CONVERSOR DE CIFRAS NUMÉRICAS A TEXTO EN POLACO

Grado en Ingeniería Informática

Aridai Santana Gil

02/04/2022

Tutorizado por:
Francisco Javier Carreras Riudavets

Índice

1. Estado actual y objetivos iniciales.

El objetivo principal de este trabajo consiste en implementar un servicio web que presente la funcionalidad de conversión de cifras numéricas a texto en polaco. En nuestro día a día es muy frecuente que escribamos cifras numéricas y, a pesar de que es más frecuente que las escribamos como dígitos, existen situaciones particulares en las que se deben escribir las cifras numéricas como texto.

Antes de adentrarnos en el tema y explicar detenidamente en qué situaciones se precisa escribir cifras numéricas como texto, analizaremos las ideas que están detrás de la formación de números mediante el uso de las palabras.

1.1. Conversión de números a texto en polaco.

Como ocurre con una gran variedad de idiomas (entre ellos el español), la conversión de cifras numéricas a texto se puede automatizar pues, los idiomas suelen seguir un patrón lógico o una serie de convenciones a la hora de formar el texto que representa a los números. Obviamente existen ciertas excepciones pues, la forma en la que los humanos nos comunicamos no siempre se ajusta a un patrón perfectamente definido.

Los números en polaco, al igual que en una gran variedad de idiomas, se pueden escribir de distintas maneras, ya sea cardinal, ordinal, fraccionario, multiplicativo, etc.

En los siguientes apartados examinaremos las maneras que más se utilizan para escribir una cifra numérica en texto en polaco.

1.1.1. Números cardinales.

El tipo de número comprendido como cardinal expresa la cantidad de elementos de un conjunto. A continuación, se muestran en la Tabla 1 los números del cero al diecinueve escritos en texto en polaco.

0	zero	**5**	pięć	**10**	dziesięć	**15**	piętnaście
1	jeden	**6**	sześć	**11**	jedenaście	**16**	szesnaście
2	dwa	**7**	siedem	**12**	dwanaście	**17**	siedemnaście

3	trzy	**8**	osiem	**13**	trzynaście	**18**	osiemnaście
4	cztery	**9**	dziewięć	**14**	czternaście	**19**	dziewiętnaście

Tabla 1– Números Cardinales 0-19

Como podemos observar si analizamos brevemente los números en la Tabla 1. Las cifras del cero al nueve tienen nombres específicos, en cambio, los números del once al diecinueve se forman empezando por la unidad, directamente seguida por **naście**, de na (y) y dziesięć (diez), escribiéndose el número once literalmente como *uno y diez*.

A partir de la primera decena, las decenas se forman poniendo la cifra multiplicadora antes de una forma de la palabra diez, excepto para diez mismo como muestra la Tabla 2.

20	dwadzieścia	**60**	sześćdziesiąt
30	trzydzieści	**70**	siedemdziesiąt
40	czterdzieści	**80**	osiemdziesiąt
50	pięćdziesiąt	**90**	dziewięćdziesiąt

Tabla 2 – Decenas Cardinal

Como podemos observar, las decenas comprendidas entre veinte y cuarenta se escriben con una terminación parecida (dzieści) y, en cambio, las decenas comprendidas entre cincuenta y noventa se escriben con otra terminación (dziesiąt). Esto es algo que veremos en prácticamente todos los tipos de números, pues es parte de la manera en la que se forma el singular y el plural en polaco debido al uso de las declinaciones. Por lo tanto, y para remarcar este detalle, los cifras que acaban en valores comprendidos entre dos y cuatro son un tipo de plural distinto a las cifras acabadas en valores comprendidos entre cinco y nueve.

Existe la creencia de que esa gran diferencia entre plurales se debe en gran parte a dos sucesos históricos, el primero de ellos se trata del método más primitivo de contar que es mediante el uso de los dedos de las manos. El segundo está basado en el sistema romano de numeración "i, ii, iii, iv"; el cinco era el contorno de la mano simplificado en una "v". Estos sucesos pueden explicar, en cierta medida, por qué el número cinco era tan fundamental en el polaco y la manera actual de formar el plural.

Los números compuestos más grandes que veinte se forman empezando por la decena, y luego la unidad separada por un espacio, como observamos en la Tabla 3.

24	dwadzieścia cztery	**29**	dwadzieścia dziewięć
38	trzydzieści osiem	**77**	siedemdziesiąt siedem
43	czterdzieści trzy	**81**	osiemdziesiąt jeden
55	pięćdziesiąt pięć	**92**	dziewięćdziesiąt dwa

Tabla 3 – Ejemplos de números cardinales compuestos

Las centenas se forman poniendo la cifra multiplicadora antes de una forma de la palabra para cien (ście, sta o set) sin espacio, excepto para cien mismo como muestra la Tabla 4.

100	sto	**400**	czterysta	**700**	siedemset
200	dwieście	**500**	pięćset	**800**	osiemset
300	trzysta	**600**	sześćset	**900**	dziewięćset

Tabla 4 - Centenas cardinal

Los miles se forman poniendo la cifra multiplicadora antes de una forma de la palabra para mil (*tysiąc*, *tysiące* o *tysięcy*) como observamos en la Tabla 5.

1000	tysiąc	**4000**	cztery tysiące	**7000**	siedem tysięcy
2000	dwa tysiące	**5000**	pięć tysięcy	**8000**	osiem tysięcy
3000	trzy tysiące	**6000**	sześć tysięcy	**9000**	dziewięć tysięcy

Tabla 5 - Millares cardinal

La lengua polaca usa la convención de nomenclatura de la escala larga. Algunas de las palabras para los números grandes se escriben de la manera que se muestra en la Tabla 6.

Número escrito	Notación científica
Bilion	10^{12}
Biliard	10^{15}
Trylion	10^{18}
Tryliard	10^{21}
Kwadrylion	10^{24}
Kwadryliard	10^{27}

Tabla 6 – Números cardinales grandes

Para resumir todo lo visto en los párrafos anteriores me parece acertado visualizar algunos ejemplos donde se junta todo lo expuesto:

- **1 365 124:** Milion trzysta sześćdziesiąt pięć tysięcy sto dwadzieścia cztery.
- **2 223 785 956:** Dwa miliardy dwieście dwadzieścia trzy miliony siedemset osiemdziesiąt pięć tysięcy dziewięćset pięćdziesiąt sześć.
- **425 829 983 283:** Czterysta dwadzieścia pięć miliardów osiemset dwadzieścia dziewięć milionów dziewięćset osiemdziesiąt trzy tysiące dwieście osiemdziesiąt trzy.

Como podemos observar, la estructura a la hora de escribir los números en polaco es muy parecida a la manera que tenemos en español aunque con ligeras diferencias. Si nos fijamos, cuando hay un único millón no se escribe "**un** millón trescientos sesenta y cinco mil…" sino que directamente se escribe "milion" porque en polaco se deduce que se trata de tan solo uno si no se incluye una cifra multiplicadora antes.

1.1.2. Números ordinales.

Los números ordinales expresan orden o sucesión en relación con los números naturales. Además, indican la posición que ocupa el elemento al que se refieren dentro de una serie ordenada.

1	pierwszy	**6**	szósty	**11**	jedenasty	**16**	szesnasty
2	drugi	**7**	siódmy	**12**	dwunasty	**17**	siedemnasty
3	trzeci	**8**	ósmy	**13**	trzynasty	**18**	osiemnasty
4	czwarty	**9**	dziewiąty	**14**	czternasty	**19**	dziewiętnasty
5	piąty	**10**	dziesiąty	**15**	piętnasty		

Tabla 7 – Números ordinales 1-19

Como podemos observar en la Tabla 7, los números ordinales del cero al diecinueve siguen cierta correlación con los números cardinales pues, al igual que con los números cardinales, las cifras del uno al nueve tienen nombres específicos, en

cambio, los números del once al diecinueve se forman empezando por la unidad, directamente seguida por **nasty**, de na (y) y dziesiąty (décimo).

A partir de veinte, las decenas se escriben como en la Tabla 8.

20	dwudziesty	**60**	sześćdziesiąty
30	trzydziesty	**70**	siedemdziesiąty
40	czterdziesty	**80**	osiemdziesiąty
50	pięćdziesiąty	**90**	dziewięćdziesiąty

Tabla 8 – Decenas ordinales

Después de analizar brevemente los números ordinales podemos deducir que no varía enormemente la estructura con la que se forman, comparándolos con los números cardinales, pero sí que varían de forma notable con la manera en la que se escriben. Las principales diferencias que existen con los números cardinales (del once al diecinueve y las decenas) radican en la declinación que se emplea para la palabra diez.

En el idioma polaco, cuando se pretende escribir un número ordinal con varias cifras numéricas se tiende a escribir como ordinal la última cifra o las últimas dos en su defecto, excepto cuando son números “redondos” como podemos observar en la Tabla 9.

Cifra numérica	Número escrito
523	pięćset dwadzieścia trzy
1350	tysiąc trzysta pięćdziesiąty
5200	pięć tysięcy dwusetny
1234534	milion dwieście trzydzieści cztery tysiące pięćset trzydziesty czwarty

Tabla 9 – Números ordinales ejemplos

1.1.3 Fracciones.

Una fracción está compuesta por dos partes, el numerador y denominador. En caso del polaco, al igual que muchos otros idiomas, el numerador se trata de un número cardinal. Sin embargo, el denominador no sigue las peculiaridades de otros idiomas debido a que no se escribe como un ordinal sino que utiliza distintas declinaciones.

Por ejemplo, podemos observar en la Tabla 10, que el número “5/8” recibe para el numerador un nombre de un número cardinal pero, en cambio, el denominador no se escribe como “ósmy” en ordinal sino que se escribe como “ósmych”.

Cifra numérica	Número escrito
$^{5}/_{8}$	pięć ósmych
$4/5$	cztery piąte
$^{2}/_{3}$	dwie trzecie

Tabla 10 – Ejemplo de fracciones

No es habitual comunicar oralmente fracciones que presenten cantidades numéricas muy grandes, esto se debe a la dificultad que suele existir en los idiomas para separar el numerador del denominador cuando se está hablando. En caso de que esto sea necesario se puede extender la norma que se sigue en la Tabla 10 y aplicarse a cualquier número.

1.1.4. Números fraccionarios.

Este tipo de números expresa la división de un todo en sus distintas partes, por lo tanto, un número fraccionario se puede entender como una porción o pedazo de algo más grande.

Por ejemplo en español, si deseamos expresar que nos hemos comido la porción número veinticinco de una tarta podríamos decir que nos comido la veinticincoava parte de la tarta.

En el idioma polaco los números fraccionarios se escriben de forma muy parecida a los números ordinales, cambiando solamente su declinación a la hora de formarse.

1.1.5 Números decimales.

Los números decimales son una manera de representar una fracción evitando la necesidad de escribir el numerador, denominador y la barra divisoria. Una fracción ejemplo como puede ser “1/4” se podría representar de forma decimal con “0.25” pues éste es el resultado que se obtiene al dividir uno entre cuatro. El punto, en este caso,

indica que se trata de un número decimal y cualquier número que sea menor a la unidad será representado con un cero delante de una coma y luego las cifras decimales correspondientes.

Si deseamos expresar de forma escrita un decimal donde su parte entera sea mayor a cero lo podemos hacer escribiendo primero la parte entera como un cardinal, luego el unificador “i” que es el equivalente a nuestra “y” en español y luego la parte decimal. Es decir el número “5.7” se puede escribir como “pięć i siedem dziesiątych”. Si hasta ahora hemos entendido cómo se forman los números en polaco nos resultará bastante peculiar de dónde sale ese “dziesiątych”. Esto se debe a que en el idioma polaco, como podemos observar en la ilustración 1 y 2, los números decimales deben indicar el orden del decimal (décimas, centésimas, etc.) Su equivalencia en español sería decir “ochenta y tres con ciento nueve milésimas”.

Ilustración 1 - Decimales uno

Ilustración 2 - Decimales dos

Como podemos observar en la Tabla 11, A partir de las milésimas se empieza a repetir un patrón, pues vendrían las “diez milésimas” cuya equivalencia en polaco es “dziesięciotysięcznych” luego las “cien milésimas” con su correspondiente equivalencia

en polaco “stutysięcznych” y posteriormente la millonésima, “milionowych” en polaco. Esto se aplicaría continuamente para números más grandes.

POSICIÓN DE LOS DÍGITOS													
milionów	**setek tysięcy**	**dziesiątek tysięcy**	**tysięcy**	**setek**	**dziesiątek**	**jedności**	**_**	**dziesiątych**	**setnych**	**tysięcznych**	**dziesięciotysięcznych**	**stutysięcznych**	**milionowych**
7	6	5	4	3	2	1	0	1	2	3	4	5	6

Tabla 11 – Posición de los dígitos

A continuación se listan varios ejemplos:

- 3.9 → trzy i dziewięć **dziesiątych** (1 dígito decimal).
- 12.34 → dwanaście i trzydzieści cztery **setne** (2 dígitos decimales).
- 25.332 → dwadzieścia pięć i trzysta trzydzieści dwa **tysięczne** (3 dígitos decimales).
- 500.2383 → pięćset i dwa tysiące trzysta osiemdziesiąt trzy **dziesięciotysięczne** (4 dígitos decimales).
- 1215.00345 → tysiąc dwieście piętnaście i trzysta czterdzieści pięć **stutysięcznych** (5 dígitos decimales).
- 2.485038 → dwa i czterysta osiemdziesiąt pięć tysięcy trzydzieści osiem **milionowych** (6 dígitos decimales).

Si la parte entera es el número cero se tiende a obviarlo y escribir directamente la parte decimal con su correspondiente orden decimal (décima, centésima, etc.).

1.1.6. Números negativos.

Los números negativos se utilizan para expresar cantidades menores que cero.

En el idioma polaco, para nombrar números negativos basta con añadir la palabra "minus" al principio de la oración, el resto del contenido no necesita ser modificado. Por lo tanto, el número "-6.3" lo podríamos escribir como "minus sześć i trzy dziesiąte".

1.1.7. Números multiplicativos.

Los números multiplicativos expresan el número de veces que se da o se repite cierta cosa, por lo tanto, son números que expresan multiplicación.

En el idioma polaco, si queremos que un número sea multiplicativo éste deberá ser positivo y entero, además para los números del uno al diez se incluirá su expresión correspondiente (p.ej. doble, triple, cuádruple, etc.). A partir del número diez se incluye el número en su forma cardinal más la palabra "razy" al final. Por ejemplo la forma multiplicativa del cardinal "15" sería "piętnaście razy".

1.1.8. Nombres de números largos.

A partir de ciertas cifras es inevitable que el ser humano tienda a utilizar patrones para formar las palabras que representen a ese número, de lo contrario sería imposible poder nombrar a los números grandes pues estos aumentan de forma exponencial. Dichos números no son utilizados en el día a día, aunque resultan de utilidad en algunas ciencias que buscan la precisión de los datos o que manejan magnitudes cósmicas; por lo tanto, se han recopilado algunos de los números más largos encontrados en libros y diccionarios.

El número cardinal más grande que se ha encontrado en esta investigación (Classicistranieri, s.f.), como se puede observar en la Tabla 12, se trata del "nonacentyliard" el cual equivale a 10^{5400}. Teniendo en cuenta que el número aproximado de átomos en el universo es de 10^{82} nos podemos hacer una idea de la increíble magnitud de dicho número.

Nombre	Notación científica
milion	10^{6}
miliard	10^{9}
bilion	10^{12}

biliard	10^{15}
trylion	10^{18}
tryliard	10^{21}
kwadrylion	10^{24}
kwadryliard	10^{27}
kwintylion	10^{30}
kwintyliard	10^{33}
sekstylion	10^{36}
sekstyliard	10^{39}
septylion	10^{42}
septyliard	10^{45}
oktylion	10^{48}
oktyliard	10^{51}
nonilion	10^{54}
noniliard	10^{57}
decylion	10^{60}
decyliard	10^{63}
undecylion	10^{66}
undecyliard	10^{69}
dodecylion	10^{72}
dodecyliard	10^{75}
tridecylion	10^{78}
tridecyliard	10^{81}
kwatuordecylion	10^{84}
kwatuordecyliard	10^{87}
kwindecylion	10^{90}
kwindecyliard	10^{93}
seksdecylion	10^{96}
seksdecyliard	10^{99}
septendecylion	10^{102}
septendecyliard	10^{105}
oktodecylion	10^{108}
oktodecyliard	10^{111}
nowemdecylion	10^{114}
nowemdecyliard	10^{117}
…	…
…	…
nowemcentyliard	10^{657}
seksacentylion	10^{3600}
nonacentyliard	10^{5400}

Tabla 12 – Nombre de números grandes

Los números grandes en polaco se forman mediante cuatro componentes. El primero de ellos, se trata de un prefijo que expresa multiplicación (try, kwadr, kwint, seks, sept, okt, nowem). El segundo componente es un segundo prefijo que se empieza aplicar después del nonillón (10^{54}) para poder crear los siguientes números. El tercer componente varía entre “y”, “ty”, “i”, “cy”, “agi”, “ginty” y ayuda a la formación de la palabra y, por último, el cuarto componente varía entre dos opciones “lion” y “liard” lo

cual se utiliza para establecer orden pues recurrentemente todos los números que incrementen de 10^6 en 10^6 (milion, bilion, trilion, kwadrylion) acaban con “lion” y el resto acaba con liard (miliard, biliard, triliard, kwadryliard). En español sería igual que la diferencia entre los millardos y el millón.

1.2. Normas para escoger si escribimos dígitos o el equivalente en palabras.

Escribir números con palabras es una práctica que no solemos realizar de forma común pues en muchas circunstancias es más rápido e intuitivo utilizar dígitos. Sin embargo, existen situaciones donde usar palabras para expresar números puede esclarecer el significado del mensaje y en otras situaciones puede ser de carácter obligatorio.

¿Cuáles son dichas situaciones? – En los siguientes apartados nos encargaremos de listar tales situaciones dando así respuesta a la pregunta.

1.2.1. Cuándo escribir con palabras.

- La regla general consiste en que si se trata de un género de ficción (p. ej. novelas) se escriben siempre las cifras numéricas con palabras. En cambio, si se trata de un género que no es ficticio se escriben con palabras solamente los números más pequeños, comprendiendo estos como números entre 0-9, 0-99 o incluso 0-999. Se tiende a escribir los números como dígitos para números más grandes (Bańko, 2012).

- En caso de que un número se encuentre al principio de una frase se escribe con palabras.
    1. **Doce (12)** personas han dado positivo en COVID-19.
    2. **Dwanaście (12)** osób uzyskało pozytywny wynik testu na obecność COVID-19.

- Hacer uso de las palabras cuando estemos en un ámbito de carácter legal o en documentos oficiales dónde sea de vital importancia la claridad de la cifra

numérica que se expresa. (p. ej. Skradziono z niego dwadzieścia pięć milionów złotych).

- Emplear palabras cuando se escriban documentos dentro del sector financiero, como por ejemplo, cheques bancarios pues es necesario dejar constancia escrita para evitar cualquier tipo de confusión que pueda dar lugar a errores económicos.

- Hacer uso de las palabras cuando un nombre contenga números ordinales o en caso de que una calle presente un nombre numérico. (p. ej. Trzecia Rzesza, Szósta Aleja).

- Los números que sean considerados como números grandes (mayores que cien o incluso mil) pero que puedan escribirse con una o dos palabras también se escriben con palabras.

1.2.2. Cuando escribir con dígitos.

- Expresar una cantidad de dinero. Se utilizan dígitos para expresar cantidades exactas de dinero (p. ej. 58.2 zł).

- Dar un dato estadístico. Los datos estadísticos se tienden a escribir con dígitos.

- Representar el nombre de una carretera. Independientemente del tipo de carretera ésta se puede representar con dígitos (p. ej. “A20”, “M50”).

- Escribir la fecha. Las fechas pueden escribirse con cifras.

- Medidas. Se emplean los dígitos para representar medidas (p.ej. 12 kg).

- Direcciones. Se utilizan dígitos en las direcciones (p. ej. Floriańska 32, 31-021 Kraków, Polonia).

- Periodos. Expresar en dígitos periodos es conveniente (p. ej. str. 34-67).

- Secciones de un libro. Utilizar dígitos para remarcar en qué capitulo, volumen o sección nos encontramos.

- Fracciones y decimales. Está a nuestra disposición el si deseamos escribir la fracción como dígito o como número, hay que tener en cuenta que si la fracción es muy grande como norma general se escribirá en dígitos. Las cantidades decimales se tienden a expresar en dígitos, escribiendo las cantidades exactas (p.ej. 0.324).

1.3 Objetivos a cubrir en el proyecto.

Como hemos podido observar a lo largo de los primeros párrafos y de la introducción inicial, es más que necesario el conocer cómo se escriben los números. Escribir números con pocas cifras numéricas es una tarea muy sencilla para cualquier persona que tenga un mínimo de conocimiento del idioma, pero si nos disponemos a escribir números mayores que cien o mil que por norma general no son comunes de escribir, entonces la dificultad se incrementa de forma notoria. En muchas ocasiones, ni los propios hablantes nativos de un idioma, e incluso filólogos expertos en un idioma concreto, son capaces de saber cómo se escriben los números más grandes pues éste es un problema del que solo te percatas cuando te enfrentas a él.

Lógicamente es una práctica poco común pero que, como pudimos observar en el apartado de “normas para escoger si escribimos dígitos o el equivalente en palabras”, a veces se hace una práctica necesaria.

Debido a todos los motivos previamente mencionados y también por otros como, por ejemplo, la mera curiosidad de saber cómo se escribiría cierto número se ha decidido ofrecer un servicio web que cubra los siguientes objetivos:

- Apoyar a otras aplicaciones informáticas o ser utilizado por distintas aplicaciones informáticas.

- Generar una herramienta que permita comprobar la ortografía en la escritura de números.

- Mostrar cómo concatenar de forma correcta las palabras que forman un número sin importar su tamaño.

- Informar de los distintos procedimientos para escribir un número con el uso de palabras.

2. Justificación de las competencias específicas cubiertas.

Durante la realización de este trabajo se han cubierto las áreas o competencias específicas que se detallan a continuación.

2.1. Trabajo de fin de grado.

TFG01: Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

Este trabajo se ha realizado de forma completamente individual por parte del autor encontrándose en todo momento bajo la supervisión de su tutor. Además se han aplicado conceptos y metodologías que se han visto a lo largo de la carrera, sobre todo en el sector de la programación orientada a objetos.

2.2. Ingeniería del Software.

IS01: Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

Durante el desarrollo de este trabajo se han aplicado en la medida de lo posible todas las buenas prácticas y estándares que sigue la ingeniería de software, obteniendo un conjunto de resultados que se obtienen en un periodo de tiempo razonable con un código modular, reutilizable y escalable.

2.3. Comunes a la ingeniería informática.

CII06: Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.

El planteamiento y diseño de la solución del problema ha sido clave para poder desarrollar una solución eficiente y escalable. Una gran parte del tiempo utilizado para la realización de este trabajo se ha centrado en la búsqueda de las mejores prácticas y soluciones posibles para cumplir con la resolución del problema original.

CII07: Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema.

Para diseñar las estructuras de los datos se han tenido en cuenta distintos elementos, tales como:

- El consumo de recursos.
- Tiempo de ejecución.
- Estructuración lógica y organizada de la información.

CII014: Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.

Como se ha mencionado anteriormente, nos hemos centrado en garantizar unos mínimos de calidad en cuanto al tiempo de ejecución y la eficiencia del servicio, para ello, y ciñéndonos a un escenario donde cientos o incluso miles de usuario utilizan este servicio nos hemos decantado por el uso e implementación de técnicas de programación paralela. Estas técnicas proporcionan un mejor tiempo de respuesta en el servicio web.

2.4. Sistemas de información

SI01: Capacidad de integrar soluciones de tecnologías de la información y las comunicaciones y procesos empresariales para satisfacer las necesidades de información de las organizaciones, permitiéndoles alcanzar sus objetivos de forma efectiva y eficiente, dándoles así ventajas competitivas.

Un servicio web no deja de ser un tipo de tecnología que emplea un conjunto de protocolos y estándares que permiten el intercambio de datos entre aplicaciones. Diferentes aplicaciones de software, independientemente del lenguaje de implementación en el que estén escritas y la plataforma en la que se ejecute, puede emplear servicios web para intercambiar distintos datos en redes de ordenadores como internet. El servicio que se ha desarrollado para este trabajo tiene la intención de cubrir necesidades que pueden darse en organizaciones de distintos dominios (p. ej. Educativo, social, económico).

2.5. Tecnologías de la información.

TI06: Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.

Como hemos ido comentando durante lo largo de las justificaciones de las competencias específicas cubiertas, se ha desarrollado un servicio web durante la realización de este trabajo, además también se ha desarrollado un cliente responsivo para poder consumir el servicio. Por lo tanto, se dispone de la capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red.

3. Aportaciones.

Podemos decir que con este trabajo se hacen aportaciones en los siguientes dominios:

- **Educativo.** Este es el dominio que más se ve beneficiado o al que más le puede aportar la creación de este servicio web. El servicio web puede ser de gran utilidad para estudiantes del idioma polaco aunque no son los únicos usuarios potenciales, sino que este servicio también puede resultar útil para nativos del propio idioma pues en muchas ocasiones, ni los propios hablantes nativos de un idioma, e incluso filólogos expertos en un idioma concreto, son capaces de saber cómo se escriben los números más grandes pues éste es un problema del que solo te percatas cuando te enfrentas a él. Lógicamente es una práctica poco común pero que, como pudimos observar en el apartado de "normas para escoger si escribimos dígitos o el equivalente en palabras", a veces se hace una práctica necesaria.

- **Económico.** Este servicio web puede llegar a ser muy útil a la hora de rellenar cheques bancarios, debido a que es exigido el indicar los importes tanto con números como con letras.

- **Jurídico/Administrativo.** En documentos de carácter oficial o donde se maneje la legalidad de algo se tiende a utilizar números escritos con palabras pues la intención principal es la claridad de esa cifra numérica, la cual debe ser perfectamente comprendida para evitar malentendidos.

- **Social.** Este servicio web puede utilizarse conjuntamente con aplicaciones que mejoren la accesibilidad mediante el uso de un conversor automático de números escritos con texto a voz. Esta mejora de la accesibilidad puede beneficiar a cierto sector de la sociedad como las personas con discapacidades visuales o de cualquier índole.

- **Complemento a la inteligencia artificial (IA).** Este servicio web puede funcionar en conjunto con cualquier aplicación que emplee inteligencia artificial para brindar la funcionalidad de asistente de voz (p.ej. Siri, Alexa, Google de Android, etc.). Debido a que las respuestas que llegan a estos asistentes de voz están escritas con cifras numéricas, la inteligencia artificial necesitaría la conversión a palabras para poder leer la respuesta correctamente al usuario final.

Una comprobación de la demanda que existe para este tipo de servicios la podemos observar en la web llamada “Números TIP”, la cual ha sido desarrollada por los miembros de la División de Lingüística Computacional del Instituto Universitario de Análisis y Aplicaciones Textuales de la ULPGC. A continuación, en las ilustraciones 3 y 4, se muestran estadísticas obtenidas por dicha página durante el año 2021.

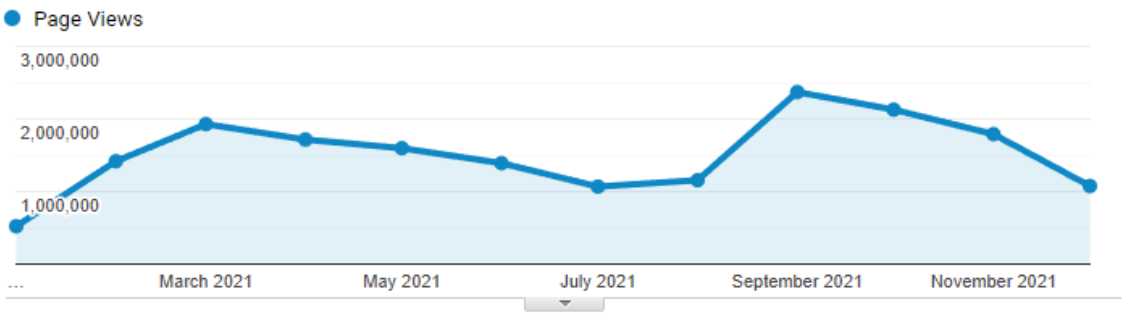


Ilustración 3 - Visitas de las páginas por mes

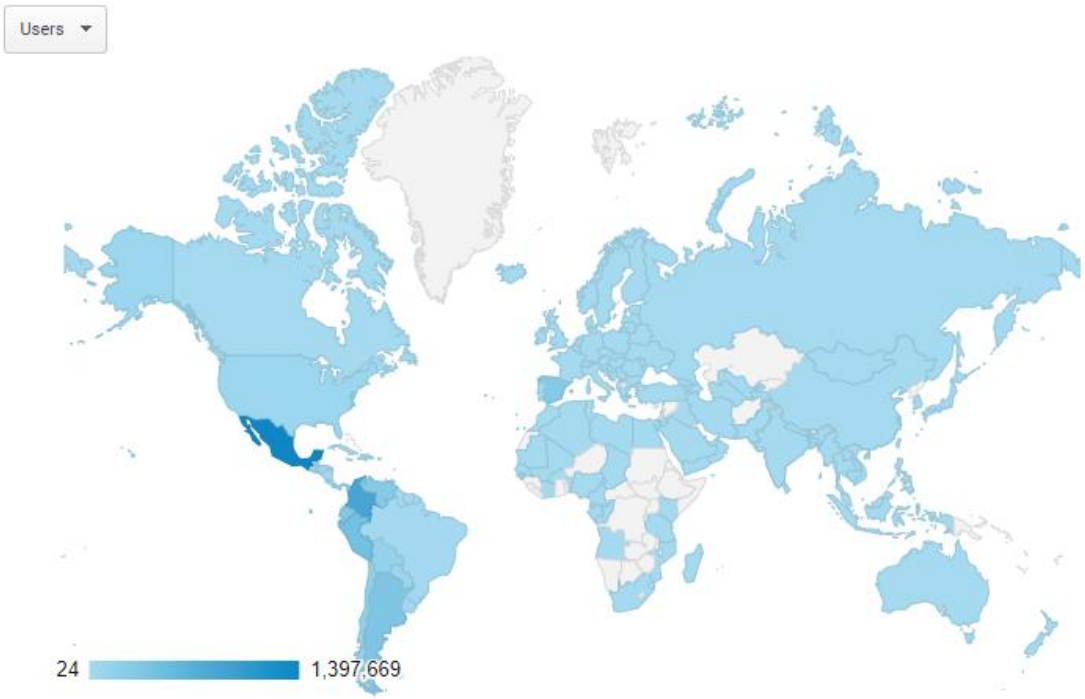


Ilustración 4 - Países desde los que se consulta el servicio

4. Diseño

El proyecto consiste en un WCF (*Windows Communication Foundation*), este tipo de plataforma de mensajería fue creada con el objetivo de permitir una programación rápida de sistemas distribuidos y el desarrollo de aplicaciones basadas en arquitecturas orientadas a servicios, con una API (*Application Programming Interface*) simple; y que puede ejecutarse en una máquina local, una LAN (*Local Area Network)*, o sobre internet en una forma segura (Wikipedia s.f.).

El elemento principal es la clase *ServicioNumerosPolaco* que implementa la interfaz *IServicioNumerosPolaco*. Esta clase llama a otras clases en función del tipo de número que se requiera convertir o la operación que se desee realizar con el objetivo de finalmente obtener un número escrito en cualquiera de las formas anteriormente descritas.

4.1. Entradas y salidas del proceso.

El primer paso que se tiene en cuenta por parte del proceso consiste en el análisis de la cadena de texto de entrada. Una vez analizada la cadena de texto de entrada, se devuelve una lista con los resultados del servicio en caso de que todo haya ido correctamente, si durante el análisis de la cadena de texto de entrada se genera algún tipo de error, se devuelve una lista con los posibles errores que han ocurrido durante el proceso de conversión.

Si se desea integrar este servicio en cualquier aplicación es conveniente por parte del equipo de integración conocer el formato de entrada esperado y las salidas que se generan con este servicio; a continuación se describirán dichas entradas y salidas.

4.1.1. Entrada.

Más adelante explicaremos en detalle cada clase, por ahora, es conveniente saber que la clase *ServicioNumerosPolaco* tiene un método llamado *Traducir*, este método es el encargado de realizar el proceso de conversión y devolver los resultados.

Dicho método tiene dos parámetros, el primero de ellos es una cadena de texto y es el número que se desea convertir escrito con dígitos numéricos, el segundo se trata del código del idioma en el que la página se encuentra traducida y también es una cadena de texto. Este código sirve porque aunque el servicio a implementar sea únicamente

para el idioma polaco, se desea tener un servicio multicultural que permita variar el idioma de la interfaz en el que se devuelven los resultados.

El número máximo de dígitos que admite el servicio varía en función del formato numérico pues, si el número está representado por notación científica, se considerará el tamaño de su base, su exponente y su posterior conversión a número decimal. Este servicio también admite números fraccionarios, negativos, y romanos aunque, estos últimos se convierten a un número entero y se trata como tal.

Para que el servicio genere un resultado, la cadena de texto introducida debe cumplir con al menos uno de los siguientes formatos aceptados. Se debe de seguir el orden indicado, de lo contrario, se devolverá un error:

- **Número entero**
  - Parte 1: [OPCIONAL]: Signo “+” o signo “–“.
  - Parte 2: [OBLIGATORIA]: De 1 a 144 dígitos numéricos.

- **Número decimal**
  - Parte 1: [OPCIONAL]: Signo “+” o signo “–“.
  - Parte 2: [OBLIGATORIA]: De 1 a 144 dígitos numéricos.
  - Parte 3: [OBLIGATORIA]: Un separador decimal (p.ej. un punto, “.”, o una coma “,”) seguido de 1 a 144 dígitos numéricos.

- **Fracción**
  - Parte 1: [OPCIONAL]: Signo “+” o signo “–“.
  - Parte 2: [OBLIGATORIA]: De 1 a 144 dígitos numéricos.
  - Parte 3: [OBLIGATORIA]: Barra separadora “/” para dividir el numerador del denominador.
  - Parte 4: [OPCIONAL]: Signo “+” o signo “–“.
  - Parte 5: [OBLIGATORIA]: De 1 a 144 dígitos numéricos.

- **Número en notación científica**
  - Parte 1: [OPCIONAL]: Signo “+” o signo “–“.
  - Parte 2: [OBLIGATORIA]: De 1 a 144 dígitos numéricos.
  - Parte 3: [OPCIONAL]: Un separador decimal (p.ej. un punto, “.”, o una coma “,”) seguido de 1 a 144 dígitos numéricos
  - Parte 4: [OBLIGATORIA]: Una letra "e" o una "E”.
  - Parte 5: [OPCIONAL]: Signo “+” o signo “–“.

- Parte 6: [OBLIGATORIA]: de 1 a 3 dígitos numéricos.

- **Número romano**
  - Formato estándar para representar a los números romanos. Se pueden escribir tanto en mayúsculas como en minúsculas.

4.1.2. Salidas.

Como se menciona en apartados anteriores, el método *Traducir* de la clase *ServicioNumerosPolaco* es el responsable de realizar todo el proceso de conversión. Para lograr este objetivo, el método *Traducir* procesa distintas tareas de conversión de forma paralela en función del análisis previo de la entrada de datos por parte del usuario, desembocando todo el proceso en una lista de resultados definitivos, siendo dicha lista el valor retornado por el método.

En caso de que se encuentre algún tipo de error durante el proceso de análisis o en la ejecución de las distintas tareas, también será retornada una lista pero, en este caso, contendrá una lista de errores.

La lista a retornar en cualquiera de los casos consiste en una implementación de la interfaz *IList* nombrada como *ArrayList*, la cual, se basa en una matriz cuyo tamaño aumenta dinámicamente cuando es necesario. Esto nos permite tener una lista de tamaño no prefijado pudiendo devolver solamente los datos que sean necesarios en cada momento.

Por lo tanto, el *ArrayList* devuelto por el método puede contener dos estructuras de datos:

- En caso de que no hayan errores, lista con todos los resultados posibles para un tipo de entrada de datos particular (p. ej. número cardinal, ordinal y multiplicativo escritos, textos de ayuda, etc.).

- En caso de que haya algún error, lista de los posibles errores que han podido ocurrir a la hora de procesar la entrada de datos por parte del usuario.

4.1.3. Tipos de salidas.

Los resultados retornados por parte del servicio varían en función del formato que tengan los datos de entrada introducidos por parte de quien consuma el servicio pues, no es lo mismo un número negativo, que un número cardinal que un número expresado en notación científica.

El límite del servicio está en 1000 "*sexquadragintillion*" menos uno. Lo cual, equivale a 10^{144} cifras numéricas.

4.2. Vista general del proceso de conversión.

Antes de analizar cómo se compone la clase *ServicioNumerosPolaco* es importante entender qué caminos se toman a la hora de procesar los datos de entrada. Estos caminos, como se ha comentado anteriormente, dependen del formato de los datos de entrada y conforman un tipo de salida u otro.

En la ilustración 5, podemos observar un diagrama de flujo que explica el comportamiento del servicio:

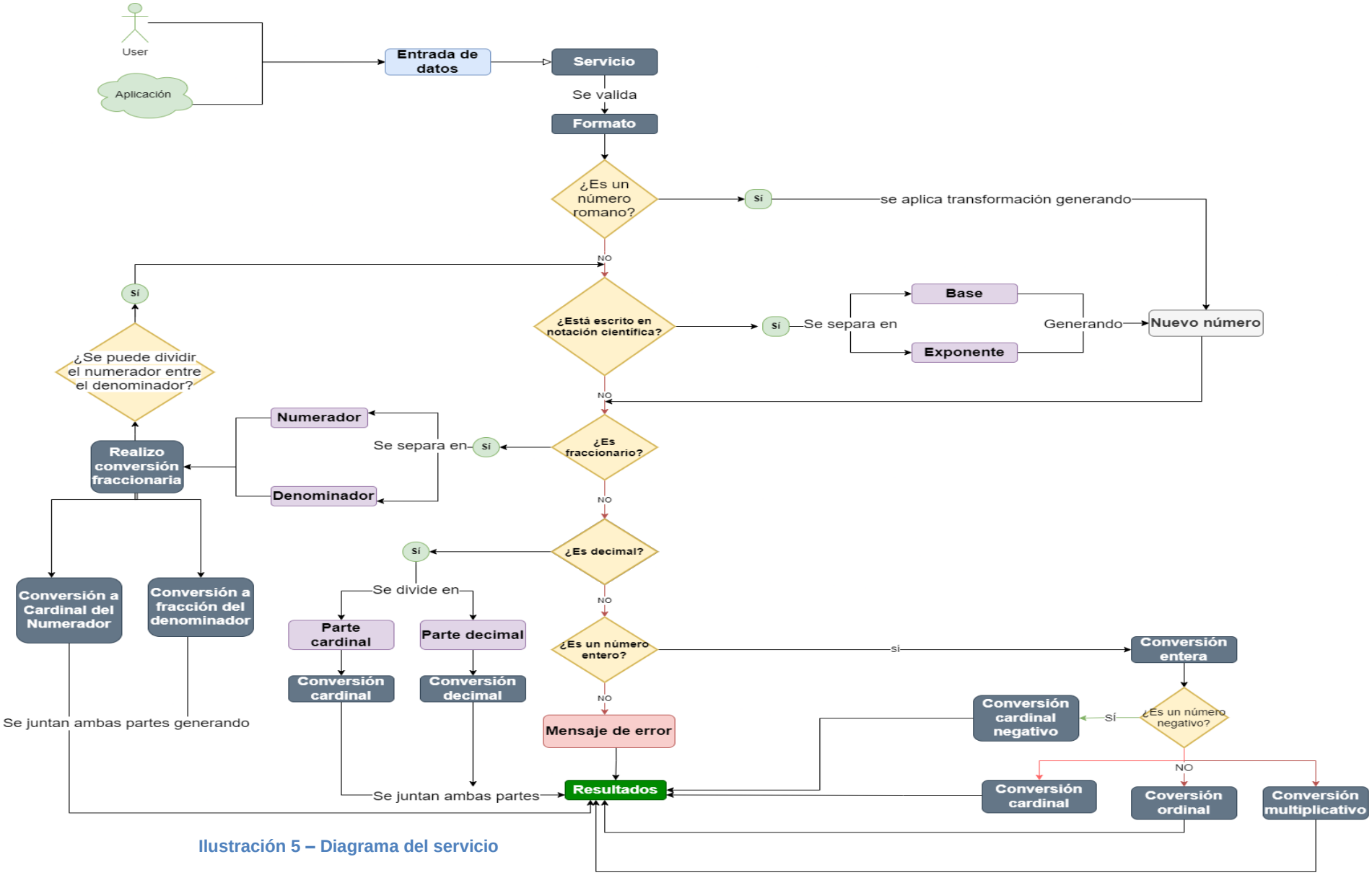


Ilustración 5 – Diagrama del servicio

A continuación procederé a explicar la ilustración 5 para ayudar a entender mejor el funcionamiento del servicio.

1. Se crea una instancia de la clase *ServicioNumerosPolaco* desde un cliente o una aplicación externa.
2. Se hace una llamada al método *Traducir*, pasando a dicho método dos parámetros. El primero de ellos, la entrada de datos que se trata del número a convertir y el segundo el código de idioma en el que se desea generar los resultados, pues esta es una aplicación multicultural.

A continuación, se realizan los siguientes procesos:

1. Se comprueba si la entrada de datos es un número romano
    a. En caso afirmativo, se convierte dicho número a un número entero y se continúa con la ejecución del método.
    b. En caso negativo, se continúa con la ejecución del método.
2. Se comprueba si la entrada de datos es un número en notación científica
    a. En caso afirmativo, se convierte dicho número a un entero o decimal aplicando la operación correspondiente.
    b. En caso negativo, se continúa con la ejecución del método.
3. A continuación se valida la entrada de los datos como se indica en el apartado "4.1.1. Entrada."
    a. En caso de que la entrada de datos sea validada se producen varios escenarios
        i. Se comprueba si el número es una fracción.
            1. En caso afirmativo, se procede a realizar una conversión fraccional separando el numerador del denominador, el numerador se procesa como una conversión cardinal y el denominador se procesa como una conversión fraccional. Una vez realizada se comprueba si la división de la fracción devuelve un número entero o decimal y se devuelve su correspondiente conversión además de la conversión fraccional previa.
            2. En caso negativo, se continúa con la ejecución del método.
        ii. Se comprueba si el número es un número decimal.
            1. En caso afirmativo, se realiza la conversión decimal, separando la parte previa al separador decimal y la parte

2. posterior. La parte previa se procesa como una conversión cardinal y la parte posterior se procesa como una conversión decimal.
3. En caso negativo, se continúa con la ejecución del método.

iii. Se comprueba si el número es entero.
1. En caso afirmativo, si el número es positivo se genera la conversión cardinal, ordinal y multiplicativa y se devuelve.
2. En caso afirmativo, si el número es negativo se genera solamente la conversión cardinal del número negativo.
3. En caso negativo, se continúa con la ejecución del método.

iv. Si no se cumple con ninguno de los formatos anteriores, se devuelve un error.

b. En caso de que la entrada de datos no haya sido validada se devuelve error.

4. Por último, se retornan los resultados del método.

4.3. Estructura del código del servicio.

El servicio se ofrece mediante la clase *ServicioNumerosPolaco*. Esta clase es la encargada de controlar la entrada de los datos y asegurarse que se cumplen todas las validaciones y se siguen todos los formatos necesarios para garantizar la robustez del código.

Para la implementación de este servicio se ha optado por un enfoque modular, separando el código en distintas clases e interfaces para hacer más ameno su entendimiento y aumentar su escalabilidad además de hacerlo más sencillo de mantener.

La responsabilidad de esta clase es comprobar que la entrada de datos sigue los formatos requeridos, transformando la entrada si se trata de un número romano o en notación científica para poder proceder a llamar a los distintos tipos de conversión. Para ello, esta clase es una implementación de la interfaz *IServicioNumerosPolaco*.

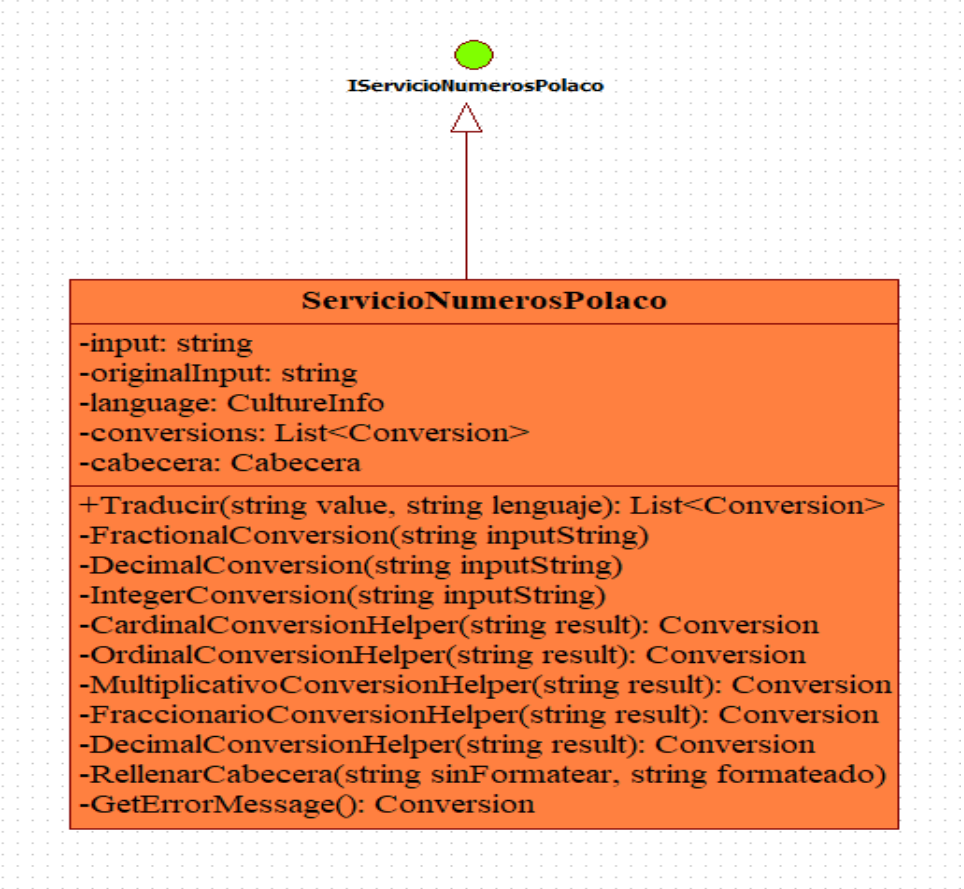


Ilustración 6 - ServicioNumerosPolaco

Como podemos observar en la ilustración 6, la clase *ServicioNumerosPolaco* que implementa la interfaz *IServicioNumerosPolaco* tiene cinco atributos.

- El primero de ellos llamado "*input*" se trata de una copia de la entrada de datos que irá siendo manipulada a lo largo del algoritmo.
- El segundo de los atributos es una copia de la entrada de datos original por parte del usuario para poder usarla en los métodos privados que existen en la clase.
- El tercer atributo se trata de un objeto de tipo "*CultureInfo*" que sirve para especificar que recurso global utilizar, estos recursos globales permiten la multiculturalidad en la aplicación.
- El cuarto atributo es una lista de objetos "*Conversion*". Este tipo de objeto se trata de un *DataContract* heredado de la interfaz IServicioNumerosPolaco que contiene todos los campos y *DataMembers* que esperan devolverse como

solución. Pues, como podemos observar, el método público Traducir devuelve una lista de tipo *Conversion* como solución.

- El quinto y último atributo se trata de la Cabecera que es otro *DataContract* heredado de la interfaz que se implementa y sirve para escribir la cabecera de la página web que consuma el servicio en el idioma que indique el parámetro "lenguaje" del método traducir.

Los métodos de la clase se pueden resumir como un único método público que es el que se llama de forma externa indicando el dato de entrada y el código del lenguaje para el objeto *CultureInfo.* El resto de métodos son privados y solo sirven para ayudar en el proceso y no tener toda la lógica junta en un solo método.

Todos los métodos que incluyen la palabra "*Conversion*" están encargados de crear las instancias de las demás clases y generar los resultados cumpliendo con el *DataContract Conversion* heredado de la interfaz.

El método *GetErrorMessage* genera un objeto *Conversion* que contiene la lista de posibles errores por parte del algoritmo.

Dentro de los correspondientes métodos de conversión se llaman a instancias de las clases que tienen la lógica donde se realiza la conversión, a continuación se detallan todas las clases e interfaces que conforman el servicio.

4.3.1. Interfaz *INumbers.*

Para realizar los distintos tipos de conversiones se ha creado una interfaz llamada *INumbers* que contiene un único método llamado *ConvertIntoWords* que devuelve una cadena de texto.

Esta interfaz es implementada por una serie de clases y cada clase corresponde a un tipo de resultado. Como podemos observar en la ilustración 7, las clases en sí no manejan la cadena de texto, esta responsabilidad es cedida a una implementación de la interfaz *IStringIterator*, dicha implementación se instancia aplicando uno de los principios **SOLID** conocido como **inversión de dependencias** a través de la **inyección de dependencias.**

La inversión de dependencias defiende el siguiente principio: "Depende de abstracciones, no de clases concretas" (Martín, 2018). Por lo tanto, las clases Cardinal, Ordinal, Decimal y Fraction al recibir en su constructor una instancia de tipo

IStringIterator e inyectándola al atributo a través de la inyección de dependencias se consigue aplicar correctamente la inversión de dependencias.

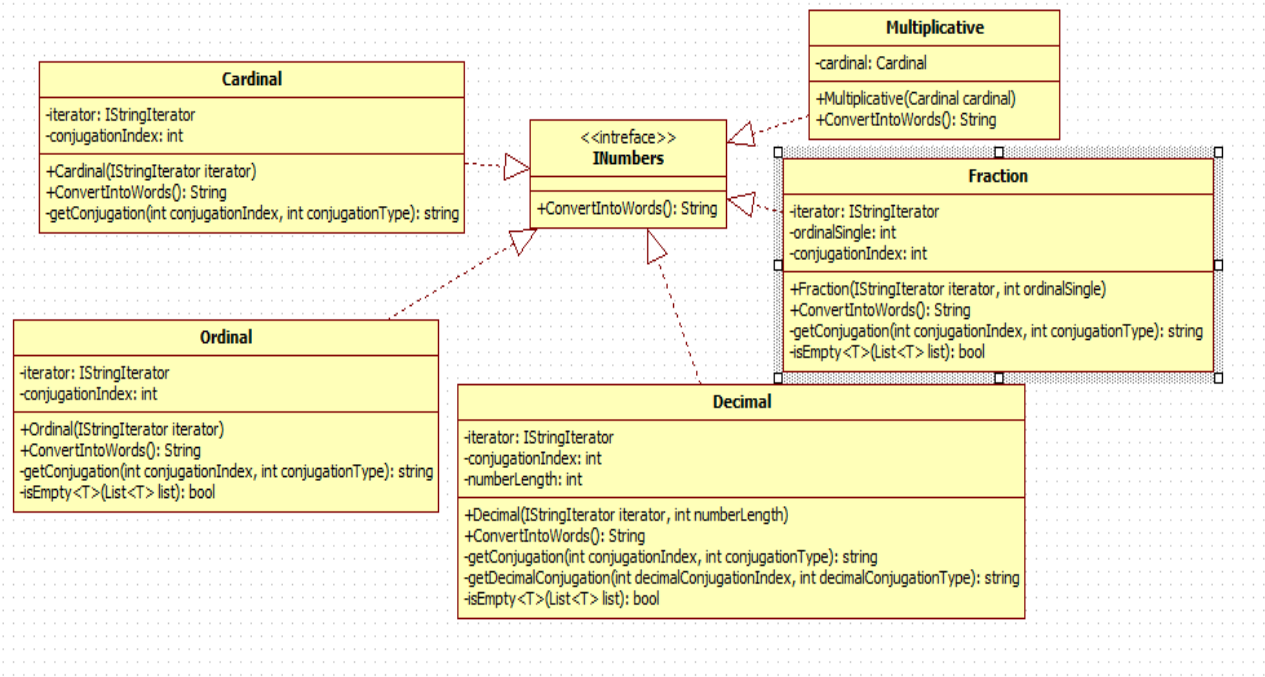


Ilustración 7 - INumbers

Además del principio SOLID anteriormente citado, también se sigue el principio conocido como **principio de abierto/cerrado** pues, gracias al uso de una interfaz el código está abierto a extensiones, pudiendo añadir un nuevo tipo de comportamiento (nuevo tipo de número) pero, se encuentra cerrado a modificaciones pues, si queremos añadir un tipo de número nuevo simplemente tendremos que crear una clase que implemente la interfaz *INumbers* (abierto a extensiones) y escribir la lógica que utilice ese tipo de número sin la necesidad de modificar el resto de clases (cerrado a modificaciones).

Por último y para terminar de defender el motivo de por qué se eligió una interfaz para implementar los distintos tipos de números, cabe destacar el **principio de responsabilidad única**, pues gracias al uso de la interfaz combinado con la inyección de dependencias que inserta una instancia de *IStringIterator*, solo existe una única razón por la que estas clases pueden cambiar, ciñéndose así al principio de responsabilidad única; el único motivo para cambiar estas clases es que el tipo de número en concreto cambie, pues si se desea cambiar, por ejemplo, la forma en la que se recorre y procesa la entrada de los datos esa responsabilidad pertenece a la clase *StringIterator* que implementa *IStringIterator*.

También cabe destacar que cada clase que implementa la interfaz *INumbers* excepto la clase *Multiplicative*, contienen una serie de arreglos estáticos como campos y, dichos arreglos, contienen los términos necesarios para nombrar los números en el proceso de fraccionamiento y concatenación del que se hablará más adelante.

4.3.2. Interfaz *IStringIterator*.

Esta interfaz es la encargada de procesar y recorrer la cadena de texto introducida por el usuario y que previamente ha sido validada. Como podemos observar en la ilustración 8, esta interfaz tiene tres métodos.

El primer método llamado *Next* es el encargado de devolver el siguiente número a procesar, como se explicó anteriormente los números se pueden procesar de tres en tres, esto evita tener que convertir el número a una variable de tipo entero, lo cual, es muchos casos sería imposible pues el límite de este tipo de datos es limitado.

El segundo método *HasNext* facilita la iteración de la cadena de texto, pudiendo usarse como condición en un bucle para saber cuándo hemos terminado de procesar la cadena.

El tercer y último método *getConjugationType* ayuda a saber qué tipo de conjugación se debe emplear en polaco pues, como se explicó anteriormente, el idioma polaco tiene distintas conjugaciones dependiendo de si el número es singular o plural.

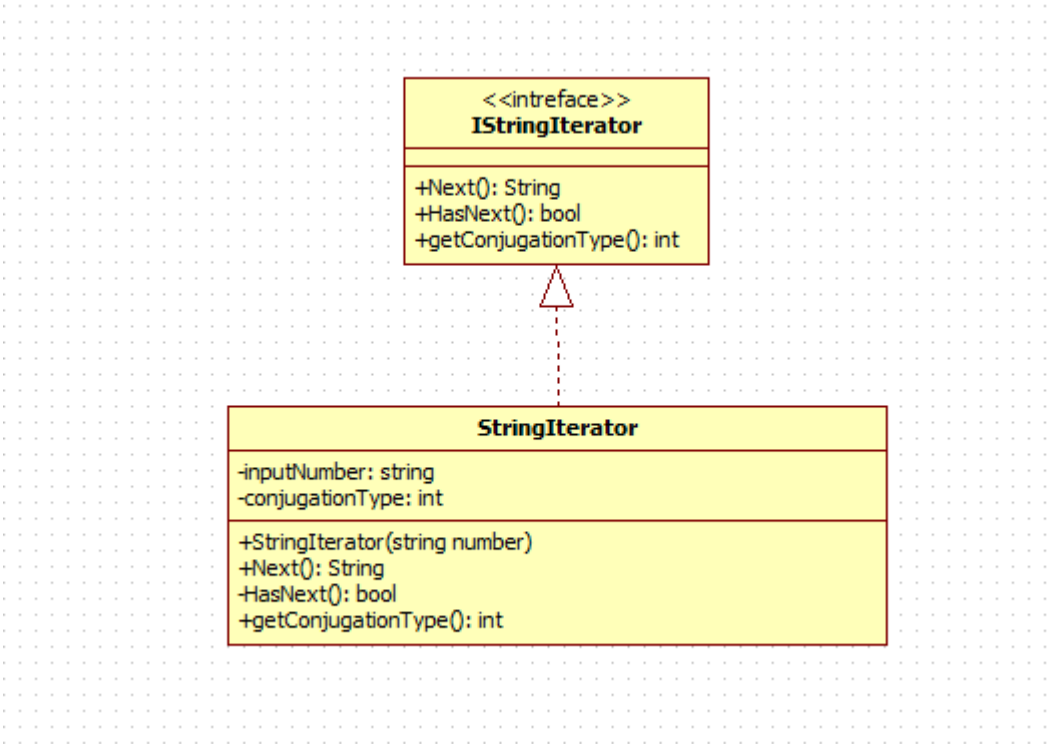


Ilustración 8 - IStringIterator

El objetivo de utilizar una implementación de una interfaz para poder procesar la cadena de texto es que, si en el futuro se desea cambiar la forma en la que se itera sobre la cadena de texto, simplemente se tendrá que crear una nueva implementación de la interfaz y pasar la instancia mediante inyección de dependencias al resto de tipos numéricos.

4.3.3. Clase *SignedNumber.*

El objetivo de esta clase es procesar un número que tenga signo, es decir, que la cadena de texto contenga un signo más o menos (‘+’ o ‘-‘). Como podemos observar en la ilustración 9, esta clase tiene dos métodos, el primero de ellos para procesar las fracciones donde su numerador o denominador tiene signo y el segundo método para procesar los números que no son fracciones pero tienen signo.

Esta distinción se debe a que en las fracciones, se debe saber el signo tanto del numerador como del denominador para saber si el resultado final es positivo o negativo pues, por ejemplo, si el numerador es negativo y el denominador también el resultado será un número positivo.

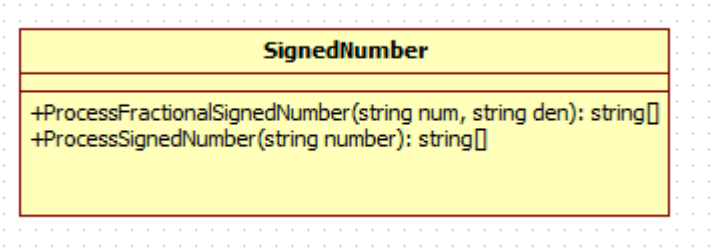


Ilustración 9 - SignedNumber

Esta clase devuelve un arreglo de tipo *string*, donde para el caso de las fracciones, se devuelve la cadena de texto con el numerador y denominador con el signo quitado y en la tercera posición, el signo final que tendrá el número. (p.ej [“23”, “12”, “minus”]).

4.3.4. Clase *ScientificNotation*

La clase *ScientificNotation* se encarga de transformar la cadena de texto de entrada a un número entero o decimal dependiendo de la base y exponente que ingrese el usuario.

Como se puede observar en la ilustración 10, el método *Convert* recibe una *string* que es la cadena de texto en formato de notación científica y devuelve otra cadena de texto con el número entero o decimal. Además, si se produce algún error se devolverá una cadena de texto con la palabra "*error*".

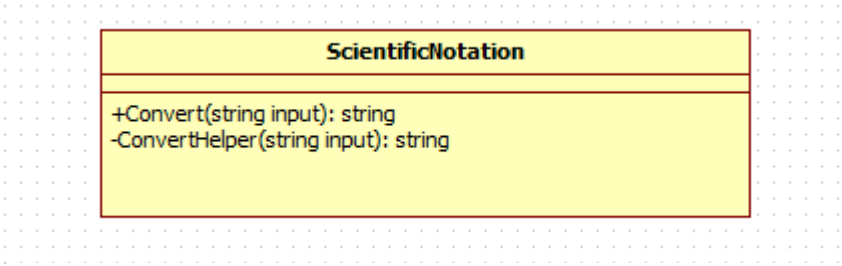


Ilustración 10 - ScientificNotation

4.3.4. Clase *RomanNumber*

La clase *RomanNumber* se encarga de transformar la cadena de texto de entrada escrita en formato de número romano a un número entero que pueda ser procesado por el servicio.

Para ello y como se puede observar en la ilustración 11, hay una serie de métodos públicos que permiten la transformación de número romano a número entero y viceversa sirviendo los métodos de esta clase como "*helpers*".

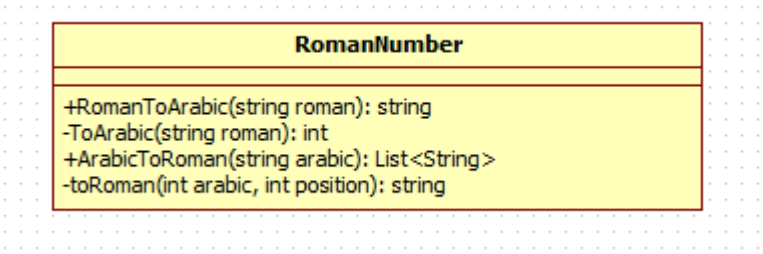


Ilustración 11 - RomanNumber

4.3.5. Clase *Validator*

La clase *Validator* es la encargada de validar que la entrada cumple con los requisitos definidos en el apartado "4.1.1. Entrada."

Como podemos observar en la ilustración 12, esta clase tiene un método principal público llamado *Validate* que recibe una cadena de texto a validar y devuelve un valor de tipo *bool*.

Para realizar esta validación la clase cuenta con una serie de métodos privados, cada uno de estos métodos comprueban los límites del algoritmo para que no haya errores inesperados.

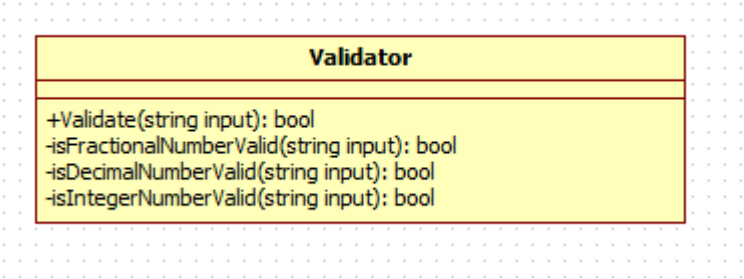


Ilustración 12 - Validator

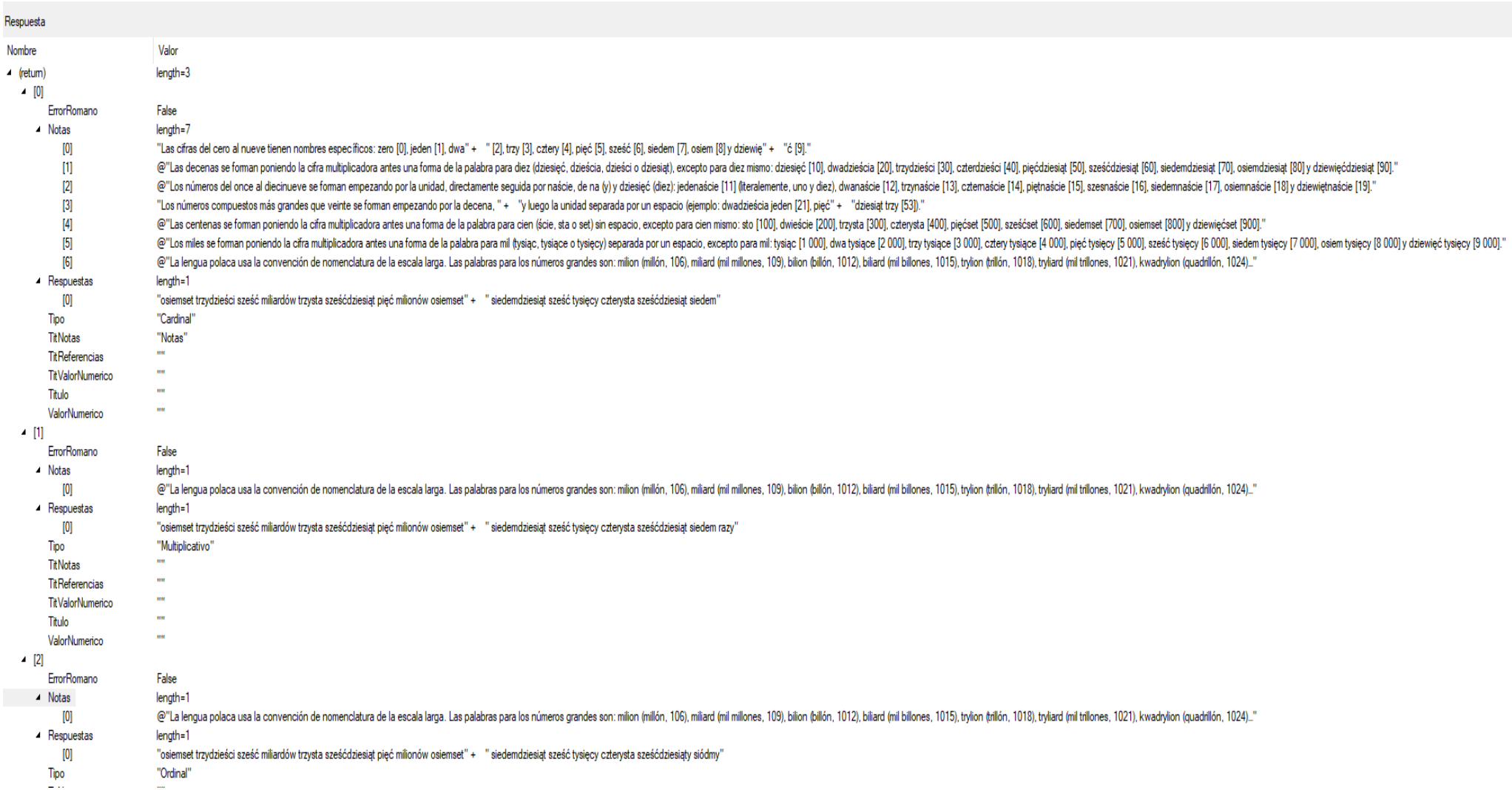
4.4. Formato de salida.

Como comentamos anteriormente en el apartado “4.1.2. Salidas”, la clase *ServicioNumerosPolaco* al llamar a su único método público *Traducir* devuelve un contenedor de objetos *Conversion*, el cual no deja de ser una lista de objetos donde cada objeto contiene una serie de cadenas de texto o listas de cadenas de texto.

A continuación se mostrarán tres ilustraciones que darán claridad y ayudarán a entender la estructura de los resultados que se obtienen al ejecutar el método *Traducir* de la clase *ServicioNumerosPolaco* representando estos números:

1. 836 365 876 467 (ilustración 13).
2. -347/689 (ilustración 14).
3. 2.468e7 (ilustración 15).

Resultados para el número: 836 365 876 467



Respuesta

Nombre	Valor
(return)	length=3
[0]	
ErrorRomano	False
Notas	length=7
[0]	"Las cifras del cero al nueve tienen nombres específicos: zero [0], jeden [1], dwa" + " [2], trzy [3], cztery [4], pięć [5], sześć [6], siedem [7], osiem [8] y dziewię" + "ć [9]."
[1]	@"Las decenas se forman poniendo la cifra multiplicadora antes una forma de la palabra para diez (dziesięć, dzieścia, dzieści o dziesiąt), excepto para diez mismo: dziesięć [10], dwadzieścia [20], trzydzieści [30], czterdzieści [40], pięćdziesiąt [50], sześćdziesiąt [60], siedemdziesiąt [70], osiemdziesiąt [80] y dziewięćdziesiąt [90]."
[2]	@"Los números del once al diecinueve se forman empezando por la unidad, directamente seguida por naście, de na (y) y dziesięć (diez): jedenaście [11] (literalemente, uno y diez), dwanaście [12], trzynaście [13], czternaście [14], piętnaście [15], szesnaście [16], siedemnaście [17], osiemnaście [18] y dziewiętnaście [19]."
[3]	"Los números compuestos más grandes que veinte se forman empezando por la decena, " + "y luego la unidad separada por un espacio (ejemplo: dwadzieścia jeden [21], pięć" + "dziesiąt trzy [53])."
[4]	@"Las centenas se forman poniendo la cifra multiplicadora antes una forma de la palabra para cien (ście, sta o set) sin espacio, excepto para cien mismo: sto [100], dwieście [200], trzysta [300], czterysta [400], pięćset [500], sześćset [600], siedemset [700], osiemset [800] y dziewięćset [900]."
[5]	@"Los miles se forman poniendo la cifra multiplicadora antes una forma de la palabra para mil (tysiąc, tysiące o tysięcy) separada por un espacio, excepto para mil: tysiąc [1 000], dwa tysiące [2 000], trzy tysiące [3 000], cztery tysiące [4 000], pięć tysięcy [5 000], sześć tysięcy [6 000], siedem tysięcy [7 000], osiem tysięcy [8 000] y dziewięć tysięcy [9 000]."
[6]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015), trylion (trillón, 1018), tryliard (mil trillones, 1021), kwadrylion (quadrillón, 1024)..."
Respuestas	length=1
[0]	"osiemset trzydzieści sześć miliardów trzysta sześćdziesiąt pięć milionów osiemset" + " siedemdziesiąt sześć tysięcy czterysta sześćdziesiąt siedem"
Tipo	"Cardinal"
TitNotas	"Notas"
TitReferencias	""
TitValorNumerico	""
Titulo	""
ValorNumerico	""
[1]	
ErrorRomano	False
Notas	length=1
[0]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015), trylion (trillón, 1018), tryliard (mil trillones, 1021), kwadrylion (quadrillón, 1024)..."
Respuestas	length=1
[0]	"osiemset trzydzieści sześć miliardów trzysta sześćdziesiąt pięć milionów osiemset" + " siedemdziesiąt sześć tysięcy czterysta sześćdziesiąt siedem razy"
Tipo	"Multiplicativo"
TitNotas	""
TitReferencias	""
TitValorNumerico	""
Titulo	""
ValorNumerico	""
[2]	
ErrorRomano	False
Notas	length=1
[0]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015), trylion (trillón, 1018), tryliard (mil trillones, 1021), kwadrylion (quadrillón, 1024)..."
Respuestas	length=1
[0]	"osiemset trzydzieści sześć miliardów trzysta sześćdziesiąt pięć milionów osiemset" + " siedemdziesiąt sześć tysięcy czterysta sześćdziesiąty siódmy"
Tipo	"Ordinal"

Ilustración 13 - Primer resultado

Resultados para el número: -347/689

Respuesta

Iniciar un

Nombre	Valor
(return)	length=2
[0]	
ErrorRomano	False
Notas	length=1
[0]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billon
Respuestas	length=1
[0]	"minus trzysta czterdzieści siedem sześćset osiemdziesiątych dziewiątych"
Tipo	"Fraccionario"
TitNotas	""
TitReferencias	""
TitValorNumerico	""
Titulo	""
ValorNumerico	""
[1]	
ErrorRomano	False
Notas	(null)
Respuestas	length=1
[0]	"minus i pięćset trzy biliony sześćset dwadzieścia osiem miliardów czterysta czte" + "rdzieści siedem milionów dwadzieścia cztery tysiące sześćset siedemdziesiąt trzy" + " biliardowe"
Tipo	"Decimal"
TitNotas	""
TitReferencias	""
TitValorNumerico	""
Titulo	""
ValorNumerico	""

Ilustración 14 - Segundo resultado

Resultados para el número: 2.468e7

Respuesta

Iniciar un nuevo pr

Nombre	Valor
(return)	length=3
[0]	
ErrorRomano	False
Notas	length=7
[0]	"Las cifras del cero al nueve tienen nombres específicos: zero [0], jeden [1], dwa" + " [2], trzy [3], cztery [4], pięć [5], sześć [6], siedem [7], osiem [8] y dziewię" + "ć [9]."
[1]	@"Las decenas se forman poniendo la cifra multiplicadora antes una forma de la palabra para diez (dziesięć, dzieścia, dzieści o dziesiąt), excepto para diez mismo: dziesięć [10], dwadzieścia [20], trzydzieści [30],
[2]	@"Los números del once al diecinueve se forman empezando por la unidad, directamente seguida por naście, de na (y) y dziesięć (diez): jedenaście [11] (literalemente, uno y diez), dwanaście [12], trzynaście [13
[3]	"Los números compuestos más grandes que veinte se forman empezando por la decena, " + "y luego la unidad separada por un espacio (ejemplo: dwadzieścia jeden [21], pięć" + "dziesiąt trzy [53])."
[4]	@"Las centenas se forman poniendo la cifra multiplicadora antes una forma de la palabra para cien (ście, sta o set) sin espacio, excepto para cien mismo: sto [100], dwieście [200], trzysta [300], czterysta [400], p
[5]	@"Los miles se forman poniendo la cifra multiplicadora antes una forma de la palabra para mil (tysiąc, tysiące o tysięcy) separada por un espacio, excepto para mil: tysiąc [1 000], dwa tysiące [2 000], trzy tysiące
[6]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015
Respuestas	length=1
[0]	"dwadzieścia cztery miliardy sześćset osiemdziesiąt milionów"
Tipo	"Cardinal"
TitNotas	"Notas"
TitReferencias	""
TitValorNumerico	"Valor numérico"
Titulo	""
ValorNumerico	"dwadzieścia cztery miliardy sześćset osiemdziesiąt milionów"
[1]	
ErrorRomano	False
Notas	length=1
[0]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015
Respuestas	length=1
[0]	"dwadzieścia cztery miliardy sześćset osiemdziesiąt milionów razy"
Tipo	"Multiplicativo"
TitNotas	""
TitReferencias	""
TitValorNumerico	""
Titulo	""
ValorNumerico	""
[2]	
ErrorRomano	False
Notas	length=1
[0]	@"La lengua polaca usa la convención de nomenclatura de la escala larga. Las palabras para los números grandes son: milion (millón, 106), miliard (mil millones, 109), bilion (billón, 1012), biliard (mil billones, 1015
Respuestas	length=1
[0]	"dwadzieścia cztery miliardy sześćset osiemdziesiąt milionowy"
Tipo	"Ordinal"
TitNotas	""

Ilustración 15 - Tercer resultado

5. Desarrollo.

El servicio WCF (*Windows Communication Foundation*) descrito a lo largo de esta memoria ha sido implementado en el lenguaje C#, utilizando el entorno de desarrollo de Microsoft llamado Visual Studio y en su versión “Microsoft Visual Studio Community 2022”. Además, para implementar el cliente del que hablaremos más adelante se ha usado el *framework* conocido como Microsoft .NET.

Existieron dos fases diferenciadas durante el desarrollo del servicio; la primera de ellas consistió en el desarrollo de una aplicación de consola en C# que devolviese los valores esperados para una cadena de texto determinada. La segunda fase se centró en la implementación del servicio como WCF adaptando el código implementado en la fase anterior y conectando el servicio a una interfaz de usuario.

El código desarrollado sigue las características detalladas a continuación:

- **Programación orientada a objetos**. Implementando una estructura modular con distintas clases y objetos que forman el servicio desarrollado.

- **Expresiones regulares.** Utilizadas para validar la entrada de datos que recibe el servicio y manejar las cadenas de texto que contienen números sin la necesidad de convertir la cadena de texto a entero.

- **Concurrencia.** Se utiliza concurrencia para ejecutar tareas de forma paralela, mejorando así el tiempo de respuesta del servicio.

- **Control de versiones.** Se ha utilizado control de versiones para facilitar el desarrollo del código, pudiendo siempre volver atrás o ver qué cambios se han realizado en el código.

- **Fraccionamiento y concatenación.** Necesario para procesar la cadena de texto que contiene el número sin la necesidad de convertirlo a entero pues, el tipo *int* tiene un tamaño limitado.

- **Uso de la clase StringBuilder.** Se ha empleado la clase StringBuilder para mejorar el rendimiento en situaciones donde se pretende modificar de forma continuada una cadena de texto.

- **Implementación de una interfaz de usuario.** Se ha creado una interfaz de usuario que consume el servicio con la intención de probar el servicio en un entorno gráfico, mejorando así la experiencia de los usuarios.

- **Pruebas de software.** Se han realizado una serie de pruebas para comprobar la calidad de los resultados del servicio.

Veamos a continuación con un mayor detalle las características anteriormente mencionadas.

5.1. Programación orientada a objetos.

Como hemos podido observar anteriormente en la explicación del diseño del servicio se ha empleado programación orientada a objetos para el desarrollo del servicio. Esto lo podemos ver claramente debido a la existencia de distintas clases con sus respectivos métodos y atributos.

Además se han utilizado patrones de diseño propios de la programación orientada a objetos como puede ser la inversión de dependencias a través de la inyección de dependencias, pasando por parámetro a un constructor de una clase una instancia de una implementación de una interfaz.

5.2. Expresiones regulares.

Antes de proceder a instanciar las distintas implementaciones de la interfaz *INumber* y empezar el proceso de conversión, es obligatorio, en caso de querer evitar errores inesperados, validar el formato de los datos de entrada. Hay muchas formas de validar que una cadena de texto cumple con un formato requerido pero, una de las más eficientes y que menos código necesitan son las expresiones regulares.

Una expresión regular se puede definir como una “secuencia de caracteres que conforma un patrón de búsqueda. Se utilizan principalmente para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones” (Wikipedia, s.f.).

La lista de caracteres y combinaciones que pueden emplearse para crear una expresión regular es realmente vasta, por lo tanto, no es pertinente explicarla en esta memoria; aun así, es interesante entender las expresiones regulares que sí se han empleado en el código y cuáles son sus funciones dentro del mismo. Por ello

examinaremos las expresiones regulares pertinentes a la clase *Validator* donde existen tres patrones, uno para las fracciones, otro para los números decimales y otro para los números enteros. Además se detallarán las expresiones regulares utilizadas para saber si el número de entrada está escrito como número romano o en notación científica.

A continuación procedemos a detallarlos:

- Fracción:

[+-]?\d+\/[+-]?\d+

Caracteres	Opcional	Descripción
[+-]?	Sí	Un signo + o un signo –
\d+	No	Uno o más dígitos numéricos.
\/	No	Una barra fraccionaria.

- Número decimal:

[+-]?([0-9]+[.,][0-9]+)

Caracteres	Opcional	Descripción
[+-]?	Sí	Un signo + o un signo –
([0-9]+[.,][0-9]+)	No	Uno o más dígitos numéricos seguidos de un separador decimal (.,) seguido, a su vez, de uno o más dígitos numéricos.

- Número entero:

^[+-]?[0-9]*$

Caracteres	Opcional	Descripción
^[+-]?	Sí	Un signo + o un signo – al comienzo de la cadena.
[0-9]+$	No	Cero o más dígitos numéricos al final de la cadena.

- Número romano:
  ^[IVXLCDMivxlcdm]+$

Caracteres	Opcional	Descripción
^[IVXLCDMivxlcdm]+$	No	Busca una cadena de texto que comience y finalice con uno o más números romanos ya estén escritos en mayúsculas o minúsculas.

- Notación científica:
  [+-]?\d+(\.\d+)?[Ee][+-]?\d+

Caracteres	Opcional	Descripción
[+-]?\d+	No	Un signo + o un signo – seguido de uno o más dígitos numéricos.
(\.\d+)?	Si	Busca cero o un separador decimal y uno o más dígitos numéricos. Es decir, busca si el número es decimal o no.
[Ee]	No	Una letra e o E para delimitar el comienzo de la parte exponencial.

En el caso de que el número cumpla alguno de estos patrones y, además, se compruebe que se encuentra entre los límites del algoritmo midiendo la longitud de la cadena, los números se descomponen a su vez en:

- Numerador y denominador para las fracciones.
- Parte entera y parte decimal para los números decimales.
- Si se trata de un número romano se pasa a número entero.
- Si se trata de un número en notación científica se devuelve una cadena con el número convertido a entero o decimal.

5.3. Concurrencia.

Se ha diseñado un servicio que puede ser consumido por distintos tipos de aplicaciones, cabiendo la posibilidad de que exista una alta demanda de peticiones al servicio, por lo tanto, es necesario que el proceso de conversión se realice de la forma más rápida posible garantizando una robustez en los resultados y en el proceso. Además, como hemos podido observar a lo largo del desarrollo de esta memoria, el

proceso de conversión se basa en tareas que pueden ejecutarse de forma independiente. Todos estos indicadores han llevado a tomar la decisión de utilizar un mecanismo que permita ejecutar de forma simultánea todas estas tareas individuales para reducir así el tiempo de respuesta.

Este mecanismo se ha implementado utilizando la biblioteca TPL (*Task Parallel Library*, o Biblioteca de Procesamiento Paralelo basado en tareas) que forma parte del framework Microsoft .NET. “La biblioteca TPL se basa en el concepto de tarea, que representa una operación asincrónica. De cierta forma, una tarea recuerda a un subproceso o elemento de trabajo *ThreadPool*, pero en un nivel más alto de abstracción. El término paralelismo de tareas hace referencia a la ejecución simultánea de una o varias tareas independientes” (Microsoft, 2022). Existen dos beneficios importantes a la hora de emplear tareas:

- Aumento de la eficacia y escalabilidad de los recursos del sistema.

  Gracias a algoritmos centrados en el balanceo de carga y la maximización del rendimiento se consigue que las tareas no afecten de manera muy negativa en el manejo de recursos por parte del sistema.

- Aumento del control empleando técnicas de programación.

  Existe un gran conjunto de técnicas de programación que se ofrecen en forma de API, las cuales permiten el uso de esperas, cancelaciones, un control de excepciones que aumenta la robustez del código, un estado que aporta gran información y además se pueden implementar de forma personalizada algunas funcionalidades.

Se nos presentan dos formas de implementar esta funcionalidad y llega la hora de establecer un juicio de cuál usar y por qué. La primera de estas formas es a través del método *Invoke* de la clase *Parallel* el cual recibe por parámetro un arreglo de *Action*, estos “*Action*” pueden ser una función lambda u otro tipo de función y es necesario crear uno por elemento de trabajo. La segunda opción se trata de crear objetos *Task* por cada elemento de trabajo y luego esperar a la ejecución de cada uno de los objetos y obtener los valores retornados.

Se ha realizado una investigación acerca de qué enfoque utilizar para este servicio y se ha llegado a la conclusión de que se usará el método *Invoke* de la clase *Parallel*. A continuación se listan los motivos de esta elección:

1. La forma en la que está pensada la clase *ServicioNumerosPolaco*. Esta clase tiene una lista de resultados como atributo y es dicho atributo el que se va modificando a medida que se ejecutan los distintos procesos, por lo tanto, no necesitamos ir acumulando los distintos resultados sino que se añaden de forma ordenada a la respuesta final.

2. Gracias a utilizar *Paralell.Invoke* nos ahorramos un proceso de integración de resultados pues, este método espera hasta que todas las tareas hayan terminado por lo tanto, nos aseguramos que el orden de los resultados sea el esperado por orden de ejecución.

3. Teniendo en cuenta que tanto *Task* como *Parallel* son decoradores de *ThreadPool,* la elección de cuál usar dependerá sobre todo en cuánto control se necesita a la hora de ejecutar las tareas paralelas. Como se comentó anteriormente, no es necesario ningún tipo de control pues, ejecutamos las acciones del método *Invoke* como funciones lambda que instancian las respectivas clases que devuelven el resultado esperado y una vez acaban todas las tareas se añaden los resultados a la lista de respuestas.

En función del número que entre se ejecutan distintas tareas dentro de las múltiples llamadas al método *Invoke*, el comportamiento es el siguiente:

- En caso de que el número sea una fracción:
    - Se ejecutan dos tareas paralelas como función lambda, la primera de ellas convierte el numerador instanciando la clase *Cardinal* y la segunda convierte el denominador instanciando la clase *Fractions*.
    - Se añade el resultado a la lista de resultados concatenando el resultado del numerador y el resultado del denominador.
    - Se realiza la división del numerador y el denominador en caso de que sus respectivas longitudes lo permitan y se trata el resultado como un número entero o decimal.

- En caso de que el número sea decimal:
    - Se ejecutan dos tareas paralelas como función lambda, la primera de ellas convierte el numerador instanciando la clase *Cardinal* y la segunda convierte el denominador instanciando la clase *Decimal*.
    - Se añade el resultado a la lista de resultados concatenando el resultado de la parte entera y el resultado de la parte decimal.

- En caso de que sea un número entero:
    - En caso de que el número sea negativo no es necesario ejecutar varias tareas de forma paralela y simplemente se devuelve el número cardinal negativo.
        - Si el número es positivo se ejecutan cuatro tareas de forma paralela utilizando funciones lambda como *Action.* La primer tarea devolvería el resultado cardinal, la segunda el resultado ordinal la tercera el resultado fraccionario y la cuarta el resultado multiplicativo.
    - Se añaden los resultados a la lista de resultados.

5.4. Control de versiones.

El control de versiones es “la práctica de rastrear y gestionar los cambios en el código de software” (Atlassian, s.f.). Este tipo de herramientas permiten a los equipos de software o programadores independientes el poder gestionar los cambios en el código a lo largo del tiempo, además este tipo de herramientas permiten trabajar de forma más rápida e inteligente.

Para realizar el código se ha utilizado control de versiones, se ha optado por utilizar *Git* que ya viene incorporado con el entorno de desarrollo de Microsoft Visual Studio.

El software de control de versiones conocido como *Git* fue pensado en la eficiencia, la confiabilidad y la compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código.

Utilizar control de versiones a la hora de desarrollar código ofrece una serie de ventajas:

- Un historial completo que contiene todos los cambios que se han realizado durante un largo plazo a todos los archivos. Estos cambios que se registran suelen incluir la creación y eliminación de archivos además de los cambios de su contenido.

- La creación de ramas puede ayudar a tener distintos enfoques para un mismo proyecto. Una rama en un control de versiones se puede entender como la rama de un árbol, esa rama es un posible camino pero siempre se puede volver al tronco principal.

- Trazabilidad. El poder trazar cada cambio que se hace en el software puede ser de gran utilidad si en el futuro se desea incluir aplicaciones externas que permitan la gestión de proyectos.

Lógicamente, se puede desarrollar software sin la necesidad de utilizar control de versiones pero hacerlo conlleva una serie de riesgos que ningún profesional debería aceptar.

5.5. Fraccionamiento y concatenación.

Para poder procesar los números tan grandes que este servicio es capaz de manejar se debe procesar como una cadena de texto la entrada de datos por parte del usuario. Para ello, la cadena de texto se procesa en grupos de tres cifras de derecha a izquierda.

El procedimiento genérico que se emplean en estos procesos, los cuales están programados dentro de las implementaciones de la interfaz *INumbers*, se detalla a continuación:

1. Al crearse una instancia de por ejemplo, la clase Cardinal, se inicializan una serie de arreglos estáticos que contienen los términos correspondientes a los grupos de tres dígitos.

2. A continuación, se inicializan una serie de variables, entre ellas la que almacenará el resultado que devuelve el método *ConvertIntoWords* heredado de la interfaz *INumbers.*

3. El siguiente paso consiste en iterar de forma continuada los dígitos de tres en tres, desde los menos significativos hasta los más significativos (de derecha a izquierda). Por lo tanto, se realiza un bucle que itera hasta que no haya siguiente usando el patrón de diseño *Iterator*, el cual, se implementa a través de la clase *StringIterator* que implementa a su vez la interfaz *IStringIterator*.

4. Una vez concluye el bucle se ordenan los resultados y se devuelven.

Para representar de una forma más clara este proceso, veamos como actuaría el código con uno de los números de ejemplo del apartado “4.4. Formato de salida”:

836 365 876 467

1. Se inician los arreglos estáticos que contienen los términos correspondientes a los grupos de tres dígitos. En este caso: {“tysięcy”. “milionów”, “miliardów“} Siguiendo la escala larga que se usa en polaco.

2. Se inicializan las variables.

3. Se procede a descomponer el número en grupos de tres dígitos de derecha a izquierda, calculándose en cada iteración las palabras que representan a cada grupo de cifras de la tabla 13 y añadiendo dicha palabra al resultado del proceso.

Iteración	Grupo	Resultados
1	467	czterysta sześćdziesiąt siedem
2	876	osiemset siedemdziesiąt sześć tysięcy czterysta sześćdziesiąt siedem
3	365	trzysta sześćdziesiąt pięć milionów osiemset siedemdziesiąt sześć tysięcy czterysta sześćdziesiąt siedem
4	836	osiemset trzydzieści sześć miliardów trzysta sześćdziesiąt pięć milionów osiemset siedemdziesiąt sześć tysięcy czterysta sześćdziesiąt siedem

Tabla 13 - Iteración de los números

4. Se devuelve el último resultado.

5.6. Uso de la clase StringBuilder.

Debido a que el tipo de objeto *String* es inmutable, cada vez que se usa uno de los métodos de la clase *System.String* se crea un objeto de cadena en la memoria, esto implica que se asigne un espacio para dicho objeto.

En situaciones donde es necesario realizar múltiples modificaciones de forma repetida en una cadena de texto, la sobrecarga que conlleva crear objetos de tipo *String* de forma constante puede acabar afectando de forma notoria al rendimiento del servicio.

Además se recomienda encarecidamente el uso de la clase *StringBuilder* cuando se pretende concatenar muchas cadenas dentro de un bucle, esto es debido a que la clase *StringBuilder* se puede usar para modificar una cadena sin crear un objeto.

Debido a que en cada implementación de la interfaz *INumbers* se realiza un bucle donde se llevan a cabo muchas concatenaciones de cadenas de texto, se ha optado por emplear la clase *StringBuilder*.

Gracias a esto el servicio ofrece un mayor rendimiento y reduce el uso de recursos innecesarios aumentando su eficiencia.

5.7. Implementación de una interfaz de usuario.

Para consumir el servicio WCF y poder probar su utilidad en un entorno de desarrollo más acertado se ha optado por implementar una interfaz de usuario.

Para implementar esta interfaz de usuario se ha usado una combinación de ASP.NET Web forms junto con Bootstrap. Web Forms forma parte del marco de ASP.NET de aplicaciones web y viene incluido en Visual Studio. Lógicamente, esta interfaz de usuario consume el servicio WFC a través de una referencia de servicio donde se incluye la dirección del servicio que actúa como *endpoint* y que se ha explicado durante lo largo de esta memoria.

Web Forms no dejan de ser páginas que los usuarios solicitan utilizando su explorador, dichas páginas se escriben combinando código HTML, script de cliente, controles de servidor y código de servidor.

Bootstrap es una biblioteca multiplataforma que facilita el diseño de sitios y aplicaciones web, contiene una serie de plantillas y elementos basados en HTML y CSS. Esta biblioteca es una de las más utilizadas en todo el mundo, dos de las organizaciones que la utilizan son la NASA y la MSNBC entre muchas otras.

Como podemos observar en la ilustración 16, se ha creado una interfaz de usuario moderna y con una paleta de colores que aumente la accesibilidad de la página. Además, se ha creado con la intención de guardar cierta simpleza y no sobrecargarla con mucho contenido innecesario.

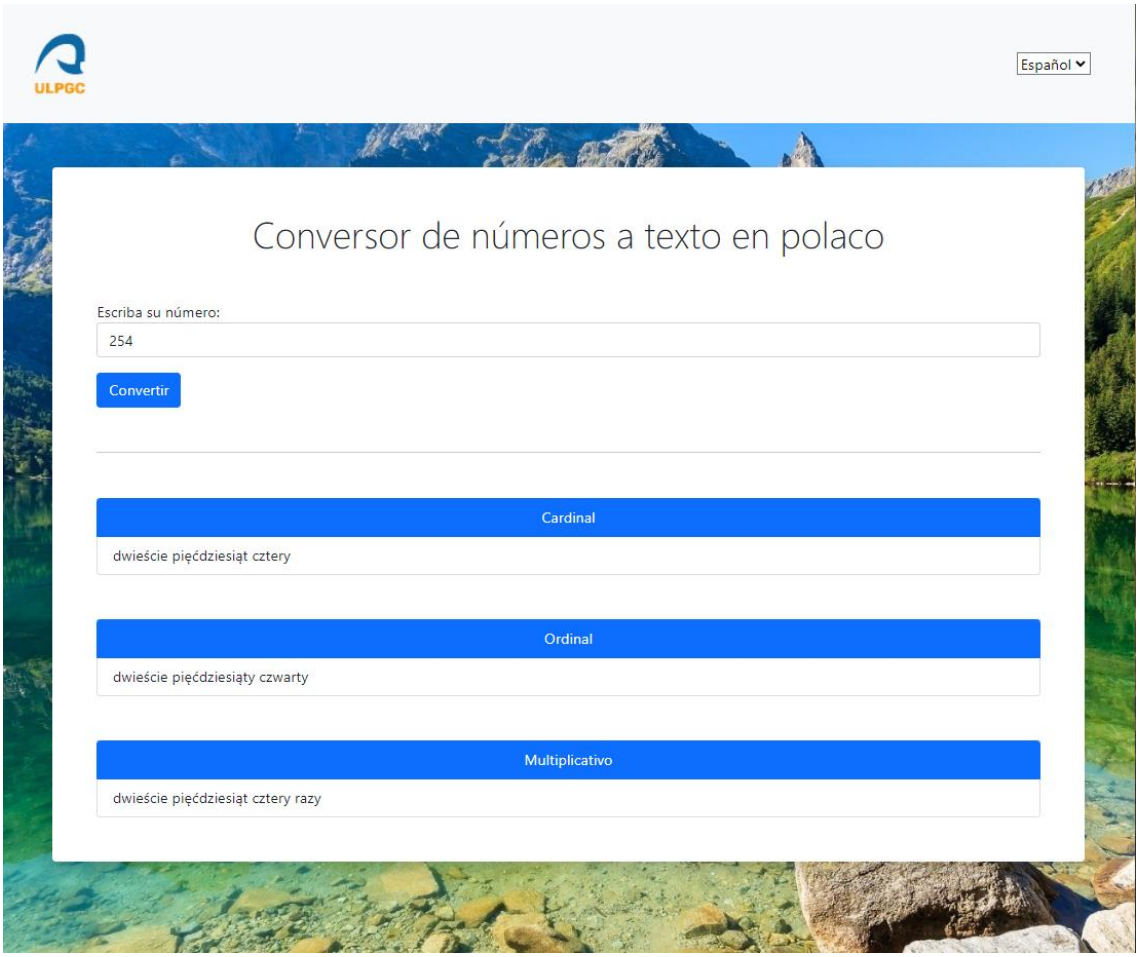


Ilustración 16 - Interfaz de usuario

5.7.1. Cliente responsive.

Hoy en día, la mayoría de personas que consumen páginas web lo hacen desde sus dispositivos móviles o tabletas. Por lo tanto, se hace prácticamente imprescindible que las interfaces de usuario sean responsivas para no perder un amplio público.

Como se comentó anteriormente, *Bootstrap* ha sido la biblioteca elegida para implementar la parte de estilo de la página, este *framework* es muy potente a la hora de crear aplicaciones responsivas.

En la ilustración 17 podemos observar como se ve la interfaz de usuario en un iPhone SE que es de las pantallas más pequeñas que existen en el mercado.

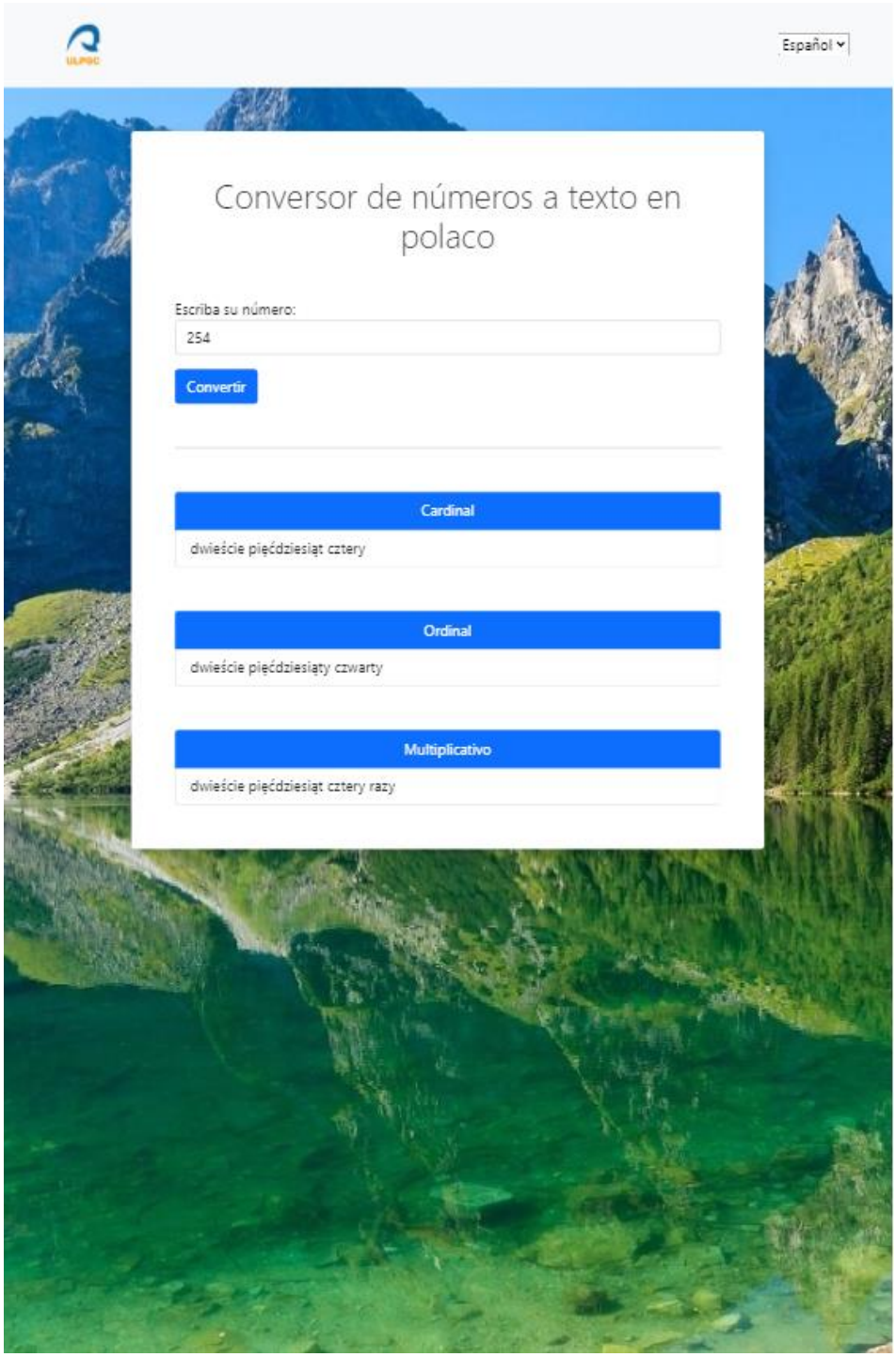


Ilustración 17 - Cliente responsivo

5.7.2. Cliente independiente de los datos.

Al consumir el servicio este devuelve una serie de resultados u otros en función del tipo de número que se desee convertir. Al ser el resultado variable necesitamos que nuestra interfaz sea dinámica y cambie en función del resultado recibido.

Para ello, el cliente ha sido programado para que una vez se hayan recibido los datos se generen de forma dinámica los elementos HTML pertinentes para cada tipo de resultado.

Al arrancar la página web se tienen los elementos que se pueden observar en la ilustración 18, el resto de elementos son generados de forma dinámica.

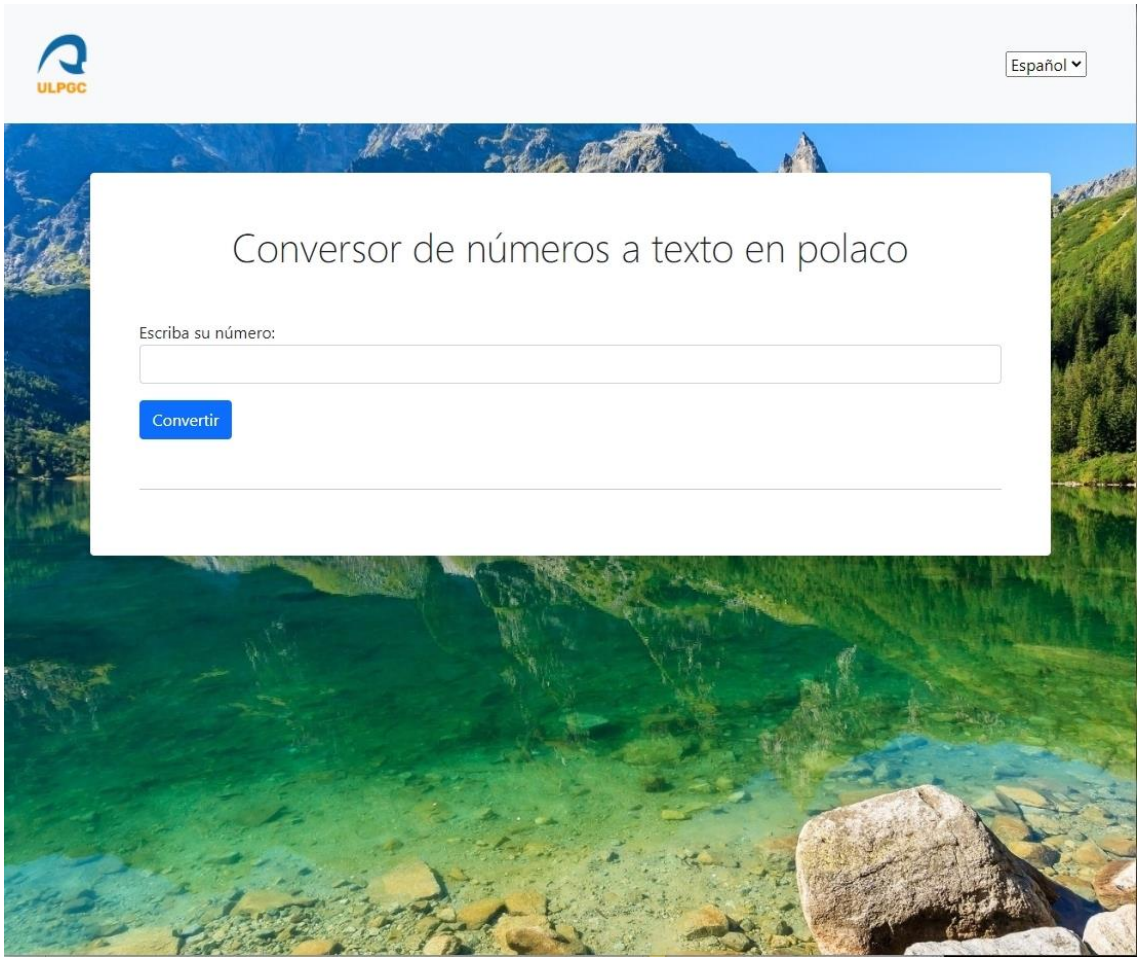


Ilustración 18 - Arranque de la interfaz

5.7.3. Internacionalización de la interfaz.

Como se ha comentado durante el desarrollo de la memoria, esta aplicación permite la multiculturalidad mediante el uso de la clase *CultureInfo* que sirve para especificar que recurso global utilizar para un código de idioma particular.

Como recordatorio, la clase *ServicioNumerosPolaco* tiene un método *Traducir* que recibe tanto el número a convertir como el objeto *CultureInfo*, por lo tanto los resultados de la aplicación pueden ser devueltos en español, inglés y polaco dependiendo del código de lenguaje que se le pase al método. Cabe destacar que si no se especifica ningún código de idioma, el idioma por defecto será el inglés.

El usuario dispone de una lista desplegable en la esquina superior derecha, donde podrá elegir el idioma de la interfaz y además recibirá los resultados por parte del servicio en dicho idioma.

5.8. Prueba de software.

Para probar el servicio se han realizado una serie de pruebas, las cuales, debido a la gran magnitud que tolera el servicio (cifras numéricas de hasta 10^{144} dígitos) no pueden comprobar todas y cada una de las combinaciones posibles pues, la potencia de procesamiento que se necesitaría para ejecutar las pruebas cada vez que se deseara sería demasiado alta.

Debido a lo explicado en el párrafo anterior, se ha tomado un enfoque distinto donde se comprueban la cantidad mínima de casos necesarios para poder asumir que el servicio se comporta de la manera esperada. Es decir, que si al comprobar que los números 1001, 1013, 1043 están formados de forma correcta se interpreta que los números de 1000 a 1099 estarán también formados de forma correcta.

Este proceso se ha aplicado sucesivamente hasta llegar al millón, asumiendo que si los resultados están bien formados hasta ese entonces y, debido a las características que presentan los números en polaco, el resto de números están bien formados pues se trata de un proceso sistemático.

6. Conclusiones y trabajos futuros.

En vista de todo lo documentado en esta memoria podemos obtener una serie de conclusiones:

- La escritura de números a texto en polaco es un proceso sistemático pero que acarrea determinadas excepciones.

- Aunque los términos que se emplean para nombrar números sean finitos, podemos referirnos a cantidades tan exorbitantes que prácticamente no podríamos ni imaginarlas.

Por último y para concluir esta memoria, se podría establecer como trabajo futuro el realizar un programa de reconocimiento de voz que admita como entrada una pregunta con cifras (p.ej. A través de un dispositivo que permita la grabación de audio) y responda el correspondiente texto escrito en palabras.

7. Bibliografía.

De idiomas y números. (s.f.). *Contar en polaco.*

https://www.languagesandnumbers.com/como-contar-en-polaco/es/pol/

Microsoft. (6 de abril de 2022). *Programación asincrónica basada en tareas.* Recuperado el 25 de abril de 2022 de https://docs.microsoft.com/es-es/dotnet/standard/parallel-programming/task-based-asynchronous-programming#:~:text=El%20t%C3%A9rmino%20paralelismo%20de%20tareas,de%20los%20recursos%20del%20sistema.

Microsoft. (6 de abril de 2022). *Biblioteca de procesamiento paralelo basado en tareas (TPL).* Recuperado el 25 de abril de 2022 de https://docs.microsoft.com/es-es/dotnet/standard/parallel-programming/task-parallel-library-tpl

Microsoft. (6 de abril de 2022). *Utilizar la clase StringBuilder en .NET.* Recuperado el 25 de abril de 2022 de https://docs.microsoft.com/es-es/dotnet/standard/base-types/stringbuilder?redirectedfrom=MSDN

Wikipedia. (s.f.). *Expresión regular.* Recuperado el 25 de abril de 2022 de https://es.wikipedia.org/wiki/Expresi%C3%B3n_regular

Bańko, M. (26 de junio de 2012). *słownie czy liczbowo?.* Słownik języka polskiego PWN.

https://sjp.pwn.pl/poradnia/haslo/slownie-czy-liczbowo;13284.html#:~:text=Og%C3%B3lna%20zasada%20jest%20taka%2C%20aby,cyfr%20do%20zapisu%20du%C5%BCych%20liczb

Marek, T. (28 de abril de 2016). *Liczebniki porządkowe.* Językowe dylematy.

https://www.jezykowedylematy.pl/2016/04/tomasz-marek-liczebniki-porzadkowe/

Classicistranieri. (s.f). Nazwy wielkich liczb.

https://www.classicistranieri.com/pl/articles/n/a/z/Nazwy_wielkich_liczb.html

Clevert. (s.f.). *Conversor de números a palabras*.

https://clevert.com.br/t/es/numbers-to-words

Martín, M. (22 de noviembre de 2018). *SOLID: los 5 principios que te ayudarán a desarrollar software de calidad.* Profile.

https://profile.es/blog/principios-solid-desarrollo-software-calidad/

Atlassian. (s.f.). *¿Qué es el control de versiones?*

https://www.atlassian.com/es/git/tutorials/what-is-version-control

Wikipedia. (s.f.). *Bootstrap (framework).* Recuperado el 25 de abril de 2022 de https://es.wikipedia.org/wiki/Bootstrap_(framework)

Wikipedia. (s.f.). *Windows Communication Foundation.* Recuperado el 25 de abril de 2022 de https://es.wikipedia.org/wiki/Windows_Communication_Foundation