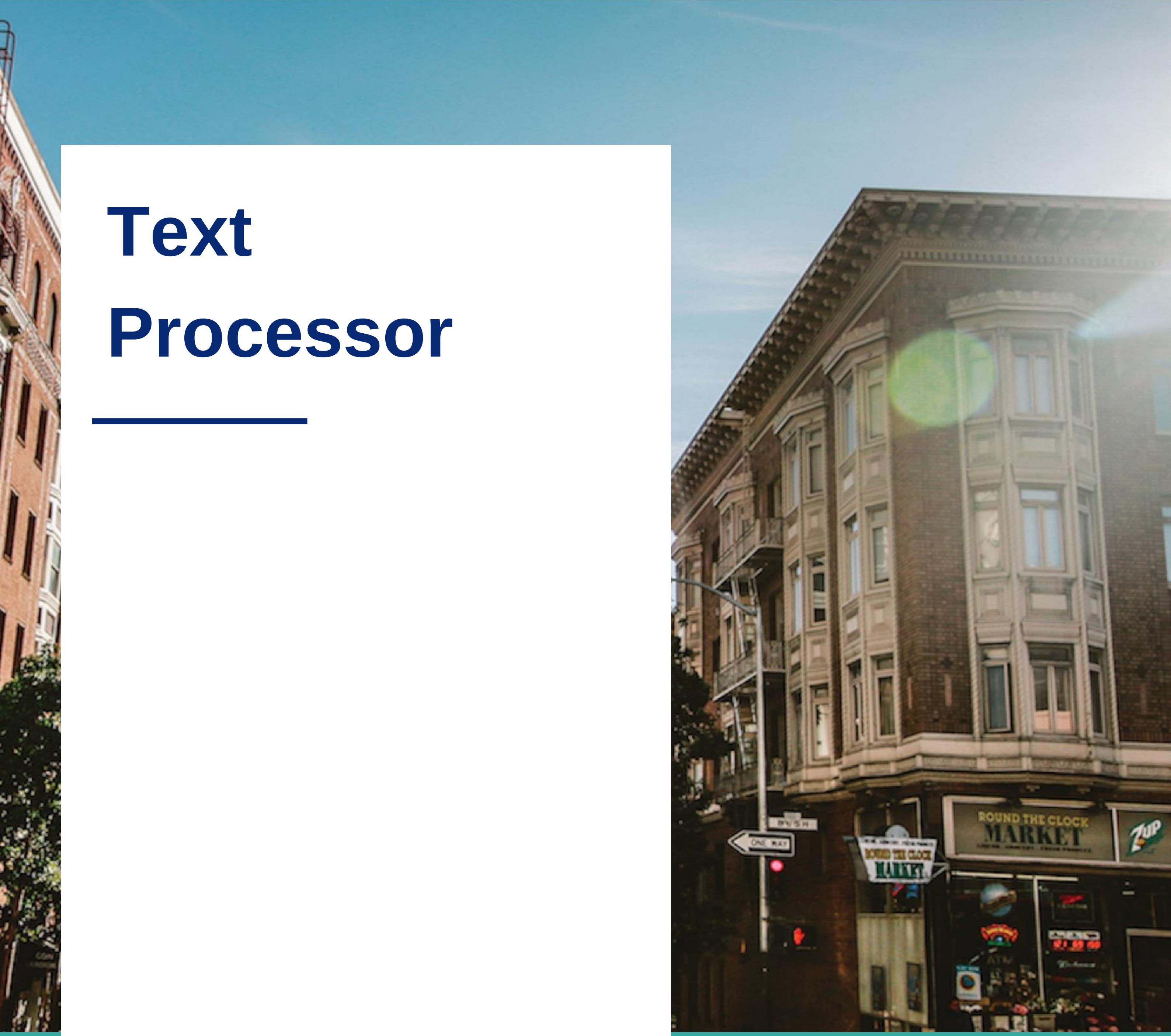


Text Processor

DECEMBER 10TH

MECALUX
Created By: MAURO GIOBERTI

MECALUX

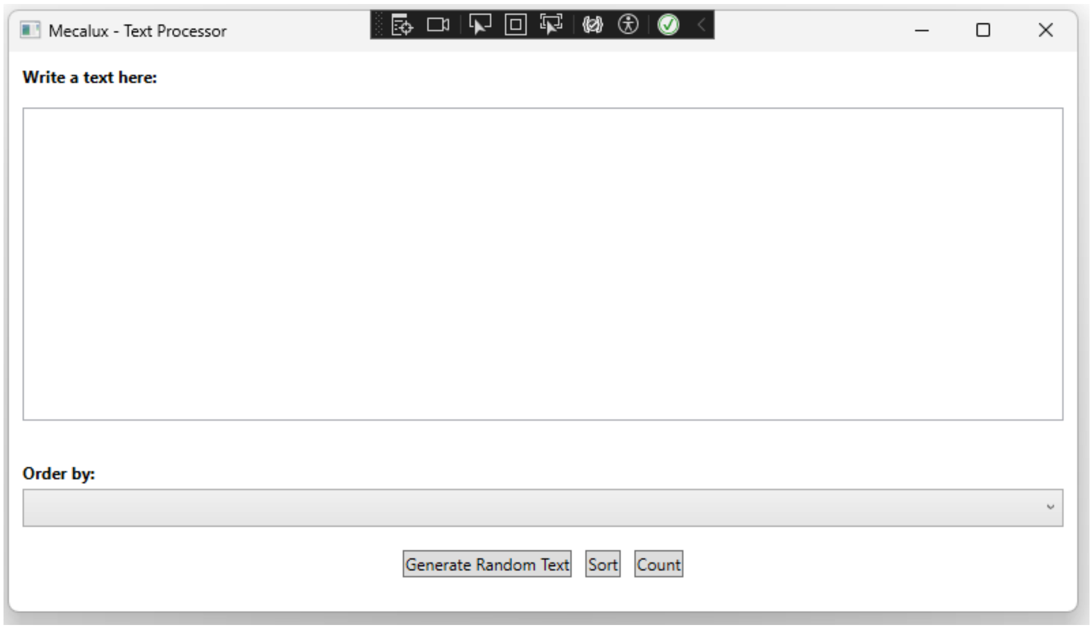
Mecalux: Text Processor

1. Introduction

Mecalux.TextProcessor **is a cutting-edge .NET 8 application designed to provide robust text processing capabilities. It features a WPF user interface for desktop interaction and a Rest API for service-oriented tasks. This application is ideal for sorting text and extracting key text statistics.**

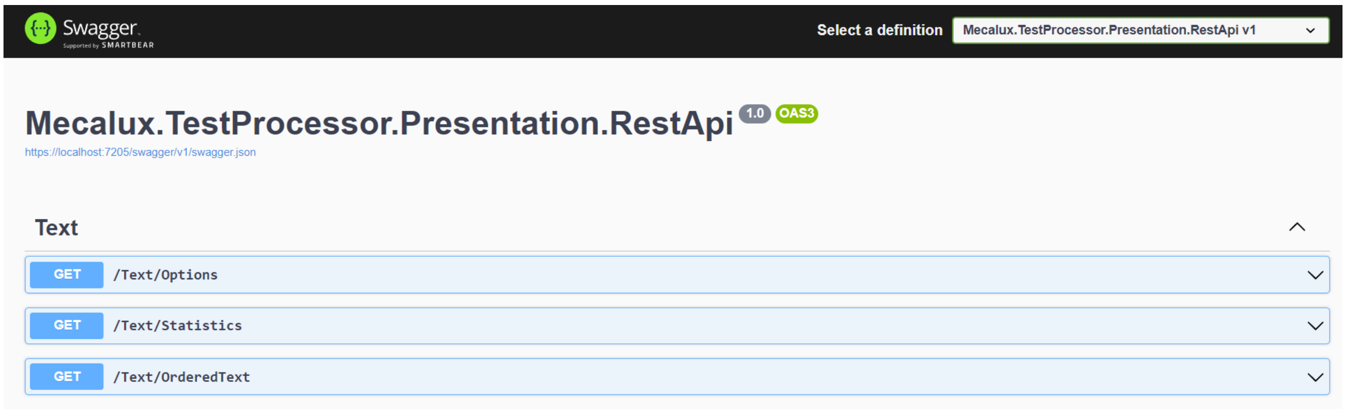
The project boasts two primary entry points, offering flexibility and convenience for different use cases. Users can interact with the application through a graphical interface provided by the WPF application, which offers an intuitive and user-friendly experience for performing text-processing tasks. Alternatively, for those preferring or requiring programmatic access, the application exposes a Rest API. This API can be accessed through tools like Swagger or Postman, providing a powerful and flexible way to integrate text-processing capabilities into various workflows or other software systems.

WPF User Interface



"This screenshot shows the WPF desktop application interface, highlighting its features and user interactions".

Swagger UI for Rest API



Mecalux.TestProcessor.Presentation.RestApi 1.0 OAS3

https://localhost:7205/swagger/v1/swagger.json

Text

GET /Text/Options

GET /Text/Statistics

GET /Text/OrderedText

"This screenshot illustrates the Swagger UI for the Rest API, showcasing the available endpoints and their usage".

3. Development Environment Setup

Prerequisites

- .NET SDK 8
- Visual Studio 2022 or later

Environment Configuration

Developers should clone the repository and open the solution in Visual Studio. After opening, restore the required NuGet packages to ensure all dependencies are correctly set up.

4. Project Structure

The project is organized into several directories, each with a specific role:

- **ResourceAccess.Contracts**: Interfaces for abstracting data access methods.
- **Business.Logic**: Business logic implementation, containing algorithms for text processing and statistical analysis.
- **Business.Logic.Tests**: Unit tests to ensure business logic reliability.
- **ResourceAccess.Domains**: Domain models representing the data structures used across the application.
- **ResourceAccess.Mappers**: Responsible for mapping domain models to other formats required by the application.

- **ResourceAccess.Repositories**: Implementations for data retrieval and manipulation.
- **ResourceAccess.Database**: Contains data persistence mechanisms, managing interactions with the database.
- **CrossCutting.Exceptions**: Exception management logic for handling application-wide errors.
- **CrossCutting.Globalization**: Facilities for supporting multiple languages and regional settings.
- **CrossCutting.Utils**: Utility functions and helpers used across various layers of the application.
- **CrossCutting.Infrastructure**: Infrastructure-related configurations and setup.
- **CrossCutting.Enums**: Enums used across different layers for maintaining constants and fixed set of values.
- **ResourceAccess.Services**: Service layer interfacing between the business logic and presentation layers.
- **Presentation.ServiceApi**: Defines and hosts the Rest API endpoints.
- **Presentation.Desktop**: The graphical user interface built using WPF.

5. User Interface (WPF)

The WPF interface is designed for ease of use. It includes a text input area, buttons for performing actions (like sorting and counting), and displays for showing the results. The UI is responsive and intuitive, making it easy for users to interact with the application.

6. Rest API

The Rest API provides endpoints for text processing tasks. These include endpoints for retrieving sorted text and text statistics. The API is built for high performance and scalability.

7. Testing

The testing strategy includes unit tests covering the business logic. These tests ensure that the core functionalities of the application work as expected. Developers are encouraged to run these tests before making changes to the codebase.

8. Maintenance and Scalability

The application is designed for easy maintenance and scalability. Developers should follow best practices for code organization and adhere to the principles of clean code.

Conclusion and Reflections

This documentation provides an overview of the **Mecalux.TextProcessor** project, illustrating the application's architecture, features, and the development journey.

The project was designed and implemented with the following key objectives in mind:

- To create a robust and scalable text processing application using .NET 8 technologies.
- To provide a user-friendly experience through a WPF-based interface, and offer programmatic access via a Rest API.
- To adhere to best practices in software development, ensuring clean, maintainable, and testable code.

Throughout the development of this project, several challenges were encountered and overcome, including:

- Designing an application architecture that effectively separates concerns and promotes scalability.
- Implementing complex text processing logic that is both efficient and accurate.
- Ensuring that the user interface is intuitive and responsive, enhancing the overall user experience.

Key learnings from this project include:

- The importance of a well-structured project architecture in simplifying maintenance and future enhancements.
- The effectiveness of unit testing in catching bugs early and ensuring the reliability of core functionalities.
- The value of user feedback in refining the user interface and improving usability.