tracking

Specific Tasks

- ☒ Enhance FormBuilder with workspace forms
- ☒ Add workspace-specific field types
- ☒ Enhance FormWizard with workspace wizards
- ☒ Implement multi-step form logic
- ☒ Add form state persistence
- ☒ Create auto-save functionality
- ☒ Add form validation system

Documentation Required

- Form builder system architecture
- Workspace-specific form patterns
- Multi-step form implementation
- Form validation and persistence
- Form template system
- Accessibility guidelines

Testing Requirements

- Form builder functionality tests
- Multi-step form workflow tests
- Form validation tests
- State persistence tests
- Auto-save functionality tests
- Accessibility compliance tests
- Cross-browser compatibility tests

Integration Points

- Integration with workspace context providers
- Form template system integration
- Validation service integration
- State persistence integration
- Auto-save service integration

Deliverables

- Enhanced FormBuilder with workspace forms
- FormWizard with workspace wizards
- Form state persistence system
- Auto-save functionality
- Form validation system
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface FormBuilderProps {  
  context?: 'consultant' | 'client' | 'admin' | 'neutral'  
  schema: FormSchema  
  initialData?: Record<string, any>  
  onSubmit?: (data: Record<string, any>) => void  
  onChange?: (data: Record<string, any>) => void  
  onValidationChange?: (errors: Record<string, string>) => void  
  template?: string  
  workspaceId?: string  
  autoSave?: boolean  
  collaborative?: boolean  
  readonly?: boolean  
  permissions?: string[]  
}
```

```
interface FormWizardProps {  
  context?: 'consultant' | 'client' | 'admin' | 'neutral'  
  steps: FormWizardStep[]  
  initialData?: Record<string, any>  
  onStepChange?: (step: number, data: Record<string, any>) => void  
  onComplete?: (data: Record<string, any>) => void  
  onCancel?: () => void  
  template?: string  
  workspaceId?: string  
  autoSave?: boolean  
  showProgress?: boolean  
  allowStepSkip?: boolean  
  permissions?: string[]  
}
```

```
interface FormSchema {  
  fields: FormField[]  
  validation?: ValidationRule[]  
  conditional?: ConditionalRule[]  
  layout?: FormLayout  
  workspaceContext?: 'consultant' | 'client' | 'admin' | 'neutral'  
  permissions?: string[]  
}
```

```
interface FormField {  
  name: string  
  type: string  
  label: string
```

```
placeholder?: string
required?: boolean
validation?: ValidationRule[]
conditional?: ConditionalRule[]
workspaceContext?: 'consultant' | 'client' | 'admin' | 'neutral'
permission?: string
props?: Record<string, any>
}
```

```
interface FormWizardStep {
  id: string
  title: string
  description?: string
  fields: FormField[]
  validation?: ValidationRule[]
  conditional?: ConditionalRule[]
  optional?: boolean
  permissions?: string[]
}
```

```
interface ValidationRule {
  type: 'required' | 'minLength' | 'maxLength' | 'pattern' | 'custom'
  value?: any
  message: string
  workspaceContext?: 'consultant' | 'client' | 'admin' | 'neutral'
}
```

```
interface ConditionalRule {
  field: string
  operator: 'equals' | 'not_equals' | 'contains' | 'greater_than' | 'less_than'
  value: any
  action: 'show' | 'hide' | 'required' | 'disabled'
}
```

Implementation Example

jsx

// FormBuilder implementation

```
function FormBuilder({ schema, context, onSubmit, autoSave }) {  
  const { workspace, hasPermission } = useWorkspace()  
  const [formData, setFormData] = useState({})  
  const [errors, setErrors] = useState({})  
  
  // Auto-save functionality  
  useEffect(() => {  
    if (autoSave) {  
      const timer = setTimeout(() => {  
        saveFormData(formData)  
      }, 2000)  
      return () => clearTimeout(timer)  
    }  
  }, [formData, autoSave])  
  
  return (  
    <Form onSubmit={handleSubmit}>  
      {schema.fields.map(field => (  
        <FormField  
          key={field.name}  
          field={field}  
          value={formData[field.name]}  
          error={errors[field.name]}  
          context={context}  
          onChange={handleFieldChange}  
        />  
      ))}  
      <FormActions>  
        <Button variant="primary" type="submit">  
          Submit  
        </Button>  
      </FormActions>  
    </Form>  
  )  
}
```

Story 4.3.2: Workspace-Specific Forms

Overview

Build workspace-specific form organisms that create specialized form components for specific business processes within different workspace contexts.

Context

- Enhanced form builder system with workspace context
- Need specialized forms for specific business processes
- Must support workspace-specific workflows and validation
- Forms should integrate with business logic and data models
- Each workspace type has unique form requirements

Requirements

1. Build ClientOnboarding component:

- Client-specific onboarding workflow
- Multi-step client information collection
- Document upload and verification
- Legal and compliance forms
- Welcome and setup completion









2. Build WorkspaceSetup component:

- Workspace configuration forms
- Team member invitation
- Permission and access setup
- Branding and customization
- Integration configuration

3. Build BillingSetup component:

- Billing information collection
- Payment method setup
- Subscription and plan selection
- Tax and compliance information
- Billing preferences and settings

Specific Tasks

-  Build ClientOnboarding component
-  Add client onboarding workflow
-  Build WorkspaceSetup component
-  Add workspace configuration
-  Build BillingSetup component
-  Add billing configuration
-  Implement workflow validation
-  Add form completion tracking

Documentation Required

- Workspace-specific form architecture
- Business process form patterns
- Workflow validation implementation
- Form completion tracking
- Integration patterns
- Accessibility guidelines

Testing Requirements

- Onboarding workflow tests
- Workspace setup tests
- Billing setup tests
- Form validation tests
- Workflow completion tests
- Integration tests
- Accessibility compliance tests

Integration Points

- Integration with workspace context providers
- Business logic service integration
- Document management integration
- Payment processing integration
- Notification system integration

Deliverables

- ClientOnboarding component with workflow
- WorkspaceSetup component with configuration
- BillingSetup component with payment setup
- Workflow validation system
- Form completion tracking
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface ClientOnboardingProps {  
  client?: Partial<Client>  
  onComplete?: (client: Client) => void  
  onCancel?: () => void  
  workspaceId: string  
  template?: string  
  steps?: OnboardingStep[]  
  autoSave?: boolean  
  collaborative?: boolean  
  permissions?: string[]  
}
```

```
interface WorkspaceSetupProps {  
  workspace?: Partial<Workspace>  
  onComplete?: (workspace: Workspace) => void  
  onCancel?: () => void  
  currentUser: User  
  template?: string  
  steps?: SetupStep[]  
  autoSave?: boolean  
  integrations?: Integration[]  
  permissions?: string[]  
}
```

```
interface BillingSetupProps {  
  workspace: Workspace  
  currentBilling?: Partial<BillingInfo>  
  onComplete?: (billing: BillingInfo) => void  
  onCancel?: () => void  
  plans?: BillingPlan[]  
  paymentMethods?: PaymentMethod[]  
  steps?: BillingStep[]  
  autoSave?: boolean  
  permissions?: string[]  
}
```

```
interface OnboardingStep {  
  id: string  
  title: string  
  description?: string  
  fields: FormField[]  
  validation?: ValidationRule[]  
  documents?: DocumentRequirement[]  
}
```

```
    optional?: boolean
    permissions?: string[]
}
```

```
interface SetupStep {
    id: string
    title: string
    description?: string
    fields: FormField[]
    validation?: ValidationRule[]
    integrations?: Integration[]
    optional?: boolean
    permissions?: string[]
}
```

```
interface BillingStep {
    id: string
    title: string
    description?: string
    fields: FormField[]
    validation?: ValidationRule[]
    paymentRequired?: boolean
    optional?: boolean
    permissions?: string[]
}
```

```
interface DocumentRequirement {
    id: string
    title: string
    description?: string
    required: boolean
    types: string[]
    maxSize?: number
    validation?: (file: File) => boolean
}
```

```
interface BillingInfo {
    plan: string
    paymentMethod: PaymentMethod
    billingAddress: Address
    taxInfo?: TaxInfo
    preferences: BillingPreferences
}
```

```
interface BillingPlan {  
  id: string  
  name: string  
  price: number  
  currency: string  
  interval: 'monthly' | 'yearly'  
  features: string[]  
  limits: Record<string, number>  
}
```

Implementation Example

// ClientOnboarding implementation

```
function ClientOnboarding({ workspaceId, onComplete }) {  
  const { workspace } = useWorkspace()  
  const [currentStep, setCurrentStep] = useState(0)  
  const [formData, setFormData] = useState({})  
  
  const onboardingSteps = [  
    {  
      id: 'basic-info',  
      title: 'Basic Information',  
      fields: [  
        { name: 'companyName', type: 'text', label: 'Company Name', required: true },  
        { name: 'industry', type: 'select', label: 'Industry', required: true },  
        { name: 'size', type: 'select', label: 'Company Size', required: true }  
      ],  
    },  
    {  
      id: 'contact-info',  
      title: 'Contact Information',  
      fields: [  
        { name: 'primaryContact', type: 'text', label: 'Primary Contact', required: true },  
        { name: 'email', type: 'email', label: 'Email Address', required: true },  
        { name: 'phone', type: 'phone', label: 'Phone Number', required: true }  
      ],  
    },  
    {  
      id: 'documents',  
      title: 'Document Upload',  
      fields: [],  
      documents: [  
        { id: 'contract', title: 'Service Contract', required: true, types: ['pdf', 'docx'] },  
        { id: 'nda', title: 'Non-Disclosure Agreement', required: false, types: ['pdf'] }  
      ],  
    }  
  ],  
  
  return (  
    <FormWizard  
      steps={onboardingSteps}  
      currentStep={currentStep}  
      onStepChange={setCurrentStep}  
      onComplete={handleComplete}  
      context={workspace.type}  
    >  
  )  
}
```



```

>
<WizardContent>
  {renderCurrentStep()}
</WizardContent>
<WizardActions>
  <Button
    variant="secondary"
    onClick={previousStep}
    disabled={currentStep === 0}
  >
    Previous
  </Button>
  <Button
    variant="primary"
    onClick={nextStep}
  >
    {isLastStep ? 'Complete' : 'Next'}
  </Button>
</WizardActions>
</FormWizard>
)
}

```

Performance Requirements

FormBuilder Performance

- Form initialization under 500ms
- Field validation under 100ms
- Auto-save execution under 200ms
- Form submission under 1 second
- Memory usage under 50MB for large forms

Workspace-Specific Forms Performance

- Workflow step navigation under 300ms
- Document upload processing under 2 seconds
- Form data persistence under 500ms
- Validation feedback under 200ms
- Complete workflow execution under 5 minutes

Accessibility Requirements

WCAG 2.1 AA Compliance

- All form fields have proper labels and descriptions
- Error messages are announced to screen readers
- Keyboard navigation through all form elements
- Focus management during multi-step workflows
- Clear indication of required fields

Keyboard Navigation

- Tab navigation through all form fields
- Enter key submits forms
- Escape key cancels workflows
- Arrow keys navigate radio/checkbox groups
- Keyboard shortcuts for common actions

Security Considerations

Data Protection

- Form data encryption in transit and at rest
- Secure file upload with virus scanning
- Input sanitization and validation
- CSRF protection for form submissions
- Rate limiting for form submissions

Permission Management

- Role-based access to form fields
- Workspace-specific form permissions
- Audit logging for form submissions
- Data retention policies
- GDPR compliance features

Testing Strategy

Unit Tests

- Individual form field validation
- Conditional logic testing
- Validation rule testing
- Auto-save functionality
- State management

Integration Tests

- Multi-step workflow completion
- Form submission with backend
- File upload integration
- Permission system integration
- Real-time collaboration

E2E Tests

- Complete onboarding workflows
- Cross-browser form functionality
- Mobile form interactions
- Accessibility compliance
- Performance under load

Storybook Documentation

Form Builder Stories

- Basic form with validation
- Multi-step wizard
- Conditional fields
- Collaborative editing
- Auto-save demonstration

Workspace Forms Stories

- Client onboarding flow
- Workspace setup wizard
- Billing configuration

- Permission-based forms
- Error handling scenarios

Migration Guide

From Legacy Forms

1. Map existing form fields to new schema
2. Update validation rules to new format
3. Implement workspace context
4. Add auto-save functionality
5. Test and validate migration

Breaking Changes

- New prop interface for forms
- Validation rule format changes
- Event handler signatures updated
- State management approach
- Permission system integration