

# **NERd: anotador eficiente de modelos estadísticos para el reconocimiento de entidades nombradas**

Instituto Tecnológico de Buenos Aires

**Juan Pablo Orsay** (L: 49373)

**Horacio Miguel Gómez** (L: 50825)

**Lic. Diego Ariel Aizemberg** (Tutor)



Proyecto final para la carrera de  
Ingeniería Informática

Buenos Aires, Argentina

2019-11-23

# Índice

<b>1</b>	<b>Introducción</b>	<b>5</b>
<b>2</b>	<b>Definición del problema</b>	<b>6</b>
2.1	Conceptos básicos . . . . .	6
2.2	Los datos son el problema . . . . .	9
<b>3</b>	<b>Estado del arte</b>	<b>10</b>
3.1	Stack de software . . . . .	10
3.2	El pipeline . . . . .	10
3.3	Algoritmo de tokenización . . . . .	10
3.4	Modelos basados en reglas . . . . .	11
3.5	Modelos de « <i>deep-learning</i> » . . . . .	12
3.6	Uso del modelo estadístico . . . . .	18
<b>4</b>	<b>NERd (Implementación)</b>	<b>20</b>
4.1	Vista lógica . . . . .	20
4.2	Vista de proceso . . . . .	38
4.3	Vista de desarrollo . . . . .	43
4.4	Vista física . . . . .	55
4.5	Escenarios . . . . .	58
<b>5</b>	<b>Resultados de entrenamiento</b>	<b>62</b>
5.1	Métrica: Valor-F . . . . .	62
5.2	Resultados obtenidos . . . . .	63
<b>6</b>	<b>Discusión</b>	<b>65</b>
6.1	Análisis de resultados . . . . .	65
6.2	Tipos de entidades . . . . .	65
6.3	Semilla inicial . . . . .	67
6.4	Entrenamiento <i>live vs offline</i> . . . . .	67
6.5	<i>Linkeo</i> de entidades con <i>Knowledge Base</i> . . . . .	68
6.6	Utilidad de la herramienta . . . . .	70
<b>7</b>	<b>Conclusiones</b>	<b>72</b>
	<b>Anexo: Datos de resultados</b>	<b>73</b>
7.1	t: 0% - v: 100%. <i>Baseline spaCy</i> . . . . .	73
7.2	t: 50% - v: 50% . . . . .	74
7.3	t: 75% - v: 25% . . . . .	76
7.4	t: 80% - v: 20% . . . . .	77

7.5	t: 85% - v: 15%	79
7.6	t: 90% - v: 10%	81
<b>Glosario</b>		<b>83</b>
<b>Referencias</b>		<b>83</b>

## Índice de figuras

1	Principales avances del estado del arte para NER en los últimos años.	8
2	Pipeline standard para los algoritmos de NER.	10
3	Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.	11
4	Estados posibles para las diferentes etapas de la CNN.	13
5	Embed.	13
6	Características lingüísticas.	14
7	Parallelogram structure in the vector space (by definition).	15
8	Encode.	16
9	Attend.	17
10	Predict.	17
11	Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.	18
12	Secuencia de transiciones para el ejemplo "Mark Watney visited Mars. <sup>en</sup> el modelo de Stack-LSTM.	18
13	Ilustración de arquitectura 4+1.	20
14	Pantalla de inicio con usuario logueado.	21
15	Pantalla de inicio sin rol de entrenador.	22
16	Pantalla de inicio sin sesión.	22
17	Pantalla de entrenamiento.	23
18	Ayuda del entrenador.	24
19	Edición de entidad.	25
20	Administración de usuarios.	26
21	Perfil de usuario.	26
22	Administración de corpus.	27
23	Información de corpus y manejo de workers.	28
24	Inferencia de entidades en sandbox.	29
25	Comparativa de modelos.	30
26	Registración.	31
27	Inicio de sesión.	31
28	Usuario anónimo.	32
29	Usuario regular.	32

30	Usuario administrador. . . . .	32
31	Swagger UI. . . . .	33
32	Ejemplo de documentación con Swagger para request. . . . .	33
33	Ejemplo de documentación con Swagger para respuestas. . . . .	34
34	Ejemplo de documentación con ReDoc. . . . .	35
35	Componentes del sistema. . . . .	39
36	Bloqueo de entrenamiento por lectura. . . . .	41
37	Flujo de relogin . . . . .	43
38	Módulos de la página web. . . . .	45
39	Página principal de Flower. . . . .	48
40	Historial de tareas en Flower. . . . .	49
41	Monitoreo en tiempo real en Flower. . . . .	49
42	Listado de colas activas en Flower. . . . .	50
43	Colas de mensaje escuchadas por un worker. . . . .	50
44	Colecciones de la base de datos. . . . .	51
45	Colección de usuarios. . . . .	52
46	Colección de textos. . . . .	53
47	Colección de snapshots. . . . .	54
48	Colección de entrenamientos. . . . .	55
49	Flujo de NER. . . . .	59
50	Pedido de texto para entrenamiento. . . . .	60
51	Cambio de modelo. . . . .	61
52	Boxplot del Valor-F vs cantidad de documentos de entrenamiento . . . . .	64
53	Taxonomía extendida de Sekine. . . . .	66
54	Wikidata - Lionel Messi. . . . .	69
55	Con NEL se sumarían Messi con Lionel y Mauricio con Macri. . . . .	70

# 1. Introducción

El **Reconocimiento de Entidades Nombradas** (*NER*) es una subtarea del área de *Natural Language Processing* (*NLP*) que busca extraer **entidades nombradas** en textos no estructurados. Estas entidades son luego clasificadas en categorías predefinidas como los nombres de personas, organizaciones, ubicaciones, expresiones de tiempo, cantidades, valores monetarios, porcentajes, entre otras.

Por ejemplo en el siguiente texto:

Este es el proyecto final de la carrera Ingeniería Informática de los alumnos Gómez y Orsay para el Instituto Tecnológico de Buenos Aires.

Se pueden detectar 3 entidades:

- **Gómez:** Persona
- **Orsay:** Persona
- **Instituto Tecnológico de Buenos Aires:** Organización

El estado-del-arte de los sistemas *NER* producen un rendimiento casi humano (Marsh & Perzanowski, 1998) (cercaos al 95 % de *valor-F*).

A pesar de estos altos valores de rendimiento, la industria tiene dificultades para poder capitalizar la efectividad de dichas sistemas y algoritmos. Es por ello que en este trabajo final hemos tomado la decisión de implementar una plataforma Open Source para revertir esta situación.

## 2. Definición del problema

### 2.1. Conceptos básicos

#### 2.1.1. Entidad nombrada

En el contexto de *extracción de información*, una *entidad nombrada* es un objeto del mundo real como lo son personas, ubicaciones, organizaciones, productos, etc; que pueden denotarse con un nombre propio. La entidad puede ser abstracto o tener existencia física. Ejemplos de entidades nombradas son «Mauricio Macri», «Ciudad Autónoma de Buenos Aires», «Apple Macbook». También se suele definir sencillamente como aquellas entidades que se pueden ver como instancias de entidad (por ejemplo, la Ciudad Autónoma de Buenos Aires es una instancia de una ciudad).

##### 2.1.1.1. Definición formal

Formalmente el concepto de «entidad nombrada» se deriva de la definición del filósofo estadounidense **Saul Kripke** de **designador rígido** (Kripke, 1980) que forma parte de la lógica modal y filosofía del lenguaje.

Un designador rígido designa a una misma entidad en todos los mundos posibles en los que esa entidad existe, y no designa nada en aquellos mundos en los que no existe.

Algunos ejemplos de designadores rígidos son:

- Nombres propios como «Saul Kripke», «Júpiter», «Londres», «4» y «Hércules».
- Descripciones definidas matemáticas como «la raíz cuadrada de 4» y «8 - 2».
- Nombres de clases naturales como «agua» y «bronce».
- Nombres de sensaciones como «dolor» y «alegría».

Por el contrario, los **designadores flácidos** pueden designar diferentes cosas en diferentes mundos posibles y **no** son *entidades nombradas*.

Por ejemplo en la oración: «Mauricio Macri es el presidente de Argentina»:

- «Mauricio Macri» y «Argentina» son *entidades nombradas*, ya que se refieren a objetos específicos.
- «presidente» y «presidente de Argentina» no son *entidades nombradas*, ya que pueden usarse para referirse a muchos objetos diferentes en mundos diferentes:
  - «presidente» puede ser de diferentes países u organizaciones que se refieren a diferentes personas.
  - «presidente de Argentina», si bien hace referencia a un mismo país, puede ser de diferentes períodos presidenciales que se refieren a diferentes personas.

### 2.1.1.2. Definición no estricta

Existe un acuerdo general en la comunidad *NER* para considerar como entidades nombradas a otro tipos de entidades que violan el principio de designador rígido. Ejemplos de esto son:

- expresiones temporales como «3 de febrero», «2019».
- expresiones numéricas, como cantidades de dinero y otros tipos de unidades
- expresiones que según contexto denotan una entidad rígida pero no en sí mismas. Por ejemplo «Alfredo Fortabat, empresario argentino, fundador de la compañía cementera Loma Negra.» puede ser considerada una entidad nombrada, sin embargo, el término «Fortabat» por sí sólo podría referirse a su viuda «María Amalia Lacroze de Fortabat», al museo de arte «Museo Fortabat» o a la localidad Argentina «Villa Alfredo Fortabat».

En este trabajo hemos priorizado esta definición laxa para tener una mayor expresividad en los tipos de entidades que vamos a detectar. Por lo tanto, en adelante, la definición del término «entidad nombrada» será utilizada bajo una definición no estricta.

### 2.1.2. Reconocimiento de entidades

El Reconocimiento de entidades nombradas a menudo se divide en dos problemas distintos:

1. Detección de nombres
2. Clasificación de los nombres según el tipo de entidad al que hacen referencia a (persona, organización, ubicación y otro)

En la primera fase los nombres se definen como tramos contiguos de tokens, sin anidamiento, de modo que «Instituto Tecnológico de Buenos Aires» es una entidad única, sin tener en cuenta el hecho de que dentro de esta, la subcadena «Buenos Aires» es en sí otra entidad.

Esta forma de definir el problema, lo reduce a un problema de segmentación.

La segunda fase requiere elegir una ontología para organizar categorías de cosas.

#### 2.1.2.1. Dificultades para encontrar mejores algoritmos

El estado del arte de *NER* desde 2014 con la introducción de Redes Neuronales ha llegado a una meseta (Honnibal & Montani, 2017). En los últimos años el diferencial capitalizado por los diferentes grupos de investigación especializados fue muy reducido. Como puede verse en la figura 1.

SYSTEM	TYPE	NER F
spaCy en_core_web_sm (2017)	neural	85.67
spaCy en_core_web_lg (2017)	neural	86.42
Strubell et al. (2017)	neural	<b>86.81</b>
Chiu and Nichols (2016)	neural	86.19
Durrett and Klein (2014)	neural	84.04
Ratinov and Roth (2009)	linear	83.45

**Figura 1:** Principales avances del estado del arte para NER en los últimos años.

Las razones por las cuales esto ocurre escapa el alcance de este trabajo pero se pueden resumir bajo la ley de los rendimientos decrecientes (Case & Fair, 1999). Se necesita mucho esfuerzo académico para obtener una mejora marginal en el estado del arte actual.

Es por esto que resulta interesante analizar que otro tipo de problemas podemos abordar en este trabajo.

#### 2.1.2.2. Dominios del problema

Existe un hilo conductor en el que todas las investigaciones mencionadas en la figura 1 coinciden; incluso los sistemas *NER* más avanzadas son frágiles, dado que los sistemas *NER* desarrollados para un dominio no suelen comportarse bien en otros dominios (Poibeau & Koseim, 2000). La puesta a punto de un sistema *NER* para un nuevo dominio conlleva un esfuerzo considerable. Esto es cierto para modelos basados en reglas y para sistemas estadísticos.

Se entiende por dominio a todos los textos que en su conjunto forman un *corpus* común. Ejemplos de estos son «Noticias periodísticas», «Textos jurídicos», «Reportes militares», «Papers académicos», etc.

## 2.2. Los datos son el problema

Por todo lo mencionado, es evidente que el cuello de botella para el avance de esta y muchas áreas de la «Inteligencia Artificial» es el la captura de datos, no los algoritmos (Honnibal & Montani, 2016).

La **anotación** en el aprendizaje automático es el proceso de etiquetar datos. Con esto, las algoritmos pueden usar los datos anotados para aprender a reconocer patrones similares cuando se presentan datos nuevos. En particular para el estado actual del arte de *NER* es necesario tener un subconjunto del cuerpo de textos a analizar (*corpus*) anotado de tal manera que se conozcan previamente la ubicación de entidades, sus nombres y tipo.

Este aprendizaje se carga en lo que se reconoce como un **modelo estadístico** y es a ese modelo al que se le pide inferir nuevos resultados. En nuestra experiencia los modelos pre-entrenados de las diferentes plataformas, librerías, frameworks y trabajos académicos resultan siempre insuficientes para el uso en producción de los mismos.

De esta manera queda bien definido el problema que queremos atacar (y el título de este trabajo).

NERd: anotador eficiente de modelos estadísticos para el reconocimiento de entidades nombradas.

Para este fin se creó un sistema informático que permitirá obtener resultados a la altura de de las soluciones del estado del arte para cualquier *corpus* de documentos que posea una cantidad suficiente de datos.

El nombre de esta herramienta es **NERd**, sigla cuyo significado en inglés es **Named Entity Recognition Duh**<sup>1</sup>!

---

<sup>1</sup>Expresión de obviedad. *Used to express your belief that what was said was extremely obvious* ("Duh definition," 2019)

### 3. Estado del arte

El análisis del estado del arte fue basado en la definición del problema.

#### 3.1. Stack de software

*Python* es el lenguaje más utilizado para resolver problemas de Machine Learning, en especial *NLP* (Elliott, 2019)

*Spacy* es el *framework* mejor ranqueado para la tarea de *NLP* (Elliott, 2019) y sabemos por la figura 1 que obtiene resultados a-la-par del estado del arte actual.

Además la implementación de *spaCy* es robusta y orientada a la creación de aplicaciones para producción, a diferencia de muchas otras librerías de *NLP* que sólo se utilizan con fines académicos.

#### 3.2. El pipeline

Todas las operaciones de análisis de lenguaje natural sobre textos no estructurados tienen como primer paso el de separar los mismos en tokens. Luego, el documento se procesa en varios pasos diferentes que consisten en el «pipeline de procesamiento». Usualmente los pasos consisten en un etiquetador, un analizador sintáctico y un reconocedor de entidades en el caso de *NER*.

Cada componente del pipeline mostrado en la figura 2 devuelve el un documento *Doc* procesado, que luego se pasa al siguiente componente.

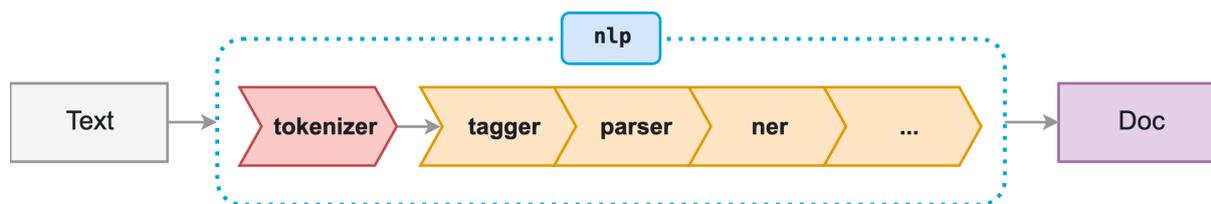


Figura 2: Pipeline standard para los algoritmos de NER.

En este capítulo estudiaremos la morfología de dicho pipeline.

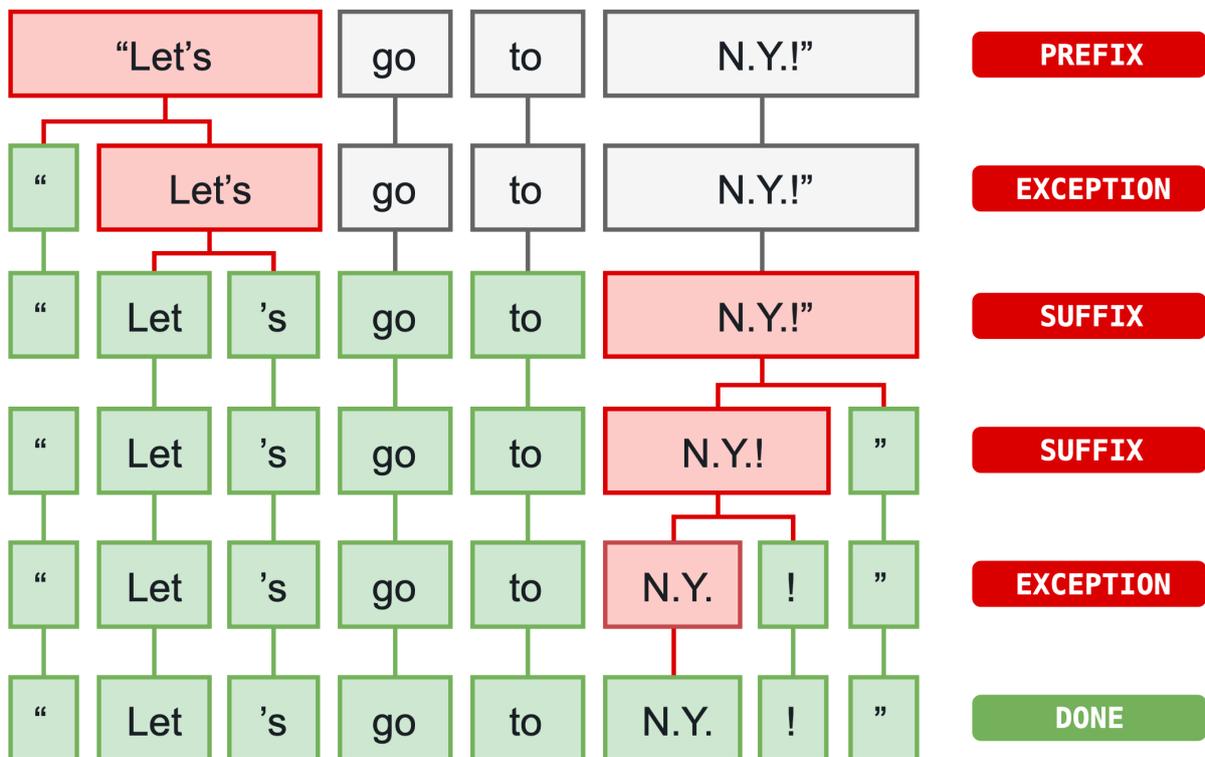
#### 3.3. Algoritmo de tokenización

Para tokenizar un texto de manera correcta no basta con separar el mismo en espacios. Dependiendo del lenguaje que se esté estudiando, existen excepciones a esta regla y otros caracteres que representan separaciones entre tokens según el contexto de los mismos.

En particular, *spaCy* posee un algoritmo de tokenización inteligente que puede ser resumido de la siguiente manera:

1. Iterar sobre subcadenas separadas por espacios en blanco.
2. Comprobar si existe una regla definida explícitamente para esta subcadena. Si existe, usarla.
3. De lo contrario, intentar consumir un prefijo. Si consumimos un prefijo, regrese al punto #2, para que los casos especiales siempre tengan prioridad.
4. Si no se puede consumir un prefijo, intente consumir un sufijo y luego regrese al punto #2.
5. Si no se puede consumir un prefijo ni un sufijo, buscar un caso especial.
6. Buscar una coincidencia de token
7. Buscar «infixos» - cosas como guiones, etc. y dividir la subcadena en tokens en todos los infixos.
8. Una vez que no se pueda consumir más de la cadena, tratarla como un token único.

Ejemplo:



**Figura 3:** Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.

### 3.4. Modelos basados en reglas

Antes de entrar en detalles de cómo trabaja el modelo estadístico de *spaCy* y entender sus fortalezas es importante esbozar brevemente el grupo de algoritmos más «naive» posible. El de los modelos basados en reglas fijas.

En estos modelos se implementan reglas finitas o expresiones regulares para la detección de las

entidades. Las principales limitaciones de este enfoque son:

- **Mucho trabajo manual:** el sistema *Rule Based* exige un profundo conocimiento del dominio, este análisis debe ser realizado por humanos expertos en el dominio.
- **Consumo de tiempo:** la generación de reglas para un sistema complejo es difícil y requiere mucho tiempo.
- **Menor capacidad de aprendizaje:** el sistema generará el resultado según las reglas, por lo que la capacidad de aprendizaje del sistema por sí mismo es baja.
- **Dominios complejos:** si el corpus demasiado complejo, la creación del sistema RB puede llevar mucho tiempo y análisis. La identificación de patrones complejos es una tarea desafiante en el enfoque RB.

### 3.5. Modelos de «*deep-learning*»

Cuando se busca mejorar el aprendizaje automático, generalmente se piensa en la eficiencia y la precisión, pero la dimensión más importante es la generalidad. Este es el modelo estadístico que usa *spaCy*.

La mayoría de los problemas de NLP pueden reducirse a problemas de aprendizaje automático que toman uno o más textos como entrada. Si podemos transformar estos textos en vectores, podemos reutilizar soluciones de aprendizaje profundo (*deep-learning*) de propósito general.

Las representaciones de palabras embebidas (*embed-words*), también conocidas como «vectores de palabras» (*word-vectors*), son una de las tecnologías de procesamiento de lenguaje natural más utilizadas en el estado del arte actual. El modelo de *deep learning* utilizado por *spaCy* puede ser descrito en cuatro pasos.

Los *word-embeddings* permiten tratar a las palabras individuales como «unidades de significado», en lugar de identificaciones completamente distintas. A este proceso se le conoce como **embed**.

Sin embargo, la mayoría de los problemas de NLP requieren la comprensión de tramos de texto más largos, no solo palabras individuales. Al juntar un conjunto de *word-embeddings* en una secuencia de vectores, se usan RNN bidireccionales para codificar los vectores en una matriz de oración. Las filas de esta matriz pueden entenderse como «vectores-de-tokens»: son sensibles al contexto del token dentro de la oración. Este paso se lo llama **encode**.

Finalmente, el mecanismo de **attend** le permite reducir la matriz de oración a un vector de oración, listo para la predicción (**predict**).

De esta manera quedan definidas las piezas que describen al modelo de *spaCy*:

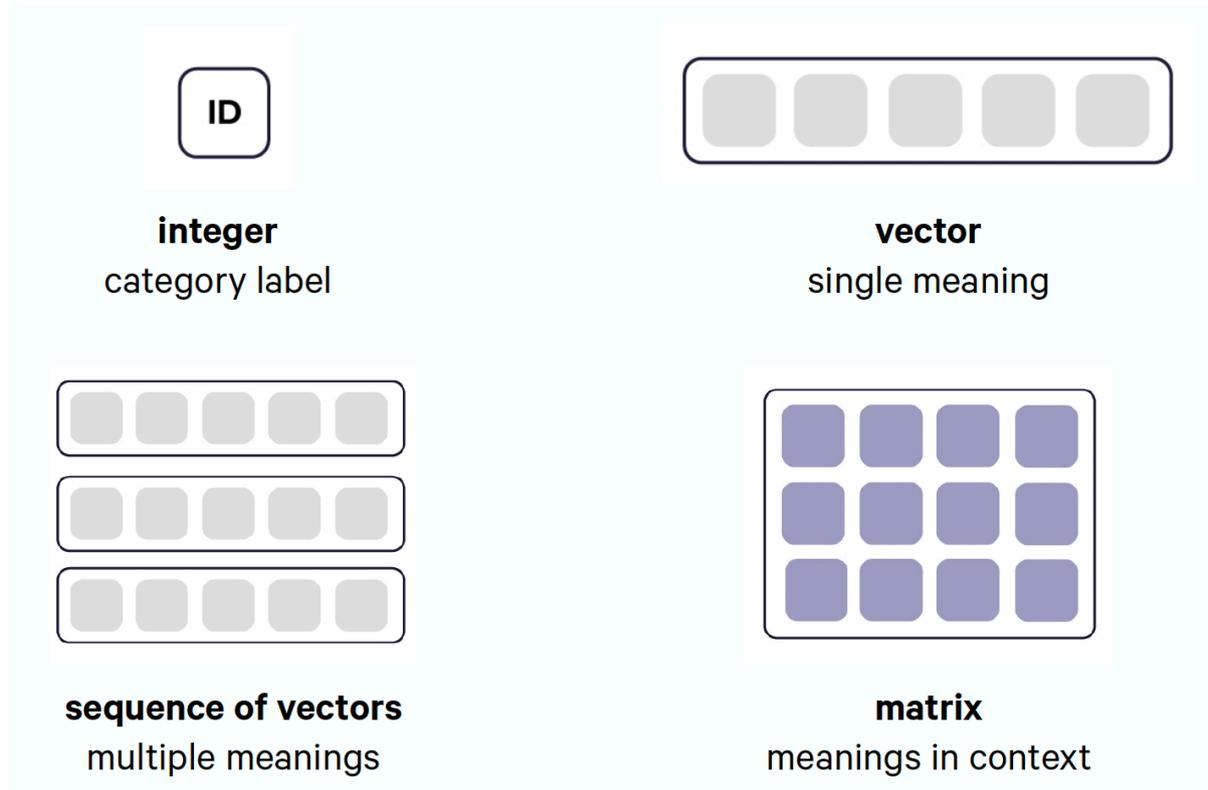


Figura 4: Estados posibles para las diferentes etapas de la CNN.

Estos 4 procesos serán descritos en detalle en las siguientes secciones.

### 3.5.1. *Embed*

Resuelve el problema: «todas las palabras se ven iguales para la computadora»

La idea de *word embeddings* es la de «embeber» el conjunto de tokens que componen términos con información adicional. El resultado de esta operación es una estructura abstracta que puede ser descrita como un «vector de significado».

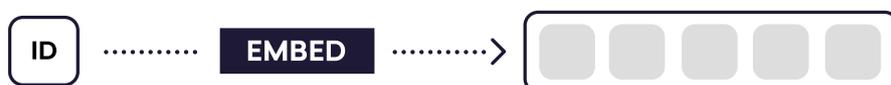


Figura 5: Embed.

Es importante destacar que en la etapa de *embed*, toda la información de significado es principalmente independiente del contexto en el cual está siendo utilizada y por esta razón es fácilmente obtenible del corpus no etiquetado de datos (los algoritmos pueden darse cuenta de que palabras están relacionadas entre sí de manera eficiente y no supervisada).

Esto permite al modelo poder inferir significado a partir de la información no anotada dentro del problema en particular a resolver (el de *NER*).

Una tabla de *word-embeddings* mapea vectores largos binarios y esparsos en vectores cortos densos y continuos, cargados de significado relevante. Existen varias estrategias para enriquecer los *tokens* con información adicional. Las dos fuentes más grandes de información embebida son las de **información lingüística** y los **word-vectors**.

### 3.5.1.1. Información lingüística

El objetivo de ésta etapa es la de encontrar las características intrínsecas de cada palabra. Las principales características lingüísticas detectadas están resumidas en el siguiente ejemplo.

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	VERB	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False
startup	startup	NOUN	NN	dobj	xxxx	True	False
for	for	ADP	IN	prep	xxx	True	True
\$	\$	SYM	\$	quantmod	\$	False	False
1	1	NUM	CD	compound	d	False	False
billion	billion	NUM	CD	pobj	xxxx	True	False

Figura 6: Características lingüísticas.

- **Text:** la palabra original texto.
- **Lemma:** La forma básica de la palabra. Es → ser. Esto permite a las siguientes etapas trabajar con una definición canónica del token.
- **POS:** la etiqueta simple de parte-del-discurso (POS).
- **Tag:** la etiqueta detallada de parte del discurso (POS).
- **Dep:** dependencia sintáctica, es decir, la relación entre tokens.
- **Shape:** la forma de la palabra: mayúsculas, puntuación, dígitos. Muy útil para detectar patrones como números telefónicos, documentos de identidad, CBUs, etc.

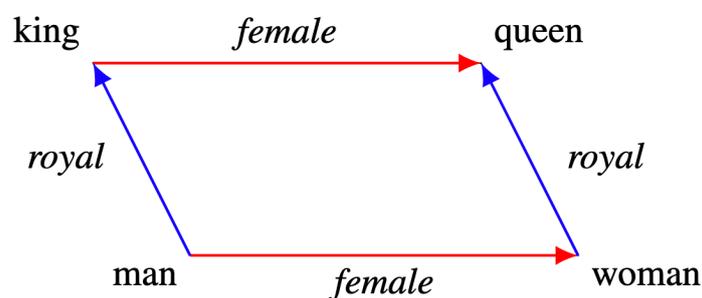
- **is alpha**: ¿el token es una secuencia de caracteres alfabéticos?
- **is stop** : ¿es el token parte de una lista de «palabras de *stop*», es decir, las palabras más comunes del idioma?

### 3.5.1.1.1. *Word Vectors*

A diferencia de la información lingüística que es obtenida en el momento, existen otros tipos de *embeddings* más poderosos que son los productos de pre-entrenamientos; como el caso de los *word-vectors*. Los mismos se generan mediante la concatenación de atributos léxicos como *NORM*, *PREFIX*, *SUFFIX* y *SHAPE*. Luego se usa una capa oculta de una red neuronal convolucional (*CNN*) para permitir una combinación no lineal de la información en estos vectores concatenados.

Los *word-vectors* son particularmente útiles para términos que no están bien representados en los datos de entrenamiento etiquetados. En nuestro uso de reconocimiento de entidades, no siempre habrá suficientes ocurrencias. Por ejemplo, en los datos de entrenamiento es posible que existan ocurrencias del término «Coca-Cola», pero ninguna del término «Manaos». Es interesante pensar a las palabras como «vectores de significado». Dentro del espacio vectorial de significados el vector «perro» se encuentra cercano al de «cachorro», «beagle», «bulldog», «poodle». Esto permite al modelo poder inferir nuevas relaciones en base a una cantidad reducida de entradas. Los *word-vectors* ponen ese hecho a disposición del modelo de reconocimiento de entidades. Si bien no existen ejemplos de «Manaos» etiquetados como «Producto»; se verá que «Manaos» tiene un vector de palabras que generalmente corresponde a los términos de un producto, por lo que puede hacer la inferencia. Y si se quiere, se puede llegar incluso al detalle de que es una «Gaseosa».

Otra forma interesante de analizar y entender los *word-vectors* en su contexto de espacio vectorial multidimensional de significados es a través del álgebra de vectores, como se muestra en el trabajo «*Towards Understanding Linear Word Analogies*» (Ethayarajh, Duvenaud, & Hirst, 2019):



**Figura 7:** Parallelogram structure in the vector space (by definition).

Es fácil de entender viendo la figura 7 que al realizar álgebra entre los diferentes vectores de significado se pueden inferir nuevos conceptos:

$$\vec{R}ey - \vec{H}ombre \approx \vec{Re}aleza$$

$$\vec{R}ey - \vec{H}ombre + \vec{M}ujer \approx \vec{Re}ina$$

### 3.5.2. Encode

Resuelve el problema: el contexto de los significados es relevante y esta siendo descartado,

El resultado de esta etapa es la de codificar vectores **independientes-de-contexto** en matrices de oración **sensibles-al-contexto**.

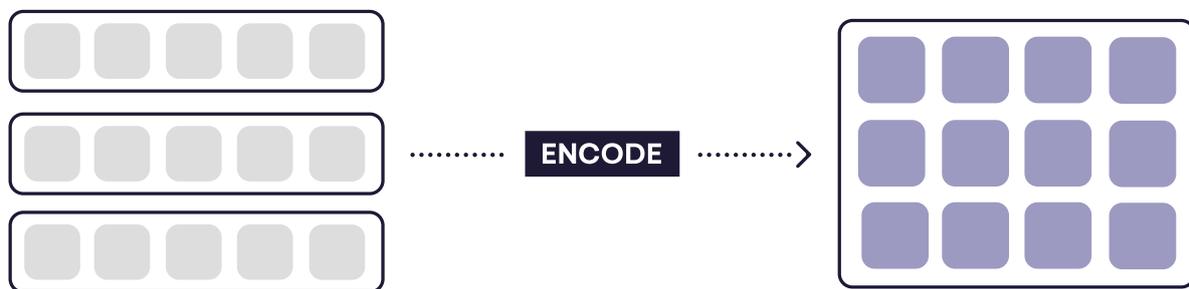


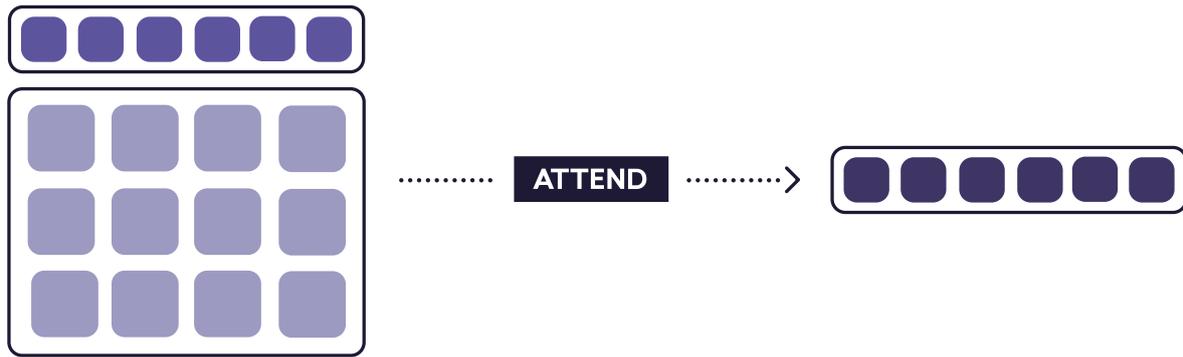
Figura 8: Encode.

La tecnología utilizada para esta etapa es una Red Neuronal Convolutiva en la que la oración es analizada como una *moving window* de tres vectores en la que cada vector analiza su significado en relación con un vector previo y un vector posterior. Es decir, cada vector se estudia dentro del contexto de los dos vectores que lo rodean. Luego los vectores subsiguientes se evalúan de igual manera, por lo que se genera naturalmente un «efecto de decaimiento» en el que el contexto de los vectores más lejanos tiene una relevancia cada vez menor.

### 3.5.3. Attend

Resuelve el problema: tenemos demasiada información para inferir un significado específico al problema a resolver.

En esta etapa toda la información generada en las etapas anteriores es analizada a través de un vector de entrada o también conocido como «vector de consulta» o «vector de contexto» representado en la figura 9 como un vector más oscuro.



**Figura 9:** Attend.

Al reducir la matriz a un vector, necesariamente se está perdiendo información. Es por eso que el «vector de contexto» es crucial: Indica que información descartar, de modo que el «vector resumen» se adapte a la red que lo consume.

El análisis de estas estrategias de consulta escapa el alcance de este trabajo pero resulta un tema interesante de investigación en sí mismo. Por ejemplo, investigaciones recientes han demostrado que el mecanismo de atención es una técnica flexible y que se pueden usar nuevas variaciones para crear soluciones elegantes y poderosas. Por ejemplo, en el estudio de Ankur Parikh et al (Parikh, Täckström, Das, & Uszkoreit, 2016) presentan un mecanismo de atención que toma dos matrices de oraciones y genera un solo vector.

### 3.5.4. *Predict*

Resuelve el problema: necesito un valor específico y no una representación genérica abstracta.

Finalmente en esta etapa tenemos un nuevo «vector de significado» que resulta de la consulta a la etapa anterior. Es necesario ahora traducir este vector a un *token* efectivo. En el caso de *NER*, el *token* que interesa obtener es el de la etiqueta de entidad.



**Figura 10:** Predict.

### 3.6. Uso del modelo estadístico

Experimentos en inglés, holandés, alemán y español muestran que se pueden obtener resultados a-la-par del estado del arte mediante el uso de un «autómata finito determinístico de pila» en conjunto con una red neuronal (Lample, Ballesteros, Subramanian, Kawakami, & Dyer, 2016).

Este autómata de pila es el nexo entre la Red Neuronal Convolutiva (CNN) que contiene el modelo estadístico para predecir entidades en el texto completo. En el mismo, se envían cada uno de los estados por los que se mueve para ir generando entidades con una heurística del tipo *greedy*.

La figura 11 muestra las posibles acciones de transición de este autómata.

- SHIFT: consume una token del input y al mueve al stack para generar una nueva entidad.
- REDUCE: mueve el stack actual al output tagueado como entity.
- OUT: consume una token del input y la mueve al output directamente.

$Out_t$	$Stack_t$	$Buffer_t$	Action	$Out_{t+1}$	$Stack_{t+1}$	$Buffer_{t+1}$	Segments
$O$	$S$	$(\mathbf{u}, u), B$	SHIFT	$O$	$(\mathbf{u}, u), S$	$B$	—
$O$	$(\mathbf{u}, u), \dots, (\mathbf{v}, v), S$	$B$	REDUCE( $y$ )	$g(\mathbf{u}, \dots, \mathbf{v}, \mathbf{r}_y), O$	$S$	$B$	$(u \dots v, y)$
$O$	$S$	$(\mathbf{u}, u), B$	OUT	$g(\mathbf{u}, \mathbf{r}_\emptyset), O$	$S$	$B$	—

Figura 11: Transiciones del modelo Stack-LSTM indicando la acción aplicada y el estado resultante.

Para saber que acción tomar se consulta el modelo estadístico. En la figura 12 se puede ver un ejemplo de como se recorre una oración bajo el stack propuesto:

Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Figura 12: Secuencia de transiciones para el ejemplo “Mark Watney visited Mars.”<sup>en</sup> el modelo de Stack-LSTM.

- Primero se empieza con un stack vacío.
- Se consume «Mark» y la CNN predice que es una posible Persona. Lo envía al stack.
- Se consume «Watney» y la CNN predice que es una posible continuación de Persona. Lo envía al stack.
- Se consume «visited» y la CNN predice que esto no forma parte de una entidad. Por lo tanto antes se REDUCE la entidad «Mark Watney» del stack actual.
- Análogamente se detecta la entidad «Mars»

La predicción que realiza la CNN tiene un puntaje de 0 a 1; un porcentaje de certeza de que la token sea de un tipo específico. A veces las tokens pueden tener más de una etiqueta candidata a

ser utilizada y éste es el método para desempatar este conflicto. Cuanto más cercano a 0,5 es ese punaje mayor es la incerteza.

## 4. NERd (Implementación)

Definido el problema, queda claro que la creación de un modelo entrenado es de vital importancia para cualquier problema de etiquetado de entidades. Es por ello que en el presente proyecto final hemos creado una herramienta para el entrenamiento eficiente de modelos estadísticos así como también una interfaz y API para poder consultar entidades.

Para organizar este capítulo vamos a realizar una descripción basada en el modelo de vistas de arquitectura 4+1 (Figura 13).

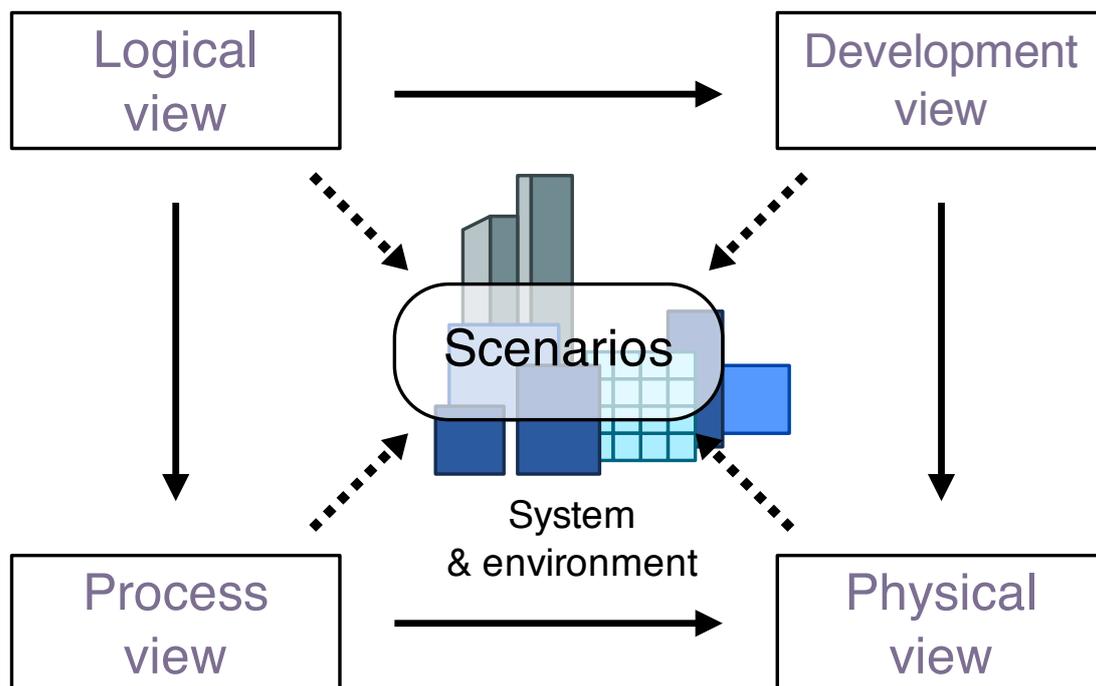


Figura 13: Ilustración de arquitectura 4+1.

Este modelo nos permite describir la aplicación de una manera genérica y ordenada.

*The «4+1» view model is rather «generic»: other notations and tools can be used, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.*

— (Kruchten, 1995)

### 4.1. Vista lógica

La vista lógica se refiere a la funcionalidad que el sistema proporciona a los usuarios finales.

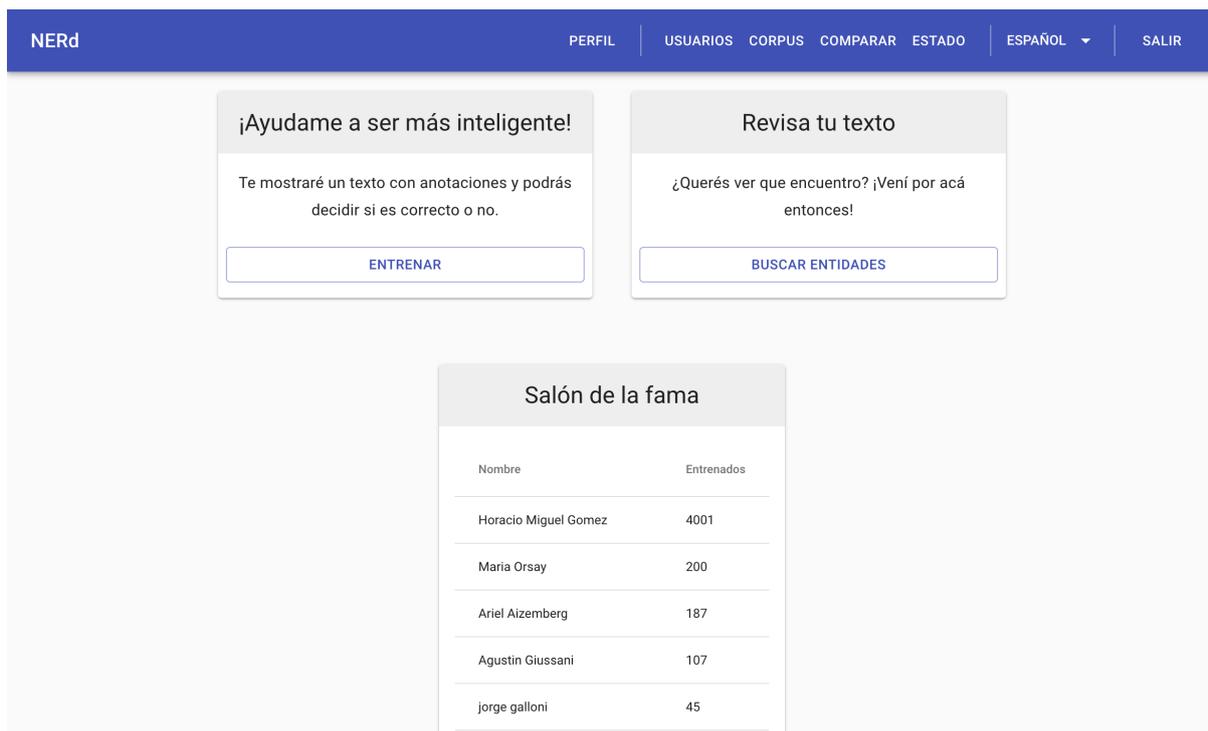
A continuación detallamos distintas partes del servicio NERd así como también de la interfaz de entrenamiento.

### 4.1.1. Web

La página *web* de *NERd* está enfocada en las tareas de mantenimiento de los servicios ofrecidos por el *API* así como también ofrece de interfaces que permiten a usuarios del sistema corregir de manera eficiente el modelo de inferencia.

#### 4.1.1.1. Inicio

La pantalla de inicio es donde se encuentran los accesos rápidos para entrenar el modelo o para buscar entidades en textos. También se encuentra aquí una lista de los 5 usuarios que más contribuyeron a entrenar el modelo. Detrás de esta funcionalidad se busca generar un espíritu competitivo entre los usuarios para que los mismos busquen contribuir más (Figura 14). A este tipo de técnicas se las denomina *gamification*.



The screenshot shows the NERd web interface. At the top is a blue navigation bar with the logo 'NERd' on the left and menu items 'PERFIL', 'USUARIOS', 'CORPUS', 'COMPARAR', 'ESTADO', 'ESPAÑOL', and 'SALIR' on the right. Below the navigation bar are two main action cards. The first card, titled '¡Ayúdame a ser más inteligente!', contains the text 'Te mostraré un texto con anotaciones y podrás decidir si es correcto o no.' and a blue button labeled 'ENTRENAR'. The second card, titled 'Revisa tu texto', contains the text '¿Querés ver que encuentro? ¡Vení por acá entonces!' and a blue button labeled 'BUSCAR ENTIDADES'. Below these cards is a 'Salón de la fama' section featuring a table with the following data:

Nombre	Entrenados
Horacio Miguel Gomez	4001
Maria Orsay	200
Ariel Aizemberg	187
Agustin Giussani	107
jorge galloni	45

**Figura 14:** Pantalla de inicio con usuario logueado.

Si la persona no cuenta con permisos de entrenador, se le sugiere que contacte a un administrador para que le otorgue el permiso (Figura 15).

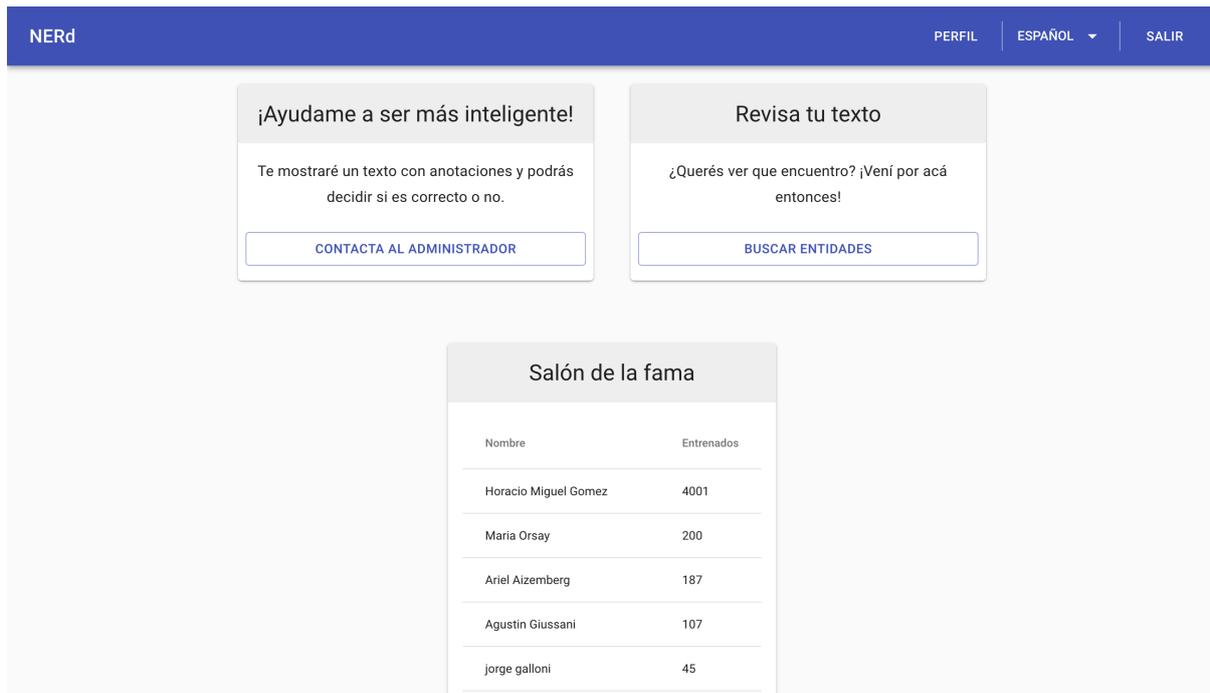


Figura 15: Pantalla de inicio sin rol de entrenador.

Si la persona visitando la página no cuenta con una sesión activa, se le invita a ingresar con una cuenta pre-existente o a registrarse (Figura 16).

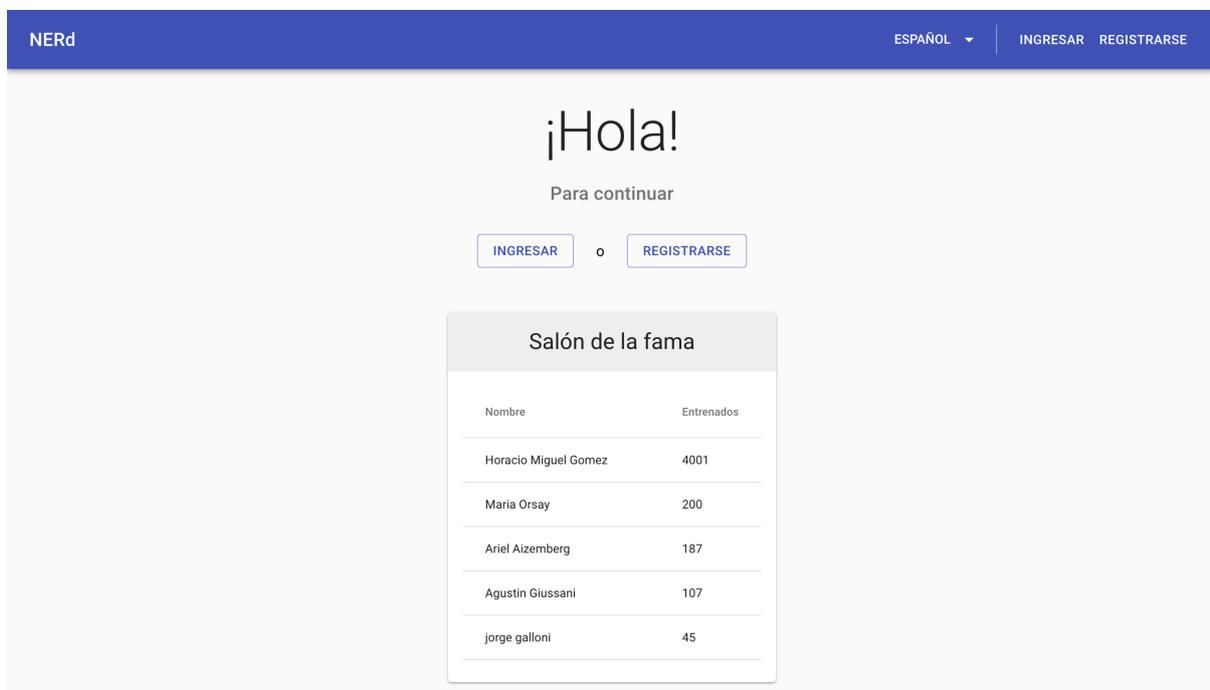


Figura 16: Pantalla de inicio sin sesión.

#### 4.1.1.2. Entrenamiento

La pantalla de entrenamiento es el núcleo de la web donde se entrena el modelo.

El usuario es presentado con un texto perteneciente al *corpus* del servicio con las entidades inferidas por el modelo actual. Con un editor especial que recibe un documento generado por *spaCy* que incluye *tokens* de palabras y entidades, le permitimos al usuario poder corregir las entidades inferidas y enviarle la corrección al servicio. Esa corrección será utilizada posteriormente a la hora de mejorar el modelo actual (Figura 17).



Figura 17: Pantalla de entrenamiento.

#### 4.1.1.2.1. Usabilidad

Tuvimos un foco fuerte en la usabilidad del componente ya que los entrenadores del servicio van a pasar prácticamente todo su tiempo en ésta pantalla, por lo que se tuvieron las siguientes consideraciones en la implementación.

Llamado a acción y ayuda

Dado que lo primero que ve el usuario es un texto con etiquetas de entidad, agregamos un título que invita al usuario a realizar acciones sobre el texto. De esta manera, le mostramos las dos acciones principales realizables desde el componente de entrenamiento: Click en alguna palabra o entidad y arrastrar un conjunto de palabras para crear una entidad nueva. Como refuerzo de este llamado a acción, agregamos un botón que al ser clickeado muestra un mensaje de ayuda con instrucciones más detalladas sobre el objetivo del entrenador y las acciones que deben de realizarse sobre el mismo (Figura 18).



¡Hacé click o arrastrá las palabras para corregir!

RESETEAR ? ACEPTAR

De cara a los **MISC Oscar**, "**LOC Roma**" se llevó el **MISC Goya** a la **MISC Mejor Película Latinoamericana**

**Como entrenar**

El objetivo es supervisar a un programa que intenta extraer información de textos.

Si ves algo que esté de más, podés removerlo haciendo clic en ello y apretando el botón BORRAR.

Si ves algo que esté mal etiquetado, hacé clic en ello y corregí la etiqueta con la que se adecue mejor.

Si querés agregar una etiqueta, hacé clic o seleccioná un grupo de palabras para crear un nuevo grupo y asignele una etiqueta

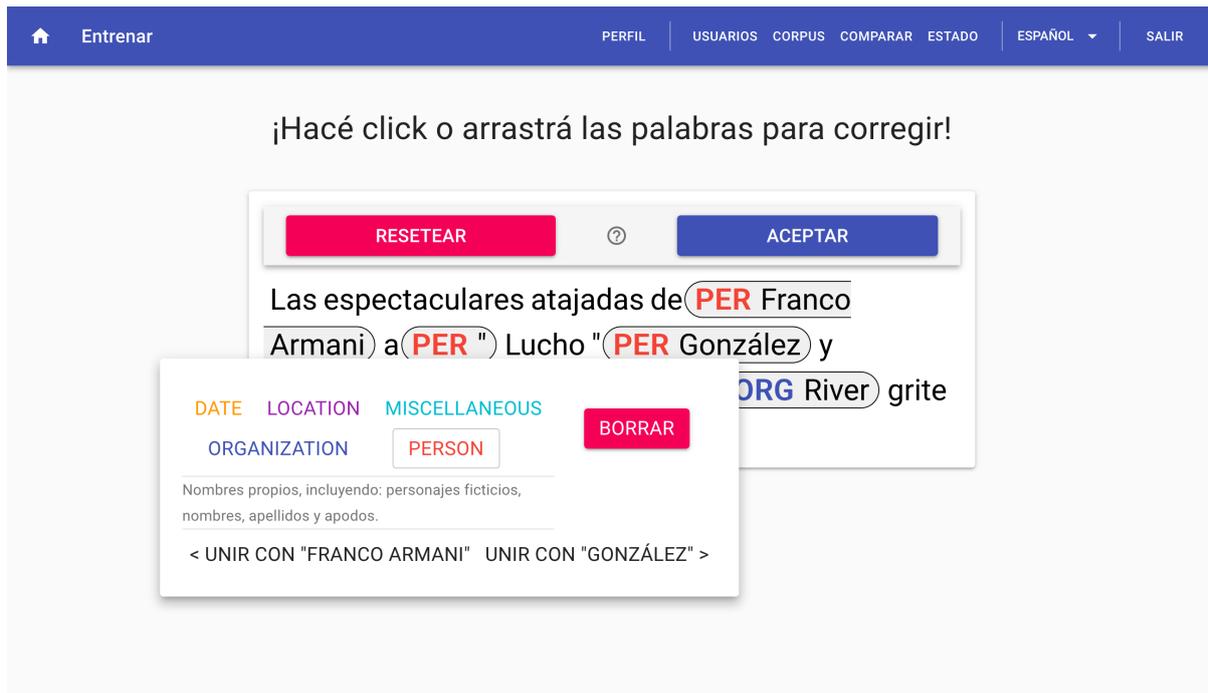
Cuando no haya más cambios para hacer, hacé click en el botón ACEPTAR.

**Figura 18:** Ayuda del entrenador.

## Etiquetado de entidades

Para etiquetar entidades decidimos ofrecer dos maneras: La más sencilla es hacer click en cada palabra. La segunda, para cuando una entidad esta definida por múltiples palabras, es arrastrar el cursor entre ellas. Dependiendo de la ubicación en la selección dentro del texto:

- Si no existen entidades en el texto actual, se le asigna por defecto el tipo *MISC* y se muestran el resto de los tipos para permitir cambiarlo de ser necesario.
- Si existen entidades antes o después, se ofrece la opción de unir la palabra actual con la entidad más próxima para el lado elegido.



**Figura 19:** Edición de entidad.

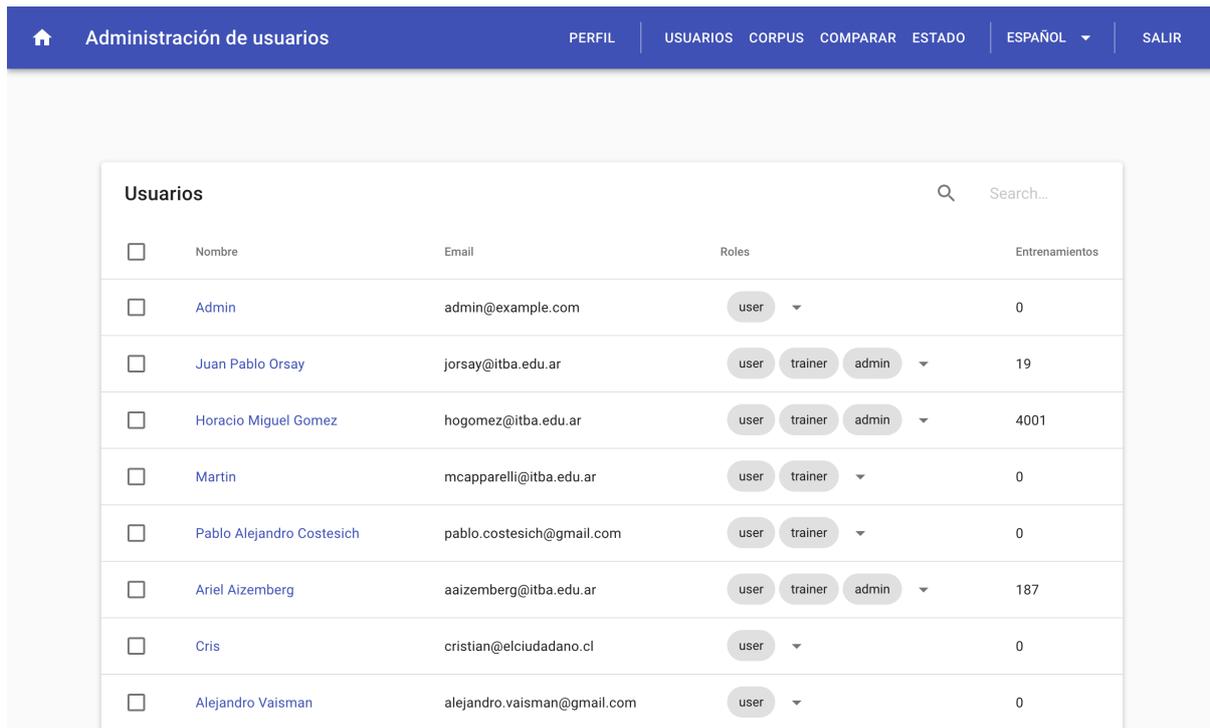
Como podemos ver en la figura 19, si el modelo infirió una entidad de manera incorrecta, ya sea por que se trata de una entidad inválida o agregó palabras de más a la etiqueta, permitimos que el usuario remueva la entidad y que después vuelva a etiquetar correctamente.

Optimización en tiempos de carga

Es esperado que un usuario entrene más de un texto, al momento de pedir un texto para mostrar, se pide el siguiente. Mediante este mecanismo de pre-carga, podemos eliminar el tiempo de espera entre texto y texto ofreciendo al usuario entrenador una experiencia completamente fluida.

#### 4.1.1.3. Administración de usuarios

La pantalla de *Administración de usuarios* permite a los usuarios con el rol de administrador poder modificar los roles de todos los usuarios del sistema, borrarlos o acceder a los detalles del usuario, tal como la lista de textos entrenados (Figura 20).



<input type="checkbox"/>	Nombre	Email	Roles	Entrenamientos
<input type="checkbox"/>	Admin	admin@example.com	user	0
<input type="checkbox"/>	Juan Pablo Orsay	jorsay@itba.edu.ar	user trainer admin	19
<input type="checkbox"/>	Horacio Miguel Gomez	hogomez@itba.edu.ar	user trainer admin	4001
<input type="checkbox"/>	Martin	mcapparelli@itba.edu.ar	user trainer	0
<input type="checkbox"/>	Pablo Alejandro Costesich	pablo.costesich@gmail.com	user trainer	0
<input type="checkbox"/>	Ariel Aizemberg	aaizemberg@itba.edu.ar	user trainer admin	187
<input type="checkbox"/>	Cris	cristian@elciudadano.cl	user	0
<input type="checkbox"/>	Alejandro Vaisman	alejandro.vaisman@gmail.com	user	0

Figura 20: Administración de usuarios.

#### 4.1.1.4. Perfil del usuario

Al hacer click Perfil, el usuario con sesión activa puede ver sus entrenamientos y cambiar su contraseña (Figura 21).



**Cambiar contraseña**

Contraseña \*  Confirmar contraseña \*

**ACTUALIZAR**

**Entrenamientos**

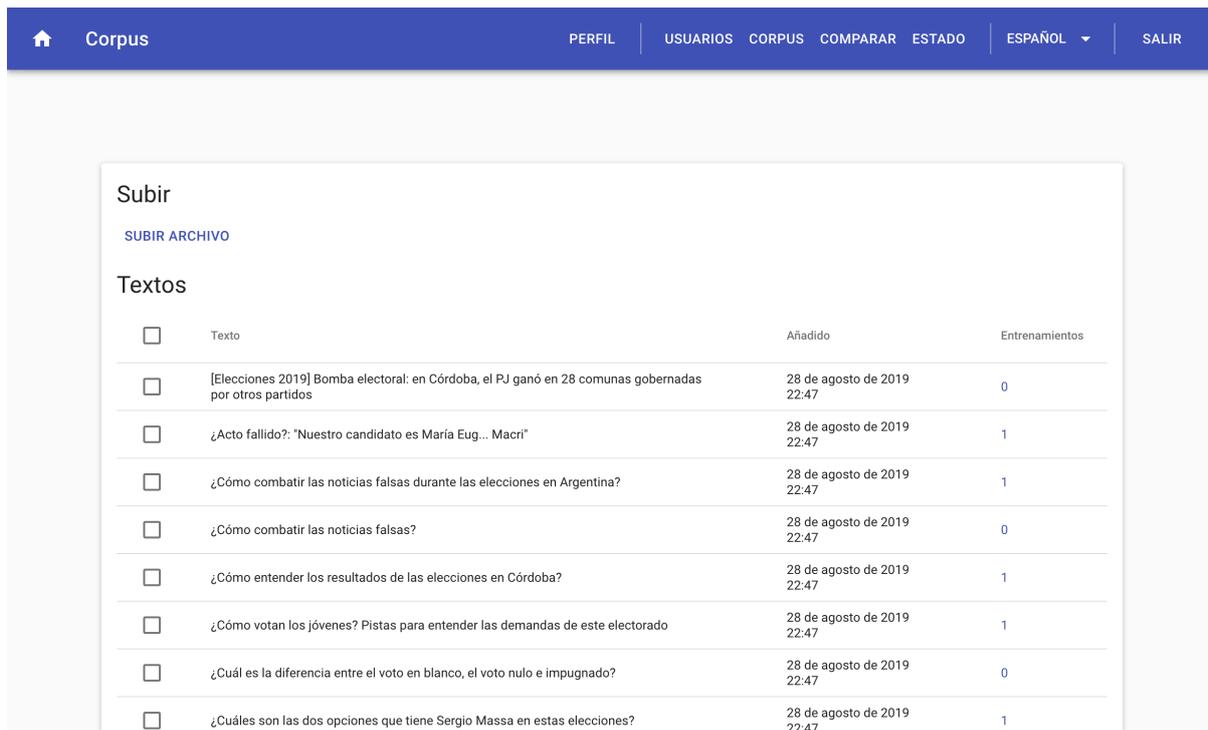
- Entrenamientos
- PER** Matías Martin habló de la desvinculación de **PER** Cabito de " Basta de Todo " : " Es una decisión dolorosa , de la cual me hago ...
- En **DATE** 2018 , el **MISC** déficit comercial con **LOC** Brasil se redujo a la mitad
- La reacción del **PER** Papa cuando una joven interrumpió su audiencia y se puso a jugar
- La ciencia empieza a entender el **MISC** alzhéimer
- Acusan a empleados de un súper de golpear y matar a un jubilado que había robado chocolates , un queso y una botella de aceite

Figura 21: Perfil de usuario.

Los usuarios con rol administrador pueden realizar las acciones mencionadas previamente pero sobre cualquier usuario haciendo click en el nombre del en la figura 20.

#### 4.1.1.5. *Corpus*

La pantalla de *corpus* permite a un usuario con el rol de administrador realizar tareas relacionadas con el conjunto de textos ingresados en el sistema (Figura 22).



<input type="checkbox"/>	Texto	Añadido	Entrenamientos
<input type="checkbox"/>	[Elecciones 2019] Bomba electoral: en Córdoba, el PJ ganó en 28 comunas gobernadas por otros partidos	28 de agosto de 2019 22:47	0
<input type="checkbox"/>	¿Acto fallido?: "Nuestro candidato es María Eug... Macri"	28 de agosto de 2019 22:47	1
<input type="checkbox"/>	¿Cómo combatir las noticias falsas durante las elecciones en Argentina?	28 de agosto de 2019 22:47	1
<input type="checkbox"/>	¿Cómo combatir las noticias falsas?	28 de agosto de 2019 22:47	0
<input type="checkbox"/>	¿Cómo entender los resultados de las elecciones en Córdoba?	28 de agosto de 2019 22:47	1
<input type="checkbox"/>	¿Cómo votan los jóvenes? Pistas para entender las demandas de este electorado	28 de agosto de 2019 22:47	1
<input type="checkbox"/>	¿Cuál es la diferencia entre el voto en blanco, el voto nulo e impugnado?	28 de agosto de 2019 22:47	0
<input type="checkbox"/>	¿Cuáles son las dos opciones que tiene Sergio Massa en estas elecciones?	28 de agosto de 2019 22:47	1

**Figura 22:** Administración de corpus.

Desde aquí es posible agregar textos al *corpus* utilizando la funcionalidad de subida de archivos. Los archivos deben ser archivos con extensión *.txt* y cada línea del archivo será agregada al *corpus* como un texto individual.

También es posible desde aquí ver todos los textos que forman parte del *corpus* así como también poder ver los entrenamientos para cada uno de los textos. Finalmente, es posible quitar textos del *corpus* así como también eliminar correcciones a las inferencias de entidades cargados por usuarios.

#### 4.1.1.6. *Estado*

La pantalla de *Estado* permite a un usuario con el rol de administrador visualizar el estado de entrenamiento del *corpus* así como también realizar diversas acciones sobre los *workers* (Figura 23).

Administración del Corpus

PERFIL | USUARIOS | CORPUS | COMPARAR | ESTADO | ESPAÑOL | SALIR

Corpus **VER**

Estado  
4589 de 43751 entrenados

Total entrenados  
4590

Sin entrenar  
0

Crear snapshot

Tipos: **DATE** Date **LOC** Location **MISC** Miscellaneous **ORG** Organization **PER** Person **NUEVO**

**CREAR**

Snapshots

<input type="checkbox"/>	Version	Creado el	Último entrenamiento	Estado	Trabajadores	Acciones
<input type="checkbox"/>	Actual	7 de octubre de 2019 17:02	hace 3 horas	Listo	1	
<input type="checkbox"/>	1	7 de octubre de 2019 7:59	hace un mes	Listo	1	
<input type="checkbox"/>	2	7 de octubre de 2019 8:57	Nunca	Listo	Ninguno	
<input type="checkbox"/>	3	7 de octubre de 2019 8:57	hace 5 días	Entrenando	Ninguno	

Reasignar trabajador

vCURRENT a v1 **APLICAR**

Figura 23: Información de corpus y manejo de workers.

#### 4.1.1.6.1. Secciones

##### Corpus

Es esta columna a la izquierda se puede ver rápidamente que porcentaje de el *corpus* contiene correcciones por usuarios así como también saber la cantidad total de correcciones del sistema (un texto puede tener más de una corrección por distintos usuarios) y también presenta un botón que permite al administrador ir a la pantalla de *Corpus*.

##### Crear *snapshot*

En esta sección, el administrador puede crear un *snapshot* del estado actual de entrenamiento del modelo. Esta funcionalidad es útil para comparar el avance del entrenamiento en diferentes momentos de la historia del mismo.

Cuando se crea un *snapshot* se incrementa la última versión disponible en uno y se gesta una nueva versión actual. Además existe la posibilidad de editar los posibles tipos de entidad del sistema en este momento. Como consecuencia de esto, el sistema se asegura que todo cambio de etiquetas tenga su correspondiente versión.

##### *Snapshots*

Sección en la cual podemos ver la lista completa de *snapshots*. Para cada *snapshot*, se muestra cuando fue la última vez que se entrenó así como también cuantos trabajadores tiene asignados.

Finalmente es posible desde aquí forzar a entrenar el modelo para ese snapshot en particular y también se presenta la opción para desentrenar, borrando el modelo guardado en el disco.

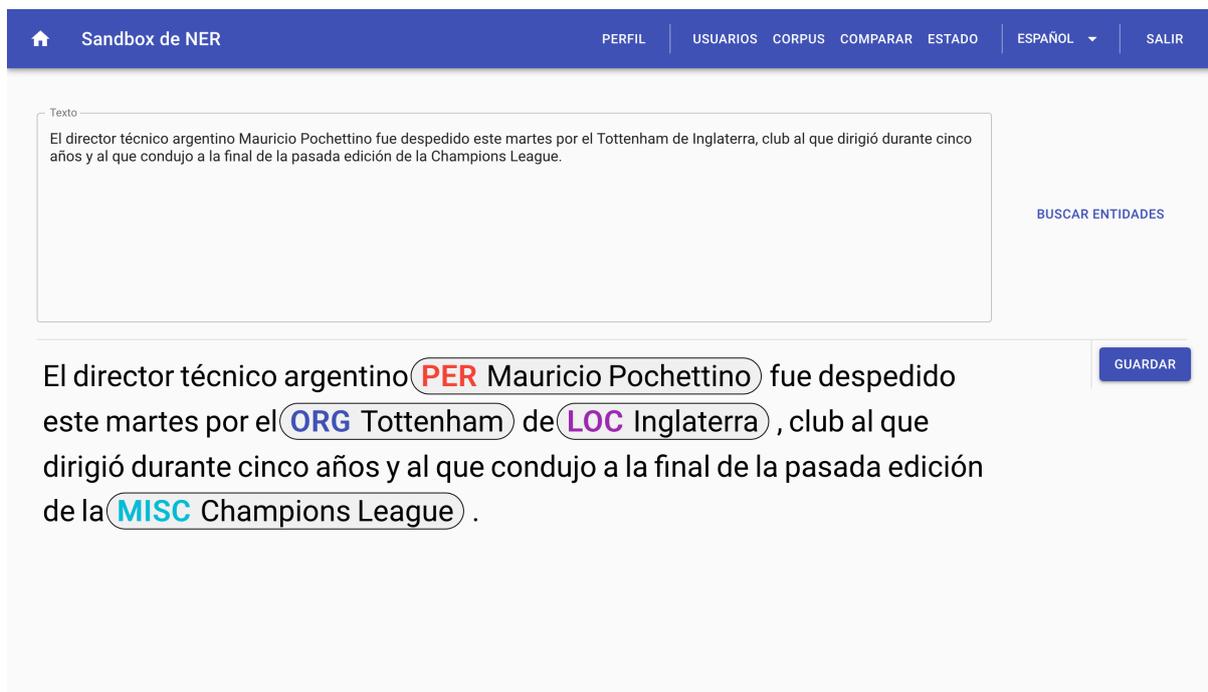
Cuando el sistema entrena un *snapshot* se tendrá en cuenta todos los textos que hayan sido anotados previamente a la creación del mismo.

Reasignar *worker*

Sección que permite reasignar *workers* (los servicios encargados de realizar operaciones de *NLP*) para que sirvan un snapshot distinto. De esta manera se pueden servir distintas versiones del modelo de inferencia para poder realizar pruebas sobre los mismos.

#### 4.1.1.7. Sandbox

Se accede a la pantalla de *sandbox* desde el inicio haciendo click en el botón de «Buscar entidades»; permite a los usuarios ingresar sus textos y obtener las entidades nombradas de los mismos haciendo consultas al servicio NERd del *snapshot* actual. Adicionalmente, si el usuario tiene el rol de entrenador, podrá corregir las entidades inferidas y agregar el texto con sus correcciones al *corpus* (Figura 24).



The screenshot shows the 'Sandbox de NER' interface. At the top, there is a navigation bar with a home icon, the title 'Sandbox de NER', and menu items: 'PERFIL', 'USUARIOS', 'CORPUS', 'COMPARAR', 'ESTADO', 'ESPAÑOL', and 'SALIR'. Below the navigation bar, there is a text input area labeled 'Texto' containing the sentence: 'El director técnico argentino Mauricio Pochettino fue despedido este martes por el Tottenham de Inglaterra, club al que dirigió durante cinco años y al que condujo a la final de la pasada edición de la Champions League.' To the right of the input area is a button labeled 'BUSCAR ENTIDADES'. Below the input area, the same sentence is displayed with entities recognized and highlighted in colored boxes: 'PER' (red) for 'Mauricio Pochettino', 'ORG' (blue) for 'Tottenham', 'LOC' (purple) for 'Inglaterra', and 'MISC' (cyan) for 'Champions League'. To the right of the output area is a button labeled 'GUARDAR'.

Figura 24: Inferencia de entidades en sandbox.

#### 4.1.1.8. Comparar (*snapshots*)

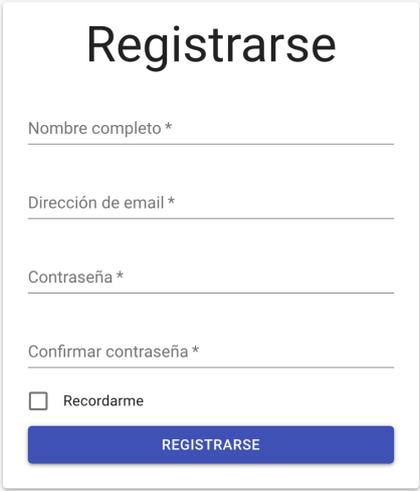
Esta sección es accesible únicamente por administradores y permite comparar las entidades inferidas por dos *snapshots* distintos, con la opción de marcar visualmente aquellos textos cuyas enti-

dades difieren. A su vez, si el usuario logueado tiene el permiso de entrenador, podrá corregir de manera inline los errores en la inferencia del modelo actual (Figura 25).

Figura 25: Comparativa de modelos.

#### 4.1.1.9. Registración

Sección únicamente accesible cuando no hay una sesión activa. Aquí se registran los usuarios con la opción de que el inicio de sesión persista luego de que se cierre la pestaña del navegador (Figura 26).



**Registrarse**

Nombre completo \*

Dirección de email \*

Contraseña \*

Confirmar contraseña \*

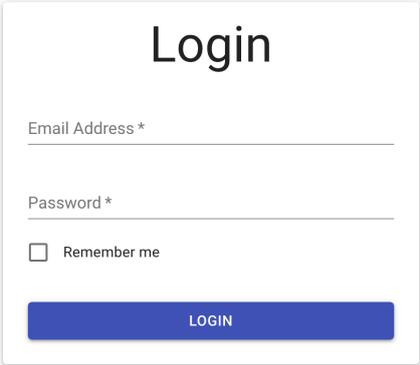
Recordarme

REGISTRARSE

**Figura 26:** Registración.

#### 4.1.1.10. Login

Sección únicamente accesible cuando no hay una sesión activa. Aquí ingresan los usuarios al sistema con la opción de que el inicio de sesión persista luego de que se cierre la pestaña del navegador (Figura 27).



**Login**

Email Address \*

Password \*

Remember me

LOGIN

**Figura 27:** Inicio de sesión.

#### 4.1.1.11. Barra de navegación

La barra de navegación muestra distinta información dependiendo del contexto actual del sistema.

Cuando no hay una sesión activa, se muestran únicamente las opciones para acceder al sistema, ya sea registrándose o ingresando utilizando credenciales (Figura 28).



Figura 28: Usuario anónimo.

Cuando hay una sesión activa el usuario cuenta con el rol de *USER* y/o de *TRAINER* pero **no** con el de *ADMIN*, podrá acceder únicamente a su perfil (Figura 29).



Figura 29: Usuario regular.

Finalmente, cuando el usuario logueado tiene el rol de *ADMIN*, tiene acceso a todas las secciones de administración del sistema (Figura ??fig:logic-status-admin)).

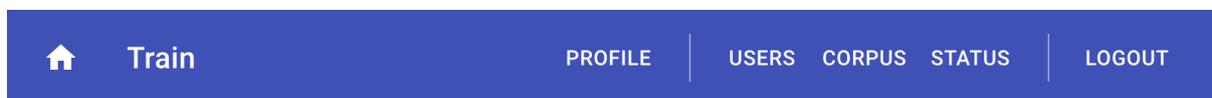


Figura 30: Usuario administrador.

### 4.1.2. API

Para este proyecto tomamos la decisión de ofrecer una *API REST*. Dado que *REST* es un estilo de arquitectura y **no un standard** históricamente ha sido difícil ofrecer este tipo de servicio de una manera amena para el developer.

En los últimos años han surgido especificaciones bien formadas para la definición de *REST APIs*, en particular *The Linux Foundation* ha promovido que la industria utilice *OPEN API*.

Es por ello que toda la especificación de nuestra *API* es *OpenAPI 3 compliant*.

#### 4.1.2.1. Documentación

Para visualizar la documentación de todos los *endpoints* del *API*, es posible utilizar dos medios: *Swagger UI* y *ReDoc*. Gracias al uso de *OpenAPI* estas documentaciones han sido autogeneradas.

#### 4.1.2.1.1. Swagger UI

La documentación provista por *Swagger UI* permite ver el detalle de cada uno de los *endpoints* del servicio así como también permite realizar consultas de prueba contra el servidor real, prestándose así como una especie de *sandbox* (Figura 31).

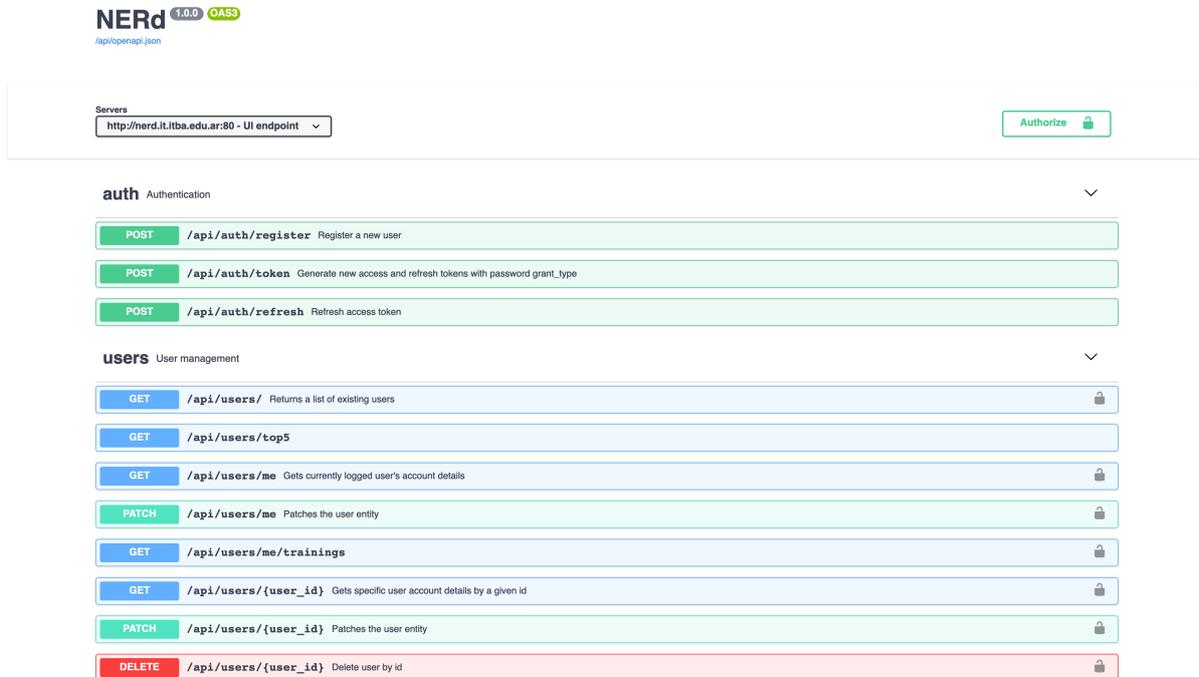


Figura 31: Swagger UI.

Para cada *endpoint*, la interfaz muestra que parámetros son requeridos (Figura 32). Así como también permite, mediante el uso del botón *Try it out*, completar esos parámetros y ejecutar el pedido al servidor, mostrando los resultados dentro de la interfaz.

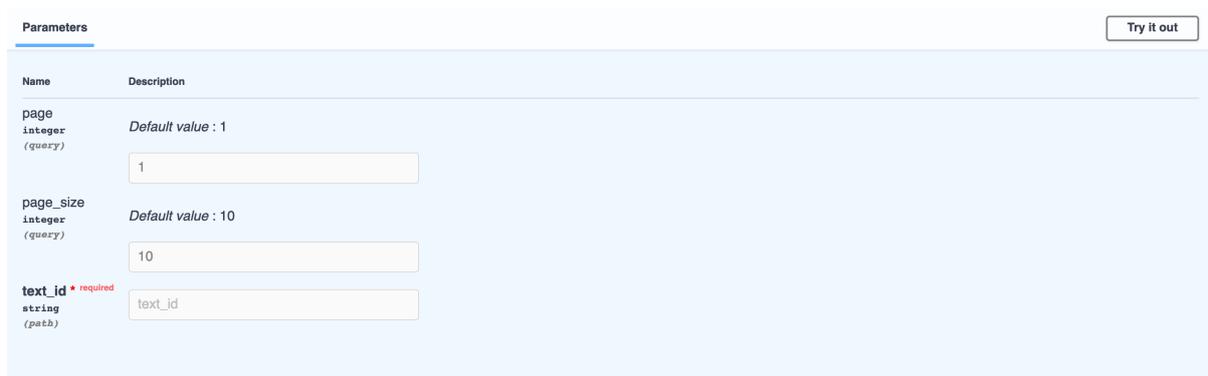


Figura 32: Ejemplo de documentación con Swagger para request.

Para las respuesta, *Swagger* muestra la forma del *JSON* de respuesta para cada *HTTP code* distinto (Figura 32)

**Responses**

Code	Description	Links
200	Get trainings	No links
401	Unauthorized	No links
422	Unprocessable Entity	No links

Media type:

Controls Accept header.

Example Value | Schema

```
[
  {
    "created_at": "2019-11-22T01:32:01.049Z",
    "id": "string",
    "document": {
      "created_at": "2019-11-22T01:32:01.049Z",
      "tokens": [
        {
          "pos": "string",
          "tag": "string",
          "head": 0,
          "start": 0,
          "id": 0,
          "dep": "string",
          "end": 0
        }
      ],
      "sents": [
        {
          "start": 0,
          "end": 0
        }
      ],
      "ents": [
        {
          "label": "string",
          "start": 0,
          "end": 0
        }
      ]
    },
    "text": "string"
  }
]
```

Media type:

Example Value | Schema

```
{
  "status": "string"
}
```

Media type:

Example Value | Schema

```
{
  "status": "string"
}
```

Figura 33: Ejemplo de documentación con Swagger para respuestas.

#### 4.1.2.1.2. ReDoc

ReDoc es una alternativa a Swagger para visualizar la documentación de una especificación de OpenAPI que presenta un diseño más moderno así como también se enfoca más en mostrar la información de manera más clara.

A diferencia de Swagger UI, ReDoc no permite realizar pruebas contra el servidor para el cual se está mostrando la documentación (Figura 34) pero ofrece una mejor experiencia de usuario a la hora de navegar la API.

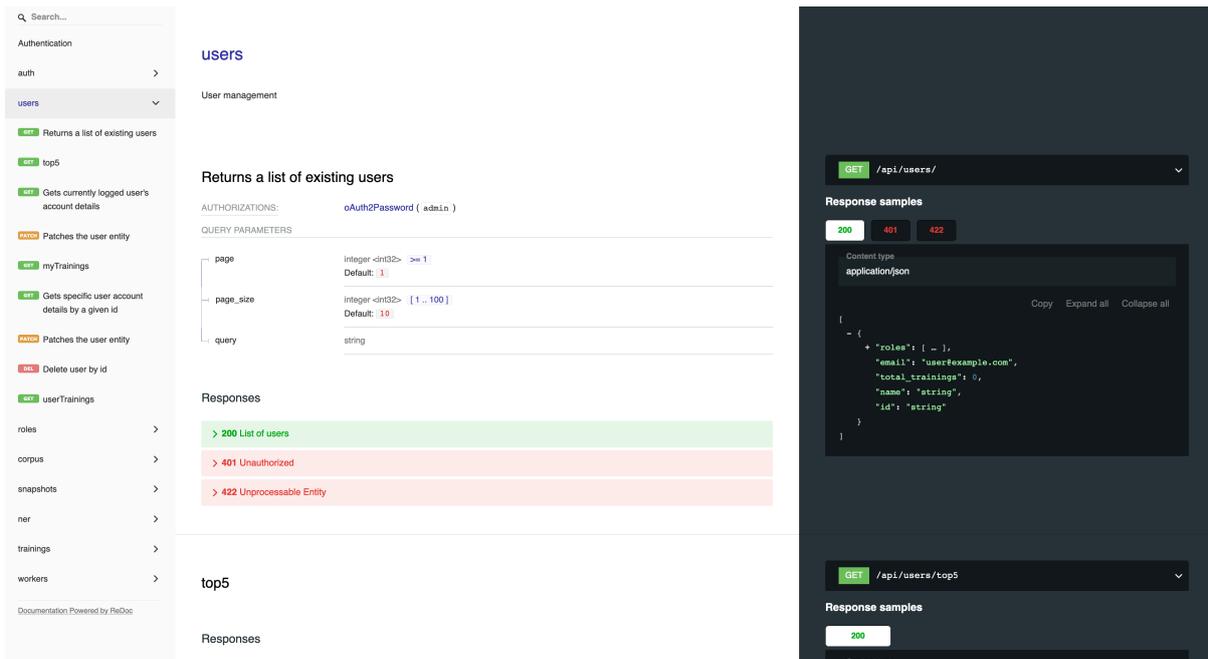


Figura 34: Ejemplo de documentación con ReDoc.

## 4.1.2.2. Endpoints

### 4.1.2.2.1. Autenticación

Rutas del API dedicadas a la autenticación de usuarios.

- **POST** /api/auth/register
  - Registrar un usuario nuevo.
- **POST** /api/auth/token
  - Genera un nuevo *token* de acceso y refresco con credenciales.
  - Utilizado para la funcionalidad de *login*.
- **POST** /api/auth/refresh
  - Refresca el *token* de acceso.
  - Utilizado cuando un *token* de acceso caducó.

### 4.1.2.2.2. Usuarios

Conjunto de operaciones relacionadas con los usuarios del sistema.

- **GET** /api/users
  - Lista de usuarios existentes.
  - Separa los resultados en páginas.
- **GET** /api/users/top5

- Lista de los 5 usuarios con más entrenamientos.
- **GET** /api/users/me
  - Retorna la información del usuario logueado.
- **PATCH** /api/users/me
  - Actualiza la información del usuario logueado.
- **GET** /api/users/me/trainings
  - Retorna los entrenamientos del usuario logueado.
  - Separa los resultados en páginas.
- **GET** /api/users/{user\_id}
  - Retorna la información del usuario especificado por user\_id.
- **PATCH** /api/users/{user\_id}
  - Actualiza la información del usuario especificado por user\_id.
- **DELETE** /api/users/{user\_id}
  - Borra al usuario especificado por user\_id.
- **GET** /api/users/{user\_id}/trainings
  - Retorna los entrenamientos del usuario especificado.

#### 4.1.2.2.3. Roles

- **GET** /api/roles
  - Retorna la lista de todos los roles asignables a usuarios del sistema.

#### 4.1.2.2.4. Corpus

Rutas dedicadas a operaciones con el *corpus* del sistema.

- **GET** /api/corpus/{text\_id}
  - Retorna los detalles del texto especificado por text\_id.
- **DELETE** /api/corpus/{text\_id}
  - Borra un texto especificado por text\_id del corpus.
- **GET** /api/corpus/{text\_id}/trainings
  - Retorna la lista de entrenamientos proporcionados por los usuarios sobre las entidades en el texto.
- **PUT** /api/corpus/{text\_id}/trainings
  - Agrega un entrenamiento para el texto con id *text\_id*.
- **POST** /api/corpus/upload
  - Permite agregar textos de manera masiva al sistema.
  - Acepta una lista de archivos .txt donde cada línea es un texto a agregar.
  - Los archivos deben ser UTF-8.

- **GET** /api/corpus
  - Lista de textos cargados en el sistema para entrenamiento.
  - Separa los resultados en páginas.
- **POST** /api/corpus
  - Agrega un texto al sistema para entrenamiento.

#### 4.1.2.2.5. Snapshots

Conjunto de operaciones relacionadas con los snapshots y workers.

- **GET** /api/snapshots
  - Listado de los snapshots disponibles.
  - Separa los resultados en páginas.
- **GET** /api/snapshots/{snapshot\_id}
  - Retorna información (tipos de entidades, fecha de creación, fecha de entrenamiento, etc.) sobre un *snapshot* específico.
- **DELETE** /api/snapshots/{snapshot\_id}
  - Borra un *snapshot* con el id especificado.
- **POST** /api/snapshots/{snapshot\_id}/force-train
  - Envía la tarea de entrenamiento a los workers que tienen el *snapshot* snapshot\_id cargado.
- **POST** /api/snapshots/{snapshot\_id}/force-untrain
  - Envía la tarea de desentrenar a los *workers* que tienen el *snapshot*. snapshot\_id cargado.
- **GET** /api/snapshots/current
  - Retorna información sobre el *snapshot* actual.
- **PUT** /api/snapshots/current
  - Crea un nuevo *snapshot* con la información provista.

#### 4.1.2.2.6. Reconocimiento de Entidades Nombradas

Conjunto de operaciones relacionadas al *Reconocimiento de Entidades Nombradas*

- **GET** /api/ner/train
  - Retorna un texto para que un usuario del sistema revise si está correctamente inferido.
  - Únicamente retorna textos que el usuario logueado no haya corregido ya
- **GET** /api/ner/compare/{first\_snapshot}/{second\_snapshot}
  - Compara el *NER* entre dos *snapshots* distintos.
- **POST** /api/ner/current/parse
  - Retorna un documento spaCy para un texto dado utilizando el *snapshot* actual.

- **POST** /api/ner/{snapshot\_id}/parse
  - Retorna un documento *spaCy* para un texto dado utilizando el *snapshot* especificado.
- **POST** /api/ner/current/entities
  - Retorna la lista de *Entidades Nombradas* para un texto dado utilizando el modelo actual.
- **POST** /api/ner/{snapshot\_id}/entities
  - Retorna la lista de *Entidades Nombradas* para un texto dado utilizando el modelo especificado.

#### 4.1.2.2.7. Entrenamientos

- **DELETE** /api/trainings/{training\_id}
  - Borra un entrenamiento.

#### 4.1.2.2.8. Workers

- **GET** /api/workers/
  - Lista de los workers disponibles.
- **POST** /api/workers/reassign
  - Reasigna un trabajador de un a versión de *snapshot* a otra.

## 4.2. Vista de proceso

La vista de proceso trata los aspectos dinámicos del sistema, explica los procesos del sistema y cómo se comunican, y se centra en el comportamiento del sistema en tiempo de ejecución. La vista de proceso aborda concurrencia, distribución, integradores, rendimiento y escalabilidad, etc.

Como podemos ver en la figura 35, existen cinco tipos de proceso que juntos forman la totalidad de NERd y su entrenador.

Para la implementación de cada una de las partes tuvimos como primer objetivo lograr el desarrollo de un proyecto que pudiera escalar horizontalmente desde su concepción, es por ello que separamos cada parte del proyecto para poder lograr una arquitectura que escale naturalmente.

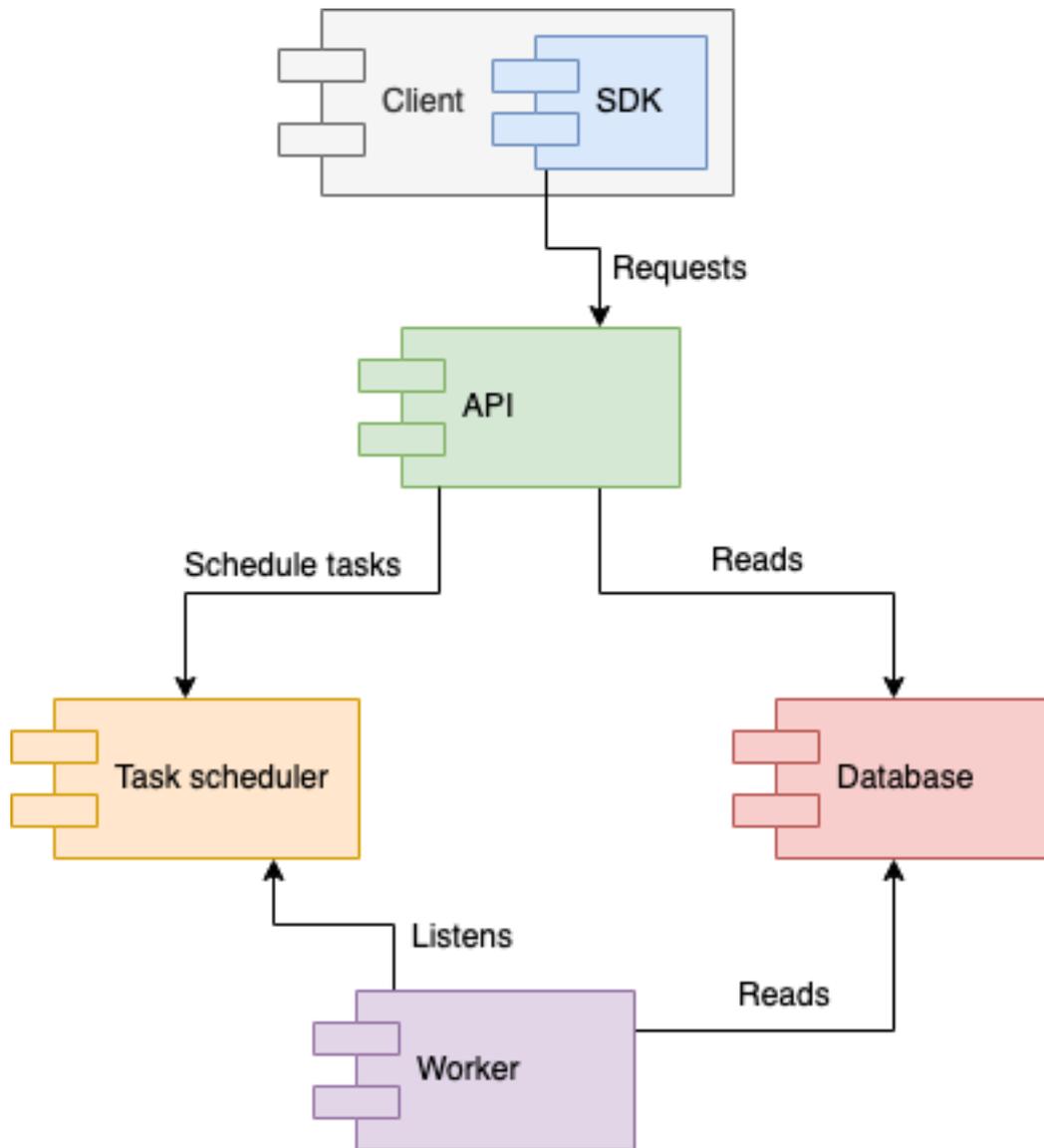


Figura 35: Componentes del sistema.

#### 4.2.1. Cliente web

Es la interfaz del entrenador que se comunica con el API utilizando un *SDK* auto-generado a partir de la definición de *OpenAPI*. Implementado en este proyecto es un cliente *web* el cual permite explotar las funcionalidades de entrenamiento así como de inferencia de entidades.

#### 4.2.2. API

Servicio en el cual los clientes pueden pedir las *Entidades Nombradas* de textos así como también permite la carga de textos con sus respectivas correcciones que luego serán utilizadas para entrenar el modelo de inferencia. El *API* se comunica el servicio de *Bases de Datos* para obtener o modificar información sobre los usuarios, los textos pertenecientes al *corpus* o información de los dis-

tintos *snapshots* y envía mensajes al *Task Scheduler* para realizar diversas tareas relacionadas con *spaCy* así como también tareas de administración sobre los *workers* disponibles.

Debido a que la *API* está implementada en *python* y este es un lenguaje cuyo intérprete posee un *lock* global ("Python," 2019), debimos implementar la aplicación como un servicio que implementa *WSGI* (Eby, 2010). Luego la aplicación es servida por *gunicorn*, un *application server* para este fin que escala horizontalmente su *pool* de *workers*.

### 4.2.3. Worker

Un *worker* se encarga de realizar las operaciones de *NER* incluyendo el entrenamiento de modelos de inferencia. Es un servicio que existe de manera independiente.

El modelo estadístico debe ser cargado en memoria para poder ser utilizado. El mismo requiere de muchos recursos y tiene una eficiencia limitada ya que puede atender una sólo consulta a la vez. Es por ello que decidimos, en lugar de que el modelo sea parte de la *API* resultando en un cuello de botella, encapsular el modelo en un servicio externo para que pueda ser escalado horizontalmente.

Como el *worker* es una unidad de trabajo que recibe tareas desde el *task scheduler*, es posible tener varios *workers* donde cada uno utiliza un modelo de inferencia distinto (generados a partir de distintos *snapshots*).

Cuando un *worker* es creado, automáticamente carga el modelo actual (*snapshot* actual) y se suscribe a la cola de mensajes para esa versión en particular.

#### 4.2.3.1. Entrenamiento y carga de snapshots

Dado que pueden haber varios *workers* utilizando el mismo modelo de *spaCy*, es de vital importancia que si existe al menos un *worker* leyendo un modelo desde el disco y otro *worker* quiere entrenar ese modelo, que no pueda hasta que el resto haya terminado de leerlo. Ésto se debe a que si el *worker* que está entrenando el modelo sobrescribe los archivos del disco mientras otro los está leyendo, va a llevar a llevar a la falla del *worker* de lectura. Como no hay ninguna limitación en la lectura, más de un *worker* puede leer en simultáneo.

Para realizar ese control de concurrencia utilizamos el campo *semaphore* del modelo de *snapshot* guardado en la base de datos (Figura 47) el cual usualmente se encuentra en 0. El semáforo es incrementado en uno cuando un *worker* comienza a leer o termina de entrenar el modelo; es decrementado en uno cuando termina de leer o comienza a entrenar el modelo. Con esas cuatro acciones es posible controlar la correcta lectura/escritura del modelo desde el disco en un entorno distribuido sin causar problemas de concurrencia.

En la figura 36 podemos ver un escenario en el que tanto *Worker 2* y *Worker 3* pueden leer en simultáneo el modelo pero *Worker 1* debe quedarse esperando hasta que el semáforo vuelva a cero (nadie leyendo o entrenando el modelo) para poder realizar la tarea de entrenamiento.

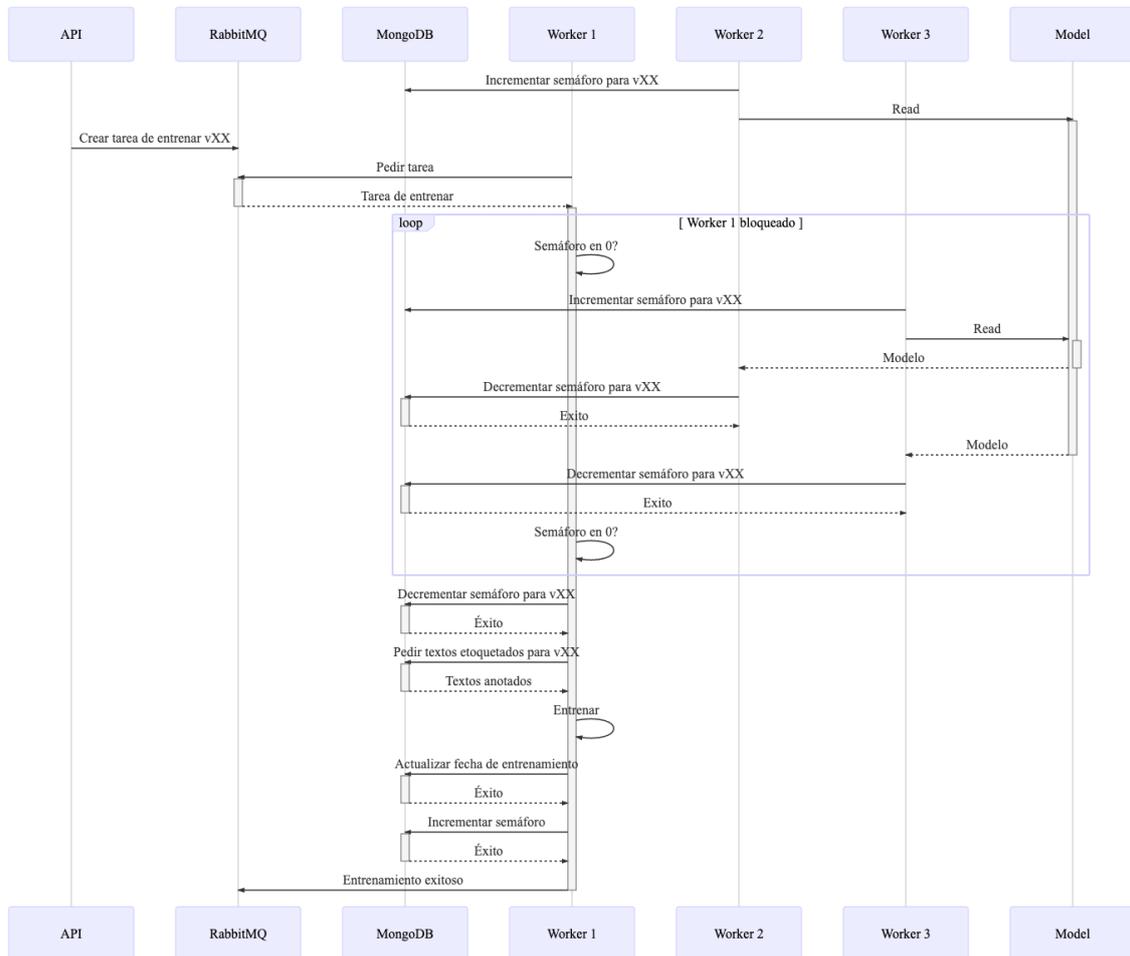


Figura 36: Bloqueo de entrenamiento por lectura.

#### 4.2.4. Task Scheduler

Recibe tareas desde el *API* tales como reconocer entidades, entrenar y desentrenar un modelo, cambiar el modelo de un worker entre otras. A el task scheduler se suscriben todos los *workers* del sistema de manera tal de poder recibir mensajes y realizar acciones. Su funcionamiento es crítico para poder manejar grandes volúmenes de consultas al motor de inferencia ya que permite generar una estructura que escale horizontalmente debido a la posibilidad de contar con la suscripción de un número arbitrario de *workers*.

Este proceso escala horizontalmente debido a que su implementación en la capa física es un *cluster*.

#### 4.2.5. Base de datos

Servicio que contiene la base de datos con la información necesaria (usuarios, snapshots) para que el servicio de *API* funcione así como también los datos necesarios por los workers (entrenamientos).

Este proceso escala horizontalmente debido a que su implementación en la capa física es un *cluster*.

##### 4.2.5.1. Autenticación

El proceso de autenticación de la *API* sigue el flujo de *OAuth2* (Group, 2017), debido a como está definido, el usuario al autenticarse con *email* y *password* recibe par de identificadores únicos llamados *access\_token* y *refresh\_token*. A partir de este momento todos los requests del usuario a la *API* poseen en su encabezado el *access\_token*. Ocurre que estos *tokens* de acceso son sólo válidos por un tiempo limitado (en nuestra implementación ese tiempo es de 15 minutos). Debido a eso, es necesario refrescar el *access\_token* utilizando el *refresh\_token* provisto en el *login* inicial. Al hacer esto el usuario recibe un nuevo par de *tokens* de igual manera a como lo hizo en el *login* con sus credenciales.

De esta manera se mejora la seguridad del sistema dado que el usuario sólo necesita enviarle al backend **una sola vez** sus credenciales sensibles.

En la figura 37 podemos ver el flujo que realiza la implementación *web* del entrenador para realizar el refresco de la sesión al recibir un 401 por parte del servidor.

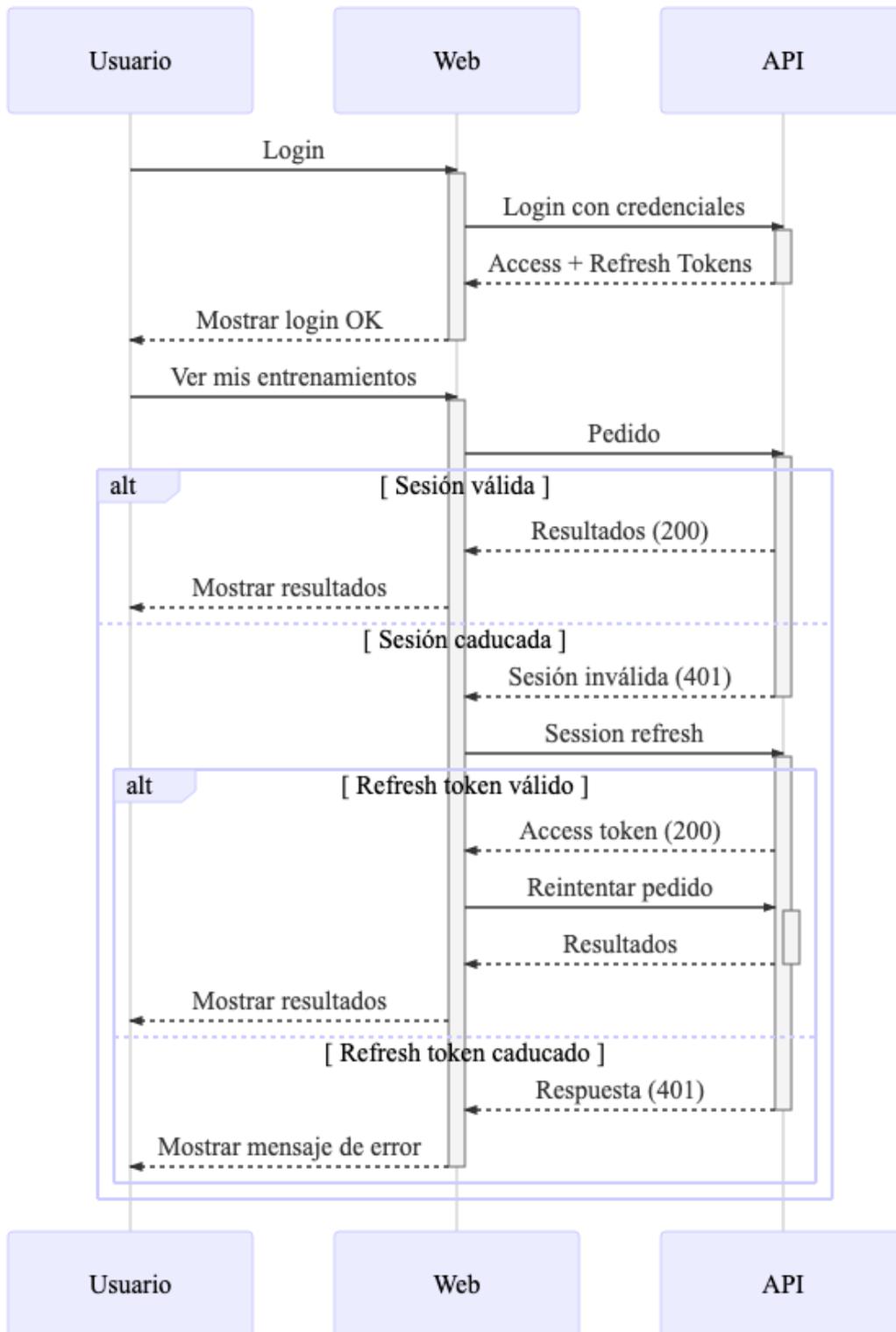


Figura 37: Flujo de relógin

### 4.3. Vista de desarrollo

La vista de desarrollo ilustra un sistema desde la perspectiva de un programador y se ocupa de la gestión de software. Esta vista también se conoce como la vista de imple-

mentación.

### 4.3.1. Cliente web

La página *web* que contiene la interfaz de administración así como también la de entrenamiento. Para implementarla decidimos utilizar el modelo de *Single Page Application* que nos permitió generar un *bundle* que puede ser hosteado en *CDNs* y nos permite tener ciclos de release independientes al los del servicio.

Poder separar las *releases* de la *web* y el servicio es importante ya que los clientes del servicio no necesariamente necesitan de la *web*, ya que pueden acceder al servicio mediante el *API REST* y una baja por cambios a la interfaz de la *web* no afecta la disponibilidad del servicio.

#### 4.3.1.1. Tecnologías

Para la implementación utilizamos el lenguaje de programación llamado **Typescript** ("Typescript," 2019), lenguaje desarrollado por *Microsoft* que compila a *JavaScript* con el agregado de *Tipos* que permiten producir código de mayor calidad debido al agregado de la posibilidad del chequeo estático de código para validar su correctitud.

Una vez definido el lenguaje, optamos por utilizar la librería llamada **ReactJS** (Facebook, 2019) desarrollada por *Facebook* para implementar la interfaz.

Sumado a **ReactJS**, incluimos diversas librerías para poder resolver varios problemas en el desarrollo de una *SPA*:

##### 4.3.1.1.1. Material UI

Provee de una biblioteca de componentes estilados siguiendo el patrón de diseño propuesto por *Google* llamado *Material Design*.

##### 4.3.1.1.2. React Router

Desarrollada por *React Training*, provee de mecanimos para la navegación en *SPAs*.

##### 4.3.1.1.3. i18next

Librería que provee mecanismos de internacionalización. A su vez, nos permitió generar los archivos de traducción mediante su extracción del código fuente utilizando la herramienta `i18next-scanner`.

#### 4.3.1.1.4. OpenAPI Generator

*OpenAPI Generator* es una herramienta que permite generar código a partir de una definición de *OpenAPI*. Debido a que el servicio implementado expone una definición de *OpenAPI*, podemos utilizar ésta librería para auto-generar todo el código necesario para comunicarnos con el servicio usando el generador de *Typescript*.

#### 4.3.1.2. Arquitectura

El código en la *web* se encuentra separado por funcionalidad, ya sean de utilidad, como modelos o funciones, o de funcionalidad, como la vista de la pantalla de inicio (Figura 38).

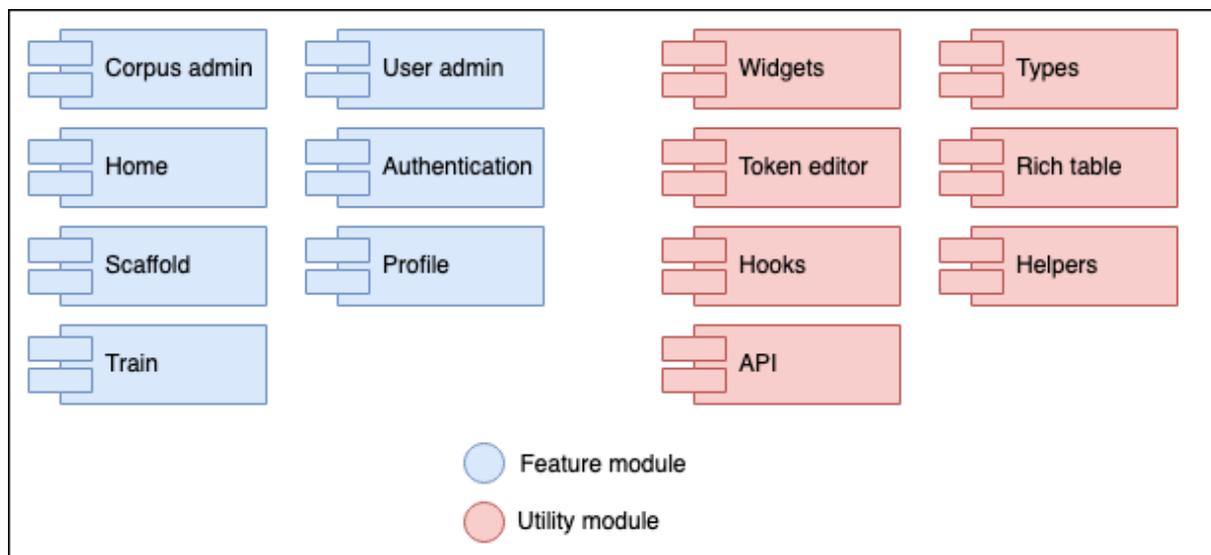


Figura 38: Módulos de la página web.

Los módulos marcados en azul son relacionados al contenido que se muestra cada ruta del sistema.

Los módulos marcados en rojo son de utilidad, ya sea por que contienen todo el código relevante a la comunicación con el *API REST* o a componentes utilizados en varias áreas de los módulos azules.

#### 4.3.1.3. Sesión

El servicio *web* ofrece la posibilidad de recordar la sesión de un usuario en el momento de realizar el login. De elegirse, las credenciales son guardadas en el *local storage* del cliente *web*, de manera tal que el usuario pueda utilizar la misma sesión por más de que haya cerrado el cliente *web*.

Si en algún momento en el futuro el *API* nos retorna algún request indicando que el usuario no tiene permisos para acceder a algún área (código *HTTP 401*) y el *token* de refrescado dejó de ser válido, las credenciales se borran del *local storage* y redirigimos al cliente a la pantalla de inicio.

Ofrecemos también la posibilidad de no guardar la sesión, lo cual hará que una vez cerrada la pestaña, el usuario deba volver a ingresar al sistema para poder acceder a los distintos servicios.

#### 4.3.1.4. Internacionalización

Así como las credenciales, las preferencias de internacionalización son guardadas en el *local storage* del cliente web, de manera tal que los usuarios del sistema no deban elegir el idioma de su preferencia cada vez que ingresan a la página.

### 4.3.2. API

Implementada utilizando el lenguaje de programación *Python*. El principal motivador de esta elección fue que las librerías que más se acercan al estado del arte en *Machine Learning* se encuentran implementadas en este lenguaje. Provee del servicio al cual acceden los usuarios, tales como la plataforma web de entrenamiento mediante la implementación de un *API REST*.

#### 4.3.2.1. Tecnologías

##### 4.3.2.1.1. Flask

Como base del proyecto utilizamos un *microframework* de aplicaciones web llamado *Flask* ("Flask," 2019) ya que por diseño utiliza pocos recursos. Por defecto no incluye ninguna solución para comunicarse con una base de datos, ni lógica para validar formularios.

##### 4.3.2.1.2. flask-smorest

*Framework* que enriquece a *Flask* agregando la posibilidad de poder crear *APIs REST* generando una especificación de *OpenAPI* que es expuesta a los usuarios del servicio en dos variantes de documentación: *ReDoc* y *Swagger UI*. Tener un *API* que se auto-documenta nos permite ofrecerle a los clientes un lugar en el cual puedan consultar el funcionamiento completo de el servicio requiriendo poco trabajo extra a la hora de desarrollar el servicio.

A su vez, provee facilidades de serialización y de-serialización de objetos en respuestas *JSON* utilizando la librería de *marshmallow* así como también utilizadas para simplificar la creación de *endpoints* que permitan paginar sus resultados (particularmente útil para listar entidades como entrenamientos, usuarios o textos).

#### 4.3.2.1.3. *flask-jwt-extended*

*Flask JWT extended* es una librería que agrega a *Flask* la funcionalidad de poder restringir el acceso a ciertos *endpoints* requiriendo que el usuario tenga un *JSON Web Token* (Jones, Bradley, Sakimura, NRI, & Microsoft, 2015) que acredite su identidad. A su vez, dado que cada *token* se asocia a una identidad (usuario) y es capaz de contener información, podemos utilizar el mismo sistema para guardar los roles del usuario y requerir que cuente con los permisos necesarios.

#### 4.3.2.1.4. *Mongoengine*

*Mongoengine* es una librería pensada para trabajar con *MongoDB* en *Python*. En su esencia es un *Document-Object Mapper* que permite convertir resultados de consultas de la base de datos en objetos de *Python* así como también permite guardar esos objetos en la base de datos.

#### 4.3.2.1.5. *marshmallow-mongoengine*

Librería que nos permite integrar la librería de serialización *Marshmallow* con *Mongoengine* simplificando el manejo de objetos entre la base de datos y los objetos que se serializan o deserializan en la capa del *API*. Es de especial utilidad la posibilidad de excluir campos a serializar ya que permite retornar únicamente lo que es de interés para el usuario, como por ejemplo sucede al retornar un objeto *User* ya nunca deberíamos devolver en una respuesta el campo *password* de la base de datos.

### 4.3.3. Worker

El proceso de *worker* fue implementado en *Python* utilizando la librería *Celery*.

Por diseño, los *workers* siempre tienen un modelo de inferencia cargado en memoria, por lo que cuando se crea uno nuevo se le asigna por defecto el *snapshot* actual (*snapshot\_id* = 0) para poder ayudar a distribuir la carga de los pedidos al *API*.

Debido a que requieren de un *snapshot* para poder realizar la tarea de *NER* así como también requieren del *corpus* anotado para poder entrenar el modelo, cada *worker* debe tener acceso a la base de datos principal de *NERd* (Figura 35).

Cada *worker* puede realizar las siguientes tareas:

- **train**: Entrenar el modelo de inferencia de *spaCy* utilizando el *snapshot* para el cual fue configurado.
- **un\_train**: Borra el modelo guardado en el disco, efectivamente «des-entrenándose». Principalmente utilizado para liberar recursos.

- **reload:** Recarga el modelo de inferencia desde el disco. Utilizado cuando algún *worker* entrenó el modelo y el actual quedó desactualizado.
- **nlp:** Realiza la tarea de *NER* utilizando el modelo cargado.
- **change\_snapshot:** Cambia el *snapshot* utilizado en el *worker*.

#### 4.3.4. Task Scheduler

El *task scheduler* que utilizamos en el *API* para poder orquestar todas las tareas relacionadas con *NER* así como también la de la administración de los *workers* es *Celery* ("Celery," 2019). *Celery* es una cola de tareas asincrónica especializada en procesamiento distribuido. Si bien soporta distintos *backends* para la funcionalidad de cola de mensajes, decidimos utilizar *RabbitMQ* ("RabbitMQ," 2019) debido a su robustez y la posibilidad de escalar horizontalmente.

Para el monitoreo de las tareas de *Celery* utilizamos la herramienta llamada *Flower* (Movsisyan, 2016), que permite consultar el estado de cada *worker* (Figura 39), historial de tareas (Figura 40) así como también estadísticas en tiempo real sobre el sistema (Figura 41) y detalles de todos los *workers* que están conectados a la cola de mensajes (Figura 41).

The screenshot shows the Flower dashboard interface. At the top, there is a navigation bar with 'Flower' and 'Dashboard' (selected), and links for 'Tasks', 'Broker', and 'Monitor'. On the right, there are links for 'Logout', 'Docs', and 'Code'. Below the navigation bar, a summary row displays statistics: Active: 0, Processed: 50, Failed: 0, Succeeded: 50, and Retried: 0. A search bar is located above the main table. The table lists two workers, both with an 'Online' status. The first worker, 'celery@077ee094fbcc', has 0 active tasks, 27 processed, 0 failed, 27 succeeded, 0 retried, and a load average of 0.93, 0.92, 0.5. The second worker, 'celery@4ff15b7741a0', has 0 active tasks, 23 processed, 0 failed, 23 succeeded, 0 retried, and a load average of 0.93, 0.92, 0.5. At the bottom left of the table, it says 'Showing 1 to 2 of 2 entries'.

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@077ee094fbcc	Online	0	27	0	27	0	0.93, 0.92, 0.5
celery@4ff15b7741a0	Online	0	23	0	23	0	0.93, 0.92, 0.5

Figura 39: Página principal de Flower.

Name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
nerd.tasks.corpus.nlp	cc6fb396-0d45-48fc-ba6f-57016955107f	SUCCESS	[Alerta sanitaria en Esquel por hantavirus: ya son tres las víctimas fatales]	{}	['text': 'Alerta sanitaria en Esquel por hantavirus: ya son tres las víctimas fatales', 'ents': [[{'start': 20, 'end': 26, 'label': 'LOC'}], 'sents': [[{'start': 0, 'end': 75}], 'tokens': [{'id': 0, 'start': 0, 'end': 6, 'pos': 'PROPN', 'tag': 'PROPN', 'dep': 'ROOT', 'head': 0}, {'id': 1, 'start': 7, 'end': 16, 'pos': 'ADJ', 'tag': 'ADJ', 'dep': 'ADV', 'head': 0}, {'id': 2, 'start': 17, 'end': 19, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 0}, {'id': 3, 'start': 20, 'end': 26, 'pos': 'PROPN', 'tag': 'PROPN', 'dep': 'case', 'head': 3}, {'id': 4, 'start': 27, 'end': 30, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 0}, {'id': 5, 'start': 31, 'end': 41, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 5}, {'id': 6, 'start': 42, 'end': 45, 'pos': 'PUNCT', 'tag': 'PUNCT', 'dep': 'punct', 'head': 9}, {'id': 7, 'start': 46, 'end': 45, 'pos': 'ADV', 'tag': 'ADV', 'dep': 'advmod', 'head': 9}, {'id': 8, 'start': 46, 'end': 45, 'pos': 'ADV', 'tag': 'ADV', 'dep': 'advmod', 'head': 9}]]]	2019-11-22 02:23:54.867	2019-11-22 02:23:54.867	21.341	celery@077ee094fbcc
nerd.tasks.corpus.nlp	6f8af3dc0-d4cd-40e9-ae1f-c1101082a62	SUCCESS	[Alerta por tormentas intensas para el sur de Córdoba]	{}	['text': 'Alerta por tormentas intensas para el sur de Córdoba', 'ents': [[{'start': 45, 'end': 52, 'label': 'LOC'}], 'sents': [[{'start': 0, 'end': 52}], 'tokens': [{'id': 0, 'start': 0, 'end': 6, 'pos': 'VERB', 'tag': 'VERB', 'dep': 'ROOT', 'head': 0}, {'id': 1, 'start': 7, 'end': 10, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 0}, {'id': 2, 'start': 11, 'end': 20, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 2}, {'id': 3, 'start': 21, 'end': 29, 'pos': 'ADJ', 'tag': 'ADJ', 'dep': 'ADJ', 'head': 2}, {'id': 4, 'start': 30, 'end': 34, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 2}, {'id': 5, 'start': 35, 'end': 37, 'pos': 'DET', 'tag': 'DET', 'dep': 'DET', 'head': 6}, {'id': 6, 'start': 38, 'end': 41, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 6}, {'id': 7, 'start': 42, 'end': 45, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 6}]]]	2019-11-22 02:24:16.337	2019-11-22 02:24:16.338	19.069	celery@4f15b7741a0
nerd.tasks.corpus.nlp	99e996a3-ea79-445c-8623-199e1141297	SUCCESS	[Alerta por tormentas fuertes para el centro del país]	{}	['text': 'Alerta por tormentas fuertes para el centro del país', 'ents': [], 'sents': [[{'start': 0, 'end': 52}], 'tokens': [{'id': 0, 'start': 0, 'end': 6, 'pos': 'VERB', 'tag': 'VERB', 'dep': 'ROOT', 'head': 0}, {'id': 1, 'start': 7, 'end': 10, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 0}, {'id': 2, 'start': 11, 'end': 20, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 2}, {'id': 3, 'start': 21, 'end': 28, 'pos': 'ADJ', 'tag': 'ADJ', 'dep': 'ADJ', 'head': 2}, {'id': 4, 'start': 29, 'end': 33, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 2}, {'id': 5, 'start': 34, 'end': 36, 'pos': 'DET', 'tag': 'DET', 'dep': 'DET', 'head': 6}, {'id': 6, 'start': 37, 'end': 43, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 6}, {'id': 7, 'start': 44, 'end': 47, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 2}]]]	2019-11-22 02:24:34.825	2019-11-22 02:24:34.825	0.019	celery@077ee094fbcc
			[Alerta por fuerte		['text': 'Alerta por fuerte temporal y posible caída de granizo en el Gran Mendoza', 'ents': [[{'start': 60, 'end': 72, 'label': 'LOC'}], 'sents': [[{'start': 0, 'end': 72}], 'tokens': [{'id': 0, 'start': 0, 'end': 6, 'pos': 'VERB', 'tag': 'VERB', 'dep': 'ROOT', 'head': 0}, {'id': 1, 'start': 7, 'end': 10, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 0}, {'id': 2, 'start': 11, 'end': 20, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 2}, {'id': 3, 'start': 21, 'end': 28, 'pos': 'ADJ', 'tag': 'ADJ', 'dep': 'ADJ', 'head': 2}, {'id': 4, 'start': 29, 'end': 33, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 2}, {'id': 5, 'start': 34, 'end': 36, 'pos': 'DET', 'tag': 'DET', 'dep': 'DET', 'head': 6}, {'id': 6, 'start': 37, 'end': 43, 'pos': 'NOUN', 'tag': 'NOUN', 'dep': 'case', 'head': 6}, {'id': 7, 'start': 44, 'end': 47, 'pos': 'ADP', 'tag': 'ADP', 'dep': 'ADP', 'head': 2}]]]				

Figura 40: Historial de tareas en Flower.

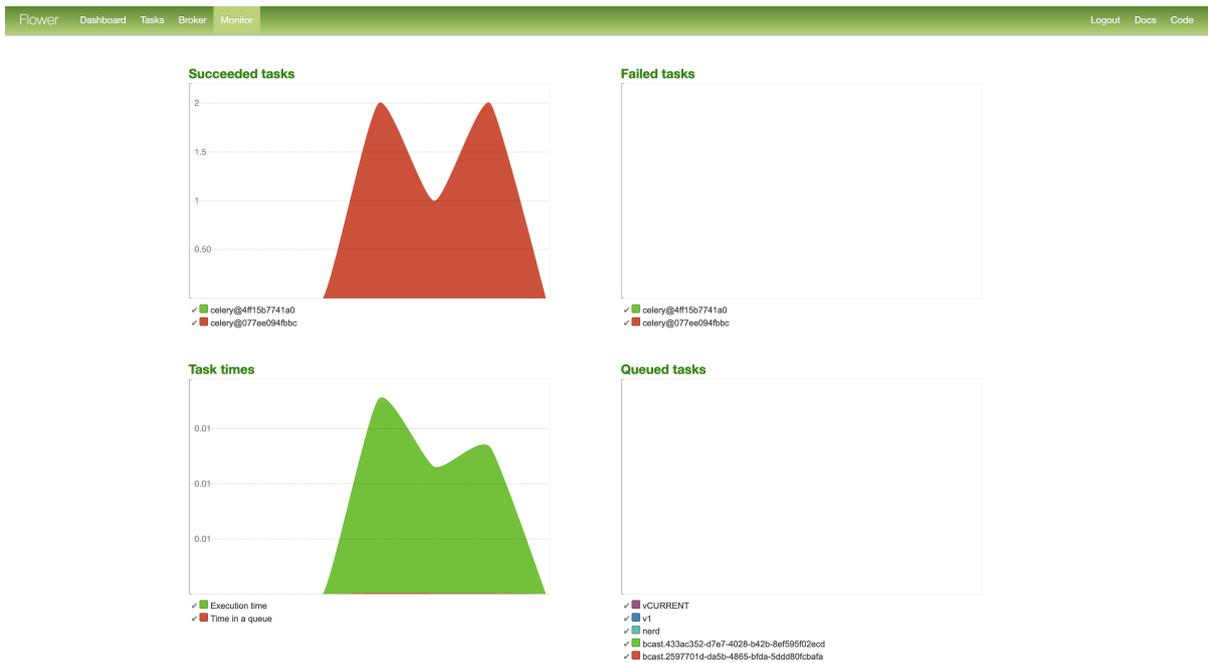


Figura 41: Monitoreo en tiempo real en Flower.

Queues					
Name	Messages	Unacked	Ready	Consumers	Idle since
bcast.2597701d-da5b-4865-bfda-5ddd80fcbafa	0	0	0	1	2019-11-21 23:00:01
bcast.433ac352-d7e7-4028-b42b-8ef595f02ecd	0	0	0	1	2019-11-21 23:00:01
nerd	0	0	0	2	2019-11-21 23:00:01
v1	0	0	0	1	2019-11-22 7:02:58
vCURRENT	0	0	0	1	2019-11-22 7:02:58

Figura 42: Listado de colas activas en Flower.

Name	Exclusive	Durable	Routing key	No ACK	Alias	Queue arguments	Binding arguments	Auto delete	
nerd	False	True	nerd	False	None	None	None	False	Cancel Consumer
bcast.2597701d-da5b-4865-bfda-5ddd80fcbafa	False	True	nerd	False	nerd_broadcast	None	None	True	Cancel Consumer
vCURRENT	False	True	vCURRENT	False	None	None	None	False	Cancel Consumer

Figura 43: Colas de mensaje escuchadas por un worker.

### 4.3.5. Base de Datos

Debido a que en el proyecto necesitamos guardar mayormente documentos (entrenamientos), optamos por utilizar una base de datos orientada a documentos: *MongoDB* ("MongoDB," 2019).

*MongoDB* nos permite tener un lugar en donde guardar todos los documentos que puede escalar tanto como sea necesario debido a su posibilidad de fragmentar<sup>2</sup> o replicar<sup>3</sup> información entre distintas instancias que pueden estar ubicadas en distintos servidores.

#### 4.3.5.1. Modelado

Para el modelado de los datos utilizamos cuatro colecciones mostradas en la figura 44.

<sup>2</sup><https://docs.mongodb.com/manual/sharding/>

<sup>3</sup><https://docs.mongodb.com/manual/replication/>

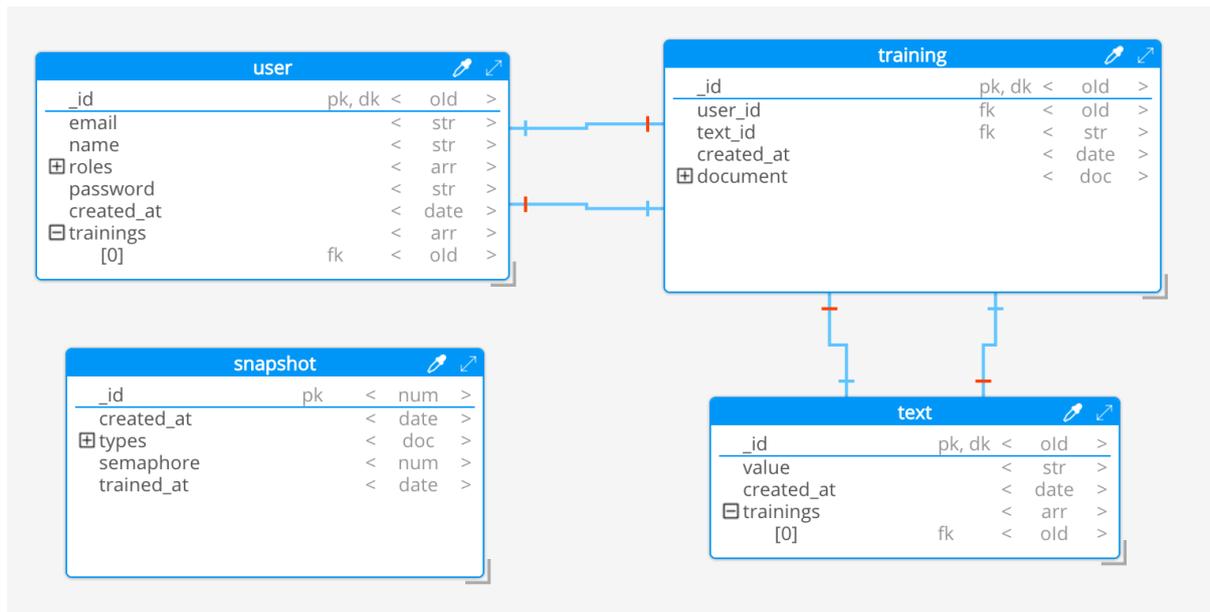


Figura 44: Colecciones de la base de datos.

#### 4.3.5.1.1. User

La colección *user* contiene la información de todos los usuarios del sistema (Figura 45). Entre los datos del usuario está el campo *trainings*, que guarda internamente una lista de *ObjectId* pertenecientes a los entrenamientos del usuario. A su vez está la lista de roles a los cuales tiene acceso el usuario.

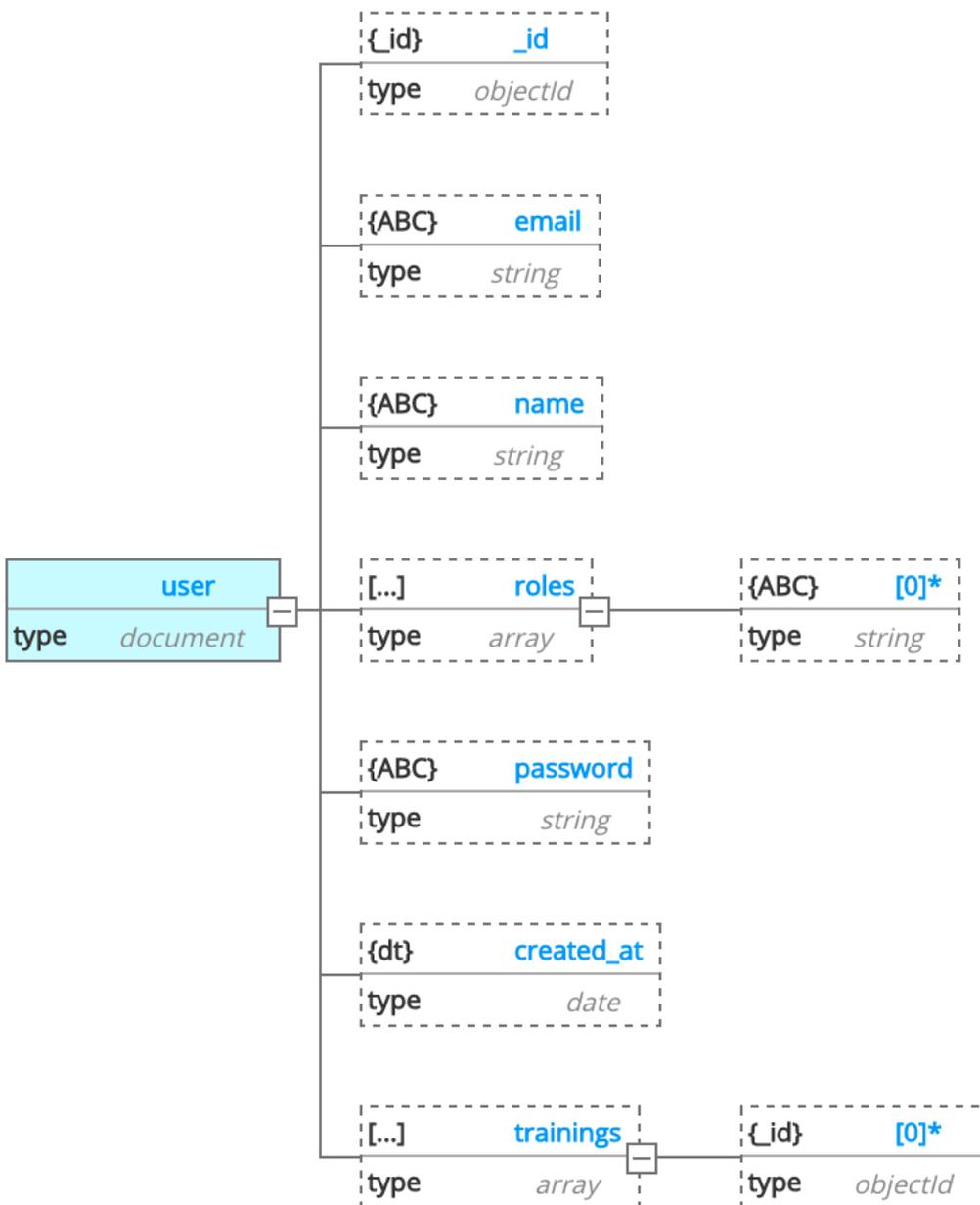


Figura 45: Colección de usuarios.

#### 4.3.5.1.2. Text

La colección *text* contiene todos los textos que forman parte del corpus que son cargados para ser mostrados en la interfaz de entrenamiento (Figura 46). Así como *user*, cada documento de *text* contiene una lista de referencias a los entrenamientos para ese texto.

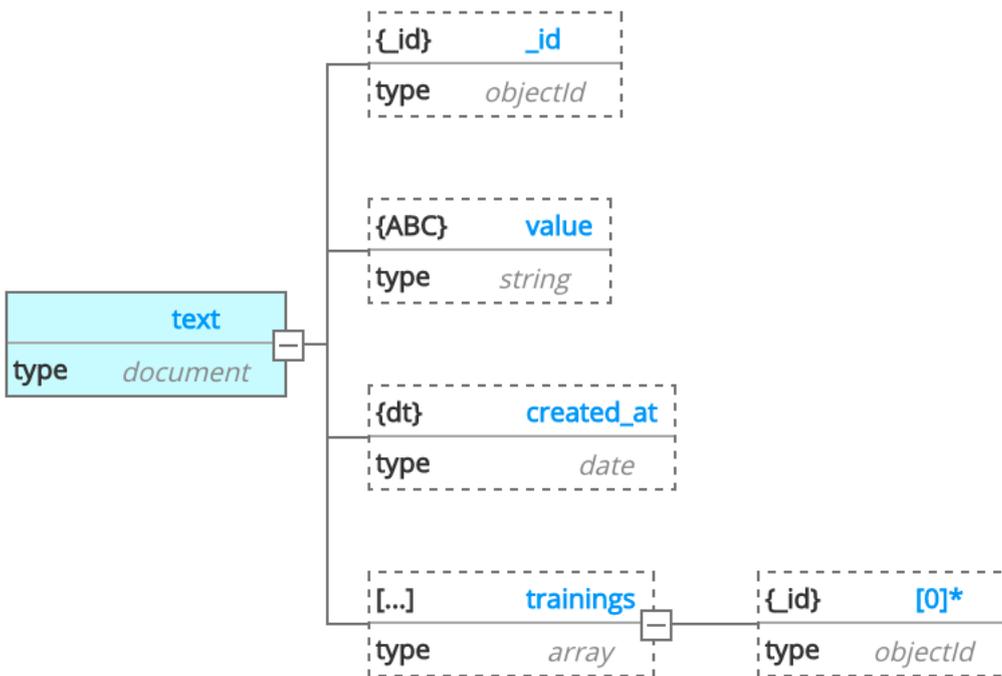


Figura 46: Colección de textos.

#### 4.3.5.1.3. Snapshot

La colección *snapshot* contiene información sobre los distintos *snapshots* creados en el sistema (Figura 47). De particular interés es el atributo *types*, que contiene un mapa de todos los tipos de entidades disponibles en el *snapshot*. También contiene el atributo *semaphore*, que sirve como un campo de control para poder organizar a los *workers* que están utilizando ese *snapshot* en particular.

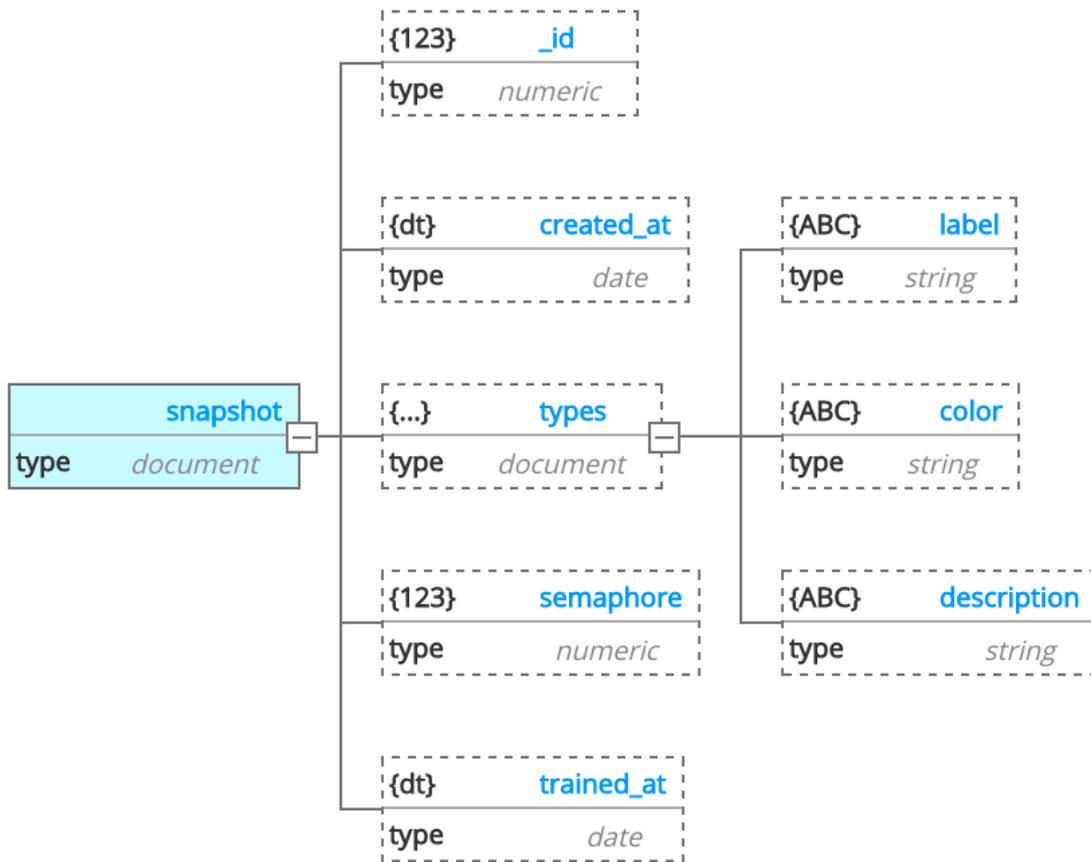


Figura 47: Colección de snapshots.

#### 4.3.5.1.4. Training

La colección *training* contiene información sobre todos los entrenamientos cargados en el sistema (Figura 48). Cada entrenamiento contiene referencia al texto al que pertenece así como también al usuario que lo creó. De manera tal de poder obtener la información de los documentos relacionados sin necesidad de duplicar información.

Dentro de un documento *training*, se encuentra el atributo *document*, que es el que internamente contiene la representación de un documento de entrenamiento que es entendido por *spaCy* y que luego será utilizado para entrenar el modelo de inferencia.

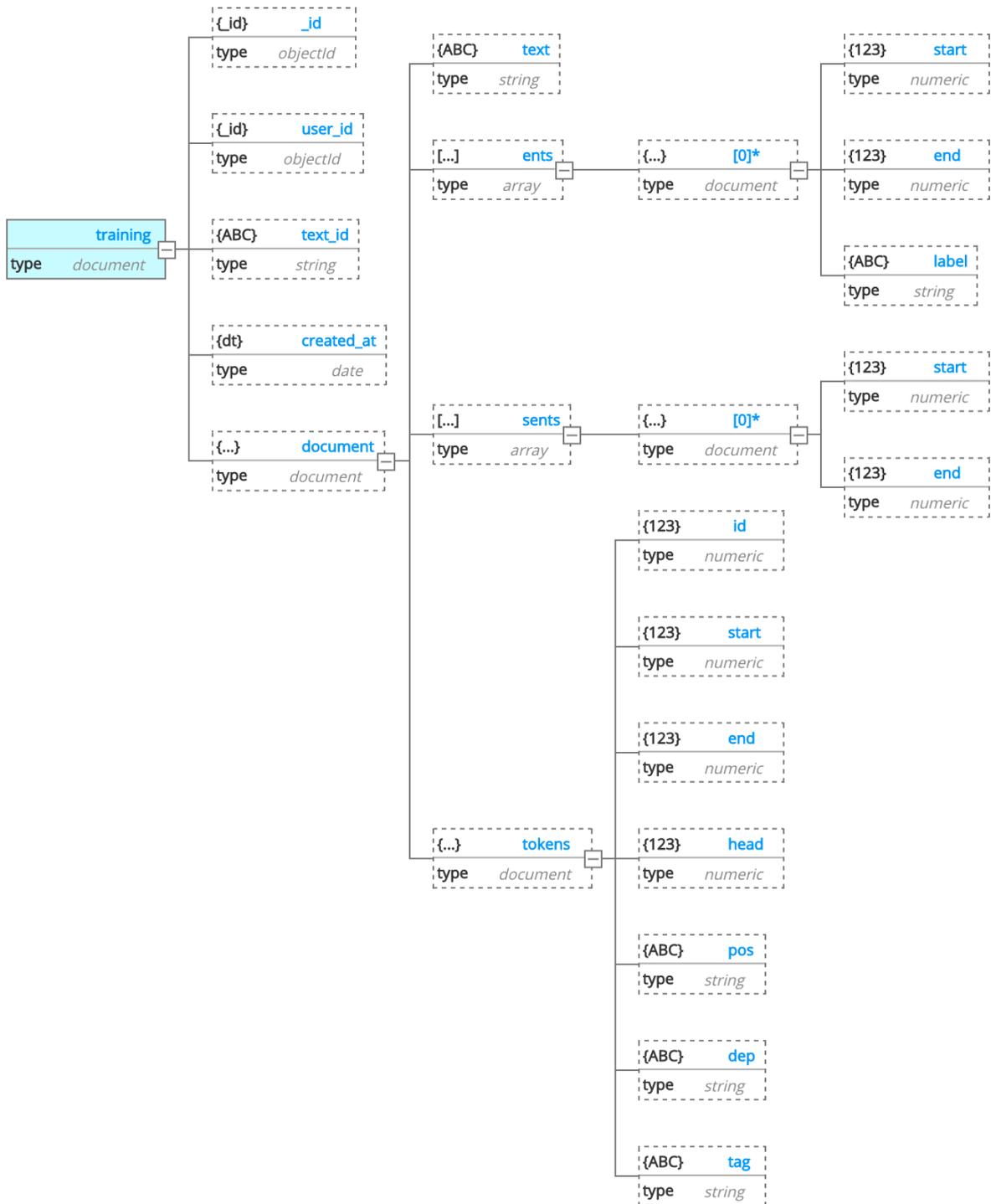


Figura 48: Colección de entrenamientos.

#### 4.4. Vista física

La vista física representa el sistema desde el punto de vista de un ingeniero de sistemas. Se refiere a la topología de los componentes de software en la capa física, así como a las conexiones físicas entre estos componentes. Esta vista también se conoce como

la vista de *deployment*.

#### 4.4.1. Docker

*Docker* ("Docker," 2019) es un conjunto de productos que permiten empaquetar aplicaciones en contenedores. Cada contenedor contiene todo lo necesario para que la aplicación funcione (código, dependencias, configuraciones, herramientas del sistema, etc.)<sup>4</sup>. Mediante el uso de virtualización, es posible distribuir y ejecutar un contenedor en más de un sistema, lo que lleva a la posibilidad de tener varias réplicas de un mismo servicio ejecutando al mismo tiempo.

Por ésta razón decidimos utilizar *Docker* como la herramienta para empaquetar los servicios principales de *NERd*.

La implementación completa de *NERd* está formada por 5 contenedores requeridos y uno opcional:

- **ui**: servidor *web* de la página de administración y entrenamiento.
- **app**: servidor de aplicaciones *web* de la *API*.
- **worker**: servicio de *NER*. (escalable)
- **rabbitmq**: cola de mensajes.
- **flower**: servicio de monitoreo de *RabbitMQ* (opcional).
- **mongodb**: base de datos.

#### 4.4.2. Deployment

##### 4.4.2.1. Docker images

Para realizar un *deployment* de nuevas versiones de cada contenedor de *docker*, existen comandos dentro del archivo *Makefile*:

```
make build # Construye nuevas imágenes para cada uno de los servicios
make release-push # Publica las nuevas imágenes a github
```

Las imágenes base que utilizamos en todos los servicios son mantenidas por *Bitnami* ("Bitnami," 2019). Tomamos esta decisión para: 1. Reducir el tamaño de las imágenes al reutilizar la capa de sistema operativo entre todas ellas. 2. Se actualizan automáticamente con todos los parches de seguridad de las plataformas elegidas. 3. Están implementadas siguiendo las mejores prácticas acerca del no uso de *root* en *containers*.

##### 4.4.2.2. Service

---

<sup>4</sup><https://www.docker.com/resources/what-container>

Para poder crear un entorno completo de *NERd*, utilizamos la herramienta llamada `docker-compose` ("Docker compose," 2019), con la cual definimos cada uno de los servicios, las conexiones de red y los volúmenes de disco necesarias entre cada contenedor de manera tal que las conexiones mostradas en la figura 35 funcionen correctamente.

A su vez, dentro de un archivo *Makefile* del proyecto definimos un conjunto de comandos que sirven para crear un entorno de desarrollo de *NERd*.

```
make setup # Inicializa todos los contenedores de NERd, creando 2 workers
make down # Destruye todos los contenedores de NERd
make stop # Apaga los contenedores de NERd (sirve liberar recursos en una máquina)
make start # Enciende los contenedores parados de NERd
```

Para realizar un *deploy* de la aplicación en un entorno productivo utilizaremos de ejemplo utilizando el servidor provisto por el *ITBA*.

Primero es necesario contar con una configuración de *SSH* que defina el *host pf-nerd*:

Host pampero

```
Hostname pampero.itba.edu.ar
User <ITBA_USER>
```

Host pf-nerd

```
Hostname <MACHINE_IP>
User <MACHINE_USER>
# Requerimos un proxy para acceder a pampero
ProxyCommand ssh pampero nc %h %p
ControlMaster auto
ControlPath ~/.ssh/sockets/%r@%h-%p
ControlPersist 600
```

Con esa configuración en el archivo `~/.ssh/config` podemos ejecutar el comando para realizar el *setup* de una instancia con las configuraciones de producción.

```
make prod-setup
```

### 4.4.3. Servicios

#### 4.4.3.1. Cliente *web*

Dentro del *container ui*, incluimos *nginx* ("Nginx," 2019) como servidor *web* de la plataforma. El mismo tiene dos responsabilidades:

1. Servir el contenido estático de la SPA.
2. Redirigir los *requests* de las urls que comienzan con `/api/` al container de API.

#### 4.4.3.2. API

Debido a que el API consume sus datos de *MongoDB* así como también requiere de una conexión mediante *Celery* a *RabbitMQ* para poder realizar las tareas de *NER*, es posible tener varias instancias del mismo de manera tal que cada una se conecte al cluster de los dos servicios. Es posible mediante el uso de un *Load Balancer* generar un cluster de servicios de API.

#### 4.4.3.3. Worker

Dado que cada *worker* está implementado como un *Celery worker*, mientras la configuración del mismo sea correcta, al levantar nuevos containers será posible escalarlo horizontalmente. Incluido en el repositorio de *NERd* está un comando de *Makefile* que permite escalar el servicio worker:

```
make scale-up # Crea un nuevo worker localmente
make scale-down # Destruye un worker localmente
```

#### 4.4.3.4. RabbitMQ cluster

*RabbitMQ* está pensado desde su concepción para poder escalar horizontalmente<sup>5</sup>, por lo que para tener varias instancias de cola de mensajes será necesario corregir la configuración del servicio.

#### 4.4.3.5. MongoDB cluster

Al igual que *RabbitMQ*, *MongoDB* fue diseñado para escalar horizontalmente (ver Base de datos), por lo que para poder escalar horizontalmente la base de datos será necesario modificar la configuración del contenedor para que forme parte de un cluster.

## 4.5. Escenarios

La descripción de una arquitectura se ilustra utilizando un pequeño conjunto de casos de uso, o escenarios, que se convierten en una quinta vista. Los escenarios describen secuencias de interacciones entre objetos y entre procesos. Se utilizan para identificar elementos arquitectónicos y para ilustrar y validar el diseño de la arquitectura. También sirven como punto de partida para las pruebas de un prototipo de arquitectura. Esta vista también se conoce como vista de caso de uso.

---

<sup>5</sup><https://www.rabbitmq.com/clustering.html>

#### 4.5.1. Inferencia de entidades utilizando el API

Como podemos ver en la figura 49, para realizar el reconocimiento de entidades nombradas, deben intervenir cuatro actores.

Cuando un *cliente* realiza un pedido al *API*, crea una nueva tarea en la cola correspondiente al snapshot especificado por la versión en el *request* en *RabbitMQ*. Luego, un único *worker* que esté escuchando la cola para esa versión va a tomar la tarea, realizar la tarea de inferencia y asignarle un resultado a esa tarea. Finalmente, *RabbitMQ* notifica al *API* de la tarea y este puede devolver finalmente el resultado al *cliente*.

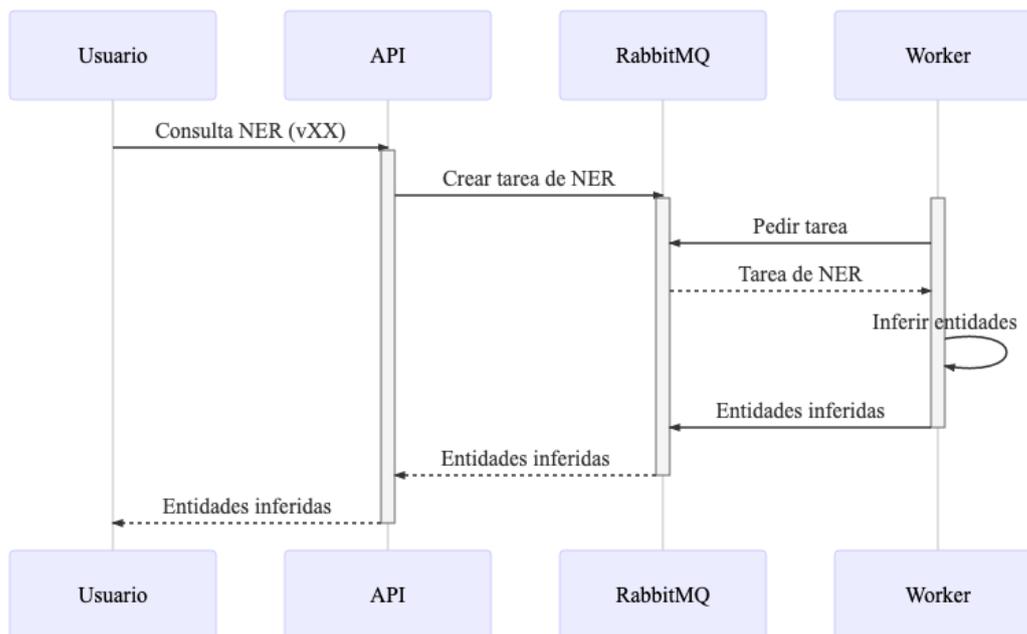


Figura 49: Flujo de NER.

#### 4.5.2. Corrección de entidades utilizando el entrenador

Para realizar una corrección, el usuario deberá realizar los siguientes pasos:

- Ingresar al sistema con credenciales que tengan el permiso para entrenar.
- Hacer *click* en el botón de **ENTRENAR**.
- Esperar a que cargue el texto.
- Revisar las entidades y corregir como sea necesario.
- Hacer *click* en el botón **ACEPTAR**.

Como podemos ver en la figura 50, para poder mostrar un texto a entrenar son necesarios cinco servicios.

El proceso de la inferencia es el mismo que el mostrado en la figura 49, agregándose la consulta a *MongoDB* al principio para poder obtener un texto que aún no haya sido entrenado por el usuario

logueado.

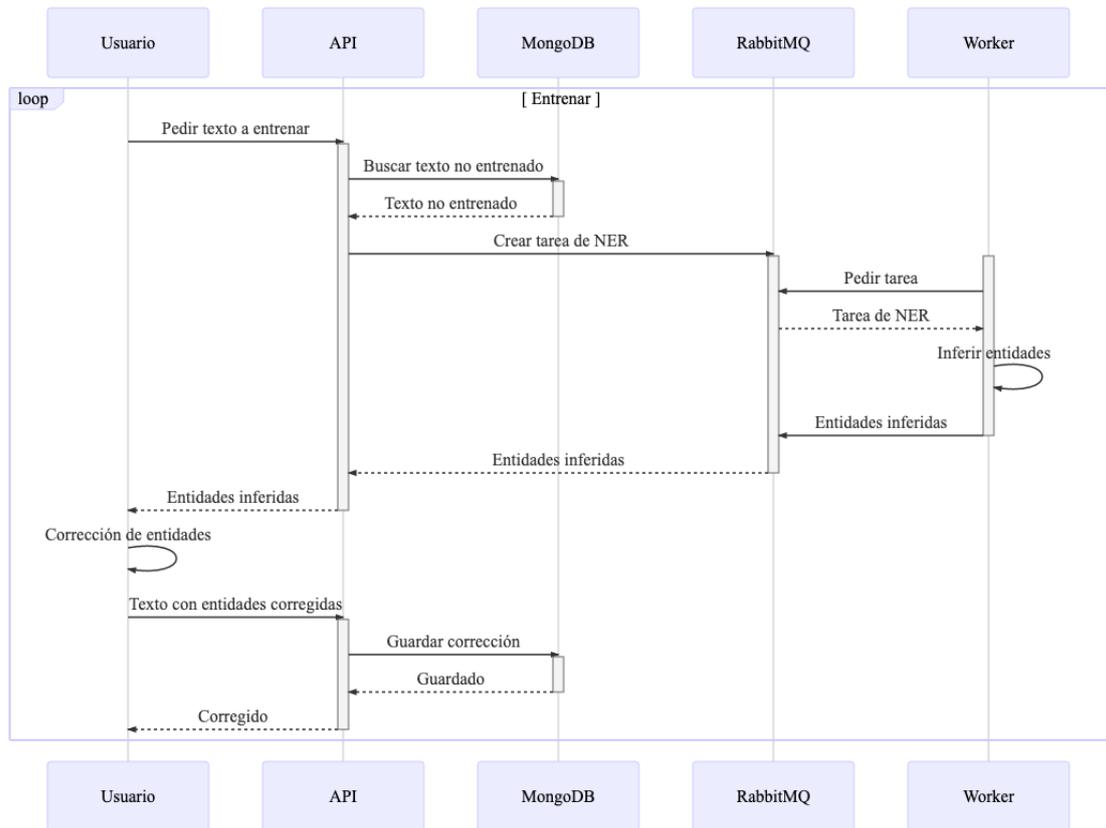


Figura 50: Pedido de texto para entrenamiento.

### 4.5.3. Habilitar snapshots

Cuando un usuario necesita comparar dos snapshots, precisa tener al menos un *worker* de cada versión levantado. En la figura 51, ante una tarea de reasignación de snapshot, el *worker* se desuscribe de la cola de mensajes del *snapshot* que está escuchando actualmente y se suscribe a la cola de mensajes del *snapshot* al que se lo quiere cambiar. Luego, se recarga el modelo.

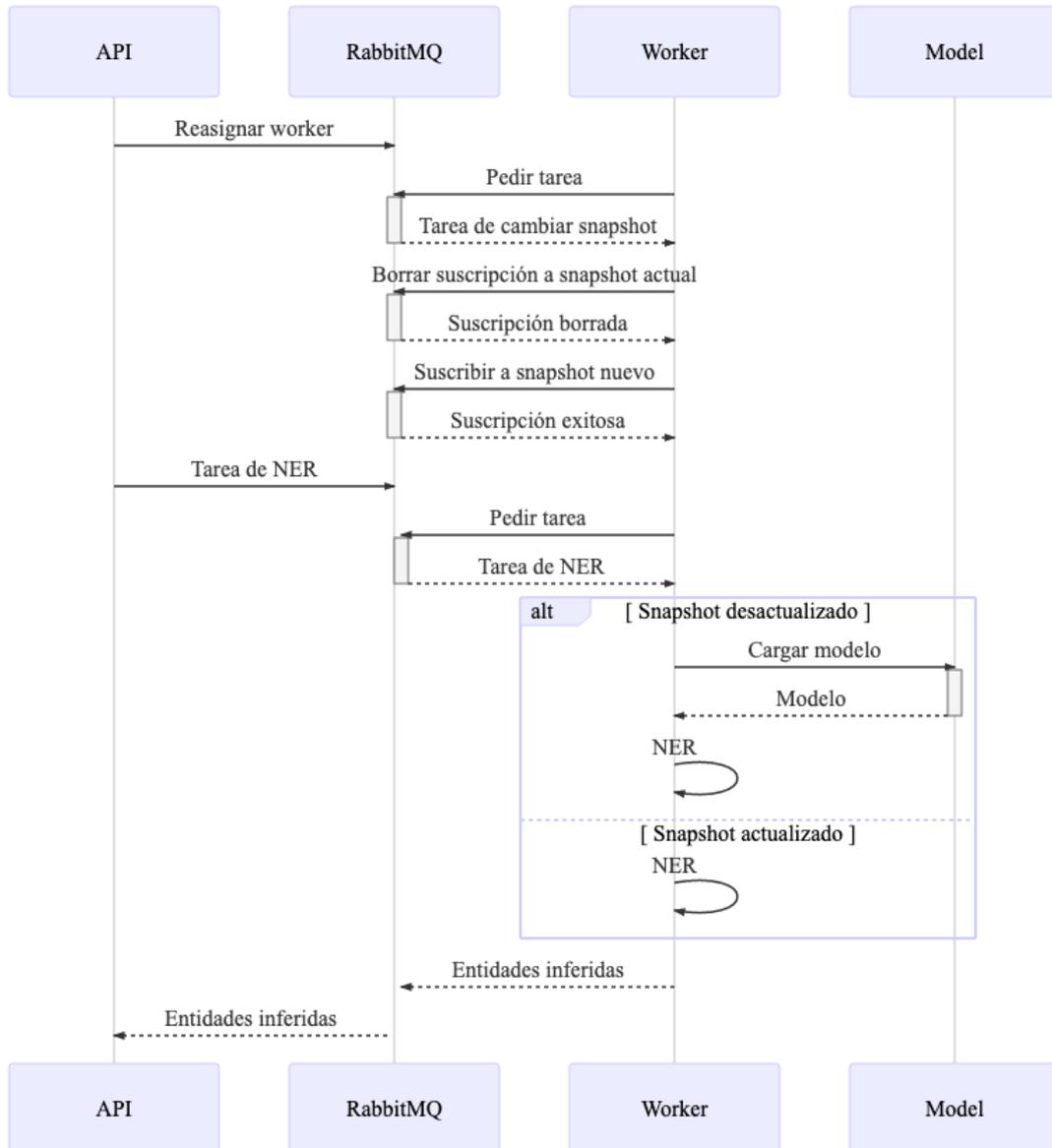


Figura 51: Cambio de modelo.

## 5. Resultados de entrenamiento

### 5.1. Métrica: Valor-F

En análisis estadístico de clasificación binaria, el valor-F es una medida de la **exactitud** de una prueba. Considera tanto la **precisión** (del inglés *precision*) como la **exhaustividad** (del inglés *recall*). Su fórmula es la siguiente:

$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

En particular cuando  $F_1$  ( $\beta = 1$ ) es la media armónica de la precisión y la exhaustividad, donde un puntaje  $F_1$  alcanza su mejor valor en 1 (precisión y *recall* perfecta) y el peor en 0.

$$F_1 = \left( \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Donde

- *precision* es el número de resultados positivos correctos dividido por el número de todos los resultados positivos devueltos por el clasificador. En particular para nuestro uso es el número de entidades nombradas que coinciden exactamente con el conjunto de evaluación. Por ejemplo, cuando se predice [Persona Horacio] [Persona Gómez], pero lo correcto era [Persona Horacio Gómez], la precisión es cero. La precisión es después promediada por cada una de la entidades nombradas.

$$\text{precision} = \frac{|\{\text{resultados positivos}\} \cap \{\text{resultados correctos}\}|}{|\{\text{resultados positivos}\}|}$$

- *recall* es el número de resultados positivos correctos dividido por el número todas las muestras que deberían haberse identificado como positivas. En nuestro uso es el número de entidades del conjunto de evaluación que aparecen exactamente en la misma posición en las predicciones.

$$\text{recall} = \frac{|\{\text{resultados positivos}\} \cap \{\text{resultados correctos}\}|}{|\{\text{resultados correctos}\}|}$$

La métrica de valor-F es ampliamente utilizada en la literatura del procesamiento de lenguaje natural *NLP* (Hand & Christen, 2018); y en especial en el ámbito del reconocimiento de entidades nombradas *NER*.

Por esa razón hemos decidido utilizar la métrica  $F_1$  para poder comparar nuestra **exactitud** con los

resultados obtenidos por los entrenamientos de otras publicaciones y sistemas.

## 5.2. Resultados obtenidos

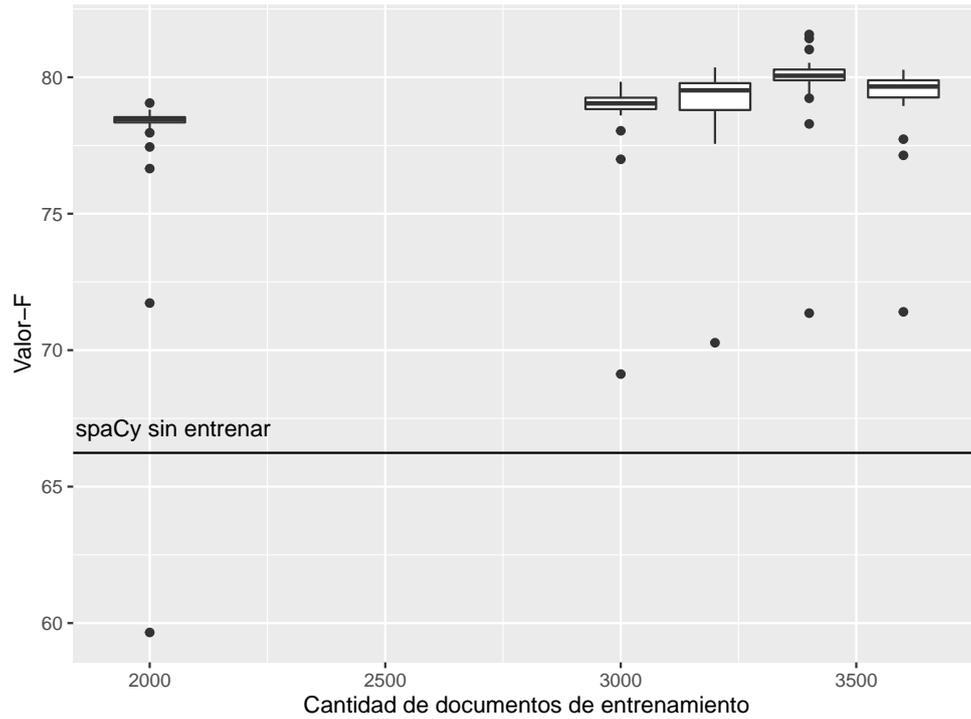
La recomendación de *spaCy* para entrenamientos de *NER* es de al menos anotar 2000 textos, por lo que en este trabajo hemos etiquetado utilizando nuestra plataforma un número suficiente como para poder utilizar en el cálculo de resultados una **muestra aleatoria** de 4000 valores.

En los entrenamientos por refuerzo es necesario tomar la decisión de cuantos textos anotados serán utilizados para entrenar y cuantos serán utilizados para evaluar el entrenamiento.

Decidimos realizar diferentes pruebas y combinaciones de cantidad de documentos de entrenamiento y evaluación para analizar el valor-F del modelo resultante:

- Entrenamiento: 0 % (0 documentos) y evaluación: 0 % (4000). Este escenario equivale a utilizar el modelo base de *spaCy*.
- Entrenamiento: 50 % (2000 documentos) y evaluación: 50 % (2000).
- Entrenamiento: 75 % (3000 documentos) y evaluación: 25 % (1000).
- Entrenamiento: 80 % (3200 documentos) y evaluación: 20 % (800).
- Entrenamiento: 85 % (3400 documentos) y evaluación: 15 % (600).
- Entrenamiento: 90 % (3600 documentos) y evaluación: 10 % (400).

En la figura 52 se puede ver que nuestro sistema alcanza una exactitud en valor-F de aproximadamente 80 puntos con relativamente pocos documentos.



**Figura 52:** Boxplot del Valor-F vs cantidad de documentos de entrenamiento

Sobre la cantidad de documentos necesarios para el entrenamiento vs evaluación podemos concluir que una relación 85 % / 15 % es la más acertada para un conjunto de 4000 documentos.

## 6. Discusión

### 6.1. Analisis de resultados

Los resultados indicados en la sección 5.2 muestran que el valor-F inicial de *spaCy* para nuestro *corpus* de noticias tiene una *score* de 66 puntos. Esta puntuación dista mucho del valor 89 que indica la librería en su documentación (Honnibal & Montani, 2019) para este idioma.

La razón de esta discrepancia ha sido mencionada en varias oportunidades a lo largo de este trabajo; el modelo de *spaCy* ha sido entrenado con un *corpus* de documentos diferente del que nosotros como usuarios de la librería tenemos interés. En particular el modelo inicial de *spaCy* ha sido entrenado con la siguiente información:

- **Spanish UD AnCora Corpus.** Contiene más de 14mil documentos de entrenamiento provenientes de noticias españolas (Zeman & Martínez Alonso, 2019).
- **WikiNER.** Proveniente de más de 50,000 documentos de wikipedia en español (Nothman, Ringland, Radford, Murphy, & Curran, 2017).

Este es un claro ejemplo de la fragilidad que poseen los modelos estadísticos en cuanto al *corpus* de interés. El modelo de *spaCy* ha sido entrenado con noticias de España, sin embargo, esas noticias no alcanzan para que el modelo pueda generalizar a las noticias de Argentina y mantener el alto grado de exactitud del modelo anterior.

En este trabajo entrenando sobre el modelo de *spaCy* con 3600 documentos hemos alcanzado subir ese valor-F de 66 a 80 puntos. Con una fracción de los documentos que *spaCy* utilizó en su entrenamiento base podemos ajustar cualquier *corpus* de documentos.

### 6.2. Tipos de entidades

La detección de entidades por defecto en español de *spaCy*, incluye las entidades *Persona*, *Organización*, *Locación* y *Misceláneo*. Cómo nuestro sistema soporta el entrenamiento de nuevas entidades tomamos la decisión de incluir un quinto tipo de entidad el de *Fecha*.

El hecho de que exista una clasificación de entidad como *Misceláneo* indica que hay otro tipo de entidades que no están siendo clasificadas específicamente y entran en esta categoría genérica. En la sección 2.1.1.1 se define formalmente que es una entidad, por lo que se puede inferir que toda entidad nombrada que cumpla la definición formal y no posea un tipo de entidad específicamente definido entra en la categoría de entidad *Miscelanea*.

Existe literatura acerca de la posible taxonomía de las entidades. En (Brunstein, 2002) se proponen 29 tipos y 64 subtipos.

Esta discusión es producto de la necesidad de agregar nuevos tipos que experimentamos al finalizar el entrenamiento de las 4000 noticias. Sin embargo lo propuesto en (Brunstein, 2002) no cubrió nuestras expectativas. Finalmente encontramos una taxonomía lo suficientemente completa para cubrir nuestras necesidades (Sekine, 2007) que puede ser resumida en la figura 53.

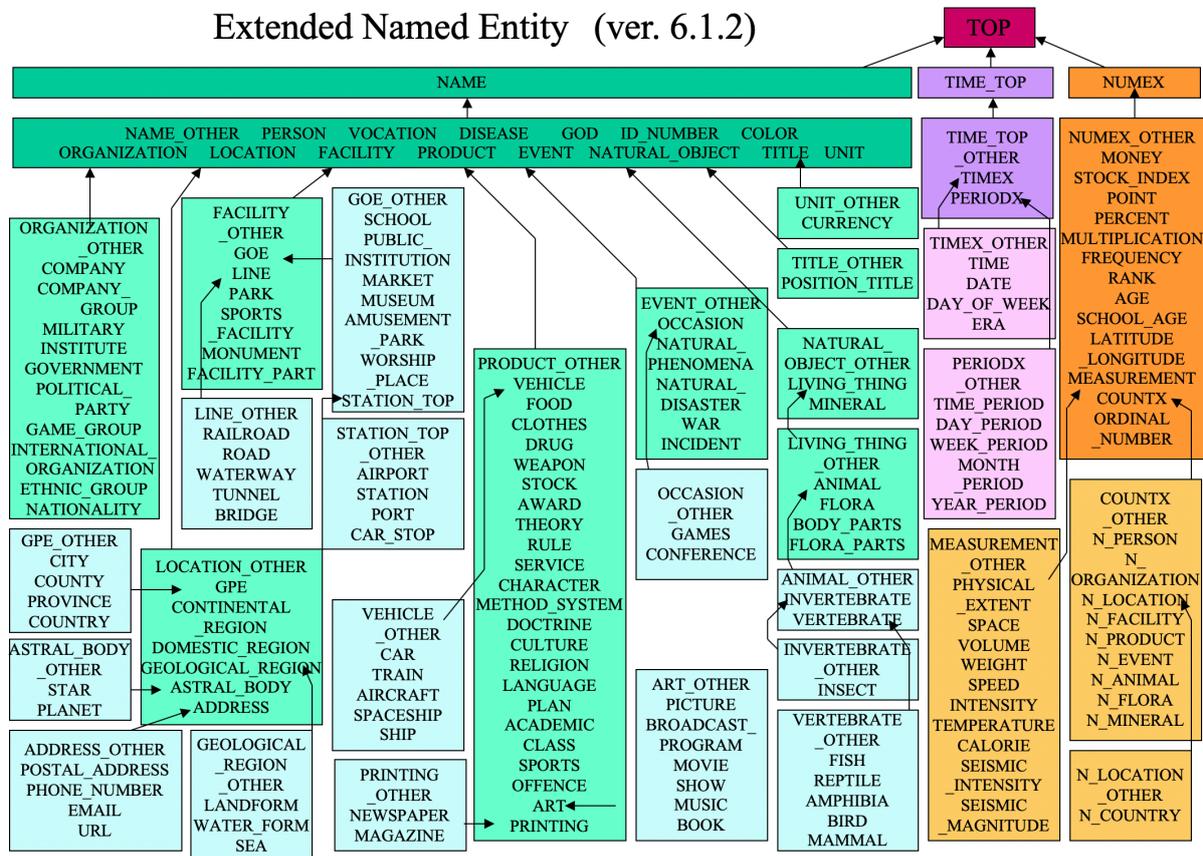


Figura 53: Taxonomía extendida de Sekine.

Haciendo un análisis de dicha taxonomía encontramos que nuestro *corpus* puede beneficiarse de los siguientes nuevos tipos de entidad:

- **Political\_Party**: diferentes actores políticos.
  - ej: peronistas, kirchneristas, macristas.
- **Offence**: ofensas.
  - ej: asesinato, abducción, fraude, aborto, robo.
- **Facility\_Name**: lugares específicos.
  - ej: Ecoparque, Disneyland, Canal de Suez, Aeropuerto de Ezeiza.
- **Product\_Name**: nombres de productos.
  - ej: Galaxy S10, Fiat Toro, Manaos.
- **Game**: nombres de juegos.
  - ej: futbol, football, basket, fortnite, ajedrez.

### 6.3. Semilla inicial

Nuestro sistema ofrece al usuario una forma sencilla y ágil de entrenar modelos estadísticos y sumar nuevos tipos de entidad. Sin embargo, entrenar un nuevo tipo de entidad suele ser una tarea inicialmente lenta y difícil dado que se necesita un número mínimo de anotaciones para que la nueva entidad empiece a tener relevancia dentro del modelo y sea inferida.

Por ejemplo, suponga se quiere agregar el nuevo tipo de entidad «Droga» al modelo. Podemos estar horas etiquetando entidades hasta cruzarnos con la primera instancia de «marihuana» y luego varias anotaciones más hasta cruzarnos con «cocaína». Además, como el sistema no conoce previamente el significado abstracto de «Droga» dichas *keywords* no van a ser identificadas previamente por el anotador

Una posible solución para evitar esta «curva de aprendizaje» es la de permitir al usuario definir una «semilla» de palabras clave (o *keywords*) a la hora de definir un tipo nuevo:

Droga: Heroína, Cocaína, Barbitúricos, Metadona, Alcohol, Ketamina, Benzodiazopines, Anfetaminas, Tabaco, Cannabis, etc.

De esta manera y gracias al proceso de *word-embeddings* visto en la sección 3.5.2 el modelo puede pre-entrenarse para identificar las palabras definidas y todas aquellas que tengan un significado similar en el espacio vectorial de significados.

### 6.4. Entrenamiento *live vs offline*

Existen dos maneras de entrenar un modelo de predicción NLP:

- *Online* o *Live*: cada nueva etiqueta que entrenamos es inmediatamente inyectada al modelo de predicciones.
- *Offline*: cada nueva etiqueta que entrenamos es guardada para ser entrenada en otro momento.

En nuestro sistemas hemos elegido la segunda opción porque al poseer la capacidad de escalar horizontalmente la cantidad de *workers*; el modelo se carga en varios servicios y de ser entrenado de manera *live* el entrenamiento iría a *workers* diferentes resultando en inconsistencias entre ellos. Existen maneras de migrar nuestro sistema a un modelo *online* y mantener la capacidad de escalar horizontalmente, pero escapa al alcance que decidimos darle a este trabajo por lo cual lo consideramos una propuesta de mejora a futuro.

Uno de los mayores beneficios de tener un modelo online, además del trivial de que esta constantemente actualizado con las últimas mejoras; es el de la capacidad de apuntar a los *tokens* que poseen una mayor incertidumbre. En la sección 3.6 hemos mencionado que puede ocurrir que algunas palabras tengan una incerteza acerca de qué etiqueta debe ser predecida. En particular

este tipo de técnicas están dentro de la literatura conocida como «*Uncertainty sampling*» que se puede resumir en buscar entidades que tengan un *score* 0,5 y dirigir el entrenamiento a esos *tokens* primero.

## 6.5. Linkeo de entidades con *Knowledge Base*

Existe otra tarea en el universo de *NLP* que está muy relacionada a la tarea de *NER*: es la tarea de *Named Entity Linking* (“*Entity linking*,” 2019).

Un corolario de la definición de **designador rígido** en la sección 2.1.1.1 es que una misma entidad puede tener varios designadores. Por ejemplo «Lionel Messi» también puede ser referido como «Leo Messi», «Leo», «La Pulga», «Lionel Andrés “Leo” Messi», etc. Todos ellos son designadores rígidos de una misma entidad, del concepto de Messi. A este concepto abstracto de entidad se le puede asignar un identificador único dentro de una base de datos de conocimiento (*knowledge base* en inglés) para luego ser utilizado en la normalización de entidades.

Ya existen *knowledge bases* universales en internet. Una de las más importante y completas es **wiki-data**. El identificador de Messi es el Q615 (“*Wikidata - lionel messi*,” 2019).

Si al etiquetar una entidad, además de decirle a nuestro sistema de qué tipo es, sumamos la funcionalidad de indicar a que identificador de entidad corresponde, se podría luego entrenar un nuevo modelo de *NEL*, separado del modelo predictor de entidades, cuya tarea exclusiva sea la de normalizar y predecir identificadores de entidades.

El sistema puede beneficiarse ya que tendría disponible, mediante *Wikidata*, de una gran cantidad información adicional acerca de la entidad incluyendo, fotos, alias, palabras clave, etc como muestra la figura 54.

The screenshot shows the Wikidata page for Lionel Messi (Q615). The page is titled "Lionel Messi (Q615)" and includes a navigation bar with "Item", "Discussion", "Read", "View history", and "Search Wikidata". The main content area displays the item's name, its description as an "Argentine association football player", and a list of alternative names in various languages. Below this, there is a section for "Statements" with a table showing the instance of Messi as a "human" with one reference. A media gallery is also present, featuring an image of Messi from the 2018 Nigeria match with a media legend and a point in time of 2018. On the right side, there is a "Wikipedia" section with a list of 140 entries for the item in various languages.

Figura 54: Wikidata - Lionel Messi.

Desde el punto de vista del usuario, puede beneficiarse porque obtener una representación normalizada de las entidades de un texto es una tarea importante en la summarización de las mismas. Por ejemplo en la creación de «nubes de palabras» a partir de muchos artículos, puede ocurrir que en algunos artículos hablen de «Mauricio» y en otros se hable de «Macri»; sin embargo sería necesario unificar estos dos keywords para que el ítem tenga una relevancia mayor. Se puede observar una nube de palabras generada utilizando entidades encontradas por nuestro sistema en la figura 55.

Personajes del día - 2019-11-14 17:49:55



Figura 55: Con NEL se sumarían Messi con Lionel y Mauricio con Macri.

Para finalizar, un dato curioso es que a esta tarea de reconocimiento de entidades más desambiguación de las mismas se lo suele llamar *NERD* (*named-entity recognition and disambiguation*) al igual que nuestro proyecto *NERd*; una coincidencia accidental pero que evidencia la necesidad a futuro de implementar esta mejora.

## 6.6. Utilidad de la herramienta

Para poner a prueba nuestra herramienta **NERd** en un entorno real participamos de la hackatón en MediaParty 2019.

(“Hackaton,” 2019) es un evento de tres días en Argentina, que reúne a 2500 emprendedores, periodistas, programadores de software y diseñadores de cinco continentes para trabajar juntos para el futuro de los medios de comunicación. Nacido de Hacks/Hackers Buenos Aires, el evento fusiona a grandes empresas como New York Times, The Guardian, Vox, ProPublica, Watchup, Neo4J o DocumentCloud y comunidades regionales de la mayor red de periodistas y desarrolladores del mundo.

Participamos en conjunto con otro proyecto final en el que van a utilizar nuestra API para hacer detección de entidades en documentos PDF.

La experiencia fue muy satisfactoria, recibimos buenas críticas sobre la usabilidad de nuestra aplicación y la gran utilidad que presta a la comunidad.

Por tal motivo recibimos el primer premio de dicha hackaton ("Mención itba," 2019)

## 7. Conclusiones

Utilizando *spaCy* para la inferencia de entidades pudimos alcanzar una exactitud aceptable de 80 puntos de valor-F con 4000 documentos anotados.

Recomendamos a quienes quieran entrenar un modelo *NER* que realicen una etapa inicial de anotación con al menos la cantidad mencionada de textos. De esta manera las inferencias del modelo serán más exactas a la hora de realizar futuros entrenamientos. Además los entrenadores tendrán la experiencia para definir qué tipos de entidades serán necesarias para etiquetar el corpus provisto.

Gracias a la implementación del proyecto *NERd* pudimos ofrecer un sistema que permita a cada comunidad generar modelos específicos para sus necesidades. A su vez, la implementación específica permite escalar al proyecto horizontalmente para poder servir tantos pedidos de *NER* como sean necesarios sin sacrificar *performance*.

El proyecto posee un *roadmap* atractivo para convertirse en una herramienta comercialmente viable y ser mejorada a futuro. La utilidad del mismo ha sido convalidada por expertos de la industria en la *MediaParty* donde fue premiada.

Creemos que ofrecer el proyecto como una herramienta de código libre, permitirá que la comunidad de *NLP* posea una poderosa herramienta para mejorar el estado del arte en cuanto a *corpus* taguados existentes.

## Anexo: Datos de resultados

- **t**: porcentaje de documentos de entrenamiento.
- **v**: porcentaje de documentos de validación.

### 7.1. t: 0% - v: 100%. *Baseline spaCy*

===== Training stats =====

Training pipeline: ner

Starting with base model 'es\_core\_news\_md'

0 training docs

4000 evaluation docs

No overlap between training and evaluation data

===== Vocab & Vectors =====

0 total words in the data (0 unique)

20000 vectors (533736 unique keys, 50 dimensions)

===== Named Entity Recognition =====

0 new labels, 0 existing labels

0 missing values (tokens with '-' label)

Good amount of examples for all labels

Examples without occurrences available for all labels

No entities consisting of or starting/ending with whitespace

===== Summary =====

5 checks passed

Training pipeline: ['ner']

Starting with base model 'es\_core\_news\_md'

Counting training words (limit=0)

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	0.000	68.147	64.433	66.238	100.000	25983	26538
2	0.000	68.147	64.433	66.238	100.000	25436	25395
3	0.000	68.147	64.433	66.238	100.000	25752	24471
4	0.000	68.147	64.433	66.238	100.000	25558	29465
5	0.000	68.147	64.433	66.238	100.000	25058	24053
6	0.000	68.147	64.433	66.238	100.000	23921	25149
7	0.000	68.147	64.433	66.238	100.000	24582	25506

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
8	0.000	68.147	64.433	66.238	100.000	29069	28158
9	0.000	68.147	64.433	66.238	100.000	28226	28367
10	0.000	68.147	64.433	66.238	100.000	23585	23769
11	0.000	68.147	64.433	66.238	100.000	24765	23512
12	0.000	68.147	64.433	66.238	100.000	23319	25613
13	0.000	68.147	64.433	66.238	100.000	23118	24513
14	0.000	68.147	64.433	66.238	100.000	21962	28556
15	0.000	68.147	64.433	66.238	100.000	24425	24294
16	0.000	68.147	64.433	66.238	100.000	27551	23843
17	0.000	68.147	64.433	66.238	100.000	25224	24975
18	0.000	68.147	64.433	66.238	100.000	24002	25027
19	0.000	68.147	64.433	66.238	100.000	24367	28260
20	0.000	68.147	64.433	66.238	100.000	25888	28075
21	0.000	68.147	64.433	66.238	100.000	23356	25695
22	0.000	68.147	64.433	66.238	100.000	28023	27458
23	0.000	68.147	64.433	66.238	100.000	27412	25016
24	0.000	68.147	64.433	66.238	100.000	24053	25072
25	0.000	68.147	64.433	66.238	100.000	24194	24086
26	0.000	68.147	64.433	66.238	100.000	24531	24725
27	0.000	68.147	64.433	66.238	100.000	24375	27681
28	0.000	68.147	64.433	66.238	100.000	28897	28069
29	0.000	68.147	64.433	66.238	100.000	24530	23338
30	0.000	68.147	64.433	66.238	100.000	23058	27724

## 7.2. t: 50% - v: 50%

===== Training stats =====

Training pipeline: ner

Starting with base model 'es\_core\_news\_md'

2000 training docs

2000 evaluation docs

1 training examples also in evaluation data

===== Vocab & Vectors =====

27595 total words in the data (6197 unique)

20000 vectors (533736 unique keys, 50 dimensions)

===== Named Entity Recognition =====

1 new label, 4 existing labels  
 0 missing values (tokens with '-' label)  
 Good amount of examples for all labels  
 Examples without occurrences available for all labels  
 No entities consisting of or starting/ending with whitespace

===== Summary =====

4 checks passed  
 1 warning  
 Training pipeline: ['ner']  
 Starting with blank model 'es'  
 Loading vector from model 'es\_core\_news\_md'  
 Counting training words (limit=0)

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	4373.230	61.762	57.692	59.658	100.000	25083	28411
2	2650.090	72.869	70.618	71.726	100.000	21033	22844
3	1879.892	77.506	75.820	76.653	100.000	22846	24881
4	1550.256	77.877	77.018	77.445	100.000	26894	22926
5	1331.972	78.330	78.058	78.194	100.000	25107	27973
6	1172.236	78.367	78.689	78.528	100.000	24348	28943
7	1021.166	79.104	79.004	79.054	100.000	22633	24151
8	891.760	78.931	78.657	78.794	100.000	23971	23087
9	825.860	78.731	78.657	78.694	100.000	28811	27209
10	763.561	78.715	78.815	78.765	100.000	23736	21812
11	680.819	78.754	78.878	78.816	100.000	22841	20951
12	579.623	78.504	78.405	78.454	100.000	21688	28305
13	494.455	78.553	78.405	78.479	100.000	23598	22923
14	469.631	78.470	78.247	78.358	100.000	21825	25855
15	421.725	78.592	78.468	78.530	100.000	19715	22731
16	410.767	78.439	78.562	78.501	100.000	24155	22225
17	411.734	78.284	78.531	78.407	100.000	21843	21950
18	347.546	78.220	78.689	78.454	100.000	22409	21145
19	353.033	78.119	78.562	78.340	100.000	21173	24639
20	260.769	78.238	78.657	78.447	100.000	22068	22087
21	306.896	78.332	78.752	78.541	100.000	22042	20555
22	248.995	78.538	78.909	78.723	100.000	20849	23016

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
23	269.101	78.300	78.720	78.510	100.000	21036	21311
24	237.393	78.228	78.499	78.363	100.000	21676	21239
25	234.766	78.439	78.562	78.501	100.000	20906	22519
26	255.452	78.612	78.562	78.587	100.000	20253	23063
27	226.127	78.432	78.531	78.481	100.000	21402	21935
28	181.469	78.327	78.499	78.413	100.000	22511	19438
29	180.268	77.904	78.026	77.965	100.000	18335	22732
30	179.335	78.023	78.121	78.072	100.000	22433	21633

### 7.3. t: 75% - v: 25%

===== Training stats =====

Training pipeline: ner

Starting with base model 'es\_core\_news\_md'

3000 training docs

1000 evaluation docs

No overlap between training and evaluation data

===== Vocab & Vectors =====

41847 total words in the data (7928 unique)

20000 vectors (533736 unique keys, 50 dimensions)

===== Named Entity Recognition =====

1 new label, 4 existing labels

0 missing values (tokens with '-' label)

Good amount of examples for all labels

Examples without occurrences available for all labels

No entities consisting of or starting/ending with whitespace

===== Summary =====

5 checks passed

Training pipeline: ['ner']

Starting with blank model 'es'

Loading vector from model 'es\_core\_news\_md'

Counting training words (limit=0)

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	5543.695	71.399	66.988	69.124	100.000	28067	25575
2	3114.421	78.567	75.483	76.994	100.000	26176	26036
3	2519.526	78.806	77.284	78.038	100.000	25530	25766
4	2120.064	79.857	78.829	79.339	100.000	25734	25163
5	1922.659	79.624	78.958	79.289	100.000	23645	25299
6	1663.921	79.807	79.858	79.833	100.000	26526	25635
7	1532.299	79.448	79.601	79.524	100.000	26120	26210
8	1311.481	79.625	79.215	79.419	100.000	25331	26163
9	1233.004	79.588	79.537	79.562	100.000	25709	25557
10	1118.481	79.486	79.537	79.511	100.000	25726	25332
11	1031.523	78.968	78.764	78.866	100.000	25399	26109
12	930.418	79.317	79.215	79.266	100.000	23443	24004
13	874.136	79.253	79.151	79.202	100.000	22272	26035
14	828.361	79.199	78.893	79.046	100.000	23957	25490
15	708.060	79.199	78.893	79.046	100.000	25963	28166
16	663.398	78.917	78.764	78.841	100.000	29668	27887
17	659.681	78.737	78.636	78.686	100.000	24656	27843
18	575.416	78.852	78.700	78.776	100.000	24306	28002
19	587.828	78.512	78.764	78.638	100.000	24038	28674
20	527.561	78.498	78.700	78.599	100.000	24642	26366
21	477.470	79.035	79.086	79.061	100.000	24744	24873
22	479.368	78.985	79.086	79.035	100.000	24633	26975
23	440.360	78.829	78.829	78.829	100.000	26582	28826
24	451.145	79.315	78.958	79.136	100.000	26343	28304
25	409.140	79.250	78.893	79.071	100.000	28135	27367
26	434.599	79.226	79.022	79.124	100.000	23172	29557
27	368.677	79.097	78.893	78.995	100.000	29263	25145
28	410.056	79.097	78.893	78.995	100.000	27527	23153
29	329.823	78.792	78.893	78.842	100.000	25775	23087
30	336.049	78.930	78.829	78.880	100.000	25315	24963

#### 7.4. t: 80% - v: 20%

===== Training stats =====

Training pipeline: ner

Starting with base model 'es\_core\_news\_md'

3200 training docs

800 evaluation docs

No overlap between training and evaluation data

===== Vocab & Vectors =====

44669 total words in the data (8237 unique)

20000 vectors (533736 unique keys, 50 dimensions)

===== Named Entity Recognition =====

1 new label, 4 existing labels

0 missing values (tokens with '-' label)

Good amount of examples for all labels

Examples without occurrences available for all labels

No entities consisting of or starting/ending with whitespace

===== Summary =====

5 checks passed

Training pipeline: ['ner']

Starting with blank model 'es'

Loading vector from model 'es\_core\_news\_md'

Counting training words (limit=0)

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	5883.516	72.743	67.964	70.273	100.000	26475	26383
2	3402.834	78.329	76.805	77.559	100.000	24611	22944
3	2753.233	78.595	78.021	78.307	100.000	24175	25389
4	2412.231	79.577	79.319	79.448	100.000	25896	23811
5	2068.848	80.130	79.805	79.967	100.000	24106	24838
6	1854.721	80.506	80.049	80.277	100.000	26573	28760
7	1602.726	80.246	79.400	79.821	100.000	25482	25610
8	1400.760	80.874	79.562	80.213	100.000	24446	25178
9	1278.537	80.542	79.562	80.049	100.000	24311	22168
10	1188.572	80.574	79.724	80.147	100.000	23460	23847
11	1057.995	80.921	79.805	80.359	100.000	25690	26429
12	982.713	79.935	79.481	79.707	100.000	24790	24607
13	911.306	79.935	79.481	79.707	100.000	23366	24298
14	872.340	79.805	79.481	79.642	100.000	25026	24863
15	744.991	79.397	79.075	79.236	100.000	23702	22732
16	724.587	79.918	79.400	79.658	100.000	25589	22711

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
17	703.524	79.577	79.319	79.448	100.000	24541	21817
18	666.757	80.016	79.238	79.625	100.000	24072	23769
19	617.329	80.230	79.319	79.772	100.000	23397	24896
20	538.848	80.180	79.400	79.788	100.000	28608	29527
21	545.278	79.951	79.238	79.593	100.000	27668	28125
22	542.003	79.657	79.075	79.365	100.000	24713	28483
23	515.150	79.085	78.508	78.795	100.000	24505	29683
24	510.062	78.694	78.183	78.438	100.000	28378	28694
25	490.366	78.694	78.183	78.438	100.000	29099	25309
26	489.131	79.003	78.427	78.714	100.000	28224	27445
27	481.775	79.038	78.589	78.813	100.000	27452	26523
28	434.892	79.055	78.670	78.862	100.000	29050	23274
29	418.859	79.218	78.832	79.024	100.000	22857	25335
30	384.182	78.763	78.508	78.635	100.000	27555	25168

## 7.5. t: 85% - v: 15%

===== Training stats =====

Training pipeline: ner

Starting with base model 'es\_core\_news\_md'

3400 training docs

600 evaluation docs

No overlap between training and evaluation data

===== Vocab & Vectors =====

47412 total words in the data (8489 unique)

20000 vectors (533736 unique keys, 50 dimensions)

===== Named Entity Recognition =====

1 new label, 4 existing labels

0 missing values (tokens with '-' label)

Good amount of examples for all labels

Examples without occurrences available for all labels

No entities consisting of or starting/ending with whitespace

===== Summary =====

5 checks passed

Training pipeline: ['ner']

Starting with blank model 'es'

Loading vector from model 'es\_core\_news\_md'

Counting training words (limit=0)

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	6149.150	72.957	69.824	71.356	100.000	25187	25398
2	3530.670	78.947	77.643	78.290	100.000	28946	30524
3	2782.778	80.537	79.295	79.911	100.000	28908	21788
4	2408.691	82.011	80.837	81.420	100.000	29169	24758
5	2083.651	81.748	81.388	81.567	100.000	28273	21302
6	1874.490	81.195	80.837	81.015	100.000	27418	24743
7	1733.739	80.154	80.066	80.110	100.000	25380	22917
8	1569.485	80.000	79.736	79.868	100.000	22546	23340
9	1404.922	80.132	79.956	80.044	100.000	25653	22216
10	1270.519	79.912	79.736	79.824	100.000	23418	23172
11	1213.184	79.868	79.956	79.912	100.000	24245	24331
12	1071.570	80.375	80.286	80.331	100.000	25123	24563
13	968.863	80.353	80.176	80.265	100.000	22790	23443
14	933.663	80.331	80.066	80.199	100.000	23651	24537
15	861.770	80.509	80.066	80.287	100.000	20112	22464
16	774.950	80.442	80.176	80.309	100.000	22588	24903
17	848.828	80.642	80.286	80.464	100.000	23963	25567
18	702.390	80.266	79.736	80.000	100.000	21939	24809
19	666.903	80.577	79.956	80.265	100.000	22971	24589
20	659.185	80.663	80.396	80.530	100.000	23862	23570
21	600.204	80.288	79.846	80.066	100.000	22904	25857
22	579.303	80.221	79.956	80.088	100.000	23899	23143
23	558.429	80.288	79.846	80.066	100.000	23462	24445
24	588.920	80.177	79.736	79.956	100.000	24245	24946
25	529.718	80.266	79.736	80.000	100.000	24724	22070
26	556.021	80.155	79.626	79.890	100.000	23382	23936
27	455.335	80.155	79.626	79.890	100.000	26675	21860
28	473.464	79.535	79.185	79.360	100.000	23786	27999
29	469.124	79.272	79.185	79.229	100.000	23863	22476
30	447.699	79.383	79.295	79.339	100.000	24570	22265

## 7.6. t: 90% - v: 10%

=====  
 ===== Training stats =====  
 =====

Training pipeline: ner  
 Starting with base model 'es\_core\_news\_md'  
 3600 training docs  
 400 evaluation docs  
 No overlap between training and evaluation data

=====  
 ===== Vocab & Vectors =====  
 =====

50207 total words in the data (8769 unique)  
 20000 vectors (533736 unique keys, 50 dimensions)

=====  
 ===== Named Entity Recognition =====  
 =====

1 new label, 4 existing labels  
 0 missing values (tokens with '-' label)  
 Good amount of examples for all labels  
 Examples without occurrences available for all labels  
 No entities consisting of or starting/ending with whitespace

=====  
 ===== Summary =====  
 =====

5 checks passed  
 Training pipeline: ['ner']  
 Starting with blank model 'es'  
 Loading vector from model 'es\_core\_news\_md'  
 Counting training words (limit=0)

Itn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
1	6337.608	73.835	69.128	71.404	100.000	24730	26330
2	3733.858	80.364	74.161	77.138	100.000	26793	23166
3	3052.388	79.718	75.839	77.730	100.000	26983	23049
4	2530.293	79.897	78.020	78.947	100.000	28707	23476
5	2252.914	81.034	78.859	79.932	100.000	25129	27980
6	2018.009	81.424	78.691	80.034	100.000	27818	27073
7	1798.934	81.739	78.859	80.273	100.000	29947	27560
8	1650.571	80.763	78.188	79.454	100.000	29033	27491
9	1484.659	81.469	78.188	79.795	100.000	25644	29321
10	1423.551	81.076	78.356	79.693	100.000	29104	28378
11	1211.619	81.109	78.523	79.795	100.000	25347	30059

---

ltn	NER Loss	NER P	NER R	NER F	Token %	CPU WPS	GPU WPS
12	1115.638	80.345	78.188	79.252	100.000	24173	28090
13	1025.299	80.936	78.356	79.625	100.000	27853	20482
14	985.754	81.109	78.523	79.795	100.000	22511	27637
15	998.443	81.501	78.356	79.897	100.000	27517	27543
16	906.401	81.436	78.020	79.692	100.000	25237	29533
17	822.168	80.944	77.685	79.281	100.000	27192	28424
18	796.657	80.977	77.852	79.384	100.000	23619	23841
19	704.390	80.803	77.685	79.213	100.000	23415	24997
20	699.090	80.729	78.020	79.352	100.000	24897	20853
21	691.900	80.763	78.188	79.454	100.000	23200	24635
22	654.682	80.696	77.852	79.249	100.000	23883	23942
23	624.907	81.250	78.523	79.863	100.000	28663	28699
24	640.695	81.391	78.523	79.932	100.000	27891	27498
25	593.737	81.786	78.356	80.034	100.000	23063	22741
26	547.825	81.469	78.188	79.795	100.000	28517	22537
27	538.115	81.786	78.356	80.034	100.000	30210	23101
28	529.483	81.501	78.356	79.897	100.000	22598	21694
29	538.596	80.977	77.852	79.384	100.000	23760	23484
30	495.002	80.522	77.685	79.078	100.000	23161	22751

---

## Glosario

Abreviación	Término
API	Application Programming Interface
CBU	Clave Bancaria Única
CDN	Content Delivery Network
CNN	Convolutional Neural Network
JWT	JSON Web Token
NEL	Named Entity Linking
NER	Named Entity Recognition
NLP	Natural Language Processing
POS	Part of speech
RB	Rule Based
REST	Representational state transfer
RNN	Recurrent Neural Network
SPA	Single Page Application
SSH	Secure Shell
WSGI	Web Server Gateway Interface

## Referencias

Bitnami. (2019). Retrieved November 22, 2019, from <https://bitnami.com/>

Brunstein, A. (2002). Annotation guidelines for answer types. Retrieved from <https://catalog.ldc.upenn.edu/docs/LDC2005T33/BBN-Types-Subtypes.html>

Case, K. E., & Fair, R. C. (1999). *Principles of economics*. Retrieved from <https://books.google.com.ar/books?id=9vvAIOBfq0kC>

Celery: Distributed task queue. (2019). Retrieved November 20, 2019, from <http://www.celeryproject.org/>

Docker. (2019). Retrieved November 20, 2019, from <https://www.docker.com/>

Docker compose. (2019). Retrieved November 20, 2019, from <https://docs.docker.com/compose/>

Duh definition. (2019). Retrieved October 14, 2019, from <https://dictionary.cambridge.org/es/diccionario/ingles/duh>

Eby, P. (2010). Python web server gateway interface. Retrieved from <https://www.python.org/dev/peps/pep-3333/>

Elliott, T. (2019). The state of the octoverse: Machine learning. Retrieved January 24, 2019, from <https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/>

Entity linking. (2019). Retrieved November 19, 2019, from [https://en.wikipedia.org/wiki/Entity\\_linking](https://en.wikipedia.org/wiki/Entity_linking)

Ethayarajh, K., Duvenaud, D., & Hirst, G. (2019). Towards understanding linear word analogies. *Proceedings of the 57th annual meeting of the association for computational linguistics*, 3253–3262. <https://doi.org/10.18653/v1/P19-1315>

Facebook. (2019). React: A javascript library for building user interfaces. Retrieved November 20, 2019, from <https://reactjs.org/>

Flask: Lightweight wsgi web application framework. (2019). Retrieved November 20, 2019, from <https://palletsprojects.com/p/flask/>

Group, I. O. W. (2017). OAuth 2.0. Retrieved October 1, 2017, from <https://oauth.net/2/>

Hackaton. (2019). Retrieved August 31, 2019, from <https://mediaparty.info/>

Hand, D., & Christen, P. (2018). A note on using the f-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3), 539–547. <https://doi.org/10.1007/s11222-017-9746-6>

Honnibal, M., & Montani, I. (2016). Practical and effective neural ner. Retrieved November 28, 2016, from [https://github.com/explosion/talks/blob/master/2016-11-28\\_The-State-of-AI-2016.pdf](https://github.com/explosion/talks/blob/master/2016-11-28_The-State-of-AI-2016.pdf)

Honnibal, M., & Montani, I. (2017). Practical and effective neural ner. Retrieved November 2, 2017, from [https://github.com/explosion/talks/blob/master/2017-11-02\\_Practical-and-Effective-Neural-NER.pdf](https://github.com/explosion/talks/blob/master/2017-11-02_Practical-and-Effective-Neural-NER.pdf)

Honnibal, M., & Montani, I. (2019). Spacy spanish model. Retrieved November 16, 2019, from <https://spacy.io/models/es>

Jones, M., Bradley, J., Sakimura, N, NRI, & Microsoft. (2015). JSON web token. Retrieved November 20, 2019, from <https://tools.ietf.org/html/rfc7519>

Kripke, S. (1980). *Naming and necessity*. Retrieved from <https://books.google.com.ar/books?id=9vvAIOBfq0kC>

Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Softw.*, 12(6), 42–50. <https://doi.org/10.1109/52.469759>

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR, abs/1603.01360*. Retrieved from <http://arxiv.org/abs/1603.01360>

Marsh, E., & Perzanowski, D. (1998). MUC-7 evaluation of IE technology: Overview of results. *Seventh message understanding conference (MUC-7): Proceedings of a conference held in fairfax, virginia*,

April 29 - may 1, 1998. Retrieved from <https://www.aclweb.org/anthology/M98-1002>

Mención itba. (2019). Retrieved October 3, 2019, from <https://www.instagram.com/p/B3Koum2peD-/>

MongoDB: The database for modern applications. (2019). Retrieved November 20, 2019, from <https://www.mongodb.com>

Movsisyan, M. (2016). Flower. Retrieved November 20, 2019, from <https://flower.readthedocs.io/>

Nginx. (2019). Retrieved November 22, 2019, from <https://www.nginx.com/>

Nothman, J., Ringland, N., Radford, W., Murphy, T., & Curran, J. R. (2017). *Learning multilingual named entity recognition from Wikipedia*. <https://doi.org/10.6084/m9.figshare.5462500.v1>

Parikh, A., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2249–2255. <https://doi.org/10.18653/v1/D16-1244>

Poibeau, T., & Kosseim, L. (2000). Proper name extraction from non-journalistic texts. *CLIN*.

Python: Thread state and the global interpreter lock. (2019). Retrieved from <https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock>

RabbitMQ. (2019). Retrieved November 20, 2019, from <https://www.rabbitmq.com/>

Sekine, S. (2007). *The definition of sekine's extended named entities*. Retrieved from [https://nlp.cs.nyu.edu/ene/version7\\_1\\_0Beng.html](https://nlp.cs.nyu.edu/ene/version7_1_0Beng.html)

Typescript. (2019). Retrieved November 22, 2019, from <https://www.typescriptlang.org/>

Wikidata - lionel messi. (2019). Retrieved November 19, 2019, from <https://www.wikidata.org/wiki/Q615>

Zeman, D., & Martínez Alonso, H. (2019). Spanish ud ancora corpus. Retrieved May 1, 2019, from [https://github.com/UniversalDependencies/UD\\_Spanish-AnCora](https://github.com/UniversalDependencies/UD_Spanish-AnCora)