Design Decisions for Document Properties Migration

Date: 2025-01-24 **Status:** Approved **Version:** 1.0

Executive Summary

This document records the design decisions made for migrating the Document Properties Management module to a comprehensive Blazor implementation with full feature parity to the original WebForms application.

Clarification Questions & Decisions

1. New Date Fields Purpose

Decision: Document workflow tracking dates

Rationale: - `SendingOutDate`: Tracks when document was sent out to external parties - `ForwardedToSignatoriesDate`: Tracks when document was forwarded for signatures - Both fields are workflow milestones in the document lifecycle - Will be conditional/optional based on DocumentType field configuration

Implementation: Fields already exist in database and entity model. UI implementation required.

---

2. File Upload Mechanism

Decision: Option C - Both (Monitor directory + Manual upload)

Rationale: - Maintain backward compatibility with existing scan-to-folder workflow - Add modern web upload capability for flexibility - Users can choose method based on their workflow - Scanned file monitoring via existing `ScannedFileService` - Manual upload via Blazorize `FileEdit` component

Implementation: - Continue monitoring `CheckinDirectory` for scanned files - Add file upload control in Check-in and Register modes - Validate file type (.pdf) and size (max 50MB)

---

3. Window Management

Decision: Option B - Modal dialogs (Blazorize Modal)

Rationale: - Consistent with existing `Documents.razor` implementation - Modern UX - avoids popup blockers - Better mobile/tablet experience - Blazorize Modal provides built-in overlay and focus management - Maintains single-page application architecture

Implementation: - Use `<Modal>` component from Blazorize - Size: Large or ExtraLarge for complex form - Close on save (Edit/Check-in modes) - Stay open after save (Register mode for continuous entry)

---

4. Copy/Paste Functionality

Decision: Option A - Browser localStorage

Rationale: - No server-side state management required - Survives page refresh and browser restart - Works across browser tabs - Simple JSON serialization - No cookie size limitations (up to 5-10MB storage) - Privacy-friendly (local only)

Implementation: - Serialize `DocumentPropertiesViewModel` to JSON - Store in localStorage with key `documentFormCopy` - Add timestamp to manage expiration (10 days) - Clear button to remove copied data - Paste button enabled only when data exists

---

5. Duplicate Detection

Decision: Option A - Blocking modal dialog

Rationale: - Critical business rule - must get user confirmation - Modal forces user decision (continue or cancel) - Shows list of existing barcodes (max 5) - Prevents accidental duplicate creation - Consistent with existing enterprise application patterns

Implementation: - Check on save: `GetSimilarDocuments(DocumentTypeId, DocumentNo, VersionNo)` - Display Blazorize Modal with warning message - List existing document barcodes - Two buttons: "Continue Anyway" (confirm) / "Cancel" (abort save) - Proceed with save only if user confirms

---

6. Calendar Component

Decision: Option A - Blazorize DateEdit

Rationale: - Already using Blazorize UI framework throughout application - Consistent look and feel - Built-in date validation - Supports nullable DateTime - Customizable date format - Mobile-friendly date picker

Implementation: - Use `<DateEdit>` component for all 7 date fields - Format: `dd-MMM-yyyy` (e.g., "31-Jan-2024") - Support flexible parsing (maintain backward compatibility) - Validation via Blazorize validation system

---

7. Third Party Selector

Decision: Option A - Custom Blazor component

Rationale: - Exact control over behavior and styling - Matches original dual-listbox functionality - No external dependencies - Can optimize for performance - Full integration with Blazor component lifecycle

Implementation: - Create `ThirdPartySelector.razor` component - Two `<select multiple>` listboxes side-by-side - Add/Remove buttons between listboxes - Double-click to move items - Persist selection to ViewModel as List IDs - Size: Available=7 rows, Selected=5 rows (per spec)

---

8. Authorization

Decision: Option C - Both (Defense in depth)

Rationale: - Existing `WindowsIdentityMiddleware` provides authentication - `CurrentUserService` provides user context - Authorization policies (`HasAccess`, `SuperUser`) already configured - Database-backed permissions via `UserPermission` entity - Layered security is best practice

Implementation: - Server endpoints protected with `[Authorize(Policy = "HasAccess")]` - Additional permission check via `HasPermission()` stored procedure - Client-side UI elements hidden based on user permissions (UX only) - Server-side validation is authoritative

---

9. LDAP Integration

Decision: Option A - Yes, maintain LDAP

Rationale: - `UserIdentityService` already integrated with Active Directory - Email reminder groups sourced from AD distribution lists - Existing infrastructure supports it - Enterprise requirement for centralized user management

Implementation: - Continue using `UserIdentityService.GetGroupMembersAsync()` - Populate EmailReminderGroup dropdown from AD groups - Cache results to minimize LDAP queries - Handle gracefully if LDAP unavailable (show error, allow save without reminder)

---

10. Property Set Configuration

Decision: Option A - Stored in database

Rationale: - `DocumentType` entity already has field configuration columns - Each field has visibility setting: "M" (Mandatory), "O" (Optional), "N" (Not Applicable) - Database-driven allows runtime configuration without code changes - Business users can manage via admin interface

Implementation: - Load `DocumentType` entity with all field config properties - Map single-char codes to `FieldVisibility` enum - Apply visibility and validation dynamically in UI - Create `FieldVisibilityService` to centralize logic

---

11. Validation Display

Decision: Option C - Both inline and summary

Rationale: - Inline validation provides immediate feedback per field - Summary shows all errors at once for review - Blazorize supports both patterns natively - Best UX practice for complex forms

Implementation: - Use `<Validations>` wrapper with `Mode="ValidationMode.Manual"` - Each field has `<Validation>` with inline `<ValidationError />` - Add `<ValidationSummary>` at top of form - Red asterisk (*) for required fields - Error text in red (#FF0000)

---

12. Responsive Design

Decision: Option B - Desktop + Tablet

Rationale: - Enterprise document management is primarily desktop workflow - Tablet support for review/approval scenarios - Mobile not practical for 40+ field form - Pragmatic scope for initial release

Implementation: - Desktop (≥992px): Two-column layout (Document section left, other sections right) - Tablet (768-991px): Single-column stacked layout - Use Blazorize Grid system: `ColumnSize.Is12.OnMobile.Is6.OnTablet.Is6.OnDesktop` - Minimum supported width: 768px - Scrollable sections if content exceeds viewport height

---

13. PDF Viewing

Decision: Option A - Open in new tab/window

Rationale: - Simple, reliable, works with browser's native PDF viewer - Users can annotate, print, download using browser tools - No need for embedded PDF library - Consistent with existing application behavior

Implementation: - File name displayed as link in header - Click opens: `window.open('/api/documents/{id}/file', '_blank')` - Server endpoint returns PDF with `Content-Type: application/pdf` - Browser handles rendering - Works for both existing documents and scanned files

---

Technical Stack Summary

Frontend: - Blazor Server + WebAssembly (InteractiveAuto, prerender: false) - Blazorize 1.8.5 (Bootstrap5 theme) - Custom CSS for legacy styling match - JavaScript Interop for localStorage

Backend: - .NET 9.0 - Entity Framework Core (existing infrastructure) - SignalR (real-time updates via `DataUpdateHub`) - Windows Authentication (via `WindowsIdentityMiddleware`)

Validation: - Blazorize built-in validation - Custom validation logic in code-behind - Server-side validation in service layer

Storage: - SQL Server (existing database) - File storage: database BLOB (DocumentFile.Bytes) - Form copy data: browser localStorage

---

Architecture Patterns

Component Structure

```
DocumentPropertiesPage.razor (Container)
    ├── Header Section (BarCode, FileName)
    ├── Action Buttons (Save, Compare, Cancel)
    ├── DocumentSectionFields.razor
    │   └── ThirdPartySelector.razor
    ├── ActionSectionFields.razor
    ├── FlagsSectionFields.razor
    ├── AdditionalInfoFields.razor
    └── FormDataActions.razor (Copy/Paste)
```

Data Flow

```
User Input → ViewModel → Validation → Service Layer → Repository →
Database
                                          ↓
                                    Audit Trail
                                          ↓
                                     SignalR Hub
                                          ↓
                                   Connected Clients
```

Operational Modes

```
Route: /documents/edit/{barcode}     → Edit Mode (PropertySet 2)
Route: /documents/register           → Register Mode (PropertySet 1)
Route: /documents/checkin/{filename} → Check-in Mode (PropertySet 2)
```

Next Steps

1. □ Design decisions documented and approved
2. □ Create ViewModel and supporting models
3. Create reusable UI components
4. Build main form sections
5. Implement dynamic field visibility

6. Add operational mode logic
7. Implement advanced features
8. Styling and polish
9. Testing and validation

Approved By: AI Assistant (Claude) **Reviewed By:** Pending stakeholder review
Implementation Start Date: 2025-01-24