FIRAT UNIVERSITY

FACULTY OF TECHNOLOGY

Software Engineering Department

YMH219 – OBJECT ORIENTED PROGRAMMING
Project Application and Documentation

OTOTECHSTIL – Textile Automation

Developer
230541039 Engin SAÇAN

JANUARY – 2026

OTOTECHSTIL

PROJECT DOCUMENTATION

Project Topic: Smart Textile Production Tracking and Quality Control System

Project Type: Desktop Automation Application (WinForms)

Platform: .NET / Windows

1.Introduction

1.1 Project Definition and Purpose

This project is a modular automation system designed to manage order, inventory, and production processes in a textile factory within a digital environment. The primary objective of the system is to eliminate manual tracking, monitor garment production stages (cutting, sewing, ironing, packaging) in real-time, and detect fabric defects (tears, stains, weaving errors) using AI-supported image processing technology, thereby minimizing the human factor.

1.2. Scope of the Project

The Ototechstil project is a comprehensive automation system that digitalizes the core business processes of textile enterprises and manages them through a centralized database. The project covers the essential data flow required by a business, ranging from customer definitions to order management.

- **Data Management Infrastructure:** Dynamic definition and management of business data (Customers, Fabric Types, Color Types, Product Types).
- **Order Management System:** Recording orders received from customers into the system, creating order details, and tracking them.
- **Production Tracking:** Monitoring the status of products in the order on the production line, tracking their current stage, and ensuring process management.
- **Integrated Quality Control Module:** Analyzing images of relevant order items with Artificial Intelligence to ensure defect control.
- **Reporting and Monitoring:** Presenting the results of entered orders and analyses in the form of visual charts and lists.

1.3. Technologies Used

Modern and sustainable software technologies compliant with industry standards were preferred in the development of the Ototechstil project. The main technologies and tools used are:

- **Programming Language and Platform:** C# programming language and the .NET 8.0 platform were used for backend coding. A modular structure was created by adhering to Object-Oriented Programming principles.
- **Development Environment (IDE):** Microsoft Visual Studio 2022 was used for coding, debugging, and design processes.
- **Database Management System:** Microsoft SQL Server (and SQLite for local deployment) was preferred as the relational database to securely store all system data.
- **ORM (Object Relational Mapping):** Entity Framework Core technology (Database-First approach) was used instead of classical SQL queries for database operations to ensure code security and speed.
- **User Interface (UI):** DevExpress UI Components were used instead of standard Windows forms to offer a more modern, aesthetic, and advanced data management experience.
- **AI and Machine Learning:** The Microsoft ML.NET library was used for image processing and defect detection operations in the quality control module. The model was trained using the Image Classification algorithm.
- **Installation and Deployment:** Inno Setup Compiler was used to present the project to the end-user as a single file and to create a professional installation wizard.

2.Project Plan

2.1. System Users

There is fundamentally a single user role in the system:

1. Administrator (Admin): The user with full authority over the system. Duties and authorities include:

- Can log in to the system and access all modules.
- Creates, updates, and deletes basic definitions (Customer, Fabric, Color, Product Type).
- Views all order and production processes, accesses statistical reports (Dashboard).
- Can perform deletion or modification of critical data.

- Has full access and authority.

2.2. Functional Requirements

The system must fulfill the following basic functions to achieve its targeted purpose:

- **Basic Data Management (CRUD):** Basic parameters such as customer, fabric type, color, and product type must be addable, updatable, and deletable in the database.
- **Order Creation:** Users must be able to create new order records according to customer requests and enter details (quantity, product features, etc.) into the system.
- **Production Status Tracking:** Production stages of created orders (Preparing, In Production, In Control, Completed, etc.) must be updatable and monitorable instantly via the system.
- **Image Upload and Processing:** In the quality control module, the user must be able to select a fabric photo from the computer and upload it to the system for analysis.
- **AI Analysis:** Uploaded images must be scanned by the integrated machine learning model, and the system must automatically detect whether the image is "Intact" or "Defective".
- **Saving Results:** AI analysis results and production data must be permanently stored in the database along with date information.

2.3. Non-Functional Requirements

The success of the system depends not only on the functionality but also on user experience, performance, and reliability criteria.

- **User-Friendly and Modern Interface:** A modern, stylish, and intuitive interface was designed using the DevExpress library. The user's learning time for the system has been minimized.
- **High Performance and Speed:** Database queries are optimized with Entity Framework Core, and AI analyses are set to run locally, not cloud-based. Thus, analysis results can be obtained in seconds without the need for an internet connection.
- **Ease of Installation:** The system is structured to be easily installed via a single installation file (setup.exe) prepared with Inno Setup, using a "Next > Next > Finish" logic.
- **Error Management and Reliability:** Error catching mechanisms (Try-Catch blocks) have been developed against potential user errors or systemic problems (e.g., uploading the wrong file format).
- **Extensibility:** Since the system is designed in a modular architecture (N-Layered Architecture), it is open to the integration of new features in the future.

- **System Compatibility:** The application is compiled to run smoothly on Windows 10 and Windows 11 operating systems (x64 architecture).

2.4. Sample Usage Scenario (Happy Path)

To understand the system's operation, here is the "Flawless Flow” scenario:

1. **The Planner** logs into the system and creates an order for 500 units of "Red Cotton Combed T-Shirts" from the firm "ABC Tekstil".
2. **The Order** triggers the Stock Module and automatically deducts the necessary amount of fabric from the inventory.
3. **The System** automatically generates a "Work Order" and marks the status as "Order Received".
4. **The Operator** presses the "Next Stage" button on the screen upon completing the process at each stage; the status transitions to the next phase.
5. **When production is finished**, the product appears on the Quality Control screen.
6. **The Camera** captures an image of the fabric, and the AI Module grants "Clean" approval.
7. **The Product** transitions to the "Completed" status.

3.Realization of the Project

3.1. Design of Modules

The Ototechstil project has been designed as modules that are integrated with each other but work independently, observing the "Single Responsibility Principle". This modular structure increases the manageability of the system and facilitates error tracking.

3.1.1. Order Management Module

This is the module where customer requests are first entered into the system and the order record is created.

- **Function:** Staff enters order information, selects the customer, determines product features, and creates the order object by entering the quantity.
- **Code Structure:** This module establishes a relational link with almost all tables in the database. Validations at the time of order creation are checked in this layer.

3.1.2. Production Tracking Module

This module manages the lifecycle and status changes of created orders within the factory.

- **Function:** Updates the current status of an existing order (Cutting, Sewing, Ironing, Packaging, etc.).
- **Design Logic:** When the status of an order changes, the process is handled as a flow. It works with State Management logic on the code side.

3.1.3. AI-Supported Quality Control Module

This service houses the system's image processing and analysis capabilities.

- **Function:** Takes physical product images and analyzes them.
- **Technology:** Developed using the Microsoft ML.NET library, this module works like a "decision mechanism" independent of the main application. It takes visual data as input and returns the classification result and confidence score as output. Industrial-quality images with fixed light, angle, and resolution standards were used in the training dataset.

3.1.4. Data Management Modules

This is the CRUD (Create, Read, Update, Delete) layer where the static data forming the cornerstones of the application is managed. Customer Definitions, Fabric Types, Color Codes, and Product Types are managed as separate classes here.

3.1.5. Statistics and Reporting Modules

This is the analysis layer that feeds from the other four modules and only performs "Reading".

- **Function:** Pulls data from other tables with complex SQL queries, groups them, and visualizes them.
- **Design:** No data entry is performed; only graphical presentation of existing data is done on the Dashboard.

3.1.6. Inventory (Stock) Module

This module tracks the enterprise's raw material (fabric) warehouse in a digital environment and guides production planning.

- **Function:** "Available Footage" information is entered into the system for each defined fabric type. When a new order is created, the fabric amount required by the order is calculated and automatically deducted from the warehouse stock.
- **Critical Stock Warning:** When the stock falls below the "Threshold Level" determined for each fabric, the system opens a red warning card to the manager on the Dashboard.

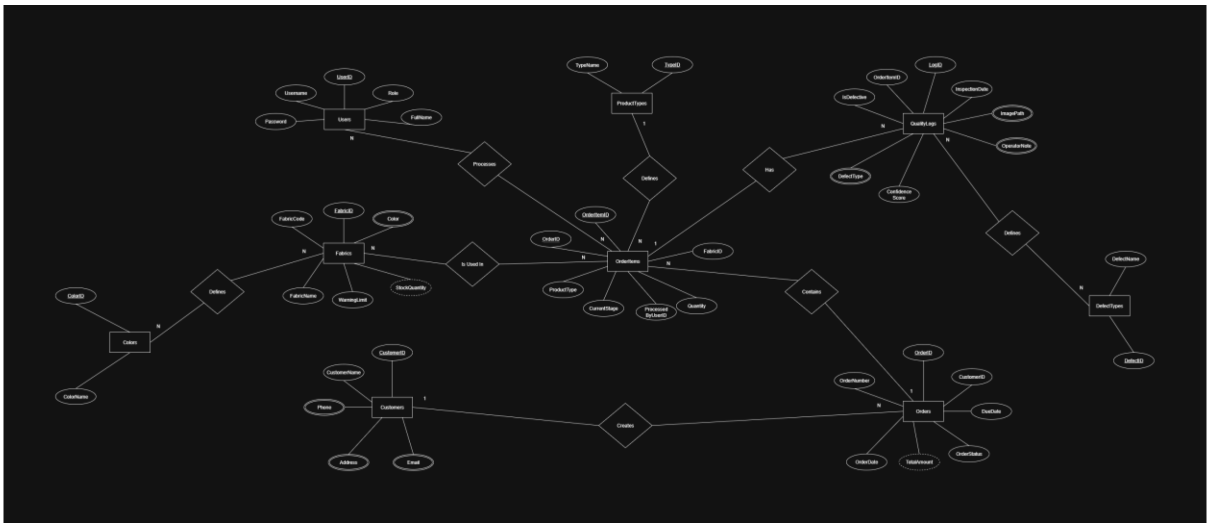
3.1.7. Validation Module

This is the security layer working to minimize user errors and prevent Corrupt Data entry into the database.

- **Design Logic:** Developed using "Fluent Validation" principles or specific Business Rules. This module activates the moment the user presses the "Save" button.

3.2. Database Design

3.2.1 ER Diagram



ER Diagram

A Small Note:

→ The classes within the Data Access Layer have been generated from database tables using the **Scaffolding** (automatic code generation) method. Consequently, the C# classes share the exact same attributes and relationships as the database tables. The **ER Diagram** should be referenced to examine the system's data structure; to avoid redundancy, a separate class diagram was not created for the Entity classes.

3.2.2 Tables

Users

Column Name	Data Type	Description	Constraints
UserID	INT	Unique user ID.	PK, Identity(1,1)
Username	NVARCHAR(50)	System login name.	Not Null
Password	NVARCHAR(50)	User password.	Not Null
Role	NVARCHAR(20)	Authority level (Admin, Operator, Quality).	Not Null
FullName	NVARCHAR(100)	Personnel name and surname information.	Nullable

Fabrics

Column Name	Data Type	Description	Constraints
FabricID	INT	Unique fabric ID.	PK, Identity(1,1)
FabricCode	NVARCHAR(50)	Stock tracking code (Barcode).	Unique, Not Null
FabricName	NVARCHAR(100)	Commercial name of the fabric (e.g., Combed Cotton).	Nullable
Color	NVARCHAR(50)	Fabric color.	FK (Colors)
StockQuantity	DECIMAL(10,2)	Current quantity in warehouse (Meters).	Default: 0
WarningLimit	DECIMAL(10,2)	Critical stock warning threshold.	Default: 100

Customers

Column Name	Data Type	Description	Constraints
CustomerID	INT	Unique customer ID.	PK, Identity(1,1)
CustomerName	NVARCHAR(100)	Company or individual name.	Not Null
Phone	NVARCHAR(20)	Contact phone number.	Nullable
Email	NVARCHAR(100)	E-mail address.	Nullable
Address	NVARCHAR(250)	Delivery and billing address.	Nullable

Orders

Column Name	Data Type	Description	Constraints
OrderID	INT	System-internal order ID.	PK, Identity(1,1)
OrderNumber	NVARCHAR(20)	Tracking number (e.g., ORD-2023-001).	Unique, Not Null
CustomerID	INT	ID of the customer placing the order.	FK (Customers)
OrderDate	DATETIME	Date the order was received.	Default: GetDate()
DueDate	DATETIME	Planned delivery date.	Nullable
TotalAmount	DECIMAL(18,2)	Total monetary amount of the order.	Nullable
OrderStatus	NVARCHAR(20)	General status (Pending, Completed).	Default: 'Pending'

OrderItems

Column Name	Data Type	Description	Constraints
OrderItemID	INT	Unique item ID.	PK, Identity(1,1)
OrderID	INT	Parent order ID it is connected to.	FK (Orders)
FabricID	INT	ID of the fabric to be used.	FK (Fabrics)
ProductType	NVARCHAR(100)	Product Type (T-Shirt, Trousers).	FK (ProductTypes)
Quantity	INT	Quantity to produce.	Not Null
CurrentStage	NVARCHAR(50)	Instant Production Stage (Cutting, Sewing..).	Default: 'Planning'
ProcessedByUserID	INT	ID of the personnel who performed the last operation.	FK (Users)

QualityLogs

Column Name	Data Type	Description	Constraints
LogID	INT	Unique log ID.	PK, Identity(1,1)
OrderItemID	INT	ID of the product item being inspected.	FK (OrderItems)
InspectionDate	DATETIME	Time of inspection.	Default: GetDate()

ImagePath	NVARCHAR(500)	File path of the analyzed image.	Nullable
IsDefective	BIT	Defect Status (0: Intact, 1: Defective).	Nullable
DefectType	NVARCHAR(50)	Detected defect (Stain, Tear).	FK (DefectTypes)
ConfidenceScore	FLOAT	AI confidence score (0.0-1.0).	Nullable
OperatorNote	NVARCHAR(200)	Notes added by the human operator.	Nullable

Colors

Column Name	Data Type	Description	Constraints
ColorID	INT	Unique color ID.	PK, Identity(1,1)
ColorName	NVARCHAR(50)	Name of the color (e.g., Navy Blue).	Unique, Not Null

ProductTypes

Column Name	Data Type	Description	Constraints
TypeID	INT	Unique type ID.	PK, Identity(1,1)
TypeName	NVARCHAR(50)	Product name (e.g., Polo T-Shirt).	Unique, Not Null

DefectTypes

Column Name	Data Type	Description	Constraints
DefectID	INT	Unique defect ID.	PK, Identity(1,1)
DefectName	NVARCHAR(50)	Defect definition (e.g., Oil Stain).	Unique, Not Null

3.2.3 Relationships

1. Customers ➔ Orders (One-to-Many)

- Linked via Customers.CustomerID (PK) and Orders.CustomerID (FK).

2. Orders ➔ OrderItems (One-to-Many)

- Linked via Orders.OrderID (PK) and OrderItems.OrderID (FK).

3. Fabrics ➔ OrderItems (One-to-Many)

- Linked via Fabrics.FabricID (PK) and OrderItems.FabricID (FK).

4. OrderItems ➔ QualityLogs (One-to-Many)

- Linked via OrderItems.OrderItemID (PK) and QualityLogs.OrderItemID (FK).

5. Users ➔ OrderItems (One-to-Many / Optional)

- Linked via Users.UserID (PK) and OrderItems.ProcessedByUserID (FK).

6. Colors ➔ Fabrics (One-to-Many)

- Linked via Colors.ColorID (PK) and Fabrics.ColorID (FK).

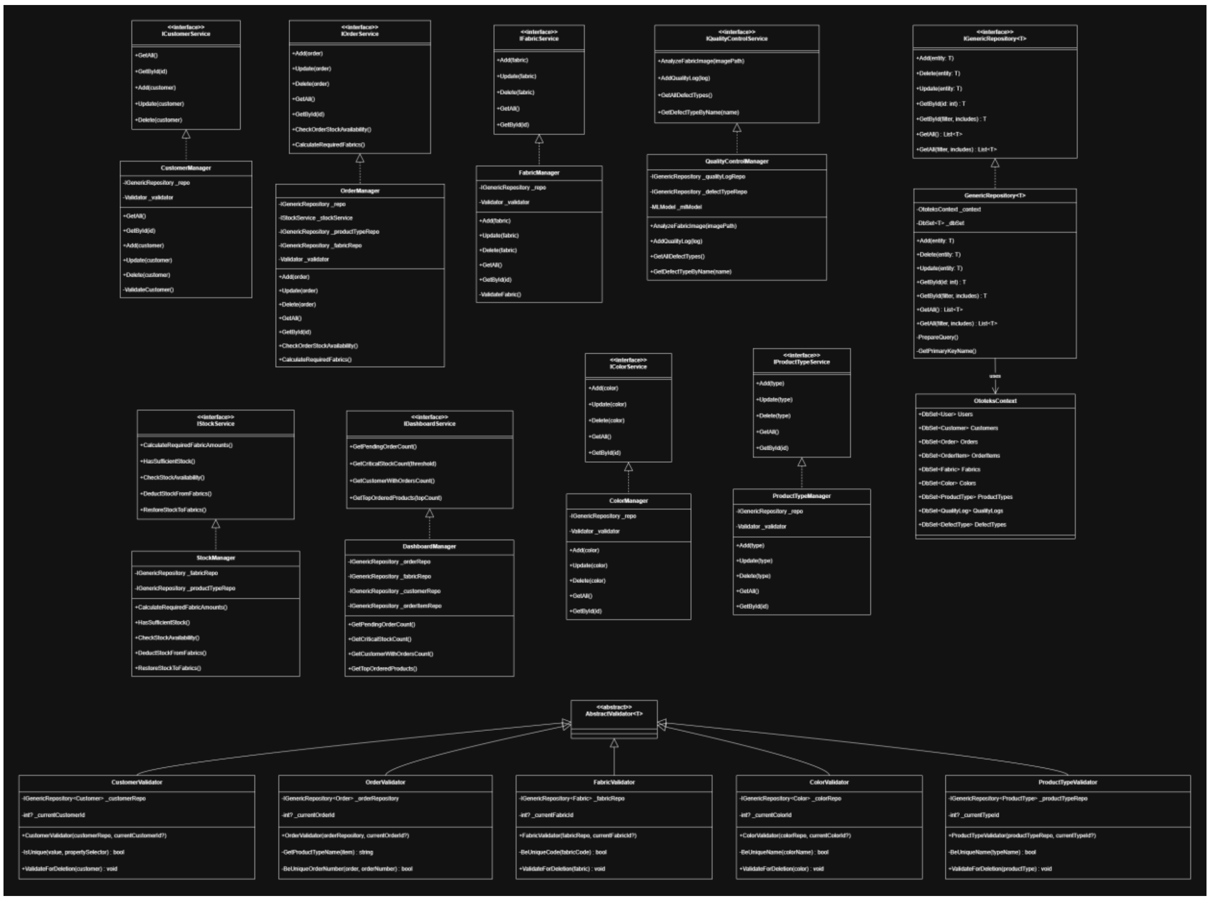
7. ProductTypes ➔ OrderItems (One-to-Many)

- Linked via ProductTypes.TypeID (PK) and OrderItems.TypeID (FK).

8. DefectTypes ➔ QualityLogs (One-to-Many)

- Linked via DefectTypes.DefectID (PK) and QualityLogs.DefectID (FK).

3.3 Class Diagram


<<interface>>
ICustomerService
+GetAll()
+GetById(id)
+Add(customer)
+Update(customer)
+Delete(customer)
CustomerManager
-IGenericRepository _repo
-Validator _validator
+GetAll()
+GetById(id)
+Add(customer)
+Update(customer)
+Delete(customer)
-ValidateCustomer()
<<interface>>
IOrderService
+Add(order)
+Update(order)
+Delete(order)
+GetAll()
+GetById(id)
+CheckOrderStockAvailability()
+CalculateRequiredFabrics()
OrderManager
-IGenericRepository _repo
-IStockService _stockService
-IGenericRepository _productTypeRepo
-IGenericRepository _fabricRepo
-Validator _validator
+Add(order)
+Update(order)
+Delete(order)
+GetAll()
+GetById(id)
+CheckOrderStockAvailability()
+CalculateRequiredFabrics()
<<interface>>
IFabricService
+Add(fabric)
+Update(fabric)
+Delete(fabric)
+GetAll()
+GetById(id)
FabricManager
-IGenericRepository _repo
-Validator _validator
+Add(fabric)
+Update(fabric)
+Delete(fabric)
+GetAll()
+GetById(id)
-ValidateFabric()
<<interface>>
IQualityControlService
+AnalyzeFabricImage(imagePath)
+AddQualityLog(log)
+GetAllDefectTypes()
+GetDefectTypeByName(name)
QualityControlManager
-IGenericRepository _qualityLogRepo
-IGenericRepository _defectTypeRepo
-MLModel _mlModel
+AnalyzeFabricImage(imagePath)
+AddQualityLog(log)
+GetAllDefectTypes()
+GetDefectTypeByName(name)
<<interface>>
IGenericRepository<T>
+Add(entity: T)
+Delete(entity: T)
+Update(entity: T)
+GetById(id: int) : T
+GetById(filter, includes) : T
+GetAll() : List<T>
+GetAll(filter, includes) : List<T>
GenericRepository<T>
-OtoteksContext _context
-DbSet<T> _dbSet
+Add(entity: T)
+Delete(entity: T)
+Update(entity: T)
+GetById(id: int) : T
+GetById(filter, includes) : T
+GetAll() : List<T>
+GetAll(filter, includes) : List<T>
-PrepareQuery()
-GetPrimaryKeyName()
uses
OtoteksContext
+DbSet<User> Users
+DbSet<Customer> Customers
+DbSet<Order> Orders
+DbSet<OrderItem> OrderItems
+DbSet<Fabric> Fabrics
+DbSet<Color> Colors
+DbSet<ProductType> ProductTypes
+DbSet<QualityLog> QualityLogs
+DbSet<DefectType> DefectTypes
<<interface>>
IStockService
+CalculateRequiredFabricAmounts()
+HasSufficientStock()
+CheckStockAvailability()
+DeductStockFromFabrics()
+RestoreStockToFabrics()
StockManager
-IGenericRepository _fabricRepo
-IGenericRepository _productTypeRepo
+CalculateRequiredFabricAmounts()
+HasSufficientStock()
+CheckStockAvailability()
+DeductStockFromFabrics()
+RestoreStockToFabrics()
<<interface>>
IDashboardService
+GetPendingOrderCount()
+GetCriticalStockCount(threshold)
+GetCustomerWithOrdersCount()
+GetTopOrderedProducts(topCount)
DashboardManager
-IGenericRepository _orderRepo
-IGenericRepository _fabricRepo
-IGenericRepository _customerRepo
-IGenericRepository _orderItemRepo
+GetPendingOrderCount()
+GetCriticalStockCount()
+GetCustomerWithOrdersCount()
+GetTopOrderedProducts()
<<interface>>
IColorService
+Add(color)
+Update(color)
+Delete(color)
+GetAll()
+GetById(id)
ColorManager
-IGenericRepository _repo
-Validator _validator
+Add(color)
+Update(color)
+Delete(color)
+GetAll()
+GetById(id)
<<interface>>
IProductTypeService
+Add(type)
+Update(type)
+Delete(type)
+GetAll()
+GetById(id)
ProductTypeManager
-IGenericRepository _repo
-Validator _validator
+Add(type)
+Update(type)
+Delete(type)
+GetAll()
+GetById(id)
<<abstract>>
AbstractValidator<T>
CustomerValidator
-IGenericRepository<Customer> _customerRepo
-int? _currentCustomerId
+CustomerValidator(customerRepo, currentCustomerId?)
-IsUnique(value, propertySelector) : bool
+ValidateForDeletion(customer) : void
OrderValidator
-IGenericRepository<Order> _orderRepository
-int? _currentOrderId
+OrderValidator(orderRepository, currentOrderId?)
-GetProductTypeName(item) : string
-BeUniqueOrderNumber(order, orderNumber) : bool
FabricValidator
-IGenericRepository<Fabric> _fabricRepo
-int? _currentFabricId
+FabricValidator(fabricRepo, currentFabricId?)
-BeUniqueCode(fabricCode) : bool
+ValidateForDeletion(fabric) : void
ColorValidator
-IGenericRepository<Color> _colorRepo
-int? _currentColorId
+ColorValidator(colorRepo, currentColorId?)
-BeUniqueName(colorName) : bool
+ValidateForDeletion(color) : void
ProductTypeValidator
-IGenericRepository<ProductType> _productTypeRepo
-int? _currentTypeId
+ProductTypeValidator(productTypeRepo, currentTypeId?)
-BeUniqueName(typeName) : bool
+ValidateForDeletion(productType) : void

Class Diagram

3.4. Software Architecture and Technical Implementation Details

The **N-Layered Architecture** model was adopted during the project development process. The system is built on three main layers:

3.4.1. Data Access Layer (DAL)

The lowest layer where the system communicates with the database.

- **Technology:** Developed using Entity Framework Core (ORM) on a SQLite database.
- **Design Pattern:** The Generic Repository Pattern was used to standardize database operations. This allows all CRUD operations to be managed from a single center instead of writing repetitive SQL queries for different modules.
- **Database Physical Location:** Due to data security standards and Windows operating system user authorization policies (UAC), the physical location of the database file is configured as the user's %AppData%\Roaming\Ototechstil directory instead of the 'Program Files' directory where the application is installed. Thanks to this architectural choice, the application can write data without needing administrator

rights, data loss risks are minimized, and the protection of user data is guaranteed even if the application is uninstalled.

3.4.2. Business Logic Layer (BLL)

The layer where data coming from the user interface is processed and rules are audited.

- **Validations:** Necessary checks (e.g., stock quantity not being negative, required fields being filled) are performed here before recording to the database.
- **Management:** Each module has its own "Manager" class (e.g., OrderManager, FabricManager). These classes act as a bridge between the data access layer and the interface.

3.4.3. Presentation Layer (UI)

The interface layer where the user interacts with the system.

- **Technology:** DevExpress WinForms components were used to increase User Experience (UX).
- **Standardization:** Common Helper classes and Interfaces were developed to prevent code repetition between forms.

4. Test Processes and Findings

During the development phase of the project, Manual Test (Black Box Testing) methods were applied to verify the functionality of the system. End-user scenarios (Order creation, AI analysis, etc.) were tried step by step directly on the system.

As a result of these tests, it was observed that validation rules in data entries worked without error, the AI module classified images correctly, and data integrity was preserved in database relationships.

5. General Evaluation of the Project

The Ototeks Automation System developed within the scope of this project was designed and successfully implemented to offer an integrated solution to production tracking and quality control problems experienced in the textile sector.

Project Advantages and Success Criteria:

- **Innovation:** Unlike standard automation programs, it minimized human error in the quality control process thanks to its image processing capability.
- **User Experience:** Using DevExpress components, a modern interface was presented that allows complex data (statistics, production flow) to be easily understood by the end-user.
- **Architectural Structure:** The modular structure designed by adhering to the "Single Responsibility Principle" created a flexible ground allowing the project to be moved to web or mobile platforms in the future.
- **Data Security:** Data consistency was largely ensured thanks to the validation layer and relational database design.