



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática

JIZT

Generación de resúmenes abstractivos en
la nube mediante Inteligencia Artificial



Presentado por Diego Miguel Lozano
en Universidad de Burgos — 17 de febrero de 2021
Tutores: Dr. Carlos López Nozal y
Dr. José Francisco Díez Pastor

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	V
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	7
Apéndice B Especificación de Requisitos	15
B.1. Introducción	15
B.2. Descripción global	17
B.3. Catalogo de requisitos	18
B.4. Especificación de requisitos	21
Apéndice C Especificación de diseño	27
C.1. Introducción	27
C.2. Elección del nombre del proyecto	27
C.3. Diseño de datos	29
C.4. Diseño procedimental	38
C.5. Diseño arquitectónico	42
C.6. Diseño de interfaces	46
Apéndice D Documentación técnica de programación	51
D.1. Introducción	51

D.2. Documentación técnica del <i>backend</i>	51
D.3. Documentación técnica de la aplicación	63
D.4. Contribuir al proyecto	65
Apéndice E Documentación de usuario	69
E.1. Introducción	69
E.2. Requisitos de usuarios	69
E.3. Instalación	70
E.4. Manual del usuario	72
Apéndice F Experimentos llevados a cabo	77
Bibliografía	81

Índice de figuras

A.1.	Horas estimadas frente horas empleadas.	2
A.2.	Diagrama Gantt del proyecto.	4
A.3.	Visualización del <i>lead</i> y <i>cycle time</i>	4
A.4.	Gráfico de <i>lead</i> y <i>cycle time</i> medios.	5
A.5.	Diagrama de flujo acumulado desde el comienzo del proyecto. .	5
A.6.	Distribución de las tareas según su tipo.	6
A.7.	Análisis DAFO de JIZT.	11
A.8.	Resumen de la licencia GNU GPLv3	12
B.1.	Diagrama de casos de uso.	21
C.1.	Diagrama de clases del <i>Dispatcher</i>	31
C.2.	Modelo relacional de la base de datos.	32
C.3.	Diagrama de clases del Pre-procesador de textos.	33
C.4.	Diagrama de clases del Codificador de textos.	34
C.5.	Diagrama de clases del Generador de resúmenes.	35
C.6.	Diagrama de clases del Post-procesador de textos.	36
C.7.	Diagrama de clases de la aplicación de JIZT.	38
C.8.	Diagrama de secuencia del <i>backend</i>	40
C.9.	Diagrama de secuencia de la aplicación.	41
C.10.	Primera aproximación para la arquitectura del <i>backend</i>	44
C.11.	Diseño final de la arquitectura del <i>backend</i>	45
C.12.	Diseño final de la arquitectura de la aplicación.	45
C.13.	Ideas, ideas, y más ideas. Pero solo unas pocas buenas.	46
C.14.	JIZT - Generación de resúmenes mediante IA.	47
C.15.	Variaciones sobre el logo de JIZT.	47
C.16.	Iteraciones sobre la pantalla principal.	48
C.17.	Aplicación en Google Play.	49

E.1. Instalar JIZT desde Google Play.	71
F.1. Algoritmo de balanceo.	78

Índice de tablas

A.1. Desglose de costes fijos del proyecto.	8
A.2. Desglose de costes directos del proyecto.	9
A.3. Desglose de costes indirectos del proyecto.	9
A.4. Listado de dependencias.	13
B.1. CU-01 Solicitar resumen	21
B.1. CU-01 Solicitar resumen	22
B.2. CU-02 Establecer longitud relativa mínima del resumen	22
B.2. CU-02 Establecer longitud relativa mínima del resumen	23
B.3. CU-03 Establecer longitud relativa máxima del resumen	23
B.4. CU-04 Consultar historial de resúmenes	24
B.5. CU-05 Consultar resumen	24
B.5. CU-05 Consultar resumen	25
B.6. CU-06 Ver texto original	25
B.7. CU-07 Compartir el resumen	25
B.7. CU-07 Compartir el resumen	26
B.8. CU-08 Borrar el resumen	26
C.1. Resultados de la encuesta.	28
F.1. Experimentos con hilos de Twitter.	79
F.2. Experimentos con textos muy largos.	80

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación de todo proyecto es un proceso de gestión organizado e integrado, centrado en las actividades necesarias para asegurar una exitosa consecución del proyecto. Esta planificación contribuye a una mejor utilización de los recursos, y a un uso óptimo del tiempo asignado a un proyecto, algo crucial en todo proyecto, y especialmente relevante en el caso de un Trabajo de Fin de Grado (TFG).

En este apartado se recogen los aspectos más relevantes en cuanto a la planificación temporal de nuestro proyecto, así como su viabilidad, tanto económica como legal.

A.2. Planificación temporal

Una de las primeras decisiones que se llevaron a cabo en el marco del proyecto JIZT, fue la elección de la metodología de desarrollo *software* que adoptaríamos.

Lo primero que llevamos a cabo fue un análisis de las necesidades y limitaciones que presentaba nuestro proyecto, las cuales eran: tiempo limitado, eficiencia y velocidad, motivación y progreso del proyecto, y satisfacción de los usuarios.

Derivado de estas necesidades, se decidió que emplearíamos una metodología ágil. Las principales metodologías ágiles consideradas fueron Scrum, Kaban y Programación Extrema (XP).

Finalmente, nos decantamos por Kanban. Esta decisión estuvo en gran medida motivada por el hecho de que una planificación temporal rígida y prefijada, como ocurre por ejemplo en Scrum, era muy difícil de llevar a cabo en este proyecto dada nuestra inexperiencia con muchas de las herramientas y *frameworks* utilizados.

Kanban nos permitía un flujo continuo de trabajo, permitiendo añadir historias de usuario no contempladas inicialmente si así se consideraba apropiado.

En total se estimaron 523,15 horas, empleando finalmente 542,92:

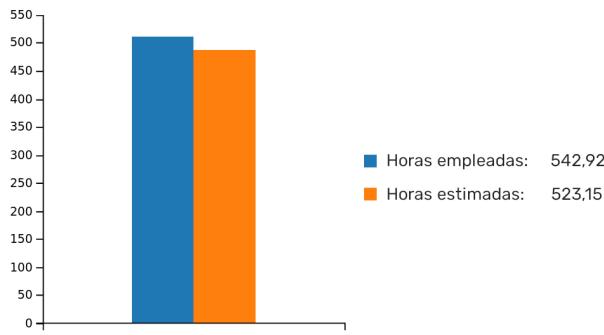


Figura A.1: Horas estimadas frente horas empleadas.

En esta planificación, se emplearon dos conceptos ágiles fundamentales:

- **Historia de usuario:** se trata de una descripción de una funcionalidad del *software* a implementar.
- ***Epics*:** agrupan historias de usuario que conformen una misma *feature*, o funcionalidad a desarrollar.

En total, se especificaron 102 historias de usuario repartidas entre 8 *epics*¹. Estos *epics* fueron, ordenados de manera cronológica:

- **Puesta en marcha:** tareas preliminares de organización y puesta en marcha del proyecto (elección de metodologías, herramientas, etc.).

¹Para la representación de las historias de usuarios y los *epics* en GitHub se emplearon *Issues* y *Milestones*, respectivamente.

- **Motor de Resumen v0.1:** implementar una primera versión del Motor de Resumen a través de los modelos preentrenados proporcionados por el módulo `transformers` de Hugging Face [1].
- **Arquitectura Microservicios v0.1:** implementar una primera versión reducida de la Arquitectura de Microservicios, configurando el componente Ingress de Kubernetes [2], y dos microservicios: el Dispatcher y el Pre-procesador de textos.
- **Arquitectura Microservicios v0.2:** continuar con la implementación de la arquitectura de microservicios, añadiendo la capacidad de realizar peticiones asíncronas y desarrollando la arquitectura dirigida por eventos. De momento, se sigue trabajando con una versión de la misma, esto es, con el Dispatcher y el Pre-procesador de textos.
- **Arquitectura Microservicios v0.3:** una vez disponemos de una versión reducida de nuestra arquitectura que funciona correctamente en local, el siguiente paso es desplegarla en Google Kubernetes Engine (GKE) [3]. Además, se deben implementar los microservicios restantes (Codificador, Motor de Resumen y Post-procesador) y la base de datos para gestionar los resúmenes.
- **Cliente v0.1:** desarrollar el cliente (aplicación) que consumirá la API y permitirá al usuario final obtener resúmenes de sus textos. Dicho cliente se implementará con ayuda de Flutter [4], por lo que en principio estará disponible en plataformas móvil, *web* y escritorio.
- **Arquitectura Microservicios v0.4:** ampliar la especificación de la API para que en las peticiones se puedan detallar todos los parámetros del resumen. Continuar con la mejora del sistema.
- **Documentación v0.1:** escribir la Memoria y los Anexos. Generar una primera versión de la documentación de la API REST y del código perteneciente a JIZT.

En la Figura A.2 se recoge un diagrama Gantt con el objetivo de facilitar la comprensión de la dimensión temporal del proyecto.

Dos conceptos importantes dentro de la metodología son *lead time* y *cycle time* [5]. Veamos qué significa cada uno de ellos.

- *Lead time:* es el período que transcurre entre la aparición de una nueva tarea en el flujo de trabajo y su salida final del sistema. Dicho de otro

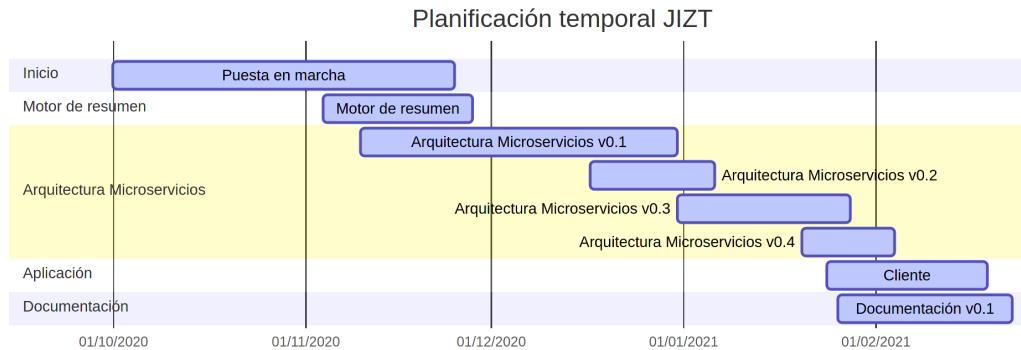


Figura A.2: El proyecto comenzó el 1 de octubre de 2020, y finalizó el 16 de febrero de 2021.

modo, es el tiempo total que el cliente está esperando la entrega de una parte del producto.

- *Cycle time*: es la cantidad de tiempo que el equipo realmente empleó en una tarea, es decir, no se cuenta el tiempo que una tarea estuvo «en espera». Por lo tanto, el tiempo del ciclo debe comenzar a medirse cuando la tarea pasa a la columna «trabajando», y no antes.

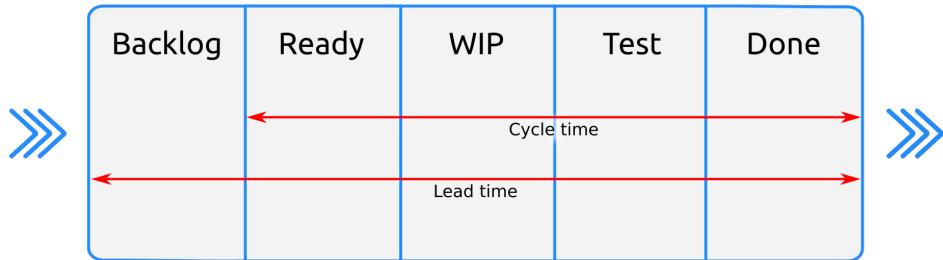
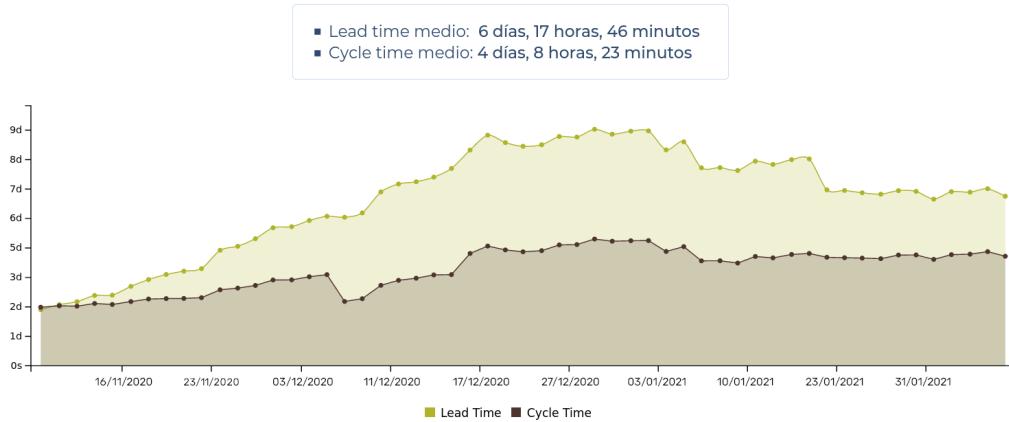


Figura A.3: Explicación gráfica del *lead* y *cycle time* sobre un tablero Kanboard.

Esta métrica nos aporta información que nos permite conocer cuánto tiempo tardaremos en entregar una determinada parte del producto. Es importante mantener el *lead* y *cycle time* tan cortos como sea posible, a fin de mantener pocas tareas «en ejecución» (WIP), permitiendo mantener un flujo constante de trabajo y aportar valor al cliente de manera frecuente.

Como vemos en la [anterior figura](#), el *lead time* medio fue de algo menos de 7 días, y el *cycle time* de 4 días y 8 horas. Como es lógico, las primeras tareas se completaron más rápido, pero según la complejidad de las mismas

Figura A.4: Gráfico de *lead* y *cycle time* medios.

fue incrementándose, también se reflejó en los tiempos. En el punto central del proyecto, se alcanzó una media de *lead time* de 9 días, aunque el *cycle time* se mantuvo por debajo de los 5, lo que indica que existía un mayor número de tareas esperando a ser atendidas.

Otro de los gráficos propios de Kanban que nos puede ofrecer información valiosa es el llamado diagrama de flujo acumulado (CFD, por sus siglas en inglés). Este gráfico muestra el número de tareas que hay en cada columna a lo largo del tiempo.

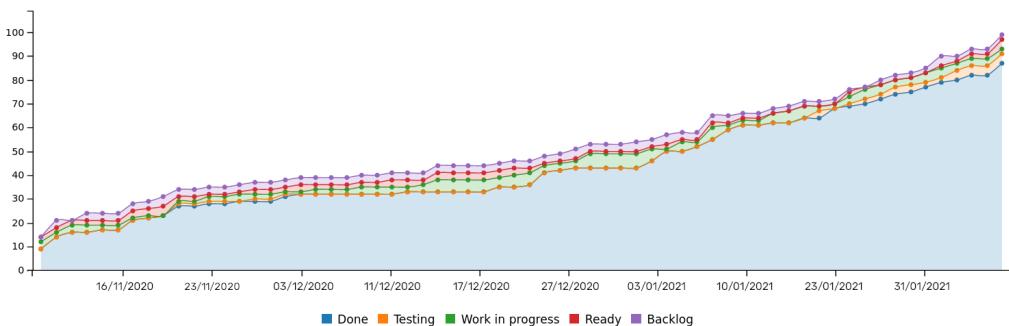


Figura A.5: Diagrama de flujo acumulado desde el comienzo del proyecto.

Como podemos apreciar en el [anterior diagrama](#), el trabajo en las diferentes columnas se distribuyó de forma correcta, no apareciendo grandes diferencias entre ellas. En este gráfico, también podemos apreciar que en la parte central del proyecto, las tareas en «*Work in progress*» fueron algo mayores que en el resto de columnas, lo cual es comprensible.

El diagrama de flujo acumulado obtenido muestra también que el ritmo de trabajo fue constante, incrementándose ligeramente hacia el final del proyecto.

Podemos visualizar también la distribución de las tareas en función de su tipo:

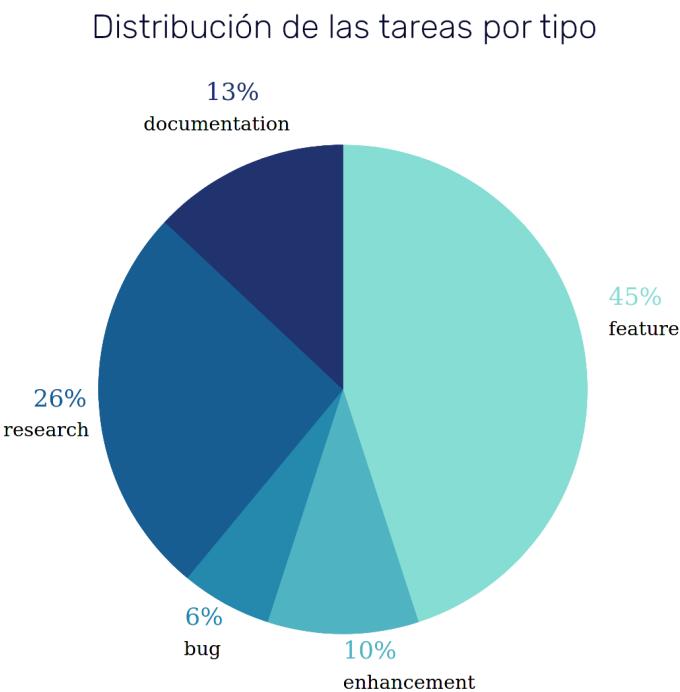


Figura A.6: Distribución de las tareas según su tipo.

Como es lógico, la mayor parte de las tareas se dedicaron a ofrecer nuevas funcionalidades (*feature*), aunque gran número de ellas se dedicaron al aprendizaje y búsqueda de información (*research*), lo cual también parece ajustarse a la realidad, puesto que como ya hemos mencionado, muchas de las herramientas y técnicas que hemos utilizado eran nuevas para nosotros.

Para finalizar esta sección, cabe mencionar que en el [repositorio del proyecto](#), y en su [tablero Kanban](#)², se puede encontrar información más detallada de cada historia de usuario y *epic*.

² Disponibles a través de <https://github.com/dmlls/jitz/milestones> y <https://kanban.jitz.it>, respectivamente.

A.3. Estudio de viabilidad

Viabilidad económica

Uno de los puntos cruciales a la hora de estudiar la viabilidad de un proyecto, y que en muchos casos determina el éxito o el fracaso del mismo, es la viabilidad económica.

En esta sección analizamos los costes y beneficios de JIZT.

Costes del proyecto

En nuestro caso, dividiremos los costes del proyecto en costes fijos, directos e indirectos.

Costes fijos

Los costes fijos son aquellos costes invariables que debemos abonar, independientemente del desarrollo del proyecto [6].

CONCEPTO	IMPORTE
Servicio GKE ¹ de Google Cloud	1449,58 €
Servicio de Internet	200,00 €
Servicio de Luz ²	225,00 €
Materiales de oficina	5,00 €
Salarios ³	9911,88 €
Salario mensual neto	1000,00 €
Retenciones por IRPF (24 %) ⁴	528,63 €
Cuotas a la Seg. Social (30,6 %) ⁵	674,01 €
Salario mensual bruto	2202,64 €
TOTAL	10341,88 €

¹ Google Kubernetes Engine [3].

² Costes calculados para 4,5 meses, con tarifa de mercado libre y potencia contratada de 3,3 kW (precio mensual medio de 50 €).

³ Costes calculados para 4,5 meses.

⁴ Según la tabla de retenciones por IRPF aplicable al ejercicio 2021 [7]

⁵ Porcentaje para autónomos según la Ley 11/2020, de 30 de diciembre, de Presupuestos Generales del Estado para el año 2021 [8].

Tabla A.1: Desglose de costes fijos del proyecto.

Costes directos

Los costes directos son aquellos costes derivados directamente del desarrollo del proyecto.

Costes indirectos

Los costes indirectos son aquellos que no dependen directamente del desarrollo del proyecto.

Costes totales del proyecto

Considerando las tres categorías de costes recogidas anteriormente, la suma de los costes totales del proyecto asciende a **13006,10 €**.

CONCEPTO	IMPORTE	IMPORTE AMORTIZ.
Costes de <i>hardware</i> ¹	2509,58 €	79,49 €
Ordenador personal	845,00 €	63,37 €
Smartphone Android	215,00 €	16,12 €
Costes de <i>software</i> ²	89,95 €	16,86 €
Adobe Illustrator	89,95 €	16,86 €
TOTAL	2599,53 €	96,35 €

¹ Se han calculado con una amortización de 5 años, habiendo sido utilizado 4,5 meses.

² Se han calculado con una amortización de 2 años, habiendo sido utilizado 4,5 meses.

Tabla A.2: Desglose de costes directos del proyecto.

CONCEPTO	IMPORTE
Dominio jizt.it	4,81 €
Cuenta de Google Play	20,76 €
Impresión de la Memoria y el cartel del TFG	40,00 €
TOTAL	65,57 €

Tabla A.3: Desglose de costes indirectos del proyecto.

Beneficios

La API REST de JIZT se ofrecerá en tres planes de suscripción diferentes.

- **Gratis:** este plan se ajusta a las necesidades de cualquier usuario regular que no vaya a realizar un uso exhaustivo del servicio. Se permiten 5.000 peticiones a la API REST, pudiendo hacerse hasta 5 peticiones por minuto. No incluye soporte técnico.
- **Estándar:** para aquellas empresas o particulares que van a realizar un uso más intensivo del servicio. Se incluyen 15.000 peticiones a la API REST, pudiendo hacerse hasta 15 peticiones por minuto. Incluye soporte técnico y de integración. El precio es de 166 €/mes.
- **Personalizado:** para aquellos usuarios cuyas necesidades no encajen en ninguno de los anteriores planes. El precio se establecerá en función de los requerimientos concretos del usuario. Por ejemplo, este plan podría incluir el despliegue del *backend* de JIZT para el cliente, bien en sus propias dependencias, o bien a través de un *cloud provider*. De

esta forma, este tipo de clientes tendrían control total sobre el *backend*, sin experimentar ninguna de las limitaciones recogidas anteriormente. El posterior mantenimiento del servicio podría estar incluido o no.

En cuanto a la aplicación, es totalmente gratuita y no contiene publicidad.

Análisis DAFO

Tras llevar a cabo un pequeño análisis de mercado, hemos identificado que las principales debilidades, amenazas, fortalezas y oportunidades de nuestro proyecto son las siguientes:

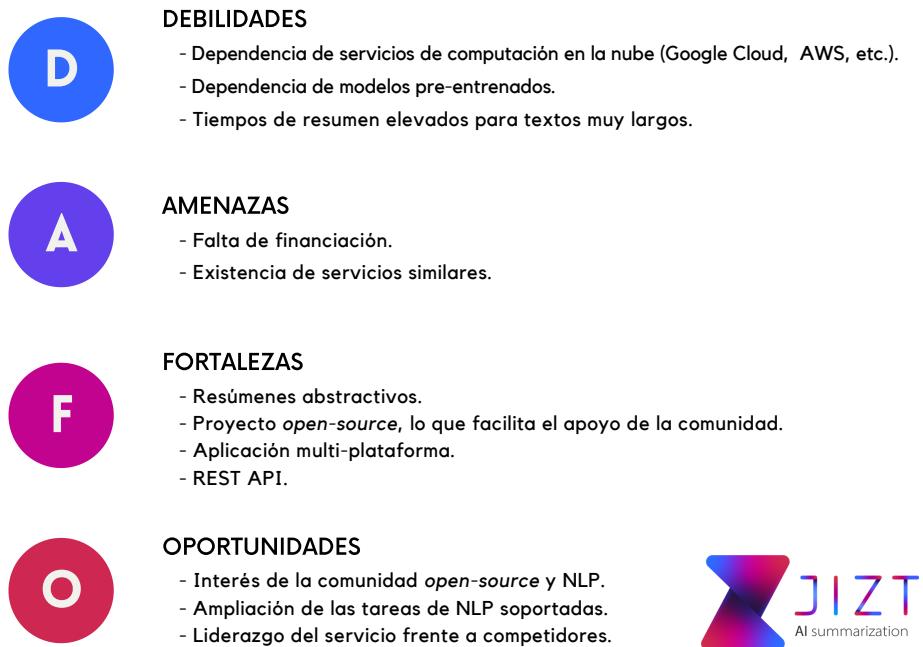


Figura A.7: Análisis DAFO de JIZT.

Viabilidad legal

Licencia del código fuente del proyecto

Desde un primer momento nuestra intención era licenciar el proyecto bajo una licencia de *Software Libre*. Dentro de este entorno, se han considerado las tres licencias más extendidas: Apache-2.0, MIT, y GPLv3.

Tras una lectura exhaustiva de las cláusulas de cada una de ellas, así como de opiniones en *blogs*, charlas, foros, etc., y tras una profunda reflexión, considerando especialmente la licencia MIT y la GPLv3, hemos tomado la decisión de que nuestro software estará licenciado bajo **GNU GPLv3** [9], cuyos puntos principales se recogen en la Figura A.8.

 dmlls/jitz is licensed under the GNU General Public License v3.0		
Permisos	Limitaciones	Condiciones
<ul style="list-style-type: none"> ✓ Uso comercial ✓ Modificación ✓ Distribución ✓ Uso de la patente ✓ Uso privado 	<ul style="list-style-type: none"> ✗ Responsabilidad ✗ Garantía 	<ul style="list-style-type: none"> ⓘ Avisos de licencia y copyright ⓘ Indicar los cambios ⓘ Publicar el código fuente ⓘ Emplear la misma licencia

Figura A.8: Resumen de la licencia GNU GPLv3. Imagen extraída y traducida de <https://github.com/dmlls/jitz/blob/main/LICENSE>.

Se puede acceder a la licencia completa a través de <https://www.gnu.org/licenses/gpl-3.0.en.html>.

Las principales razones de nuestra elección son:

- Pese a que la licencia MIT pueda parecer más permisiva en un primer lugar, ya que no obliga a que el código fuente se mantenga abierto en un futuro, creemos que a largo plazo esta «permisividad», paradójicamente, puede resultar en una limitación de sí misma. Esto es, el hecho de que ese supuesto *software* «libre» se pueda volver *software* «cerrado», lo excluye en primer lugar de esa definición de «libre», en nuestra opinión.
- Este proyecto no podría existir sin las contribuciones de *software* libre anteriores. Por ello, queremos asegurar que este proyecto siempre se mantenga abierto para poder ayudar a otros y retroalimentarse con los aportes de la comunidad.
- El simple hecho de elegir una licencia, conlleva un sinnúmero de implicaciones morales, económicas, sociales, etc., pero es algo necesario, ya que el *software* sin licencia explícita se toma por defecto como licenciado bajo *copyright*.

Listado de dependencias

Todas las dependencias del proyecto se encuentran licenciadas bajo licencias compatibles con GNU GPLv3.

A continuación, se recoge una lista detallada de las mismas:

Dependencia	Versión	Descripción	Licencia
<i>Backend</i>			
NLTK	3.5	Utilidades de NLP.	Apache v2.0
transformers	4.1.1	Modelos pre-entrenados.	Apache v2.0
truecase	0.0.12	Truecaser.	Apache v2.0
confluent-kafka	1.5.0	Cliente de Kafka para Python.	Apache v2.0
strimzi-operator	0.21.0	Kafka en Kubernetes.	Apache v2.0
postgres-operator	4.5.1	PostgreSQL en Kubernetes.	Apache v2.0
blingfire	0.1.3	Utilidades de NLP.	MIT
marshmallow	4.5.1	Serialización y deserialización.	Apache v2.0
<i>Aplicación</i>			
hive	1.4.4+1	Base de datos noSQL.	Apache v2.0
flutter_bloc	6.1.2	Patrón bloc en Flutter.	MIT
flutter_svg	0.19.2+1	Soporte para SVG.	MIT
clipboard	0.1.2+8	Portapapeles.	BSD
json_serializable	3.5.1	Serialización de JSON.	BSD
json_annotation	3.1.1	Anotaciones de JSON.	BSD
share	0.6.5+4	Soporte para compartir en diferentes plataformas.	BSD
http	0.12.2	Soporte para HTTP.	BSD

Tabla A.4: Listado de dependencias.

Licencia de la documentación del proyecto

La totalidad de la documentación de JIZT se distribuye bajo licencia GNU Free Documentation License (GFDL) [10].

Esta licencia es una adaptación al contexto de la documentación de la GNU General Public License (GPL), la cual está pensada para licenciar código fuente.

La GFDL da permiso a los lectores de copiar, redistribuir y modificar una obra (excepto las «secciones invariables») y exige que todas las copias y derivados estén disponibles bajo la misma licencia. Las copias también pueden venderse comercialmente, pero, si se producen en grandes cantidades (más de 100), el documento original o el código fuente deben ponerse a disposición del destinatario de la obra [10].

Se puede acceder a la licencia completa en <https://www.gnu.org/licenses/fdl-1.3.html>.

Licencia de la Memoria y los Anexos del proyecto

Los documentos referentes a la Memoria y a los Anexos (excepto los apéndices D: Documentación técnica de programación y E: Documentación de usuario), se encuentran licenciados bajo Creative Commons Atribución-CompartirIgual 4.0 Internacional (CC BY-SA 4.0).

Esta licencia se puede resumir en los siguientes puntos [11]:

- Cualquiera es libre de:
 - Compartir: copiar y redistribuir el material en cualquier medio o formato.
 - Adaptar: remezclar, transformar y construir a partir del material para cualquier propósito, incluso comercialmente.
- Bajo los siguientes términos:
 - Atribución: se debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerse en cualquier forma razonable, pero no de forma tal que sugiera que aquel que lo usa, o su uso tienen el apoyo del licenciante.
 - *CompartirIgual*: si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la la misma licencia del original.

Se puede acceder a la licencia completa a través de <https://creativecommons.org/licenses/by-sa/4.0/legalcode.es>.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se recoge una descripción del sistema *software* a desarrollar. Más concretamente, se presentan aspectos como el propósito, alcance y objetivos generales del mismo, así como los requisitos específicos que el sistema debe implementar.

La forma en la que hemos organizado la Especificación está basada parcialmente en las recomendaciones del estándar IEEE 830-1998, considerado como la principal referencia para la aplicación de buenas prácticas en la escritura del Documento de Especificación de Requisitos (SRS).

Propósito

El propósito de la presente Especificación de Requisitos tiene como objetivo ofrecer una descripción detallada del sistema *software* desarrollado.

Este documento está orientado principalmente al equipo de desarrollo del proyecto, pero queda también a libre disposición de cualquier persona que pudiera estar interesada.

Alcance

El sistema *software* a desarrollar recibirá el nombre de JIZT. El objetivo y funcionalidad principal de JIZT será la generación de resúmenes abstractivos en la nube. Dicho sistema estará orientado tanto a usuarios con conocimientos de informática básicos, así como usuarios con experiencia en el campo.

Para los primeros, es decir, usuarios con conocimientos básicos, se ofrecerá una aplicación multiplataforma desarrollada con la usabilidad en mente.

Para aquellos usuarios más avanzados, además de la aplicación, se proporcionará una API REST, permitiéndoles realizar peticiones directamente, e incluso desarrollar sus propias aplicaciones que consuman dicha API.

Se pondrá especial atención en el desarrollo de una documentación completa, accesible y de fácil comprensión. Esta documentación recogerá tanto los aspectos técnicos de la aplicación y la API REST, así como manuales de uso para el usuario y para el desarrollador.

Definiciones, siglas, y abreviaciones

A continuación se recogen las definiciones, siglas y abreviaciones más relevantes para el proyecto:

- **NLP**: Procesamiento de Lenguaje Natural (*Natural Language Processing*).
- **Resumen extractivo**: aquel resumen generado a partir de la *extracción* de las frases del texto consideradas más relevantes. Es decir, este tipo de resúmenes contienen únicamente frases tomadas literalmente del texto original.
- **Resumen abstractivo**: aquel resumen que incluye palabras o expresiones que no aparecen en el texto original.
- **API REST**: servicio *web* que proporciona una serie de *endpoints* para llevar a cabo operaciones HTTP (métodos), que proporcionan acceso para crear, recuperar, actualizar o eliminar los recursos del servicio.
- **Endpoint**: punto de entrada en un canal de comunicación cuando dos sistemas interactúan. En una API REST los *endpoints* son generalmente URLs.
- **HTTP**: *Hypertext Transfer Protocol*, protocolo para sistemas de información distribuidos, colaborativos e hipermedia. Este protocolo es la base de la comunicación de datos para la World Wide Web (WWW).
- **URL**: *Uniform Resource Locator*, referencia a un recurso web que especifica su ubicación en una red informática y un mecanismo para recuperarlo.
- **Frontend**: la infraestructura *frontend* incluye todos aquellos elementos del sistema con los que el usuario final interactúa.

- **Backend:** parte de la infraestructura que contiene el conjunto de servidores, bases de datos, APIs y los sistemas operativos que alimentan el *frontend* de una aplicación.

B.2. Descripción global

Objetivos generales

Los principales objetivos del sistema *software* a desarrollar son los siguientes:

- Implementar un servicio de generación de resúmenes abstractivos en la nube, empleando modelos pre-entrenados del estado del arte.
- Implementar el *backend* que haga posible esta generación de resúmenes, así como una API REST para exponer el *backend*.
- Diseñar la arquitectura del *backend* con aspectos como la flexibilidad, la escalabilidad y la alta disponibilidad como pilares fundamentales.
- Implementar un *frontend* (aplicación multiplataforma) que consuma la REST API y dé la posibilidad de generar resúmenes a cualquier usuario.
- Implementar una interfaz (tanto en el *backend* como en el *fontend*) que permita a los usuarios especificar los parámetros con los que el resumen será generado, como por ejemplo, su longitud relativa al texto original.

Características del usuario

Los usuarios potenciales de JIZT se pueden dividir en dos categorías fundamentales:

- Estudiantes de edades comprendidas entre los 15 y 25 años. El uso principal que harán del producto será el resumen de textos académicos.
- Personas de edades comprendidas entre los 25 y 50 años, cuyo principal uso será el resumen de noticias y artículos periodísticos.

En cuanto al sexo, el uso por parte tanto de hombres como mujeres será aproximadamente equivalente.

Un tercer tipo de usuario, no incluido anteriormente, serían partidos interesados en el uso del producto (empresas, particulares, etc.), que fueran a hacer un uso extensivo y exhaustivo del mismo. Este tipo de usuario requiere unas prestaciones más exigentes, pudiendo llegar a solicitar el despliegue de JIZT en sus propias dependencias.

Es por ello que el producto desarrollado deberá ser lo más independiente del entorno en el que se despliegue como sea posible.

B.3. Catalogo de requisitos

A continuación se detallan los requisitos funcionales y no funcionales del producto a desarrollar. Estos requisitos son globales al proyecto y, por tanto, involucran tanto al *backend* como a la aplicación a desarrollar.

Requisitos funcionales

- **RF-1 Solicitar resumen:** la API REST debe proporcionar un *end-point* para que el usuario pueda solicitar un resumen de un texto.
 - **RF-1.1 Elegir modelo de generación de resumen:** el usuario deberá ser capaz de especificar el modelo de generar su resumen¹.
 - **RF-1.2 Especificar longitud relativa del resumen:** el usuario deberá ser capaz de especificar la longitud del resumen generado, de manera relativa al texto original.
 - **RF-1.3 Especificar parámetros del resumen:** el usuario deberá ser capaz de especificar los parámetros concretos con los que se generará su resumen².
- **RF-2 Historial de resúmenes:** el usuario podrá acceder a los resúmenes que ha generado recientemente.
- **RF-3 Compartir resumen:** se le brindará al usuario la opción de compartir el resumen generado a través de otra aplicación a su elección.

¹ Este requisito solo se ha implementado en el *backend*. En la aplicación decidimos no incluirlo dado que por el momento solo hacemos uso de un modelo.

² Este requisito solo se ha implementado en el *backend*. En la aplicación se añadirá en futuras iteraciones, al considerarse no prioritario, dado que se trata de opciones avanzadas.

- **RF-4 Copiar resumen:** se le brindará al usuario la opción de copiar el resumen generado.
- **RF-5 Borrar resumen:** el usuario deberá ser capaz de borrar permanentemente un resumen previamente generado.
- **RF-6 Pegar desde el portapapeles:** la aplicación ofrecerá una opción para que el usuario pueda pegar el texto a resumir desde el portapapeles de forma sencilla.
- **RF-7 Mostrar metadatos:** el sistema brindará al usuario los metadatos relativos al resumen generado, como la hora a la que fue creado.
- **RF-8 Pre-procesado del texto:** el sistema será capaz de recibir texto con errores de formateo (exceso de espacios, saltos de carro situados en mitad de una frase, etc.), así como caracteres «extraños». Independientemente de lo anterior, el resumen generado aparecerá correctamente formateado y sin los mencionados caracteres.
- **RF-9 Textos arbitrariamente largos:** el sistema será capaz de producir resúmenes de cualquier texto, independientemente de la longitud de los mismos. No se espera, no obstante, que el tiempo de resumen de textos extremadamente largos esté por debajo del orden del minuto.

Requisitos no funcionales

- **RNF-1 Escalabilidad:** la arquitectura del sistema deberá permitir el escalado del mismo de forma rápida y sencilla.
 - **RNF-1.1 Autoescalado:** el sistema podrá escalarse de manera automática en momentos en los que la carga de trabajo así lo requiera. Del mismo modo, cuando dicha carga remita, deberá disminuir su escala, a fin de consumir los mínimos recursos posibles.
- **RNF-2 Alta disponibilidad:** el sistema deberá garantizar el acceso al mismo por parte de los usuarios en el 99,99 % de los casos.
 - **RNF-2.1 Tolerancia frente a fallos:** el sistema deberá ser capaz de recuperarse de forma automática de posibles errores o problemas de funcionamiento de cualquiera de sus componentes en un tiempo menor a los 2 minutos.

- **RNF-3 Eficiencia:** el sistema deberá ser capaz de generar un elevado número de resúmenes provenientes de diferentes usuarios de forma simultánea, sin que el tiempo medio de resumen se vea afectado.
- **RNF-4 Seguridad lógica y de datos:** se debe garantizar la correcta protección de todos los datos manejados por el sistema.
- **RNF-5 Privacidad:** se debe asegurar la protección de los datos de carácter personal.
 - **RNF-5.1 Anonimidad:** en ningún caso se recopilará información de los usuarios que permita determinar la identidad de los mismos. No obstante, el sistema no es responsable de garantizar que los textos introducidos no contienen información de carácter personal.
- **RNF-6 Usabilidad:** el tiempo medio de aprendizaje de la aplicación por parte de los usuarios deberá ser inferior a los 5 minutos. Además, el sistema contará con documentación en línea detallada del producto.
- **RNF-7 Multiplataforma:** se distribuirán los binarios de la aplicación necesarios para su ejecución en móvil (Android e iOS), *web* (Google Chrome, Mozilla Firefox, Safari y Microsoft Edge), y escritorio (Linux, Apple y Windows).
- **RNF-8 Tamaño reducido:** el peso de la aplicación no debe superar los 30 MB.

B.4. Especificación de requisitos

En esta sección, nos centramos en la definición de los casos de uso de nuestro producto.

Dado que el usuario interactuará únicamente con la aplicación (*frontend*), el *backend* no se considera en este caso, aunque sigue siendo vital para que los casos de uso de la aplicación se puedan completar con éxito.

Diagrama de casos de uso

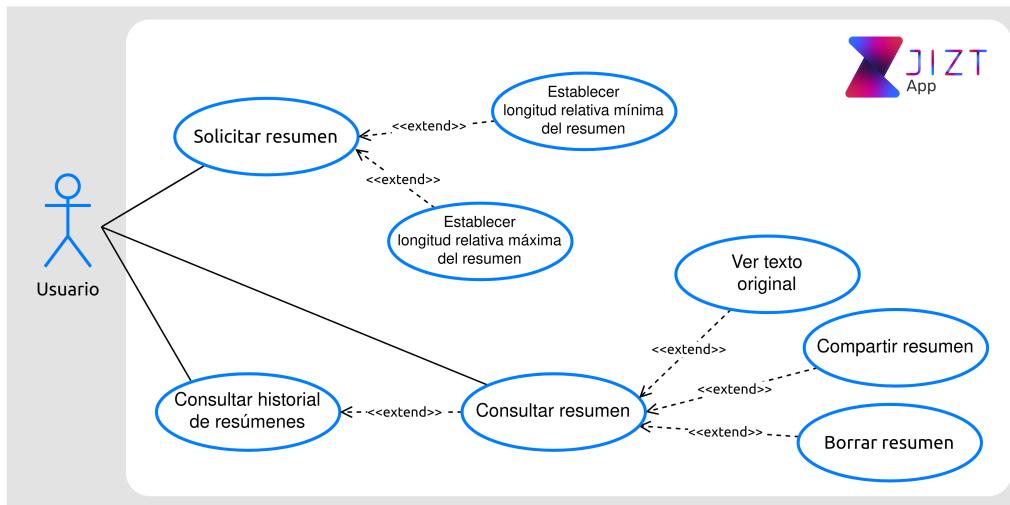


Figura B.1: Diagrama de casos de uso.

Actores

Existe un único actor: el usuario que hace uso de la aplicación.

Casos de uso

CU-01	Solicitar resumen
Descripción	Solicitar la generación de un resumen a partir de un texto.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-1, RF-6, RF-8, RF-9

Tabla B.1: CU-01 Solicitar resumen

CU-01	Solicitar resumen
Precondición	La API REST se encuentra accesible.
Flujo normal	<ol style="list-style-type: none"> 1. El usuario inicia la aplicación. 2. El usuario hace <i>click</i> en el área de texto. 3. El usuario introduce el texto a resumir o, alternativamente, lo pega desde el portapapeles. 4. El usuario pulsa en el botón «Resumir». 5. Se muestra un indicador de «procesando». 6. Se muestra un indicador de «resumen completado». 7. Se muestra el resumen generado.
Postcondición	El usuario ha obtenido el resumen de su texto.
Excepciones	API REST inaccesible.
Incluye	-
Extiende	-
Prioridad	Muy alta.
Frecuencia de uso	Muy alta.
Importancia	Crítica.
Comentarios	-

Tabla B.1: CU-01 Solicitar resumen

CU-02	Establecer longitud relativa mínima del resumen
Descripción	Establecer la longitud mínima que puede tener el resumen generado de manera relativa al texto original.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-1.2
Precondición	-
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el cuadro de texto en la pantalla principal. 2. El usuario ajusta la longitud mínima a través del <i>slider</i> que aparece en la parte inferior de la pantalla.
Postcondición	Se ha establecido la longitud mínima.
Excepciones	-
Incluye	-

Tabla B.2: CU-02 Establecer longitud relativa mínima del resumen

CU-02	Establecer longitud relativa mínima del resumen
Extiende	-
Prioridad	Alta.
Frecuencia de uso	Alta.
Importancia	Alta.
Comentarios	-

Tabla B.2: CU-02 Establecer longitud relativa mínima del resumen

CU-03	Establecer longitud relativa máxima del resumen
Descripción	Establecer la longitud máxima que puede tener el resumen generado de manera relativa al texto original.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-1.2
Precondición	-
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el cuadro de texto en la pantalla principal. 2. El usuario ajusta la longitud máxima a través del <i>slider</i> que aparece en la parte inferior de la pantalla.
Postcondición	Se ha establecido la longitud máxima.
Excepciones	-
Incluye	-
Extiende	-
Prioridad	Alta.
Frecuencia de uso	Alta.
Importancia	Alta.
Comentarios	-

Tabla B.3: CU-03 Establecer longitud relativa máxima del resumen

CU-04	Consultar historial de resúmenes
Descripción	Visualizar la lista de resúmenes generados previamente.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-2, RF-3, RF-4, RF-5, RF-7
Precondición	Haber generado al menos un resumen previamente.
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pulsa en «Ver todos» en la pantalla principal. 2. Se muestra la lista de resúmenes previos.
Postcondición	Se visualizan los resúmenes generados.
Excepciones	-
Incluye	-
Extiende	CU-07
Prioridad	Alta.
Frecuencia de uso	Alta.
Importancia	Alta.
Comentarios	Si aún no se ha generado ningún resumen, la lista se mostrará vacía.

Tabla B.4: CU-04 Consultar historial de resúmenes

CU-05	Consultar resumen
Descripción	Consultar un resumen generado previamente.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-2
Precondición	Haber generado al menos un resumen previamente.
Flujo normal	<p>Flujo 1:</p> <ol style="list-style-type: none"> 1. El usuario pulsa en uno de los resúmenes que aparecen en el inferior de la pantalla. <p>Flujo 2 (alternativa):</p> <ol style="list-style-type: none"> 1. El usuario pulsa en «Ver todos» en la pantalla principal. 2. Se muestra la lista de resúmenes previos. 3. El usuario pulsa en uno de los resúmenes.
Postcondición	Se ha mostrado el resumen seleccionado.

Tabla B.5: CU-05 Consultar resumen

CU-05	Consultar resumen
Excepciones	-
Incluye	-
Extiende	CU-06
Prioridad	Muy alta.
Frecuencia de uso	Alta.
Importancia	Crítica.
Comentarios	-

Tabla B.5: CU-05 Consultar resumen

CU-06	Ver texto original
Descripción	Visualizar el texto a partir del cual se ha generado el resumen.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-2, RF-7
Precondición	Haber generado un resumen.
Flujo normal	1. El usuario pulsa en «Original».
Postcondición	Se ha mostrado el texto original.
Excepciones	-
Incluye	-
Extiende	CU-07
Prioridad	Alta.
Frecuencia de uso	Alta.
Importancia	Crítica.
Comentarios	-

Tabla B.6: CU-06 Ver texto original

CU-07	Compartir el resumen
Descripción	Compartir el resumen generado a través de otra aplicación.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-3

Tabla B.7: CU-07 Compartir el resumen

CU-07	Compartir el resumen
Precondición	Haber generado un resumen.
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pulsa en el ícono de compartir. 2. Se muestra una lista de aplicaciones. 3. El usuario pulsa en la aplicación a través de la cual quiere compartir el resumen
Postcondición	Se ha compartido el resumen.
Excepciones	-
Incluye	-
Extiende	CU-07
Prioridad	Media.
Frecuencia de uso	Media.
Importancia	Media.
Comentarios	-

Tabla B.7: CU-07 Compartir el resumen

CU-08	Borrar el resumen
Descripción	Borrar un resumen generado.
Autor	Diego Miguel Lozano
Requisitos relacionados	RF-5
Precondición	Haber generado un resumen.
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pulsa en el ícono de borrar. 2. Se elimina el resumen y la aplicación vuelve a la pantalla principal.
Postcondición	Se ha borrado el resumen.
Excepciones	-
Incluye	-
Extiende	CU-07
Prioridad	Alta.
Frecuencia de uso	Baja.
Importancia	Alta.
Comentarios	-

Tabla B.8: CU-08 Borrar el resumen

Apéndice C

Especificación de diseño

C.1. Introducción

En este apéndice se recogen las características de diseño que implementan los requisitos descritos en el apéndice anterior, a fin de establecer los criterios que se deberán seguir en el desarrollo del sistema.

C.2. Elección del nombre del proyecto

Para la elección del nombre bajo el cual se englobara todo el servicio de resumen de textos en la nube, se consideraron numerosas opciones, de las cuales, las más relevantes fueron:

- **Summit**: acrónimo procedente de *Summarize it*. Además, «summit» tiene el significado de «cumbre» o «cima» en inglés, sugiriendo el alto rendimiento y potencia del servicio, alcanzando los más *altos* resultados.
- **Jizt**: esta palabra, pronunciada en inglés, suena casi idénticamente a «*gist*», la cual se puede traducir como «esencia» o «quid de la cuestión».
- **Halb**: al revés se lee «*blah*» (de *blah, blah, blah*, en inglés) y en alemán significa «mitad», evocando a la idea de resumen. Además se podría ver como una referencia al supercomputador HAL 9000 de la película y posterior novela *2001: Una Odisea del Espacio*.

De estas tres, se juzgaron más interesantes las dos últimas; la primera resulta menos original debido a que ya existen numerosas empresas, productos y proyectos con este nombre.

En ese momento, se llevó a cabo una pequeña encuesta para determinar el nombre final. Dado que el proyecto aspira a tener un carácter internacional, el objetivo de la encuesta no era tanto obtener un gran numero de votaciones, sino que las que tuviéramos fueran muy variadas: en total se preguntó a 22 personas de 15 nacionalidades diferentes.

A continuación, se incluye una tabla resumiendo los resultados de la encuesta:

	Elección de Nombre	
	<i>jizt</i>	<i>halb</i>
España	2	3
Alemania	3	0
México	1	1
Taiwán	1	0
Japón	0	1
Escocia	0	1
China	0	1
Francia	0	1
Turquía	0	1
Vietnam	1	0
Hungría	1	0
Bielorrusia	1	0
Corea del Sur	1	0
Malasia	1	0
Serbia	1	0
Total	13	9

Tabla C.1: Resultados de la encuesta.

Aunque los resultados de la encuesta estuvieron bastante ajustados, finalmente la balanza se decantó del lado de JIZT. Algunos de los participantes destacaron su «sonido vibrante y moderno». Por otro lado, el término Halb pareció no convencer a los alemanes, quienes la veían como una palabra demasiado convencional (recordamos que en alemán tiene el significado de «mitad»).

C.3. Diseño de datos

JIZT comprende dos dominios independientes: el *backend* y el *frontend*. El primero, el *backend*, incluye la API REST, así como los diferentes microservicios que hacen posible la generación de resúmenes en la nube. A su vez, el *frontend* está conformado por la aplicación multiplataforma que se comunica con el *backend* para proporcionar los resúmenes a los usuarios.

Dado que, como decíamos, el diseño de datos es independiente en el caso del *backend* y del *frontend* (siempre que se introduzcan mecanismos para transformar los datos de un dominio al otro, el cual es nuestro caso), atenderemos a cada uno de estos diseños por separado.

Backend

El *backend* sigue una arquitectura de microservicios. Debido a la propia naturaleza de esta arquitectura, cada uno de estos microservicios tiene una implementación independiente del resto.

Por tanto debemos atender a las clases y sus relaciones en cada microservicio por separado. No obstante, existen dos clases que van a aparecer en todos los microservicios. Se trata de las clases relacionadas con Kafka, esto es, el consumidor y el productor¹:

- **Productor de Kafka (*Producer*)**: esta clase contiene la lógica que posibilita el envío los mensajes correspondientes a uno o varios Kafka *topics*. Los Kafka *topics* concretos a los que envía el mensaje varían en función del microservicio.
- **Consumidor de Kafka (*Consumer*)**: un consumidor presta atención constante a los *topics* a los que está suscrito, y en el caso de que se haya producido algún mensaje, lo consume. Los Kafka *topics* concretos a los que un consumidor está suscrito varían en función del microservicio.

Una vez introducidas estas dos clases comunes, veamos las clases concretas de cada microservicio.

¹ Para más información sobre Kafka, se recomienda acudir al capítulo de «Técnicas y herramientas» de la Memoria.

Microservicio *Dispatcher*

El *Dispatcher* se encarga de recibir las peticiones de los clientes, validarlas, y reenviarlas hacia el microservicio correspondiente. Como tal, implementa la API REST.

Este microservicio cuenta con las siguientes clases:

- **Servicio *Dispatcher* (*DispatcherService*)**: clase principal del microservicio. Dirige las interacciones entre las instancias del resto de clases.
- **Resumen (*Summary*)**: se trata de la representación de un resumen, por lo que es una de las clases centrales en todo el proyecto. Entre sus campos más importantes se encuentran su *id*, el cual permite identificar inequívocamente un resumen, así como el texto del propio resumen, y el texto fuente a partir del cual se ha generado el resumen.
- **Interfaz Resumen DAO (*SummaryDAOInterface*)**: interfaz DAO (*Data Access Object*) para abstraer el acceso a los datos independientemente de la base de datos empleada.
- **Factoría Resumen DAO (*SummaryDAOFactor*)**: clase que recoge las instancias particulares de los DAOs correspondientes cada base de datos.
- **PostgreSQL DAO Resumen (*SummaryDAOPostgresql*)**: DAO concreto para la base de datos PostgreSQL. esta base de datos almacena los resúmenes generados.
- **Esquema para Petición de Texto Plano (*PlainTextRequestSchema*)**: esta clase contiene la estructura (campos) que debe seguir el cuerpo de las peticiones HTTP de los clientes a la hora de realizar una operación POST sobre la API REST. Se incluye la distinción de «texto plano», dado que en un futuro se podrán realizar peticiones enviando una URL o un documento, y por lo tanto necesitaremos otro esquema diferente para cada uno de estos casos. Este esquema se emplea también para los mensajes producidos por el *Dispatcher* al *topic* del *Pre-procesador* (el siguiente microservicio).
- **Esquema para Respuesta (*ResponseSchema*)**: estructura (campos) del cuerpo de la respuesta HTTP de la API REST ante una petición POST o GET por parte del cliente. Contiene los detalles del resumen producido.

- **Esquema para Mensajes Consumidos**

(*TextPostprocessingConsumedMsgSchema*): estructura (campo) que presenta un mensaje consumido por el *Dispatcher*. Estos mensajes procederán del *topic* del microservicio Post-procesador de textos.

Diagrama de clases

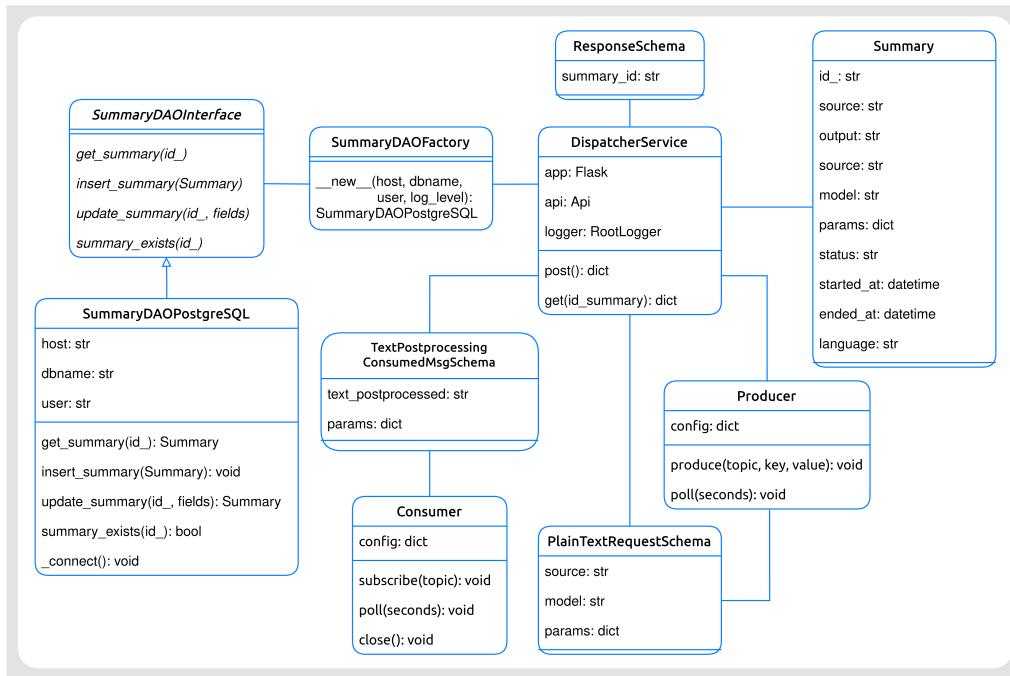


Figura C.1: Diagrama de clases del *Dispatcher*.

Modelo relacional de la base de datos

A su vez, la base de datos gobernada por el *Dispatcher* y encargada de almacenar los resúmenes generados, cuenta con el siguiente esquema de tablas:

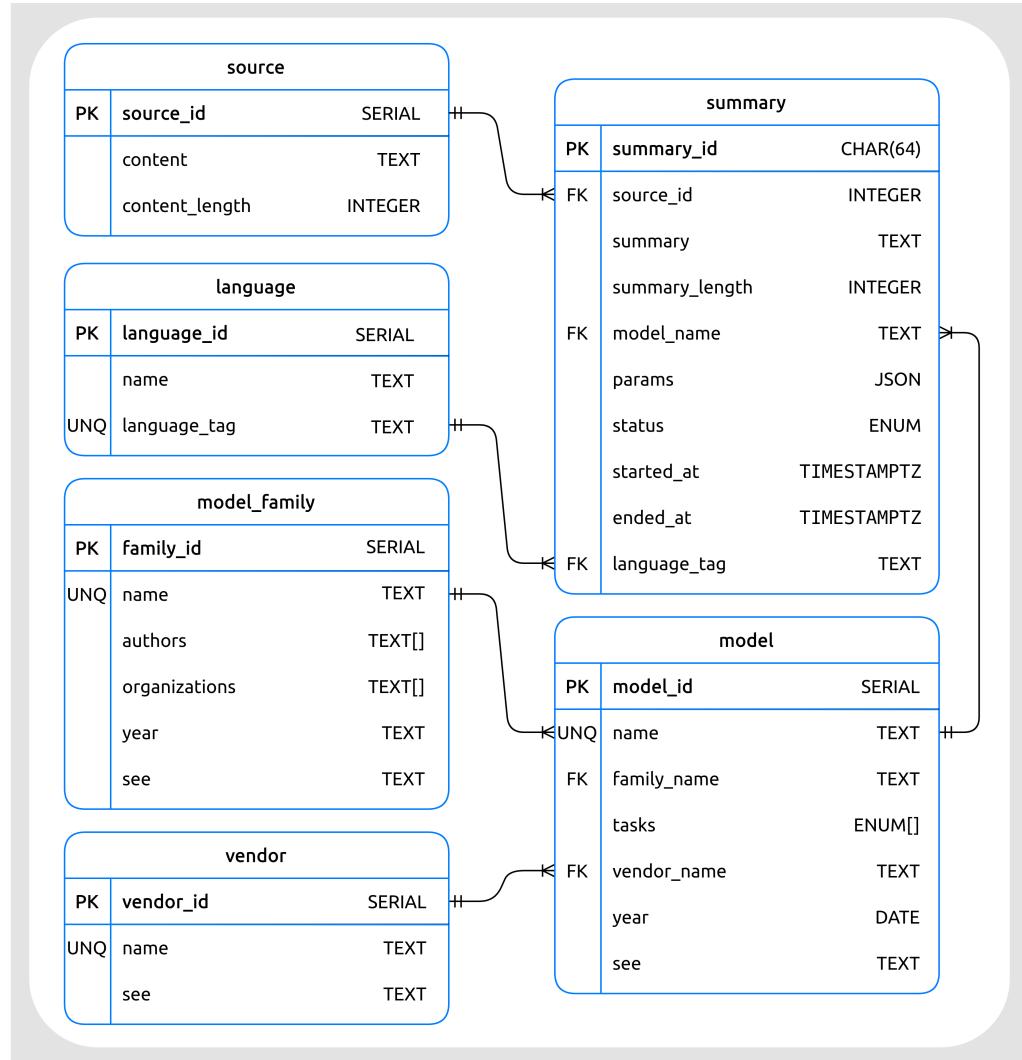


Figura C.2: Modelo relacional de la base de datos (tipos de datos de PostgreSQL).

Microservicio Pre-procesador de textos

El Pre-procesador se encarga de realizar un primer procesado de los textos de entrada, a fin de que sean lo más cercanos como sea posible a la entrada que espera el modelo generador de resúmenes.

Este microservicio está compuesto por siguientes clases:

- **Pre-procesador de textos (*TextPreprocessor*)**: esta clase es la encargada de realizar el pre-procesado del texto.
- **Esquema para Mensajes Consumidos (*TextPreprocessingConsumedMsgSchema*)**: estructura (campos) que presenta un mensaje consumido por el Pre-procesador. Estos mensajes procederán del microservicio *Dispatcher*.
- **Esquema para Mensajes Producidos (*TextEncodingProducedMsgSchema*)**: estructura (campos) que presenta un mensaje producido al *topic* del siguiente microservicio (el Codificador de textos).

Diagrama de clases

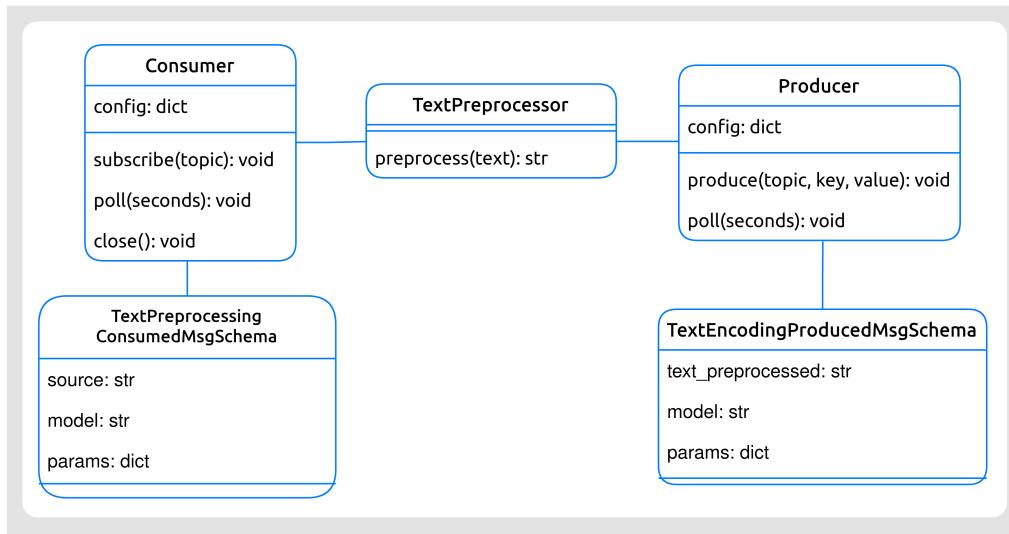


Figura C.3: Diagrama de clases del Pre-procesador de textos.

Microservicio Codificador de textos

Este microservicio se encarga de codificar el texto y dividirlo en fragmentos menores, a fin de respetar la longitud máxima de entrada del modelo generador de resúmenes.

El Codificador cuenta con las siguientes clases:

- **Codificador y divisor de textos (*SplitterEncoder*)**: esta clase es la encargada de realizar el codificado y división del texto.
- **Esquema para Mensajes Consumidos (*TextEncodingsConsumedMsgSchema*)**: estructura (campos) que presenta un mensaje consumido por el Codificador. Estos mensajes procederán del microservicio Pre-procesador de textos.
- **Esquema para Mensajes Producidos (*TextSummarizationProducedMsgSchema*)**: estructura (campos) que presenta un mensaje producido al *topic* del siguiente microservicio (el Generador de resumen).

Diagrama de clases

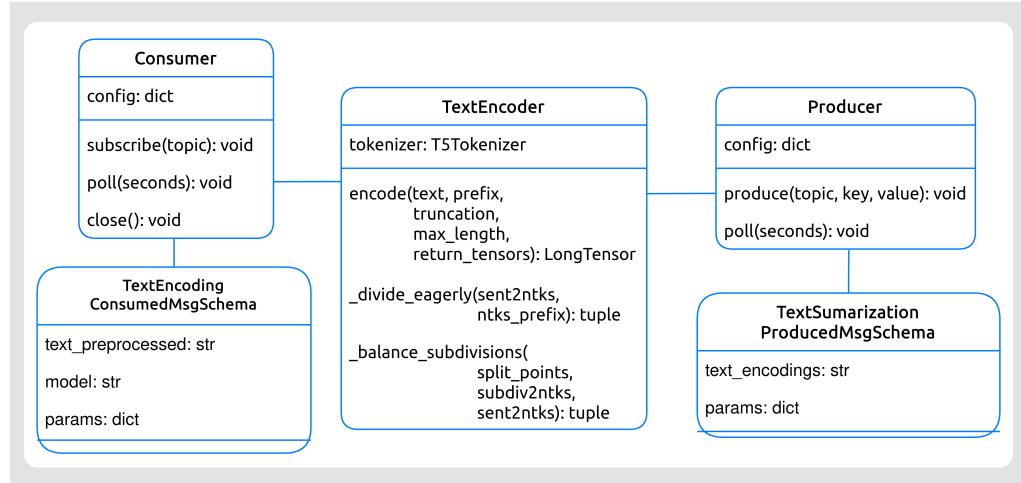


Figura C.4: Diagrama de clases del Codificador de textos.

Microservicio Generador de resúmenes

Este microservicio se encarga de generar resúmenes a partir del texto codificado que recibe del Codificador.

Las clases del Generador de resúmenes son:

- **Generador de resúmenes (*Summarizer*)**: esta clase es la encargada de generar los resúmenes.
- **Esquema para Mensajes Consumidos (*TextSummarizationConsumedMsgSchema*)**: estructura (campos) que presenta un mensaje consumido por el Generador de resúmenes. Estos mensajes procederán del microservicio Codificador de textos.
- **Esquema para Mensajes Producidos (*TextPostprocessingProducedMsgSchema*)**: estructura (campos) que presenta un mensaje producido al *topic* del siguiente microservicio (el Post-procesador de textos).

Diagrama de clases

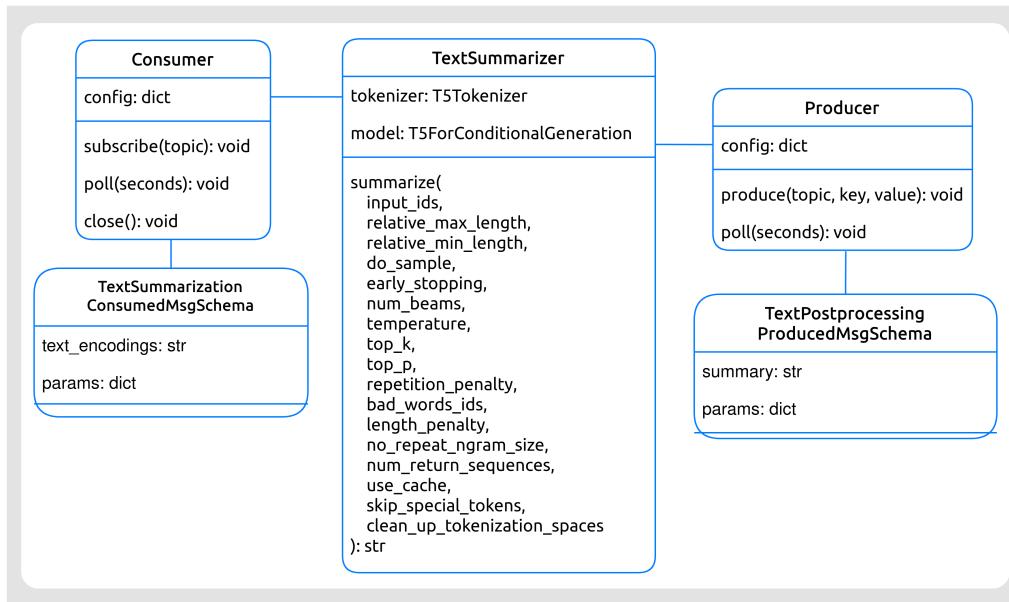


Figura C.5: Diagrama de clases del Generador de resúmenes.

Microservicio Post-procesador de textos

Este microservicio lleva a cabo el post-procesado del resumen entregado por el Generador de resúmenes.

El Post-procesador de textos tiene las siguientes clases.

- **Post-procesador de textos (*TextPostprocessor*):** esta clase es la encargada de post-procesar el texto.
- **Esquema para Mensajes Consumidos (*TextPostprocessingConsumedMsgSchema*):** estructura (campos) que presenta un mensaje consumido por el Post-procesador de textos. Estos mensajes procederán del microservicio Generador de resúmenes.
- **Esquema para Mensajes Producidos (*ReadyProducedMsgSchema*):** estructura (campos) que presenta un mensaje producido al siguiente *topic*, *Ready*.

Diagrama de clases

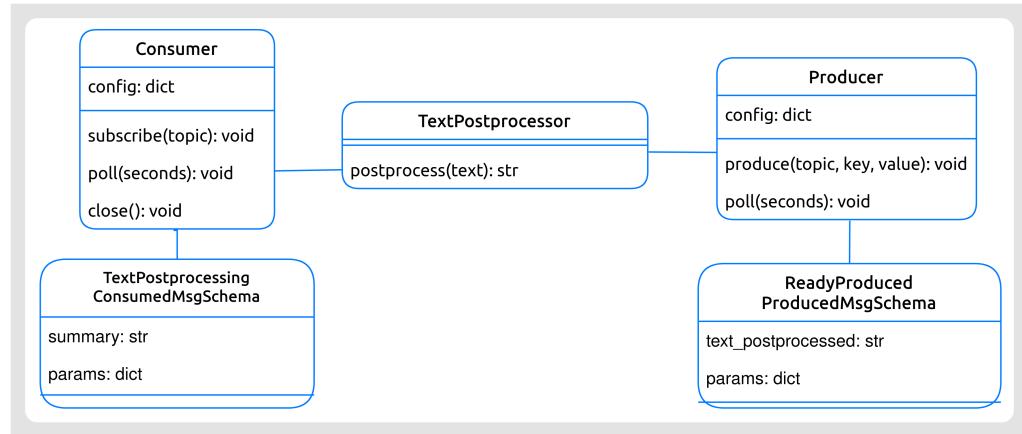


Figura C.6: Diagrama de clases del Post-procesador de textos.

Frontend

El *frontend* está compuesto por la aplicación multiplataforma desarrollada siguiendo los patrones de *Clean Architecture* y Diseño guiado por el dominio *Domain Driven Design*. Para conocer los detalles concretos de

la implementación de la arquitectura de la *app*, se recomienda acudir a la sección «Motivación tras las arquitecturas desarrolladas» del capítulo «Aspectos relevantes» de la Memoria.

Las clases identificadas en la aplicación son las siguientes:

- **JiztApp**: esta clase representa la aplicación en su conjunto. En el `main` de la aplicación, se ejecuta una instancia de esta clase.
- **NewSummaryPage**: se corresponde con la pantalla desde la cual los usuarios pueden solicitar nuevos resúmenes.
- **SummaryPage**: se corresponde con la pantalla en la que se muestra el resumen generado.
- **NewSummaryCubit**: clase encargada de transformar los eventos procedentes de los usuarios en la pantalla `NewSummaryPage` (por ejemplo, un *click*), en acciones particulares de la aplicación.
- **SummaryCubit**: análoga a la anterior solo que en este caso para la pantalla `SummaryPage`.
- **JiztRepository**: clase que encapsula y centraliza la lógica de acceso a las fuentes de datos.
- **JiztCacheClient**: clase que gestiona el acceso a la base de datos local.
- **JiztApiClient**: clase que gestiona la comunicación con la REST API de JIZT.
- **Summary**: representa un resumen. En realidad, disponemos de tres representaciones de un resumen, cada una correspondiéndose con cada uno de los dominios (capa de dominio, caché y REST API). No obstante, la estructura de las mismas es análoga; por consiguiente, las sintetizamos todas en una única clase para simplificar el esquema de diseño.

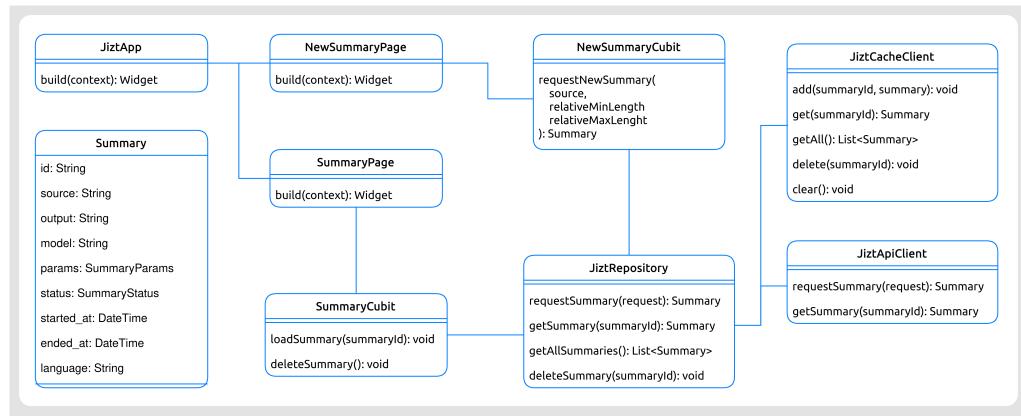


Figura C.7: Diagrama de clases de la aplicación de JIZT.

C.4. Diseño procedimental

En esta sección repasamos una vez más los pasos que se llevan a cabo tanto en el *backend* a la hora de atender las peticiones de los clientes y generar los resúmenes, como en la aplicación, encargada de gestionar los resúmenes de los usuarios.

En la sección correspondiente al *backend* del capítulo «Técnicas y Herramientas», explicábamos cómo se gestionaba la comunicación entre los clientes y la REST API a través de peticiones HTTP. Recordemos el proceso:

1. El cliente realiza una petición HTTP POST, incluyendo en el cuerpo el texto a resumir, así como los parámetros del resumen a generar.
2. Ingress (API *Gateway*) comprueba que dicha petición se está haciendo a un *endpoint* válido, y en ese caso la redirige hacia el *Dispatcher*.
3. El *Dispatcher* realiza una serie de comprobaciones:
 - a) Se consulta en la base de datos si ya existe un resumen generado para ese texto con esos parámetros. En ese caso, se responde al cliente con los datos del resumen (**output**, **source**, **started_at**, **ended_at**, etc.). El **status** del resumen será **completed** («completado»).
 - b) En caso contrario, se responde con el mismo esquema de datos, solo que el **output** será **null**, y el **status** será **summarizing** («resumiendo»). Al mismo tiempo, se produce un mensaje al

topic del pre-procesador de textos, contenido el texto y los parámetros del resumen, comenzando el proceso de generación.

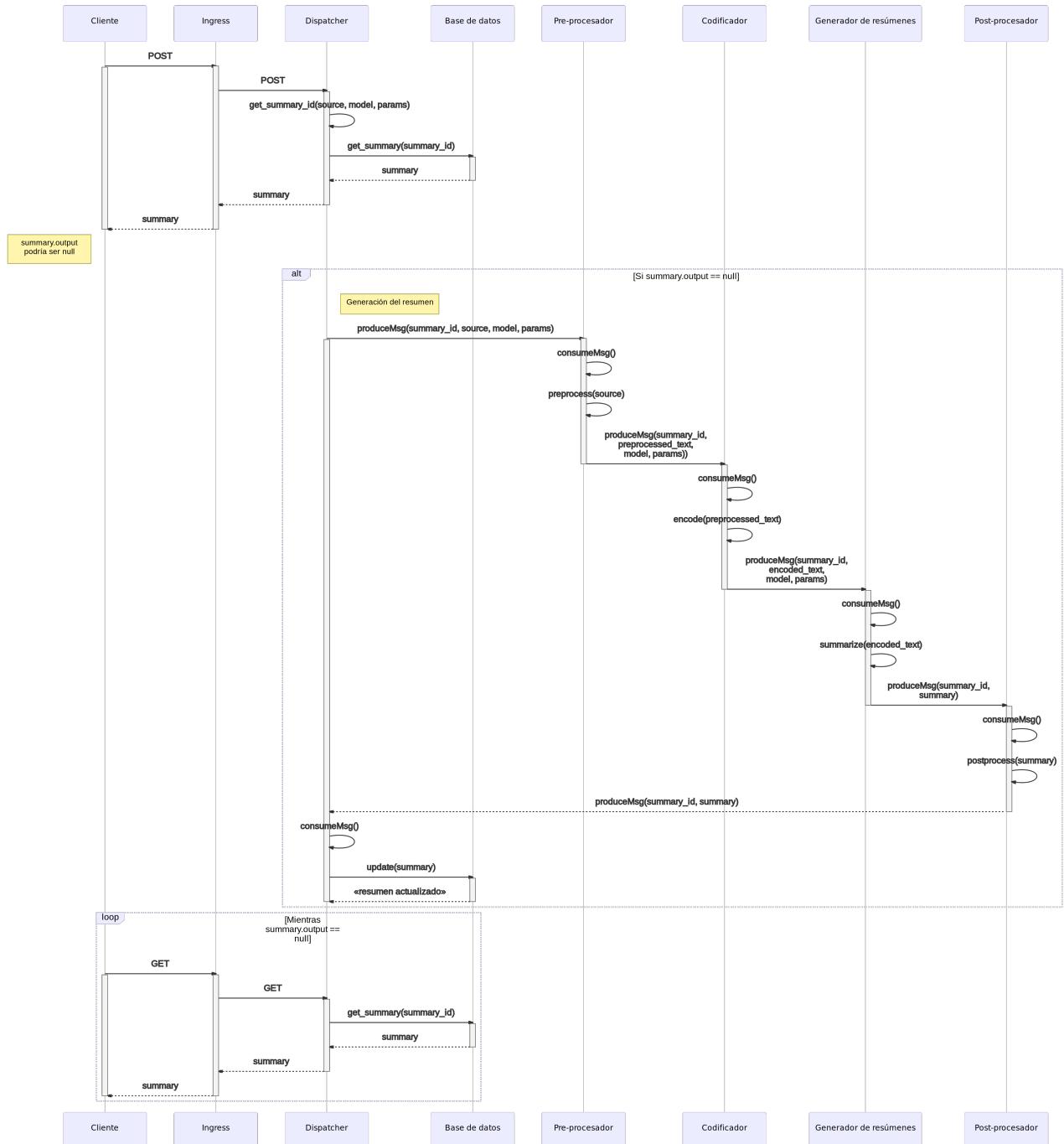
4. El pre-procesador está constantemente comprobando si existen mensajes nuevos en su *topic*. En ese caso los consume, realiza las tareas de pre-procesado, y produce el resultado en el *topic* del codificador.
5. Este proceso continua de forma análoga hasta llegar al post-procesador, el cual produce el resumen final al *topic* «Listo» (*Ready*). El *Dispatcher*, en ese momento, consume el mensaje, y actualiza la base de datos.
6. Mientras el proceso de resumen se completa, el cliente realiza peticiones GET de manera periódica, hasta que la API REST finalmente responde con el resumen generado.

La [Figura C.8](#) ilustra este proceso de forma gráfica, a través de un diagrama de secuencia.

A su vez, en la aplicación, es decir, el *frontend*, de llevan a cabo los siguientes pasos:

1. El usuario solicita generar un nuevo resumen, pulsando para ello sobre la pantalla.
2. El evento se transforma a través del *cubit* en una petición de resumen al repositorio.
3. El repositorio realiza una petición HTTP POST al *backend* para obtener el *id* del resumen a partir del texto fuente, los parámetros del resumen, y el modelo.
4. Una vez obtenido el *id* del resumen, se consulta la base de datos local (capa de caché) para conocer si ya se dispone localmente del resumen solicitado. En ese caso, se devuelve el resumen, y se actualiza la pantalla para mostrárselo al usuario.
5. En caso contrario, se realizan peticiones GET periódicas a la API REST hasta que el resumen se completa. Finalmente, se actualiza la pantalla.

En la [Figura C.9](#), se incluye un diagrama de clases que describe el proceso de manera más detallada.

Figura C.8: Diagrama de secuencia del *backend*.

C.4. Diseño procedural

41

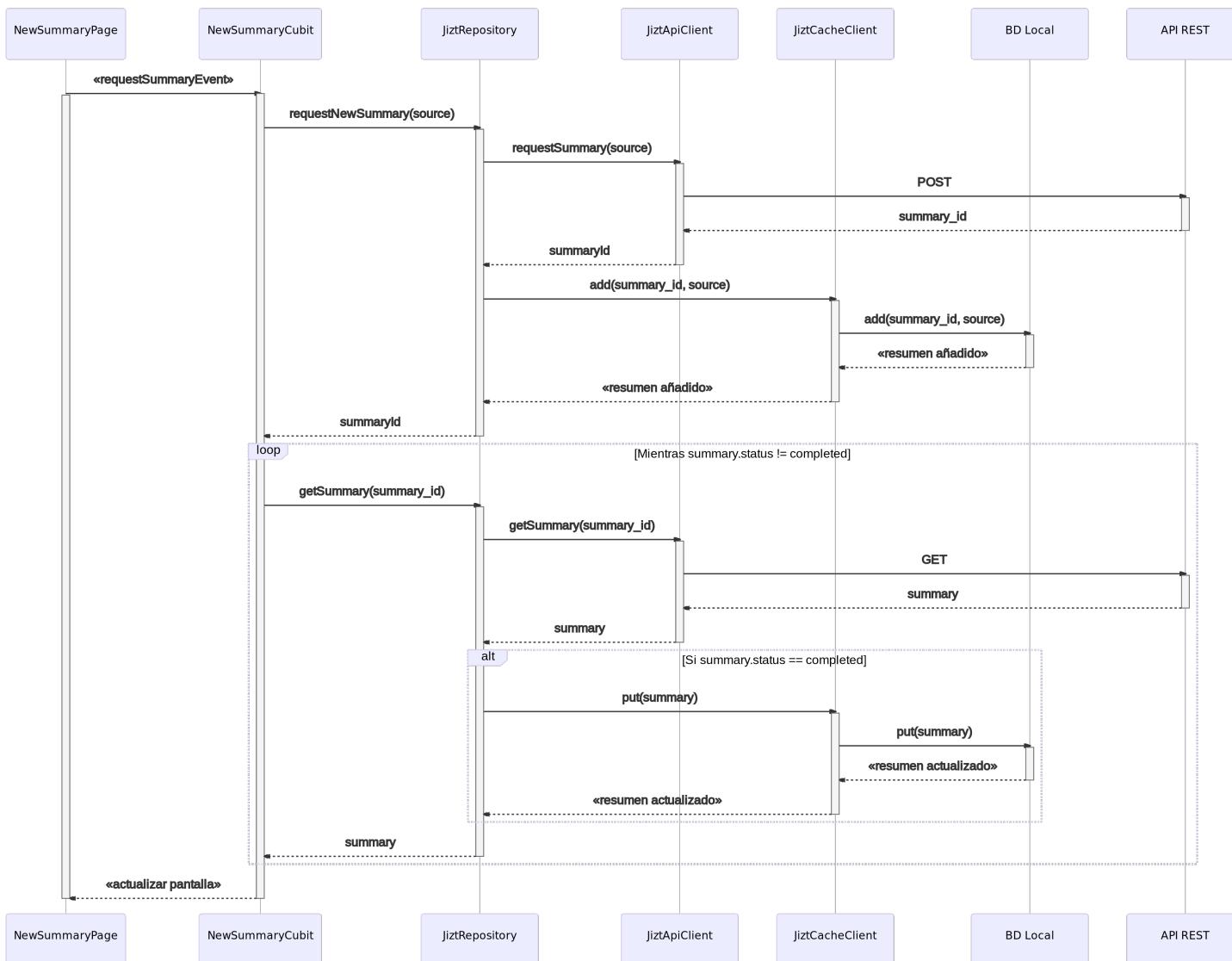


Figura C.9: Diagrama de secuencia de la aplicación.

C.5. Diseño arquitectónico

Dado que el diseño arquitectónico conforma uno de los principales aspectos a destacar de todo el proyecto, se ha incluido de manera detallada en la Memoria.

Backend

Por tanto, en esta sección llevaremos a cabo un resumen de los principales puntos de la arquitectura, e incluiremos información adicional acerca de los retos que surgieron, y cómo se resolvieron, dando con el diseño arquitectónico final.

Como se indica en la Memoria, para el *backend* se ha desarrollado una **arquitectura de microservicios**, a través de la cual podemos separar cada paso en la generación de resúmenes (pre-procesado, codificación, generación del resumen, post-procesado), en módulos independientes, favoreciendo los siguientes aspectos:

- Gracias a la división en microservicios, conseguimos una gran flexibilidad; los cambios en uno de los pasos en la generación del resumen no influyen al resto.
- Facilita la detección y corrección de cuellos de botella en el proceso, dado que podemos monitorizar de forma precisa el rendimiento de cada microservicio.
- La arquitectura es fácilmente escalable tanto en términos de los microservicios ya existentes, aumentando el número de réplicas de cada microservicio (permitiendo la generación en paralelo de varios resúmenes), como en términos de la adicción de nuevos microservicios, por ejemplo, diferentes modelos para distintos idiomas.
- Asegura una alta disponibilidad, ya si uno de los microservicios falla, el sistema crea una nueva instancia y finaliza el microservicio defectuoso. Adicionalmente, con una arquitectura de microservicios eliminamos cualquier posible punto de único fallo (*single point of failure*).

Otro de los principales aspectos de la arquitectura es cómo se lleva a cabo la comunicación y el correcto enrutado de los mensajes entre los diferentes microservicios. Al tratarse de un proceso secuencial, la salida de un microservicio será la entrada del siguiente.

Inicialmente, se pensó en resolver esta situación mediante el patrón de *routing-slips* (hojas de ruta) [12]. Con este patrón se incluye en el propio mensaje la ruta que este debe seguir, por lo que, implementando un *router* en cada microservicio que interpretara la hoja de ruta, podríamos, resolver el problema del enrutado. En cuanto a cómo se llevaría a cabo la comunicación, esta se podría hacer mediante peticiones HTTP, de forma que cada microservicio implementara su propia REST API.

Siguiendo esta estrategia, junto con el patrón de API *gateway*, a través que se ofrece un punto de entrada al *backend*, el diseño de la arquitectura quedaba como se muestra en la [Figura C.10](#).

Poco después, conocimos acerca de la **arquitectura dirigida por eventos** [13], y de Kafka [14], una de las herramientas más apropiadas y avanzadas a día de hoy para este tipo de arquitectura².

Con este nuevo diseño, la arquitectura se simplificaba en gran medida, y problemas como el escalado, o la entrega fiable de mensajes, corrían a cargo de Kafka, quienes gestionaba estos y otros aspectos de manera automática.

La arquitectura final del *backend* se ilustra en la [Figura C.11](#).

Frontend

Gracias a la vibrante comunidad de Flutter [4], dar con la arquitectura más apropiada para la aplicación resultó un proceso más fluido.

Antes de comenzar este proyecto, habíamos oído del concepto de *Clean Architecture* [15], aunque lo conocíamos únicamente de manera superficial. No obstante, en nuestra formación de Flutter, apareció de nuevo, y esta vez sí que profundizamos en él.

A continuación, nos informamos sobre el patrón BLoC [16], muy popular también dentro de la comunidad Flutter. Por suerte, existe asimismo un paquete para la implementación de este patrón [17].

Finalmente, la arquitectura quedó como se muestra en la [Figura C.12](#).

La información completa acerca de la arquitectura de la aplicación se puede encontrar en la Memoria.

² De nuevo, referimos al lector a la Memoria, donde se recogen las principales ventajas, tanto de este patrón arquitectónico, como de Kafka.

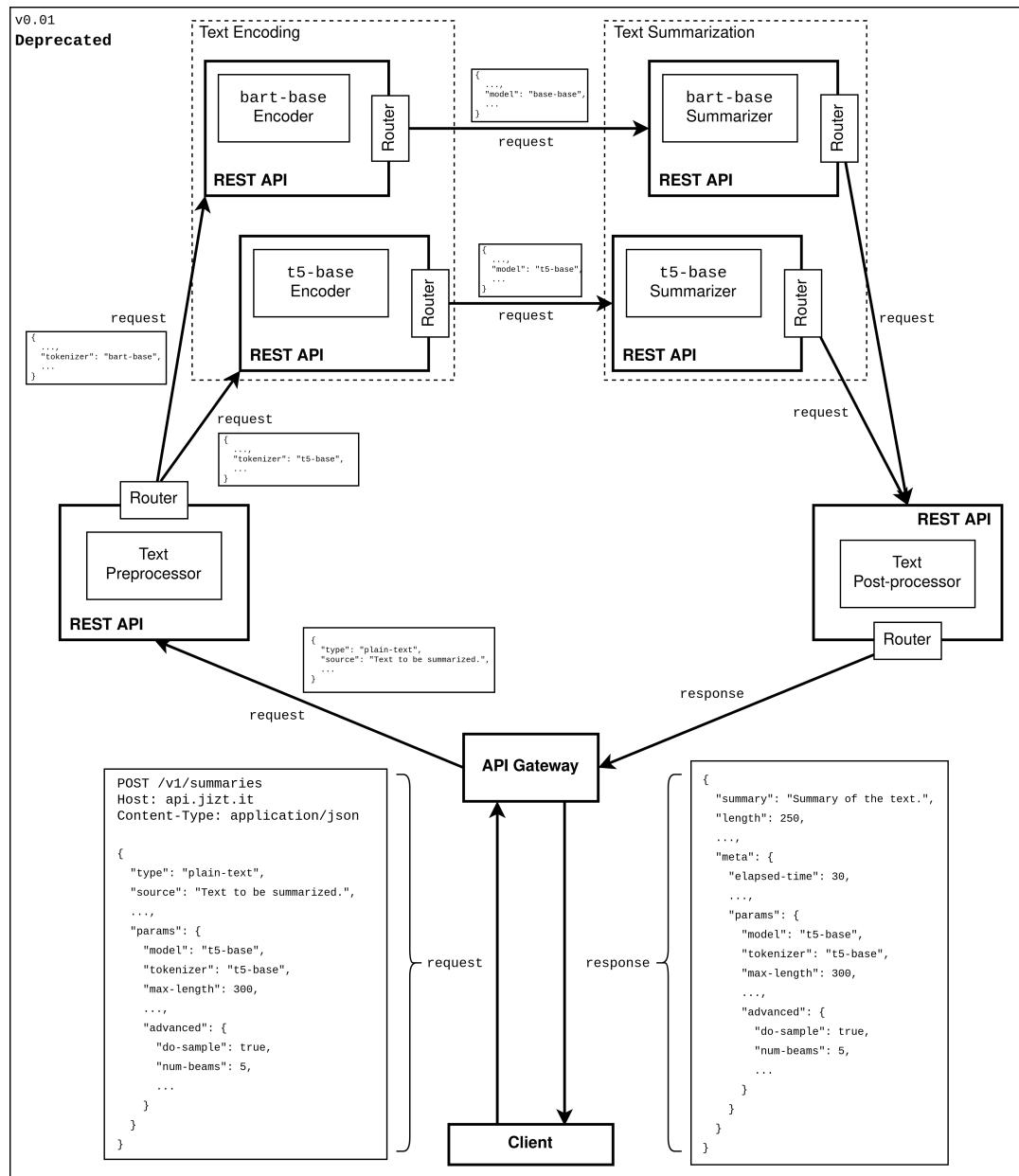


Figura C.10: Primera aproximación para el diseño arquitectónico del *backend*.

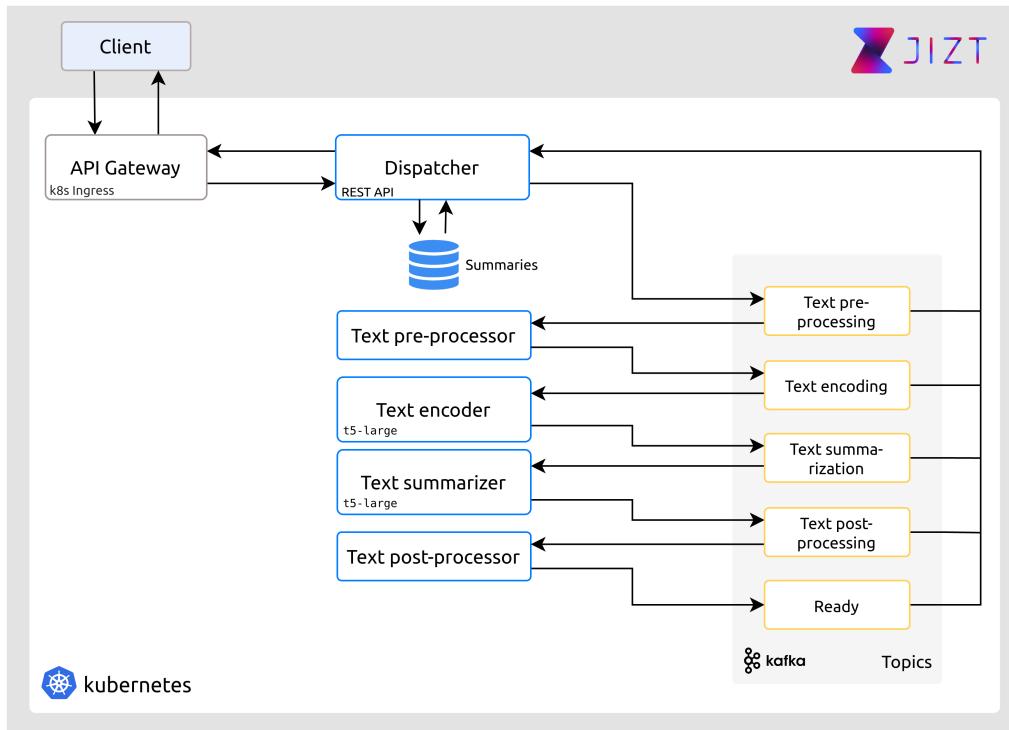
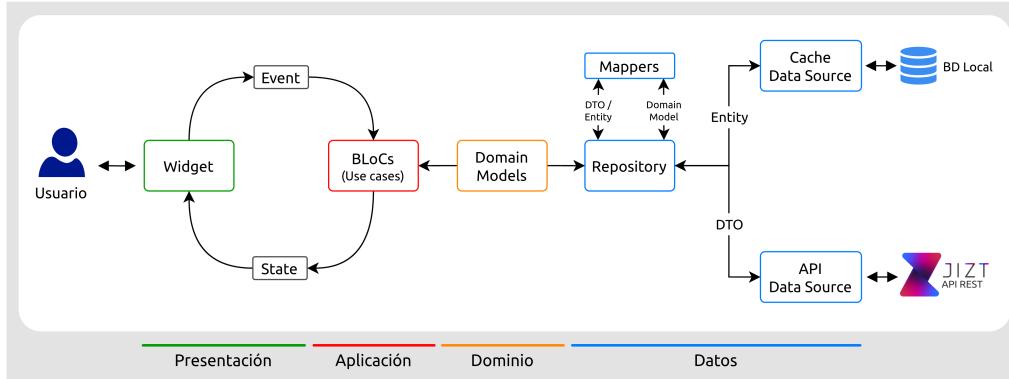
Figura C.11: Diseño final de la arquitectura del *backend*.

Figura C.12: Diseño final de la arquitectura de la aplicación.

C.6. Diseño de interfaces

Diseño del logo

Una de las intenciones detrás de este proyecto siempre ha sido tratar de crear una cierta imagen corporativa y de producto. Como consecuencia, el diseño de nuestra carta de presentación, es decir, nuestro logo, ocupó un papel central en las primeras iteraciones del proyecto.

En la búsqueda creativa de un logo atractivo, moderno y memorable, se experimentó con numerosos posibles diseños en papel.

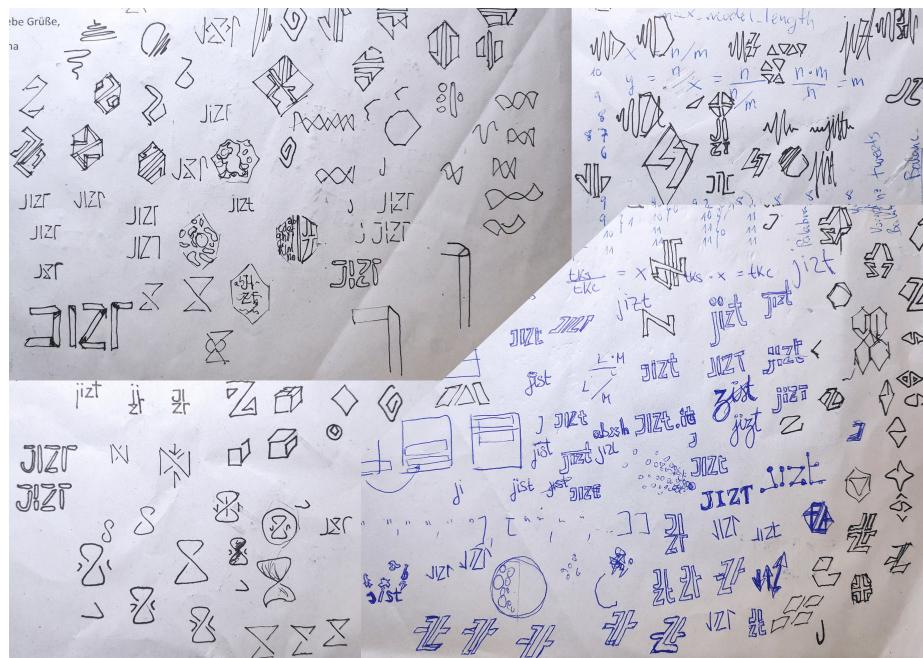


Figura C.13: Ideas, ideas, y más ideas. Pero solo unas pocas buenas.

Finalmente, dimos con un diseño que parecía tener potencial. Cogimos nuestro ordenador, y nos sumergimos en Adobe Illustrator, un editor de gráficos vectoriales muy popular. Por experiencia previa en diseño gráfico (autodidacta), sabemos que, teniendo una buena idea de partida, equivale a tener una gran parte del trabajo hecho.

Así pues, el logo final, nuestra carta de presentación, acabó luciendo como se muestra en la Figura C.14. En nuestra opinión, cumple con los requisitos esperados.



Figura C.14: JIZT - Generación de resúmenes mediante IA.

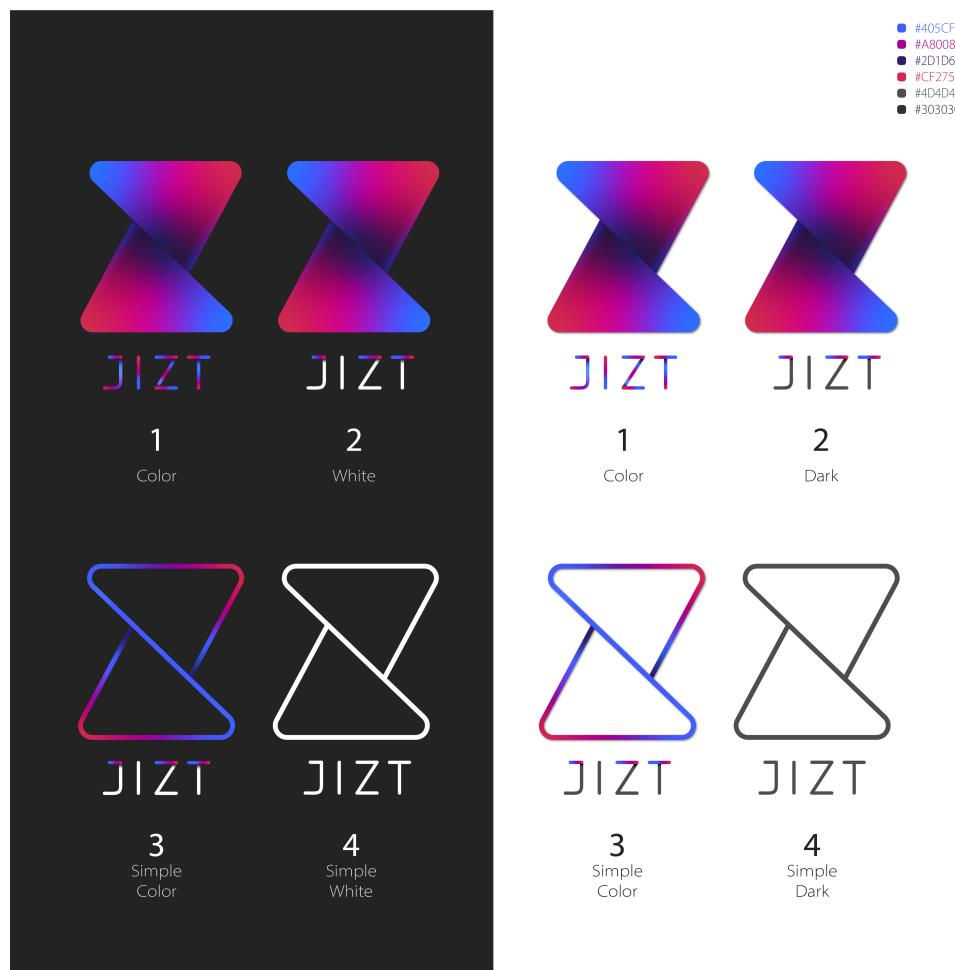


Figura C.15: Variaciones sobre el logo de JIZT.

Diseño de la interfaz gráfica de la aplicación

Para el diseño de la interfaz gráfica de usuario de la aplicación, trabajamos directamente sobre el ordenador, esta vez con el programa Inkscape, también editor de gráficos vectoriales, pero en este caso *open-source* y gratuito.

Se llevaron a cabo diferentes iteraciones hasta dar con un diseño que nos acabó pareciendo adecuado.

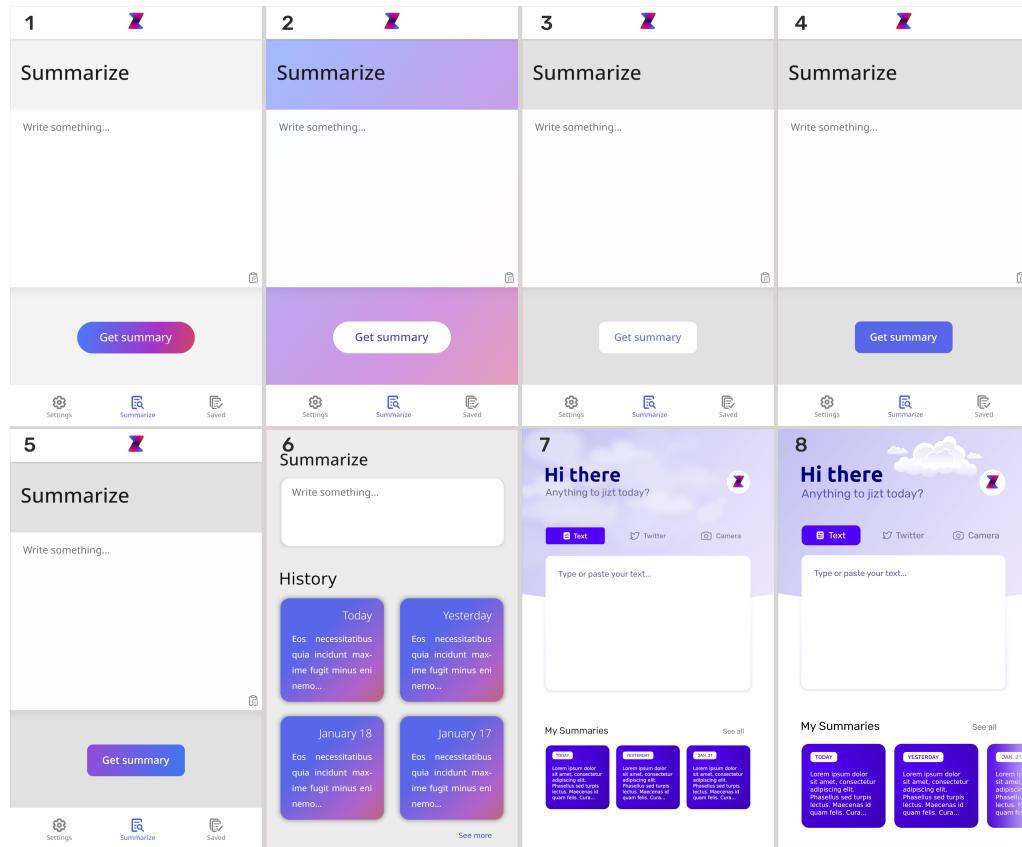


Figura C.16: Iteraciones sobre la pantalla principal.

Siguiendo el estilo de la pantalla principal, se diseñaron el resto de pantallas, las cuales se muestran en la [Figura C.17](#).

C.6. Diseño de interfaces

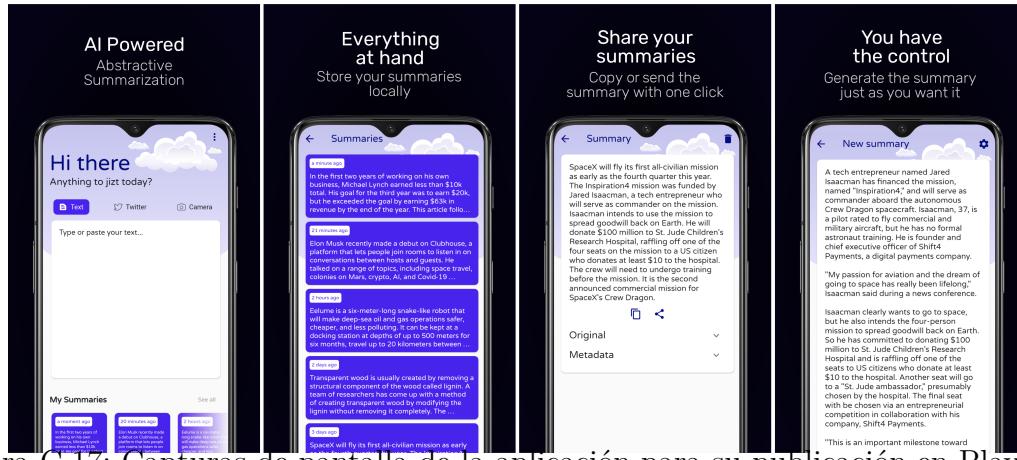


Figura C.17: Capturas de pantalla de la aplicación para su publicación en Play Store.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se detalla toda la información que pueda ser de ayuda para cualquier desarrollador interesado en desplegar y/o contribuir al proyecto.

De igual modo a como hemos venido haciendo en los anteriores apéndices, separaremos los detalles referentes al *backend* y al *frontend*, ya que se trata de entornos completamente independientes, cada uno con sus características y peculiaridades concretas.

D.2. Documentación técnica del *backend*

Estructura de directorios

La estructura de directorios del *backend* de JIZT, a la cual se puede acceder a través de github.com/dmlls/jitz, es la siguiente:

- **/src**: este directorio contiene el código fuente completo del *backend*.
- **/src/helm**: ficheros relativos a Helm [18].
- **/src/helm/jitz**: ficheros para el despliegue de los componentes relativos a JIZT en Kubernetes.
- **/src/helm/jitz/charts**: dependencias de paquetes Helm externos.
- **/src/helm/jitz/jitzlibchart**: librería de plantillas Helm para los componentes de JIZT.

- `/src/helm/jizt/setup`: *scripts* para el despliegue de JIZT en Kubernetes.
- `/src/services`: microservicios de JIZT.
- `/src/services/dispatcher`: microservicio *Dispatcher*.
- `/src/services/postgres`: esquemas iniciales de la base de datos.
- `/src/services/t5_large_text_encoder`: microservicio Codificador.
- `/src/services/t5_large_text_summarizer`: microservicio Generador de resúmenes.
- `/src/services/text_postprocessor`: microservicio Postprocesador de texto.
- `/src/services/text_preprocessor`: microservicio Preprocesador de texto.
- `/test`: pruebas unitarias y de sistema.

Manual del programador

En esta sección, se recogen todos los detalles necesarios para la instalación y despliegue de la arquitectura de microservicios en Kubernetes.

Instalación y despliegue del *backend*

El primer paso será descargarnos el código fuente correspondiente al *backend*, clonando el repositorio del proyecto alojado en GitHub:

```
git clone https://github.com/dmlls/jizt.git
```

Una vez disponemos del código fuente, debemos proceder primero a la instalación y provisión de los prerequisitos necesarios para la instalación de JIZT.

Nota importante: el despliegue del *backend* requiere de **conocimientos intermedios de Kubernetes**. En el presente manual, trataremos de explicar el proceso de instalación de la manera más detallada posible. No obstante, **no podemos condensar todo el conocimiento necesario de Kubernetes en este manual**, dado que para ello ya existe la propia documentación oficial de Kubernetes, la cual se recomienda encarecidamente revisar antes de comenzar la instalación de JIZT. Dicha documentación es accesible a través de <https://kubernetes.io/es/docs/home>.

En cualquier caso, para cualquier tipo de ayuda o duda acerca de la instalación o utilización de JIZT, se puede contactar con el equipo de soporte técnico de JIZT a través de la siguiente dirección de correo electrónico: jizt@diegomiguel.me. Estaremos encantados de atenderle.

Prerrequisitos

La infraestructura en la nube de JIZT se apoya en la utilización de diferentes plataformas y *frameworks* que se deben instalar con antelación.

Se recomienda ejecutar la instalación de JIZT a través de una máquina **GNU/Linux**. La instalación a través de otros sistemas operativos no ha sido probada y, por tanto, no se asegura una instalación exitosa.

Programas requeridos

Antes de comenzar con la instalación de JIZT, se deben instalar localmente los siguientes programas:

- **kubectl**: herramienta de línea de comandos de Kubernetes, empleada para el despliegue y la gestión de aplicaciones en Kubernetes. Se pueden encontrar las instrucciones de instalación en <https://kubernetes.io/es/docs/tasks/tools/install-kubectl>.
- **helm**: CLI para Helm, un popular gestor de paquetes para Kubernetes. Para su instalación, se deben seguir los pasos recogidos en <https://helm.sh/docs/intro/install>.
- **pgo**: CLI para el operador de PostgreSQL de Crunchy. A través de esta herramienta, podemos conectarnos con nuestra base de datos PostgreSQL desplegada en Kubernetes. Se puede consultar <https://access.crunchydata.com/documentation/postgres-operator/4.5.1/installation/pgo-client> para su instalación.
- SDK de Google Cloud: este SDK nos permite conectarnos y gestionar *clústers* de Kubernetes alojados en Google Cloud. En <https://cloud.google.com/sdk/docs/install> se detalla el proceso de instalación.

Se considera, asimismo, que el programador cuenta con **git** de antemano. En caso contrario, puede acceder a las instrucciones para su instalación en <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>.

Requisitos de *hardware*

La arquitectura ha de desplegarse, como es lógico, en un servidor físico. Existen dos alternativas para ello:

- **Bare metal**: con esta opción, se dispone de un servidor dedicado en el que es necesario instalar todas las dependencias (listadas a continuación) manualmente.

- ***Cloud provider***: esta es la opción de despliegue recomendada. En este caso, contratamos un servicio *cloud* con un *cloud provider* concreto, como puede ser Google Cloud, Amazon Web Services (AWS), o Microsoft Azure. Estos *cloud providers* disponen de servicios concretos para el despliegue de aplicaciones en Kubernetes (como Google Kubernetes Engine, GKE, en el caso de Google Cloud). Esto facilita en gran medida la labor de despliegue.

Despliegue en Google Kubernetes Engine (GKE)

Google Kubernetes Engine, GKE, ha sido la plataforma con la que se ha trabajado en el desarrollo del proyecto. Es, por tanto, la **opción que más garantías de correcto funcionamiento ofrece**.

Google Cloud ofrece actualmente 300\$ de crédito gratuitos durante 90 días la primera vez que nos registramos en el servicio.

Para la creación de una cuenta de Google Cloud, y la creación de un *clúster* de Kubernetes en GKE, se deben seguir los pasos recogidos en la siguiente página: <https://cloud.google.com/kubernetes-engine/docs/quickstart>.

La configuración recomendada a la hora de crear un nuevo *clúster* es la siguiente:

- **Tipo de ubicación**: zonal.
- **Zona principal**: europe-west2-a.
- **Canal de versiones**: Canal rápido.
- **Versión**: 1.18.12-gke.1206.
- **Nodos**:
 - **Número de nodos**: 1.
 - **Tipo de máquina**: e2-standard-4 (4 vCPUS, 16GB de RAM).
 - **Tipo de imagen**: Container-Optimized OS con Docker (cos).

Además, se debe crear un Disco Persistente de GCE, siguiendo los pasos listados en <https://cloud.google.com/compute/docs/disks/add-persistent-disk?hl=es-419>. En este disco, se almacenarán los modelos de generación de lenguaje que empleará JIZT.

Actualmente, se hace uso del modelo **t5-large** implementado por Hugging Face, el cual se puede descargar a través del siguiente link: <https://huggingface.co/t5-large>. Este modelo se divide a su vez en el *tokenizer*, encargado de

la codificación, y el modelo propiamente dicho, el cual se encarga de la generación de los resúmenes.

Una vez tenemos el modelo descargado localmente, los cargaremos en el Disco Persistente creado anteriormente. Para ello, nos conectamos a través de `ssh` o desde la Consola de Google Cloud (más información en <https://cloud.google.com/compute/docs/instances/connecting-to-instance?hl=es-419#console>), y copiamos el modelo a las siguientes rutas:

- El `tokenizer` debe estar alojado en el directorio `/home/text_encoder/models`.
- El modelo debe situarse en `/home/text_summarizer/models`.

Finalizados estos pasos, estamos en disposición de instalar las dependencias de JIZT.

Instalación del Operador de PostgreSQL

Este operador nos permite desplegar PostgreSQL en Kubernetes. Para su instalación y despliegue, se deben seguir los pasos recogidos en <https://access.crunchydata.com/documentation/postgres-operator/4.5.1/installation/other/google-cloud-marketplace>.

Una vez instalado el operador, nos conectaremos al *clúster* de PostgreSQL creado del siguiente modo: <https://access.crunchydata.com/documentation/postgres-operator/4.5.1/tutorial/connect-cluster>.

Una vez conectados al *clúster*, debemos ejecutar el *script* SQL situado en el directorio del proyecto de JIZT: `src/services/postgres/schemas.sql`. Este *script* proveerá la base de datos con la estructura de tablas iniciales.

Finalmente debemos crear un *secret* de Kubernetes conteniendo los detalles de conexión a la base de datos, esto es, el nombre de usuario y la contraseña especificados a la hora de instalar el *clúster* de PostgreSQL. Para ello, se deben seguir los pasos indicados en <https://kubernetes.io/docs/concepts/configuration/secret/#creating-a-secret>. El nombre del secreto debe ser “`pg-dispatcher`”.

Instalación de JIZT

La instalación de los prerrequisitos es, probablemente, la parte más complicada de todo el proceso de instalación.

Por suerte, para la instalación de JIZT, hemos creado un *script* bash que automatiza el despliegue de Strimzi (Kafka en Kubernetes) y de los componentes propios de JIZT¹.

Dicho *script* se encuentra en el directorio `src/helm/setup/setup.sh`. Si se ejecuta sin especificar ninguna opción, se instalan los componentes tanto de Strimzi como de JIZT. El *script* también admite la opción `-p` o `--partial` para realizar una instalación parcial únicamente de los componentes de JIZT. Esto es útil para el caso de que ya hayamos instalado previamente los componentes relativos a Strimzi.

Podemos comprobar la correcta instalación de los componentes ejecutando: `kubectl get deployment`. Deberían aparecer los siguientes *deployments*:

- `dispatcher-deployment`
- `jizt-cluster-entity-operator`
- `t5-large-text-encoding-deployment`
- `t5-large-text-summarization-deployment`
- `text-postprocessing-deployment`
- `text-preprocessing-deployment`

Fichero de configuración de Kubernetes

A través del fichero `src/helm/jizt/values.yaml`, se pueden configurar los parámetros de instalación y despliegue de JIZT en Kubernetes.

A continuación, se detallan los parámetros que este fichero permite modificar:

- **`namespace`**: *namespace* (espacio de nombres) en el que se instalarán los componentes de JIZT. Para más información sobre los *namespaces* de Kubernetes, se puede consultar <https://kubernetes.io/es/docs/concepts/overview/working-with-objects/namespaces>.
- **`ingress`**: configuración de Ingress. Para más información sobre el componente Ingress de Kubernetes, se puede visitar <https://kubernetes.io/docs/concepts/services-networking/ingress>.
- **`replicaCount`**: número de réplicas global para todos los *deployments*. Es decir, si por ejemplo se configura con 2 réplicas, todos los *deployments* relativos a JIZT tendrán 2 *pods*.

¹ Este *script* solo se ha probado en GNU/Linux. No se asegura su correcto funcionamiento en otros sistemas operativos.

- **autoscaling**: permite activar o desactivar el auto-escalado, de modo que en momentos de mayor carga, el número de réplicas aumente automáticamente. Desactivado por defecto.
- **dispatcher**: configuración relativa al *deployment* del *Dispatcher*.
 - **name**: nombre del microservicio.
 - **ports**: puertos relativos al microservicio.
 - **svc**: puerto del *service* asociado al *deployment*.
 - **container**: puerto de la aplicación que se ejecuta en la imagen Docker que emplean los *pods* del *deployment*.
 - **image**: imagen Docker que ejecutan los *pods* del *deployment*.
- Los campos del **dispatcher** son comunes al resto de microservicios, esto es **textPreprocessor**, **t5LargeTextEncoder**, **t5LargeTextSummarizer**, y **textPostprocessor**.
- **t5LargeTextEncoder** y **t5LargeTextSummarizer**: además de los campos comunes a **dispatcher**, estos microservicios disponen de la siguiente configuración:
 - **volumeMounts**: configuración relativa al Volumen Persistente en el que se almacenan los modelos de *tokenización* y generación de resúmenes.
 - **modelsMountPath**: ruta (directorio) en la que se encuentran almacenados los modelos.
 - **tokenizerPath**: ruta del *tokenizer*. Esto es, la ruta absoluta del *tokenizer* será **modelMountPath/tokenizerPath**.
 - **modelPath**: ruta del modelo generador de resúmenes. Esto es, la ruta absoluta del modelo generador será **modelMountPath/modelPath**.
- **t5LargeTextSummarizer**: además de los campos mencionados anteriormente, este microservicio permite establecer los parámetros por defecto para la generación de resúmenes a través del campo **params**. El significado de cada uno de los parámetros se recoge en la sección correspondiente a la [Documentación de la API REST](#).
- **postgres**: configuración referente a la base de datos PostgreSQL.
 - **host**: *host* de PostgreSQL.

- **dbName**: nombre de la base de datos.
- **secret**: información relativa al *secret* de Kubernetes que contiene el nombre de usuario y la contraseña de PostgreSQL.
- **modelsPV**: configuración relativa al Volumen Persistente en el que se alojan los modelos.
- **kafka**: configuración de Kafka. Para más información, se puede consultar la documentación de Strimzi en <https://strimzi.io/docs/0.5.0>.
 - **name**: nombre del *clúster* de Kafka.
 - **namespace**: espacio de nombres en el que se desplegarán los componentes de Kafka.
 - **version**: versión de Kafka.
 - **replicas**: número de réplicas de los componentes de Kafka.
 - **resources**: límites de recursos para los componentes de Kafka.
 - **memoryRequests**: memoria solicitada por los componentes de Kafka.
 - **memoryLimits**: límites de memoria que los componentes de Kafka pueden solicitar.
 - **config**: otras configuraciones.
 - **autoCreateTopics**: permitir a Kafka crear *topics* de manera automática. Desactivado por defecto.
 - **messageMaxBytes**: máximo tamaño que un mensaje puede tener. Por defecto 1MB.
 - **topicReplicationFactor**: factor de replicación de los *topics*.
 - **zookeeper**: configuración relativa al *zookeeper* de Kafka. Se puede encontrar más información relativa a este componente en <https://kafka.apache.org/documentation/#zk>.
- **topics**: configuración relativa a los *topics* de Kafka. Para más información acerca de los *topics* de Kafka, visitar https://kafka.apache.org/documentation/#intro_topics.
 - **partitions**: número de particiones (común a todos los *topics*).
 - **replicas**: número de réplicas (común a todos los *topics*).
 - **retentionMs**: máximo tiempo de retención de los mensajes en los *topics*. Si tras este tiempo, el mensaje no ha sido consumido, se descarta. Por defecto fijado a 10 minutos.

Especificación de la API REST

En esta sección se incluye documentación detallada sobre la API REST de JIZT. Más concretamente, se especifican los *endpoints* existentes, así como las operaciones HTTP permitidas, y la estructura tanto de las peticiones HTTP, como de las respuestas del *backend*.

Nota: en <https://docs.api.jizt.it> se puede encontrar la documentación en línea referente a la API REST. Dado que dicha documentación está en inglés, recogemos su traducción al español. El usuario puede referirse a cualquiera de las dos documentaciones, ya que son idénticas.

Operación POST - Solicitar resumen

Endpoint: <https://api.jizt.it/v1/summaries/plain-text>

Petición HTTP:

- Content-Type: application/json
- Atributos del JSON del cuerpo de la petición²:
 - **source** (*string*) **obligatorio**: el texto a resumir.
 - **model** (*string*): el modelo a emplear para la generación del resumen. Valores permitidos: "t5-large".
 - **params** (*object*): parámetros del resumen.
 - **relative_max_length** (*number <float>*): longitud máxima del resumen a generar, relativa a la longitud del texto original. Por ejemplo, un valor de 0.4 significa que la longitud del resumen será, como máximo, un 40 % de la longitud del texto original.
 - **relative_min_length** (*number <float>*): longitud mínima del resumen a generar, relativa a la longitud del texto original.
 - **do_sample** (*boolean*): usar muestreo o no. Si se establece como falso, se emplea búsqueda voraz³.
 - **early_stopping** (*boolean*): detener la búsqueda si se terminan **num_beams** frases o no.

² Todos los atributos no marcados como «obligatorio» se pueden omitir. En ese caso, tomarán valores por defecto.

³ Para más información sobre los distintos tipos de generación de lenguaje, referirse al capítulo de «Conceptos Teóricos» de la Memoria

- **num_beams** (*integer <int32>*): número de *beams* en la *beam_search*. Si se fija como 1, no se lleva a cabo *beam search*. Cuanto mayor sea el número, más tiempo tardará en generarse el resumen, pero su calidad será probablemente mejor.
- **temperature** (*number <float>*): valor utilizado para modular las probabilidades del siguiente *token*.
- **top_k** (*integer <int32>*): número de *tókenes* con mayor probabilidad a considerar al realizar muestreo *top-k*.
- **top_p** (*number <float>*): si se fija a valores menores de 1, solo se consideran los *tókenes* más probables cuya suma de probabilidades es igual o mayor a *top_k*.
- **repetition_penalty** (*number <float>*): penalización aplicada a la repetición de *tókenes*. Si se establece a 1.0 no se aplica penalización.
- **length_penalty** (*number <float>*): penalización aplicada de manera exponencial a la longitud de las secuencias generadas. Si se establece a 1.0 no hay penalización. Valores por debajo de 1.0 resultarán en resúmenes más cortos.
- **no_repeat_ngram_size** (*integer <int32>*): longitud de los *n-gramas* que se pueden repetir una única vez. Por ejemplo, si se fija a 2, las palabras “Ingeniería Informática” solo podrán aparecer una vez en el resumen.
- **language** (*string*): el idioma del texto. Valores permitidos: "en".

Respuestas

Status code: 200 OK

- *Response schema: application/json*
- Atributos del JSON del cuerpo de la respuesta:
 - **summary_id** (*string*): el *id* del resumen.
 - **started_at** (*string <date-time>*): el instante de tiempo en el que se solicitó por primera vez un resumen con un texto de entrada, parámetros, y modelo específicos.
 - **ended_at** (*string <date-time>*): el instante de tiempo en el que se finalizó por primera vez un resumen con un texto de entrada, parámetros, y modelo específicos. Restando este valor al anterior, se puede conocer el tiempo que tomó la generación del resumen.

- **status** (*string*): el estado del resumen. Posibles valores: "preprocessing", "encoding", "summarizing", "postprocessing".
- **output** (*string*): el resumen generado.
- **model** (*string*): modelo empleado para la generación del resumen. Valores posibles: "t5-large".
- **params** (*object*): los parámetros con los que se ha generado el resumen (mismo esquema que en la petición).
- **language** (*string*): el idioma del resumen. Valores posibles: "en".

Status code: 502 Server Error

- *Response schema: text/html; charset=utf-8*

Operación GET - Consultar resumen

Endpoint: `https://api.jizt.it/v1/summaries/plain-text/{summaryId}`

Parámetros del path:

- **summaryId** (*string*) **obligatorio:** el *id* del resumen a consultar.

Respuestas

Status code: 200 OK

- *Response schema: application/json*
- Atributos del JSON del cuerpo de la respuesta: mismos que en la respuesta HTTP 200 de la operación POST.

Status code: 404 Not found

- *Response schema: application/json*
- Atributos del JSON del cuerpo de la respuesta:
 - **errors** (*string*): mensaje de error.

Status code: 502 Server Error

- *Response schema: text/html; charset=utf-8*

Operación GET - Salud del servidor

Endpoint: <https://api.jizt.it/v1/healthz>

Respuestas

Status code: 200 OK

- *Response schema:* text/html; charset=utf-8

Status code: 502 Server Error

- *Response schema:* text/html; charset=utf-8

Pruebas del sistema

Las pruebas de sistema para el *backend* de JIZT comprenden varias pruebas unitarias, de cada una de las etapas en la generación de resumen (pre-procesado, codificación, resumen y post-procesado) y una prueba de integración que prueba el correcto funcionamiento de la API REST y el *backend* en su conjunto.

Prerrequisitos

Para poder ejecutar las pruebas, se requiere tener instalado el *framework* de pruebas para Python `pytest`.

Las instrucciones para su instalación se pueden encontrar en <https://docs.pytest.org/en/stable/getting-started.html>. Si se dispone de pip, se puede instalar fácilmente ejecutando:

```
pip install -U pytest.
```

Ejecución de las pruebas

Para ejecutar todas las pruebas, basta con ejecutar: `pytest`, bien desde el directorio raíz del proyecto, o desde el directorio de `tests`.

Si se quieren ejecutar únicamente alguna de las pruebas en específico, se puede pasar el nombre del fichero al invocar a `pytest`.

D.3. Documentación técnica de la aplicación

En esta sección, se recoge toda la información necesaria para poder trabajar sobre el código fuente de la aplicación, así como compilarlo.

Estructura de directorios

La estructura de directorios de la aplicación de JIZT, a la cual se puede acceder a través de github.com/dmlls/jitz-app, es la siguiente:

- `/`: el directorio raíz contiene todos los ficheros que componen el proyecto Flutter de la aplicación.
- `/lib`: se trata del directorio principal del proyecto, donde se encuentra el código fuente de la aplicación, escrito en Dart.
- `/lib/home`: implementación de la interfaz gráfica y de la lógica de la pantalla principal de la aplicación.
- `/lib/new_text_summary`: implementación de la interfaz gráfica y de la lógica de la pantalla para generar resúmenes a partir de texto.
- `/lib/summaries`: implementación de la interfaz gráfica y de la lógica de la pantalla que lista todos los resúmenes generados por el usuario.
- `/lib/summary`: implementación de la interfaz gráfica y de la lógica de la pantalla de detalle de un resumen.
- `/lib/utils`: utilidades varias.
- `/lib/widgets`: *widgets* de Flutter adicionales.
- `/modules/data`: componentes relativos a la capa de datos de la aplicación.
- `/modules/domain`: componentes relativos a la capa de dominio de la aplicación.
- `/assets/drawables`: imágenes y fuentes de las que hace uso la aplicación.
- `/android`: proyecto de Android.
- `/ios`: proyecto de iOS.
- `/web`: proyecto para *web*.

Manual del programador

Una vez introducida la estructura de directorios, explicaremos cómo podemos compilar la aplicación para las diferentes plataformas, y qué programas necesitamos para ello.

Prerrequisitos

Lo primero que debemos hacer, es instalar Flutter en nuestro ordenador. Flutter está disponible para GNU/Linux, macOS, Windows, y Chrome OS. A diferencia del caso del *backend*, el cual recomendamos instalar y desplegar empleando un dispositivo GNU/Linux, la compilación de la *app* se puede llevar a cabo en el sistema operativo que el programador considere oportuno.

Las instrucciones para instalar Flutter en los distintos sistemas operativos se pueden encontrar en <https://flutter.dev/docs/get-started/install>.

Se considera, asimismo, que el programador cuenta con `git` de antemano. En caso contrario, puede acceder a las instrucciones para su instalación en <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>.

Entorno de desarrollo

Para conseguir la mejor experiencia de desarrollo en Flutter, se recomienda emplear uno de los siguientes tres IDEs:

- **Android Studio:** se trata del IDE que proporciona oficialmente Google para el desarrollo de aplicaciones Android [19].
- **IntelliJ IDEA:** es otro de los IDEs recomendados para el desarrollo en Flutter, en este caso desarrollado por JetBrains. Como curiosidad, Android Studio se construyó sobre este IDE [20].
- **Visual Studio Code:** este es el tercer IDE recomendado, desarrollado por Microsoft [21].

Estos tres IDEs cuentan con sendos *plugins* para la programación en Dart en el contexto de Flutter. Se puede encontrar más información de como configurar cada uno de los IDEs a través de la documentación oficial de Flutter: <https://flutter.dev/docs/development/tools/android-studio>.

Compilación de la aplicación

La compilación de la *app* requiere únicamente de tres comandos:

- Clonar el repositorio desde GitHub:
`git clone https://github.com/dmlls/jizt-app.git`
- Descargar los paquetes requeridos por la aplicación:
`flutter pub get`

- Compilar la aplicación para la plataforma deseada:

```
flutter build [appbundle|apk|ios|web|linux|windows|macos]
```

- **appbundle**: compila a un App Bundle de Android.
- **apk**: compilar a una APK de Android.
- **ios**: compilar una aplicación para iOS. Solo se puede ejecutar desde MacOS.
- **web**: compilar para *web*⁴.
- **linux**: compilar para GNU/Linux⁵. Solo se puede ejecutar desde Linux.
- **windows**: compilar para Windows. Solo se puede ejecutar desde Windows.
- **macos**: compilar para MacOS. Solo se puede ejecutar desde MacOS.

Con estos tres sencillos pasos, obtendremos un binario de la aplicación para la plataforma deseada.

D.4. Contribuir al proyecto

A fin de organizar las posibles contribuciones de la comunidad al proyecto JIZT, se ha creado un documento que contiene algunas pautas y recomendaciones a la hora de contribuir.

Dicho documento es válido tanto para el caso del *backend*, como de la aplicación. Se ha escrito en un registro informal, a fin de animar todo tipo de contribuciones, por muy pequeñas que parezcan, ya que no es necesario tener conocimientos de programación para contribuir al proyecto.

El documento se puede consultar en línea a través de <https://github.com/dmlls/jitz/blob/main/CONTRIBUTING.md>,

En cualquier caso, lo incluimos en este documento también, por considerarlo de interés:

⁴ El soporte de Flutter para *web* se encuentra aún en fase *beta* [22]. No se recomienda su uso en producción.

⁵ El soporte de Flutter para escritorio (GNU/Linux, MacOS y Windows) se encuentra en fase *alfa* [23]. No se recomienda su uso fuera del entorno de desarrollo.

Contribuir a JIZT

En primer lugar, queremos darte las gracias por tu interés en contribuir a JIZT. Las contribuciones de la comunidad son esenciales para hacer de JIZT un proyecto aún mejor. Queremos que dichas contribuciones sean lo más fáciles de llevar a cabo como sea posible. Existen una serie de pautas que pedimos que sigan aquellos que quieran contribuir al proyecto, a fin de mantener una buena organización.

¿Cómo contribuir al proyecto?

Existen numerosas formas en las que puedes contribuir al proyecto. Por ejemplo:

- Ayudando a mantener, corregir errores o traducir la documentación para que más personas puedan conocer acerca de JIZT.
- Enviando reportes de errores o cualquier otro tipo de problemas.
- Escribiendo *patches* para corregir *bugs* conocidos.
- Aportando ideas para nuevas funcionalidades.
- Desarrollando código que implemente nuevas funcionalidades solicitadas.
- Contándole a tus amigos lo increíble que es JIZT.
- Escribiendo *posts* o *blogs* acerca de JIZT, o tutoriales para que otros aprendan a usarlo.

¿Cómo solicitar incluir código al proyecto?

Para contribuir con tu código al proyecto, crea primero una *Pull request* y realiza las modificaciones o adiciones pertinentes.

Alternativamente, puedes hacer un *Fork* manual, realizar las modificaciones, y más tarde abrir una *Pull request*.

Por favor, no olvides explicar de manera detallada y clara los cambios implementados en la *Pull request*. De lo contrario, no podremos considerar tu contribución.

¿Cómo reportar errores?

Los reportes de errores nos ayudan a mejorar. Si has encontrado un error, puedes abrir una *Issue* utilizando la [plantilla](#) habilitada para ello.

Por favor, revisa primero que el error que has encontrado no ha sido reportado previamente.

Nota importante: si encuentras una vulnerabilidad de seguridad, NO abras una *Issue*. Envíanos un mensaje a jizt@diegomiguel.me.

¿Cómo solicitar nuevas funcionalidades?

Si tienes una idea genial de una funcionalidad que crees que JIZT debería tener, ¡no dudes en contárnosla! Para ello, crea una [nueva discusión](#) con el título “[Feature Request]” seguido de tu idea.

Por favor, revisa primero que la funcionalidad deseada no ha sido sugerida previamente.

Trataremos de atender todas las peticiones, pero trata de ser paciente, aún somos un equipo pequeño :)

¿Te ha quedado alguna duda? ¡Escríbenos a jizt@diegomiguel.me! Estaremos encantados de ayudarte.

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección se recogen los requisitos que requiere nuestra aplicación, así como los detalles para la instalación y uso de la misma por el usuario final¹.

E.2. Requisitos de usuarios

Para la versión **Android**, se deben cumplir los siguientes requisitos:

- Mínimo 18 MB de espacio de almacenamiento libre. El paquete de instalación tiene un tamaño de 6,3 MB, y una vez instalada ocupa 16,36 MB. No obstante, ese tamaño aumentará ligeramente según se vayan almacenando resúmenes.
- Versión de Android igual o superior a la 4.1 (*JellyBean* - API 16).

En el caso de **iOS**:

- Mínimo 20 MB de espacio de almacenamiento libre. En este caso, el peso del paquete es de 6,9 MB, y una vez instalada ocupa 17,2 MB.
- Versión de iOS 8 o superior.

¹ Por ahora, no se incluyen los detalles referentes a la versión de escritorio, dado que el soporte de Flutter para estas plataformas está aún en fase *alfa* [23].

Y en el caso de la versión ***web***:

- Los navegadores *web* soportados son Google Chrome², Mozilla Firefox, Safari o Edge.

En todos los casos se requiere conexión a Internet para generar nuevos resúmenes (los resúmenes ya generados se pueden consultar *offline*). La *app* de JIZT no requiere de permisos adicionales en ninguna de las plataformas.

La aplicación solo está disponible por el momento en inglés, dado que este es el único lenguaje soportado actualmente por el servicio de generación de resúmenes.

E.3. Instalación

En esta sección detallamos el proceso de instalación en las diferentes plataformas.

Android

Instalación recomendada

Se recomienda que el usuario instale la aplicación desde Google Play.

Para ello, simplemente basta con buscar la aplicación «JIZT AI Summarization» y pulsar en «Instalar».

²Hemos probado la *app* en navegadores basados en Chromium, como Brave, y también parecen funcionar.

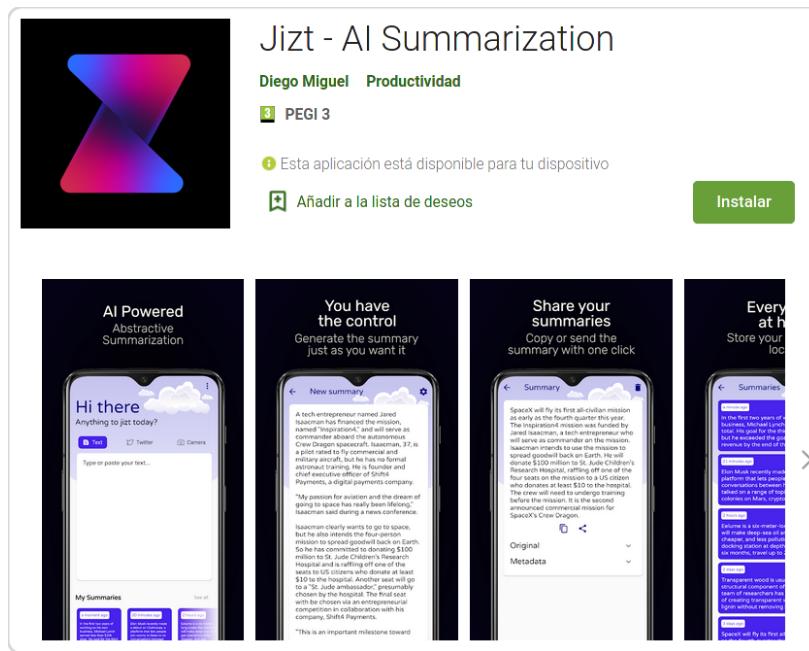


Figura E.1: Instalar JIZT desde Google Play.

iOS

Por el momento, la aplicación no ha sido publicada en la App Store, e iOS no proporciona ninguna manera oficial para la instalación de aplicaciones desde fichero³.

Por tanto, se recomienda a los usuarios que accedan desde su navegador móvil a la versión *web* de JIZT (ver siguiente sección).

Web

Se puede acceder a la aplicación directamente a través de app.jizt.it, sin ser necesario realizar ninguna instalación.

³ Como aclaración al margen de Manual de Usuario, la aplicación no ha sido publicada en la App Store por su elevado precio (99\$ al año por la cuenta de desarrollador, frente a los 25\$ de por vida, en el caso de Play Store).

E.4. Manual del usuario

Una vez instalada la aplicación, el usuario está en disposición de comenzar a utilizarla. El funcionamiento en interfaz de la aplicación en las diferentes plataformas es homogéneo, por lo que todo lo explicado a continuación es válido para cualquiera de ellas.

Generar un nuevo resumen

La generación de resúmenes se trata de una de las funciones principales de la aplicación.

Los pasos que debemos seguir para generar un nuevo resumen son los siguientes:

1. En la pantalla de inicio, pulsar sobre el campo de texto central, el cual contiene escrito «*Type or paste your text*» (en español, «Escribe o pega tu texto»).
2. Escribir el texto o pulsar en el ícono de la esquina superior derecha, el sirve para pegar el texto desde el portapapeles.
3. Pulsar en «*Summarize*» («resumir»).
4. Se mostrará una barra que simboliza que el resumen está siendo generado.
5. Una vez completado el resumen, se mostrará una nueva pantalla con el resumen.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/1-generar-nuevo-resumen.mp4>.

Ajustar la longitud del resumen a generar

JIZT nos permite establecer la longitud deseada del resumen generado. Esta longitud se establece como un porcentaje de la longitud del texto original.

Para ajustar la longitud del resumen que vamos a generar, debemos seguir los dos primeros pasos indicados en la sección «[Generar un nuevo resumen](#)».

Una vez en la pantalla de nuevo resumen, podemos ajustar el *slider* que aparece en la parte inferior de la pantalla, estableciendo la longitud mínima y máxima de nuestro resumen.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/2-ajustar-longitud.mp4>.

Ver todos los resúmenes generados

La aplicación muestra en la pantalla principal una vista previa de los últimos resúmenes generados en forma de lista deslizable. Pulsando sobre cualquiera de ellos, se accede a los detalles del mismo.

Si se quieren ver todos los resúmenes, se puede pulsar en «*See all*» («ver todos»). Se mostrará una nueva pantalla en la que aparece una lista con todos los resúmenes, ordenados temporalmente de más recientes a más antiguos. Se puede pulsar sobre cualquiera de ellos para obtener más detalles.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/3-ver-todos-resúmenes.mp4>.

Borrar un resumen

Para borrar un resumen, se puede pulsar en el símbolo que aparece en la esquina superior derecha en la pantalla de «*Summary*» (resumen).

Se puede acceder a esta pantalla de tres formas diferentes:

1. Tras generar un resumen, se muestra dicha pantalla por defecto.
2. Haciendo *click* en cualquiera de los resúmenes que aparecen en la parte inferior de la pantalla principal.
3. Pulsando en «*See all*» («ver todos») y haciendo *click* en cualquiera de los resúmenes.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/4-borrar-resumen.mp4>.

Copiar un resumen

Para copiar un resumen, se debe estar en la pantalla de «*Summary*» (resumen). Para acceder a esta pantalla, seguir cualquiera de las alternativas listadas en la sección “[Borrar un resumen](#)”.

Una vez en esta pantalla, se debe pulsar en el siguiente icono:

Tras pulsar dicho icono, el texto se habrá copiado al portapapeles de nuestro dispositivo.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/5-copiar-resumen.mp4>.

Compartir un resumen

Para copiar un resumen, se debe estar en la pantalla de «*Summary*» (resumen). Para acceder a esta pantalla, seguir cualquiera de las alternativas listadas en la sección “[Borrar un resumen](#)”.

Una vez en esta pantalla, se debe pulsar en el siguiente icono:

A continuación, se mostrará una lista de aplicaciones a través de las cuales se puede compartir el resumen.

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/6-compartir-resumen.mp4>.

Ver el texto a partir del cual se generó un resumen

Para ver el texto original de un resumen, se debe estar en la pantalla de «*Summary*» (resumen). Para acceder a esta pantalla, seguir cualquiera de las alternativas listadas en la sección «[Borrar un resumen](#)».

Una vez en dicha pantalla, se debe pulsar sobre «*Original*».

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/7-ver-original.mp4>.

Obtener más información acerca de un resumen

Para obtener más información de un resumen, se debe estar en la pantalla de «*Summary*» (resumen). Para acceder a esta pantalla, seguir cualquiera de las alternativas listadas en la sección «[Borrar un resumen](#)».

Una vez en dicha pantalla, se debe pulsar sobre «*More info*» («Más información»).

Se puede visualizar un vídeo que recoge el proceso en <https://github.com/dmlls/jizt/blob/main/docs/video-tutorials/8-m%C3%A1s-info.mp4>.

Generar un resumen a partir de un documento

Por el momento, esta opción no está disponible. No obstante, pronto será implementada.

Generar un resumen a partir de una imagen

Por el momento, esta opción no está disponible. No obstante, pronto será implementada.

Apéndice F

Experimentos sobre el algoritmo de división de texto

En la sección de «Codificación del texto» del capítulo de «Conceptos Teóricos» de la Memoria, se introduce el problema que presentan muchos de los modelos de generación del lenguaje, entre ellos el modelo T5, el cual utilizamos.

Dicho problema deriva del hecho de que estos modelos admiten un número de *tókenes* de entrada máximo, 512 en el caso del modelo T5.

Por tanto, desarrollamos un algoritmo que divide el texto en fragmentos, sin que ninguna frase quede partida, y siendo las diferencias entre el número de *tókenes* en cada una de las divisiones tan pequeñas como sea posible.

Para ello, primero dividimos el texto de manera voraz, de forma que se van añadiendo frases a una subdivisión siempre que estas quepan sin ser fragmentadas.

Esto puede resultar en que la última subdivisión contenga un número de frases muy inferior a la del resto de subdivisiones. Para solucionar este problema, se realiza un «balanceo» de las subdivisiones, moviendo frases de unas a otras, pero conservando su orden.

La [Figura F.1](#) muestra el proceso de manera visual a través de un ejemplo:



Figura F.1: Proceso de balanceo de las subdivisiones generadas de forma voraz. Ejemplo con una longitud máxima de 100 *tókenes*.

En las tablas recogidas en las siguientes páginas, se muestra el número de *tókenes* por subdivisión resultante de aplicar la división voraz, y el número de *tókenes* por subdivisión una vez realizado el balanceo. Para los textos de los experimentos, se emplearon hilos de Twitter, teniendo el más corto 191 palabras, y artículos extraídos de Internet, el más largo de los cuales contiene 46.911 palabras.

Las diferencias entre el número de *tókenes* antes y después de realizar la división son claras, lo que indica el correcto funcionamiento del algoritmo propuesto. Además, el incremento de tiempo que introduce el balanceo es, por lo general, mínimo¹.

¹ Posteriormente a la realización de este experimento, se optimizó notablemente el rendimiento global del algoritmo para el modelo T5. Para más información, se puede consultar la siguiente *Issue* en el repositorio del proyecto en GitHub: <https://github.com/dmlls/jitz/issues/97>.

Codificación con división de texto (hilos reales de Twitter)			
Palabras/ <i>tókenes</i>		Voraz	Balanceado
191 280 (4 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[280] 0,0 5,23 ms	[280] 0,0 5,25 ms
394 534 (11 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[506, 28] 239,0 8,15 ms	[273, 261] 6,0 8,21 ms
646 906 (16 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[504, 402] 51,0 13,4 ms	[448, 458] 5,0 13,8 ms
1.089 1.417 (23 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[488, 477, 452] 15,06 26,10 ms	[478, 487, 452] 14,84 25,70 ms
1.536 2.069 (32 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[509, 508, 507, 501, 44] 184,92 41 ms	[437, 419, 409, 407, 397] 13,54 41,2 ms
1.871 4.531 (43 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[409, 426, ..., 475, 414] 39,70 34,7 ms	[409, 426, ..., 422, 467] 32,11 35,4 ms
2.060 3.255 (51 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[501, 497, ..., 402, 509] 54,40 26,1 ms	[501, 497, ..., 402, 509] 37,65 26,7 ms
3.753 5.293 (103 tweets)	Tokens/subdiv. Desv. estándar Tiempo	[504, 505, ..., 503, 386] 36,75 62,1 ms	[504, 505, ..., 469, 437] 19,72 62,9 ms

Tabla F.1: Resultados en la codificación de hilos de Twitter reales. Desviación estándar calculada sobre el número de *tókenes* por subdivisión.

Codificación con división de texto (textos muy largos)			
Palabras/ <i>tókenes</i>	Voraz	Balanceado	
5.022	Tokens/subdiv. [498, 499, ... , 497, 77]	[498, 499, ... , 418, 396]	
8.051	Desv.		
	estándar 99,88	39,21	
	Tiempo 118 ms	120 ms	
10.058	Tokens/subdiv. [498, 499, ... , 496, 415]	[498, 499, ... , 461, 484]	
15.761	Desv.		
	estándar 23,54	15,52	
	Tiempo 250 ms	251 ms	
20.203	Tokens/subdiv. [498, 499, ... , 481, 311]	[498, 499, ... , 425, 420]	
30.989	Desv.		
	estándar 29,51	18,95	
	Tiempo 470 ms	488 ms	
46.911	Tokens/subdiv. [498, 499, ... , 505, 121]	[498, 499, ... , 392, 385]	
69.772	Desv.		
	estándar 35,05	17,01	
	Tiempo 1,08 s	1,11 s	

Tabla F.2: Resultados en la codificación de textos muy largos. Desviación estándar calculada sobre el número de *tókenes* por subdivisión.

Por último, queremos mencionar que, al comienzo del proyecto, escribimos diferentes *notebooks* de Jupyter que explican cómo llegamos a ciertas conclusiones a la hora de realizar el pre-procesado y post-procesado de los textos, así como un pequeño estudio comparativo realizado con dos modelos de generación de textos. Se puede acceder a dichos *notebooks* a través de <https://github.com/dmlls/jitz/tree/main/notebooks>.

Bibliografía

- [1] Hugging Face. *Transformers*. Sep. de 2020. URL: <https://huggingface.co/transformers/index.html>. Último acceso: 08/02/2021.
- [2] Kubernetes. *Ingress*. Feb. de 2021. URL: <https://kubernetes.io/docs/concepts/services-networking/ingress>. Último acceso: 08/02/2021.
- [3] Google Cloud. *Google Kubernetes Engine (GKE)*. Oct. de 2020. URL: <https://cloud.google.com/kubernetes-engine>. Último acceso: 08/02/2021.
- [4] Flutter. *Flutter - Hermosas apps nativas en tiempo record*. Sep. de 2020. URL: <https://esflutter.dev>. Último acceso: 08/02/2021.
- [5] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, abr. de 2010. ISBN: 0984521402.
- [6] Juan Antonio Pérez López. *Fundamentos de la dirección de empresas*. RIALP, abr. de 2018. ISBN: 9788432149184.
- [7] Agencia Tributaria. *Cuadro informativo tipos de retención aplicables*. 2021. URL: https://www.agenciatributaria.es/static_files/Sede/Programas_ayuda/Retenciones/2021/CUADRO_TIPOS_RETENC_IRPF21.doc. Último acceso: 08/02/2021.
- [8] Agencia Estatal Boletín Oficial del Estado. *Boletín Oficial del Estado, Núm. 341*. Dic. de 2020. URL: <https://www.boe.es/eli/es/1/2020/12/30/11/dof/spa/pdf>. Último acceso: 08/02/2021.
- [9] The GNU Operating System y the Free Software Movement. *GNU General Public License v3.0*. Jun. de 2007. URL: <https://www.gnu.org/licenses/gpl-3.0.en.html>. Último acceso: 09/02/2021.

- [10] The GNU Operating System y the Free Software Movement. *GNU Free Documentation License*. Nov. de 2008. URL: <https://www.gnu.org/licenses/fdl-1.3.html>. Último acceso: 09/02/2021.
- [11] Creative Commons. *CC BY-SA 4.0*. Feb. de 2019. URL: <https://creativecommons.org/licenses/by-sa/4.0/deed.es>. Último acceso: 12/02/2021.
- [12] Codit. *The Routing Slip Pattern*. Jul. de 2019. URL: <https://www.codit.eu/blog/the-routing-slip-pattern>. Último acceso: 10/02/2021.
- [13] Wikipedia - La enciclopedia libre. *Arquitectura dirigida por eventos*. Ago. de 2020. URL: https://es.wikipedia.org/wiki/Arquitectura_dirigida_por_eventos. Último acceso: 10/02/2021.
- [14] Apache Software Foundation. *Apache Kafka*. Nov. de 2020. URL: <https://kafka.apache.org>. Último acceso: 10/02/2021.
- [15] Robert Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education, 2015. ISBN: 9780134494166.
- [16] Didier Boelens. *Reactive Programming - Streams -BLoC*. Ago. de 2018. URL: <https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc>. Último acceso: 10/02/2021.
- [17] Felix Angelov. *Bloc Package*. Dic. de 2020. URL: <https://pub.dev/packages/bloc>. Último acceso: 10/02/2021.
- [18] Helm. *Helm - The package manager for Kubernetes*. Dic. de 2020. URL: <https://helm.sh>. Último acceso: 11/02/2021.
- [19] Wikipedia - La enciclopedia libre. *Android Studio*. Ene. de 2021. URL: https://en.wikipedia.org/wiki/Android_Studio. Último acceso: 11/02/2021.
- [20] Wikipedia - La enciclopedia libre. *IntelliJ IDEA*. Ene. de 2021. URL: https://en.wikipedia.org/wiki/IntelliJ_IDEA. Último acceso: 11/02/2021.
- [21] Wikipedia - La enciclopedia libre. *Visual Studio Code*. Ene. de 2021. URL: https://en.wikipedia.org/wiki/Visual_Studio_Code. Último acceso: 11/02/2021.
- [22] Flutter. *Web FAQ*. Oct. de 2020. URL: <https://flutter.dev/docs/development/platform-integration/web>. Último acceso: 11/02/2021.
- [23] Flutter. *Desktop support for Flutter*. Feb. de 2021. URL: <https://flutter.dev/desktop>. Último acceso: 11/02/2021.



Atribución-CompartirIgual 4.0 Internacional
(CC BY-SA 4.0)