Number to Words Converter

A modern web application that converts numerical values into their **written word equivalents**, formatted as proper currency with **dollars and cents**.
This project demonstrates **ASP.NET Core MVC**, **C# 12**, and clean software architecture principles with responsive, user-friendly design.

🎯 What is this tool?

The **Number to Words Converter** transforms numbers into written words.
For example:

- `123.45` → **ONE HUNDRED AND TWENTY-THREE DOLLARS AND FORTY-FIVE CENTS**
- `1000.00` → **ONE THOUSAND DOLLARS**
- `0.99` → **NINETY-NINE CENTS**

✨ Features

- **Decimal Support** – Handles numbers with decimal points (e.g., 123.45)
- **Currency Format** – Outputs in proper dollar & cent representation
- **Large Number Support** – Up to millions
- **Real-time Conversion** – Instant results as you type
- **Copy Function** – One-click result copy (📋)
- **Dark/Light Mode** – Toggle between themes
- **Responsive Design** – Mobile-first, works on all devices

📝 How to Use

1. **Enter a number** – Type any number in the input field (e.g., `123.45`)
2. **Click Convert** – Press the "Convert" button
3. **Copy the result** – Use the 📋 button to copy

💡 Examples

Input	Output
123.45	ONE HUNDRED AND TWENTY-THREE DOLLARS AND FORTY-FIVE CENTS
1000.00	ONE THOUSAND DOLLARS
0.99	NINETY-NINE CENTS

🏗️ Software Architecture & Design Decisions

🎯 Why ASP.NET Core MVC?

- **Robust Framework** – Enterprise-grade with built-in security & performance
- **High Performance** – Cross-platform, lightweight, efficient concurrency
- **Security First** – Request validation, CSRF protection, secure defaults
- **Separation of Concerns** – MVC pattern enforces clean architecture

💻 Why C# Instead of Other Languages?

- **vs JavaScript/Node.js** – Strong typing, performance, maintainability
- **vs Python/Django** – Faster, better Windows/enterprise integration
- **vs Java/Spring** – Modern syntax, better cross-platform support

🏛️ Architecture Principles

- **Single Responsibility Principle** – Each class has one responsibility
- **Dependency Inversion** – Modules depend on abstractions, not details
- **Separation of Concerns** – Clear boundaries across layers
- **Testability** – Components designed for unit testing

📊 System Flow

Flow:

1. User enters a number
2. Input validated with `decimal.TryParse()`
3. Split into **dollars & cents**
4. Each part converted to words
5. Output formatted:
    - `"XXX DOLLARS"` if cents = 0
    - `"XXX DOLLARS AND YY CENTS"` otherwise

🛠️ Technology Stack

Backend

- Framework: **ASP.NET Core 9.0**
- Language: **C# 12.0**
- Pattern: **MVC**
- Architecture: **Layered Architecture**

Frontend

- Views: **Razor (HTML5)**

- Styling: **CSS3** + Custom Design System
- Interactivity: **Vanilla JavaScript**
- Responsive: **Mobile-First Design**

Development

- IDE: **Visual Studio / VS Code**
- Package Manager: **NuGet**
- Version Control: **Git**
- Deployment: **Cross-platform (.NET Core)**

🎨 Design Philosophy

- 🧹 **Clean Code** – Readable, maintainable, best practices
- 🔧 **Maintainability** – Easy to extend & debug
- 📈 **Scalability** – Built for growth in data & features
- 🎯 **User Experience** – Simple, accessible, intuitive

🚀 Getting Started

Prerequisites

- .NET SDK 9.0+
- Visual Studio 2022 / VS Code
- Git

Run Locally

```
# Clone repository
git clone https://github.com/ChienHuang0818/Amount2Words.git

# Navigate to project
cd Amount2Words

# Run the app
dotnet watch run
```

Visit: `http://localhost:5087/Convert`
