Design and Approach Document for Number to Words Converter

By Isaiah Locke

Selected Approach

For this project, I chose to implement a **custom decimal-to-words conversion service** in C#. This service, NumberToWordsService, directly converts numerical values (up to quintillions and beyond) into English words, specifically formatted for currency with dollar and cent values. This approach prioritizes precision, efficiency, and showcases my own coding skills by handling all conversion logic without relying on external libraries or prebuilt models.

Reasons for the Selected Approach

1. **Focus on Demonstrating Core Skills**:
    - This project is intended to highlight my coding abilities rather than my use of external libraries or prebuilt language models. Implementing the solution entirely in C# allows me to demonstrate my understanding of algorithms, language syntax, and problem-solving without relying on third-party tools.
    - Building the solution from scratch lets me show how I handle complex number processing, language-specific formatting, and custom error handling.
2. **Efficiency and Simplicity**:
    - Implementing the converter as a custom service in C# provides me with direct control over number processing and formatting, allowing for efficient, fast conversions without additional dependencies or overhead.
    - By managing currency-specific requirements (like singular/plural handling for dollars and cents) within the code, I reduce complexity and avoid the need for external integrations.
3. **Precision and Customizability**:
    - This approach allows me to handle both whole and fractional parts of the number precisely, particularly for dollars and cents.
    - The solution is designed to divide numbers into standard groups (thousands, millions, billions, etc.), making it easy to expand or adapt for future needs, like other currencies or languages.
4. **Reliability and Local Execution**:
    - The solution is entirely self-contained within the .NET framework, with no reliance on external services or APIs, which ensures reliability and constant availability.
    - Running locally also guarantees predictable output without the risks of network dependency, latency, or unexpected external changes.

Alternative Solutions Considered

1. Using a Large Language Model (LLM)

An alternative approach I considered was using a large language model (LLM) for the number-to-words conversion. An LLM could theoretically handle flexible language-based conversions, potentially supporting very large numbers and nuanced formatting requirements.

Reasons Against Using an LLM

1. **Scope and Complexity**:
    - Integrating an LLM, particularly for this kind of task, would introduce unnecessary complexity. It would require setup, API integration, and custom handling to ensure consistent responses, adding time and overhead.
    - Given the project's timeline, I felt a custom solution was both more efficient and within scope.
2. **Dependency on External Services**:
    - Using an LLM generally requires a third-party API, adding external dependencies that could introduce latency, additional costs, and availability issues.
    - Relying on external services for a task this straightforward would increase operating costs and require managing API rate limits.
3. **Lack of Predictability for Large Values**:
    - While LLMs handle language tasks well, they may not consistently handle large numbers or currency-specific formatting rules as predictably as a custom solution.
4. **Focus on Demonstrating My Own Skills**:
    - An LLM-based solution would hide much of my individual code. By building a custom solution, I can show my own skills in structuring algorithms and handling specific formatting and data-processing needs.

2. Using a Numerical Library with Built-in Number-to-Words Conversion

Another approach I considered was using a third-party numerical or text-processing library that includes number-to-words conversion capabilities. Some libraries provide simple number-to-words conversion, potentially saving development time.

Reasons Against Using a Third-Party Numerical Library

1. **Limited Customizability**:
    - Many libraries provide generic number-to-words conversions without specific handling for currency, including things like "dollar" vs. "dollars" or other custom formatting needs.
    - Custom requirements, such as specific currency formatting, handling large numbers, and flexibility for edge cases, might not be supported by a third-party library.
2. **External Dependency**:

- Using a third-party library would introduce additional dependencies, making the project reliant on external updates, maintenance, and licensing.
- External libraries often require extra configuration and validation to ensure compatibility with .NET.

3. **Focus on Individual Capability**:
    - Building the solution in C# allows me to fully demonstrate my problem-solving skills without needing to rely on outside packages. This project shows my mastery of the C# language and my ability to design an efficient algorithm from scratch.
4. **Cost and Maintenance**:
    - Some libraries come with licensing fees, which would add to the project's cost. A custom solution eliminates any licensing costs and gives me full control over the code.

Conclusion

In the end, I chose a custom C# service for converting numbers to words because it's the most effective and efficient approach for this project. It provides complete control over the conversion logic, handles large numbers accurately, and is easy to maintain. While alternatives like an LLM or third-party libraries were considered, they introduce unnecessary complexity, dependencies, and costs. Implementing the solution directly in C# allows me to fully showcase my coding skills, delivering a precise, reliable solution without external dependencies.