# Orders Exercise Requirements Checklist

## Advanced .NET Exercise: Order Management API

---

## Module 1: Advanced AutoMapper Patterns for Orders (20 minutes)

### Task 1.1: Create Advanced Order Profile (8 minutes)

#### Step 1: Enhanced Entities and DTOs

**Requirements:**

- [ ] Create `OrderCategory` enum with values: Fiction (0), NonFiction (1), Technical (2), Children (3)
- [ ] Update `Order` entity to include:
  - [ ] `string Title` property
  - [ ] `string Author` property
  - [ ] `string ISBN` property
  - [ ] `OrderCategory Category` property
  - [ ] `decimal Price` property
  - [ ] `DateTime PublishedDate` property
  - [ ] `string? CoverImageUrl` property (optional)
  - [ ] `bool IsAvailable` property (default based on stock)
  - [ ] `int StockQuantity` property (default 0)
- [ ] Create `OrderProfileDto` with properties:
  - [ ] `Guid Id`
  - [ ] `string Title`
  - [ ] `string Author`
  - [ ] `string ISBN`
  - [ ] `string CategoryDisplayName`
  - [ ] `decimal Price`
  - [ ] `string FormattedPrice`
  - [ ] `DateTime PublishedDate`
  - [ ] `DateTime CreatedAt`
  - [ ] `string? CoverImageUrl`
  - [ ] `bool IsAvailable`
  - [ ] `int StockQuantity`

    - [ ] `string PublishedAge`
    - [ ] `string AuthorInitials`
    - [ ] `string AvailabilityStatus`
- [ ] Create `CreateOrderProfileRequest` with properties:
    - [ ] `string Title`
    - [ ] `string Author`
    - [ ] `string ISBN`
    - [ ] `OrderCategory Category`
    - [ ] `decimal Price`
    - [ ] `DateTime PublishedDate`
    - [ ] `string? CoverImageUrl` (optional)
    - [ ] `int StockQuantity` (default 1)

**Step 2: Advanced Order Mapping Profile**

**Requirements:**

- [ ] Create `AdvancedOrderMappingProfile` class inheriting from `Profile`
- [ ] Map `CreateOrderProfileRequest` to `Order` with:
    - [ ] Auto-generated `Id` (Guid.NewGuid())
    - [ ] Auto-generated `CreatedAt` (DateTime.UtcNow)
    - [ ] Auto-set `IsAvailable` based on StockQuantity > 0
    - [ ] Ignore `UpdatedAt`
- [ ] Map `Order` to `OrderProfileDto` using custom resolvers:
    - [ ] `CategoryDisplayResolver` for category display names
    - [ ] `PriceFormatterResolver` for currency formatting
    - [ ] `PublishedAgeResolver` for human-readable published age
    - [ ] `AuthorInitialsResolver` for author initials
    - [ ] `AvailabilityStatusResolver` for stock status

**Custom Value Resolvers Required:**

- [ ] **CategoryDisplayResolver**:
    - Fiction → "Fiction & Literature"
    - NonFiction → "Non-Fiction"
    - Technical → "Technical & Professional"
    - Children → "Children's Orders"
    - Default → "Uncategorized"
- [ ] **PriceFormatterResolver**:
    - Format as currency using ToString("C2")
- [ ] **PublishedAgeResolver**:
    - < 30 days → "New Release"
    - < 365 days → "X months old"
    - < 1825 days (5 years) → "X years old"

    - = 1825 days → "Classic"

- [ ] **AuthorInitialsResolver**:
    - Two+ names → First letter of first and last names (uppercase)
    - Single name → First letter (uppercase)
    - No name → "?"
- [ ] **AvailabilityStatusResolver**:
    - Not available → "Out of Stock"
    - Available, 0 stock → "Unavailable"
    - Available, 1 stock → "Last Copy"
    - Available, ≤5 stock → "Limited Stock"
    - Available, >5 stock → "In Stock"

### Task 1.2: Conditional Mapping Based on Order Category (7 minutes)

**Requirements:**

- [ ] Conditional `CoverImageUrl` mapping:
    - [ ] Return actual URL for Fiction, NonFiction, Technical categories
    - [ ] Return null for Children category (content filtering)
- [ ] Conditional `Price` mapping:
    - [ ] Apply 10% discount for Children category (Price * 0.9m)
    - [ ] Return actual price for all other categories

### Task 1.3: Update CreateOrderHandler (5 minutes)

**Requirements:**

- [ ] Update method signature to accept `CreateOrderProfileRequest`
- [ ] Update logging to include Title, Author, Category, ISBN information
- [ ] Check ISBN uniqueness instead of email uniqueness
- [ ] Use advanced mapping for Order creation
- [ ] Return `OrderProfileDto` instead of basic response
- [ ] Update cache key to "all_orders"
- [ ] Maintain existing functionality (validation, caching, exception handling)

---

## Module 2: Structured Logging & Telemetry for Orders (20 minutes)

### Task 2.1: Create Order-Specific Logging Infrastructure (8 minutes)

**LogEvents Constants Required:**

- [ ] `OrderCreationStarted = 2001`
- [ ] `OrderValidationFailed = 2002`
- [ ] `OrderCreationCompleted = 2003`
- [ ] `DatabaseOperationStarted = 2004`
- [ ] `DatabaseOperationCompleted = 2005`
- [ ] `CacheOperationPerformed = 2006`
- [ ] `ISBNValidationPerformed = 2007`
- [ ] `StockValidationPerformed = 2008`

**OrderCreationMetrics Record Required:**

- [ ] `string OperationId`
- [ ] `string OrderTitle`
- [ ] `string ISBN`
- [ ] `OrderCategory Category`
- [ ] `TimeSpan ValidationDuration`
- [ ] `TimeSpan DatabaseSaveDuration`
- [ ] `TimeSpan TotalDuration`
- [ ] `bool Success`
- [ ] `string? ErrorReason` (optional)

**LoggingExtensions Required:**

- [ ] Update `LogOrderCreationMetrics()` method:
  - [ ] Log order-specific metrics with event ID
  - [ ] Include Title, ISBN, Category in log message
  - [ ] Include all timing measurements in milliseconds
  - [ ] Include success/failure status

## Task 2.2: Implement Performance Tracking for Orders (7 minutes)

**Requirements:**

- [ ] Track operation start time for order creation
- [ ] Generate unique operation ID (8 characters)
- [ ] Use logging scope for entire order operation
- [ ] Log operation start with Title, Author, ISBN, Category details
- [ ] Add ISBN validation logging with ISBNValidationPerformed event
- [ ] Add stock validation logging with StockValidationPerformed event
- [ ] Time validation phase separately (ISBN uniqueness check)
- [ ] Log validation failures with order-specific details
- [ ] Time database operations separately
- [ ] Log database operation start and completion with OrderId
- [ ] Log cache operations with "all_orders" cache key
- [ ] Calculate total operation duration
- [ ] Log comprehensive OrderCreationMetrics for success cases

- [ ] Log error metrics in catch block with order details
- [ ] Re-throw exceptions for global handler

### Task 2.3: Correlation ID Middleware (5 minutes)

**Requirements:**

- [ ] Use same CorrelationMiddleware as user exercise
- [ ] Ensure correlation IDs flow through order operations
- [ ] Test correlation with order-specific logging

---

# Module 3: Complex Order Validation Scenarios (25 minutes)

### Task 3.1: Create Advanced Order Validation Infrastructure (10 minutes)

**CreateOrderProfileValidator Requirements:**

- [ ] Inject `ApplicationContext` and `ILogger<CreateOrderProfileValidator>`
- [ ] **Title Validation Rules**:
  - [ ] Not empty with custom message
  - [ ] Minimum 1 character, maximum 200 characters
  - [ ] Must not contain inappropriate content (validate against word list)
  - [ ] Must be unique for the same author (async database check)
- [ ] **Author Validation Rules**:
  - [ ] Not empty with custom message
  - [ ] Minimum 2 characters, maximum 100 characters
  - [ ] Must contain only valid characters (letters, spaces, hyphens, apostrophes, dots)
- [ ] **ISBN Validation Rules**:
  - [ ] Not empty with custom message
  - [ ] Must be valid ISBN format (10 or 13 digits, may contain hyphens)
  - [ ] Must be unique in system (async database check)
- [ ] **Category Validation Rules**:
  - [ ] Must be valid enum value
- [ ] **Price Validation Rules**:
  - [ ] Must be greater than 0
  - [ ] Must be less than $10,000
- [ ] **PublishedDate Validation Rules**:
  - [ ] Cannot be in the future
  - [ ] Cannot be before year 1400
- [ ] **StockQuantity Validation Rules**:
  - [ ] Cannot be negative
  - [ ] Cannot exceed 100,000 (reasonableness check)

- [ ] **CoverImageUrl Validation Rules**:
  - [ ] Must be valid image URL when provided
  - [ ] Must be HTTP/HTTPS protocol
  - [ ] Must end with image extensions (.jpg, .jpeg, .png, .gif, .webp)
- [ ] **Business Rules Validation**:
  - [ ] Must pass all complex business rules (async)

**Validation Methods Required:**

- [ ] `BeValidTitle()`: Check against inappropriate words list
- [ ] `BeUniqueTitle()`: Async database check for Title+author combination with logging
- [ ] `BeValidAuthorName()`: Regex validation for allowed characters
- [ ] `BeValidISBN()`: Format validation (10 or 13 digits)
- [ ] `BeUniqueISBN()`: Async database check with logging
- [ ] `BeValidImageUrl()`: URL and image extension validation
- [ ] `PassBusinessRules()`: Complex async business rule validation

**Business Rules Required:**

- [ ] **Rule 1**: Daily order addition limit check (max 500 per day)
- [ ] **Rule 2**: Technical orders minimum price check ($20.00)
- [ ] **Rule 3**: Children's order content restrictions (check Title against restricted words)
- [ ] **Rule 4**: High-value order stock limit (>$500 = max 10 stock)
- [ ] All rules must include appropriate logging

## Task 3.2: Create Custom Order Validation Attributes (8 minutes)

**ValidISBNAttribute Requirements:**

- [ ] Inherit from `ValidationAttribute` and implement `IClientModelValidator`
- [ ] Implement `IsValid()` method to validate ISBN format (10 or 13 digits)
- [ ] Remove hyphens and spaces before validation
- [ ] Implement `AddValidation()` for client-side validation
- [ ] Add data attributes for client ISBN validation

**OrderCategoryAttribute Requirements:**

- [ ] Inherit from `ValidationAttribute`
- [ ] Accept allowed categories in constructor
- [ ] Generate error message with allowed categories list
- [ ] Implement `IsValid()` method to check category against allowed list

**PriceRangeAttribute Requirements:**

- [ ] Inherit from `ValidationAttribute`
- [ ] Accept min and max price in constructor (as double, convert to decimal)

- [ ] Generate error message with currency formatting
- [ ] Implement `IsValid()` method for price range validation

### Task 3.3: Implement Conditional Order Validation (7 minutes)

**Conditional Validation Requirements:**

- [ ] **Technical Order Conditions**:
  - [ ] Price minimum $20.00 (stricter than base rule)
  - [ ] Must contain technical keywords in Title
  - [ ] Must be published within last 5 years (cross-field validation)
- [ ] **Children's Order Conditions**:
  - [ ] Price maximum $50.00
  - [ ] Title must be appropriate for children (no inappropriate words)
- [ ] **Fiction Order Conditions**:
  - [ ] Author name minimum 5 characters (full name requirement)
- [ ] **Cross-Field Validation**:
  - [ ] Expensive orders (>$100) must have limited stock (≤20 units)
  - [ ] Technical orders must be recent (published within 5 years)

**Helper Methods Required:**

- [ ] `ContainTechnicalKeywords()`: Check Title against technical keywords list
- [ ] `BeAppropriateForChildren()`: Check Title against inappropriate words for children

---

## Module 4: Integration & Testing for Orders (15 minutes)

### Task 4.1: Update Program.cs Configuration (5 minutes)

**Requirements:**

- [ ] Register both AutoMapper profiles (`OrderMappingProfile`, `AdvancedOrderMappingProfile`)
- [ ] Register `CreateOrderProfileValidator` as scoped service
- [ ] Register all validators from assembly containing `CreateOrderProfileValidator`
- [ ] Add `CorrelationMiddleware` to pipeline
- [ ] Update endpoint mapping to `/orders` with order-specific documentation

### Task 4.2: Create Order Integration Tests (10 minutes)

**Test Class Requirements:**

- [ ] Name: `CreateOrderHandlerIntegrationTests`

- [ ] Implement `IDisposable`
- [ ] Set up in-memory database with unique name
- [ ] Configure AutoMapper with both order profiles
- [ ] Set up memory cache
- [ ] Mock `ILogger<CreateOrderHandler>`
- [ ] Create handler instance with all dependencies

**Test Methods Required:**

- [ ] **Test 1**: `Handle_ValidTechnicalOrderRequest_CreatesOrderWithCorrectMappings`
  - [ ] Arrange: Create valid Technical order request with all properties
  - [ ] Act: Call handler
  - [ ] Assert: Verify Created result type
  - [ ] Assert: Check CategoryDisplayName = "Technical & Professional"
  - [ ] Assert: Check AuthorInitials for two-name author
  - [ ] Assert: Check PublishedAge calculation
  - [ ] Assert: Check FormattedPrice starts with currency symbol
  - [ ] Assert: Check AvailabilityStatus based on stock
  - [ ] Assert: Verify OrderCreationStarted log called once
- [ ] **Test 2**: `Handle_DuplicateISBN_ThrowsValidationExceptionWithLogging`
  - [ ] Arrange: Create existing order in database with specific ISBN
  - [ ] Arrange: Create request with same ISBN
  - [ ] Act & Assert: Verify ValidationException thrown
  - [ ] Assert: Check exception message contains "already exists"
  - [ ] Assert: Verify OrderValidationFailed log called once
- [ ] **Test 3**: `Handle_ChildrensOrderRequest_AppliesDiscountAndConditionalMapping`
  - [ ] Arrange: Create valid Children's order request
  - [ ] Act: Call handler
  - [ ] Assert: Check CategoryDisplayName = "Children's Orders"
  - [ ] Assert: Check Price has 10% discount applied
  - [ ] Assert: Check CoverImageUrl is null (content filtering)

**Test Infrastructure Requirements:**

- [ ] Proper disposal of context and cache
- [ ] Unique database name per test run
- [ ] Proper exception handling in tests
- [ ] Mock verification for logging calls with order-specific events

---

# Assessment & Deliverables Requirements

## Expected Outputs (All Required):

- [ ] **Advanced AutoMapper Profile** with custom resolvers and conditional logic for orders
- [ ] **Enhanced CreateOrderHandler** with comprehensive logging and performance tracking for order operations
- [ ] **Complex FluentValidation** with async rules and order-specific business logic
- [ ] **Correlation Middleware** for request tracking
- [ ] **Integration Tests** demonstrating all order functionality

## Assessment Criteria (Total: 100 points):

**AutoMapper (25 points):**

- [ ] **10 points**: Custom value resolvers implemented correctly
    - CategoryDisplayResolver with order-specific names
    - PriceFormatterResolver with currency formatting
    - PublishedAgeResolver with order age logic
    - AuthorInitialsResolver with proper name parsing
    - AvailabilityStatusResolver with stock-based status
- [ ] **10 points**: Conditional mapping logic works
    - Category-based CoverImageUrl filtering for children's orders
    - Category-based Price discount for children's orders
    - Mappings produce expected results for all categories
- [ ] **5 points**: Clean, maintainable mapping configuration
    - Well-organized order mapping profile classes
    - Clear method names and order-specific structure
    - Proper separation of concerns

**Logging (25 points):**

- [ ] **10 points**: Structured logging with correlation IDs
    - CorrelationMiddleware properly implemented
    - Order operation logging scopes created correctly
    - Correlation IDs flow through order requests
- [ ] **10 points**: Performance metrics captured
    - All order timing measurements implemented
    - OrderCreationMetrics properly populated with order data
    - Metrics logged in structured format with ISBN, Title, Category
- [ ] **5 points**: Appropriate log levels and events
    - Correct LogLevel usage for order operations
    - Proper order-specific EventId constants used (2001-2008)
    - Meaningful log messages with order context

**Validation (30 points):**

- [ ] **15 points**: Complex async validation rules
    - ISBN uniqueness checks working
    - Title+Author uniqueness validation implemented

    - Order business rules validation implemented
    - Proper async/await usage with order context
    - Cancellation token support
- [ ] **10 points**: Order-specific business rule validation
    - Daily order addition limit check
    - Category-specific price rules (Technical $20+, Children $50-)
    - Content filtering for children's orders
    - High-value order stock constraints
    - Technical order recency requirements
    - Proper error handling and logging
- [ ] **5 points**: Custom validation attributes
    - ValidISBNAttribute working with 10/13 digit validation
    - OrderCategoryAttribute working
    - PriceRangeAttribute working with currency
    - Client-side validation support

**Integration (20 points):**

- [ ] **10 points**: All components work together
    - Program.cs properly configured for order management
    - All order services registered correctly
    - Middleware pipeline correct for order operations
    - No runtime errors in order workflows
- [ ] **10 points**: Comprehensive test coverage
    - All three test methods pass
    - Proper mock verification with order-specific events
    - Order edge cases covered (categories, pricing, stock)
    - Clean test setup and disposal

## Performance Requirements:

- [ ] **Sub-100ms average creation time** for successful order creation requests
- [ ] **Proper memory management** (no leaks in order tests)
- [ ] **Efficient database queries** (single query for ISBN uniqueness checks)
- [ ] **Appropriate caching** (order cache invalidation working)

## Code Quality Requirements:

- [ ] **All code files properly organized** for order management structure
- [ ] **Consistent naming conventions** throughout order domain
- [ ] **Proper error handling** (no unhandled exceptions in order operations)
- [ ] **Clean separation of concerns** (each order class has single responsibility)
- [ ] **Comprehensive XML documentation** on public order methods
- [ ] **No compiler warnings** or errors in order code

## Bonus Challenges (Optional, +10 points each):

- [ ] **Category-based Caching**: Implement separate cache keys for different order categories
    - Different cache keys for Fiction, Technical, etc.
    - Category-specific cache invalidation strategy
    - Performance improvement measurable
- [ ] **Order Metrics Dashboard**: Create endpoint to expose order creation and inventory metrics
    - New endpoint returning order aggregated metrics
    - Real-time order performance data
    - Proper DTO for order metrics response
- [ ] **Multi-language Order Support**: Support order Titles and descriptions in multiple languages
    - Resource files for order metadata in multiple languages
    - Culture-specific order information
    - Proper fallback to default language
- [ ] **Inventory Optimization**: Implement batch order creation with stock management
    - Accept array of order requests
    - Batch order database operations
    - Transaction management for order inventory
    - Parallel processing where appropriate

### Time Management Requirements:

- [ ] **Module 1 completed within 20 minutes** (order mapping)
- [ ] **Module 2 completed within 20 minutes** (order logging)
- [ ] **Module 3 completed within 25 minutes** (order validation)
- [ ] **Module 4 completed within 15 minutes** (order integration)
- [ ] **Total exercise completed within 80 minutes**

### Success Metrics:

- [ ] **All order tests pass** without modification
- [ ] **No runtime exceptions** during order operations
- [ ] **Logging produces readable, structured output** for order workflows
- [ ] **AutoMapper mappings work correctly** for all order scenarios
- [ ] **Validation prevents invalid order data** from being saved
- [ ] **Performance metrics show** order operation timing breakdown
- [ ] **Correlation IDs appear** in all related order log entries

---

# Final Checklist Before Submission:

### Code Files Required:

- [ ] `Features/Orders/OrderCategory.cs`
- [ ] `Features/Orders/Order.cs` (updated with order properties)

- [ ] `Features/Orders/DTOs/AdvancedOrderDtos.cs`
- [ ] `Common/Mapping/AdvancedOrderMappingProfile.cs`
- [ ] `Common/Logging/LoggingModels.cs` (updated for orders)
- [ ] `Common/Logging/LoggingExtensions.cs` (updated for orders)
- [ ] `Common/Middleware/CorrelationMiddleware.cs`
- [ ] `Validators/CreateOrderProfileValidator.cs`
- [ ] `Validators/Attributes/ValidISBNAttribute.cs`
- [ ] `Validators/Attributes/OrderCategoryAttribute.cs`
- [ ] `Validators/Attributes/PriceRangeAttribute.cs`
- [ ] `Features/Orders/CreateOrderHandler.cs` (updated for orders)
- [ ] `Program.cs` (updated for order endpoints)
- [ ] `Tests/CreateOrderHandlerIntegrationTests.cs`

### Verification Steps:

- [ ] **Build succeeds** without errors or warnings
- [ ] **All order tests pass** in test runner
- [ ] **API starts successfully** and responds to order requests
- [ ] **Swagger documentation** shows updated order endpoints
- [ ] **Order logs appear in console** with proper formatting
- [ ] **Order database operations work** (create, check ISBN uniqueness)
- [ ] **Order caching behavior verified** (check cache invalidation)
- [ ] **Exception handling working** for order operations
- [ ] **Order category mapping works** for all categories
- [ ] **ISBN validation works** for both 10 and 13 digit formats