Number to Word Converter Project Document

Project Overview

This project involves creating a web application that converts numerical inputs into words using a .NET Core backend and a React TypeScript frontend. The decision to use this stack was influenced by factors such as flexibility, maintainability, and a rich development experience.

Selected Approach: .NET Core with React TypeScript

- **Backend and Frontend Separation**: By using .NET Core for the backend API and React TypeScript for the frontend, we achieve a clean separation of concerns. This modularity allows for better organization of code, making it easier to maintain and scale the application over time.
- **.NET Core Advancements**: .NET Core is a modern framework that provides high performance, cross-platform capabilities, and a rich set of libraries, making it a strong choice for building APIs.
- **React's Popularity**: React is one of the most popular frontend libraries, with a large community and extensive resources. This means better support for developers and a wealth of libraries and tools to enhance the application's functionality.
- **Independent Development**: The frontend and backend can be developed independently, allowing for different teams to work simultaneously, improving development efficiency.
- **TypeScript Benefits**: Using TypeScript in the React application provides static typing, which helps catch errors during development rather than at runtime. This enhances code reliability and reduces the likelihood of bugs, making the codebase easier to work with and maintain.
- **Model Consistency**: TypeScript allows for creating interfaces and types that can represent the data structures used across both the frontend and backend, ensuring consistency.
- **High Performance**: Both .NET Core and React are optimized for performance. .NET Core's asynchronous capabilities allow handling many concurrent requests efficiently, while React's virtual DOM optimizes rendering performance.
- **Scalability**: The chosen architecture supports horizontal scaling, allowing the application to handle increased load as needed, which is vital for future growth.

Reasons Against Using .NET Built-in React Template:

The built-in .NET React template enforces a pre-defined project structure that may not align with the specific needs or preferences of the development team, and limiting customization Additionally, using this template might restrict access to the latest React features or TypeScript configurations, while creating a React TypeScript application independently allows for the utilization of modern build tools that better suit our development workflow. Furthermore, building the application independently gives the opportunity of a deeper understanding of how React and TypeScript work

together, which is beneficial for team skill development, and offers the flexibility to adopt new technologies, ensuring the application remains modern and competitive.

Conclusion

The decision to utilize .NET Core with a React TypeScript frontend allows for a robust, flexible, and maintainable application architecture. By avoiding the built-in .NET React template, we have more control over the project structure, configurations, and tools, which ultimately enhances the development process and supports our long-term goals for scalability and performance.