Universidade Vigo

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria del Trabajo de Fin de Grao que presenta

D. Alfonso Rúa Martínez

para la obtención del Título de Graduado en Ingeniería Informática

Desarrollo de técnicas para reconocimiento de entidades nombradas en textos

Julio, 2021

Trabajo de Fin de Grao Nº: EI 20/21-101

Tutor/a: Rosalía Laza Fidalgo
Área de conocimiento: Lenguajes y Sistemas Informáticos
Departamento: Informática

Agradecimientos

Quisiera agradecer a **Rosalía**, **Reyes** y **Moncho**, investigadores del grupo SING, quienes me han acogido para hacer este proyecto, me han dedicado su tiempo y me han apoyado. Ha sido corto pero intenso.

A **Iria**, por estar siempre a mi lado.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Actualmente, el paradigma informático conocido como la revolución de los datos o Big Data[1] está en auge. Bajo esta filosofía, cada vez se genera y guardan una mayor cantidad de datos digitales con el objetivo de intentar sacarles algún tipo de partido. En muchos casos, la información guardada no está estructurada (por ejemplo, se guarda de forma textual) por lo que su análisis no resulta una tarea sencilla.

Los datos textuales se procesan típicamente mediante técnicas de Inteligencia Artificial (IA). Sin embargo, para su análisis, los datos deben estar previamente estructurados en formato matriz ("datasets"). El proceso de estructuración de datos recopilados de distintas fuentes (Tweets[2], e-mails, comentarios de YouTube, etc) se engloba en las tareas de pre-procesamiento y representación e implican la utilización de técnicas de Procesamiento de Lenguaje Natural (NLP[3], Natural Language Processing).

Existen multitud de herramientas que se pueden emplear para realizar el preprocesamiento y representación de los datos. Entre ellas, varios profesores de la Escuela Superior de Ingeniería Informática (ESEI) han desarrollado una herramienta conocida como NLPA[4] (Natural Language Pre-processing Architecture) que está disponible para su descarga. NLPA incorpora multitud de técnicas de NLP para pre-procesar y representar los datos textuales. Sin embargo, el soporte para algunas técnicas como el Reconocimiento de Entidades Nombradas (NER[5], Name Entity Recognition) tiene varias limitaciones. Particularmente, el soporte actual de esta tarea se hace mediante el uso de la librería de NLP desarrollada por la Universidad de Standford que incorpora modelos inteligentes para efectuar el reconocimiento NER. Al usar Maven, los modelos se descargan serializados durante el proceso de construcción del software. El principal inconveniente de estos modelos es su peso, ya que ocupan una gran cantidad de espacio de almacenamiento y de memoria RAM (Random Access Memory). Asimismo, también es necesario tener en cuenta que son sistemas de tipo "caja negra", es decir, no es posible ver su funcionamiento interno. Sin embargo, además de emplear modelos inteligentes como los CRF (Conditional Random Fields), es posible desarrollar NER usando mecanismos basados en reglas.

El uso de técnicas basadas en reglas[6] para la implementación de modelos NER permite que el proceso funcione de forma más precisa, transparente y eficiente que las técnicas basadas en modelos inteligentes.

Tomando como base el interés en la realización de procesamiento NER de textos aplicando técnicas basadas en reglas, surge la idea de este Trabajo de Fin de Grado, en adelante TFG, cuyo fin es encontrar entidades Leet speak en textos.

Leet speak[7] (l337 en u jerga), consiste en realizar modificaciones en las palabras, normalmente substituyendo letras por números y símbolos, de manera que no sean detectadas por un computador, pero sí sean reconocibles al ojo humano. Un ejemplo sería la frase "mi casa es azul" que se podría modificar a "m¡ c4$@ 3s 4zuL".

La principal utilidad de esto tipo de lenguaje es la de esconder determinadas palabras frente a un computador. Ejemplos de ello son:

Un e-mail publicitario que pretende que el filtro anti-spam[8] no detecte el producto que se oferta (ej: la palabra viagra, es sustituida por la palabra v¡agra), jugando a menudo con la fuente de letra para que el ojo humano lea perfectamente la palabra.

El título de un video en Youtube o Instagram en el que se desean escribir palabras censuradas[9], por ejemplo “sex” o “porn” y se adaptarían como “se×” o “p0rn”, para no ser detectadas por los algoritmos de censura de estas plataformas.

1.1 Objetivos

En este apartado se recoge el objetivo principal que se persigue en el TFG junto con los subobjetivos necesarios para poder llevarlo a cabo.

1.1.1 Objetivo principal

El objetivo principal de este TFG es averiguar si un texto contiene entidades de tipo Leet speak o no, y en caso afirmativo detectar cuales y a que palabras hacen referencia.

Como se ha mencionado en la introducción, este trabajo está enmarcado dentro del proyecto NLPA, el cual ya contenía tareas que se encargaban de localizar el texto Leet. No obstante, como también se ha comentado, estas tareas dependientes del modelo NLP de Standford consumen una gran cantidad de recursos, principalmente RAM.

Para resolver estos inconvenientes, este TFG desarrollará una implementación propia en base a reglas, que no dependerá de tecnologías de terceros y se ejecutará de forma eficiente en el menor tiempo posible.

1.1.2 Objetivos específicos

Segmentar palabras

Es un proceso conocido como *tokenización*, consistente en dividir el texto en partes más pequeñas, en este caso palabras. Con esto conseguimos la unidad lógica con la cual trabajaremos, puesto que una palabra leet no depende de que la palabra anterior o siguiente también lo sean.

En otro nivel, también segmentaremos cada palabra en todos las subpalabras posibles, de forma que podamos determinar si cada conjunto de caracteres consecutivos está actuando o no como un carácter leet.

Obtener diccionarios de símbolos leet

Para saber si una palabra es una candidata a ser una palabra leet, tenemos que saber primero que símbolos pueden sustituir a que letras de forma que sean similares para el ojo humano. Para ello se recopilarán en archivos JSON todas las letras con sus respectivos caracteres o conjunto de caracteres leet equivalentes.

Obtener diccionarios de palabras

La lógica de nuestro procesado se basa en, una vez encontradas todas las posibles palabras resultantes de hacer los intercambios leet, decidir si alguna de ellas es una palabra real. Para lo cual echaremos mano de diccionarios que contengan un gran volumen de palabras y estén disponibles en varios idiomas.

Encontrar todas las posibles palabras que surgen de una palabra leet

Para encontrar todas las posibles palabras en las que una palabra leet puede resultar, es necesario realizar todas las permutaciones de caracteres leet encontrados en la palabra original. Por ejemplo, la palabra l33t:

Puede ser traducida a “leet” puesto que los “3” actúan como “e”; pero también puede ser traducida como “bet” ya que la “l3” equivale a una “B”.

Traducir el texto cambiando las palabras leet por sus equivalentes

Tras haber encontrado las palabras leet y sus equivalentes, se debe ofrecer la opción de devolver el texto traducido, cambiando las palabras leet por las palabras a las que hacen referencia. Ello debe poder decidirse a la hora de crear la tarea que detecta leet.

Crear una aplicación web

Se desarrollará una aplicación web para hacer más cómoda la realización de pruebas, así como demostraciones del algoritmo y posibilitar el análisis y traducción de textos por parte de un usuario no técnico.

Asimismo, se pondrá dicha aplicación web a disposición del público a través de un dominio.

1.2 Resumen de la solución propuesta

Para lograr la consecución de los objetivos anteriormente mencionados, se realizará el desarrollo de un pipe que lleve a cabo todas las tareas necesarias para ello.

Como mencionábamos, este proyecto forma parte de NLPA, el cual se basa en una arquitectura de pipeline, es por ello por lo que la tarea de detección de leet ha de desarrollarse mediante un pipe, de forma que pueda ser incorporada al proyecto NLPA con facilidad.

Para que el pipe de detección de entidades Leet pueda funcionar, es preciso previamente saber en qué idioma está escrito el texto, por ello es necesario que este pipe vaya precedido de otros pipes disponibles en NLPA que se encargan de estas tareas.

La Figura 1 muestra de forma simplificada el funcionamiento de la solución propuesta.

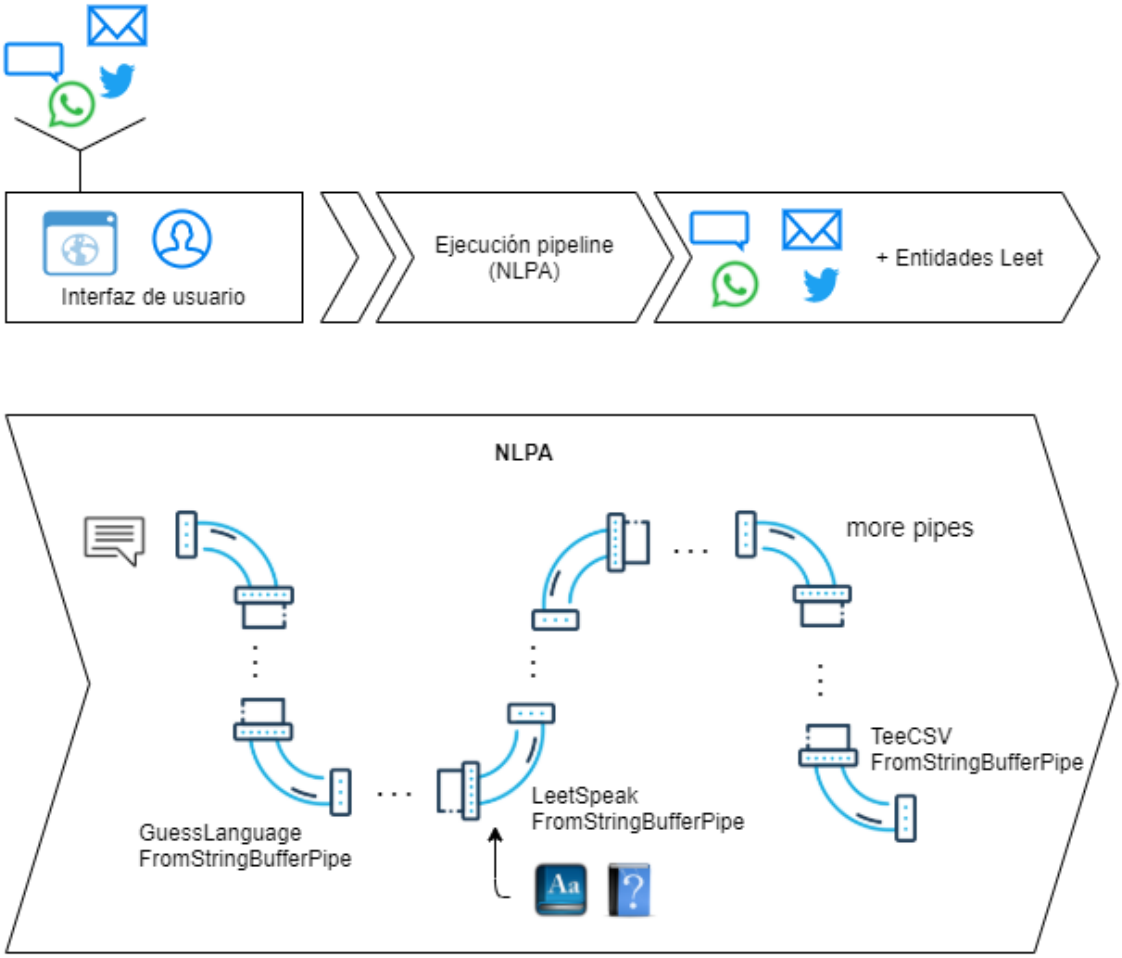

Figura 1. Esquema de la solución propuesta

La lógica del pipe *LeetSpeakFromStringBufferPipe* podría resumirse de la siguiente manera:

El texto original llega al pipe, junto con la propiedad que nos indica en que idioma está escrito. Antes de procesar el texto, se crean estructuras de datos a partir de los diccionarios tanto de palabras en varios idiomas como de símbolos y equivalencias leet, puesto que serán comunes para todos los textos a procesar.

El texto se divide en palabras, y cada una de estas palabras en subpalabras, ej:

l33t -> l, 3, 3, t, l3, 33, 3t, l33, 33t, l33t

Cada una de estas subpalabras se compara con las equivalencias leet, de forma que, si coincide alguna, se anota su posición y su equivalencia, ej:

3 -> e l3-> b

Con ello se calculan todas las palabras resultantes de realizar las permutaciones posibles:

l33t -> le3t, l3et, leet, b3t, bet

Ahora se comprueba si alguna de estas palabras es una palabra real con ayuda de los diccionarios, en nuestro caso leet y bet lo son, con lo cual se anotan en la propiedad de salida del pipe.

Se repiten estos pasos para cada palabra en la que se ha dividido el texto.

Finalmente, en caso de que se requiera la traducción, se procede a cambiar las palabras leet encontradas por sus equivalentes.

Por otro lado, para cumplir todos los objetivos de este proyecto, ha de llevarse a cabo el desarrollo de una aplicación web que sea simple para el público general. Dicha aplicación mostrará dos cuadros de texto; a la izquierda se podrá introducir texto en leet que será traducido en texto normal mediante el botón “Descifrar” y en el cuadro de la derecha se podrá introducir texto normal para que nos sea devuelto texto en formato leet mediante el botón “Cifrar” (en el Anexo I se muestra cómo se lleva a cabo esta otra tarea).

1.3 Metodología usada

A la hora de desarrollar un producto de software es crucial seleccionar una metodología adecuada a las necesidades de nuestro proyecto, que nos ayude a planificarlo y controlar su desarrollo.

Se ha optado por la metodología Scrum[10], una de las metodologías ágiles más reconocidas para la gestión de proyectos. Scrum se basa en entregas iterativas e incrementales del producto denominadas “sprints” con duración de una a tres semanas.

El funcionamiento consiste en determinar los requisitos del producto (historias de usuario) para conformar el Backlog de Producto. Se seleccionan las historias a tratar en el siguiente sprint y se descomponen en tareas. Es importante no introducir cambios en los requisitos ni en las tareas durante la duración de un sprint. Otra actividad importante es la revisión del sprint, para garantizar la calidad del producto.

Los roles que intervienen en el proceso son, el Product Owner, encargado de elegir las funcionalidades a desarrollar y responder a las dudas del equipo de desarrollo; el equipo de desarrollo es el encargado de desarrollar el software; y el Scrum Master, que es el encargado de dirigir al equipo de desarrollo y comunicarse con el Product Owner.

En la Figura 2 se muestra de forma gráfica el funcionamiento de Scrum.

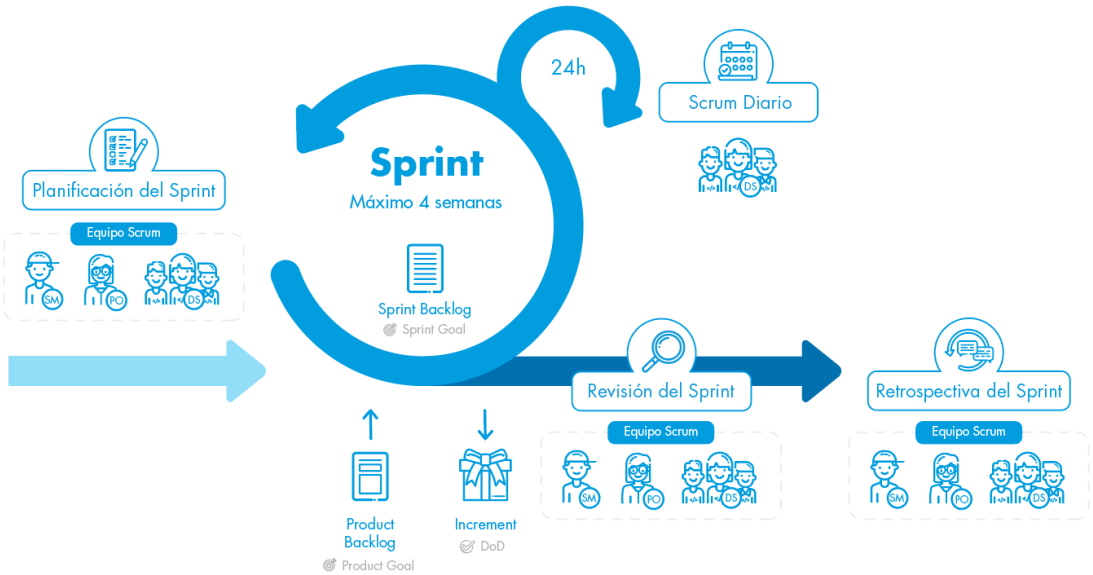

Figura 2. Funcionamiento de metodología Scrum.

Una vez expuestas las bases de Scrum, es necesario realizar ciertos cambios para ajustarnos al ámbito de un TFG, a saber:

El equipo de desarrollo en este caso está conformado por una única persona, el alumno Alfonso Rúa Martínez, el cual a su vez asumirá el rol de Scrum Master. Por otro lado, Jose Ramón Méndez, Rosalía Laza y Reyes Pavón, investigadores del grupo SING, toman el rol de Product Owner.

Las reuniones Daily se llevan a cabo mediante reuniones mensuales o quincenales en las que el alumno muestra los resultados del trabajo desarrollado y presenta las dudas o problemáticas que hayan surgido durante ese período.

Los gráficos de Burndown típicos de la metodología Scrum son sustituidos por diagramas de Gantt[11], para facilitar una visión general del progreso de las tareas.

Con todos estos cambios, la metodología Scrum se adapta tanto a las necesidades del proyecto, como a las necesidades del TFG, como a la disponibilidad de alumno y tutores.

2. Planificación y seguimiento

Se tratará a continuación la planificación del proyecto, así como la comparativa con la ejecución real, pudiendo ser visualizado de forma clara mediante tablas y diagramas de Gantt.

2.1 Planificación

La planificación sigue la base de los sprints de la metodología scrum descrita anteriormente. En concreto, se ha optado por el desarrollo a lo largo de ocho sprints, cada uno con una duración de dos semanas y con una carga de 20 horas semanales (equivalente a media jornada), lo que conforman 40 horas de trabajo por sprint y 320 horas para el proyecto en su conjunto.

Antes de la planificación de los sprints se ha construido el backlog de producto a través de historias de usuario, en las cuales se incluye una estimación de las horas de trabajo necesarias para completarlas mediante PH (puntos de historia) equivalentes a una jornada de 8 horas cada uno.

A continuación, se muestra la tabla de sprints planificados y el diagrama de Gantt.

Tabla 1. Planificación de Sprints

SPRINT	Fecha inicio	Fecha fin	PH
Sprint 1	29/03/2021	09/04/2021	5
crear diccionario de palabras	29/03/2021	30/03/2021	1
crear diccionario de equivalencias leet	31/03/2021	05/04/2021	2
crear esqueleto del pipe leet	06/04/2021	09/04/2021	2
Sprint 2	12/04/2021	23/04/2021	5
añadir funcionalidad identificación leet	12/04/2021	23/04/2021	5
Sprint 3	26/04/2021	07/05/2021	5
añadir funcionalidad traducción	26/04/2021	07/05/2021	5
Sprint 4	10/05/2021	21/05/2021	5
crear esqueleto interfaz web	10/05/2021	21/05/2021	5
Sprint 5	24/05/2021	04/06/2021	5
optimizar pipe leet	24/05/2021	04/06/2021	5
Sprint 6	07/06/2021	18/06/2021	5
crear pipe reverse leet	07/06/2021	10/06/2021	2
adaptar interfaz web al uso de NLPA	11/06/2021	18/06/2021	3
Sprint 7	21/06/2021	02/07/2021	5
documentación	21/06/2021	24/06/2021	2
testeo	25/06/2021	02/07/2021	3
Sprint 8	05/07/2021	16/07/2021	5
documentación(cont)	05/07/2021	12/07/2021	3
creación de manuales	13/07/2021	16/07/2021	2
TOTAL	29/03/2021	16/07/2021	40

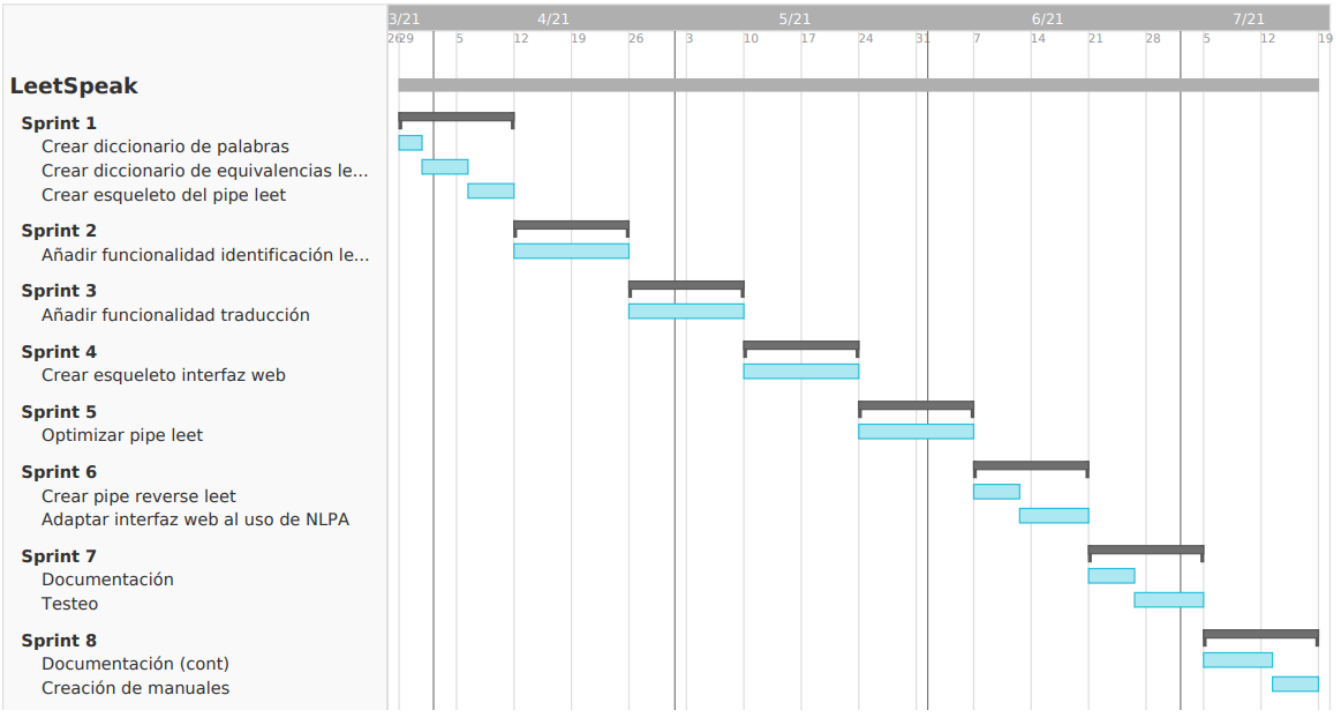

Figura 3. Diagrama de Gantt planificación.

2.2 Seguimiento

Ahora se muestra la tabla de sprints y diagrama de Gantt correspondientes a la ejecución real del proyecto.

Tabla 2. Ejecución de Sprints

SPRINT	Fecha inicio	Fecha fin	PH
Sprint 1	29/03/2021	09/04/2021	5
crear diccionario de palabras	29/03/2021	30/03/2021	1
crear diccionario de equivalencias leet	31/03/2021	05/04/2021	2
crear esqueleto del pipe leet	06/04/2021	09/04/2021	2
Sprint 2	12/04/2021	23/04/2021	5
añadir funcionalidad identificación leet	12/04/2021	23/04/2021	5
Sprint 3	26/04/2021	07/05/2021	5
añadir funcionalidad traducción	26/04/2021	07/05/2021	5
Sprint 4	10/05/2021	21/05/2021	5
crear esqueleto interfaz web	10/05/2021	21/05/2021	5
Sprint 5	24/05/2021	04/06/2021	5
optimizar pipe leet	24/05/2021	04/06/2021	5

Sprint 6	07/06/2021	18/06/2021	5
crear pipe reverse leet	07/06/2021	10/06/2021	2
adaptar interfaz web al uso de NLPA	11/06/2021	18/06/2021	3
Sprint 7	21/06/2021	02/07/2021	5
documentación	21/06/2021	24/06/2021	2
testeo	25/06/2021	02/07/2021	3
Sprint 8	05/07/2021	16/07/2021	5
documentación(cont)	05/07/2021	12/07/2021	3
creación de manuales	13/07/2021	16/07/2021	2
TOTAL	29/03/2021	16/07/2021	40

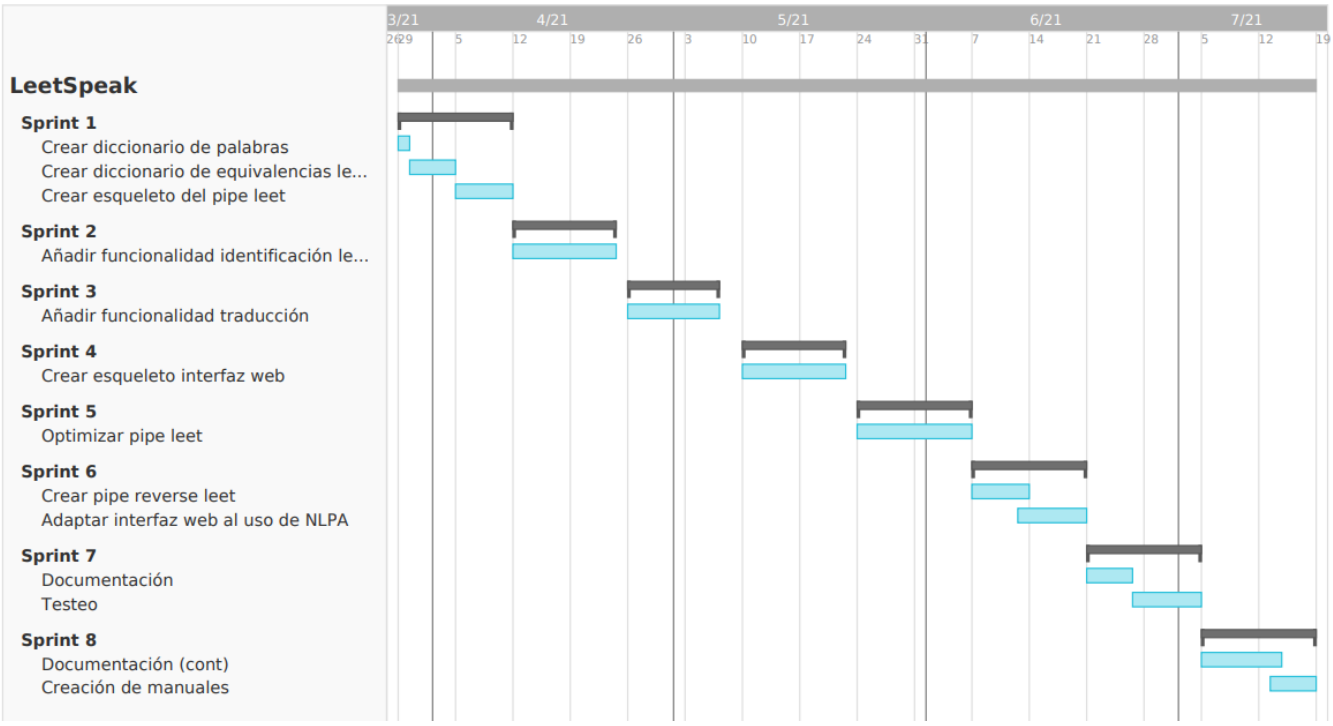

Figura 4. Diagrama de Gantt ejecución.

Como se puede observar, el tiempo total empleado ha sido menor que el estimado en la planificación, puesto que las estimaciones han seguido una lógica conservadora basada en la sucesión de Fibonacci[12].

No obstante, también se puede observar cómo determinadas tareas han consumido más tiempo del que se estimaba, producto generalmente de la inexperiencia con los frameworks utilizados para determinadas tareas.

3. Arquitectura y tecnologías

En esta sección se tratará la arquitectura de la solución desarrollada para cumplir los objetivos expuestos anteriormente, además, se tratarán las tecnologías y productos de terceros empleados para tal fin.

3.1 Arquitectura del sistema

La arquitectura que se emplea en este proyecto es la denominada arquitectura en pipeline[13]. Esta, se basa en dividir la tarea de procesamiento en subtareas más pequeñas denominadas pipes, que procesan una secuencia de datos y su salida se corresponde con la entrada al siguiente pipe.

En la Figura 5 se muestra el diagrama de flujo de nuestro sistema.

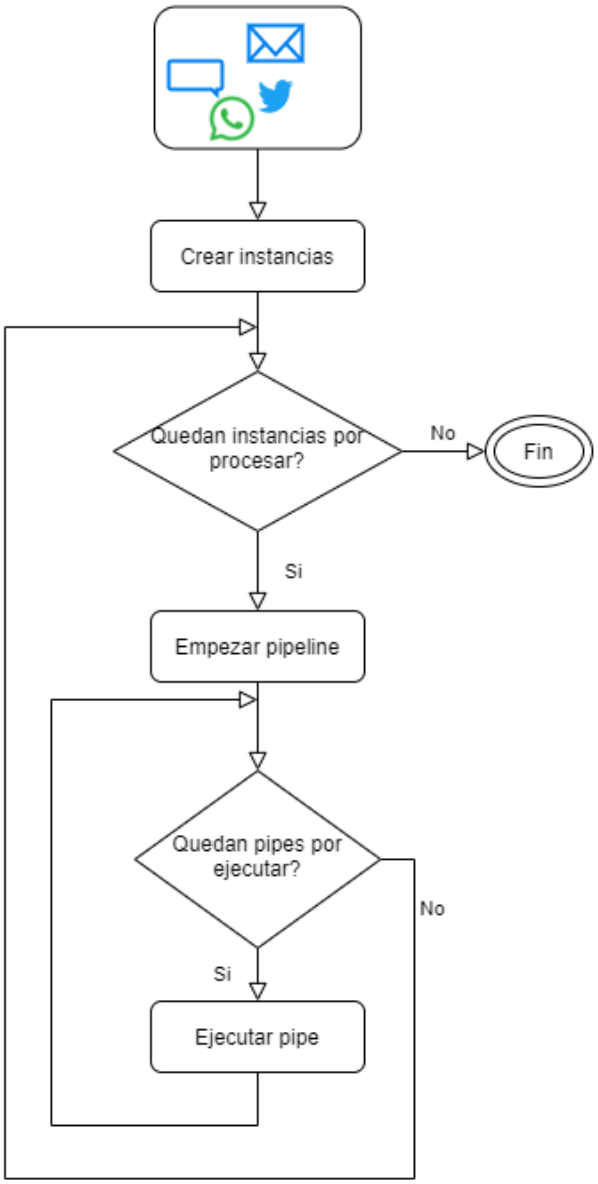

Figura 5.Diagrama de flujo del sistema.

En ella podemos ver como los textos provenientes de diversas fuentes se transforman en instancias, las cuales permiten el tratamiento de forma unificada para los datos. Tras esto, las diferentes instancias creadas entran en el pipeline, para ser procesadas por los pipes contenidos en él. En caso de que el lenguaje detectado no esté soportado, la instancia esté corrupta o exista alguna excepción no controlada en un pipe, la instancia que ha producido dichos problemas es invalidada y se aborta su ejecución, pasando a introducir en el pipeline la instancia siguiente. Este proceso se repite para todas las instancias, siendo capaz de efectuarse de forma concurrente.

Por otra parte, para implementar la interfaz de usuario ha sido necesario emplear una arquitectura diferente, denominada MVC[14] (Modelo Vista Controlador).

En la Figura 6 se muestra el diagrama de la arquitectura MVC.

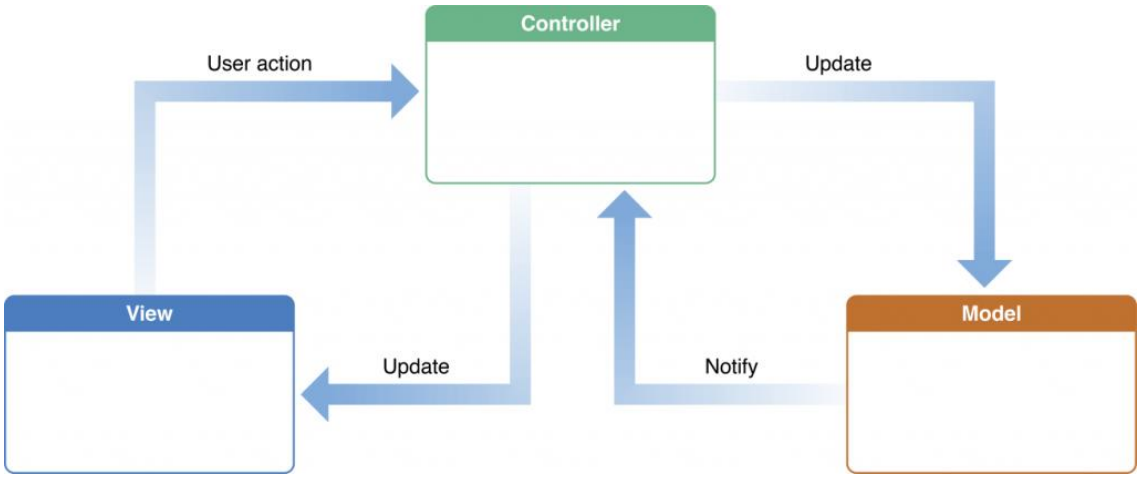

Figura 6. Esquema de arquitectura MVC

Esta arquitectura se compone de tres elementos:

La vista, muestra los datos y permite la interacción con el usuario. El modelo, es el encargado de alojar los datos. El controlador, es el responsable de comunicar vista y modelo, solicitando los datos al modelo y comunicándoselos a la vista.

En nuestro caso, debido al uso de Spring Boot y las necesidades de nuestra aplicación, este esquema se simplifica como podemos ver en la Figura 7.

Sin la necesidad de un modelo que almacene información en una base de datos, podemos trabajar directamente manipulando los datos obtenidos de la vista.

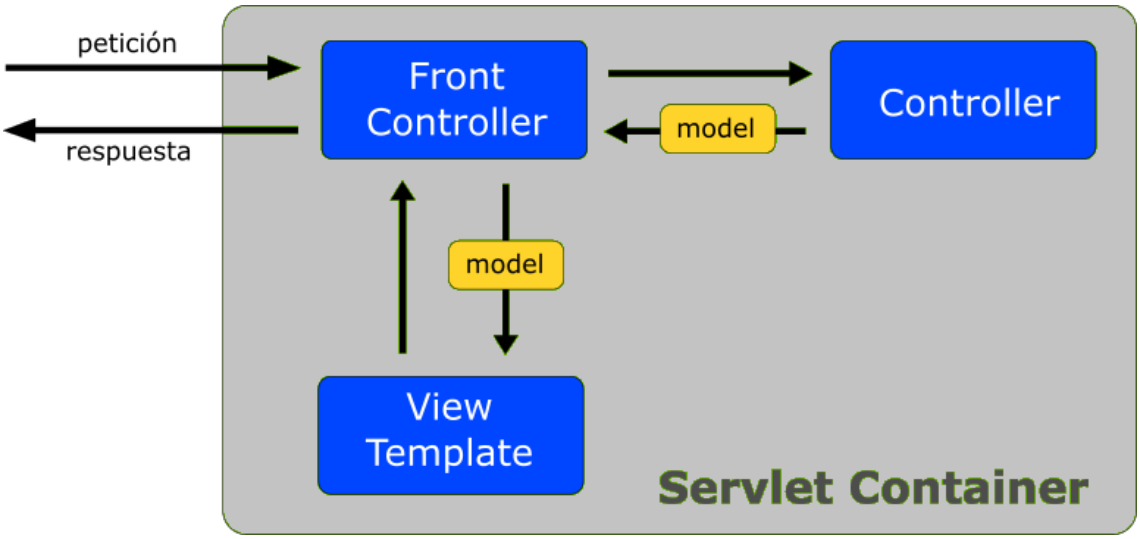

Figura 7. Modelo MVC adaptado a SpringBoot

3.2 Tecnologías e integración de productos de terceros

En este apartado se describen las tecnologías y herramientas de terceros empleados durante el desarrollo de este proyecto.

Java: lenguaje de programación creado en 1995. Es el más utilizado a fecha de hoy con más de 9 millones de desarrolladores. Las aplicaciones escritas en java se compilan en bytecode, lo que permite su ejecución en cualquier máquina JVM (Java Vitrual Machine), independientemente de la arquitectura o sistema operativo del equipo anfitrión. La gran mayoría del código desarrollado en este proyecto está escrito en Java.[15]

Javax.json: paquete del lenguaje Java que permite el acceso y creación de archivos JSON a través de una API. Ha sido utilizado para la gestión de los diccionarios de equivalencias leet. [16]

BDP4J: Se trata de un proyecto del grupo SING que implementa una arquitectura de pipeline que permite la ejecución de tareas en serie o en paralelo. Permite ser usado como framework para incorporar nuevos pipes. Define la arquitectura empleada en el desarrollo de este proyecto.[4]

NLPA: es un plugin diseñado para su uso mediante BDP4J y que incorpora pipes para el preprocesamiento de texto. Algunos de estos pipes son utilizados en este proyecto y, en general, son la base para los nuevos pipes desarrollados.[4]

Spring Boot: es una herramienta que permite crear un proyecto mediante Spring Framework pero eliminando configuraciones repetitivas requeridas a la hora de desplegar la aplicación. A su vez, Spring Framework es un framework de java que facilita el desarrollo de aplicaciones web. En este proyecto se ha optado por Spring Boot para desarrollar la interfaz gráfica.[17]

BootStrap: es un kit de herramientas de código abierto para desarrollos web limpios y responsive con HTML, CSS y JavaScript. Con él se puede crear la aplicación web mediante el uso de sus componentes modulares, como menús, botones, formularios… Es usado ampliamente por programadores ”front-end” debido a su facilidad de uso y resultados adaptables a cualquier navegador. [18]

Git: es un sistema de control de versiones distribuido y de código abierto, el cual es usado como herramienta de colaboración entre desarrolladores. Dispone de una funcionalidad de ramas muy útil para el desarrollo simultaneo entre varias personas. En concreto se ha utilizado GitHub para almacenar el código desarrollado en un repositorio remoto. [19]

Maven: es una herramienta que automatiza la creación de un proyecto de software, como es la importación de dependencias, la compilación del código o el empaquetado en archivos JAR. Asimismo, provee una estructura de directorios estándar para organizar los archivos.[20]

Visual Studio Code: es un editor de texto desarrollado por Microsoft, muy popular entre los desarrolladores de software debido a la gran versatilidad que le aportan sus numerosos plugins. Ofrece numerosas herramientas como el autocompletado inteligente o la integración con cualquier lenguaje, en concreto para Java existe un plugin desarrollado por RedHat. Todo el código de este proyecto ha sido escrito y testado en VS Code.[21]

Microsoft Word: es un programa para el procesamiento de textos. Ha sido utilizado para la creación de este documento debido a la sencillez a la hora de formatear el texto y referenciarlo.

Visual Paradigm: es una herramienta UML que permite la creación de todo tipo de objetos relacionados con el desarrollo de software. En concreto, ha sido utilizada para la creación de los diagramas de clase y de secuencia presentes en este documento.[22]

TeamGantt: es una aplicación online para la creación de diagramas de Gantt, permite la gestión de las tareas y el tiempo dedicado a estas. Ha sido utilizada para la elaboración de los diagramas de Gantt de este proyecto por su simplicidad a la hora de crear y exportar los diagramas.[23]

4. Especificación y análisis de requisitos

En este apartado se definen las historias de usuario[24] que formarán el Backlog de producto, su estructura contiene:

- Nombre breve y descriptivo.
- Descripción de la funcionalidad (“como X quiero Y para Z”)
- Criterios de aceptación, que determinarán cuando la funcionalidad está terminada y es aceptable por el cliente.

Existen dos roles de usuario considerados en este proyecto:

El primero es el rol de investigador, una persona con conocimientos de Leet speak, NER y preprocesado de texto, el cual usará el software que se desarrolla en este proyecto para procesar textos a gran escala, junto con otros pipes, en un entorno de consola de comandos (ej: para filtrar miles de correos electrónicos en busca de spam).

Por otro lado, está el rol de usuario común, que es cualquier persona que quiera probar la aplicación desde una interfaz gráfica sencilla y sin necesidad de realizar ninguna instalación.

Con todo esto, y con el fin de dar solución a los objetivos planteados en el primer apartado, se crean las siguientes historias de usuario:

HU01- Crear diccionario de palabras

COMO investigador

QUIERO tener diccionarios de palabras

PARA saber qué palabras existen en cada idioma

Criterios de aceptación

- Cada diccionario contiene la mayor cantidad posible de palabras de un idioma (del orden de 1.000.000)
- Se tienen diccionarios para los idiomas soportados en NLPA

HU02-Crear diccionario de equivalencias leet

COMO investigador

QUIERO tener un diccionario de equivalencias leet

PARA saber qué símbolos equivalen a que letras

Criterios de aceptación

- El formato del diccionario es JSON, para facilitar su acceso desde el pipe.
- El diccionario contiene, por cada entrada, un conjunto de símbolos y sus letras equivalentes.

HU03-Crear pipe leet speak

COMO investigador

QUIERO disponer de un pipe de leet speak

PARA poder localizar las entidades leet entro de un texto y utilizar dicha información

Criterios de aceptación

- Debe poder seleccionarse o no la traducción del texto mediante una variable de entrada.
- La propiedad de salida debe incluir todas las entidades leet localizadas en el texto.
- Si no existe diccionario para el lenguaje detectado, no se efectuará ningún cálculo ni alteración sobre el texto.

HU04-Añadir funcionalidad identificación leet

COMO investigador

QUIERO que se identifique de forma correcta y completa todas las entidades leet

PARA poder hacer uso de esta información

Criterios de aceptación

- Todas las entidades leet que contenga el texto son identificadas y asociadas a sus equivalentes.
- Si la equivalencia de la entidad leet no es una palabra del idioma asociado al texto, entonces esta entidad no será identificada como leet.

HU05-Añadir funcionalidad traducción

COMO investigador

QUIERO obtener un texto normal equivalente al texto con entidades leet

PARA utilizar el texto equivalente para un procesado completo mediante machine learning

Criterios de aceptación

- Para un texto de entrada, el texto de salida es alterado de forma que las entidades leet son sustituidas por las palabras equivalentes.

- En caso de que haya más de una palabra equivalente posible para una entidad leet, se mostrarán todas las posibles equivalencias separadas por barras verticales. Por ejemplo: “l33t” se traduce en “leet|bet”.

HU06-Crear interfaz web

COMO usuario común

QUIERO disponer de una interfaz web

PARA poder probar a traducir texto leet a texto normal y viceversa de forma intuitiva

Criterios de aceptación

- La interfaz permite la introducción de texto para su traducción en ambos sentidos.

- La interfaz ha de ser simple, multidispositivo y responsive.

- La interfaz con sus funcionalidades ha de estar disponible en un dominio de internet.

HU07-Crear pipe reverse leet

COMO usuario común

QUIERO poder crear mis propias entidades leet en base a un texto normal

PARA hacer uso de las cualidades cripticas del leet speak

Criterios de aceptación

- Para un texto de entrada, el texto de salida es alterado de forma que la mitad de los caracteres de las palabras son transformados en símbolos leet equivalentes.

5. Diseño del software

En este apartado se muestran los aspectos más importantes del diseño del software desarrollado.

En la Figura 8, se ofrece una visión global de como interaccionan las distintas clases y componentes del sistema. Se puede ver en un tono grisáceo las clases pertenecientes al proyecto del grupo SING, y en blanco los desarrollos específicos para este TFG.

A continuación, se describen las clases que se han añadido junto con una explicación de sus métodos y atributos.

LeetSpeakFromStringBufferPipe: es la clase que se encarga de la detección y traducción de las entidades Leet contenidas en un texto. Almacena dichas entidades detectadas en una propiedad de la instancia procesada.

- HashMap<String, LinkedList<String>> rosetta: contiene las entidades leet detectadas en el texto y sus traducciones.

- HashMap<Pattern, LinkedList<String>> dictionary: contiene el alfabeto junto con sus equivalentes Leet.

- HashMap<String, LinkedList<Pattern>> words: contiene una lista de palabras para cada idioma del que se posea diccionario.

- String langProp: parámetro que indica el idioma empleado en el texto.

- String leetProp: parámetro que indica el nombre de la propiedad que almacena las entidades Leet identificadas en el texto.

- Boolean translateLeetFlag: parámetro que indica la elección de traducción. True implicará la traducción del texto y False únicamente la identificación de las entidades Leet y su anotación en la propiedad antes indicada.

- Instance pipe(Instance carrier): método que procesa la instancia carrier según los parámetros mencionados.

ReverseLeetSpeakFromStringBufferPipe: es la clase encargada de la transformación de un texto mediante equivalentes Leet.

- HashMap<String, LinkedList<String>> dictionary: contiene el alfabeto junto con sus equivalentes Leet.

- Instance pipe(Instance carrier): método que procesa la instancia carrier de acuerdo a la funcionalidad indicada.

Por otra parte, se explican las clases correspondientes con el desarrollo de la interfaz web, se omite la explicación de la vista mediante un HTML, un CSS y un JS debido a su simplicidad. [25]

LeetApplication: simplemente posee el main de la aplicación con una llamada a SpringApplication.run que inicia la interfaz de usuario.

MainController: clase que interacciona con la vista uniéndola con las funcionalidades que esta ofrece a través de MainTransformer.

- @GetMapping("/") public String homePage(Model model): método que limpia los datos de la vista y la refresca.

-@PostMapping("/getTexto") public String getTexto(String textIn, Model model): método que se ejecuta al pulsar el botón “Descifrar” y que toma el texto presente en el cuadro de texto izquierdo, llama a transformToLeet() de la siguiente clase y muestra el resultado en el cuadro de texto derecho; refrescando la página.

-@PostMapping("/getLeet") public String getLeet(String textOut, Model model): método que se ejecuta al pulsar el botón “Cifrar” y que toma el texto presente en el cuadro de texto derecho, llama a transformToNormal() de la siguiente clase y muestra el resultado en el cuadro de texto izquierdo; refrescando la página.

MainTransformer: clase que almacena la lógica y pipes a procesar, y la ofrece mediante los métodos transformToLeet y transformToNormal.

- String transformToLeet(String textIn): método que recibe texto y lo transforma a su equivalente empleando entidades Leet mediante el pipe *ReverseLeetSpeakFromStringBufferPipe.*

- String transformToNormal(String textIn): método que recibe texto con entidades Leet y lo traduce a texto normal.

En la Figura 9 podemos ver un diagrama de secuencia de la traducción del texto Leet. Para facilitar la visualización se han obviado los pipes previos (ya existentes en NLPA) que son necesarios antes de la ejecución del pipe *LeetSpeakFromStringBufferPipe*, como muestra de ellos se representa el pipe *GuessLenguageFromStringBufferPipe*.

De forma análoga, resulta el funcionamiento del proceso inverso, en el que se modifica un texto normal a uno con equivalentes Leet, sin más que cambiar el pipe *LeetSpeakFromStringBufferPipe* por *ReverseLeetSpeakFromStringBufferPipe*.

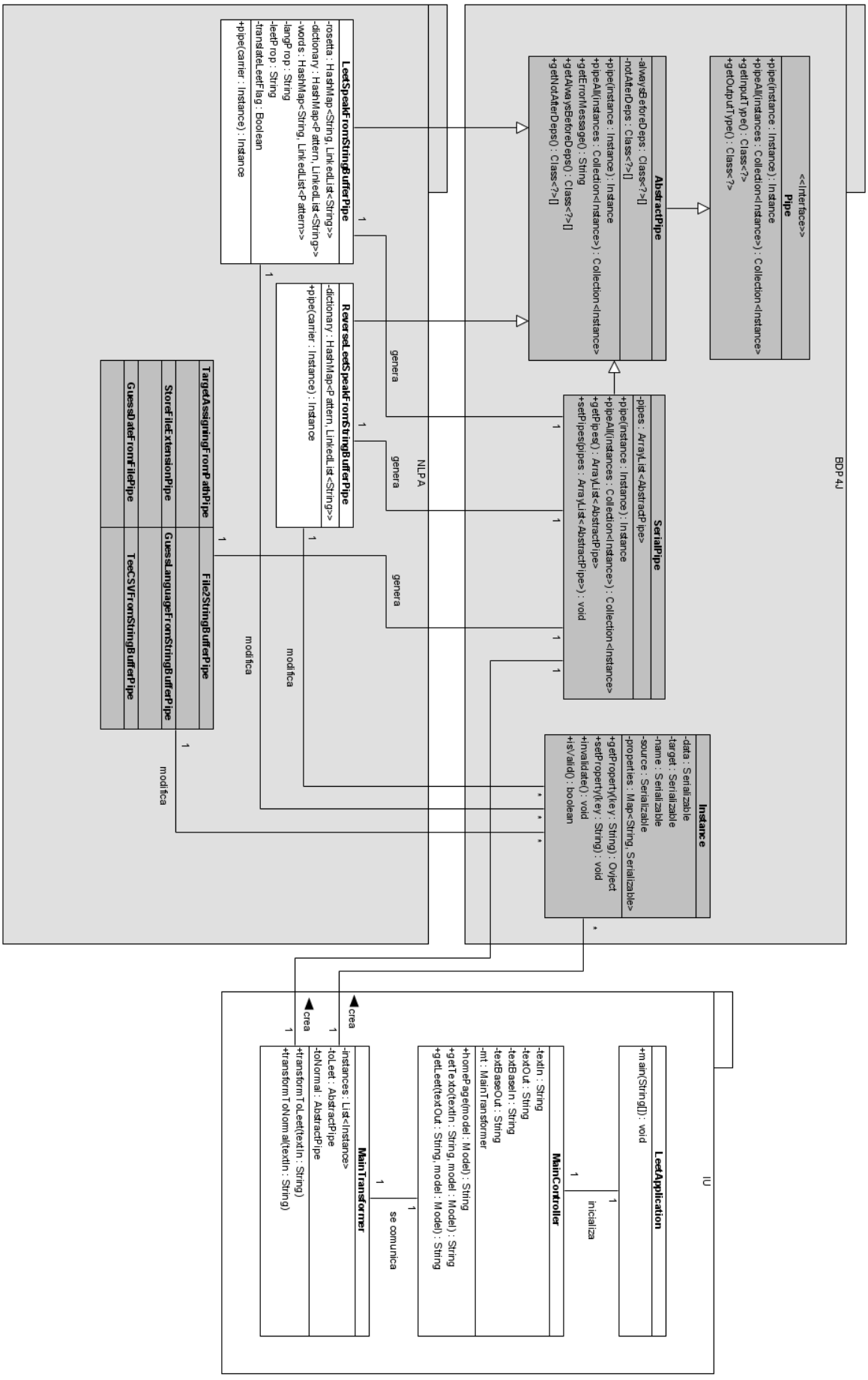

Figura 8. Diagrama de clases.

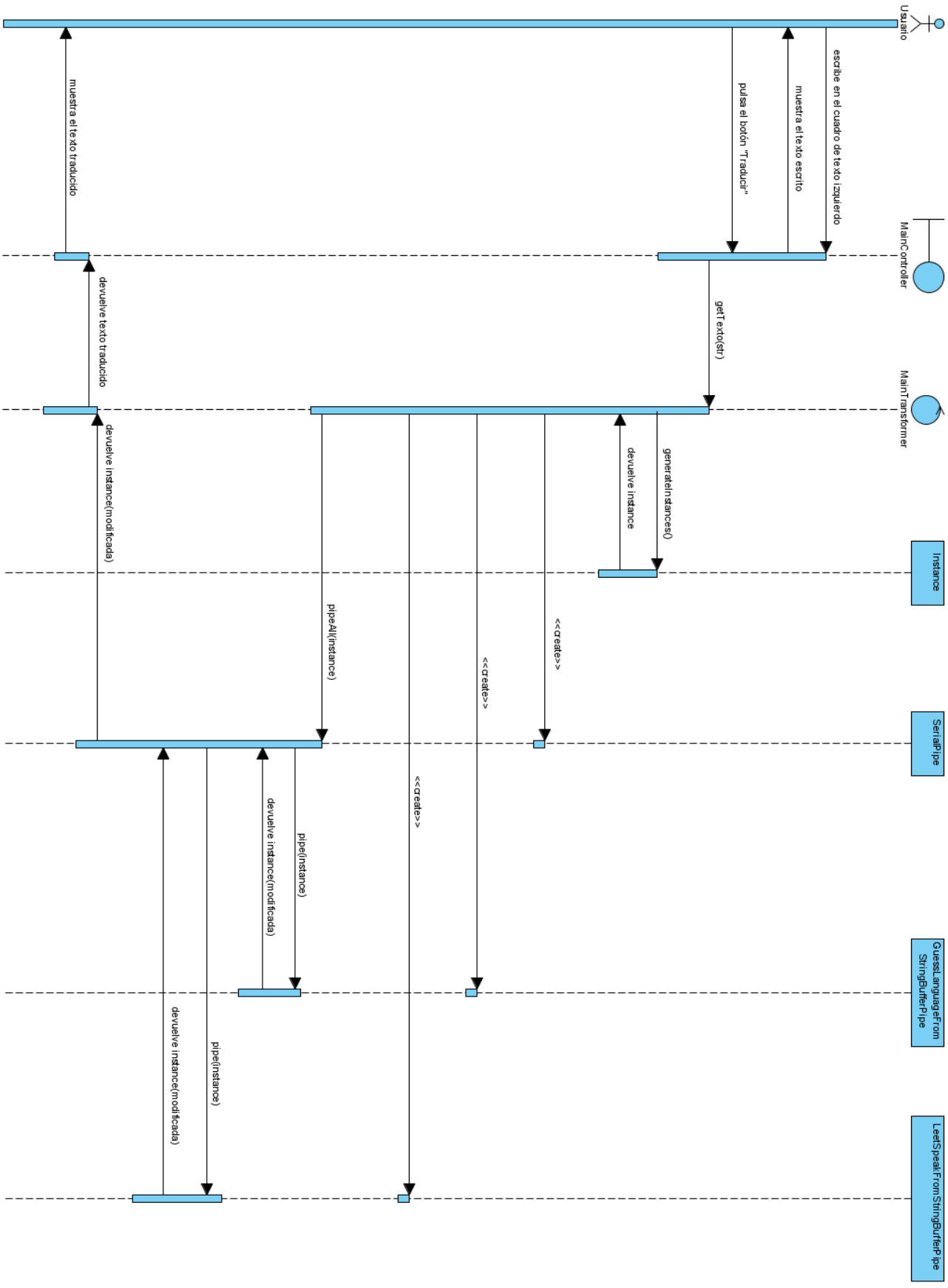

Figura 9.Diagrama de secuencia.

6. Gestión de datos e información

En esta sección se especifican las técnicas empleadas para gestionar los datos e información relevante de la aplicación.

Como punto de partida, la aplicación carga los textos mediante el objeto Instance, que atravesará el pipeline. El objeto instancia almacena el texto original (source) y el texto que ha de ser procesado y puede ser modificado (data). Cuando una instancia pasa por un pipe, este puede alterar los datos almacenados en data, tras esto la instancia entra al siguiente pipe.

Algunos pipes también pueden añadir propiedades a la instancia, es el caso de *GuessLanguageFromStringBufferPipe*, que detecta el idioma del texto y lo almacena en su respectiva propiedad.

Nuestro pipe *LeetSpeakFromStringBufferPipe* también añade una propiedad al texto "NER_leetSpeak", en la cual se almacenan todas las palabras Leet detectadas en el texto.

Estas propiedades son fácilmente visualizables mediante el pipe de NLPA *TeeCSVFromStringBufferPipe*, que se encarga de generar un archivo en formato csv que contendrá los datos de la instancia junto con sus propiedades previamente calculadas en el pipeline.

Para la identificación de las entidades Leet, primero debemos identificar los caracteres Leet que puedan equivaler a una letra del abecedario, tal como introdujimos en el apartado *1.2 Resumen de la solución propuesta*. Para esta finalidad nos ayudamos de diccionarios. Esta aplicación maneja fundamentalmente tres diccionarios:

El primero de ellos, usado en el caso del pipe Leet, es en realidad un conjunto de diccionarios de idiomas en el cual se almacenan la mayoría de las palabras de cada idioma en archivos .txt tal como se muestra en la Figura 10. Estos diccionarios ya formaban parte de los recursos de NLPA puesto que eran utilizados en el pipe encargado de detectar el idioma del texto. En nuestro pipe solo nos interesan las palabras, por lo que importamos únicamente la primera palabra de cada linea.

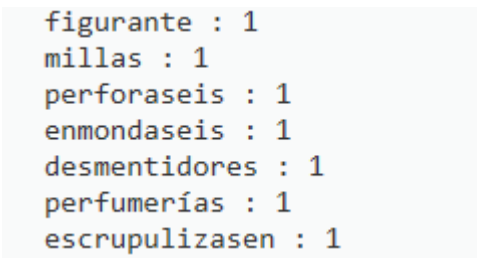

Figura 10.Diccionarios de palabras

Los otros dos diccionarios, usados en los pipes Leet y ReverseLeet respectivamente, reciben el nombre de leet.json y reverseLeet.json, y almacenan equivalencias entre letras alfabéticas y símbolos que equivalgan a estas, como podemos ver en las Figuras 11 y 12. El archivo reverseLeet.json es el inverso de leet.json con la salvedad de que se han eliminado algunas equivalencias leet (muy dependientes de la tipografía) para facilitar la lectura del texto resultante tras el pipe ReverseLeet. El archivo leet.json separa los caracteres alfabéticos mediante guiones mientras que el archivo reverseLeet.json separa los equivalentes Leet mediante espacios, debido a que en el segundo caso el espacio es el único carácter que jamás se usará en un equivalente leet.

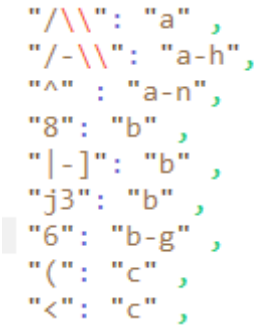

Figura 12.Diccionario leet.

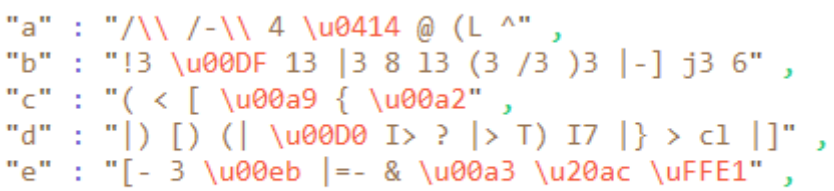

Figura 11.Diccionario reverseLeet.

Cabe destacar el uso del formato JSON elegido para construir los diccionarios. Como se mencionó anteriormente JSON es un formato de texto basado en pares clave-valor, sencillo de escribir y leer por los humanos y también fácilmente interpretable por las máquinas. Es ligero e independiente del lenguaje, con lo cual los diccionarios podrán ser utilizados para futuros desarrollos con facilidad.[26]

La información de las equivalencias de símbolos leet se ha recopilado de diferentes fuentes en la red especializadas en LeetSpeak.[27][28]

7. Pruebas llevadas a cabo

En este apartado, se detallan las pruebas realizadas para comprobar el funcionamiento del software desarrollado. Las comprobaciones se han realizado principalmente mediante pruebas de caja negra, con esta técnica se trata de verificar la funcionalidad sin tener en cuenta la lógica interna del software, únicamente prestando atención a la entrada y la salida del programa[29]. Asimismo, se han realizado pruebas sobre la interfaz gráfica de forma manual.

Las pruebas de los pipes se basan en las condiciones de aceptación de las historias de usuario especificadas anteriormente.

En la Tabla 4 se muestran las pruebas realizadas, los resultados esperados y los obtenidos para el pipe *LeetSpeakFromStringBufferPipe*.

Tabla 3. Pruebas de LeetSpeakFromStringBufferPipe.

PRUEBA	RESULTADO ESPERADO	RESULTADO OBTENIDO
Identificar/ traducir entidades Leet en un texto sin entidades leet	El texto obtenido es idéntico al texto de entrada, la propiedad NER_leetSpeak de salida está vacía.	El texto obtenido es idéntico al texto de entrada, la propiedad NER_leetSpeak de salida está vacía.
Identificar/ traducir entidades Leet en un texto con entidades leet	El texto obtenido es el resultante de realizar la sustitución de las entidades Leet por sus equivalentes en caso de que se especifique la traducción y en caso contrario es idéntico al texto de entrada. La propiedad NER_leetSpeak de salida contiene todas las entidades Leet que contiene el texto de entrada.	El texto obtenido es el resultante de realizar la sustitución de las entidades Leet por sus equivalentes en caso de que se especifique la traducción y en caso contrario es idéntico al texto de entrada. La propiedad NER_leetSpeak de salida contiene todas las entidades Leet que contiene el texto de entrada.
Identificar/ traducir entidades Leet en un texto que no se haya determinado su idioma	El texto no ha sido modificado y la propiedad NER_leetSpeak de salida está vacía.	El texto no ha sido modificado y la propiedad NER_leetSpeak de salida está vacía.
Identificar/ traducir entidades Leet en un texto que su idioma no tenga diccionario	El texto obtenido es el resultante de realizar la sustitución de las entidades Leet provenientes de palabras inglesas por sus equivalentes en caso de que se especifique la traducción y en caso contrario es idéntico al texto de entrada. La propiedad NER_leetSpeak de salida contiene solo las entidades Leet provenientes de palabras inglesas del texto de entrada.	El texto obtenido es el resultante de realizar la sustitución de las entidades Leet provenientes de palabras inglesas por sus equivalentes en caso de que se especifique la traducción y en caso contrario es idéntico al texto de entrada. La propiedad NER_leetSpeak de salida contiene solo las entidades Leet provenientes de palabras inglesas del texto de entrada.

También se han realizado, análogamente, pruebas sobre el pipe *ReverseLeetFromStringBufferPipe* (especificado en el Anexo I), cuyos resultados podemos ver en la Tabla 5.

Tabla 4.Pruebas de ReverseLeetFromStringBufferPipe.

PRUEBA	RESULTADO ESPERADO	RESULTADO OBTENIDO
Traducir un texto a uno equivalente con entidades Leet	La mitad de los caracteres del texto original son sustituidos por equivalentes Leet.	Aproximadamente la mitad de los caracteres del texto original han sido sustituidos por equivalentes Leet.
Traducir, de nuevo, un texto a uno equivalente con entidades Leet	Cada vez que se ejecuta el pipe sobre el mismo texto de entrada arroja un texto de salida diferente (con equivalentes Leet distintos).	El texto obtenido cambia cada vez que se ejecuta el pipe. Todos los textos de salida son correctos conforme a la prueba anterior.

Finalmente, como indicábamos, se han realizado también pruebas sobre la interfaz gráfica.

Tabla 5. Pruebas de la interfaz gráfica.

PRUEBA	RESULTADO ESPERADO	RESULTADO OBTENIDO
Pulsar botones descifrar/cifrar sin texto	Se actualiza la web sin cambios, devolviendo el texto vacío en ambos cuadros de texto.	Se actualiza la web sin cambios, devolviendo el texto vacío en ambos cuadros de texto.
Pulsar cifrar con texto	Se actualiza la web devolviendo en el cuadro de texto derecho el texto con las entidades Leet sustituidas por sus equivalentes, resultante de procesar el texto de entrada que se toma del cuadro de texto izquierdo.	Se actualiza la web devolviendo en el cuadro de texto derecho el texto con las entidades Leet sustituidas por sus equivalentes, resultante de procesar el texto de entrada que se toma del cuadro de texto izquierdo.
Pulsar descifrar con texto	Se actualiza la web devolviendo en el cuadro de texto izquierdo el texto con equivalencias Leet resultante de procesar el texto de entrada que se toma del cuadro de texto derecho.	Se actualiza la web devolviendo en el cuadro de texto izquierdo el texto con equivalencias Leet resultante de procesar el texto de entrada que se toma del cuadro de texto derecho.
Pulsar actualizar página f5	Se actualiza la web sin cambios, devolviendo el texto vacío en ambos cuadros de texto.	Se actualiza la web sin cambios, devolviendo el texto vacío en ambos cuadros de texto.
Pulsar enlaces del footer	El navegador es redirigido a las webs correspondientes (linkedin y github).	El navegador es redirigido a las webs correspondientes (linkedin y github).

8. Manual de usuario

En este apartado se muestran los manuales de usuario, tanto de instalación como de uso, así como los requisitos mínimos para ejecutar el software.

8.1 Requisitos mínimos

Podemos dividir los requisitos mínimos en función de nuestro rol (especificados en la sección *4.Especificación y análisis de requisitos*):

Si el rol es el de usuario común, el único requisito para probar la aplicación es el uso de un navegador y la disponibilidad de conexión a internet. La aplicación se puede usar en el dominio www.leettraductor.com.

Si el rol es el de investigador, los requisitos mínimos son:

Java JDK versión 8 o superior.[30]

Apache Maven versión 3.8.0 o superior.[31]

Espacio de almacenamiento de al menos 2,5Gb en disco.

8.2 Manual de instalación

El software está alojado en Github de forma pública, con lo que para proceder a su instalación solo es necesario clonarlo en el computador mediante un comando git:

$: git clone https://github.com/sing-group/nlpa

Este comando descargará automáticamente todos los pipes necesarios para la ejecución de la aplicación. Ahora, sobre el repositorio descargado ejecutamos una instalación limpia con Maven:

$: mvn clean install

Con ello tendremos preparado todo el entorno para ejecutar la aplicación según a nuestras necesidades, en concreto, puede editarse el archivo Main.java que cuelga del directorio src/main/java/org/nlpa para añadir las fuentes de texto y configurarlo según los procesos que se quieran hacer sobre estas instancias.

8.3 Manual de uso

De nuevo, depende del rol que asumamos a la hora de utilizar la aplicación:

Con el rol de usuario común, el usuario hará uso del navegador para ejecutar la aplicación en el dominio www.leettraductor.com. La Figura 13 muestra el entorno gráfico indicando los puntos de interacción con el mismo.

Para traducir texto con entidades leet a texto normal:

El usuario hace click en el cuadro de texto (1) y escribe el texto que desea traducir, tras esto pulsa el botón (3), lo cual hará que en el cuadro de texto (2) aparezca el texto traducido.

Para “cifrar” texto normal sustituyendo palabras por equivalentes leet:

El usuario hace click en el cuadro de texto (2) y escribe el texto que desea traducir, tras esto pulsa el botón (4), lo cual hará que en el cuadro de texto (1) aparezca el texto traducido. Si se desea una traducción equivalente pero distinta se vuelve a pulsar el botón (4).

Otras funcionalidades:

En caso de algún tipo de error no previsto se recomienda que el usuario actualice la web pulsando en (7) o mediante F5.
Pulsando en el link (5) el usuario puede acceder al perfil Linkedin del alumno.
Pulsando en el link (6) el usuario puede acceder al repositorio en Github del equipo de investigación SING.

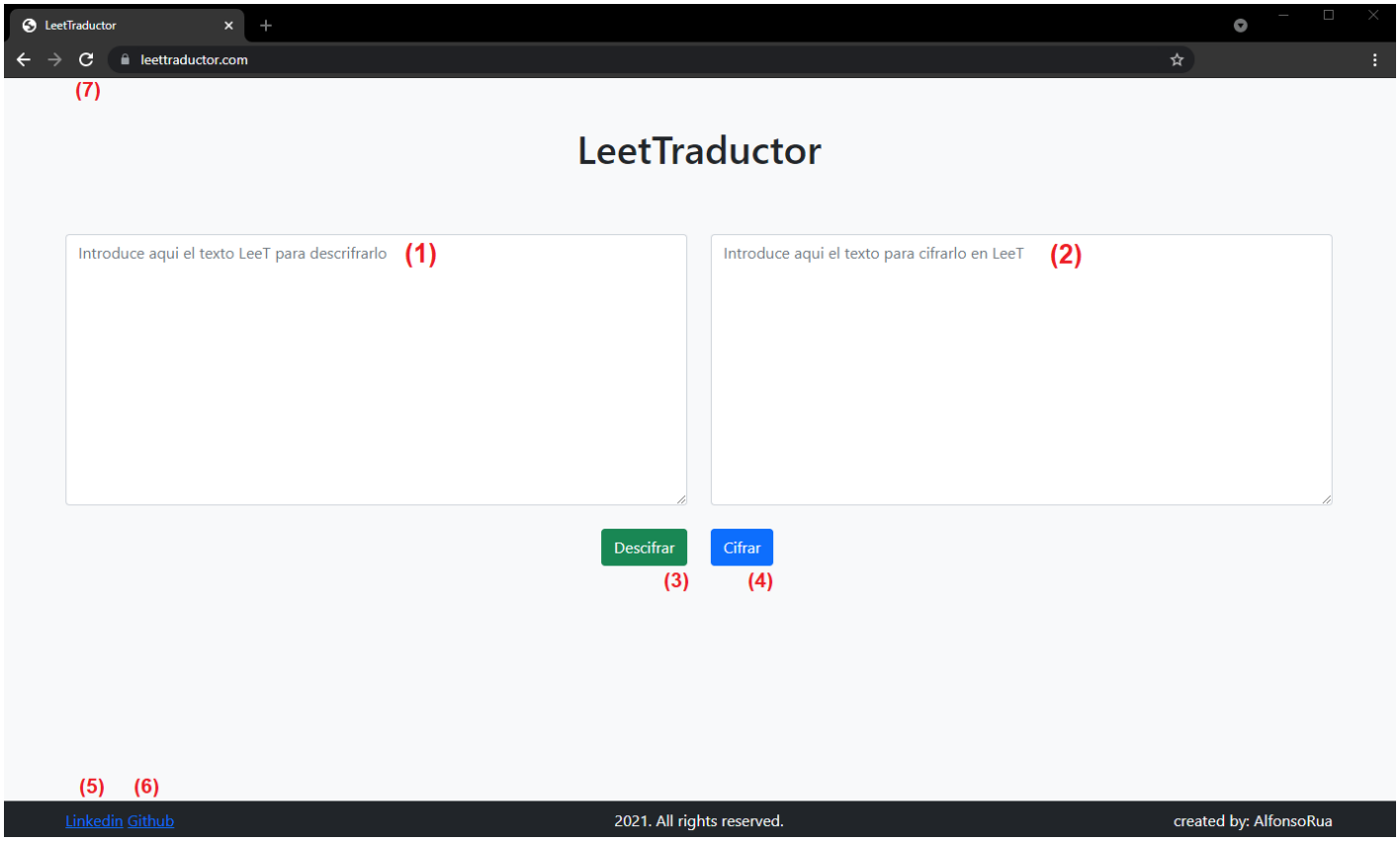

Figura 13. Interfaz de usuario.

Por otra parte, el usuario con el rol de investigador, tras instalar el repositorio tal como indicamos en el apartado anterior, puede editar el archivo Main.java de la siguiente forma:

Se pueden añadir textos de entrada a una colección mediante el constructor Instance(Serializable data, Serializable target, Serializable name, Serializable source) o mediante el método generateInstances(String path).

Para configurar el pipeline debemos crearlo mediante:
AbstractPipe p = new SerialPipes(new AbstractPipe[]{ })
y dentro de los corchetes especificaremos los pipes que nos interesan, en concreto para nuestra funcionalidad necesitaremos los siguientes:

Previos:
new TargetAssigningFromPathPipe(), new StoreFileExtensionPipe(),
new GuessDateFromFilePipe(), new File2StringBufferPipe(),
*new GuessLanguageFromStringBufferPipe(),

Funcionalidad concreta a elegir entre:
**new LeetSpeakFromStringBufferPipe(true),
new ReverseLeetSpeakFromStringBufferPipe(),

Posterior:
new TeeCSVFromStringBufferPipe("nombreSalida.csv", true)

Para ejecutar el pipeline debemos llamar al método p.pipeAll(Collection<Instance> instances).

Ejecutando el método main de esta clase (tras compilar o mediante algún idle como VSCode) obtendremos en el directorio /output el archivo nombreSalida.csv con los datos procesados.

*No necesario para el uso de *ReverseLeetSpeakFromStringBufferPipe*.

** El parámetro booleano que se pasa como entrada indica si se desea traducir o únicamente identificar entidades leet.

9. Conclusiones y trabajo futuro

En este apartado se presentan las conclusiones relativas al trabajo realizado, en concreto, las principales aportaciones que este TFG supone, así como las conclusiones y posibles vías de trabajo futuro.

9.1 Principales aportaciones

Considerando los objetivos expuestos en la primera sección de este documento, y el desarrollo realizado en este proyecto podemos concluir que se han realizado las siguientes aportaciones:

· Creación de un pipe (*LeetSpeakFromStringBufferPipe*) que permite identificar las entidades leet en un texto.

· Creación de una funcionaliad (dentro del pipe anterior) que permite sustituir las entidades leet de un texto por su equivalente.

· Creación de un diccionario de equivalencias leet, los cuales son la base para la identificación de entidades leet por cualquier método y quedarán disponibles para la comunidad.

· Creación de una interfaz web, que facilita el uso de las funcionalidades desarrolladas para usuarios sin conocimientos de programación y sin la necesidad de descargar ni instalar nada. Todo ello disponible mediante el dominio www.leettraductor.com.

9.2 Conclusiones

La principal conclusión de este proyecto es que lo que a priori parece sencillo, a menudo no lo es.

El principal obstáculo para discernir si una palabra es leet o no por medio de reglas, radica en que no todas las palabras con símbolos o números son leet (ej: cifras, dni…). Lo que nos lleva al uso inevitable de diccionarios.

A la hora de calcular las posibles palabras que forma una entidad leet nos encontramos con la mayor fuente de ineficiencia de nuestro proyecto: la explosión combinatoria[32]. Esto es, debido a que cada símbolo de una palabra puede equivaler a varios caracteres, y cada conjunto de símbolos a su vez también pueden equivaler a otros caracteres diferentes; las posibles palabras generadas por una entidad leet con muchos símbolos equivalentes crece exponencialmente, por ejemplo: 35745? (estas?) genera más de 10.000 posibilidades que han de ser comprobadas contra el diccionario para saber si alguna de ellas es una palabra real.

Asimismo, la permutación de las equivalencias leet para llegar a todas estas palabras posibles tampoco es ni mucho menos trivial; puesto que han de obtenerse todas las

permutaciones posibles, sin repetirse y sin destruir las combinaciones que puedan dar lugar a otras combinaciones también válidas. No obstante, no se pueden incluir combinaciones producto de tomar los símbolos sustituidos para formar nuevas equivalencias.

Pese a todo, se ha logrado un software bastante eficiente y completo, el cual funciona prácticamente de forma instantánea en condiciones típicas como los correos spam.

En lo personal, el desarrollo de este proyecto me ha dado la oportunidad de utilizar herramientas como BDP4J o la arquitectura en pipeline que eran desconocidas para mí. Por otra parte, he podido sacar fruto a la formación recibida durante los años de carrera, así como en las prácticas en empresa, lo cual me motiva a seguir adquiriendo conocimiento y desarrollándome como ingeniero de software. Finalmente, ha sido gratificante trabajar para un proyecto del grupo de investigación SING, puesto que quien antaño había sido mi profesor, ahora fue más bien un compañero de desarrollo.

Con todo ello, es innegable que la realización de este TFG ha sido desafiante y satisfactorio a partes iguales.

9.3 Vías de trabajo futuro

Existen una serie de mejoras y ampliaciones en las funcionalidades de este proyecto que podría interesar implementar en un futuro:

Selección de mejor opción de traducción en base a palabras adyacentes

En los casos en los que una entidad leet pueda traducirse como varias palabras diferentes (ej: l3et -> bet o leet), sería idóneo que el programa tuviese algún modo de predecir cual es la palabra que se deseaba ocultar mediante leet. Para ello, una primera aproximación podría ser la proximidad de cada una de estas opciones al campo semántico de las palabras adyacentes a la entidad leet[33]. Por ejemplo, si la frase es “you can l3et for free in our casino” resulta fácil predecir que la palabra que se buscaba ocultar era “bet” en lugar de “leet”.

Ampliación de diccionarios

Una de las ampliaciones más sencillas a la par que útiles consistiría en la creación de nuevos diccionarios para extender la funcionalidad a nuevos idiomas. Asimismo, también resultaría conveniente ampliar los diccionarios actuales con una mayor cantidad de palabras y extranjerismos.

Creación automática de diccionarios con palabras leet

Otra opción, para reducir la carga de trabajo del pipe, consistiría en anotar de forma automática en un diccionario cada nueva entidad leet que se encuentre junto con su equivalencia. De esta forma, el pipe primero comprobaría si cada palabra del texto de entrada está incluida en este nuevo diccionario, de ser así su identificación y traducción sería instantánea, y en caso de no estar incluida se seguiría con el resto de los cálculos del pipe que fuesen necesarios.

10. Referencias

[1] C. W. Tsai, C. F. Lai, H. C. Chao, and A. V. Vasilakos, “Big data analytics: a survey,” *J. Big Data*, 2015, doi: 10.1186/s40537-015-0030-3.

[2] R. A. Cortez Reyes, “Extracción de conocimiento a partir de textos obtenidos de Twitter,” *Entorno*, 2018, doi: 10.5377/entorno.v0i65.6048.

[3] M. Zhou, N. Duan, S. Liu, and H. Y. Shum, “Progress in Neural NLP: Modeling, Learning, and Reasoning,” *Engineering*. 2020, doi: 10.1016/j.eng.2019.12.014.

[4] M. Novo-Lourés, R. Pavón, R. Laza, D. Ruano-Ordas, and J. R. Méndez, “Using natural language preprocessing architecture (NLPA) for big data text sources,” *Sci. Program.*, 2020, doi: 10.1155/2020/2390941.

[5] H. Shelar, G. Kaur, N. Heda, and P. Agrawal, “Named Entity Recognition Approaches and Their Comparison for Custom NER Model,” *Sci. Technol. Libr.*, 2020, doi: 10.1080/0194262X.2020.1759479.

[6] N. Kannaiya Raja, N. Bakala, and S. Suresh, “NLP: Rule based name entity recognition,” *Int. J. Innov. Technol. Explor. Eng.*, 2019, doi: 10.35940/ijitee.K2047.0981119.

[7] “Leet speak.” [Online]. Available: https://es.wikipedia.org/wiki/Leet_speak.

[8] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, “Machine learning for email spam filtering: review, approaches and open research problems,” *Heliyon*, 2019, doi: 10.1016/j.heliyon.2019.e01802.

[9] G. M. Barrientos, R. Alaiz-Rodríguez, V. González-Castro, and A. C. Parnell, “Machine learning techniques for the detection of inappropriate erotic content in text,” *Int. J. Comput. Intell. Syst.*, 2020, doi: 10.2991/ijcis.d.200519.003.

[10] Scrum.org, “¿Qué es Scrum?,” *Scrum.org*, 2020.

[11] J. Dubois, “Aplicación práctica del diagrama de Gantt en la administración de un proyecto,” *Fac. Ciencias Económicas Univ. Nac. Tucumán*, 2012.

[12] R. Popli and N. Chauhan, “A sprint-point based estimation technique in scrum,” in *Proceedings of the 2013 International Conference on Information Systems and Computer Networks, ISCON 2013*, 2013, doi: 10.1109/ICISCON.2013.6524182.

[13] R. D. Chamberlain *et al.*, “Auto-pipe: Streaming applications on architecturally diverse systems,” *Computer (Long. Beach. Calif).*, 2010, doi: 10.1109/MC.2010.62.

[14] M. S. Singh, “MVC Framework: A Modern Web Application Development Approach and Working,” *Int. Res. J. Eng. Technol.*, 2020.

[15] ORACLE, “¿Qué es Java?,” *Oracle Corporation*. 2016.

[16] “Javax.json.” [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/json/package-summary.html.

[17] A. L. Davis, “Spring Boot,” in *Spring Quick Reference Guide*, 2020.

[18] Tutorials Point, “Bootstrap responsive web development,” *Tutorials Point*, 2017.

[19] J. D. Blischak, E. R. Davenport, and G. Wilson, “A Quick Introduction to Version Control with Git and GitHub,” *PLoS Comput. Biol.*, 2016, doi: 10.1371/journal.pcbi.1004668.

[20] The Apache Software Foundation, “Maven – Welcome to Apache Maven,” *maven.apache.org*. 2016.

[21] Microsoft, “Documentation for Visual Studio Code,” *Visual Studio Code*, 2020. .

[22] V. Paradigm and C. Edition, “Visual Paradigm,” *Vis. Paradig.*, 2004.

[23] “Team Gantt.” [Online]. Available: https://www.teamgantt.com/.

[24] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, “Improving agile requirements: the Quality User Story framework and tool,” *Requir. Eng.*, 2016, doi: 10.1007/s00766-016-0250-x.

[25] M. Otto, “Bootstrap · The most popular HTML, CSS, and JS framework in the world.,” *Bootstrap Website*, 2020. .

[26] P. Bourhis, J. L. Reutter, and D. Vrgoč, “JSON: Data model and query languages,” *Inf. Syst.*, 2020, doi: 10.1016/j.is.2019.101478.

[27] “Leet speak cheat sheet.” [Online]. Available: https://www.gamehouse.com/blog/leet-speak-cheat-sheet/.

[28] “A guide to leetspeak.” [Online]. Available: https://www.ionos.com/digitalguide/online-marketing/social-media/what-is-leetspeak/.

[29] H. Bhasin, E. Khanna, and S. Sudha, “Black Box Testing based on Requirement Analysis and Design Specifications,” *Int. J. Comput. Appl.*, 2014, doi: 10.5120/15311-4024.

[30] “Java JDK.” [Online]. Available: https://www.oracle.com/es/java/technologies/javase/javase-jdk8-downloads.html.

[31] “Maven.” [Online]. Available: https://maven.apache.org/.

[32] M. M. Chiu and L. Khoo, “A new method for analyzing sequential processes: Dynamic multilevel analysis,” *Small Gr. Res.*, 2005, doi: 10.1177/1046496405279309.

[33] S. Trott, T. T. Torrent, N. Chang, and N. Schneider, “(Re)construing Meaning in NLP,” 2020, doi: 10.18653/v1/2020.acl-main.462.

Anexo I

En este anexo se mostrará la lógica del pipe *ReverseLeetSpeakFromStringBufferPipe*, el cual toma un texto cualquiera y lo transforma sustituyendo las palabras por entidades leet equivalentes.

La motivación principal de este pipe es la de disponer de un generador de textos con entidades Leet para su uso en la aplicación web por parte del usuario común.

Su funcionamiento es similar a los demás pipes del proyecto, pero su complejidad es mucho más simple. Podemos ver la lógica analizando su función pipe() en la Figura13:

```
public Instance pipe(Instance carrier) {
        StringBuffer sb = (StringBuffer) carrier.getData();
        Pattern pattern = Pattern.compile("[a-zA-Z]");
        Matcher matcher = pattern.matcher(sb);
        String leet;
        String noleet;
        int last = 0;
        while(matcher.find(last)){
            Random rnd = new Random();
            int rd= rnd.nextInt();
            if(rd> 0){ //0 for 50% // -1*Integer.MAX_VALUE/2 for 75%
                noleet=matcher.group().toLowerCase();
                leet= dictionary.get(noleet).get(rnd.nextInt(dictionary.get(noleet).size()));
                sb.replace(matcher.start(0),last=matcher.end(0),leet);
            }else{
                last=matcher.end(0);
            }
        }
        return carrier;
    }
```

Figura 14. Lógica de ReverseLeetSpeakFromStringBufferPipe

Para cada carácter alfabético del texto de entrada se decide si se va a modificar o no (lo cual se hace mediante la clase Random) en una proporción del 50%. En caso de que el resultado sea positivo, se busca ese carácter alfabético en el diccionario proveniente de reverseLeet.json, y se selecciona al azar una de sus equivalencias Leet para sustituirla. El otro 50% de las veces se deja el carácter original.

De esta forma se consiguen resultados distintos (pero equivalentes) con cada ejecución del pipe sobre el mismo texto de entrada.