**COBOL-.NET Converter**

## Overview:

Experian has substantial amount of legacy Mainframe applications running critical business processes. Due to aging infrastructure, limited expertise, high maintenance costs and challenges in integrating with modern systems, there is a strategic need to modernize these COBOL applications to contemporary .NET 8 WebAPI platforms.

## Objective:

COBOL programs often contain complex business logic, intricate data structures, business rules that are challenging to understand and migrate manually. The manual conversion process is time-consuming, error-prone, and requires deep expertise in both legacy COBOL systems and modern .NET technologies. To accelerate and automate this modernization journey, UST has developed a COBOL to .NET 8 WebAPI Converter leveraging Generative AI. This solution intelligently analyzes COBOL source code, extracts business logic, identifies data patterns, maps legacy constructs to modern equivalents, and generates production-ready .NET 8 WebAPI applications with comprehensive Entity Framework Core integration.

## Approach:

The conversion process comprises of 3 main phases with sub-phases of each phase. Each phase builds upon the previous phase's outputs to ensure comprehensive and accurate conversion. The phases and sub-phases are detailed below:

| **Phase** | **Sub-phases** | **Details** |
| --- | --- | --- |
| **Analysis** | File Classification | All uploaded COBOL artifacts are classified into categories: COBOL Code, JCL Scripts, Copybooks, VSAM Definitions, BMS Maps, Control Files, and Standards Documents |
| COBOL Structure Analysis | Parsing of COBOL source code to extract divisions, sections, paragraphs, data structures, business logic, and control flow patterns |
| Requirements Generation | AI-powered extraction of business requirements and technical specifications from COBOL source code and associated artifacts |
| **Conversion** | Target Architecture Design | Generation of .NET 8 WebAPI project structure based on analyzed COBOL components and identified business patterns |
| Business Logic Mapping | Conversion of COBOL procedure division logic to C# service methods with preserved business rules and validation logic |
| Data Structure Migration | Transformation of COBOL data structures to C# entities with Entity Framework Core integration |
| API Endpoint Generation | Creation of RESTful API controllers based on COBOL program entry points and business functions |
| Database Integration | Implementation of Entity Framework Core with proper repository patterns for data access |
| **Validation** | Unit Test Generation | Automated creation of comprehensive unit tests for all generated C# components |
| Functional Test Creation | Generation of functional test cases based on extracted business requirements |

## Details of Each Phase:

### Analysis Phase:

The Analysis phase involves a comprehensive assessment of the COBOL program to understand its functionality, dependencies, and business logic. This phase includes inventorying codebases, identifying integration points with other systems, and evaluating technical debt to prioritize modernization efforts. By mapping out these elements, the Analysis phase ensures a clear roadmap for converting legacy systems to a modern .NET 8 WebAPI platform while mitigating risks.

The analysis phase includes the following sub-phases

**File Classification:**

This phase involves identifying COBOL-related file types within the said program and automatically classifies them into the categories given below

* **COBOL Source Code:** Main program files (.cbl, .cobol, .cob) containing business logic
* **JCL Scripts:** Job Control Language files (.jcl) for batch processing and system operations
* **Copybooks:** Reusable data structure definitions (.cpy) shared across programs
* **VSAM Definitions:** Virtual Storage Access Method file definitions for data storage
* **BMS Maps:** CICS screen definitions for user interface components
* **Control Files:** Configuration and parameter files (.ctl, .cfg)
* **Standards Documents:** PDF, DOC, DOCX files containing coding standards and business rules

**COBOL Structure Analysis:**

The analysis engine performs parsing of COBOL source code:

* **Division Analysis:** Identification and parsing of Environment, Data, Identification and Procedure divisions
* **Data Structure Extraction:** Analysis of Working-Storage Section, File Section, and Linkage Section to identify data elements, their relationships, and usage patterns
* **Business Logic Identification:** Extraction of business rules, validation logic, calculations, and decision trees from the Procedure Division
* **Control Flow Mapping:** Analysis of PERFORM statements, conditional logic, and program flow to understand execution patterns
* **External Dependencies:** Identification of CALL statements, copybook inclusions, and external file references

**Requirements Generation:**

Using AI the system generates:

* **Business Requirements:** Documentation of business rules, processes, and functional requirements extracted from COBOL logic
* **Technical Requirements:** Detailed technical specifications of COBOL programs and related files .

This phase involves transformation of a COBOL program into .NET 8 WebAPI applications by translating business logic, data structures, and file operations into corresponding C# code. Conversion of COBOL records, group items, and PIC clauses into .NET classes, RESTful APIs, and database interactions, while manually refactoring complex logic to ensure functional equivalence is also included as part of the conversion phase. Conversion phase involves the following sub-phases

**Target Architecture Design:**

Based on the analysis results, the system designs a .NET 8 WebAPI architecture:

* **Project Structure:** Standard .NET 8 solution with appropriate folder organization (Controllers, Models, Services, Repositories, Data)
* **Entity Framework Integration:** Database context design with entity mappings for COBOL data structures
* **Dependency Injection:** Service registration and dependency configuration
* **Security Framework:** Authentication and authorization setup
* **Logging and Monitoring:** Structured logging implementation with Serilog

**Business Logic Mapping:**

The conversion engine transforms COBOL logic to C# implementations with appropriate error handling:

* **COBOL PERFORM to C# Methods:** Conversion of COBOL paragraphs and sections to C# service methods
* **Conditional Logic:** Translation of COBOL conditional statements like IF-THEN-ELSE, EVALUATE etc. statements to corresponding relevant C# conditional structures
* **Arithmetic Operations:** Conversion of COBOL arithmetic (ADD, SUBTRACT, MULTIPLY, DIVIDE) to C# equivalents
* **Data Validation:** Implementation of COBOL validation rules as C# validation attributes and custom validators
* **File Operations:** Migration of COBOL file I/O operations to Entity Framework Core database operations

**Data Structure Migration:**

Transformation of COBOL data elements to corresponding and relevant .NET data elements:

* **Record Structures:** Conversion of COBOL 01-level records to C# entity classes
* **Field Definitions:** Mapping of COBOL PIC clauses to appropriate C# data types with validation attributes
* **Hierarchical Data:** Transformation of COBOL group items to C# nested objects or separate entities
* **Database Schema:** Generation of Entity Framework migrations and database configuration

**API Endpoint Generation:**

Creation of RESTful API structure:

* **Controller Classes:** Generation of API controllers based on COBOL program functions
* **HTTP Methods:** Mapping of COBOL operations to appropriate REST endpoints (GET, POST, PUT, DELETE)
* **Request/Response Models:** Creation of DTOs (Data Transfer Objects) for API communication
* **Validation Integration:** Implementation of model validation using Data Annotations

### Validation Phase:

This phase ensures the converted .NET 8 WebAPI application accurately replicates the functionality of the original COBOL program by unit testing C# code, integration testing RESTful APIs, and verifying database interactions against the original COBOL specifications to confirm functional equivalence. Validation phase is used to mitigates risks and ensures the modernized system maintains data integrity and business continuity**.**

**Unit Test Case Generation:**

Automated creation of test cases for coverage:

**Functional Test Case Generation:**

Automated creation of functional test cases for coverage:

## Input Files and Supported Formats:

**COBOL Source Files:**

* Extensions: .cbl, .cobol, .cob, .pco, .ccp
* Content: COBOL programs with all four divisions
* Special handling for CICS commands and embedded SQL

**Job Control Language (JCL):**

* Extensions: .jcl, .job, .cntl
* Content: Job definitions, step procedures, file allocations
* Mapping to .NET background services and scheduled tasks

**Copybooks:**

* Extensions: .cpy, .copybook, .inc
* Content: Data structure definitions, constants, working storage layouts
* Conversion to C# models and shared data contracts

**Standards Documents:**

* Extensions: .pdf, .docx, .txt
* Content: Coding standards, business rules documentation
* Integration through RAG for context-aware conversion

## Generated .NET 8 WebAPI Structure:

**ProjectName/**

   ├── **Controllers/**            # REST API endpoints

   ├── **Models/**

   │       ├── **Entities/**          # Entity Framework entities

   │       ├── **DTOs/**             # Data Transfer Objects

   │       └── **ViewModels/**       # API response models

   ├── **Services/**

   │       ├── **Interfaces/**       # Service contracts

   │       └── **Implementations/** # Business logic

   ├── **Repositories/**

   │       ├── **Interfaces/**       # Repository contracts

   │       └── **Implementations/** # Data access logic

   ├── **Data/**

   │       ├── **ApplicationDbContext.cs**

   │       ├── **Configurations/**   # Entity configurations

   │       └── **Migrations/**       # EF Core migrations

   ├── **Infrastructure/**       # External integrations

   ├── **Program.cs**            # Application startup

   ├── **appsettings.json**     # Configuration

   └── **ProjectName.csproj**    # Project file

**ProjectName.Tests/**

   ├── **Unit/**                # Unit tests

## Challenges:

**Complex Business Logic Extraction:**

COBOL programs often contain deeply embedded business rules that are intertwined with system-level operations, making it challenging to separate pure business logic from infrastructure concerns.

**Data Type Mapping Complexity:**

COBOL's unique data types (COMP fields, packed decimals, REDEFINES clauses) require sophisticated mapping to appropriate .NET data types while preserving precision and business logic.

**Performance Characteristics:**

Mainframe COBOL applications are optimized for batch processing and high-volume transactions, requiring careful consideration of .NET performance patterns and scalability approaches.

**Testing Legacy Business Rules:**

Validating that converted .NET code maintaining the same business behavior as the original COBOL application requires comprehensive test case generation and business rule verification.

## Recommendations:

After completing multiple COBOL-to-.NET 8 conversions across various industries and complexity levels, we found that the COBOL-.NET 8 WebAPI Converter performs best when used on well-structured programs with clear documentation and simple CICS use, automating 70-95% of the conversion.

It achieves moderate success with mixed-style or moderately complex systems, while highly complex or poorly documented programs need more manual effort. Key factors for success include thorough analysis, strong coding standards, clear business rule documentation, and a phased, iterative approach. Providing complete files, involving business users for validation, thorough testing, and proper change management also lead to the best results. Overall, the converter efficiently modernizes legacy systems while preserving critical business logic.