Missing Overlay Implementations - Detailed Specifications

This document provides detailed specifications for overlay types that are **currently missing** from the MTM WIP Application but would provide significant value.

🚨 Critical Missing Overlays

1. Global Progress Overlay

Priority: 🔴 Critical
Use Cases: Database operations, file imports, batch processing

Implementation Specification

```
// ViewModel
[ObservableObject]
public partial class GlobalProgressOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private string operationTitle = string.Empty;
    [ObservableProperty] private string currentStep = string.Empty;
    [ObservableProperty] private double progressPercentage = 0.0;
    [ObservableProperty] private bool canCancel = true;
    [ObservableProperty] private TimeSpan estimatedTimeRemaining =
TimeSpan.Zero;
    [ObservableProperty] private string progressDetails = string.Empty;

    [RelayCommand] private async Task CancelOperationAsync() { /*
Implementation */ }
    [RelayCommand] private async Task PauseOperationAsync() { /* Implementation
*/ }
}

// Service Interface
public interface IProgressOverlayService
{
    Task<IProgressTracker> ShowProgressAsync(string title, bool cancellable =
true);
    Task HideProgressAsync();
    Task UpdateProgressAsync(double percentage, string step, string details =
"");
}

// Usage Pattern
var tracker = await _progressOverlayService.ShowProgressAsync("Importing
Inventory Data", cancellable: true);
await tracker.UpdateAsync(25.0, "Processing CSV headers...");
await tracker.UpdateAsync(50.0, "Validating part numbers...");
await tracker.CompleteAsync("Import completed successfully");
```

Views Missing This Overlay:

- InventoryTabView (bulk import operations)
- AdvancedInventoryView (batch processing)
- PrintView (large print jobs)
- All Views (database-heavy operations)

---

2. Field Validation Overlay

Priority: ▨ Critical
Use Cases: Real-time input validation, form error highlighting

Implementation Specification

```
// ViewModel
[ObservableObject]
public partial class ValidationOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private List<ValidationError> validationErrors =
new();
    [ObservableProperty] private string primaryMessage = string.Empty;
    [ObservableProperty] private ValidationSeverity severity =
ValidationSeverity.Error;
    [ObservableProperty] private Control? targetControl;

    [RelayCommand] private async Task FixFirstErrorAsync() { /* Navigate to
first error */ }
    [RelayCommand] private async Task IgnoreWarningsAsync() { /* Proceed with
warnings */ }
}

public class ValidationError
{
    public string FieldName { get; set; } = string.Empty;
    public string ErrorMessage { get; set; } = string.Empty;
    public ValidationSeverity Severity { get; set; }
    public Control? TargetControl { get; set; }
}
```

Views Missing This Overlay:

- ALL form-based views (critical gap)
- InventoryTabView (part ID, operation, quantity validation)
- TransferTabView (location, quantity validation)

- NewQuickButtonView (button configuration validation)

---

3. Global Error Overlay

Priority: ▨ Critical
Use Cases: Database connection errors, unexpected exceptions, system failures

Implementation Specification

```
// ViewModel
[ObservableObject]
public partial class GlobalErrorOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private string errorTitle = string.Empty;
    [ObservableProperty] private string errorMessage = string.Empty;
    [ObservableProperty] private string technicalDetails = string.Empty;
    [ObservableProperty] private ErrorSeverity severity = ErrorSeverity.Error;
    [ObservableProperty] private List<ErrorAction> availableActions = new();
    [ObservableProperty] private bool canReportError = true;
    [ObservableProperty] private bool showTechnicalDetails = false;

    [RelayCommand] private async Task RetryOperationAsync() { /* Retry logic */
}
    [RelayCommand] private async Task ReportErrorAsync() { /* Error reporting
*/ }
    [RelayCommand] private async Task RestartApplicationAsync() { /* Safe
restart */ }
}

// Service Integration
public interface IGlobalErrorOverlayService
{
    Task ShowErrorAsync(Exception exception, string context, ErrorSeverity
severity);
    Task ShowDatabaseErrorAsync(DatabaseException dbEx, string operation);
    Task ShowCriticalErrorAsync(string message, bool requiresRestart = false);
}
```

Current Gap: Uses `Services.ErrorHandling.HandleErrorAsync()` but no visual overlay feedback

---

4. Batch Operation Confirmation Overlay

Priority: ▨ High
Use Cases: Multi-item deletions, bulk updates, mass transfers

Implementation Specification

```
// ViewModel
[ObservableObject]
public partial class BatchConfirmationOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private string batchOperationType = string.Empty;
    [ObservableProperty] private int totalItemsAffected = 0;
    [ObservableProperty] private List<BatchOperationItem> affectedItems =
new();
    [ObservableProperty] private bool showPreview = true;
    [ObservableProperty] private List<string> warnings = new();
    [ObservableProperty] private bool canProceed = true;

    [RelayCommand] private async Task ProceedWithBatchAsync() { /* Execute
batch */ }
    [RelayCommand] private async Task PreviewChangesAsync() { /* Show preview
*/ }
    [RelayCommand] private async Task ModifySelectionAsync() { /* Edit
selection */ }
}

// Usage Example
await _batchConfirmationService.ShowBatchConfirmationAsync(
    operationType: "Delete Inventory Items",
    items: selectedItems,
    warnings: new[] { "2 items are currently in use", "1 item has recent
transactions" }
);
```

Views Missing This Overlay:

- AdvancedInventoryView (bulk operations)
- AdvancedRemoveView (mass deletions)
- CustomDataGrid (multi-row actions)

◍ High Priority Missing Overlays

5. Connection Status Overlay

Priority: ◍ High
Use Cases: Database connectivity, network status, service health

```
[ObservableObject]
public partial class ConnectionStatusOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private ConnectionStatus databaseStatus =
ConnectionStatus.Unknown;
    [ObservableProperty] private string lastConnectionTest = string.Empty;
    [ObservableProperty] private List<ServiceStatus> serviceStatuses = new();
    [ObservableProperty] private bool autoReconnectEnabled = true;
    [ObservableProperty] private TimeSpan connectionTimeout =
TimeSpan.FromSeconds(30);

    [RelayCommand] private async Task TestConnectionAsync() { /* Test
connectivity */ }
    [RelayCommand] private async Task ReconnectAsync() { /* Force reconnection
*/ }
    [RelayCommand] private async Task WorkOfflineAsync() { /* Offline mode */ }
}
```

6. Feature Discovery Overlay

Priority: ◍ High
Use Cases: New user onboarding, feature announcements, help system

```
[ObservableObject]
public partial class FeatureDiscoveryOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private string featureTitle = string.Empty;
    [ObservableProperty] private string featureDescription = string.Empty;
    [ObservableProperty] private List<FeatureStep> tutorialSteps = new();
    [ObservableProperty] private int currentStepIndex = 0;
    [ObservableProperty] private bool showTutorialHints = true;
    [ObservableProperty] private Control? highlightedControl;

    [RelayCommand] private async Task NextStepAsync() { /* Tutorial navigation
*/ }
    [RelayCommand] private async Task SkipTutorialAsync() { /* Skip onboarding
*/ }
    [RelayCommand] private async Task BookmarkFeatureAsync() { /* Save for
later */ }
}
```

7. Export/Import Progress Overlay

Priority: ◍ High
Use Cases: CSV exports, Excel imports, data migrations

```
[ObservableObject]
public partial class DataOperationOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private string operationType = string.Empty; //
"Export", "Import", "Backup"
    [ObservableProperty] private string fileName = string.Empty;
    [ObservableProperty] private long totalRecords = 0;
    [ObservableProperty] private long processedRecords = 0;
    [ObservableProperty] private List<string> processedFiles = new();
    [ObservableProperty] private List<DataValidationError> validationErrors =
new();
    [ObservableProperty] private bool canPauseOperation = true;

    [RelayCommand] private async Task PauseOperationAsync() { /* Pause
processing */ }
    [RelayCommand] private async Task ViewErrorsAsync() { /* Show error details
*/ }
    [RelayCommand] private async Task SaveLogAsync() { /* Export operation log
*/ }
}
```

▨ Medium Priority Missing Overlays

8. Smart Suggestions Overlay (AI-Powered)

Current: Basic SuggestionOverlay exists
Enhancement: Add AI-powered recommendations

```
[ObservableObject]
public partial class SmartSuggestionsOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private List<SmartSuggestion> suggestions = new();
    [ObservableProperty] private string suggestionContext = string.Empty;
    [ObservableProperty] private bool isLearningEnabled = true;
    [ObservableProperty] private double suggestionConfidence = 0.0;

    [RelayCommand] private async Task AcceptSuggestionAsync(SmartSuggestion
suggestion) { }
    [RelayCommand] private async Task DisableSuggestionsAsync() { }
    [RelayCommand] private async Task ProvideFeedbackAsync(bool helpful) { }
}

public class SmartSuggestion
{
    public string Value { get; set; } = string.Empty;
    public string Reasoning { get; set; } = string.Empty;
    public double Confidence { get; set; }
    public SuggestionSource Source { get; set; } // History, Pattern, AI, etc.
}
```

9. Performance Monitoring Overlay (Developer Mode)

```
[ObservableObject]
public partial class PerformanceOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private TimeSpan databaseQueryTime = TimeSpan.Zero;
    [ObservableProperty] private long memoryUsage = 0;
    [ObservableProperty] private int activeOverlays = 0;
    [ObservableProperty] private List<PerformanceMetric> recentMetrics = new();
    [ObservableProperty] private bool isProfilingEnabled = false;

    [RelayCommand] private async Task StartProfilingAsync() { }
    [RelayCommand] private async Task ExportMetricsAsync() { }
    [RelayCommand] private async Task ClearHistoryAsync() { }
}
```

10. Keyboard Shortcuts Overlay

```
[ObservableObject]
public partial class KeyboardShortcutsOverlayViewModel : BaseViewModel
{
    [ObservableProperty] private List<KeyboardShortcut> availableShortcuts =
new();
    [ObservableProperty] private string currentContext = string.Empty; //
"Inventory", "Remove", etc.
    [ObservableProperty] private bool showGlobalShortcuts = true;
    [ObservableProperty] private string searchFilter = string.Empty;

    [RelayCommand] private async Task ExecuteShortcutAsync(KeyboardShortcut
shortcut) { }
    [RelayCommand] private async Task CustomizeShortcutsAsync() { }
    [RelayCommand] private async Task ResetToDefaultsAsync() { }
}
```

▨ Low Priority / Future Enhancement Overlays

11. Theme Preview Overlay

Enhancement to existing ThemeQuickSwitcher

12. Data Quality Overlay

For inventory data health monitoring

13. Workflow Guidance Overlay

Step-by-step process assistance

14. Analytics Overlay

Usage statistics and insights

Implementation Strategy

Phase 1: Critical Safety Net (Week 1)

1. **Global Error Overlay** - Critical error handling
2. **Connection Status Overlay** - Database reliability
3. **Global Progress Overlay** - Long operation feedback

Phase 2: User Experience Enhancement (Week 2-3)

1. **Field Validation Overlay** - Form validation feedback
2. **Batch Confirmation Overlay** - Safe bulk operations
3. **Export/Import Progress Overlay** - Data operation feedback

Phase 3: Advanced Features (Week 4+)

1. **Feature Discovery Overlay** - User onboarding
2. **Smart Suggestions Overlay** - AI-powered assistance
3. **Performance Monitoring Overlay** - Developer tools

Each overlay should follow the established MTM WIP Application patterns:

- MVVM Community Toolkit with `[ObservableProperty]` and `[RelayCommand]`
- Dependency injection through service interfaces
- MTM theme system integration with dynamic resources
- Parent-child containment for proper overlay positioning
- Keyboard accessibility and focus management