# **App overview**

A web page with a backend web server that converts numerical input into words and passes these words as a string output parameter onto the web page.

# **Aims/Objective**

To convert numerical monetary values to the words.

# **Functional requirements**

The application should have the following functionalities:

* Receive numerical input from the user
* Allow the user select the country currency [AUD, USD, NGN]
* Convert the numerical input to words
* Display the words on the web page with the selected currency

**Test plan** – wrong input, out of limit input, correct input, etc?

Acceptance testing:

* Can the application receive user input?
  + Correct input
  + Wrong or out of bounds input – Is a warning message shown to the user?
* Can the user select from the list of currencies?
* Can the application convert the numbers to words?
  + Edge numbers e.g. 99, 999, 99.99, 100, 001, 001.00
  + Correct usage of the singular and plural form of currency

**Design and development plan**

\*method (tdd/bdd, ddd, why?), architecture (ntier, why?), design pattern (builder?, factory, why), schematic diagrams?\*

\*Reasons for approach (bdd, tdd, specflow, full stack system v api system, github v azure devops, azure app service why?\*) and why you decided against other possible solutions\*

\*[Readme.md document containing instructions on how to build, host and interact your solution]\*

\*What methodology is used?\*

\*What type of architecture is used?\*

\*Which design patterns are used?\*

\*[Schematic diagram of app architecture]\*

\*[Class diagram]\*

Classes:

Services

IText - Value

Text (Singular, Plural) - Value

ITextFactory – GetSingularForm, GetPluralForm\* Extension method used

Currency – Country, Code, MajorName, MinorName

Number – Digits, Word, MinimumPlaceValue (Tag), GetWord

BasicNumber

TensNumber (With tags)\*

Models:

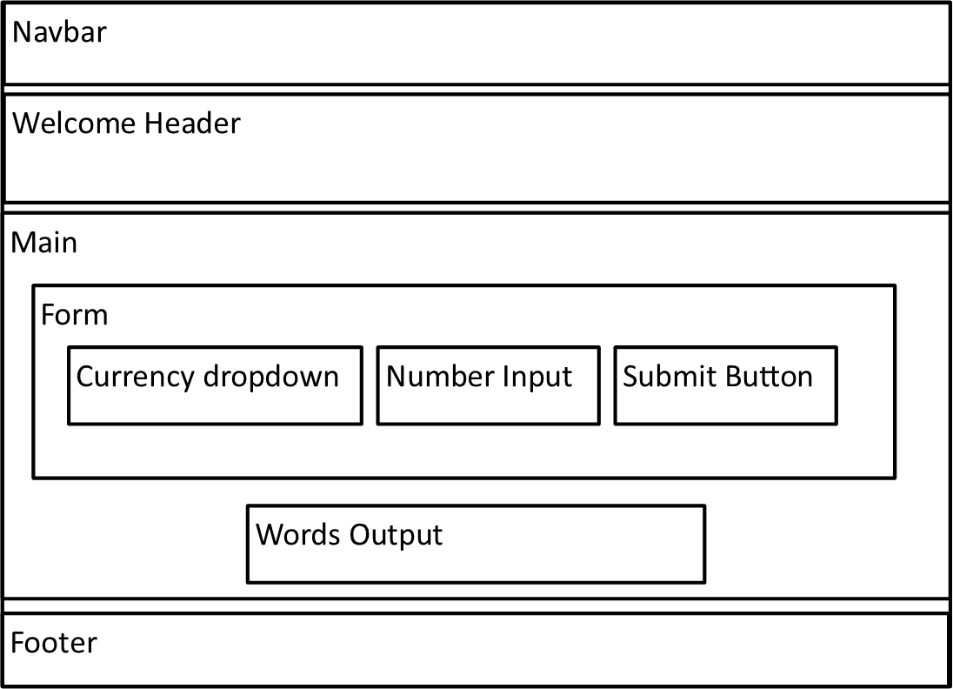
Data:

The application is built with C# and Blazor on .NET 8, using a full stack approach to speed up development. The API approach is not used because there are no external systems that need to be connected yet. For future development, services can be transformed into APIs. The methodology used is the Test Driven Development (TDD) methodology, to ensure quality application test coverage is achieved and the requirements are met. For CI/CD, the code is maintained in a GitHub (Azure DevOps\*) repository and deployed to Azure App Service.

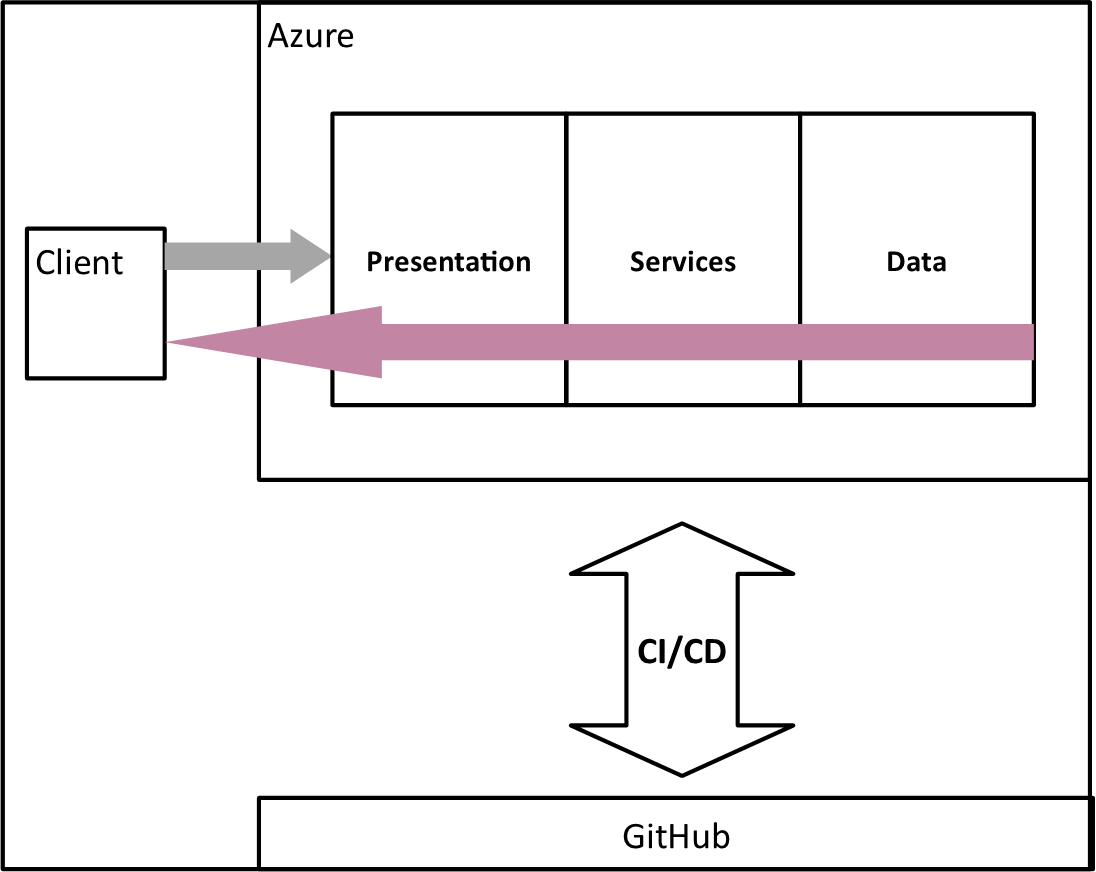
The application follows an N-Tier layered architecture, with the following simplified layers:

* Presentation – The application frontend and start-up project of the solution.
* Services – The application business logic and services. This is also a class library project in the solution.
* Data – The application data layer. For simplicity, an AppData class is used\*

**Frontend Wireframe:**



**Schematic/architectural\* diagram:**



Range of numeric value:

* long (64-bit signed integer): Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807; Default value: 0L

Currency Data:

* Australian Dollar(s) (AUD) (Cent(s))
* Nigerian Naira (NGN) (Kobo)
* United States Dollar(s) (USD) (Cent(s))

Words Data (Dictionary):

1. 0 = Zero\*
2. 1 = One
3. 2 = Two
4. 3 = Three
5. 4 = Four
6. 5 = Five
7. 6 = Six
8. 7 = Seven
9. 8 = Eight
10. 9 = Nine
11. 10 = Ten
12. 11 = Eleven
13. 12 = Twelve
14. 13 = Thirteen
15. 14 = Fourteen
16. 15 = Fifteen
17. 16 = Sixteen
18. 17 = Seventeen
19. 18 = Eighteen
20. 19 = Nineteen
21. 20 = Twenty
22. 30 = Thirty
23. 40 = Forty
24. 50 = Fifty
25. 60 = Sixty
26. 70 = Seventy
27. 80 = Eighty
28. 90 = Ninety
29. 100 = Hundred
30. 1,000 = Thousand
31. 1,000,000 = Million
32. 1,000,000,000 = Billion
33. 1,000,000,000,000 = Trillion
34. 1,000,000,000,000,000 = Quadrillion
35. 1,000,000,000,000,000,000 = Quintillion

Number to Words Algorithm:

* Split into two based on ***decimal place***
* To the right of decimal place is cent value
* To the left of decimal place is dollar value
* For cent value, if 1, use singular for currency, else use plural
* For dollar value, if 1, use singular for currency, else use plural
* For cent value, if present or not zero, **[Limit is 2 decimal places]**:
  1. **If value greater than 20:**
     1. **Modulus value by 10, if result not zero, then check result from data and return “-{result}” (B)**
     2. **Subtract modulus result of previous step from initial value, \*if not zero\*, then check result from data and return “ *and* {result}” (A)**
     3. ***Divide by 10, save integer quotient and multiply by 10, then check result from data and return “ and {result}”***
     4. ***Multiply integer quotient from previous step by 10 and subtract from initial value, if not zero, then check result from data and return “ {result}” to be appended to text from previous step***
  2. **Else, if 20 or less than, but not zero:**
     1. **Check result from data and return “ and {result}” (A)**
* **Final string text for cent value should be “(A)(B)”**
* For dollar value, if present or not zero:
  1. Divide numbers into groups of three by dividing and modulusing by 1000, until integer quotient from division is 0
  2. 1st group has tag 0, 2nd group tag 3, 3rd group tag 6, 4th group has tag 9, 5th group has tag 12, 6th group has tag 15. **[Limit is 6 groups (tag 15)]**
  3. In each group, if value greater than 99:
     1. Modulus value by 100, then save answer
     2. **If answer of previous step greater than 20:**
        + **Modulus answer of step II by 10, if not zero, then check result from data and return “-{result}” (C)**
        + **Subtract modulus result of previous step from initial answer of step II, \*if not zero\*, then check result from data and return “ and {result}” (B)**
     3. **Else, if answer is 20 or less than, but not zero:**
        + **Check answer from data and return “ and {result}” (B)**
     4. Subtract modulus answer of step I from initial value of parent step 3 and save result
     5. Divide result of previous step by 100, save integer quotient, then check result from data and return “ {result} hundred” (A)
     6. Final string text for cent value should be “(A)(B)(C)”
  4. Else
     1. **If value of parent step 3 greater than 20:**
        + **Modulus value by 10, if result not zero, then check result from data and return “-{result}” (B)**
        + **Subtract modulus result of previous step from initial value of parent step 3, \*if not zero\*, then check result from data and return “ {result}” (A)**
     2. **Else, if 20 or less than, but not zero:**
        + **Check result from data and return “ {result}” (A)**
     3. Convert tag into words: tag with 0 is skipped, value of all other tags are gotten from data.
     4. Final string text for cent value should be “(A)(B)”