Facultad

de

Ciencias

HERRAMIENTA WEB PARA LA REVISIÓN LIGERA DE TEXTOS

(WEB TOOL FOR QUICK PROOFREADING)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Carlos Lavín Ruiz

Director: Alfonso de la Vega Ruiz

Junio – 2025

Resumen

Este Trabajo Fin de Grado (TFG) presenta el desarrollo de QuickED, una aplicación web para la edición rápida y comparativa de textos. La herramienta permite a los usuarios introducir y modificar textos, visualizando de forma clara las diferencias entre versiones mediante un sistema de resaltado eficiente. La aplicación está desarrollada con una arquitectura basada en componentes utilizando frameworks modernos y desplegada en abierto mediante Cloudflare Pages, asegurando un acceso rápido y fiable. Durante el desarrollo se implementaron diversas pruebas, incluyendo unitarias, de integración y de aceptación, garantizando la calidad y estabilidad del software. Además, se automatizó el despliegue y las pruebas mediante integración continua con GitHub Actions. La aplicación web resultante de este proyecto se encuentra accesible en la siguiente URL: https://quicked.pages.dev/

Abstract

This Final Degree Project presents the development of QuickED, a web application designed for quick text editing and comparison. The tool allows users to input and modify texts while clearly visualizing differences between versions through an efficient highlighting system. The application is built using a component-based architecture with modern frameworks and is publicly deployed via Cloudflare Pages, ensuring fast and reliable access. Throughout development, various tests, including unit, integration, and acceptance tests, were implemented to guarantee software quality and stability. Additionally, deployment and testing were automated using continuous integration with GitHub Actions. The web application resulting from this project is available at the following URL: https://quicked.pages.dev/

Palabras clave

Herramienta web, edición de textos, resaltado de cambios, revisión colaborativa

KeyWords

Web Tool, text editing, change highlighting, collaborative revision

Índice

Índice de figuras

Índice de tablas

1. Introducción

La revisión de textos es una práctica muy común en varios entornos, tanto académicos como profesiones. Continuamente, estudiantes, investigadores, docentes y trabajadores de diversos ámbitos se ven envueltos en la lectura y corrección de textos redactados por otras personas. Estos textos varían desde correos electrónicos formales hasta fragmentos de artículos científicos, propuestas de proyectos o comunicaciones de relevancia. En cualquier caso, es común que una persona le pida a otra una revisión rápida con el fin de mejorar la redacción, aclarar ideas o asegurar que no haya faltas de ortografía y/o de gramática.

En estas situaciones, la persona que se encarga de revisar el texto suele introducir algunas modificaciones directamente sobre el texto original, con la intención de mejorarlo sin alterar el significado global del mismo. Sin embargo, una vez realizadas las modificaciones, aparece un nuevo desafío: comunicar de forma clara y comprensible cuales han sido los cambios realizados. Además, es importantísimo revisar cuidadosamente las modificados, ya que el editor podría, incluso sin darse cuenta, alterar el significado de ciertas partes del texto, sobre todo si no es un experto en la materia en cuestión. Este paso, que en principio parece trivial, puede convertirse en una tarea costoso si no se dispone de herramientas específicas que permitan visualizar las diferencias fácilmente o gestionas un historial de ediciones.

1.1. Motivación

A pesar de que hay muchas plataformas de edición y colaboración online, ninguna de las soluciones existentes responde de forma adecuada al caso concreto de revisiones ligeras de texto con control de cambios. Herramientas como Google Docs o Microsoft Word ofrecen funcionalidades avanzadas para la edición entre varias personas y el seguimiento de modificaciones, pero su uso conlleva una serie de pasos previos que pueden resultar innecesarios para revisiones puntuales, como puede ser la creación de un nuevo documento, la configuración de persones de acceso, gestión del enlace, etc.

Aunque este proceso es adecuado para grandes revisiones o en grandes proyectos con varias personas involucradas, resulta poco práctico en situaciones en las que la agilidad y eficiencia sean primordiales. La necesidad de una solución directa y fácilmente accesible motiva el desarrollo de este TFG. En este escenario, los usuarios buscan solucione que les permitan editar, comparar textos y compartir el resultado de forma directa, sin necesidad de registrarse, configurar permisos o aprender a utilizar herramientas relativamente complejas.

Por otro lado, las herramientas que sí ofrecen una experiencia más simple y accesible, como los comparadores de online (text-diff[1]), suelen limitarse a mostrar las diferencias entre ambas versiones, sin permitir la edición directa ni el resaltado de cambios. Tampoco permiten, en la mayoría de los casos, compartir

fácilmente el resultado del análisis, lo que dificulta la colaboración o la revisión entre 2 o más personas. Estas herramientas, aunque son útiles en determinados contextos, no ofrecen una solución completa para la revisión colaborativa de textos.

Esta ausencia de soluciones intermedias ha motivado el desarrollo de este proyecto, cuyo objetivo es cubrir esa distancia entre lo excesivamente complejo y lo insuficientemente funcional.

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Grado es diseñar y desarrollar una herramienta web que facilite la revisión ligera de textos, permitiendo a los usuarios realizar modificaciones con control de cambios de forma clara, intuitiva, eficiente y compartir los resultados. Esta herramienta busca simplificar la dinámica habitual en la que una persona revisa un texto ajeno y luego debe comunicar las modificaciones realizadas, una tarea que, en ausencia de los medios adecuados, puede resultar confusa, tediosa o propensa a errores.

La herramienta propuesta, denominada QuickED, se enfocará en ofrecer una experiencia de uso basada en la sencillez y la rapidez. Permitirá que cualquier usuario pueda introducir un texto, modificarlo libremente y visualizar de manera inmediata y de un simple vistazo las diferencias respecto al original. Además, ofrecerá la posibilidad de exportar o compartir el resultado de forma sencilla, lo que facilitará la comunicación entre la persona que revisa el texto y la autora de este.

El propósito final es reducir la carga operativa asociada a este tipo de revisiones, eliminando pasos innecesarios y ofreciendo una solución directa y eficaz que pueda ser utilizada tanto en contextos cotidianos, académicos y/o profesionales. Con esta herramienta se busca mejorar el proceso de colaborar en textos, aprovechando mejor el tiempo dedicado a la revisión y asegurando una comunicación precisa y transparente de los cambios realizados.

2. Tecnologías y herramientas utilizadas

Durante el desarrollo del presente TFG, se ha hecho uso de diversas tecnologías y herramientas que sen utilizado para la creación, prueba, despliegue y gestión del proyecto. A continuación, se describen brevemente cada una de ellas, especificando su propósito y aplicación.

2.1. JavaScript, HTML y CSS

En este proyecto, estos tres lenguajes han constituido la base fundamental del desarrollo, permitiendo programar la lógica de negocio, definir la estructura de la interfaz y personalizar la apariencia para mejorar la experiencia de usuario.

JavaScript[2] es el lenguaje de programación utilizado para implementar la lógica de la aplicación web. HTML (HyperText Markup Language) se emplea para estructurar el contenido y los elementos de las páginas web. CSS (Cascading Style Sheets) es utilizado para definir la presentación visual y el diseño de la interfaz de usuario.

Entre las bibliotecas utilizadas en el desarrollo con JavaScript, destaca el uso de js-diff[3], una librería que permite comparar dos textos y detectar diferencias entre ambas versiones. La elección de js-diff se debe a su facilidad de uso y a su amplio soporte para distintos tipos de comparaciones (palabras, líneas, caracteres).

2.2. Svelte y SvelteKit

Svelte[4] es un framework moderno para el desarrollo de interfaces de usuario. Su principal característica que la diferencia del restos es que traslada gran parte del procesamiento al momento de la compilación, generando código JavaScript altamente optimizado y eficiente para el navegador.

SvelteKit[5], por su parte, es el framework de aplicaciones basado en Svelte, que proporciona funcionalidades avanzadas como enrutamiento, renderizado del lado del servidor (SSR) y compatibilidad con múltiples plataformas de despliegue, permitiendo el desarrollo de aplicaciones web completas de manera eficiente.

Ambas tecnologías se han utilizado conjuntamente para construir y estructurar la interfaz de usuario, gestionar rutas y facilitar el despliegue, lo que ha permitido desarrollar una aplicación web eficiente e interactiva.

Se optó por Svelte y SvelteKit frente a otros frameworks como React o Vue dada si simplicidad, rendimiento y menor complejidad a nivel de configuración. Además, su curva de aprendizaje es más accesible, lo que agilizó el desarrollo inicial.

2.3. Vite y Vitest

Vite[6] es una herramienta de desarrollo y construcción de proyectos frontend que destaca por su rapidez y eficiencia, gracias al uso de módulos ES nativos y a su sistema de recarga en caliente. Permite una experiencia de desarrollo ágil y moderna.

Vitest[7] es un framework de pruebas unitarias diseñado para integrarse con Vite, lo que permite la ejecución eficiente de pruebas y la verificación del correcto funcionamiento del código durante el desarrollo.

Durante el desarrollo, Vite ha servido como entorno principal de trabajo y sistema de compilación, mientras que Vitest se ha utilizado para implementar pruebas automáticas, contribuyendo a asegurar la calidad del código.

La elección de Vite se debe a su velocidad y experiencia de desarrollo moderna frente a herramientas más tradicionales como Webpack[8]. A diferencia de Webpack, Vite proporciona tiempos de arranque casi instantáneos lo que mejora significativamente la comprobación del funcionamiento en entornos de desarrollo. Su integración nativa con SvelteKit también supuso una ventaja clave para mantener el stack tecnológico.

2.4. Git, GitHub y GitHub Actions

Git[9] es un sistema de control de versiones distribuido comúnmente utilizado en el desarrollo de software. Permite gestionar el historial de cambios en el código, facilitando la colaboración entre varios desarrolladores, el seguimiento de versiones y la integración continua en proyectos software.

GitHub es una plataforma web que utiliza Git para alojar repositorios y facilitar la colaboración en proyectos de desarrollo.

GitHub Actions[10] es un sistema de integración y entrega continua (CI/CD) que permite automatizar flujos de trabajo como la ejecución de pruebas y el despliegue de la aplicación.

Estas herramientas se han integrado en el desarrollo para gestionar versiones y automatizar tareas, mejorando así la eficiencia y la fiabilidad del proceso.

2.5. Cloudflare

Cloudflare[11] es una plataforma de servicios en la nube que ofrece soluciones de seguridad, rendimiento y despliegue para aplicaciones web.

En este proyecto, se ha utilizado Cloudflare Pages[12] para alojar la aplicación, lo que permite gestionar sitios web estáticos y aplicaciones de manera segura y optimizada, beneficiándose de la infraestructura global de Cloudflare para mejorar la disponibilidad y tiempos de respuesta.

2.6. Visual Studio Code

Visual Studio Code[13] es un editor de código fuente multiplataforma desarrollado por Microsoft. Ofrece soporte para una amplia variedad de lenguajes de programación, integración con sistemas de control de versiones, depuración y una extensa variedad de extensiones.

Ha sido la herramienta principal de desarrollo, facilitando la escritura y organización del código, la depuración y la integración con el resto de las herramientas utilizadas.

Se eligió Visual Studio Code frente a otros editores como Sublime Text[14] o Atom debido a su amplia comunidad, el soporte constante por parte de Microsoft y a la gran cantidad de extensiones disponibles, muchas de las cuales están específicamente diseñadas para entornos como Svelte o Vite. Su integración con Git y herramientas de depuración también ayudaron a centrar el trabajo en una sola aplicación.

3. Metodología

Desde el inicio del desarrollo de mi TFG, se optó por seguir una metodología ágil. Esta decisión fue consensuada con mi director de TFG, quien también asumió el rol de Product Owner en el proyecto. El objetivo principal era tener un enfoque que facilitara una evolución constante, una corrobación del trabajo continuo y la capacidad de adaptación ante posibles cambios o mejoras.

3.1. Metodologías ágiles vs. fuertemente planificadas

A la hora de gestionar proyectos, especialmente en el ámbito del desarrollo software, existen distintas dos metodologías principales y contrapuestas que permiten organizar, planificar y ejecutar el trabajo.

3.1.1. Metodología fuertemente planificada

Las metodologías fuertemente planificadas están basadas en una planificación exhaustiva inicial. El modelo más representativo de este enfoque es el modelo en cascada, que divide el proyecto en fases secuenciales: análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento. Cada fase debe completarse antes de comenzar la siguiente, y cualquier cambio suele implicar volver atrás, lo cual es costoso en términos de tiempo y recursos.

Esta metodología es útil cuando los requisitos del proyecto están claramente definidos desde el principio y es poco probable que cambien. Sin embargo, su rigidez puede impedir la incorporación de nuevas funcionalidades durante el desarrollo. Además, como no hay entregas del producto hasta fases avanzadas, se corre el riesgo de que el producto final no cumpla del todo con las expectativas del cliente o usuario final.

3.1.2. Metodología ágil

Las metodologías ágiles surgen como una respuesta a las limitaciones de los enfoques tradicionales, especialmente en entornos inciertos o dinámicos. Formalizadas en el Manifiesto Ágil (2001)[15], estas metodologías se fundamentan en varios valores clave:

En primer lugar, se da más importancia a las personas y sus interacciones que a los procesos y herramientas, reconociendo que la comunicación y el trabajo en equipo son esenciales para el éxito del proyecto.

En segundo lugar, se prioriza un producto funcional sobre una documentación exhaustiva, ya que lo principal es entregar valor real de forma continua.

En tercer lugar, se valora la colaboración con el cliente por encima del contrato inicial, lo que implica una participación activa y constante del cliente durante el desarrollo.

Por último, se pone énfasis en la capacidad de respuesta ante el cambio por encima del seguimiento de un plan rígido, lo que permite adaptarse a nuevas funcionalidades o circunstancias.

3.2. Metodología ágil aplicada al TFG

La aplicación de la metodología ágil en mi TFG se organizó principalmente en torno a reuniones semanales con el Product Owner. Estas sesiones tenían como objetivo revisar los avances, comprobar los resultados obtenidos y organizar los futuros trabajos. Este sistema de revisión constante permitió que cada etapa estuviese correctamente finalizada antes de avanzar, evitando así problemas que se acumulan.

El trabajo se dividió en tareas concretas que fueron abordadas de continua. En cada sprint, se priorizaban los aspectos más importantes. Este enfoque también resultó útil para organizar y estructurar la redacción de la memoria para que el resultado final fuese coherente con el desarrollo del proyecto.

Gracias a esta metodología, el desarrollo del TFG se mantuvo flexible, organizado y cuyo objetivo principal fue la obtención de resultados mientras se aprendía.

3.2.1. Fases

La primera fase consistió en la recopilación y análisis de los requisitos principales de la aplicación. Esta etapa se centró en entender con claridad las necesidades funcionales del sistema, así como los objetivos finales del proyecto. Los requisitos extraídos fueron recogidos en un documento compartido al que podían acceder tanto el Product Owner como el equipo de desarrollo (compuesto solo por mí). Este documento fue actualizado a medida que surgieron nuevas funcionalidades o se modificaron las anteriores.

La segunda fase se dedicó al análisis de frameworks de desarrollo adecuados para el proyecto, así como al proceso de aprender y familiarizarme con ellos. Durante este periodo se probaron distintas tecnologías en función de su compatibilidad con los objetivos del sistema, su curva de aprendizaje y su capacidad para facilitar una experiencia de desarrollo ágil. Tras esta etapa de investigación, se optó por el uso de Svelte y SvelteKit

Con los requisitos iniciales definidos y el framework seleccionado, se dio comienzo al primer sprint, centrado en implementar la funcionalidad básica de la aplicación. En esta fase inicial, se desarrolló la capacidad de introducir un texto original, editar dicho texto, y visualizar simultáneamente ambas versiones (texto

base y texto final) en paralelo. En esta etapa aún no estaba presente la visualización clara de diferencias entre las versiones.

En los sprints posteriores, y una vez que el Product Owner dio el visto bueno a la funcionalidad base, se incorporaron mejoras al sistema. Se añadió la posibilidad de volver a editar el texto final si se detectaban errores, así como la opción de alternar entre la visualización de ambos textos o únicamente del texto final. La funcionalidad más relevante de esta fase fue la introducción del resaltado de diferencias entre los dos textos, una característica clave para la aplicación.

En una etapa más avanzada del desarrollo, se abordaron funcionalidades que mejoraron la experiencia de usuario. Se implementó un sistema para generar y copiar links URL que permiten compartir fácilmente la edición. Además, se añadieron opciones para copiar directamente el texto final al portapapeles y para que el usuario pudiese crear fácilmente un nuevo archivo QuickED, permitiendo usar el texto final como base para una nueva edición.

Finalmente, en el último sprint, se desplegó la aplicación utilizando Cloudflare Pages. Esto permitió que el producto final estuviera accesible públicamente desde cualquier navegador, cumpliendo así uno de los objetivos fundamentales del proyecto: ofrecer una herramienta funcional y disponible para cualquier usuario que desee utilizarla.

El enfoque progresivo de esta metodología permitió mantener una estructura de trabajo flexible, capaz de adaptarse a nuevas funcionalidades sin comprometer la coherencia del desarrollo. Cada sprint fue acompañado de una revisión conjunta con el Product Owner, lo que garantizó que el resultado de cada fase fuese correcto y que cualquier error pudiese corregirse de manera inmediata.

4. Análisis de requisitos

En esta sección se describen los requisitos funcionales y no funcionales de la aplicación QuickED, así como la idea inicial del proyecto.

4.1. Idea inicial

QuickED está ideado originalmente para ser utilizado de forma inmediata, sin necesidad de instalación o configuración por parte del usuario. El uso básico sería el siguiente:

1. Introducir un texto original en el campo correspondiente y seleccionar la opción de empezar a editar.

2. Editar el contenido en una segunda área y confirmar los cambios.

3. Visualizar las diferencias entre ambas versiones, en el modo de solo texto final o en el de ambos textos.

4. Compartir el resultado generado a través de una URL que tenga ambos textos para que lo reciba otro usuario.

Además, si se accede a QuickED mediante una URL con parámetros, el contenido se carga en los campos detexto, lo que facilita el uso.

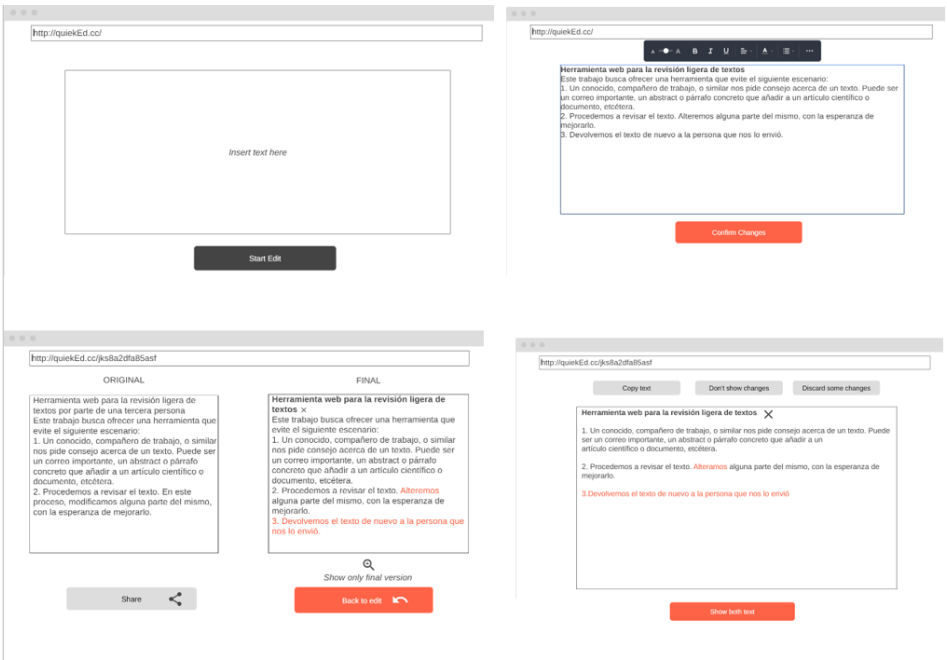

Figura 1. Mockups iniciales

En la figura 1 podemos observar algunos mockups resultantes de las ideas recopiladas durante las primeras reuniones.

Esta propuesta inicial fue modificándose a lo largo del desarrollo del TFG, adaptándose siempre a las necesidades del Product Owner siempre que fuese viable.

4.2. Roles

En primer lugar, cabe destacar la existencia de dos roles diferenciados en la aplicación. El primero de ellos (usuario) es el encargado de enviar un texto a otra persona (el editor) para que lo revise.

El segundo rol (editor), recibe el texto y utiliza la herramienta QuickED para realizar correcciones. Una vez finalizada la edición, el editor comparte la URL con el usuario, quien podrá ver los cambios realizados.

4.3. Requisitos funcionales

Los requisitos funcionales finales se presentan en forma de historias de usuario. Por simplicidad, se incluyen las historias más relevantes.

4.3.1. Introducir texto base

Descripción: Yo, como editor, quiero poder introducir un texto base de manera que tenga una base sobre la que editar y posteriormente ver los cambios.

Pruebas de aceptación:

- Prueba 00: Éxito con texto menor a límite máximo*.
  1) El editor accede a la página web de la aplicación.
  2) Se muestra el título, subtítulo, contador de caracteres, caja de texto y botón de empezar a editar.
  3) Se verifica que aparecen correctamente todos elementos.
  4) El editor pega o escribe el texto base en la caja de texto.
  5) El sistema actualiza el contador de caracteres.
  6) Se verifica que el texto se muestra correctamente y el contador muestra el número de caracteres correcto

- Prueba 01: Éxito con texto superior a límite máximo
  1) El editor accede a la página web de la aplicación.
  2) Se muestra el título, subtítulo, contador de caracteres, caja de texto y botón de empezar a editar.
  3) Se verifica que aparecen correctamente todos elementos.
  4) El editor pega o escribe el texto base con una longitud mayor a la permitida en la caja de texto**.

5) El sistema actualiza el contador de caracteres y lo resalta de color rojo.
6) Se verifica que el texto se muestra correctamente y el contador muestra el número de caracteres correcto.

* Se ha considerado necesario limitar el número de caracteres del texto debido a las restricciones impuestas por las URL de compartición. Este aspecto se aborda con mayor detalle en la sección de implementación.

** Se ha procurado evitar el inconveniente habitual que ocurre al pegar un texto en un campo con límite de caracteres, donde parte del contenido puede ser truncado sin previo aviso. En este caso, el sistema admite el texto completo y notifica al usuario en caso de superar el límite establecido.

4.3.2. Empezar a editar

Descripción: Yo, como editor, quiero poder empezar a editar el texto base de manera que pueda mejorarlo y posteriormente compararlo.

Pruebas de aceptación:

- Prueba 00: Éxito
  1) El editor selecciona la opción de empezar a editar con un texto válido
  2) El sistema cambia el subtítulo y el botón pasando a una nueva interfaz
  3) Se verifica que el texto es el mismo y se modificó la interfaz.

- Prueba 01: Error; límite de caracteres superado
  1) El editor seleccionar la opción de empezar a editar con un texto que excede el límite de caracteres
  2) El sistema muestra un mensaje advirtiendo al usuario que el límite de caracteres se ha excedido
  3) Se verifica que aparece correctamente el aviso
  4) El usuario cierra el aviso
  5) El sistema muestra al usuario el texto base.
  6) Se verifica que aparece la interfaz con el texto correspondiente y el contador de caracteres resaltado

- Prueba 02: Error; no hay texto
  1) El editor selecciona la opción de empezar a editar con el texto en blanco
  2) El sistema muestra un mensaje de advertencia indicando al usuario que el texto está vacío
  3) Se verifica que aparece correctamente el aviso
  4) El usuario cierra el aviso
  5) El sistema muestra la interfaz anterior con el texto base
  6) Se verifica que aparece la interfaz base correctamente

4.3.3. Confirmar cambios

Descripción: Yo, como editor, quiero poder finalizar mis modificaciones para obtener un texto final que se pueda comparar con el original.

- Prueba 00: Éxito
  1) El editor selecciona la opción de confirmar cambios con un texto válido
  2) El sistema cambia de interfaz mostrando ambos textos en paralelo.
  3) El sistema actualiza la URL con los parámetros base y final que corresponden a ambos textos
  4) Se verifica que la URL se ha actualizado correctamente y se muestran los textos final y base en la nueva interfaz

- Prueba 01: Error; límite de caracteres superado
  1) El editor seleccionar la opción de confirmar cambios con un texto que excede el límite de caracteres
  2) El sistema muestra un mensaje advirtiendo al usuario que el límite de caracteres se ha excedido
  3) Se verifica que aparece correctamente el aviso
  4) El usuario cierra el aviso
  5) El sistema muestra al usuario el texto editado.
  6) Se verifica que aparece la interfaz con el texto correspondiente y el contador de caracteres resaltado

4.3.4. Copiar URL

Descripción: Yo, como editor, quiero poder copiar un link a la página de manera que el receptor pueda acceder fácilmente al texto con los cambios.

Pruebas de aceptación:

- Prueba 00: Éxito
  1) El editor selecciona la opción de copiar URL
  2) El sistema muestra al usuario la URL remarcada y la opción de copiarla o cancelar la operación
  3) El usuario selecciona la opción de copiar
  4) El sistema copia en el portapapeles del dispositivo la URL correcta
  5) Se verifica que se copió la URL
  6) Se verifica que la URL lleva al usuario a la página correcta

- Prueba 01: Cancelar
  1) El editor selecciona la opción de copiar URL
  2) El sistema muestra al usuario la URL remarcada y la opción de copiarla o cancelar la operación
  3) El usuario selecciona la opción de cancelar
  4) El sistema vuelve a la interfaz anterior
  5) Se verifica que se vuelve a la interfaz correcta y no se copió la URL

4.3.5. Mostrar ambos textos (inicial y final)

Descripción: Yo, como usuario, quiero poder ver al mismo tiempo el texto inicial y el final de manera que pueda visualizar las diferencias.

Pruebas de aceptación:

- Prueba 00: Éxito
  1) El usuario selecciona la opción de confirmar cambios con el texto ya editado
  2) El sistema actualiza la URL de la página
  3) El sistema muestra la nueva interfaz con el texto base y el editado
  4) Se verifica que aparece la nueva interfaz y la URL correspondientes

- Prueba 01: Error; límite de caracteres superado
  1) El editor seleccionar la opción de confirmar cambios con un texto que excede el límite de caracteres
  2) El sistema muestra un mensaje advirtiendo al usuario que el límite de caracteres se ha excedido
  3) Se verifica que aparece correctamente el aviso
  4) El usuario cierra el aviso
  5) El sistema muestra al usuario el texto base.
  6) Se verifica que aparece la interfaz con el texto correspondiente y el contador de caracteres resaltado

4.3.6. Mostrar texto con cambios resaltados

Descripción: Yo, como usuario, quiero poder ver el texto con los cambios resaltados de manera que pueda identificar fácilmente las modificaciones realizadas y decidir si me agradan o no.

Pruebas de aceptación:

- Prueba 00: Éxito
  1) El usuario selecciona la opción de remarcar cambios
  2) El sistema muestra las diferencias entre ambos textos de una forma clara
  3) Se verifica que se resaltan correctamente los cambios realizados diferenciando entre eliminaciones y añadidos

4.3.7. Empezar nueva edición

Descripción: Yo, como usuario, quiero poder iniciar una nueva edición desde la interfaz de una edición anterior de manera que pueda acceder de forma sencilla a la pantalla principal y continuar trabajando fácilmente.

Pruebas de aceptación:

- Prueba 00: Éxito; Empezar desde 0
  1) El usuario selecciona la opción de empezar un nuevo QuickED desde la interfaz que muestra los resultados de una edición anterior
  2) El sistema muestra las opciones para el nuevo QuickED
  3) Se verifica que aparecen correctamente las opciones
  4) El usuario selecciona la opción de empezar desde cero
  5) El sistema muestra la interfaz base con el texto en blanco
  6) Se verifica que aparece la interfaz base con el texto en blanco

- Prueba 01: Éxito; Usar el texto final como texto base
  1) El usuario selecciona la opción de empezar un nuevo QuickED desde la interfaz que muestra los resultados de una edición anterior
  2) El sistema muestra las opciones para el nuevo QuickED
  3) Se verifica que aparecen correctamente las opciones
  4) El usuario selecciona la opción de usar el texto final como texto base
  5) El sistema muestra la interfaz base con el texto final antiguo como el texto base
  6) Se verifica que aparece la interfaz con el texto correcto

- Prueba 02: Cancelar
  1) El usuario selecciona la opción de empezar un nuevo QuickED
  2) El sistema muestra las opciones para el nuevo QuickED
  3) Se verifica que aparecen correctamente las opciones
  4) El usuario selecciona la opción de cancelar
  5) El sistema muestra la interfaz anterior
  6) Se verifica que se vuelve a la interfaz correcta

4.4. Requisitos no funcionales

4.4.1. Interfaz en inglés

Descripción: Yo, como usuario, quiero que la interfaz esté en inglés, de manera que la herramienta pueda ser utilizada por un mayor número de personas a nivel internacional.

4.4.2. Interfaz sencilla y minimalista

Descripción: Yo, como usuario, quiero que la interfaz sea sencilla y minimalista de manera que evite confusiones y facilite el uso intuitivo de la aplicación.

Además, se persigue como objetivo adicional el diseño de una aplicación que permita la edición de textos de forma ágil, reduciendo en la medida de lo posible el número de interacciones necesarias.

4.4.3. Adaptabilidad a pantallas pequeñas

Descripción: Yo, como usuario, quiero que la interfaz se adapte a diferentes tipos de pantallas de manera que resulte práctico utilizar la aplicación independientemente del dispositivo usado.

5. Diseño

5.1. Arquitectura de componentes

La arquitectura basada en componentes es un enfoque de desarrollo donde la interfaz de usuario se divide en pequeñas unidades independientes, llamadas componentes, que encapsulan su propia lógica, estructura y estilos. Este modelo ha ganado popularidad en frameworks modernos como React y Svelte (el utilizado en este proyecto), ya que favorece la reutilización, la modularidad y la mantenibilidad en el desarrollo de aplicaciones web.

He elegido el diseño actual por su simplicidad, eficacia y alineación con los objetivos del proyecto. Esta estructura permite un desarrollo ágil, una curva de aprendizaje reducida y una integración sencilla con las características de SvelteKit.

A diferencia de otras arquitecturas más complejas, como el modelo de tres capas (presentación, lógica de negocio y acceso a datos), el patrón MVC (Modelo-Vista-Controlador) o el patrón MVP (Modelo-Vista-Presentador), este enfoque evita sobreingeniería innecesaria.

Las principales razones para descartar otras arquitecturas han sido:

-Simplicidad del dominio del problema: El proyecto no maneja flujos de datos complejos ni requiere una lógica extensa.

-Eficiencia y rapidez en el desarrollo: El modelo de componentes facilita una implementación directa, lo que reduce el tiempo de desarrollo y permite iteraciones rápidas.

-Aprovechamiento del enfoque reactivo de Svelte: Svelte proporciona un sistema de gestión de estado integrado, reactivo y eficiente que elimina la necesidad de estructuras adicionales.

-Mantenimiento accesible: La estructura plana facilita la comprensión y el mantenimiento del código por parte de futuros desarrolladores, sin necesidad de que estos entiendan arquitecturas más elaboradas.

Como posible desventaja, cabe destacar que, si la aplicación pretendiera crecer significativamente en el futuro, la falta de una separación clara de responsabilidades en componentes modulares podría dificultar su escalabilidad y mantenimiento. Sin embargo, dado que uno de los objetivos es ofrecer una aplicación ligera y con funcionalidades limitadas, evitando convertirse en una herramienta compleja como Google Docs, esta desventaja no debería representar un problema relevante en este contexto.

5.2. Componentes

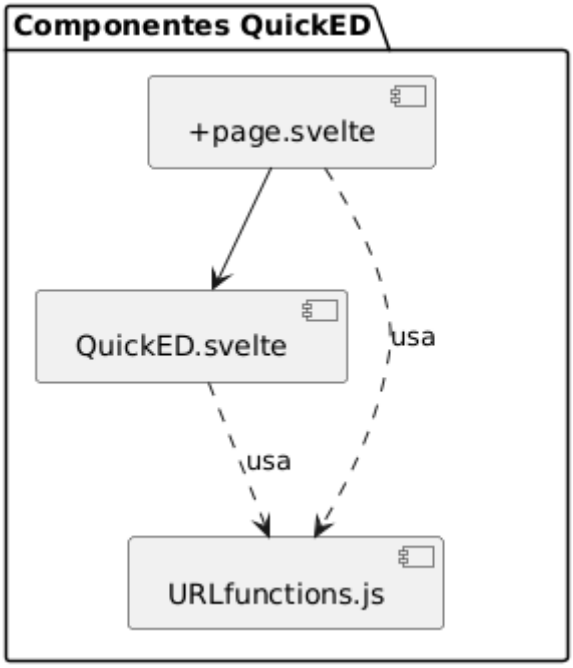

Figura 2. Diagrama de componentes

La aplicación desarrollada sigue el modelo de arquitectura basada en componentes. En la figura 2 se pueden observar los 3 componentes de la aplicación:

-Componente de Interfaz (QuickED.svelte): Este componente central define la estructura de la interfaz y gestiona la interacción con el usuario. Implementa elementos como botones y cajas de texto, además de contener la lógica asociada a los estados. Toda la lógica de presentación y control del estado de la aplicación reside en este archivo.

-Página principal (+page.svelte): Este archivo actúa como punto de entrada de la aplicación, inicializando los datos y pasando lo necesario al componente QuickED. Además, utiliza funciones auxiliares para gestionar los parámetros de la URL.

-Funciones Auxiliares (URLfunctions.js): Este archivo proporciona funcionalidades de apoyo, como la codificación y decodificación de parámetros en URLs. Aunque cumple un papel funcional dentro de la aplicación, no constituye una capa separada de lógica de negocio ni afecta directamente a la presentación visual.

5.3. Ausencia de persistencia de datos

La aplicación no incorpora una capa de persistencia, es decir, no almacena los datos introducidos o modificados en ningún sistema de almacenamiento permanente, ya sea local o remoto. La información gestionada por el usuario es volátil y se mantiene únicamente en la memoria del navegador durante la sesión activa. Se tomó esta decisión debido a varios aspectos:

- Reducción de la complejidad: La no necesidad de una base de datos simplifica considerablemente la arquitectura, facilitando tanto el desarrollo como el despliegue y mantenimiento de la aplicación.

- Enfoque en la inmediatez: La aplicación está orientada a realizar ediciones rápidas y puntuales, sin requerir conservar datos ni gestionar cuentas de usuario.

En lugar de implementar un sistema de persistencia, la aplicación permite compartir el resultado de la edición mediante una URL que codifica tanto el texto original como el modificado en los parámetros de la URL. Esta estrategia presenta varias ventajas:

- Facilidad de uso y portabilidad: El usuario puede compartir el estado de su edición simplemente copiando y enviando la URL.

- Descentralización: Se evita la dependencia de estructura adicional para almacenar o recuperar datos, lo que reduce fallos.

Frente a la alternativa de utilizar una base de datos convencional, esta solución se ajusta mejor a la ligereza de la aplicación, evitando la sobreingeniería y manteniendo una experiencia de usuario directa y sencilla. Asimismo, mantiene el objetivo de ofrecer una herramienta que sin excesiva complejidad cumpla las funcionalidades requeridas.

5.4. Estados

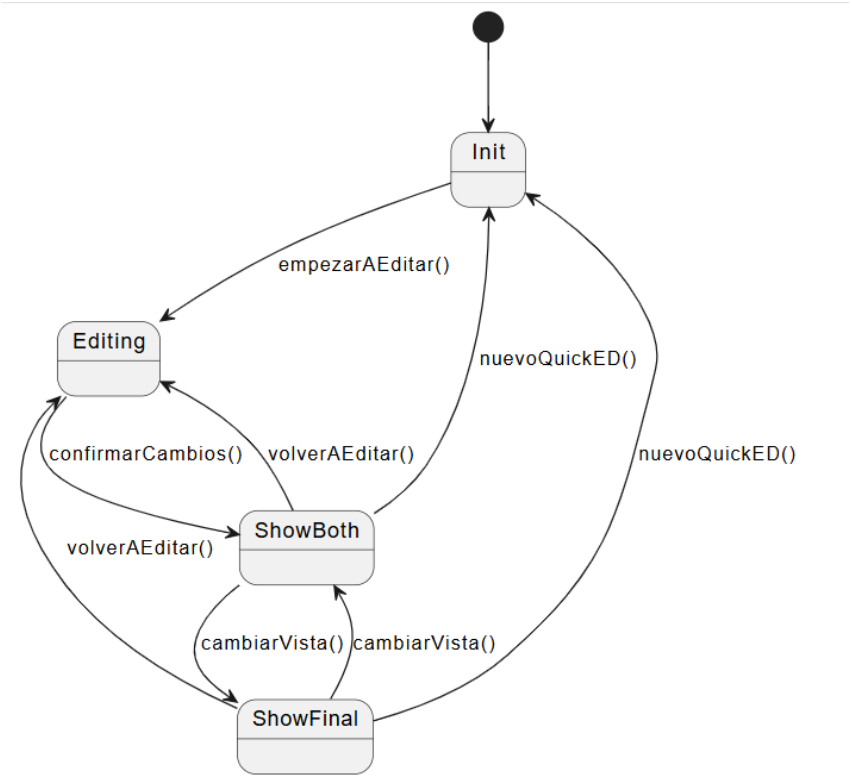

Figura 3. Diagrama de estados

El flujo de la aplicación está determinado por una seria de estados (Figura 3), que define qué elementos deben mostrarse en cada momento y qué acciones están disponibles. Los principales estados son:

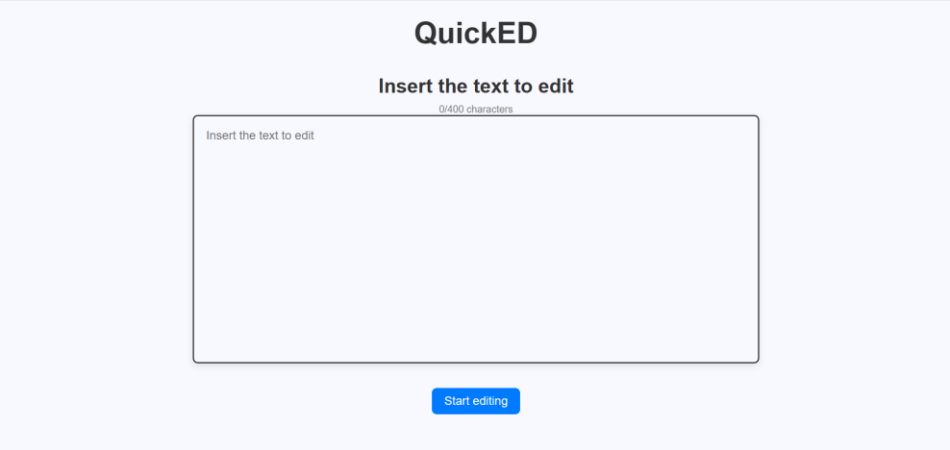

Figura 4. Estado inicial

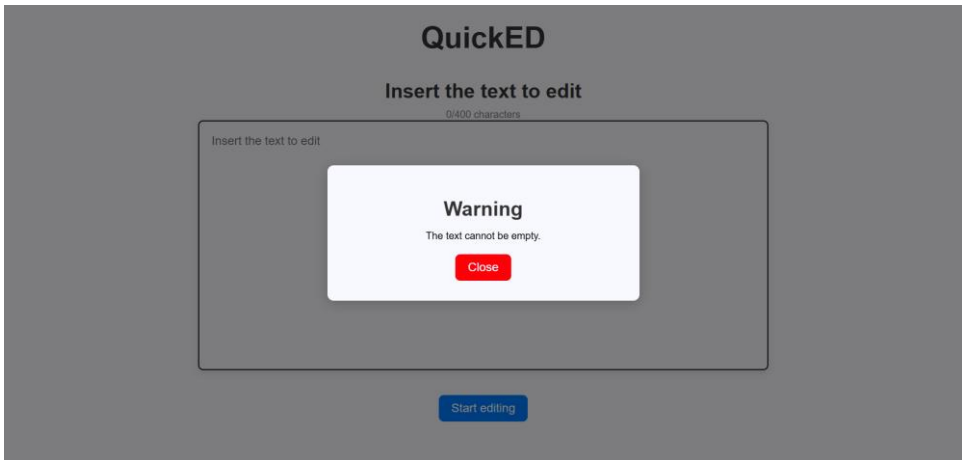

Figura 5. Mensaje de advertencia de texto vacío

Estado inicial (Init): Se muestra al usuario una caja de texto vacía donde introducir el texto base. Durante este estado, hay un contador de caracteres que muestra en tiempo real la longitud del texto introducido. De esta forma, el usuario intentará que el texto no sobrepase el límite. Además, se controla que el texto no esté vacío (Figura 5) ni supere la longitud máxima permitida. En caso de que el texto no sea válido, aparece un modal de advertencia que informa al usuario de que no puede continuar con la edición.

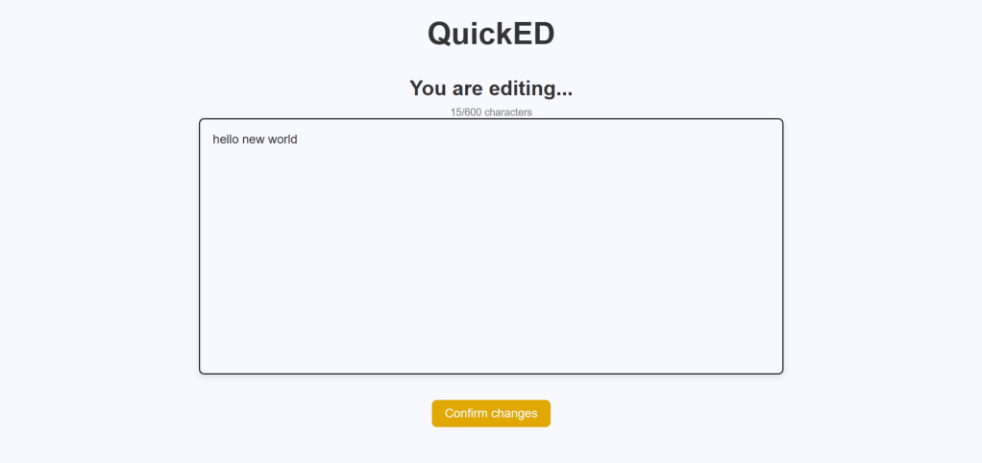

Figura 6. Estado de edición

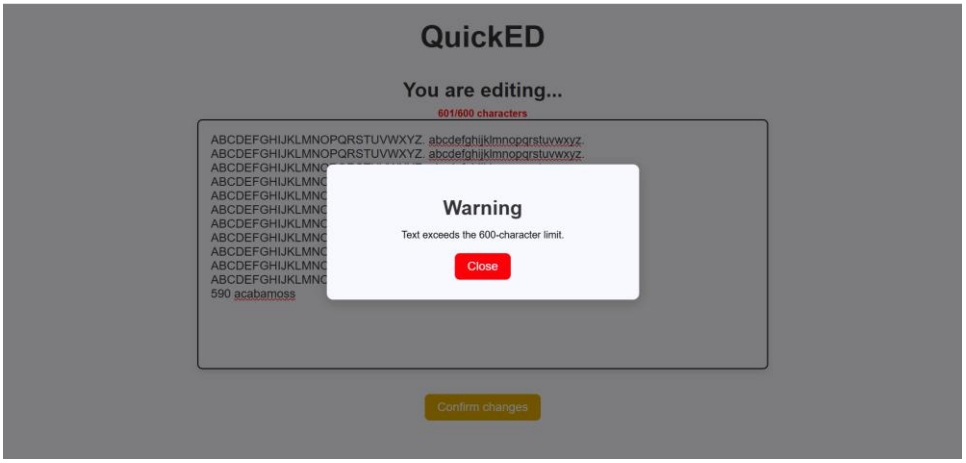

Figura 7. Mensaje de alerta por exceso de caracteres

Estado de edición (Editing): Tras la confirmación del texto base válido, aparece una “nueva” caja de texto donde el usuario puede editar el texto base (Figura 6). Al igual que en el estado inicial, se incorpora un contador de caracteres para controlar la longitud del texto final, y se muestra un modal de advertencia en caso de que el texto esté vacío o exceda la longitud máxima permitida (Figura 7).

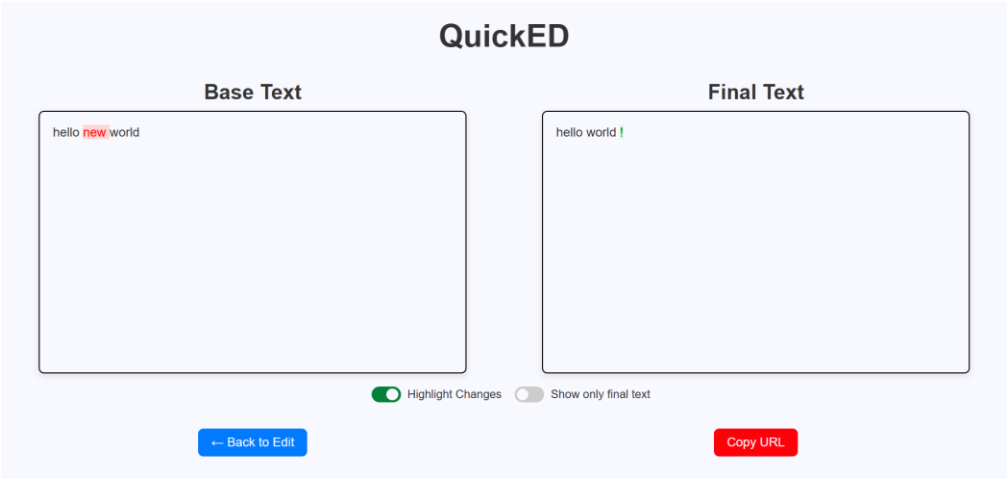

Figura 8. Estado de comparación o Mostrar Ambos

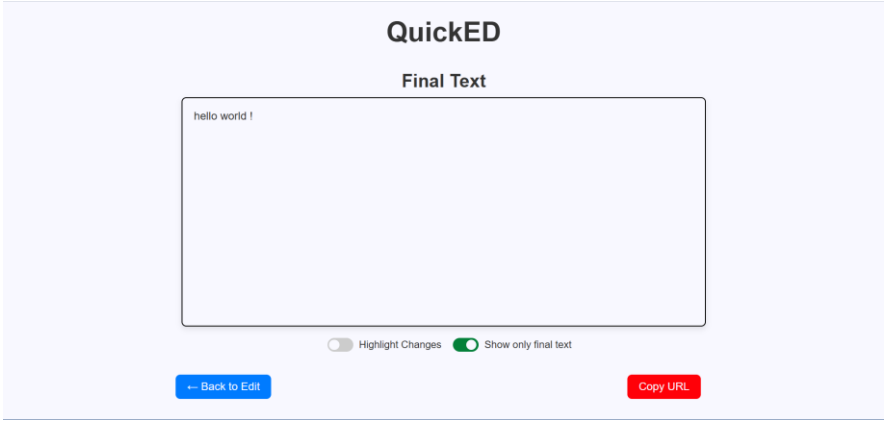

Figura 9. Estado de texto final.

Estado de comparación (ShowBoth) y estado de texto final (ShowFinal): Tras introducir el texto editado y confirmarlo, aparece una vista con los dos textos base y final (Figura 8). Para desplazarse entre el estado de mostrar ambos textos y mostrar solo el texto final (Figura 9) se utiliza un interruptor con 2 posiciones (activado o desactivado) tal y como se puede apreciar en las figuras 8 y 9.

Modo editor: Dependiendo del rol que tengas en la aplicación (editor o usuario) aparecen distintas funcionalidades en los modos de comparación y texto final. Para controlarlo, se utiliza una variable booleana. Cuando el usuario actúa como editor (es decir, tras haber realizado modificaciones sobre un texto), la interfaz muestra opciones como la posibilidad de reeditar el contenido y generar un enlace para compartir los cambios realizados.

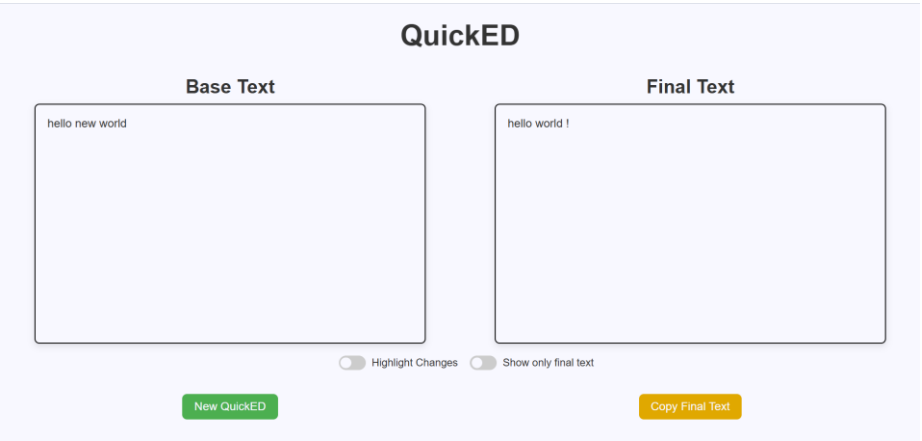

Figura 10. Estado de comparación desde el punto de vista del usuario.

En cambio, si el acceso se produce mediante una URL, aparecen únicamente las acciones de copiar el texto final o iniciar un nuevo QuickED. Dicha interfaz se puede observar en la Figura 10.

6. Implementación

En esta sección se comenta con algo más de detalle las características de la implementación del proyecto.

6.1. Referencia al proyecto

El proyecto completo se encuentra publicado en un repositorio de GitHub (https://github.com/clr474/QuickED) bajo la licencia Apache 2.0[16], lo que lo convierte al proyecto en software de código abierto y reutilizable, tanto en contextos personales como profesionales.

6.2. Estructura del proyecto

6.2.1. Svelte y SvelteKit

El proyecto QuickED está desarrollado con SvelteKit, un framework moderno para construir aplicaciones web rápidas y eficientes. SvelteKit se basa en Svelte, un framework de componentes que permite escribir interfaces en archivos .svelte, los cuales combinan tres tipos de código: HTML (estructura), CSS (estilos) y JavaScript o TypeScript (lógica de la aplicación)

A diferencia de otros frameworks como React o Vue, Svelte no ejecuta su framework en el navegador. En su lugar, compila la lógica de los componentes .svelte a JavaScript puro durante la fase de construcción (build). Estos archivos contienen el código necesario para crear y actualizar el DOM (Document Object Model), es decir, la representación de la página web que permite modificar su contenido y comportamiento desde JavaScript.

Aunque la lógica principal se transforma en JavaScript, en el proceso también se generan los archivos HTML y CSS necesarios para la ejecución y el despliegue de la aplicación.

El archivo +page.svelte es el punto de entrada principal de la aplicación. No existe un main.js ni un index.html en el proyecto, ya que estos archivos son generados automáticamente por SvelteKit durante la compilación. El prefijo + en archivos como +page.svelte indica que el archivo forma parte del sistema de rutas del framework. SvelteKit utiliza esta convención para saber qué archivos deben ser tratados como páginas, layouts, endpoints, etc.

6.2.2. Compilación de componentes Svelte

El desarrollo de QuickED ha utilizado Vite[6], una herramienta moderna que actúa como empaquetador y servidor de desarrollo. Vite ha sido diseñada para ofrecer tiempos de arranque muy rápidos y recarga instantánea del navegador,

lo que la convierte en una opción especialmente adecuada para proyectos con SvelteKit.

Durante el proceso de compilación, cada archivo .svelte se transforma en un archivo JavaScript optimizado que contiene las instrucciones para crear los elementos HTML, los estilos CSS encapsulados y el código JavaScript necesario para gestionar interactividad y estado.

```
<script>
  let mensaje = "Hola mundo";
</script>

<h1>{mensaje}</h1>

<style>
  h1 { color: blue; }
</style>
```

Figura 11. Ejemplo básico de aplicación Svelte.

A modo de ejemplo, el fragmento de código de la Figura 11 muestra una aplicación Svelte simplificada. Este fragmento se compilará a JavaScript que construye un <h1> azul con el texto "Hola mundo", y actualiza su contenido si cambia la variable mensaje.

6.2.3. Entorno de desarrollo

Durante el desarrollo, se utilizó un servidor de desarrollo que se actualiza automáticamente el navegador cuando se detectan cambios en el código. Esto permitió ver el resultado de las modificaciones en tiempo real, sin necesidad de recargar manualmente la página.

Para iniciar el servidor, se ejecutó el comando: *npm run dev*. Esto lanzó la aplicación en http://localhost:5173).

6.2.4. Producción

Cuando la aplicación estubo lista para desplegarse, se generó una versión optimizada mediante el comando: *npm run build*. Este comando ejecutó la compilación de todos los componentes .svelte, generó los archivos HTML, JavaScript y CSS finales y empaquetó los recursos en una estructura estática dentro de la carpeta build/, lista para ser servida desde cualquier host. Para previsualizar localmente la aplicación resultante, se utilizó el comando *npm run preview* el cual abrió un nuevo servidor local.

6.2.5. Modelo de comunicación

La aplicación se encuentra diseñada como una SPA (Single Page Application), es decir, toda la interacción del usuario ocurre sin recargar la página, lo que mejora la experiencia de uso.

La comunicación entre los distintos componetetes de la aplicación se basa en la integración que facilita el framework Svelte. A diferencia de arquitecturas más fragmentadas, en QuickED se optó por un enfoque cohesionado que favorece la mantenibilidad y la sencillez:

- Los archivos .svelte gestionan tanto la lógica de presentación como el estado interno del componente, lo que reduce la necesidad de estructuras externas para el control del flujo de datos.

- Las funciones auxiliares se importan directamente dentro de los componentes que las requieren, sin intermediarios ni capas adicionales, lo que simplifica el trazado del flujo de ejecución.

- Los estilos se aplican de forma global mediante un archivo CSS, sin interferir con la lógica de la aplicación ni crear dependencias cruzadas.

En cuanto al estado de la aplicación, Svelte mantiene variables reactivas dentro de los componentes para guardar datos esenciales como los textos en edición, los modos de operación y el estado actual del proceso. Estas variables persisten mientras el componente está montado, permitiendo una actualización automática y eficiente de la interfaz cuando cambian. En este proyecto concreto, las variables en cuestión serían el texto base y el texto editado.

6.3. Funciones

El componente QuickED.svelte es el núcleo funcional de la aplicación. Toda la lógica de interacción se gestiona desde este componente, el cual organiza el flujo de la aplicación en torno a una serie de estados.

A continuación se detallan las funcionalidades más relevantes:

6.3.1. Generación y lectura de URL

Una de las funcionalidades más importantes de la aplicación es la generación de enlaces personalizados que permiten compartir el resultado de una edición. Esta característica se basa en la codificación de los textos original y final, y su incorporación como parámetros en URL, lo que posibilita recuperar la información de ambos textos, sin necesidad de almacenarla en un servidor.

Desde el punto de vista técnico, esta funcionalidad se implementa mediante tres elementos clave:

- Codificación de textos y generación del enlace
  La función generateURL, definida en el archivo URLfunctions.js, se encarga de:
  - Aplicar una codificación URL a los textos pasados como parámetros llamando a la función encodeURIComponent.
  - Construir una URL que incluye los textos final y base condificados como parámetros (?base=...&final=...).

- Lectura y decodificación de parámetros en la carga de la aplicación
  Al iniciar la aplicación, el archivo +page.svelte ejecuta:
  - La función readURLParameters, también definida en URLfunctions.js, analiza URL (window.location.search).
  - Esta función recupera los valores de los parámetros base y final, los decodifica con decodeURIComponent, y los devuelve como objetos JavaScript (baseT y finalT).

- Inicialización del componente principal con datos precargados
  Una vez recuperados los textos, +page.svelte:
  - Asigna los valores a las variables baseText y finalText.
  - Pasa estas variables al componente principal QuickED.svelte.

De esta forma, aparecen automáticamente los textos en estado de comparación.

En caso de que no se detecten parámetros en la URL, el archivo +page.svelte asigna a las variables correspondientes textos vacíos (""). Esto indica que la aplicación debe iniciar en su estado base.

A la hora de diseñar esta funcionalidad, uno de los aspectos clave fue la limitación en la longitud máxima que pueden alcanzar las URLs para ser interpretadas correctamente por navegadores. Aunque no existe un estándar oficial rígido, en la práctica se considera que el límite seguro para la mayoría de los navegadores modernos (como Chrome, Firefox o Edge) ronda los 2000 a 2500 caracteres. Superar estas longitudes puede provocar errores de carga o truncamiento de datos.

Inicialmente se contempló el uso de la codificación en Base64[17], una técnica común para transmitir datos binarios en entornos de texto plano. Sin embargo, esta codificación incrementa el tamaño del contenido original en aproximadamente un 33%, es decir, cada bloque de 3 bytes se convierte en 4 caracteres Base64, lo que equivale a una relación de crecimiento de 4:3. Para textos extensos, este incremento puede suponer demasiados caracteres adicionales.

Por esta razón, se optó por una estrategia de codificación más simple y eficiente, utilizando la función estándar de JavaScript encodeURIComponent[18]. Esta

técnica no codifica letras, números ni caracteres comunes como el guion o el punto, manteniendo una relación de tamaño 1:1 para la gran mayoría de caracteres. Solo los caracteres especiales (como espacios, símbolos de puntuación o caracteres no ASCII) aumentan su tamaño, generalmente a través de una representación hexadecimal (%XX), con una relación de tamaño de 3:1. Para textos alfabéticos (como suele ser el caso en esta aplicación), el crecimiento total en tamaño suele ser mucho menor que el de Base64, lo que permite utilizar textos mas largos sin comprometer las URL.

Se calcularon distintos escenarios en los que la proporción de caracteres especiales en el texto varía. El objetivo era obtener un punto de equilibrio en el que la URL total no superase los 2000 caracteres incluso con un porcentaje elevado de caracteres especiales

El análisis llevó a la conclusión de que se podían admitir hasta 1000 caracteres combinados entre el texto base y el texto final (por ejemplo, 400 base y 600 editado), sin comprometer la funcionalidad. Finalmente se utilizó un caso extremo en el que hasta un 47,5% del texto esté compuesto por caracteres especiales. El tamaño total de la URL generada estaría en el límite. Para los cálculos se tuvo en cuenta la presencia de una URL base, en este caso, utilice una longitud de 50 caracteres.

La siguiente tabla detalla la relación entre la proporción de caracteres especiales y la longitud final estimada de la URL, usando 1000 como longitud del texto:

% Caracteres especiales	**N° de caracteres especiales**	**Longitud codificada**	**URL total (con URL base)**
0%	0	1000	1050
10%	100	1000 + 2×100 = 1200	1250
20%	200	1000 + 2×200 = 1400	1450
30%	300	1000 + 2×300 = 1600	1650
40%	400	1000 + 2×400 = 1800	1850
47.5%	**475 (máximo)**	1000 + 950 = 1950	**2000 (límite práctico)**
50%	500	1000 + 1000 = 2000	**2050 (excede límite)**
60%	600	1000 + 1200 = 2200	2250

Tabla 1. Comparativa de longitudes de URL codificadas

Cabe destacar que un porcentaje de caracteres especiales tan elevado como el 47,5% es muy improbable en un uso real de la herramienta, dado que la mayoría de los textos editados están compuestos casi enteramente por caracteres comunes. Aun así, se ha decidido adoptar este valor como umbral para garantizar un funcionamiento correcto en escenarios exigentes. No obstante, se sabe que en casos extremos, por ejemplo, al procesar textos con gran cantidad de símbolos, emojis, o codificaciones no latinas, podría superarse el límite establecido, provocando un fallo en la URL. Aunque estos casos son residuales, se asume dicha limitación como inherente a esta estrategia de codificación, considerando que el beneficio compensa este pequeño riesgo.

6.3.2. Remarcado de cambios

Una de las funcionalidades más importante es el sistema de remarcado de cambios entre el texto original y el texto editado. Para implementar esta funcionalidad se optó por el uso de la librería js-diff[3], una herramienta ligera y versátil que permite comparar cadenas de texto. En concreto, se utilizó su modo de comparación a nivel de palabras, lo que permite obtener diferencias desde el punto de vista semántico, en lugar de producir resultados muy fragmentados como ocurre con la comparación carácter a carácter (útil para faltas de ortografía principalmente), o demasiado generales como en la comparación por líneas. Este enfoque permite al usuario identificar de forma precisa qué se ha añadido o eliminado.

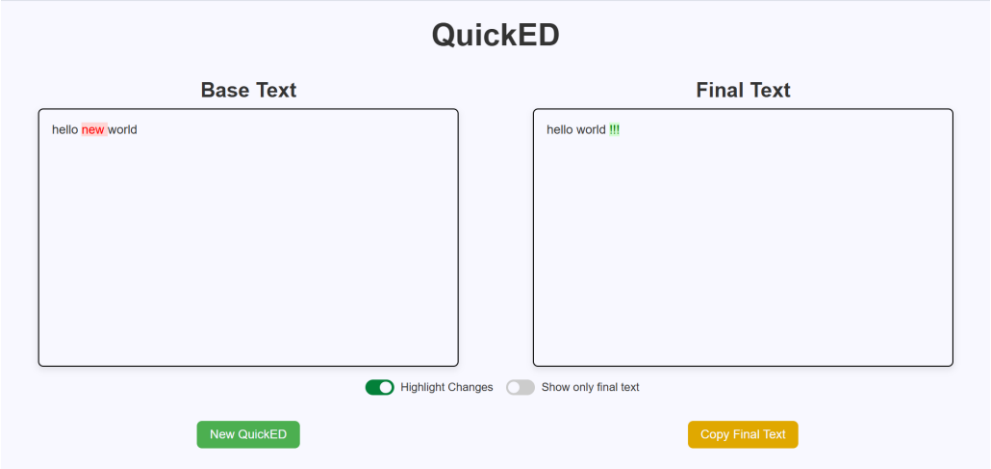

Figura 12. Cambios resaltados en el modo comparación

A partir del resultado proporcionado por diff, se generan elementos HTML con clases CSS asociadas a los cambios. Las palabras eliminadas se destacan con un fondo rojo, mientras que las añadidas se resaltan en verde claro (Figura 12). Este esquema de color, junto con la estructura de resaltado por palabra, proporciona una experiencia visual intuitiva y rápida de interpretar.

6.3.3. Generar un nuevo QuickED

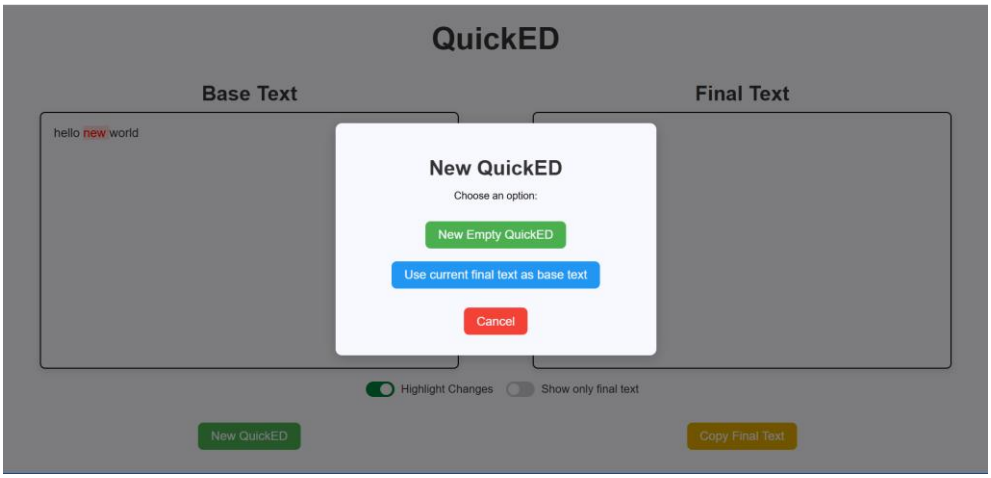

Figura 13. Nuevo QuickED

Una vez finalizada una edición y visualizados los cambios, el usuario puede continuar trabajando gracias a la funcionalidad de “Generar un nuevo QuickED”, que “reinicia” la aplicación pero con la opción de utilizar el texto final de la edición actual como nuevo texto base o no (Figura 13). Esta acción permite al usuario continuar una cadena de revisiones sucesivas, algo útil en procesos de corrección iterativa.

6.4. Estilos

La interfaz, aunque visualmente sencilla, está diseñada mediante CSS personalizado. La hoja de estilos (page.css) define tanto los estilos globales como específicos para los distintos estados y elementos interactivos: botones, campos de texto, alertas, etc.

Cada destacar el color de fondo, originalmente se utilizó un blanco puro, pero a recomendación del Producto Owner se modificó a un tono blanco un poco más apagado, evitando un excesivo daño a la vista si la iluminación de la habitación es baja.

Figura 14. QuickED en un dispositivo móvil

Se ha hecho un esfuerzo especial para garantizar una mejor compatibilidad con dispositivos móviles, incorporando “media queries” que ajustan automáticamente el tamaño de las cajas de texto cuando el ancho de la pantalla es reducido, en este caso concreto, menos de 800px (Figura 14).

En términos de experiencia de usuario, hay varios elementos dinámicos: los campos de texto cambian de aspecto al seleccionarlos para poder escribir y los botones cambian levemente el color al pasar el ratón por encima.

6.5. Despliegue en Cloudflare Pages

Con el objetivo de facilitar el acceso público a la, se ha llevado a cabo el despliegue del proyecto mediante Cloudflare Pages[12], una plataforma moderna para el alojamiento de sitios estáticos. El sistema se ha configurado de manera que cualquier cambio realizado en el repositorio de GitHub de la rama main se traduzca automáticamente en una nueva versión del sitio web.

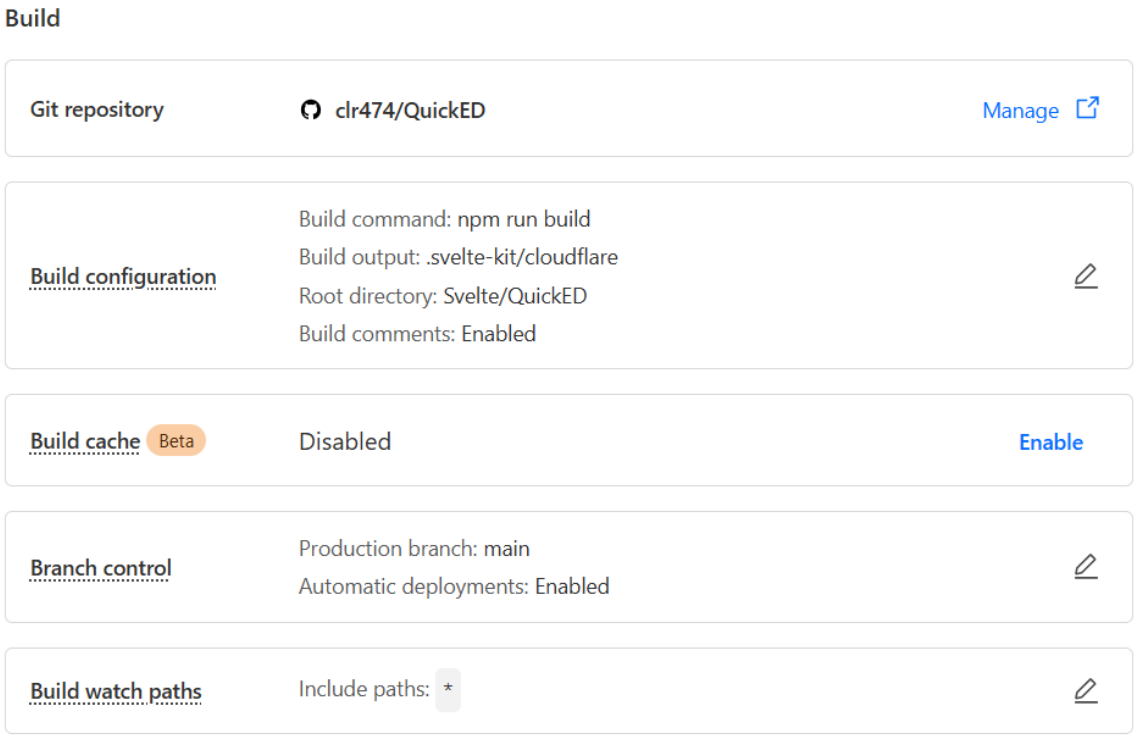

Figura 15. Configuración Build para QuickED en Cloudflare Pages

El proceso de integración entre GitHub y Cloudflare resultó sencillo y directo. Desde el panel de administración de Cloudflare Pages, se inició la creación de un nuevo proyecto usando la opción de conectar con un repositorio de GitHub. Tras dar los permisos necesarios, Cloudflare tiene acceso al repositorio donde está alojado el proyecto. En la figura 15 se puede ver la configuración final para la aplicación QuickED

Una vez elegido el repositorio, la plataforma identifica las ramas disponibles (como main o develop) y permite configurar cuál de ellas se utilizará como rama principal para los despliegues. Posteriormente, se definen los parámetros de construcción del entorno: como se utilizó el framework Svelte, Cloudflare propuso configuraciones automáticas adaptadas a dicho entorno (como pr ejemplo el comando de compilación npm run build). Además, hay que especificar el directorio donde se almacenan los archivos resultantes del proceso de compilación que será el contenido final servido al usuario.

Con esta configuración, el proceso de despliegue está automatizado. Cada vez que se realice un push a la rama main, Cloudflare Pages compilará el proyecto y publicará una nueva versión. Esto elimina la necesidad de realizar despliegues manuales y garantiza que la aplicación se mantenga sincronizada con el código de la rama principal.

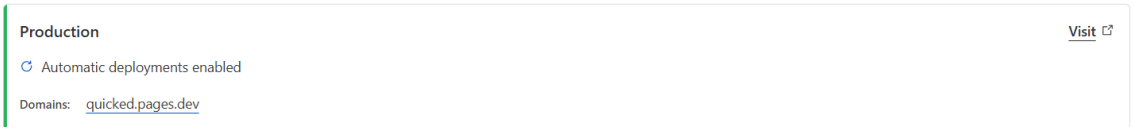

Figura 16. Ventana de producción y dominio de QuickED

La versión online de la aplicación es accesible públicamente a través del siguiente enlace: https://quicked.pages.dev/. En la figura 16 se puede observar el dominio de la app en producción.

Además, la plataforma permite vincular un dominio personalizado en caso de que se desee ofrecer una dirección más reconocible, aunque se optó por mantener la proporcionada.

Esta solución ofrece múltiples ventajas, entre las que destacan la integración nativa con GitHub, la alta disponibilidad proporcionada por su red de distribución de contenido global[19], y la escalabilidad automática sin necesidad de configurar ni mantener infraestructura adicional.

7. Pruebas

Durante el desarrollo de la aplicación, se comprobó el correcto funcionamiento mediante distintos tipos de pruebas. Estas pruebas han permitido detectar y corregir errores, garantizar la funcionalidad del sistema y asegurar una óptima experiencia de usuario.

7.1. Framework

Para la ejecución de las pruebas en la aplicación se utilizó Vitest[7], un framework de testing rápido diseñado para entornos de desarrollo modernos. Vitest proporciona una API clara y flexible que facilita la implementación de pruebas unitarias y de integración.

Además, se utilizó Testing Library para Svelte[20], una librería orientada a la simulación de interacciones con los componentes de la interfaz como botones, campos de texto y validaciones, asegurando que la interfaz responda correctamente a las acciones del usuario.

En las pruebas de interfaz, para la simulación de eventos y la validación de funcionalidades relacionadas con la interacción en la interfaz, se utilizaron funciones como fireEvent[21] que permiten reproducir comportamientos de elementos, como por ejemplo pulsar un botón.

Además, para las pruebas de integración, se usó Spy Mocks[22] para la correcta actualización de la URL durante el proceso de edición de textos.

Se optó por Vitest en lugar de otras herramientas como Selenium[23] para mantener todas las pruebas dentro del mismo contexto de ejecución, evitando la complejidad que implican las pruebas en navegadores externos y de esta forma asegurar una mayor integración con el entorno de desarrollo.

7.2. Pruebas unitarias

Para garantizar que cada parte de la aplicación funcionase correctamente, se crearon pruebas unitarias.

Dado que la aplicación añade los textos base y final en los parámetros de la URL, se implementaron pruebas unitarias para garantizar la correcta generación y recuperación de datos. Estas pruebas comprobaron distintos escenarios, como la codificación de caracteres especiales, cadenas vacías y caracteres no convencionales (emojis, por ejemplo).

```
5   const originalLocation = 'TestQuickEDgenerateURL';
6
7   it('Normal strings', () => {
8     const url = generateURL('hello', 'world', originalLocation);
9     expect(url).toBe('TestQuickEDgenerateURL/?base=hello&final=world');
10  });
```

Figura 17. Prueba unitaria

La figura 17 representa el fragmento de código de una prueba unitaria que verifica el correcto funcionamiento de la función generateURL. En concreto, comprueba si la función genera una URL correcta con cadenas de texto normales.

- Se definió una variable originalLocation con un valor inicial (línea 5).

- it('Normal strings', () => { ... }), representa la declaración de una prueba, que en este caso llama a generateURL con parámetros.

-Compara el resultado obtenido con el esperado, utilizando expect(url).toBe(...).

Si la comparación es correcta, el test pasará correctamenta.

7.3. Pruebas de integración

Además de asegurar que los módulos individuales funcionasen correctamente, se comprobó que se integrara correctamente en el proyecto.

Uno parte que se comprobó, fue la interacción entre el editor de texto y las URLs.

```
6   it('Updates the URL when transitioning to ShowBoth state', async () => {
7       // Mock window.history.pushState
8       const pushStateMock = vi.spyOn(window.history, 'pushState');
9       const { getByPlaceholderText, getByText } = render(QuickED);
10      const textarea = getByPlaceholderText('Insert the text to edit');
11      await fireEvent.input(textarea, { target: { value: 'Base text' } });
12      await fireEvent.click(getByText('Start editing'));
13      // Simulate user editing and confirming changes
14      const editTextarea = document.querySelector('textarea.edit') || getByPlaceholderText('Edit the text');
15      await fireEvent.input(editTextarea, { target: { value: 'Final text' } });
16      await fireEvent.click(getByText('Confirm changes'));
17      // Expect pushState to have been called (URL updated)
18      expect(pushStateMock).toHaveBeenCalled();
19      const lastCall = pushStateMock.mock.calls[pushStateMock.mock.calls.length - 1];
20      expect(lastCall[2]).toContain(window.location.origin);
21      pushStateMock.mockRestore();
22  });
```

Figura 18. Prueba de integración

La figura 18 representa el fragmento de código de una prueba de integración que verifica que la URL se actualiza correctamente cuando el usuario edita y confirma cambios en el texto.

- Mockea la función window.history.pushState para controlar que URL se actualiza (línea 8).

- Edita el texto en la interfaz, utilizando fireEvent para insertar y modificar el contenido en las cajas de texto (líneas 10-12).

- Inicia la edición, modifica los cambios y los confirma (líneas 14-16).

- Verifica que pushState ha sido llamado y que la nueva URL es correcta (líneas 18-20).

- Restaura el mock después de la prueba, evitando interferencias en otros tests (línea 21).

7.4. Pruebas de interfaz

Dado que la aplicación tiene una fuerte interacción con el usuario, se llevaron a cabo pruebas de interfaz para evaluar aspectos relacionados con la presentación visual y la usabilidad.

Uno de los aspectos revisados fue la adaptación a diferentes dispositivos. La aplicación debía ser funcional tanto en pantallas grandes como en dispositivos móviles, por lo que se probaron distintos dispositivos y se verificó que la que la visualización fuese correcta y adecuada.

Además, se comprobó que los controles, como el interruptor para cambiar entre el estado de comparación y de texto final, funcionasen correctamente. También se comprobó que los mensajes de advertencia aparecieran en situaciones como el exceso de caracteres.

Algunas pruebas se realizaron de forma manual, inspeccionando el diseño en navegadores y dispositivos físicos, y otras mediante herramientas de simulación de interacciones, mencionadas anteriormente, para asegurar el correcto funcionamiento.

7.5. Pruebas de sistema/aceptación

Primero, el equipo de desarrollo realizó pruebas de interfaz donde se comprobó los diferentes estados del sistema y las acciones disponibles para el usuario. Esto permitió detectar y corregir posibles problemas antes de la evaluación externa.

Posteriormente, el sistema fue examinado por el Product Owner, quien se aseguró que la aplicación cumpliera con las expectativas funcionales. En esta fase, se verificó que todas las pruebas de aceptación que se habían definido previamente se ejecutaran correctamente.

Durante este proceso, se identificó un comportamiento inesperado en la parte de comparación de textos. En concreto, al reemplazar un guion (-) entre dos palabras por un espacio, el sistema de comparación no mostraba correctamente la modificación. En lugar de mostrar solo el cambio del caracter, resaltaba tanto el espacio añadido como el guion eliminado, generando una representación visual incorrecta. El problema radica en que, tras eliminar el guion, el sistema cambia la posición del espacio que sigue a la palabra, llevándolo antes del guion eliminado en lugar de reemplazarlo correctamente. A pesar de varios intentos de corregirlo el error se mantiene en la versión actual.

Este comportamiento se puede apreciar en el siguiente enlace: https://quicked.pages.dev/?base=Modiff%20is%20an%20under-development%20two-way%20model%20comparison%20tool%20that%20makes%20use%20of%20a%20line-based%20diff%20to%20find%20model%20element%20differences%20efficiently%2C%20this%20is%2C%20without%20having%20to%20match%20and%20fully%20compare%20all%20elements%20of%20two%20model%20versions.&final=Modiff%20is%20a%20(currently%20under-development)%20two-way%20model%20comparison%20tool%20that%20makes%20use%20of%20a%20line-based%20diff%20to%20find%20model%20element%20differences%20efficiently%2C%20this%20is%2C%20without%20having%20to%20match%20and%20fully%20compare%20all%20elements%20of%20two%20model%20versions

Aunque este error no compromete la funcionalidad esencial del sistema, representa una limitación que debe abordarse en futuras versiones para mejorar la precisión del resaltado de cambios.

Finalmente, para obtener una perspectiva externa, la aplicación fue probada por potenciales usuarios finales, quienes ofrecieron comentarios sobre su usabilidad y claridad.

7.6. Integración continua

Siempre que fue posible, se priorizó la automatización de pruebas para reducir el esfuerzo manual y mejorar la eficiencia en la detección de errores. Se utilizó un sistema de pruebas unitarias e integración que permitía ejecutar comprobaciones de manera rápida y repetitiva, asegurando que cada modificación en el código no introdujera errores inesperados.

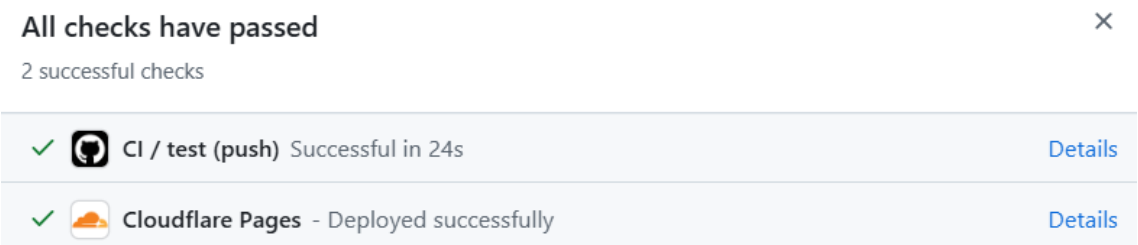

Figura 19. Resultado resumen de GitHub Actions

Para garantizar el funcionamiento del código a lo largo del desarrollo, se estableció un proceso de automatización del lanzamiento de pruebas mediante un sistema de integración continua (GitHub Actions[10]), lo que permitió validar de forma inmediata cada modificación realizada en el proyecto al subirlo al repositorio. Esta automatización se configuró para ejecutarse cada vez que se realizase un push a las ramas develop y/o main, asegurando que el código fuese correcto. En la figura 19 se puede observar el resumen de la ejecución de las pruebas de GitHub Actions y del despliegue en Cloudflare Pages

8. Trabajos futuros

A pesar de que el proyecto actual cumple con los objetivos establecidos en cuanto a la edición rápida de textos, existen varias mejoras que podrían abordarse en futuros desarrollos. Algunas de las funcionalidades más relevantes que se proponen para futuras versiones son las siguientes:

8.1. Modo oscuro

Se propone la incorporación de un modo oscuro o dark mode que permita a los usuarios alternar entre diferentes esquemas de color, con el objetivo de mejorar la accesibilidad y reducir la fatiga visual, especialmente en entornos con poca iluminación.

8.2. Gestión individual de cambio

Actualmente, se muestra la diferencia entre el texto original y el final con todas las diferencias remarcadas. Como mejora, se plantea la posibilidad de implementar un sistema de control de cambios que permita aceptar o rechazar modificaciones de forma individual.

8.3. Almacenamiento de textos en servidor

En la versión actual, la forma de compartir un archivo editado es mediante una URL que codifica ambos textos (base y final) directamente en la URL. Dado que esto presenta limitaciones importantes en cuanto a la longitud máxima de los textos, se propone implementar un servidor que permita almacenar los textos en una base de datos o sistema de archivos, creando una URL única por documento. Esto permitiría compartir enlaces mucho más cortos (por ejemplo, del tipo https://quicked.com/doc/abc123) y eliminar casi por completo la limitación en el número de palabras o tamaño del texto.

8.4. Mejora del texto mediante LLM

Se propone la integración de un sistema de procesamiento basados en Modelos de Lenguaje de gran escala (LLMs[24], por sus siglas en inglés), con el objetivo de sugerir y aplicar mejoras automáticas en la edición de los textos. Este módulo permitiría optimizar aspectos gramaticales, de estilo y de coherencia textual, contribuyendo a una edición más eficiente.

Asimismo, se plantea el desarrollo de una interfaz interactiva que permita visualizar los cambios sugeridos por el modelo, otorgando al usuario la capacidad de aceptarlos, rechazarlos o modificarlos de manera individual.

A pesar de las ventajas de esta funcionalidad, se considera prioritario preservar el enfoque colaborativo de la herramienta. Por ello, la inteligencia artificial se integraría como un complemento a la intervención humana, manteniéndose el control final de la edición en manos del usuario.

8.5. Conclusiones

Estas mejoras no solo enriquecerían la experiencia del usuario, sino que también aportarían mayor robustez, escalabilidad y flexibilidad a la herramienta, facilitando su aplicación en contextos más exigentes o profesionales.

No obstante, cuantas más funcionalidades se añadan a la herramienta, más cerca nos encontraremos de tener una aplicación completa y pesada como puede ser Microsoft 365 o Google Docs. Por lo tanto, la decisión de añadir funcionalidades debe ser un proceso meditado y progresivo en el tiempo.

9. Conclusiones

El desarrollo de esta herramienta ha alcanzado los objetivos propuestos inicialmente: ofrecer una aplicación web funcional para la edición rápida de textos, basada en la comparación entre versiones mediante una interfaz clara, accesible y eficiente. A lo largo del proyecto, se han abordado distintas fases del ciclo de vida del software, desde el análisis y diseño hasta la implementación, pruebas y despliegue.

El uso de tecnologías modernas como JavaScript y Svelte, junto con la automatización de pruebas y el despliegue continuo mediante GitHub y Cloudflare Pages, ha permitido construir un sistema robusto, escalable y fácil de mantener. Las pruebas realizadas, tanto automáticas como manuales, han garantizado una experiencia de usuario satisfactoria y una aplicación libre de errores graves. Además, se han identificado ciertas limitaciones, como el tratamiento de diferencias en algunos casos particulares, que marcan las tareas iniciales para futuras mejoras.

Desde un punto de vista personal, este proyecto ha supuesto una experiencia enriquecedora. Ha permitido aplicar conocimientos adquiridos durante la carrera en un entorno más práctico, al tiempo que se han aprendido nuevas competencias en áreas como el control de versiones o el despliegue en la nube. Además, ha reforzado la capacidad de organización y resolución de problemas.

Gracias a este trabajo, se ha obtenido una visión más clara del proceso completo de desarrollo de software, desde la concepción de una idea hasta su en producción. Esta experiencia no solo contribuye a consolidar una base técnica sólida, sino que también supone un impulso para una futura incorporación al mundo laboral en el ámbito del desarrollo web o de herramientas orientadas a la productividad.

10. Bibliografía

[1] «Text Diff - Free online text compare tool». Accedido: 9 de junio de 2025. [En línea]. Disponible en: https://text-diff.com/

[2] «MDN Web Docs». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://developer.mozilla.org/es/

[3] «diff - npm». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://www.npmjs.com/package/diff

[4] «Svelte • Web development for the rest of us». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://svelte.dev/

[5] «Introduction • Docs • Svelte». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://svelte.dev/docs/kit/introduction

[6] «Vite | Next Generation Frontend Tooling». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://vite.dev/

[7] «Vitest | Next Generation testing framework». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://vitest.dev/

[8] «webpack». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://webpack.js.org/

[9] «Git». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://git-scm.com/book/en/v2

[10] «Documentación de GitHub Actions - Documentación de GitHub». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://docs.github.com/es/actions

[11] «Welcome to Cloudflare | Cloudflare Docs». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://developers.cloudflare.com/

[12] «Cloudflare Pages». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://pages.cloudflare.com/

[13] «Documentation for Visual Studio Code». Accedido: 4 de junio de 2025. [En línea]. Disponible en: https://code.visualstudio.com/docs

[14] «Sublime Text - Text Editing, Done Right». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://www.sublimetext.com/

[15] «Manifesto for Agile Software Development». Accedido: 9 de junio de 2025. [En línea]. Disponible en: https://agilemanifesto.org/

[16] «Apache License, Version 2.0». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://www.apache.org/licenses/LICENSE-2.0

[17] «RFC 4648 - The Base16, Base32, and Base64 Data Encodings». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://datatracker.ietf.org/doc/html/rfc4648

[18] «encodeURIComponent() - JavaScript | MDN». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent

[19] «Red de entrega de contenido (CDN): funciones y beneficios | Cloudflare». Accedido: 8 de junio de 2025. [En línea]. Disponible en: https://www.cloudflare.com/es-es/learning/cdn/what-is-a-cdn/

[20] «Intro | Testing Library». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://testing-library.com/docs/svelte-testing-library/intro/

[21] «Firing Events | Testing Library». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://testing-library.com/docs/dom-testing-library/api-events/#fireevent

[22] «Mock Functions | Vitest». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://vitest.dev/api/mock.html

[23] «Selenium». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://www.selenium.dev/

[24] «¿Qué son los grandes modelos de lenguaje (LLM)? | IBM». Accedido: 10 de junio de 2025. [En línea]. Disponible en: https://www.ibm.com/es-es/think/topics/large-language-models