Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Desarrollo del Módulo Interactivo 'Stance & Source Check' para el Fomento de la Alfabetización Mediática en la Plataforma DeStance

Autor: Daniel Ramón Robertson
Tutor(a): Jelena Bobkina, Elena Domínguez Romero

Madrid, junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Desarrollo del Módulo Interactivo 'Stance & Source Check' para el Fomento de la Alfabetización Mediática en la Plataforma DeStance

junio 2025

Autor: Daniel Ramón Robertson

Tutor:

Jelena Bobkina, Elena Domínguez Romero

Lingüística Aplicada a la Ciencia y a la Tecnología

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

El presente Trabajo de Fin de Grado aborda el desarrollo del módulo interactivo "Stance & Source Check" para la plataforma educativa DeStance, en el marco del proyecto de investigación RACISMMAFF. En un contexto donde la desinformación sobre inmigración y racismo prolifera en medios digitales, este trabajo responde a la necesidad crítica de desarrollar herramientas pedagógicas que fomenten la alfabetización mediática. El objetivo principal fue diseñar e implementar un módulo web que permita a estudiantes universitarios desarrollar competencias para identificar marcadores lingüísticos de posicionamiento discursivo y evaluar la credibilidad de fuentes informativas.

La metodología empleada combinó un enfoque de desarrollo iterativo con principios de diseño pedagógico, implementando una arquitectura full-stack con SvelteKit, Node.js y MongoDB. La solución técnica incorporó Rangy.js para gestionar interacciones complejas de selección de texto, superando las limitaciones de las APIs nativas del navegador. El módulo resultante guía a los usuarios a través de un proceso estructurado de cuatro fases: evaluación de fiabilidad inicial, análisis interactivo de marcadores lingüísticos, evaluación sistemática de fuentes y retroalimentación personalizada. Las principales aportaciones incluyen un sistema robusto de highlighting persistente, algoritmos de comparación flexible entre selecciones de usuarios y marcadores expertos, y una integración exitosa con el ecosistema MOOC existente.

Entre las limitaciones identificadas destacan la dependencia de curación manual de contenido y la ausencia de pruebas automatizadas exhaustivas, aspectos que se proponen como líneas de trabajo futuro.

Abstract

This Final Degree Project addresses the development of the interactive module "Stance & Source Check" for the DeStance educational platform, within the framework of the RACISMMAFF research project. In a context where misinformation about immigration and racism proliferates in digital media, this work responds to the critical need to develop pedagogical tools that promote media literacy. The main objective was to design and implement a web module that enables university students to develop competencies in identifying linguistic markers of discursive positioning and evaluating the credibility of information sources.

The methodology employed combined an iterative development approach with pedagogical design principles, implementing a full-stack architecture with SvelteKit, Node.js, and MongoDB. The technical solution incorporated Rangy.js to manage complex text selection interactions, overcoming the limitations of native browser APIs. The resulting module guides users through a structured four-phase process: initial reliability assessment, interactive analysis of linguistic markers, systematic source evaluation, and personalized feedback. Key contributions include a robust persistent highlighting system, flexible comparison algorithms between user selections and expert markers, and successful integration with the existing MOOC ecosystem.

Among the identified limitations are the dependence on manual content curation and the absence of comprehensive automated testing, aspects proposed as future lines of work.

Tabla de contenidos

1. Introducción

1.1 Contextualización del Proyecto: Plataforma DeStance y Objetivos de RACISMMAFF

En este panorama europeo de creciente superdiversidad, donde las dinámicas de movilidad generan contextos estratificados y, en ocasiones, la naturalización de desigualdades, se observa la emergencia de ideologías xenófobas y prácticas de radicalización. El lenguaje y las prácticas comunicativas son vehículos frecuentes para la diseminación de estas ideologías, lo que subraya la necesidad imperante de desarrollar metodologías de investigación lingüística capaces de identificar, analizar y abordar dichas prácticas, especialmente en los discursos institucionales, políticos y mediáticos [14].

Es en este marco donde se inscribe el proyecto de investigación STANCE STRATEGIES IN IMMIGRATION AND RACISM-RELATED DISCOURSE: ANALYSIS AND APPLICATIONS IN AFFECTIVE LEARNING PRACTICES (RACISMMAFF). Financiado por el Ministerio de Ciencia e Innovación del Gobierno de España (MCIN) y el Fondo Europeo de Desarrollo Regional (FEDER), el proyecto RACISMMAFF es mantenido por un equipo de investigación de la Universidad Complutense de Madrid, liderado por las profesoras Elena Domínguez Romero y Marta Carretero, con colaboración de otras instituciones académicas, en particular la Universidad Politécnica de Madrid. Este proyecto se fundamenta en investigaciones previas (EUROEVIDMOD, EVIDISPRAG, STANCEDISC) y tiene como objetivo principal profundizar en el conocimiento sobre cómo los usuarios del lenguaje se posicionan discursivamente en relación con la inmigración y el racismo. De forma crucial, RACISMMAFF no se limita al análisis teórico, sino que busca la aplicación de estos hallazgos en prácticas afectivas de aprendizaje, con la meta de influir positivamente en el discurso de estudiantes universitarios y futuros docentes de la Unión Europea, y así permear en la sociedad. El proyecto se centra en el análisis de expresiones de posicionamiento (epistémico, afectivo y representación del discurso) para fomentar el pensamiento crítico y las capacidades interculturales.

El presente Trabajo Fin de Grado se desarrolla precisamente en el seno de esta iniciativa, buscando contribuir a sus objetivos aplicados. La plataforma DeStance, cuyo desarrollo y mantenimiento constituyen el núcleo de este trabajo, se ha concebido como una herramienta pedagógica directamente alineada con los fines de RACISMMAFF. DeStance busca operacionalizar los hallazgos de RACISMMAFF proporcionando un entorno práctico para el análisis del posicionamiento discursivo. Su ámbito de uso se proyecta hacia estudiantes universitarios, en línea con el público objetivo de RACISMMAFF, facilitando el reconocimiento y la reflexión crítica sobre las estrategias de posicionamiento en textos reales. Funcionalmente, DeStance se encaja dentro del proyecto RACISMMAFF como una aplicación pedagógica concreta de sus principios de investigación y de su enfoque en el aprendizaje afectivo, buscando dotar a los estudiantes de herramientas para identificar y cuestionar los discursos que perpetúan la desigualdad y la xenofobia, y fomentando así una mayor conciencia intercultural y empatía.

1.2 Problema General y Justificación: Desinformación y Necesidad de Alfabetización Mediática

En la era digital actual, la proliferación de desinformación y discursos polarizantes, especialmente en temas de alta sensibilidad social como la inmigración y el racismo, representa un desafío considerable [5]. Este fenómeno, a menudo denominado "desorden informativo" [5], no solo dificulta el acceso a información veraz, sino que también puede exacerbar tensiones sociales y estigmatizar a colectivos vulnerables [6]. Ante esta situación, emerge la necesidad imperativa de dotar a los ciudadanos, y específicamente a los usuarios de plataformas educativas como DeStance, de competencias en alfabetización mediática crítica. Esto implica no solo consumir información, sino también analizarla, identificando sesgos, evaluando fuentes y comprendiendo las estrategias discursivas empleadas, como el posicionamiento (stance) del autor [1] [19].

Según el informe del Reuters Institute Digital News Report 2024 [12], el 59% de los encuestados expresan preocupación por distinguir entre hechos y ficción en las noticias online, un incremento del 3% respecto a 2023. En el contexto específico de la inmigración, la influencia de las redes sociales es particularmente potente; un estudio reciente sobre la difusión del sentimiento antiinmigrante en Twitter en el Reino Unido reveló que, aunque la comunidad antiinmigración es más pequeña, es significativamente más densa y activa, con el 1% de los usuarios más activos generando más del 23% de los *tweets* antiinmigración y el 21% de los *retweets*, y su contenido se propaga 1.66 veces más rápido que los mensajes proinmigración [13]. Estas dinámicas de amplificación y rápida diseminación subrayan la urgencia de desarrollar herramientas educativas efectivas para combatir la desinformación y fomentar un discurso online más equilibrado.

1.3 Solución Propuesta: El Módulo Interactivo "Stance & Source Check"

Para responder a esta necesidad dentro del contexto de RACISMMAFF, este TFG se centra en la **concepción, diseño e implementación de una nueva solución software**: un módulo interactivo e integrado en DeStance, bautizado como "**Stance & Source Check**". El propósito fundamental de este módulo es ofrecer a los usuarios una herramienta práctica y guiada para entrenar su capacidad de análisis crítico. A través de la interacción con fragmentos de noticias y publicaciones en redes sociales, en adelante denominados “snippets”, los usuarios aprenderán a identificar marcadores lingüísticos clave de posicionamiento y a realizar una evaluación básica de la fiabilidad de las fuentes de información.

La interactividad del módulo permitirá a los estudiantes no solo consumir información, sino también aplicar los conocimientos adquiridos, recibir retroalimentación y comparar sus análisis, promoviendo así un aprendizaje más profundo y significativo. Al centrarse en la "fuente" (Source Check), el módulo también busca desarrollar competencias en la evaluación de la credibilidad y el posible sesgo de la información, una habilidad indispensable en la era de la desinformación. De esta manera, "Stance & Source Check" aspira a ser una

contribución tangible a los objetivos del proyecto RACISMMAFF, empoderando a los estudiantes para que se conviertan en consumidores y productores de información más conscientes, críticos y responsables, capaces de identificar y contrarrestar los discursos de odio y las narrativas simplificadas que a menudo caracterizan el debate público sobre inmigración y diversidad.

1.4 Relevancia y Retos desde la Ingeniería Informática

Desde la perspectiva de la Ingeniería Informática, este proyecto representa una oportunidad para aplicar conocimientos técnicos avanzados en la resolución de un problema educativo y social relevante. La principal motivación y, a su vez, el mayor reto, reside en la construcción de un sistema software robusto, usable y mantenible que cumpla con los requisitos funcionales y se integre fluidamente en una plataforma existente.

1.4.1 Desafío en la Interacción con Texto: Selección y Resaltado Avanzado

Un aspecto técnico particularmente desafiante es la implementación de la funcionalidad de selección y resaltado interactivo de texto (*highlighting*). Permitir que los usuarios seleccionen fragmentos de texto de forma precisa dentro del navegador, los clasifiquen y visualicen estos resaltados de manera persistente y fiable, especialmente cuando las selecciones pueden cruzar múltiples elementos del DOM, requiere una solución técnica sofisticada. Tras evaluar las limitaciones de las APIs nativas del navegador para casos complejos, se tomó la decisión estratégica de adoptar Rangy.js, una librería JavaScript especializada en la manipulación avanzada de rangos y selecciones, como pilar tecnológico para esta funcionalidad. Esto implica el reto de integrar y utilizar eficazmente esta librería dentro del ecosistema reactivo de SvelteKit.

1.4.2 Desafío en la Arquitectura Full-Stack y Tecnologías Específicas

El desarrollo implica trabajar sobre la pila tecnológica completa de DeStance:

- **Frontend:** Utilizar SvelteKit para construir una interfaz de usuario reactiva, dinámica y eficiente, gestionando el estado de la aplicación mediante *stores* y asegurando una comunicación fluida con el backend.
- **Backend:** Desarrollar servicios y endpoints API RESTful en Node.js (con Express) que sean escalables y seguros, gestionando la lógica de negocio y la interacción con la base de datos.
- **Base de Datos:** Diseñar y gestionar modelos de datos NoSQL (MongoDB/Mongoose) capaces de almacenar eficientemente tanto el contenido curado (snippets) como las ricas interacciones del usuario, incluyendo las selecciones serializadas generadas por Rangy.js.

- **Integración:** Asegurar que el nuevo módulo se integre sin fisuras con los sistemas de autenticación, navegación y estilos visuales de la plataforma DeStance.

1.5 Objetivos del Trabajo Fin de Grado

El **objetivo general** de este TFG es diseñar e implementar el módulo interactivo "Stance & Source Check" en la plataforma DeStance, aplicando principios y metodologías de ingeniería del software para asegurar su funcionalidad, integración y mantenibilidad.

Los **objetivos específicos** que guían el desarrollo son:

1. Diseñar la **arquitectura software detallada** del módulo, incluyendo modelos de datos avanzados y una API RESTful granular, conforme al plan técnico revisado.

2. Implementar los **componentes del backend** (Node.js/Express/Mongoose) para gestionar la lógica de negocio y los endpoints API definidos.

3. Desarrollar los **componentes interactivos del frontend** (SvelteKit/TypeScript), implementando la funcionalidad de **selección y resaltado de texto con Rangy.js** y la gestión del estado de la interfaz.

4. Integrar el módulo en la plataforma DeStance, asegurando la coherencia visual (UI/UX) y funcional.

5. Aplicar un **plan de pruebas software** (unitarias, integración, E2E) para validar la funcionalidad y robustez del módulo.

1.6 Metodología de Desarrollo Software Aplicada

El proyecto seguirá un **enfoque iterativo e incremental**, adaptando principios de metodologías ágiles. Se priorizará la entrega temprana de funcionalidades clave para permitir la prueba y el *feedback*. Las fases principales incluirán:

Análisis de Requisitos -> Diseño de Arquitectura y UI/UX -> Implementación Backend -> Implementación Frontend -> Integración -> Pruebas (Funcionales, Usabilidad, E2E) -> Documentación.

Se utilizará Git para el control de versiones y se seguirán las guías de estilo y buenas prácticas del proyecto DeStance.

1.7 Estructura de la Memoria

La presente memoria documenta el proceso de concepción, diseño, implementación y validación del módulo "Stance & Source Check". Tras esta Introducción (Capítulo 1), el Capítulo 2 establece el marco conceptual y tecnológico necesario para comprender el contexto y las decisiones de diseño.

El Capítulo 3 detalla el diseño de la arquitectura software y la funcionalidad del módulo. El Capítulo 4 describe el proceso de implementación técnica. El Capítulo 5 presenta la estrategia de pruebas llevada a cabo y su validación. Finalmente, el Capítulo 6 expone las conclusiones principales del trabajo, evalúa el cumplimiento de los objetivos y propone líneas de trabajo futuro.

2. Marco Conceptual y Fundamento Tecnológico

Para abordar el diseño e implementación del módulo 'Stance & Source Check', es necesario establecer una base conceptual que justifique su funcionalidad y un fundamento tecnológico que describa las herramientas empleadas. Este módulo se integra como una nueva funcionalidad dentro de DeStance, una plataforma educativa existente diseñada para el desarrollo y evaluación de competencias interculturales en el ámbito universitario, enmarcada en el proyecto de investigación RACISMMAFF. DeStance proporciona un Entorno Virtual de Aprendizaje (EVA) donde los estudiantes acceden a diversos cursos y actividades. El módulo 'Stance & Source Check' se concibe como el 'Curso 4' dentro de esta plataforma, enfocándose específicamente en dotar a los estudiantes de herramientas para el análisis crítico de la desinformación, particularmente en temas de inmigración y racismo. Este capítulo explora los conceptos clave que sustentan la necesidad y el diseño pedagógico del módulo, las tecnologías fundamentales para su construcción interactiva, la arquitectura de la plataforma DeStance en la que se apoya, y las consideraciones de diseño UX/UI específicas para software educativo.

2.1 Fundamentos Conceptuales del Módulo

El módulo "Stance & Source Check" es una herramienta tecnológica diseñada para responder a una problemática social y educativa urgente. Su desarrollo se sustenta en la comprensión de conceptos clave como la desinformación, la alfabetización mediática crítica y el análisis del posicionamiento discursivo. Estos fundamentos no solo justifican la existencia del módulo, sino que también informan su diseño funcional y pedagógico dentro del entorno interactivo que propone.

2.1.1 Desinformación y Fake News: Impacto en Inmigración y Racismo

El entorno digital contemporáneo se caracteriza por una sobreabundancia de información, donde la *desinformación* (información falsa creada con intención de dañar) y las *fake news* (noticias falsas) se propagan con gran facilidad [5]. Este fenómeno, conocido como "desorden informativo" [5], tiene un impacto particularmente pernicioso en debates sobre temas socialmente sensibles como la inmigración y el racismo. Frecuentemente, se difunden narrativas que emplean datos falsos, tergiversaciones o discursos de odio para estigmatizar a colectivos migrantes o minorías étnicas, alimentando la polarización y la xenofobia [6]. La ingeniería informática puede contribuir significativamente desarrollando herramientas que ayuden a los usuarios a navegar este complejo ecosistema informativo. El módulo "Stance & Source Check" se crea como una de estas herramientas, buscando mitigar el impacto de la desinformación al capacitar al usuario en su detección y análisis.

2.1.2 Alfabetización Mediática Crítica: Herramientas y Enfoques

La respuesta fundamental a la desinformación no reside únicamente en la detección automática, sino en el fomento de la **alfabetización mediática crítica** en los individuos. Este concepto va más allá de saber usar la tecnología; implica desarrollar la capacidad de analizar, evaluar y cuestionar los mensajes recibidos, identificando sus fuentes, intenciones y posibles sesgos. En el ámbito educativo, se promueven enfoques prácticos que enseñan a los usuarios estrategias concretas, como la verificación lateral de fuentes o el análisis detenido del lenguaje utilizado [2]. El módulo "Stance & Source Check" se diseña como una herramienta tecnológica que busca implementar pedagógicamente estos enfoques, ofreciendo un entorno interactivo para la práctica guiada.

2.1.3 Análisis del Posicionamiento Discursivo (Stance)

Un componente clave del análisis crítico de textos, especialmente de aquellos con potencial persuasivo o manipulador, es la comprensión del **posicionamiento discursivo** o *stance*. Desde la lingüística, se entiende como la forma en que un autor expresa sus actitudes, evaluaciones y grado de certeza sobre lo que dice [1]. Identificar los **marcadores lingüísticos** a través de los cuales se manifiesta este posicionamiento es una habilidad fundamental para detectar sesgos.

La taxonomía de siete marcadores lingüísticos especializados adoptada en este TFG (**epistemic-high**, **epistemic-low**, **emotional-negative**, **emotional-positive**, **bias-loaded**, **authority-claim**, **statistical-claim**) se basa en marcos teóricos del análisis del discurso y la evaluación en el lenguaje, como los propuestos por Martin y White [1] en su desarrollo de la Teoría de la Valoración (Appraisal Theory). Cada categoría ha sido seleccionada por su relevancia para el análisis de la desinformación:

- **Marcadores Epistémicos (alta/baja certeza):** Permiten evaluar la seguridad con la que el autor presenta la información, crucial para detectar afirmaciones infundadas (p.ej., "claramente", "parece que") [15].
- **Marcadores Emocionales (positivo/negativo):** Revelan el uso de lenguaje cargado afectivamente para influir en la percepción del lector (p.ej., "terrible crisis", "excelente solución") [15].
- **Lenguaje Sesgado (Bias-loaded):** Identifica términos inherentemente no neutrales que buscan predisponer al lector (p.ej., "invasión de inmigrantes" vs. "llegada de inmigrantes") [16].
- **Apelación a la Autoridad (Authority-claim):** Destaca el uso de referencias a expertos o figuras de autoridad, que pueden ser legítimas o falaces (p.ej., "científicos afirman") [16].
- **Afirmación Estadística (Statistical-claim):** Señala el uso de datos numéricos, que pueden ser precisos, imprecisos o manipulados (p.ej., "un 90% de los casos") [17].

La identificación de estos marcadores dota al estudiante de un metalenguaje para articular su análisis crítico.

Desde una perspectiva técnica, aunque el análisis profundo del stance mediante Procesamiento del Lenguaje Natural (PLN) es un campo complejo [7], nuestro módulo se enfoca en ayudar al usuario a identificar estos marcadores, lo que presenta retos específicos para el diseño de la **interfaz de usuario** y la **interacción con el texto**.

2.2 Estado del Arte en Tecnologías para la Interacción con Texto

Una de las funcionalidades centrales y técnicamente más exigentes del módulo "Stance & Source Check" es la capacidad del usuario para seleccionar fragmentos de texto dentro de los snippets y aplicarles una clasificación mediante highlighting. Implementar esta interacción de forma robusta en el entorno heterogéneo de los navegadores web presenta desafíos de ingeniería específicos.

2.2.1 Desafíos del Resaltado (Highlighting) en el DOM

El Document Object Model (DOM) representa el contenido HTML como un árbol de nodos. Cuando un usuario selecciona texto, esta selección puede abarcar desde una porción de un único nodo de texto hasta múltiples nodos, incluyendo diferentes párrafos, elementos de formato (negritas, cursivas) o incluso imágenes. Los principales desafíos técnicos son:

- **Identificación Precisa**: Determinar exactamente qué nodos y qué porciones (*offsets*) de esos nodos han sido seleccionados.
- **Manejo de Nodos Cruzados**: Gestionar selecciones que empiezan en un nodo HTML y terminan en otro.
- **Visualización**: Aplicar estilos visuales (resaltado) a la selección sin romper la estructura del DOM ni interferir con otras funcionalidades.
- **Persistencia**: Guardar la información de la selección de forma que pueda ser reconstruida fielmente más tarde, incluso si el DOM se renderiza de nuevo.

2.2.2 APIs Nativas del Navegador y sus Limitaciones

Los navegadores modernos proporcionan APIs nativas para abordar estos problemas:

- `window.getSelection()`**:** Devuelve un objeto `Selection` que representa el rango de texto seleccionado actualmente.
- **API** `Range`**:** Ofrece una interfaz para manipular fragmentos del documento, incluyendo sus nodos de inicio y fin, y sus *offsets* dentro de esos nodos [8]. Permite operaciones como obtener el texto, borrarlo o,

crucialmente, envolverlo con nuevos elementos (`range.surroundContents()`).

Si bien estas APIs son la base, su uso directo para aplicaciones complejas presenta limitaciones significativas:

- **Inconsistencias entre Navegadores:** Históricamente, han existido diferencias sutiles en la implementación entre navegadores.
- **Complejidad de `surroundContents`:** La función para envolver puede fallar o comportarse de forma inesperada con selecciones complejas.
- **Serialización:** Las APIs nativas no ofrecen un método estándar y robusto para *serializar* (guardar como string o JSON) y *deserializar* (restaurar) una selección de forma que sobreviva a recargas o cambios en el DOM. Implementar esto manualmente es propenso a errores.

2.2.3 Librerías JavaScript para Manipulación de Rangos: Introducción a Rangy.js

Dadas las limitaciones de las APIs nativas y la complejidad inherente al problema, se investigaron librerías JavaScript diseñadas para abstraer y simplificar la manipulación de rangos y selecciones. De entre las opciones disponibles, se seleccionó **Rangy.js** para este proyecto.

Si bien las **APIs nativas del navegador** (`window.getSelection()` y la API `Range`) han mejorado significativamente y constituyen la base para la manipulación de selecciones, su uso directo para aplicaciones complejas como la requerida presenta aún desafíos. Estos incluyen la gestión de la serialización robusta de selecciones a través de recargas de página o cambios en el DOM, y el manejo consistente de highlighting a fragmentos de texto que pueden cruzar múltiples nodos HTML, especialmente si estos nodos son dinámicamente alterados por frameworks como SvelteKit. Aunque se consideró el uso exclusivo de estas APIs nativas, la complejidad de implementar manualmente una serialización y una lógica de `'class applier'` a prueba de errores y compatible entre navegadores excedía el alcance práctico del TFG, desviando el foco del desarrollo de la lógica pedagógica del módulo.

Otras librerías de highlighting más recientes tienden a enfocarse en casos de uso más simples o no ofrecen el mismo nivel de control granular sobre la serialización y la aplicación de clases que Rangy.js, o bien introducen un mayor acoplamiento con frameworks específicos de UI de una manera que no era deseable para la abstracción buscada.

Rangy.js [9] es una librería madura y ampliamente utilizada que:

- **Abstrae las Diferencias:** Proporciona una API unificada y consistente que funciona a través de la mayoría de los navegadores modernos.
- **Ofrece Módulos Avanzados:** Incluye módulos específicos para:
    - **Serialización:** Permite guardar y restaurar selecciones de forma fiable, crucial para nuestro requisito de persistencia.

    - **Aplicación de Clases CSS (`ClassApplier`):** Facilita enormemente la tarea de aplicar y quitar estilos (resaltados) a las selecciones, manejando internamente los casos complejos de envoltura y división de nodos.

- **Mantenimiento:** Aunque su desarrollo principal puede no ser tan activo como antes (dado que las APIs nativas han mejorado), su base es sólida y probada.

La **decisión de ingeniería** de incorporar Rangy.js se basa en una evaluación coste-beneficio: aunque introduce una dependencia externa, el ahorro de tiempo y la robustez ganada en la implementación de la funcionalidad de *highlighting* (el núcleo técnico del frontend) superan con creces el coste de integración. Permite centrar los esfuerzos de desarrollo en la lógica de la aplicación y la UX, en lugar de reinventar soluciones complejas y propensas a errores para la manipulación del DOM.

2.3 Arquitectura Tecnológica de la Plataforma DeStance

El módulo "Stance & Source Check" no se construye en el vacío, sino que se integra como una nueva pieza dentro de la plataforma DeStance existente. Comprender la arquitectura y las tecnologías base de DeStance es fundamental para justificar las decisiones de diseño e implementación tomadas en este TFG, asegurando la coherencia y la correcta integración del nuevo módulo. La plataforma sigue un patrón de **arquitectura *full-stack*** moderno, desacoplando las responsabilidades del cliente (*frontend*) y del servidor (*backend*).

2.3.1 Frontend: SvelteKit y Principios Reactivos

La interfaz de usuario de DeStance está construida con **SvelteKit**, un *framework* para el desarrollo de aplicaciones web basado en **Svelte**. A diferencia de otros frameworks populares que operan principalmente en tiempo de ejecución (como React o Vue), Svelte funciona como un **compilador**: convierte los componentes `.svelte` en código JavaScript altamente optimizado durante el proceso de *build*. Esto a menudo se traduce en aplicaciones más rápidas y con un menor peso inicial.

SvelteKit extiende Svelte proporcionando funcionalidades cruciales para una aplicación completa:

- **Enrutamiento Basado en Ficheros:** La estructura de directorios en `src/routes` define las rutas de la aplicación, simplificando la gestión de las diferentes páginas y vistas.

- **Renderizado Híbrido:** Permite combinar renderizado en el lado del servidor (SSR) y en el lado del cliente (CSR), optimizando tanto el rendimiento inicial como la interactividad.

- **Carga de Datos y Acciones del Servidor:** A través de ficheros como `+page.server.ts`, SvelteKit facilita la obtención de datos y la ejecución

de lógica en el servidor antes de renderizar una página, así como la gestión de envíos de formularios (`actions`).

- **Gestión de Estado:** Svelte incorpora un sistema de reactividad y *stores* (simples, derivables) para gestionar el estado de la aplicación de forma eficiente.

Para este TFG, SvelteKit proporciona el entorno ideal para construir la **interfaz interactiva** del módulo, gestionar el **estado de las selecciones del usuario** y comunicarse con el *backend*.

2.3.2 Backend: Node.js, Express y API RESTful

El *backend* de DeStance se ha desarrollado utilizando **Node.js**, un entorno de ejecución de JavaScript del lado del servidor. Esto permite utilizar un lenguaje coherente en todo el *stack* (JavaScript/TypeScript) y aprovechar el vasto ecosistema de paquetes de Node (npm).

Sobre Node.js, se utiliza **Express**, un *framework* web minimalista y flexible, para construir la **API RESTful** que sirve como puente de comunicación entre el *frontend* y la base de datos. La API define una serie de *endpoints* (rutas) que el frontend puede consumir para:

- Obtener datos (como los *snippets* de noticias).
- Enviar datos (como las respuestas y selecciones del usuario).
- Gestionar la autenticación y autorización de usuarios (mediante *middleware* como `check-auth.js`).

La arquitectura del *backend* sigue un patrón **Controlador-Servicio-Modelo**, donde las rutas llaman a controladores, estos implementan la lógica de negocio (a veces delegando en servicios) y los modelos interactúan con la base de datos.

2.3.3 Persistencia de Datos: MongoDB y Mongoose ODM

Para el almacenamiento de datos, DeStance utiliza **MongoDB**, una base de datos **NoSQL orientada a documentos**. Este tipo de base de datos es particularmente adecuado para aplicaciones web modernas por su flexibilidad en el esquema y su escalabilidad. En MongoDB, los datos se almacenan en formato BSON (similar a JSON), lo que facilita su manipulación desde JavaScript/TypeScript.

Para interactuar con MongoDB desde el entorno Node.js, se emplea **Mongoose**, una librería ODM (*Object Data Modeling*). Mongoose permite:

- **Definir Esquemas:** Establecer una estructura (aunque flexible) para los documentos de cada colección (ej. `users`, `questionnaires`, y ahora `fakeNewsSnippets`, `userFakeNewsResponses`).

- **Validar Datos:** Aplicar reglas de validación antes de guardar los datos.
- **Modelar Relaciones:** Gestionar referencias entre diferentes colecciones.
- **Abstraer Consultas:** Proporcionar una API más intuitiva y orientada a objetos para realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar) sobre la base de datos.

Esta pila tecnológica (SvelteKit + Node.js/Express + MongoDB) proporciona una base sólida y moderna sobre la cual se implementará el módulo "Stance & Source Check", asegurando la compatibilidad y la posibilidad de integración con las funcionalidades existentes de la plataforma DeStance.

2.4 Consideraciones de Diseño UX/UI para Software Educativo Interactivo

Más allá de la robustez técnica, el éxito de una herramienta como "Stance & Source Check" depende intrínsecamente de su capacidad para ofrecer una **experiencia de usuario (UX)** eficaz, atractiva y, sobre todo, que facilite el aprendizaje. El diseño de software educativo presenta consideraciones particulares, donde la usabilidad debe ir de la mano de la pedagogía.

2.4.1 Principios de Usabilidad y Accesibilidad (a11y)

Para garantizar que el módulo sea fácil y agradable de usar, su diseño se fundamentará en principios heurísticos de usabilidad consolidados, como los propuestos por Jakob Nielsen [3]. Esto implica asegurar:

- **Visibilidad del Estado del Sistema:** El usuario debe saber en todo momento en qué fase del análisis se encuentra (leyendo, seleccionando, verificando, recibiendo feedback) a través de indicadores visuales claros.
- **Control y Libertad del Usuario:** Permitir deshacer acciones (como eliminar un *highlight*) y navegar de forma predecible.
- **Consistencia:** Mantener la coherencia visual y de interacción con el resto de la plataforma DeStance, utilizando sus guías de estilo existentes.
- **Prevención de Errores y Ayuda:** Diseñar la interfaz para minimizar errores y ofrecer ayuda contextual (ej. *tooltips* explicando los tipos de marcadores).
- **Diseño Minimalista:** Evitar la sobrecarga cognitiva, mostrando solo la información y controles relevantes en cada paso.

Adicionalmente, se prestará atención a la **accesibilidad web (a11y)**, siguiendo las directrices WCAG [4]. Esto es especialmente relevante en la funcionalidad de highlighting, asegurando que los colores tengan suficiente contraste y que, en la medida de lo posible, la interacción sea manejable mediante teclado o tecnologías de asistencia.

2.4.2 Patrones de Diseño para Interacción y Feedback Educativo

El diseño de la interacción del módulo "Stance & Source Check" se beneficia del estudio de patrones específicos para software educativo y herramientas de de análisis crítico [10]. La implementación adopta un enfoque de **flujo guiado estructurado** que conduce al usuario a través de cuatro fases secuenciales diseñadas para desarrollar progresivamente sus habilidades de pensamiento crítico.

Flujo de Interacción de Cuatro Fases

El módulo implementa un **patrón de análisis progresivo** que separa la respuesta emocional del análisis racional:

1. **Captura de Evaluación Inicial de Fiabilidad:** Se registra la percepción preliminar e instintiva del usuario sobre la fiabilidad del snippet antes de emprender un análisis detallado. Este paso inicial tiene como objetivo pedagógico fomentar la **metacognición**, permitiendo al estudiante tomar conciencia de sus juicios iniciales y compararlos posteriormente con la evaluación razonada que realizará tras el análisis de los marcadores lingüísticos y de la fuente. Establece una línea base para la reflexión sobre cómo un escrutinio más profundo puede confirmar o modificar las primeras impresiones.

2. **Análisis Interactivo de Texto**: Fase de highlighting donde el usuario identifica marcadores lingüísticos utilizando Rangy.js para una interacción robusta con el DOM. Esta fase implementa el patrón de **aprendizaje por descubrimiento guiado**.

3. **Evaluación Estructurada de Fuentes**: Checklist sistemático que aplica el patrón de **verificación metodológica**, donde cada criterio de evaluación se presenta como una decisión binaria con explicación contextual.

4. **Feedback Personalizado**: Sistema de retroalimentación que compara las selecciones del usuario con los marcadores identificados por expertos, proporcionando métricas cuantitativas y recomendaciones específicas.

Interacción de Highlighting Avanzada

La funcionalidad de highlighting representa el núcleo técnico más complejo del módulo. Utilizando Rangy.js como solución de ingeniería para las limitaciones de las APIs nativas del navegador, el sistema implementa:

- **Selección Fluida Multi-Elemento**: Gestión de selecciones que cruzan múltiples nodos del DOM mediante ClassAppliers especializados para cada tipo de marcador lingüístico.

- **Popup Contextual Inteligente**: Interfaz emergente que presenta siete categorías de marcadores (*epistemic-high, epistemic-low, emotional-*

negative, emotional-positive, bias-loaded, authority-claim, statistical-claim) con ejemplos y descripciones pedagógicas.

- **Persistencia de Selecciones**: Serialización automática mediante Rangy que permite reconstruir exactamente las selecciones del usuario incluso después de recargas o cambios en el DOM.

Feedback Educativo Progresivo

El sistema de feedback implementa un **patrón de evaluación multidimensional** que proporciona:

- **Métricas Cuantitativas**: Cálculo automático de precisión en identificación de marcadores (`selectionsAccuracy`) y evaluación de fuentes (`sourceAccuracy`), con una puntuación general ponderada.

- **Retroalimentación Cualitativa**: Generación de explicaciones textuales personalizadas basadas en la comparación con el análisis experto del snippet.

- **Recomendaciones Adaptativas**: Sugerencias específicas para mejorar las habilidades de análisis crítico, adaptadas al patrón de errores observado en la sesión.

El diseño de la UX implementa principios de **gamificación educativa** [18] mediante indicadores visuales de progreso, validaciones que habilitan progresivamente las secciones, y un sistema de logros implícito que motiva la práctica continuada. La integración con el ecosistema MOOC existente asegura coherencia visual y funcional, manteniendo la curva de aprendizaje establecida en los cursos 1-3 mientras introduce la complejidad necesaria para el análisis de desinformación.

3. Diseño del Módulo "Stance & Source Check"

3.1 Ingeniería de Requisitos

La ingeniería de requisitos constituye el proceso fundamental para comprender y definir las funcionalidades que debe implementar el módulo "Stance & Source Check" dentro de la plataforma DeStance. Este proceso se centra en capturar las necesidades funcionales y no funcionales que permitirán a los usuarios desarrollar habilidades críticas para el análisis de desinformación en temas de inmigración y racismo, cumpliendo con los objetivos pedagógicos del proyecto RACISMMAFF.

El objetivo principal de este módulo es dotar a los usuarios de habilidades para identificar marcadores lingüísticos de posicionamiento (stance markers) y evaluar la fiabilidad de las fuentes de información, fomentando así una mayor alfabetización mediática y una respuesta más crítica ante la desinformación.

3.1.1 Requisitos Funcionales (RF)

Los requisitos funcionales describen las **acciones y operaciones** que el sistema debe ser capaz de realizar. Se han identificado los siguientes requisitos funcionales clave, organizados por complejidad de implementación:

- **RF1: Presentación de Snippets**
  - **Descripción:** El sistema debe presentar al usuario snippets predefinidos aleatoriamente, priorizando aquellos que el usuario no ha completado previamente. Estos snippets contendrán texto y, opcionalmente, una imagen asociada para proporcionar un contexto visual.
  - **Detalles:** Los snippets incluyen soporte para múltiples formatos (*news-article, tweet*) y categorización temática (*immigration, racism, general*). La selección implementa un algoritmo que evita repeticiones y asegura diversidad de tipos de contenido.
  - **Prioridad:** Alta.
  - **Estado:** Completo
  - **Objetivo de aprendizaje:** Exponer al estudiante a diversos ejemplos de información y desinformación, simulando un entorno mediático real, para iniciar el proceso de análisis crítico.

- **RF2: Sistema de Highlighting Interactivo Avanzado**
  - **Descripción:** El sistema debe permitir al usuario seleccionar porciones de texto dentro del snippet presentado.
  - **Detalles:** Se utilizará la librería **Rangy.js** para permitir selecciones complejas y gestionar el resaltado visual del texto seleccionado.
  - **Prioridad:** Alta.
  - **Estado:** Completo

- **Objetivo de aprendizaje:** Capacitar al estudiante en la técnica de seleccionar y aislar fragmentos textuales específicos como base para un análisis lingüístico detallado.

- **RF3: Clasificación de Marcadores Lingüísticos Especializados**
  - **Descripción:** Una vez que el usuario ha seleccionado una porción de texto, el sistema debe permitirle clasificarla según una lista predefinida de siete tipos de marcadores lingüísticos especializados.
  - **Detalles:** Tipos implementados: epistemic-high (alta certeza), epistemic-low (incertidumbre), emotional-negative (emoción negativa), emotional-positive (emoción positiva), bias-loaded (lenguaje cargado), authority-claim (referencias de autoridad), statistical-claim (afirmaciones estadísticas). La interfaz presenta un popup contextual con ejemplos. El texto resaltado adoptará un estilo visual distintivo según la clasificación asignada. El sistema permitirá al usuario ver y modificar sus selecciones y clasificaciones.
  - **Prioridad:** Alta.
  - **Estado:** Completo
  - **Objetivo de aprendizaje:** Desarrollar la habilidad del estudiante para identificar, diferenciar y categorizar diversos tipos de marcadores lingüísticos de posicionamiento (stance) utilizados en los discursos.

- **RF4: Captura de Evaluación Inicial de Fiabilidad**
  - **Descripción:** El sistema debe registrar la evaluación inicial del usuario antes del análisis crítico, implementando la separación metodológica entre pre-análisis y pos-análisis.
  - **Detalles**: Captura de reacción categórica (confiable, mixto, no confiable) con escala de confianza (1-5). Esta funcionalidad permite estudios posteriores sobre la evolución entre la respuesta inicial y la respuesta final tras el ejercicio de análisis crítico.
  - **Prioridad:** Media.
  - **Estado:** Completo
  - **Objetivo de aprendizaje:** Fomentar la autoconciencia del estudiante sobre sus propias reacciones emocionales iniciales ante la información, reconociendo posibles sesgos afectivos antes de un análisis racional.

- **RF5: Evaluación Estructurada de Credibilidad de Fuentes**
  - **Descripción:** El sistema debe proporcionar un checklist sistemático de cinco criterios específicos para evaluar la credibilidad de la fuente del snippet.
  - **Detalles:** Criterios implementados: `hasAuthor` (identificación clara del autor), `hasDate` (fecha de publicación), `hasCredibleSource` (reputación de la fuente), `hasEvidence` (evidencia o datos de respaldo), `hasMultipleSources` (múltiples perspectivas), `avoidsLoadedLanguage` (no usa lenguaje cargada que provoca al lector=. Incluye cálculo automático de score de confiabilidad y captura del veredicto final del usuario.

- **Prioridad:** Alta.
- **Estado:** Completo
- **Objetivo de aprendizaje:** Guiar al estudiante en la aplicación de un método sistemático para evaluar la fiabilidad de las fuentes de información basándose en criterios establecidos.

- **RF6: Sistema de Feedback Inteligente con Métricas Detalladas**
  - **Descripción:** Tras el envío del análisis, el sistema debe generar feedback personalizado comparando las selecciones del usuario con los marcadores identificados por expertos.
  - **Detalles**: Cálculo de métricas cuantitativas (`selectionsAccuracy`, `sourceAccuracy`, `overallScore`, `timeSpent`), generación de explicaciones cualitativas personalizadas, y provision de recomendaciones específicas para mejora. El sistema incluye la explicación experta completa del snippet analizado.
  - **Prioridad:** Alta.
  - **Estado:** Completo
  - **Objetivo de aprendizaje:** Proporcionar al estudiante una retroalimentación constructiva y personalizada sobre su desempeño en el análisis, identificando aciertos y áreas de mejora para refinar sus habilidades críticas.

- **RF7: Integración Completa con Ecosistema MOOC**
  - **Descripción:** El módulo debe integrarse *seamlessly* como Course 4 dentro del sistema MOOC existente, manteniendo consistencia de navegación, autenticación y experiencia de usuario.
  - **Detalles:** Implementación de dashboard específico con estadísticas de progreso, radar charts de habilidades desarrolladas, y métricas comparativas con otros cursos. Integración con el sistema de tabs del dashboard principal y mantenimiento de patrones de autenticación establecidos.
  - **Prioridad:** Alta.
  - **Estado:** Completo
  - **Objetivo de aprendizaje:** Asegurar una experiencia de usuario fluida y coherente dentro de la plataforma DeStance, permitiendo el seguimiento del progreso en el contexto del ecosistema MOOC existente.

3.1.2 Requisitos No Funcionales (Rendimiento, Usabilidad, Mantenibilidad)

Los requisitos no funcionales describen las **cualidades y restricciones** del sistema, definiendo cómo debe operar en lugar de qué debe hacer específicamente.

- **RNF1: Usabilidad y Experiencia de Usuario**

- **Descripción:** La interfaz debe proporcionar una experiencia intuitiva y educativamente efectiva, siguiendo los principios de diseño establecidos en la plataforma DeStance y las heurísticas de usabilidad de Nielsen [11].
- **Criteros:** Flujo de trabajo claro con validaciones progresivas, feedback visual inmediato en las interacciones de highlighting, y minimización de la carga cognitiva mediante la presentación secuencial de tareas.
- **Métrica:** Tasa de finalización exitosa de análisis > 90% en pruebas de usabilidad.
- **Estado:** Parcial - No se realizaron pruebas de usabilidad formales, por lo que la métrica no se ha medido.

- **RNF2: Rendimiento de Interacción**
  - **Descripción:** La interacción con el resaltado de texto (seleccionar, clasificar, visualizar) debe ser fluida y sin retrasos perceptibles. Las respuestas de la API (obtener snippet, guardar respuesta) deben ser rápidas.
  - **Métrica:** Tiempo de respuesta del resaltado < 200ms; Tiempo de respuesta de la API < 500ms bajo carga normal.
  - **Estado:** Parcial - No se realizaron pruebas de carga formales para API.

- **RNF3: Compatibilidad Cross-Browser**
  - **Descripción:** El módulo debe funcionar correctamente en las últimas versiones de los navegadores web modernos (Chrome, Firefox, Safari, Edge).
  - **Métrica:** Superación de pruebas de compatibilidad en los navegadores objetivo.
  - **Estado:** Cumplido - Probado en Chrome, Firefox, Edge, Safari y Opera.

- **RNF4: Robustez y Manejo de Errores**
  - **Descripción:** El sistema debe ser robusto, manejando errores de forma controlada, por ejemplo, por pérdida de conexión u errores de API. La serialización y deserialización de las selecciones debe ser consistente y precisa.
  - **Métrica:** Tasa de fallos < 1%; Implementación de mecanismos de manejo de errores.
  - **Estado:** Cumplido - Implementado manejo de errores en API y lógica de fallback para deserialización.

- **RNF5: Mantenibilidad y Extensibilidad**
  - **Descripción:** El código debe estar bien estructurado, documentado y seguir las convenciones del proyecto existente para facilitar futuras modificaciones y depuraciones.

- **Métrica:** Revisión de código; en una situación ideal, adherencia a guías de estilo, pero al no existir, al estilo de código presente en el proyecto.
- **Estado:** Cumplido - Código estructurado y comentado, seguimiento de convenciones.

- **RNF6: Seguridad**
  - **Descripción:** Se debe garantizar que solo los usuarios autenticados puedan acceder al módulo y enviar respuestas. La comunicación con la API debe ser segura (HTTPS). Se deben aplicar las políticas de seguridad existentes en DeStance.
  - **Métrica:** Cumplimiento de los requisitos de autenticación y autorización (`checkAuth`); Uso de HTTPS.
  - **Estado:** Cumplido - Uso de checkAuth y HTTPS

- **RNF7: Escalabilidad**
  - **Descripción:** Aunque no es el foco principal del TFG, la arquitectura debe permitir, en la medida de lo posible, un futuro crecimiento en el número de usuarios y snippets sin una degradación significativa del rendimiento.
  - **Métrica:** Diseño modular; Uso eficiente de la base de datos.
  - **Estado:** Parcial – Diseño modular y uso de índices en BD pero no se han realizado pruebas de escalabilidad.

3.2 Diseño de la Arquitectura Software

Para dar respuesta a los requisitos identificados en la sección anterior, se ha diseñado una arquitectura de software basada en la pila tecnológica existente de la plataforma DeStance, adoptando un enfoque de aplicación web moderna con separación de responsabilidades entre el frontend y el backend. A diferencia de los cursos 1-3 del MOOC que manejan análisis de texto simple mediante conteo de palabras clave, el Course 4 requiere una arquitectura técnicamente sofisticada capaz de gestionar interacciones complejas de highlighting, persistencia de selecciones serializadas, y algoritmos de comparación con marcadores prefijados para cada snippet.

La separación de responsabilidades sigue el patrón arquitectónico establecido en la plataforma (SvelteKit + Node.js/Express + MongoDB), pero introduce capas especializadas para el manejo de la complejidad específica del análisis de desinformación y la interacción avanzada con texto.

3.2.1 Vista de Componentes y Diagrama de Arquitectura y Componentes

La arquitectura del módulo se estructura en tres capas principales con componentes especializados que gestionan diferentes aspectos de la funcionalidad educativa e interactiva.

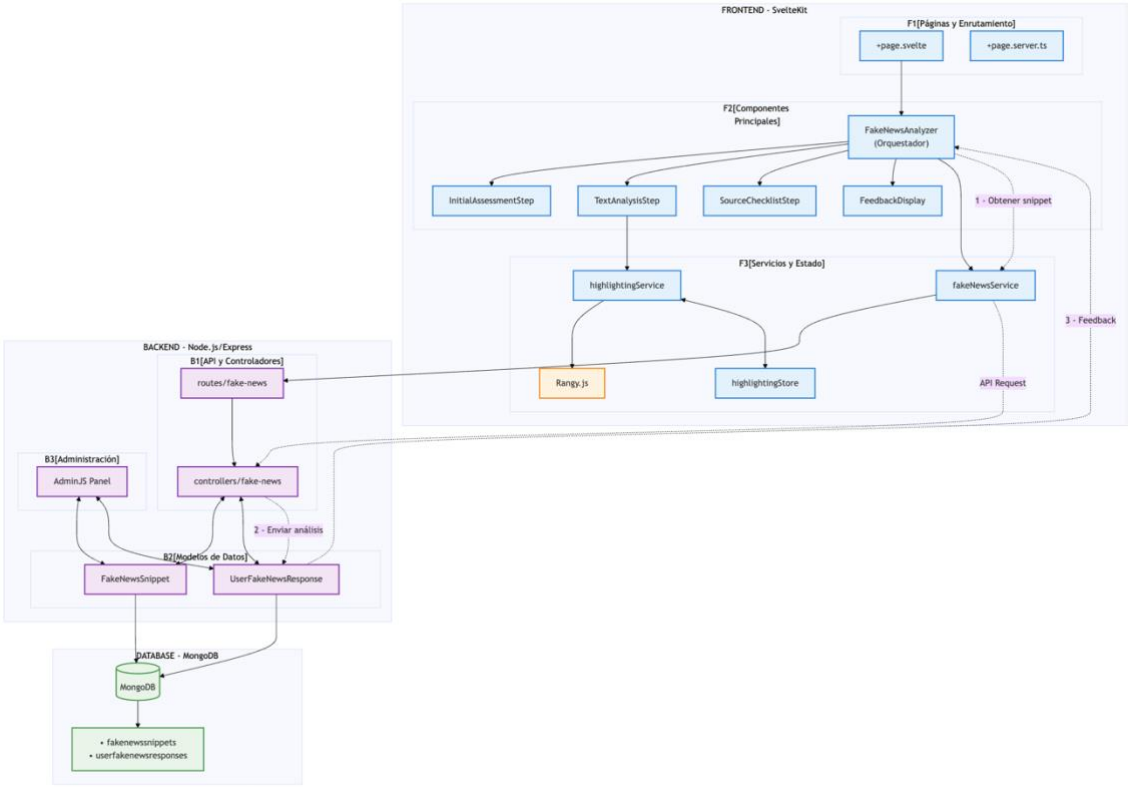


***Figura 3.1**: Diagrama de Despliegue Conceptual en Formato Mermaid (ver el Anexo 1 para ver el código Mermaid).*

Frontend (SvelteKit) - Capa de Presentación

Tecnologías: SvelteKit, TypeScript, Vite, TailwindCSS, SCSS, Rangy.js

Integración con el MOOC Existente:

- **Ruta Principal**: `/routes/mooc/course4/+page.svelte` siguiendo el patrón establecido
- **Integración Dashboard**: Extensión de `/routes/dashboards/mooc/+page.svelte` con `Course4Dashboard.svelte` para integración con el selector de cursos
- **Servidor de Datos**: `+page.server.ts` para carga inicial de datos del servidor
- **Consistencia de Navegación**: Mantenimiento de patrones de autenticación y enrutamiento

Componentes de Análisis Core:

1. **`FakeNewsAnalyzer.svelte`** - Orquestador principal que gestiona el flujo completo de cuatro fases secuenciales, incluyendo la validación de prerrequisitos entre pasos y la gestión del estado global de la sesión de análisis.
2. **`InitialAssessmentStep.svelte`** - Componente especializado para captura de evaluación inicial, implementando escalas de reacción y confianza mediante interfaces de selección optimizadas para diferentes dispositivos.
3. **`TextAnalysisStep.svelte`** - Núcleo de la interacción de highlighting que integra SnippetDisplay con funcionalidades de resumen de selecciones, validación de mínimos requeridos, y navegación hacia la siguiente fase.
4. **`SourceChecklistStep.svelte`** - Implementación del checklist estructurado de evaluación de fuentes con cálculo dinámico de scores de confiabilidad y captura del veredicto final del usuario.
5. **`FeedbackDisplay.svelte`** - Sistema complejo de presentación de resultados que incluye métricas detalladas, comparación con análisis experto, recomendaciones personalizadas, y opciones de navegación hacia nuevos análisis.

Componentes de Interacción Especializada:

6. **`SnippetDisplay.svelte`** - Renderizador de contenido que soporta múltiples formatos (artículos, tweets), gestión de imágenes, y modo interactivo/no-interactivo según el contexto de uso.
7. **`SelectionPopup.svelte`** - Interfaz contextual que presenta los siete tipos de marcadores lingüísticos con descripciones pedagógicas, ejemplos, y diseño optimizado para selección rápida.

Capa de Servicios y Estado:

8. **`fakeNewsService.ts`** - Servicio principal de comunicación con la API que gestiona la obtención de snippets personalizados, envío de análisis, recuperación de historial, y generación de estadísticas de usuario.
9. **`highlightingService.ts`** - Abstracción especializada de Rangy.js que maneja la inicialización de ClassAppliers, serialización/deserialización de selecciones, gestión de eventos de selección, y cleanup de recursos.
10. **`course4Service.ts`** - Adaptador de integración con el sistema MOOC que transforma datos específicos del módulo al formato esperado por los componentes dashboard existentes, incluyendo generación de datos para radar charts.

Gestión de Estado Global:

11. **`highlightingStore.ts`** - Store Svelte especializado que gestiona el estado de todas las selecciones del usuario, configuración del modo de selección, gestión de popup contextual, y acciones para manipulación de highlights.
12. **`fakeNews.ts (types)`** - Sistema completo de tipos TypeScript que define interfaces para todos los aspectos del módulo, incluyendo las siete categorías de marcadores, estructuras de respuesta, y tipos de feedback.

Backend (Node.js/Express) – Capa Lógica

Tecnologías: Node.js, Express.js, Mongoose ODM, AdminJS

Componentes Principales:

1. **Modelos de Datos Especializados**:
   - `models/fake-news-snippet.js` - Esquema avanzado para almacenamiento de fragmentos curados que incluye marcadores expertos con índices precisos, información detallada de fuentes, y metadatos de uso y performance.
   - `models/user-fake-news-response.js` - Modelo complejo para respuestas de usuario que incorpora todas las fases del análisis, selecciones serializadas de Rangy, métricas calculadas, y timestamps detallados.
2. **Controladores de Lógica de Negocio**:
   - `Controllers/fake-news.js` - Implementa algoritmos de personalización de snippets, cálculo de métricas de accuracy mediante comparación con marcadores expertos, generación de feedback inteligente, y gestión de estadísticas agregadas.
3. **API RESTful Granular**:
   - `routes/fake-news.js` - Endpoints especializados que manejan la obtención personalizada de snippets, procesamiento de análisis complejos, y provisión de datos para estadísticas de dashboard.
4. **Integración AdminJS**:
   - Paneles de administración especializados para gestión de contenido de snippets, visualización de respuestas de usuarios, y herramientas de curación de contenido educativo.

> Database (MongoDB) - Capa de Persistencia

Colecciones Especializadas:

- `fakenewssnippets` - Almacenamiento de contenido curado con marcadores expertos, información de fuentes, y metadatos de uso
- `userfakenewsresponses` - Respuestas completas de usuarios con selecciones serializadas, métricas calculadas, y datos de sesión

Integración con Ecosistema Existente:

- Mantenimiento de coherencia con colecciones users existentes
- Aprovechamiento de sistemas de autenticación y autorización establecidos
- Implementación de índices optimizados para consultas de personalización y estadísticas

3.2.2 Diseño de la API RESTful

La API RESTful constituye el puente de comunicación esencial entre el frontend, donde el usuario interactúa con el módulo, y el backend, donde reside la lógica de negocio y el acceso a los datos. El diseño de esta API se enfocó en proporcionar un conjunto de

endpoints cohesivos, semánticamente claros y eficientes, que permitieran al frontend solicitar la información necesaria y enviar los análisis de los usuarios de manera estructurada. Se buscó una API granular, optimizada para el flujo de trabajo específico del análisis de noticias falsas, diferenciándose de las APIs más simples de los cursos preexistentes en la plataforma DeStance.

3.2.2.1 Endpoints Principales del Módulo

Se diseñaron tres endpoints principales bajo el prefijo **`/api/fake-news`**, cada uno respondiendo a una necesidad funcional clave del módulo:

1. **Obtención de Snippets de Noticias para Análisis (`GET /api/fake-news/snippet/:userId`)**
    - **Propósito del Diseño:** Este endpoint se diseñó para proveer al frontend el siguiente snippet que un usuario específico (**`userId`**) debe analizar. Una consideración clave en su diseño fue la necesidad de evitar la repetición de snippets ya completados por el usuario, asegurando así una experiencia de aprendizaje variada y progresiva.
    - **Autenticación:** Se especificó que este endpoint requeriría autenticación, garantizando que solo los usuarios validados puedan acceder al contenido.
    - **Lógica de Negocio Conceptual:** El diseño contempla que el backend, al recibir una petición a este endpoint:
        1. Identifique los snippets previamente analizados y completados por el **`userId`** solicitante.
        2. Seleccione un snippet de entre aquellos marcados como activos y que aún no hayan sido completados por dicho usuario. Se consideró la posibilidad de una selección aleatoria entre los candidatos para diversificar la experiencia.
        3. Registre el uso de este snippet, por ejemplo, incrementando un contador de visualizaciones o usos.
        4. Devuelva la información del snippet al frontend, pero omitiendo deliberadamente los datos correspondientes al análisis experto, como los **`expertMarkers`** o la **`expertExplanation`**, ya que esta información solo debe revelarse al usuario después de que haya completado su propio análisis.
    - **Respuesta Esperada (Conceptual):** En caso de éxito, la API devolvería un mensaje de confirmación junto con los datos del snippet seleccionado: texto, imagen opcional, tipo, formato, tema e información básica de la fuente. Si no hubiera más snippets disponibles para el usuario, se diseñó una respuesta específica para comunicar esta situación.

2. **Envío del Análisis del Usuario (`POST /api/fake-news/response`)**

- **Propósito del Diseño:** Este endpoint se diseñó para permitir al frontend enviar el análisis completo que el usuario ha realizado sobre un snippet. Este es el endpoint más complejo en términos de los datos que maneja y la lógica de procesamiento que desencadena en el backend.

- **Autenticación:** También se especificó que este endpoint requeriría autenticación.

- **Datos Esperados (Conceptual):** El diseño del cuerpo de la petición para este endpoint incluye identificadores clave (`userId`, `snippetId`), la valoración preliminar del usuario, el conjunto de selecciones de texto realizadas y clasificadas (incluyendo su representación serializada generada por Rangy.js), las respuestas del usuario al checklist de evaluación de la fuente, su veredicto sobre la fiabilidad de la misma, y el tiempo empleado en el análisis.

- **Lógica de Procesamiento Conceptual:** Tras recibir estos datos, el diseño contempla que el backend:
    1. Valide la integridad y corrección de los datos recibidos.
    2. Recupere el snippet original con la información experta completa.
    3. Compare las selecciones del usuario con los `expertMarkers` y las respuestas del checklist de la fuente con las respuestas expertas.
    4. Calcule las métricas de rendimiento del usuario: precisión en la identificación de marcadores (`selectionsAccuracy`), precisión en la evaluación de la fuente (`sourceAccuracy`), y una puntuación global (`overallScore`).
    5. Guarde la respuesta completa del usuario, incluyendo las métricas calculadas, en la base de datos.
    6. Genere un objeto de feedback estructurado y personalizado para el usuario.

- **Respuesta Esperada (Conceptual):** En caso de éxito, la API devolvería un mensaje de confirmación, un resumen del rendimiento del usuario (las métricas calculadas) y el objeto de feedback detallado, que incluye la explicación experta del *snippet* y comentarios sobre el análisis del usuario.

3. **Obtención del Historial de Análisis del Usuario (`GET /api/fake-news/history/:userId`)**

    - **Propósito del Diseño:** Este endpoint se diseñó para permitir al frontend solicitar y recuperar el historial de los análisis de snippets que un usuario específico ha completado previamente.

    - **Autenticación:** Requeriría autenticación.

    - **Uso Previsto:** La información obtenida a través de este endpoint está destinada principalmente a poblar el dashboard del Curso 4, donde el usuario puede revisar su progreso, sus puntuaciones pasadas y, potencialmente, acceder a los detalles de sus análisis anteriores.

- **Respuesta Esperada (Conceptual):** La API devolvería un listado de los análisis previos del usuario, incluyendo información clave como el identificador del snippet analizado, un resumen del texto del snippet, el tema, y las principales métricas de rendimiento obtenidas en cada análisis. Se consideró la posibilidad de paginación o limitación del número de resultados devueltos.

3.2.2.2 Diferenciación con APIs Existentes en DeStance

El diseño de la API para el módulo "Stance & Source Check" se diferencia significativamente de las APIs de los cursos 1-3 existentes en la plataforma DeStance. Mientras que las APIs anteriores se centraban en operaciones CRUD más simples sobre datos textuales básicos, como respuestas a preguntas abiertas o selecciones de lengua materna, la nueva API fue diseñada para manejar un flujo de interacción multiestado mucho más complejo y estructuras de datos más ricas. Las principales diferencias radican en:

- **Complejidad del Flujo:** La API del Curso 4 soporta un proceso de análisis secuencial de varios pasos, donde el estado intermedio, como las selecciones de texto, es crucial.

- **Estructura de Datos:** Maneja objetos de datos complejos que incluyen metadatos detallados, selecciones de texto serializadas y métricas de rendimiento calculadas, en contraste con el texto plano o campos simples de los otros cursos.

- **Procesamiento en el Backend:** Implica una lógica de procesamiento más sofisticada en el backend, incluyendo la comparación algorítmica de las respuestas del usuario con datos expertos y la generación dinámica de feedback inteligente.

- **Naturaleza del Feedback:** El feedback proporcionado es mucho más detallado y estructurado, incluyendo métricas cuantitativas, análisis cualitativos y recomendaciones, a diferencia del feedback más básico o directo de otros módulos.

3.2.2.3 Diseño de Algoritmos de Comparación y Generación de Feedback

Una parte integral del diseño de la API y la lógica del backend es cómo se evalúa el trabajo del usuario y cómo se le proporciona feedback útil.

- **Cálculo de Precisión en Selecciones:** Se diseñó una estrategia para calcular la precisión con la que el usuario identifica los marcadores lingüísticos. Conceptualmente, este algoritmo compararía cada selección clasificada por el usuario con el conjunto de marcadores definidos por los expertos para ese snippet. La comparación consideraría no solo la coincidencia del tipo de marcador asignado, sino también la similitud del texto seleccionado. Se previó la necesidad de normalizar el texto, por ejemplo, convirtiéndolo a minúsculas y eliminando espacios extra, y de aplicar una lógica de coincidencia que pudiera tolerar variaciones menores o solapamientos parciales entre la selección del usuario y el marcador experto, en lugar de requerir una coincidencia exacta y literal en todos los casos.

- **Generación de Feedback Personalizado:** El diseño del sistema de feedback se basó en tres pilares:
    1. **Feedback Algorítmico:** Basado directamente en las métricas cuantitativas calculadas: precisión en selecciones, precisión en fuentes, puntuación general.
    2. **Feedback Experto:** Proporcionando al usuario la explicación pre-redactada por los expertos para el snippet analizado, una vez que el usuario ha completado su propio análisis.
    3. **Recomendaciones Adaptativas:** Se conceptualizó la generación de sugerencias o recomendaciones textuales que se adaptaran al rendimiento específico del usuario, señalando áreas de mejora o reforzando aciertos.

3.2.2.4 Estrategia de Manejo de Errores

Para asegurar una comunicación robusta entre el frontend y el backend, se diseñó una estrategia de manejo de errores basada en el uso de códigos de estado HTTP semánticamente correctos. Por ejemplo, un error de validación de datos por parte del cliente resultaría en un código 400 (Bad Request), mientras que un intento de acceso a un recurso no encontrado devolvería un 404 (Not Found). Las respuestas de error incluirían, además del código de estado, un cuerpo en formato JSON con un mensaje descriptivo del error, facilitando así la depuración y la presentación de información útil al usuario desde el frontend.

3.2.2.5 Autenticación y Autorización

El diseño de la API especificó que todos sus endpoints estarían protegidos y requerirían autenticación. Se decidió reutilizar el sistema de autenticación basado en tokens JWT ya implementado en la plataforma DeStance. Esto asegura que solo los usuarios registrados y validados puedan interactuar con el módulo. Adicionalmente, para endpoints que manejan datos específicos de un usuario, como la obtención de su historial, el diseño contempla la validación de la autorización, asegurando que un usuario solo pueda acceder a su propia información.

Este diseño conceptual de la API RESTful establece las bases para una interacción clara, segura y funcional entre el cliente y el servidor, soportando todas las operaciones necesarias para la experiencia de aprendizaje interactiva del módulo "Stance & Source Check". La implementación detallada de estos endpoints y la lógica de los controladores se abordará en el Capítulo 4.

3.2.3 Diseño del Modelo de Datos (MongoDB/Mongoose)

Un aspecto crucial en el diseño de la arquitectura del módulo "Stance & Source Check" es la definición de una estructura de datos robusta y adecuada para almacenar tanto el contenido pedagógico como las interacciones y el progreso de los usuarios. Se optó por una base de datos NoSQL, MongoDB, dada su flexibilidad para manejar estructuras de datos complejas y evolutivas, lo cual es ideal para un proyecto de investigación en desarrollo. El diseño de los modelos de datos se centró en dos entidades principales: los fragmentos de noticias (snippets) que se presentan a los usuarios y las respuestas que los usuarios generan al interactuar con dichos snippets.

3.2.3.1 Modelo de Contenido Curado: `FakeNewsSnippet`

Este modelo almacena los fragmentos de noticias cuidadosamente curados junto con los marcadores identificados, sirviendo como la "verdad fundamental" contra la cual se evalúan las respuestas de los usuarios.

Para representar los fragmentos de noticias, se diseñó un modelo conceptual denominado **`FakeNewsSnippet`**. Este modelo tiene como finalidad encapsular toda la información necesaria para presentar un caso de estudio al usuario y para evaluar su análisis posterior. Los elementos clave que se conceptualizaron para este modelo incluyen:

- **Identificación y Contenido Base:** Un identificador único para el snippet, el texto principal del fragmento de noticia y una URL opcional a una imagen asociada, para proveer contexto visual.

- **Clasificación del Contenido:** Atributos para categorizar el snippet según su veracidad como "fake", "misleading", "real-biased", "real-neutral"..., su formato mediático como "news-article" o "tweet", y su temática principal entre "immigration", "racism" y "general"). Esta clasificación es fundamental para la selección de contenido relevante y para los objetivos pedagógicos del proyecto RACISMMAFF.

- **Marcadores Expertos (`expertMarkers`):** Se diseñó una estructura para almacenar un conjunto de "marcadores lingüísticos expertos". Cada marcador conceptualmente contendría el texto exacto del marcador dentro del snippet, el tipo de marcador lingüístico al que corresponde de entre los siete tipos definidos: **`epistemic-high`**, **`epistemic-low`**, **`emotional-negative`**, **`emotional-positive`**, **`bias-loaded`**, **`authority-claim`**, y **`statistical-claim`**, opcionalemte la posición exacta del marcador en el texto mediante índices de inicio y fin, y una breve explicación pedagógica del marcador. Esta estructura es la "verdad fundamental" contra la cual se comparará el análisis del usuario.

- **Información de la Fuente (`sourceInfo`):** Para fomentar la evaluación crítica de las fuentes, se incluyó una sección para detallar la fuente del snippet. Esta contendría el nombre de la fuente, su dominio web, si aplica, una valoración experta sobre su fiabilidad general, y un conjunto de respuestas expertas a un checklist de criterios de credibilidad, por ejemplo, si la fuente tiene autor identificable, fecha de publicación, si aporta evidencias, etc.

- **Explicación Experta General (`expertExplanation`):** Un campo destinado a una explicación global redactada por un experto sobre las características más relevantes del snippet, sus posibles sesgos, o las estrategias de desinformación que emplea. Esta explicación se presentaría al usuario como parte del feedback.

- **Metadatos de Gestión:** Atributos como un indicador de si el snippet está activo y disponible para los usuarios, quién lo creó, y estadísticas de uso (cuántas veces se ha usado, la precisión media obtenida por los usuarios con él).

3.2.3.2 Modelo de Respuesta del Usuario: `UserFakeNewsResponse`

Para registrar la interacción completa de un usuario con un snippet particular, se diseñó el modelo conceptual **`UserFakeNewsResponse`**. Este modelo está pensado para capturar la progresión del usuario a través del flujo de análisis y su desempeño. Los componentes principales de este diseño son:

- **Identificadores de Vinculación:** Referencias al usuario (**`userId`**) que realizó el análisis y al snippet (**`snippetId`**) que fue analizado, además de un identificador de sesión único para el intento de análisis.

- **Evaluación Inicial de Fiabilidad (`initialAssessment`):** Una estructura para almacenar la evaluación inicial instintiva del usuario antes de iniciar el análisis crítico, incluyendo la naturaleza del análisis (reliable, mixed o unreliable) y su intensidad.

- **Selecciones del Usuario (`userSelections`):** Un listado de las selecciones de texto que el usuario ha realizado y clasificado sobre el snippet. Para cada selección, el diseño contempla almacenar el texto exacto seleccionado, el tipo de marcador lingüístico que el usuario le asignó, y, de forma crucial, una representación serializada de la selección. Esta serialización que se planeó realizar con Rangy.js es clave para poder reconstruir o analizar las selecciones del usuario de forma persistente. También se consideró almacenar información contextual de la selección, como el texto inmediatamente anterior y posterior, y el párrafo donde se encuentra.

- **Análisis de la Fuente por el Usuario (`sourceAnalysis`):** Una estructura para guardar las respuestas del usuario al checklist de evaluación de la credibilidad de la fuente, su veredicto final sobre la fiabilidad de la misma y el nivel de confianza que el usuario tiene en su propia evaluación.

- **Métricas de Rendimiento (`performance`):** Un conjunto de campos destinados a almacenar las métricas calculadas automáticamente después de que el usuario envía su análisis. Esto incluye el tiempo total empleado, la precisión en la identificación de los marcadores lingüísticos, comparando con los `expertMarkers`, la precisión en la evaluación de la fuente, comparando con la `sourceInfo` experta, y una puntuación global del análisis.

- **Estado del Análisis:** Indicadores sobre si el análisis ha sido completado y la fecha de envío.

3.2.3.3 Consideraciones de Diseño Adicionales

Más allá de la estructura de los campos, el diseño de los modelos de datos también contempló la necesidad de:

- **Relaciones y Referencias:** Establecer referencias claras entre **`UserFakeNewsResponse`**, **`User`** (el modelo de usuario general de DeStance) y **`FakeNewsSnippet`**.

- **Validación de Datos:** Definir reglas de validación para los campos como con los tipos de datos, valores permitidos para campos enumerados y rangos para puntuaciones.

- **Extensibilidad:** Diseñar los modelos de forma que puedan ser extendidos en el futuro si surgen nuevos requisitos pedagógicos o de análisis.

- **Optimización para Consultas:** Aunque la optimización detallada con índices es más propia de la fase de implementación, el diseño ya anticipaba la necesidad de realizar consultas eficientes, por ejemplo, para recuperar el historial de un usuario o para seleccionar snippets no completados.

Este diseño conceptual de los modelos de datos sienta las bases para una persistencia de información estructurada y significativa, que no solo soporta la funcionalidad interactiva del módulo, sino que también habilita futuras investigaciones sobre cómo los usuarios interactúan con la desinformación y cómo desarrollan sus competencias críticas. La implementación concreta de estos modelos utilizando Mongoose ODM se detallará en el Capítulo 4.

3.2.4 Diseño de la Integración con el Sistema MOOC Existente

Un requisito fundamental del proyecto es que el nuevo módulo de análisis de noticias se integre de manera fluida y coherente dentro de la plataforma de Entorno Virtual de Aprendizaje (EVA) DeStance, específicamente en su sistema MOOC. Este diseño de integración busca preservar la experiencia de usuario existente, al tiempo que introduce la nueva funcionalidad como una extensión natural de los cursos ofrecidos. Se conceptualizó el nuevo módulo como "Course 4", siguiendo la numeración de los cursos preexistentes.

3.2.4.1 Integración en el Dashboard Principal del MOOC

El dashboard principal del MOOC en DeStance es el punto central donde los estudiantes acceden a los diferentes cursos y visualizan su progreso. El diseño para la integración del nuevo módulo en este dashboard contempló varios aspectos:

- **Ampliación del Selector de Cursos:** Se previó la modificación del componente selector de cursos existente para incluir una nueva opción correspondiente al "Course 4". Esto permitiría a los usuarios navegar y seleccionar el módulo de análisis de noticias de la misma manera que acceden a los otros cursos.

- **Componente de Dashboard Específico para el Curso 4:** Siguiendo el patrón arquitectónico de la plataforma DeStance, se diseñó la necesidad de un nuevo componente Svelte, denominado conceptualmente `Course4Dashboard.svelte`. Este componente sería el responsable de renderizar la vista específica del dashboard para el Curso 4, mostrando estadísticas particulares de este módulo, tales como el número de snippets analizados, la precisión promedio en la identificación de marcadores y en la evaluación de fuentes, el tiempo total invertido, y visualizaciones de datos relevantes para el desarrollo de habilidades de pensamiento crítico.

- **Consistencia con el Sistema de Pestañas:** El diseño aseguró que el `Course4Dashboard` se integrara armoniosamente con el sistema de pestañas o navegación similar que pudiera existir en el dashboard general del MOOC, permitiendo al usuario cambiar entre las vistas de los diferentes cursos sin una disrupción en la experiencia.

3.2.4.2 Estructura de Rutas y Carga de Datos

Para la navegación, se mantendría la jerarquía de rutas existente en SvelteKit, definiendo una nueva ruta principal para el módulo, por ejemplo `/mooc/course4`. El acceso al dashboard específico del curso seguiría el patrón de la plataforma, como `/dashboards/mooc` con la lógica para mostrar el contenido del Curso 4 cuando este sea seleccionado.

En cuanto a la carga de datos para la página del Curso 4 (el `+page.svelte` correspondiente a la ruta `/mooc/course4`), se diseñó un enfoque que considera tanto la eficiencia como la experiencia de usuario. La información básica del usuario y los datos de autenticación serían gestionados a través de la función `load` en el archivo `+page.server.ts` de SvelteKit, asegurando que los datos críticos estén disponibles desde el servidor. Sin embargo, para la obtención del *snippet* de noticia específico a analizar y otros datos altamente dinámicos o que dependen de interacciones recientes del usuario (como si hay nuevos snippets disponibles que no haya completado), el diseño contempla que esta lógica resida principalmente en el lado del cliente. Esto permitiría una carga inicial de la página más rápida y una mayor flexibilidad para la lógica de selección de contenido, que podría invocar servicios API según sea necesario una vez la página esté interactiva en el navegador.

3.2.4.3 Coherencia de la Experiencia de Usuario

Mantener una experiencia de usuario consistente con el resto de la plataforma DeStance fue un pilar del diseño de integración. Esto abarca:

- **Autenticación:** El nuevo módulo reutilizaría el sistema de autenticación y gestión de sesiones existente en DeStance, asegurando que solo los usuarios autenticados puedan acceder y que sus progresos se asocien correctamente a sus perfiles.

- **Estilos Visuales y Componentes UI:** Se seguirían las guías de estilo, la paleta de colores y los componentes de UI ya establecidos en DeStance, para que el Curso 4 se sienta como una parte integral de la plataforma y no como un añadido disonante.

- **Navegación y Flujo:** Los patrones de navegación generales, la forma de progresar dentro del curso y la presentación de la información seguirían las convenciones de los cursos 1-3, adaptándolos donde fuera necesario para la naturaleza más compleja e interactiva del análisis de noticias. A pesar de la mayor complejidad del flujo de cuatro pasos del módulo "Stance & Source Check" en comparación con los formularios más simples de otros cursos, se buscaría una presentación clara y guiada.

3.2.4.4 Adaptación de Datos para Visualizaciones Compartidas

El dashboard del MOOC podría incluir componentes de visualización de datos genéricos, como gráficos radar que muestran el progreso en diferentes

competencias o áreas. Dado que las métricas generadas por el módulo "Stance & Source Check" tales como `selectionsAccuracy`, `sourceAccuracy`, `overallScore`, y `timeSpent`, son específicas y difieren de las de otros cursos, se diseñó la necesidad de una capa de adaptación.

Se conceptualizó un servicio o conjunto de funciones, que en la implementación podría materializarse como `course4Service.ts`. La responsabilidad principal de este "servicio adaptador" sería transformar los datos y métricas específicas del Curso 4 al formato y estructura que esperan los componentes de visualización compartidos del dashboard MOOC. Por ejemplo, podría calcular una "eficiencia de tiempo" o una "tasa de completitud" y mapearlas a las categorías esperadas por un gráfico radar común a todos los cursos.

3.2.4.5 Consideraciones de Mantenibilidad y Extensibilidad

Finalmente, el diseño de la integración se realizó con la vista puesta en la mantenibilidad y la posibilidad de futuras extensiones. Se buscó que el código del módulo del Curso 4 estuviera razonablemente autocontenido, con puntos de integración claramente definidos con el sistema MOOC. Esto facilitaría futuras modificaciones o evoluciones del módulo sin impactar desproporcionadamente el resto de la plataforma. El patrón de integración, con un componente de dashboard específico y un servicio adaptador, se diseñó para ser potencialmente extensible, de manera que si en el futuro se añadieran otros cursos avanzados con características y datos distintivos, podrían seguir un enfoque similar para su integración.

Este enfoque de diseño para la integración busca un equilibrio entre ofrecer una funcionalidad nueva y especializada y mantener la cohesión y la usabilidad de la plataforma DeStance en su conjunto. La implementación de estos conceptos de diseño se detallará en el Capítulo 4, mostrando cómo se utilizaron las capacidades de SvelteKit y la arquitectura de componentes para lograr esta integración.

3.3 Diseño de la Interfaz y Experiencia de Usuario (UI/UX)

El diseño UI/UX fue un pilar fundamental en la concepción del módulo, buscando crear un entorno de aprendizaje que no solo fuera funcionalmente robusto, sino también intuitivo, atractivo y pedagógicamente efectivo. A diferencia de las interacciones más simples de los cursos previos en la plataforma DeStance, el análisis interactivo de texto y la evaluación de fuentes requerían una arquitectura de interfaz cuidadosamente planificada. El diseño se centró en guiar al usuario a través de un proceso estructurado, minimizando la carga cognitiva y maximizando las oportunidades de aprendizaje reflexivo.

3.3.1 Diseño del Flujo de Usuario Interactivo y Gestión de Estados del Frontend

Se diseñó un flujo de usuario interactivo que guía al estudiante a través de una secuencia lógica de tareas, cada una enfocada en un aspecto particular del análisis crítico de la información. Este flujo se complementó con un diseño

conceptual para la gestión del estado en el frontend, necesario para manejar la información dinámica generada durante la interacción del usuario.

3.3.1.1 Arquitectura de Flujo de Cuatro Pasos Secuenciales

El núcleo de la experiencia de usuario se diseñó en torno a un proceso de análisis estructurado en cuatro pasos secuenciales y obligatorios. Este diseño busca descomponer la compleja tarea del análisis crítico en fases manejables, permitiendo al usuario concentrarse en un conjunto limitado de objetivos en cada momento y construir su comprensión de manera progresiva. Los cuatro pasos conceptuales son:

1. **Paso 1: Captura de la Evaluación Inicial de Fiabilidad.**
    - **Objetivo de Diseño:** Registrar la evaluación instintiva inicial del usuario sobre la fiabilidad del snippet de noticia antes de que comience un análisis más racional y detallado. Este diseño busca fomentar la autoconciencia sobre posibles juicios previos y establecer una línea base para la reflexión posterior sobre cómo el análisis detallado puede modificar o confirmar esta percepción inicial.
    - **Interacción Diseñada:** Se conceptualizó una interfaz donde el snippet se presenta inicialmente en un modo de solo lectura. Al usuario se le ofrecerían opciones claras y sencillas para indicar su evaluación preliminar de la fiabilidad del snippet. Se diseñó que la continuación al siguiente paso estaría condicionada a la completitud de esta fase.

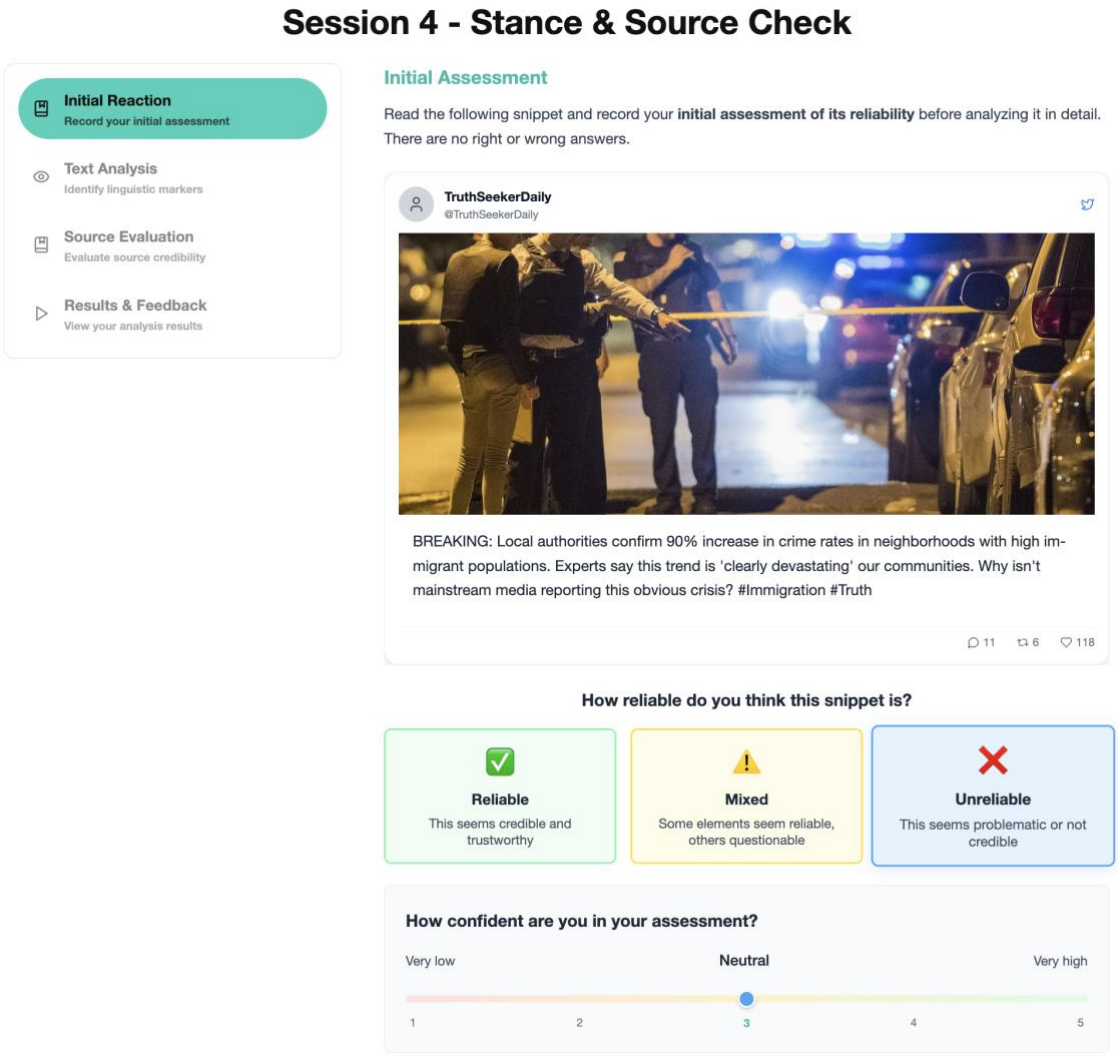


***Figura 3.2**: Ilustra la interfaz de este primer paso, mostrando el snippet en modo lectura y las opciones para registrar la evaluación inicial.*

2. **Paso 2: Análisis Interactivo del Texto y Marcado.**
    - **Objetivo de Diseño:** Permitir al usuario interactuar directamente con el texto del snippet para identificar y clasificar fragmentos que considere relevantes según una tipología de marcadores lingüísticos predefinida. Esta es la fase de highlighting o resaltado.
    - **Interacción Diseñada:** Se diseñó que, al entrar en este paso, el texto del snippet se volvería interactivo. El usuario podría seleccionar porciones de texto. Tras cada selección, aparecería un mecanismo contextual (como un pequeño panel o popup) que le permitiría clasificar la selección con uno de los siete tipos de marcadores (certeza alta/baja, emoción negativa/positiva, lenguaje cargado, apelación a la autoridad, afirmación estadística). Los fragmentos seleccionados y clasificados se resaltarían visualmente en el texto, utilizando preferiblemente diferentes estilos o colores para cada tipo de marcador, facilitando una visión general de su análisis. Se previó un resumen dinámico de las selecciones realizadas. La progresión al siguiente paso

requeriría que el usuario haya realizado al menos una selección y clasificación.

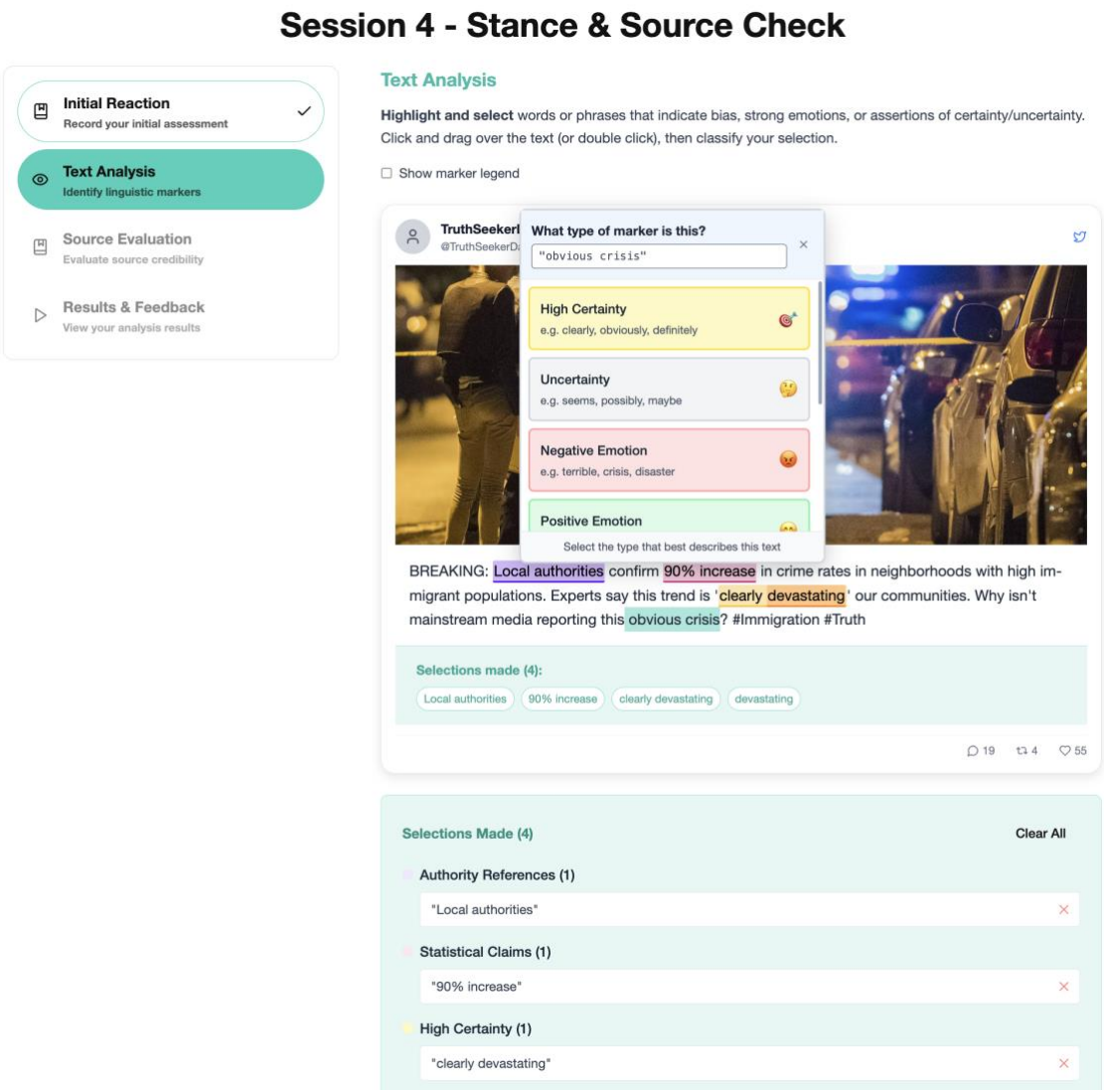


***Figura 3.3**: Ilustra la interfaz del segundo paso, con el texto del snippet activado para selección, el popup contextual para la clasificación de marcadores y los resaltados aplicados por el usuario.*

3. **Paso 3: Evaluación Estructurada de la Fuente.**
    - **Objetivo de Diseño:** Guiar al usuario en la aplicación de un conjunto de criterios sistemáticos para evaluar la credibilidad y fiabilidad de la fuente del snippet de noticia.
    - **Interacción Diseñada:** Se conceptualizó un checklist o cuestionario estructurado con cinco criterios de evaluación específicos (presencia de autor, fecha, reputación de la fuente, evidencia aportada, múltiples perspectivas). Cada criterio se presentaría como una pregunta clara acompañada de explicaciones o ayudas contextuales. El diseño también incluyó la idea de un cálculo automático de una puntuación de fiabilidad basada en las respuestas del usuario para ayudar al usuario, así

como la solicitud de un veredicto final sobre la fiabilidad de la fuente y una indicación de su nivel de confianza en esa evaluación. Todos los elementos de este paso serían obligatorios para continuar.

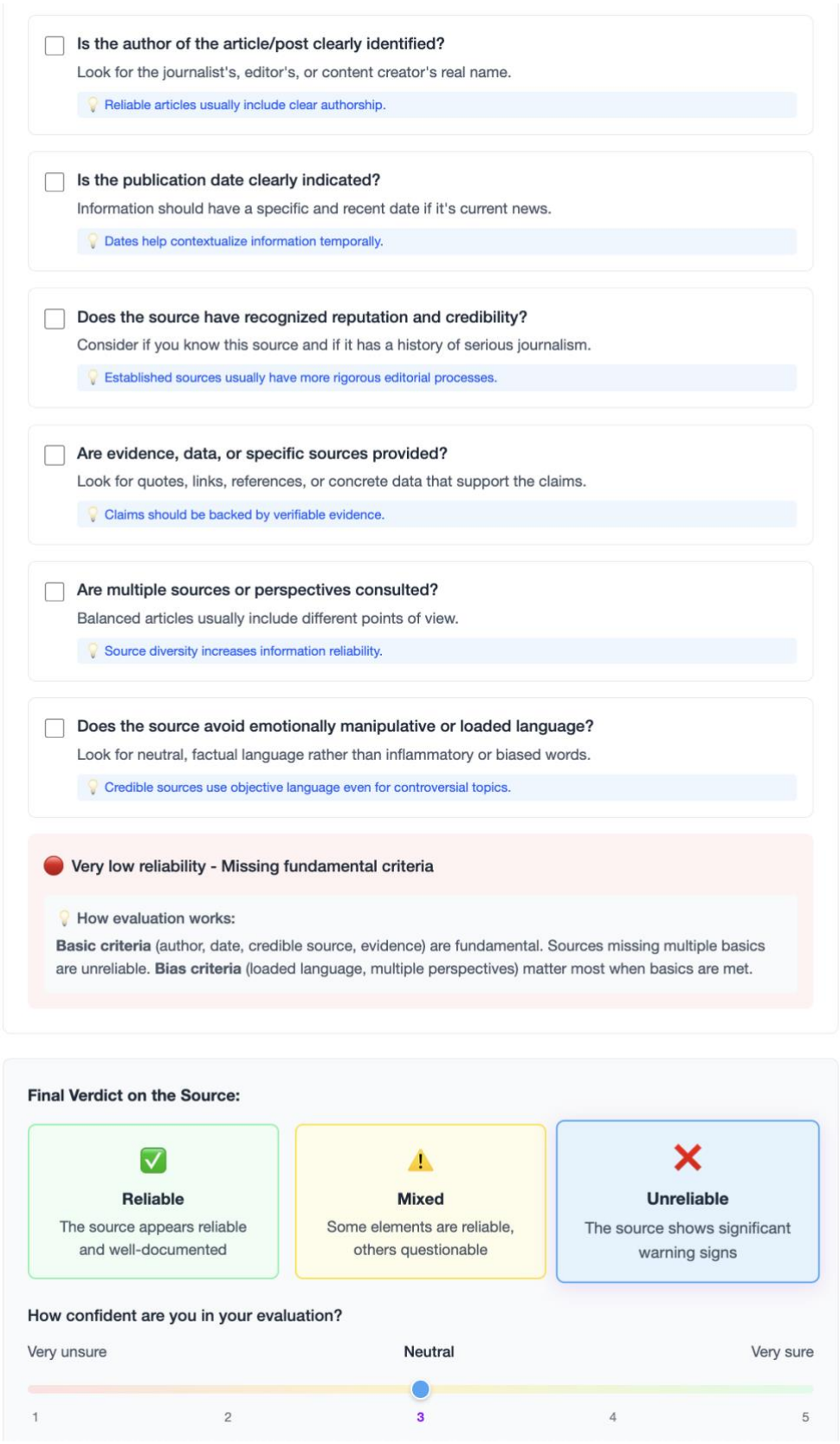


***Figura 3.4**: Ilustra la interfaz del tercer paso, donde se presenta el checklist para la evaluación de la fuente, junto con los campos para el veredicto final y el nivel de confianza.*

4. **Paso 4: Presentación del Feedback y Reflexión.**
    - **Objetivo de Diseño:** Proporcionar al usuario una retroalimentación detallada y constructiva sobre su análisis, comparando sus respuestas con las de los expertos y ofreciendo explicaciones que fomenten el aprendizaje y la mejora de sus habilidades.
    - **Interacción Diseñada:** Se diseñó una pantalla final que presentaría un resumen del desempeño del usuario. Esto incluiría métricas cuantitativas, como la precisión en la identificación de marcadores y en la evaluación de la fuente, y una puntuación global, una comparación visual o textual de sus selecciones con los marcadores identificados por los expertos, la explicación experta completa del snippet, y comentarios o recomendaciones personalizadas basadas en su rendimiento. Se contempló también la opción de que el usuario pudiera iniciar el análisis de un nuevo snippet desde esta pantalla.

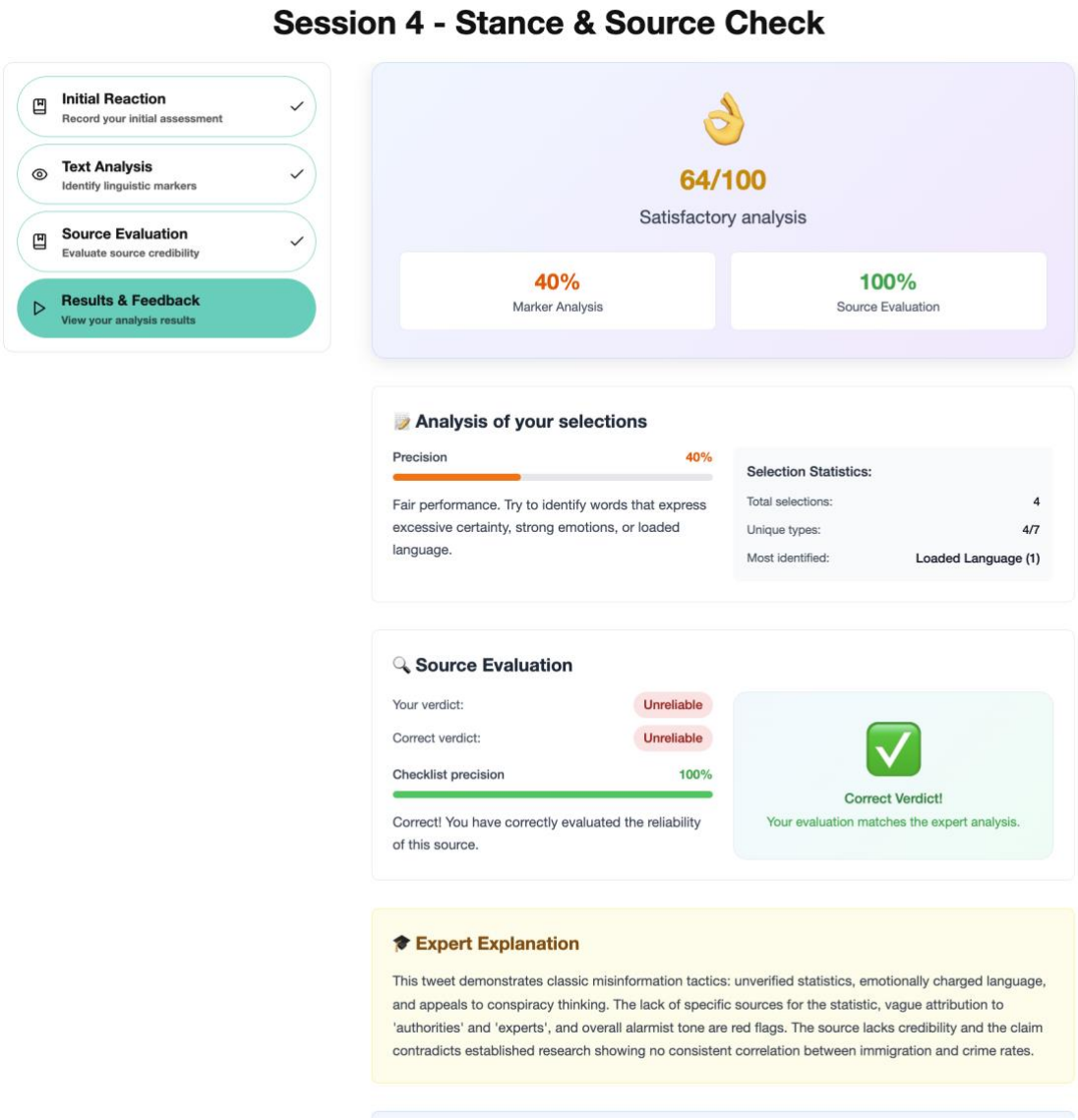


***Figura 3.5**: Ilustra la pantalla de feedback, donde se visualizan las métricas de desempeño, la comparación con el análisis experto y las recomendaciones personalizadas.*

Este flujo secuencial y validado busca asegurar que el usuario complete cada etapa del proceso analítico, promoviendo un enfoque metódico y reflexivo.

3.3.1.2 Diseño de la Gestión de Estados del Frontend

Para soportar la interactividad y el flujo dinámico descrito, se reconoció la necesidad de una estrategia clara para la gestión del estado en el frontend. El diseño contempló:

- **Estado Global Compartido:** Para información que necesita ser accesible o modificada por múltiples componentes y que persiste a través de diferentes pasos del análisis, como las selecciones de texto realizadas por el usuario (`userSelections`) o el estado de la interfaz de resaltado (si está activa, qué texto está seleccionado actualmente, la visibilidad del popup

de clasificación), se diseñó la necesidad de utilizar un sistema de gestión de estado global. En el contexto de Svelte, esto apuntaba al uso de stores reactivos. Se previó un store principal dedicado a la funcionalidad de resaltado.

- **Estado Local de Componentes:** Para datos que son específicos de un componente o de un paso particular y no necesitan ser compartidos ampliamente, como las respuestas temporales a un formulario dentro de un paso antes de ser consolidadas, el diseño permitiría el uso de variables reactivas locales dentro de los propios componentes Svelte.

- **Gestión del Flujo entre Pasos:** Se diseñó un mecanismo centralizado, probablemente en un componente orquestador principal, para controlar la transición entre los cuatro pasos del análisis. Este mecanismo se basaría en una variable de estado que indicaría el paso activo (`currentStep`) y en lógica de validación para permitir o denegar el avance.

Esta aproximación híbrida a la gestión del estado, combinando stores globales para el estado transversal y estado local para la lógica de componente, se consideró la más adecuada para equilibrar la compartición de datos con el encapsulamiento y la mantenibilidad.

3.3.2 Diseño de los Componentes Visuales (Frontend)

La interfaz de usuario se conceptualizó como un conjunto de componentes Svelte modulares e interconectados, cada uno con una responsabilidad específica dentro de la experiencia de usuario global.

3.3.2.1 Arquitectura de Componentes Jerárquica

Se diseñó una arquitectura de componentes jerárquica, con un componente principal actuando como orquestador del flujo de análisis y conteniendo los componentes específicos de cada paso. Esta estructura jerárquica facilita la organización del código, el paso de datos y la gestión de eventos. Los principales componentes conceptualizados fueron:

- **Componente Orquestador Principal (`FakeNewsAnalyzer`):** Este componente de alto nivel sería el responsable de:
    - Gestionar el estado general de la sesión de análisis (por ejemplo, el snippet actual, el paso actual del flujo).
    - Coordinar la transición entre los cuatro pasos del análisis.
    - Comunicarse con los servicios del frontend para interactuar con la API del backend (solicitar snippets, enviar respuestas).
    - Renderizar condicionalmente el componente de paso activo.

- **Componentes de Paso Específicos:** Para cada uno de los cuatro pasos del flujo de usuario, se diseñó un componente Svelte dedicado:
    - **`InitialAssessmentStep`**: Encargado de la interfaz para la captura de la evaluación inicial de fiabilidad del snippet.
    - **`TextAnalysisStep`**: Responsable de la interfaz para el análisis interactivo del texto y la clasificación de marcadores. Este componente, a su vez, contendría o coordinaría otros subcomponentes como el visualizador del snippet y el popup de selección.

- `SourceChecklistStep`: Encargado de presentar el checklist para la evaluación de la fuente.
- `FeedbackDisplay`: Responsable de mostrar los resultados y el feedback al usuario.

- **Componentes de Soporte Especializados:** Además de los componentes de paso, se identificó la necesidad de componentes más pequeños y reutilizables para funcionalidades específicas:
  - `SnippetDisplay`: Un componente dedicado a renderizar el contenido del snippet de noticia (texto e imagen). Este componente tendría que ser capaz de operar en un modo interactivo (para el resaltado) y en un modo de solo lectura.
  - `SelectionPopup`: El panel contextual que aparece al seleccionar texto, ofreciendo las opciones de clasificación de marcadores.
  - Componentes de UI genéricos definidas ya en el código base de DeStance.

3.3.2.2 Responsabilidades Clave de los Componentes Diseñados

- `FakeNewsAnalyzer` **(Orquestador):**
  - **Diseño:** Gestionar el `currentStep` del análisis. Cargar el `currentSnippet` a través de un servicio. Acumular las respuestas de cada paso (`initialAssessment`, `userSelections` del `highlightingStore`, `sourceAnalysis`). Orquestar el envío del análisis completo al backend y la recepción del `feedback`. Manejar estados `de` carga y errores generales.

- `InitialAssessmentStep`**:**
  - **Diseño:** Recibir el `currentSnippet` como propiedad. Presentar el texto del snippet en modo no interactivo (utilizando `SnippetDisplay` en modo solo lectura). Ofrecer controles para que el usuario seleccione su evaluación inicial de fiabilidad (confiable, mixto, no confiable). Validar que se ha proporcionado una respuesta. Emitir un evento con los datos de la evaluación inicial cuando el usuario continúe.

- `TextAnalysisStep`**:**
  - **Diseño:** Recibir el currentSnippet. Utilizar SnippetDisplay en modo interactivo. Integrarse con el highlightingStore y el highlightingService (ver sección 3.3.4) para permitir la selección y clasificación de texto. Mostrar el SelectionPopup cuando sea necesario. Presentar un resumen de las selecciones realizadas. Validar que se haya realizado un mínimo de selecciones. Emitir un evento para continuar al siguiente paso.

- `SourceChecklistStep`**:**
  - **Diseño:** Presentar los cinco criterios de evaluación de fuentes con sus descripciones. Ofrecer controles en forma de checkboxes para cada criterio. Permitir al usuario seleccionar un veredicto global sobre la fiabilidad de la fuente y su nivel de confianza. Validar que todas las preguntas hayan sido respondidas. Emitir un evento con los datos del análisis de la fuente.

- `FeedbackDisplay`**:**

    - **Diseño:** Recibir los datos del feedback (métricas, explicación experta, etc.) como propiedad. Presentar esta información de manera clara y estructurada. Ofrecer una opción para iniciar un nuevo análisis.

- **`SnippetDisplay`:**
    - **Diseño:** Recibir el objeto **`snippet`** y un indicador de **`isInteractive`**. Renderizar el texto y la imagen. Si es interactivo, colaborar con el **`highlightingService`** para habilitar la selección y mostrar los resaltados aplicados por el usuario (que provendrían del **`highlightingStore`**). Si no es interactivo, solo mostrar el texto y, potencialmente, resaltados preexistentes si se estuviera revisando un análisis.

- **`SelectionPopup`:**
    - **Diseño:** Recibir su visibilidad y posición como propiedades. Mostrar los botones de los siete tipos de marcadores. Al hacer clic en un marcador, emitir un evento con el **`MarkerType`** seleccionado para que **`TextAnalysisStep`** o el **`highlightingService`** puedan procesarlo.

Este diseño de componentes busca una separación clara de responsabilidades y una alta cohesión dentro de cada componente, facilitando el desarrollo, las pruebas y la mantenibilidad del frontend. La implementación específica de estos componentes se detallará en el Capítulo 4.

3.3.3 Consideraciones sobre Pruebas de Usabilidad

Si bien el diseño de la interfaz y la experiencia de usuario se ha guiado por principios de usabilidad heurísticos y un flujo de interacción estructurado pedagógicamente, es importante mencionar que no se llevaron a cabo pruebas de usabilidad formales con usuarios finales. La validación del diseño UI/UX se realizó de manera iterativa por el desarrollador y mediante el feedback de las tutoras del proyecto.

La realización de pruebas de usabilidad exhaustivas con el público objetivo (estudiantes universitarios) se considera un paso crucial y se plantea como una línea de trabajo futuro fundamental. Estas pruebas permitirían recoger datos empíricos sobre la facilidad de uso, la eficiencia en la compleción de tareas, la satisfacción del usuario y la efectividad pedagógica de la interfaz, llevando a refinamientos basados en evidencia. Este aspecto se retomará en el capítulo de limitaciones y líneas futuras.

3.4 Diseño del Contenido: Creación y Curación de Snippets

La efectividad pedagógica del módulo depende intrínsecamente de la calidad, diversidad y relevancia del contenido utilizado. Por ello, se dedicó una fase de diseño a la conceptualización de los snippets de noticias y los marcadores expertos asociados. A diferencia de contenido generado automáticamente o

plantillas simples, este módulo requiere un proceso de curación manual y experto.

3.4.1 Tipología de Contenido Especializada

Para asegurar una experiencia de aprendizaje rica y variada, se diseñó una tipología de contenido que clasifica los snippets según varios ejes:

3.4.1.1 Clasificación por Veracidad y Sesgo

Se definió una taxonomía de cuatro categorías principales para reflejar el espectro de información que los usuarios encuentran en el entorno digital:

- `fake` **(Información Completamente Falsa):** Snippets que contienen afirmaciones factualmente incorrectas, datos inventados o eventos inexistentes. El objetivo pedagógico es entrenar la detección de desinformación explícita.

- `misleading` **(Información Parcialmente Cierta pero Engañosa):** Snippets que pueden usar datos reales pero presentados fuera de contexto, estableciendo correlaciones falsas u omitiendo información crucial de manera selectiva. El objetivo es desarrollar habilidades de análisis contextual y detección de manipulación sutil.

- `real-biased` **(Información Veraz con Sesgo Evidente):** Snippets que presentan hechos correctos pero utilizando un lenguaje fuertemente emotivo, seleccionando únicamente los hechos que apoyan una perspectiva particular, u omitiendo otros puntos de vista relevantes. El objetivo es entrenar la identificación de sesgo incluso en información que es factualmente correcta.

- `real-neutral` **(Información Equilibrada y Factual):** Snippets que se esfuerzan por una presentación objetiva de los hechos, citando fuentes diversas y presentando múltiples perspectivas. Estos sirven como ejemplos de referencia de periodismo de calidad y permiten a los usuarios calibrar sus habilidades de evaluación.

3.4.1.2 Clasificación por Formato Mediático

Para reflejar la diversidad de formatos en los que se consume la información hoy en día, se diseñó el soporte para al menos dos formatos principales:

- `news-article` **(Artículo de Noticias Tradicional):** Fragmentos más extensos, simulando la estructura de un artículo periodístico, con posibilidad de título, subtítulo y varios párrafos.

- `tweet` **(Publicación de Redes Sociales):** Contenido breve, imitando el formato de plataformas como Twitter, incluyendo características como lenguaje informal, posible uso de hashtags o menciones, y una presentación visual distintiva.

3.4.1.3 Clasificación Temática Especializada

Dada la inserción del módulo en el proyecto RACISMMAFF, se priorizaron temáticas específicas, aunque también se contempló contenido general para ampliar la aplicabilidad de las habilidades:

- `immigration` **(Inmigración):** Tema central del proyecto de investigación, abarcando aspectos como políticas migratorias, integración social, impacto económico y narrativas sobre refugiados.
- `racism` **(Racismo y Discriminación):** Contenido relacionado con la discriminación racial, étnica o cultural, racismo institucional, microagresiones y movimientos sociales.
- `general` **(Temas Diversos):** Snippets sobre otros temas de actualidad (política, economía, salud, tecnología, medio ambiente) para permitir la práctica y transferencia de las habilidades de análisis crítico a diferentes contextos.

3.4.2 Diseño de la Estructura de Marcadores Expertos

La "verdad fundamental" contra la cual se evalúa el análisis del usuario reside en los marcadores identificados por expertos para cada snippet. El diseño de esta estructura fue crucial.

3.4.2.1 Sistema de Marcación Lingüística Especializada

La "verdad fundamental" contra la cual se evalúa el análisis del usuario reside en los marcadores identificados por expertos para cada snippet. El diseño de esta estructura fue crucial, seleccionando un sistema de siete categorías de marcadores lingüísticos basados en conceptos de análisis del discurso y la teoría del posicionamiento (stance), como se introdujo en la sección 2.1.3 [1][14][15][16].

La elección de estas siete categorías no es arbitraria, sino que responde a la necesidad de dotar a los estudiantes de un conjunto de herramientas analíticas específicas y manejables para diseccionar los mecanismos persuasivos y de posicionamiento en textos, especialmente aquellos susceptibles de contener desinformación. El objetivo pedagógico es ir más allá de una simple lectura superficial, permitiendo al estudiante identificar patrones discursivos que construyen la credibilidad (o la falta de ella), el sesgo y la intención del autor. Cada categoría de marcador se seleccionó por su recurrencia en discursos informativos y persuasivos y por su utilidad directa en el fomento del pensamiento crítico:

- **Marcadores Epistémicos (Posicionamiento sobre el conocimiento):**
  - `epistemic-high`: Expresiones de alta certeza (ej. "claramente", "sin duda").
  - `epistemic-low`: Expresiones de incertidumbre o baja certeza (ej. "parece", "posiblemente").
  - **Justificación:** Identificar el grado de certeza con el que se presentan las afirmaciones es fundamental. La desinformación a menudo se presenta con una falsa autoridad (epistemic-high sin sustento) o, por el contrario, utiliza la ambigüedad (epistemic-low)

para introducir dudas infundadas o evitar la responsabilidad directa [15]. Enseñar a los estudiantes a cuestionar la base de esta certeza o incertidumbre es clave.

- **Marcadores Emocionales (Carga afectiva):**
  - `emotional-negative`: Palabras o frases con connotación fuertemente negativa (ej. "terrible", "crisis").
  - `emotional-positive`: Palabras o frases con connotación fuertemente positiva (ej. "excelente", "maravilloso").
  - **Justificación**: El lenguaje emotivo es una poderosa herramienta de persuasión. La desinformación frecuentemente apela a las emociones (miedo, indignación, esperanza) para eludir el escrutinio racional y movilizar opiniones [15]. Reconocer cuándo y cómo se utiliza la carga emocional ayuda a los estudiantes a separar el contenido fáctico de la manipulación afectiva.
- **Otros Marcadores Relevantes para el Análisis Crítico:**
  - `bias-loaded`: Lenguaje cargado, no neutral, que busca influir en la percepción (ej. "invasión" vs "llegada masiva").
    - **Justificación**: Más allá de la emoción explícita, ciertas elecciones léxicas conllevan sesgos inherentes que enmarcan la información de una manera particular [16]. Identificar este lenguaje cargado es vital para entender la perspectiva del autor y los posibles intentos de manipulación sutil.
  - `authority-claim`: Apelaciones a la autoridad, a menudo vagas o no verificables (ej. "expertos dicen").
    - **Justificación**: Los argumentos de autoridad pueden ser válidos, pero en la desinformación suelen ser falaces, citando autoridades inexistentes, no expertas en el tema, o de forma anónima para evitar la verificación [16]. Capacitar a los estudiantes para que evalúen la legitimidad de estas apelaciones es crucial.
  - `statistical-claim`: Uso de datos o afirmaciones estadísticas, que pueden ser precisas, imprecisas o sin fuente.
    - **Justificación**: Las cifras y estadísticas a menudo se perciben como objetivas y convincentes. Sin embargo, pueden ser inventadas, sacadas de contexto o presentadas sin la fuente original, lo que impide su verificación [17]. Este marcador ayuda a los estudiantes a desarrollar un escepticismo saludable hacia las afirmaciones numéricas y a buscar su origen y metodología.

Para cada marcador experto en un snippet, el diseño contempla almacenar el texto exacto, el tipo de marcador, opcionalmente su posición precisa (índices de inicio y fin), y una breve explicación pedagógica.

3.4.2.2 Proceso de Curación Experta

Se diseñó un flujo de trabajo conceptual para la creación y validación de los snippets y sus marcadores expertos:

1. **Creación/Selección del Snippet Base:** Un experto selecciona un fragmento de una noticia real o redacta un snippet ficticio que cumpla

los objetivos pedagógicos (representando un tipo de veracidad/sesgo, formato y tema).
2. **Marcación Experta:** El experto identifica y anota todos los marcadores lingüísticos relevantes en el snippet según la tipología definida, junto con la información de la fuente y la explicación global.
3. **Revisión y Validación:** Idealmente, un segundo experto revisaría la marcación para asegurar la consistencia y precisión.
4. **Prueba Piloto (Opcional):** Se podría probar el snippet con un pequeño grupo de usuarios para calibrar su dificultad y claridad.
5. **Activación:** Una vez validado, el snippet se marcaría como activo para su uso en el módulo.

Este proceso de curación busca asegurar que el contenido sea de alta calidad, pedagógicamente sólido y relevante para los objetivos de aprendizaje.

3.4.2.3 Estructura Pedagógica del Sistema de Evaluación de Fuentes

Para asegurar que el módulo enseñe efectivamente la diferenciación entre tipos de contenido, se diseñó una correlación específica entre las categorías de snippets y los resultados esperados en el checklist de evaluación de fuentes. Esta estructura pedagógica permite a los usuarios aprender a identificar no solo marcadores lingüísticos, sino también patrones de credibilidad en las fuentes.

***Tabla 3.1**: Estructura Pedagógica por Tipo de Snippet*

Tipo de Snippet	Checklist Score Objetivo	Fallas Características	Objetivo Pedagógico	reliabilityLevel
real-neutral	6/6	Ninguna	Modelo de periodismo de calidad	**reliable**
real-biased	5/6	avoidsLoadedLanguage	Detectar sesgo en fuentes técnicamente correctas	**mixed**
misleading	3-4/6	hasMultipleSources, avoidsLoadedLanguage	Identificar omisiones y contexto incompleto	**unreliable**
fake	0-2/6	Múltiples criterios básicos	Reconocer desinformación evidente	**unreliable**

El campo `reliabilityLevel` representa la evaluación global del artículo específico con tres categorías:

- **reliable**: Contenido neutral y factualmente correcto (`real-neutral`)
- **mixed**: Contenido factualmente correcto pero con sesgo evidente (`real-biased`)
- **unreliable**: Contenido falso o engañoso (`fake`, `misleading`)

Esta estructura permite al usuario aprender a distinguir entre diferentes grados de confiabilidad, reconociendo que el sesgo no invalida automáticamente la información factual.

3.4.3 Diseño de la Integración con AdminJS para la Gestión de Contenido

Para facilitar la creación, edición y gestión de la base de datos de snippets y para la supervisión de las respuestas de los usuarios, se diseñó la integración con AdminJS, una herramienta de panel de administración para aplicaciones Node.js.

3.4.3.1 Panel de Administración Especializado

Se conceptualizó la configuración de AdminJS para ofrecer vistas y formularios adaptados a los modelos `FakeNewsSnippet` y `UserFakeNewsResponse`. Esto incluiría:

- Para `FakeNewsSnippet`: Campos de formulario para el texto, URL de imagen, selección de tipo, formato y tema. Una interfaz amigable para definir los `expertMarkers` (posiblemente un editor JSON o campos repetibles para cada marcador) y la `sourceInfo` (incluyendo el checklist experto). Un campo para la `expertExplanation` y un conmutador para el estado `isActive`.

- Para `UserFakeNewsResponse`: Vistas de solo lectura para examinar las respuestas enviadas por los usuarios, mostrando sus selecciones, su evaluación de la fuente y las métricas de rendimiento calculadas.

3.4.3.2 Flujo de Trabajo de Curación a través de AdminJS

El panel de AdminJS sería la herramienta principal para que los educadores y administradores del contenido gestionen el ciclo de vida de los snippets: desde su creación inicial y marcación experta, pasando por su revisión y activación, hasta su posible desactivación o modificación.

Este diseño integral del contenido y su gestión busca asegurar que el módulo "Stance & Source Check" disponga de un corpus de materiales de aprendizaje de alta calidad, diverso y fácilmente administrable, lo cual es esencial para su éxito a largo plazo.

4. Implementación del Sistema

Este capítulo detalla el proceso de construcción y desarrollo técnico del módulo "Stance & Source Check", traduciendo los conceptos y especificaciones de diseño, expuestos en el Capítulo 3, en una solución software funcional e integrada en la plataforma DeStance. Se describirán las herramientas específicas utilizadas, la configuración del entorno de desarrollo, y se profundizará en la implementación de los componentes clave del backend y del frontend. Se pondrá especial énfasis en la lógica de resaltado interactivo con Rangy.js, que constituye una de las principales contribuciones técnicas de este trabajo, así como en los desafíos encontrados y las soluciones aplicadas durante la fase de codificación.

4.1 Configuración del Entorno y Dependencias

La base para cualquier desarrollo de software robusto reside en una correcta configuración del entorno de trabajo y una gestión eficiente de las dependencias. Para el Course 4 se partió de la infraestructura tecnológica existente en la plataforma DeStance, asegurando la compatibilidad y la coherencia con el ecosistema del proyecto RACISMMAFF.

El entorno de desarrollo se estableció sobre el sistema operativo de la máquina donde fue realizada el trabajo, macOS, utilizando Visual Studio Code como IDE principal, gracias a su amplio soporte para JavaScript, TypeScript y Svelte, así como sus capacidades de depuración y extensiones para mejorar la productividad. El control de versiones se gestionó mediante Git, utilizando un repositorio privado para el proyecto DeStance, lo que facilitó el trabajo colaborativo, el seguimiento de cambios y la gestión de diferentes ramas de desarrollo (para este módulo, `mooc/fake-news`).

La gestión de las dependencias del proyecto, tanto para el backend como para el frontend, se realizó a través de `npm`, el gestor de paquetes por defecto de Node.js. Las dependencias se separan por el front y el backend, las dependencias clave que sustentan la funcionalidad del módulo son:

Dependencias Relevantes del Backend:

- **Node.js:** Entorno de ejecución de JavaScript del lado del servidor, base sobre la cual opera todo el backend. Se especificó el uso de Node.js versión 22, gestionable mediante `nvm` (Node Version Manager) para asegurar la consistencia entre entornos.

- **Express.js:** Framework web minimalista y flexible para Node.js, utilizado para construir la API RESTful que comunica el frontend con el backend y la base de datos.

- **Mongoose:** Librería ODM para MongoDB y Node.js, esencial para definir esquemas, validar datos, modelar relaciones y realizar consultas a la base de datos de forma estructurada.

- **MongoDB:** Sistema de base de datos NoSQL orientado a documentos, donde se almacena toda la información persistente del módulo, desde los snippets de noticias hasta las respuestas de los usuarios. Su instalación

y configuración incluyeron la importación de la clave pública de MongoDB, la adición del repositorio, la instalación del paquete **`mongodb-org`**, y la posterior creación de una base de datos y un usuario administrador con autenticación habilitada para asegurar el acceso.

- **`jsonwebtoken` y `bcrypt`:** Utilizados para la gestión de la autenticación mediante tokens JWT y el hashing seguro de contraseñas, respectivamente, aprovechando la infraestructura de autenticación existente en DeStance.

- **`cors`:** Middleware para habilitar el Cross-Origin Resource Sharing, permitiendo que el frontend (posiblemente en un dominio o puerto diferente durante el desarrollo) pueda realizar peticiones a la API del backend.

- **`dotenv`:** Módulo para cargar variables de entorno desde un archivo **`.env`** durante el desarrollo, facilitando la configuración de parámetros sensibles como las cadenas de conexión a la base de datos.

- **AdminJS y sus conectores (`@adminjs/express`, `@adminjs/mongoose`):** Para la creación y configuración del panel de administración que permite la gestión del contenido del módulo, como los *snippets* y las respuestas de los usuarios.

Dependencias Relevantes del Frontend:

- **SvelteKit:** Framework para la construcción de aplicaciones web robustas y de alto rendimiento, basado en Svelte. Proporciona enrutamiento, renderizado del lado del servidor (SSR) y otras funcionalidades esenciales para el desarrollo del frontend.

- **Svelte:** Compilador que convierte los componentes **`.svelte`** en código JavaScript altamente optimizado, permitiendo construir interfaces de usuario reactivas de manera eficiente.

- **TypeScript:** Superset de JavaScript que añade tipado estático, mejorando la robustez del código, la detección temprana de errores y la mantenibilidad del proyecto frontend. Todos los nuevos componentes y servicios del frontend se desarrollaron utilizando TypeScript.

- **Rangy.js:** Librería JavaScript especializada en la manipulación avanzada de rangos y selecciones de texto en el DOM. Su inclusión fue una decisión técnica crucial para implementar la funcionalidad de highlighting interactivo de manera fiable y robusta, superando las limitaciones de las APIs nativas del navegador para selecciones complejas. Se utilizaron sus módulos **`rangy-core`**, **`rangy-classapplier`** y **`rangy-serializer`**.

- **Vite:** Herramienta de *bundling* y servidor de desarrollo extremadamente rápido que utiliza SvelteKit por debajo, mejorando significativamente la experiencia de desarrollo con HMR instantánea.

- **TailwindCSS:** Framework de CSS orientado a utilidades, utilizado en la plataforma DeStance para la estilización de componentes, permitiendo un desarrollo rápido y consistente de la interfaz de usuario.

- **uuid:** Librería para generar identificadores únicos universales (UUIDs), utilizada en el frontend para asignar IDs únicos a las selecciones de los usuarios antes de enviarlas al backend.

El proceso de puesta en marcha del entorno de desarrollo comenzaba con la clonación del repositorio del proyecto. A continuación, se procedía con la instalación de Node.js v22 (preferiblemente usando `nvm`) y MongoDB Community Edition, siguiendo las guías específicas para el sistema operativo, que incluían la configuración de repositorios y la habilitación del servicio **`mongod`**. Posteriormente, se instalaban las dependencias de npm ejecutando npm install en los directorios raíz del backend (`back/`) y del frontend (`front/`). Un paso crucial era la configuración de las variables de entorno; para ello, se copiaban los archivos `.env.example` a `.env` (en el backend) y a `.env.development` y `.env.production` (en el frontend), ajustando valores como **`MONGO_DB_URL`** con las credenciales de la base de datos recién creada, **`JWT_KEY`**, **`ADMIN_COOKIE_SECRET`** para el backend, y **`VITE_API_URL`** para el frontend. Una vez completada la instalación de dependencias y la configuración, el backend se iniciaba con **`npm run start`** tras un **`npm run admin:bundle`** para los componentes de AdminJS y el frontend SvelteKit con **`npm run dev`**, estableciendo así un ciclo de desarrollo ágil y funcional. El acceso al panel de AdminJS requería, además, la modificación manual del flag `admin` a `true` para el usuario deseado en la colección `userauths` de MongoDB.

4.2 Implementación del Backend

La implementación del backend se centró en materializar la lógica de negocio y la capa de persistencia diseñadas, utilizando Node.js, Express y Mongoose. Esto implicó el desarrollo detallado de los modelos de datos, la codificación de los controladores que orquestan las operaciones y la definición de las rutas API que exponen la funcionalidad al frontend.

4.2.1 Desarrollo de Modelos Mongoose Avanzados

Los modelos de datos **`FakeNewsSnippet`** y **`UserFakeNewsResponse`**, diseñados conceptualmente en el Capítulo 3, se implementaron como esquemas Mongoose.

El modelo `FakeNewsSnippet` (definido en **`back/models/fake-news-snippet.js`**), encargado de almacenar los fragmentos de noticias curados. Elementos cruciales de su implementación incluyen:

```
const fakeNewsSnippetSchema = mongoose.Schema(
    {
        text: { type: String, required: true },
        imageUrl: { type: String },
        publishedDate: { type: String, required: false },
        type: {
            type: String,
            enum: ["real-neutral", "real-biased", "fake", "misleading"],
            required: true,
        },
        format: {
            type: String,
            enum: ["news-article", "tweet"],
            default: "news-article",
        },
        topic: {
            type: String,
            enum: ["immigration", "racism", "general"],
            default: "general",
        },

        expertMarkers: [expertMarkerSchema],
        sourceInfo: sourceInfoSchema,
        expertExplanation: { type: String, required: true },
        isActive: { type: Boolean, default: true },
        createdBy: { type: String, default: "admin" },
        usage: {
            timesUsed: { type: Number, default: 0 },
            averageAccuracy: { type: Number, default: 0 },
            lastUsed: { type: Date },
        },
    },
    { timestamps: true }
);
```

- **`expertMarkers`**: Un array de esquemas anidados (**`expertMarkerSchema`**) donde cada elemento define un marcador lingüístico identificado por expertos. Este esquema anidado contiene campos como **`text`**, **`markerType`**, **`startIndex`**, **`endIndex`** y **`explanation`**.

```
const expertMarkerSchema = mongoose.Schema({
    text: { type: String, required: true },
    markerType: {
        type: String,
        enum: [
            "epistemic-high", // High certainty (clearly, obviously)
            "epistemic-low", // Uncertainty (seems, possibly)
            "emotional-negative", // Negative emotion (terrible, crisis)
            "emotional-positive", // Positive emotion (excellent, wonderful)
            "bias-loaded", // Loaded language (invasion, destroying)
            "authority-claim", // Authority references (experts say)
            "statistical-claim", // Statistical claims (90% of...)
        ],
        required: true,
    },
    startIndex: { type: Number, required: false },
    endIndex: { type: Number, required: false },
    explanation: { type: String },
});
```

- **sourceInfo**: Un esquema anidado (**sourceInfoSchema**) que detalla la fuente del snippet, con campos como **name**, **domain**, **reliabilityLevel** y un sub-objeto **checklistAnswers** para las respuestas expertas a los criterios **hasAuthor**, **hasDate**, **hasCredibleSource**, **hasEvidence**, **hasMultipleSources** y **avoidsLoadedLanguage.**

```
const sourceInfoSchema = mongoose.Schema({
    name: { type: String, required: true }, // "The Fake Daily"
    domain: { type: String }, // "thefakedaily.com"
    reliabilityLevel: {
        type: String,
        enum: ["reliable", "mixed", "unreliable"],
        required: true,
        default: "unreliable",
    }, // Reliability of the snippet not the source
    checklistAnswers: {
        hasAuthor: { type: Boolean, required: true, default: false },
        hasDate: { type: Boolean, required: true, default: false },
        hasCredibleSource: { type: Boolean, required: true, default: false },
        hasEvidence: { type: Boolean, required: true, default: false },
        hasMultipleSources: { type: Boolean, required: true, default: false },
        avoidsLoadedLanguage: { type: Boolean, required: true, default: false },
    },
});
```

- **`expertExplanation`**: Un campo **`String`** para la explicación detallada del experto.

El modelo `UserFakeNewsResponse` (definido en **`back/models/user-fake-news-snippet.js`**), fue diseñado para capturar la interacción completa del usuario con un snippet:

- **`userSelections`**: Un array de esquemas anidados (**`userSelectionSchema`**) que almacena cada selección de texto del usuario. Cada selección incluye **`id`** (UUID generado en el frontend), **`text`**, **`markerType`**, el campo crucial **`serializedSelection`** (que guarda la representación de Rangy.js), y **`context`** (con **`before`**, after, **`paragraphIndex`**).

```
const userSelectionSchema = mongoose.Schema({
    id: { type: String, required: true }, // UUID generated in frontend
    text: { type: String, required: true }, // Text selected by the user
    markerType: {
        type: String,
        enum: [
            "epistemic-high", // High certainty (clearly, obviously)
            "epistemic-low", // Uncertainty (seems, possibly)
            "emotional-negative", // Negative emotion (terrible, crisis)
            "emotional-positive", // Positive emotion (excellent, wonderful)
            "bias-loaded", // Loaded language (invasion, destroying)
            "authority-claim", // Authority references (experts say)
            "statistical-claim", // Statistical claims (90% of...)
        ],
        required: true,
    },
    serializedSelection: { type: String, required: true }, // Rangy serialization
    context: {
        before: { type: String }, // Text before the selection
        after: { type: String }, // Text after the selection
        paragraphIndex: { type: Number }, // Paragraph index
    },
    timestamp: { type: Date, default: Date.now },
});
```

- **initialAssessment**: Un esquema anidado (**initialAssessmentSchema**) para la reacción emocional inicial del usuario.

```
const initialAssessmentSchema = mongoose.Schema({
    initialReliabilityAssessment: {
        type: String,
        enum: ["reliable", "unreliable", "mixed"],
        required: true,
    },
    emotionalIntensity: { type: Number, min: 1, max: 5 },
    timestamp: { type: Date, default: Date.now },
});
```

- **sourceAnalysis**: Un esquema anidado (**sourceAnalysisSchema**) para las respuestas del usuario al checklist de evaluación de fuentes y su veredicto.

```
const sourceAnalysisSchema = mongoose.Schema({
    checklistAnswers: {
        hasAuthor: { type: Boolean, required: true },
        hasDate: { type: Boolean, required: true },
        hasCredibleSource: { type: Boolean, required: true },
        hasEvidence: { type: Boolean, required: true },
        hasMultipleSources: { type: Boolean, required: true },
        avoidsLoadedLanguage: { type: Boolean, required: true },
    },
    reliabilityVerdict: {
        type: String,
        enum: ["reliable", "unreliable", "mixed"],
        required: true,
    },
    confidence: { type: Number, min: 1, max: 5 },
    timestamp: { type: Date, default: Date.now },
});
```

- **`performance`**: Un objeto anidado crucial donde se almacenan las métricas calculadas tras el análisis del usuario, como **`timeSpent`**, **`selectionsAccuracy`**, **`sourceAccuracy`** y **`overallScore`**.

```
// Calculated metrics - Use nested path instead of subdocument to avoid undefined issues
performance: {
    timeSpent: { type: Number, default: 0 }, // total seconds
    selectionsAccuracy: { type: Number, min: 0, max: 100, default: 0 }, // % match with experts
    sourceAccuracy: { type: Number, min: 0, max: 100, default: 0 }, // % correct checklist answers
    overallScore: { type: Number, min: 0, max: 100, default: 0 }, // overall score
    feedbackGenerated: { type: Boolean, default: false },
},
```

Para encapsular la lógica de negocio directamente relacionada con los datos, se implementaron métodos de instancia en **`UserFakeNewsResponse`**, como **`calculateSelectionsAccuracy()`** y **`calculateSourceAccuracy()`**. Estos métodos comparan las entradas del usuario con los datos expertos del **`FakeNewsSnippet`** correspondiente.

A continuación, se muestra un fragmento ilustrativo de la implementación del método **`calculateSelectionsAccuracy()`** en **`user-fake-news-response.js`**, que evidencia la lógica de comparación:

```
// Method to calculate selections accuracy
userFakeNewsResponseSchema.methods.calculateSelectionsAccuracy = function (expertMarkers) {
    // Ensure performance object exists
    if (!this.performance) {
        this.performance = {};
    }

    if (!this.userSelections || this.userSelections.length === 0) {
        this.performance.selectionsAccuracy = 0;
        return 0;
    }

    if (!expertMarkers || expertMarkers.length === 0) {
        this.performance.selectionsAccuracy = 0;
        return 0;
    }

    let matches = 0;
    let totalExpertMarkers = expertMarkers.length;

    // Comparison logic
    this.userSelections.forEach((userSelection) => {
        const matchingExpert = expertMarkers.find(
            (expert) => {
                // Check if marker types match
                if (expert.markerType !== userSelection.markerType) {
                    return false;
                }

                // Check if the selected text overlaps with expert marker
                const userText = userSelection.text.toLowerCase().trim();
                const expertText = expert.text.toLowerCase().trim();

                // Exact match
                if (userText === expertText) {
                    return true;
                }

                // Check if user selection contains expert text or vice versa
                if (userText.includes(expertText) || expertText.includes(userText)) {
                    return true;
                }

                return false;
            }
        );

        if (matchingExpert) {
            matches++;
        }
    });

    const accuracy = totalExpertMarkers > 0 ? (matches / totalExpertMarkers) * 100 : 0;
    this.performance.selectionsAccuracy = Math.round(accuracy);
    return Math.round(accuracy);
};
```

Este método itera sobre las selecciones del usuario, comparando el **markerType** y el texto (con cierta flexibilidad para solapamientos o inclusiones) con los **expertMarkers** para calcular la precisión. De forma similar, **calculateSourceAccuracy()** compara las respuestas del **checklistAnswers** del usuario con las correctas.

Además, se implementaron métodos estáticos como **UserFakeNewsResponse.getUserResponses()** para obtener el historial de un usuario y **UserFakeNewsResponse.getSnippetStats()** para estadísticas agregadas. Finalmente, se definieron índices estratégicos en ambos esquemas Mongoose para optimizar el rendimiento de las consultas frecuentes, como **fakeNewsSnippetSchema.index({ type: 1, isActive: 1 });** y **userFakeNewsResponseSchema.index({ userId: 1, snippetId: 1 }, { unique: true });**.

```
// Static method to get user responses
userFakeNewsResponseSchema.statics.getUserResponses = function (userId, limit = 10) {
    return this.find({ userId }).populate("snippetId", "text type topic").sort({ createdAt: -1 }).limit(limit);
};

// Static method to get snippet statistics
userFakeNewsResponseSchema.statics.getSnippetStats = function (snippetId) {
    return this.aggregate([
        { $match: { snippetId: new mongoose.Types.ObjectId(snippetId), isCompleted: true } },
        {
            $group: {
                _id: null,
                totalResponses: { $sum: 1 },
                avgSelectionsAccuracy: { $avg: "$performance.selectionsAccuracy" },
                avgSourceAccuracy: { $avg: "$performance.sourceAccuracy" },
                avgOverallScore: { $avg: "$performance.overallScore" },
                avgTimeSpent: { $avg: "$performance.timeSpent" },
            },
        },
    ]);
};
```

```
// Indexes to optimize queries
userFakeNewsResponseSchema.index({ userId: 1, snippetId: 1 }, { unique: true });
userFakeNewsResponseSchema.index({ userId: 1, createdAt: -1 }); // -1 for descending order, newest first
userFakeNewsResponseSchema.index({ snippetId: 1, isCompleted: 1 });
```

4.2.2 Implementación de Controladores y Rutas API

Los controladores del backend, implementados en **back/controllers/fake-news.js**, contienen la lógica de negocio principal, mientras que las rutas, definidas en **back/routes:fake-news.js**, exponen esta lógica a través de endpoints HTTP.

Las rutas se definieron utilizando Express Router:

```
import express from "express";
import * as fakeNewsController from "../controllers/fake-news.js";
import { checkAuth } from "../middleware/check-auth.js";

const router = express.Router();

// Get next snippet for authenticated user
router.get("/snippet/:userId", checkAuth, fakeNewsController.getSnippetForUser);

// Submit complete user analysis
router.post("/response", checkAuth, fakeNewsController.submitUserResponse);

// Get user's analysis history
router.get("/history/:userId", checkAuth, fakeNewsController.getUserHistory);


export default router;
```

Todos los endpoints están protegidos por el middleware **`checkAuth`**, asegurando que solo los usuarios autenticados puedan acceder.

Las rutas definidas en detalle se muestran a continuación:

- **`GET /api/fake-news/snippet/:userId`**: Provee un snippet adecuado para el usuario.
  - **Request Params**:
    - **`userId`**: (string) ID del usuario que solicita el snippet.
  - **Response Body (Éxito - 200 OK):**

```
{
    "message": "Snippet retrieved successfully",
    "snippet": {
        "_id": "string",
        "text": "string",
        "imageUrl": "string" | null,
        "type": "string", // "real-neutral", "real-biased", "fake", "misleading"
        "format": "string", // "news-article", "tweet"
        "topic": "string", // "immigration", "racism", "general"
        "sourceInfo": {
            "name": "string",
```

- **Response Body (Error - 404 Not Found - No más snippets):**

```
{
    "message": "No more snippets available for this user",
    "hasMore": false
```

- **Response Body (Error - 500 Internal Server Error)**:

```
{
    "message": "Internal server error",
    "error": "string" // Mensaje de error detallado
```

- **`POST /api/fake-news/response`**: Gestiona el envío del análisis del usuario.
    - **Request Body**: Corresponde al tipo **`SubmitAnalysisRequest`** en **`front/src/lib/types/fakeNews.ts`**.

```
{
    "userId": "string",
    "snippetId": "string",
    "initialAssessment": { // Opcional, puede ser null
        "initialReliabilityAssessment": "reliable" | "unreliable" | "mixed"
        "confidence": number, // 1-5
        "timestamp": "Date"
    } | null,
    "userSelections": [
        {
            "id": "string", // UUID
            "text": "string",
            "markerType": "string", // "epistemic-high", etc.
            "serializedSelection": "string", // Rangy serialization
            "context": {
                "before": "string" | null,
                "after": "string" | null,
                "paragraphIndex": number | null
            },
            "timestamp": "Date"
        }
    ],
    "sourceAnalysis": {
        "checklistAnswers": {
            "hasAuthor": boolean,
            "hasDate": boolean,
```

- **Response Body (Éxito - 200 OK):** Corresponde al tipo **`SubmitAnalysisResponse`** en **`front/src/lib/types/fakeNews.ts`**

```
{
    "message": "Analysis submitted successfully",
    "performance": {
        "timeSpent": number,
        "selectionsAccuracy": number, // 0-100
        "sourceAccuracy": number, // 0-100
        "overallScore": number // 0-100
    },
    "feedback": {
        "overallScore": number,
        "expertExplanation": "string",
        "selectionsAnalysis": {
            "accuracy": number,
            "feedback": "string"
        },
        "sourceAnalysis": {
            "accuracy": number,
```

- **Response Body (Error - 400 Bad Request - Campos faltantes):**

```
{
    "message": "Missing required fields",
    "required": ["userId", "snippetId", "sourceAnalysis"]
```

- **Response Body (Error - 404 Not Found - Snippet no encontrado)**:

```
{
    "message": "Snippet not found"
}
```

- **Response Body (Error - 409 Conflict - Análisis ya completado):**

```
{
    "message": "You have already completed the analysis of
this snippet"
}
```

- **Response Body (Error - 500 Internal Server Error)**:

```
{
    "message": "Internal server error",
    "error": "string" // Mensaje de error detallado
```

- **`GET /api/fake-news/history/:userId`**: Recupera el historial de análisis del usuario.
  - **Request Params**:
    - **`userId`**: (string) ID del usuario.
  - **Request Query Params (Opcional)**:
    - **`limit`**: (number) Número máximo de respuestas a devolver (por defecto 10).
  - **Response Body (Éxito - 200 OK)**:

```
{
    "message": "History retrieved successfully",
    "responses": [ // Array de UserFakeNewsResponse simplificado
        {
            "_id": "string",
            "snippetId": { // Objeto poblado
                "_id": "string",
                "text": "string", // Solo un fragmento o título podría ser
                "type": "string",
                "topic": "string"
            },
            "performance": {
                "overallScore": number,
                "selectionsAccuracy": number,
                "sourceAccuracy": number
            },
            "submittedAt": "Date",
```

  - **Response Body (Error - 500 Internal Server Error)**:

```
{
    "message": "Internal server error",
    "error": "string" // Mensaje de error detallado
```

El controlador **fakeNewsController** implementa las siguientes funciones principales:

- **getSnippetForUser**: Esta función se encarga de proveer un snippet adecuado para el usuario.
    1. Consulta **UserFakeNewsResponse** para obtener los **snippetId** de los análisis ya completados por el **userId**.
    2. Busca en **FakeNewsSnippet** los snippets que están activos (**isActive: true**) y cuyo **_id** no está en la lista de completados.
    3. Selecciona un snippet aleatoriamente de los disponibles. Se recuperan todos los candidatos y la selección aleatoria se hace en la capa de aplicación para poder operar sobre documentos Mongoose completos.
    4. Invoca el método **incrementUsage()** del snippet seleccionado.
    5. Devuelve el snippet al frontend, pero excluyendo campos sensibles como **expertMarkers** y **expertExplanation**.

```
/**
 * Get next snippet for user - ensures user hasn't completed it before
 */
export const getSnippetForUser = async (req, res) => {
    try {
        const userId = req.params.userId;

        // Find snippets already completed by this user
        const completedResponses = await UserFakeNewsResponse.find({
            userId,
            isCompleted: true,
        }).select("snippetId");

        const completedSnippetIds = completedResponses.map((response) => response.snippetId);

        // Get random snippet that user hasn't completed
        const criteria = {
            isActive: true,
            ...(completedSnippetIds.length > 0 && { _id: { $nin: completedSnippetIds } }),
        };

        // Use find() instead of aggregation to get proper Mongoose documents
        const availableSnippets = await FakeNewsSnippet.find(criteria);

        if (!availableSnippets || availableSnippets.length === 0) {
            return res.status(404).json({
                message: "No more snippets available for this user",
                hasMore: false,
            });
        }

        // Get random snippet from available ones
        const randomIndex = Math.floor(Math.random() * availableSnippets.length);
        const snippet = availableSnippets[randomIndex];

        // Now we can safely call incrementUsage on the Mongoose document
        await snippet.incrementUsage();

        // Return snippet without expert data (user shouldn't see this)
        const responseSnippet = {…
        };

        res.status(200).json({
            message: "Snippet retrieved successfully",
            snippet: responseSnippet,
        });
    } catch (error) {
        console.error("Error fetching snippet:", error);
        res.status(500).json({
            message: "Internal server error",
            error: error.message,
        });
    }
};
```

- **submitUserResponse**: Gestiona el envío del análisis del usuario.
    1. Valida los datos recibidos (**userId**, **snippetId**, **sourceAnalysis**, etc.).
    2. Verifica si ya existe una respuesta completada para ese **userId** y **snippetId** para evitar duplicados.
    3. Recupera el **FakeNewsSnippet** completo de la base de datos, incluyendo los **expertMarkers** y **sourceInfo.checklistAnswers**.
    4. Crea una nueva instancia de **UserFakeNewsResponse** o actualiza una existente si no estaba completada.
    5. Invoca **calculateSelectionsAccuracy()** y **calculateSourceAccuracy()** en la instancia de la respuesta.
    6. Calcula el **overallScore** (media ponderada: 60% selecciones, 40% fuente).
    7. Guarda **timeSpent**, marca **isCompleted** como true y establece **submittedAt**.
    8. Genera un objeto de **feedback** estructurado que incluye las puntuaciones, la **expertExplanation** del snippet, y comentarios sobre el desempeño del usuario. Este feedback se devuelve al cliente.

```
/**
 * Submit complete user analysis
 */
export const submitUserResponse = async (req, res) => {
    try {
        const { userId, snippetId, affectiveResponse, userSelections, sourceAnalysis, timeSpent } = req.body;

        // Validate required fields
        if (!userId || !snippetId || !sourceAnalysis) {
            return res.status(400).json({
                message: "Missing required fields",
                required: ["userId", "snippetId", "sourceAnalysis"],
            });
        }

        // Check if response already exists
        const existingResponse = await UserFakeNewsResponse.findOne({
            userId,
            snippetId,
        });

        if (existingResponse && existingResponse.isCompleted) {
            return res.status(409).json({
                message: "You have already completed the analysis of this snippet",
            });
        }

        // Get the snippet with expert data for comparison
        const snippet = await FakeNewsSnippet.findById(snippetId);
        if (!snippet) {
            return res.status(404).json({
                message: "Snippet not found",
            });
        }

        // Create or update response
        const sessionId = uuidv4();
        const responseData = {…
        };

        let userResponse;
        if (existingResponse) {
            // Update existing response
            Object.assign(existingResponse, responseData);
            userResponse = existingResponse;
        } else {
            // Create new response
            userResponse = new UserFakeNewsResponse(responseData);
        }

        // Ensure performance object exists before calculations
        if (!userResponse.performance) {
            userResponse.performance = {…
            };
        }

        // Calculate performance metrics
        console.log("Calculating selections accuracy...");
        const selectionsAccuracy = userResponse.calculateSelectionsAccuracy(snippet.expertMarkers || []);
        console.log("Selections accuracy calculated:", selectionsAccuracy);
```

```
        console.log("Calculating source accuracy...");
        const sourceAccuracy = userResponse.calculateSourceAccuracy(snippet.sourceInfo?.checklistAnswers || {});
        console.log("Source accuracy calculated:", sourceAccuracy);

        // Calculate overall score (weighted average)
        const overallScore = Math.round(selectionsAccuracy * 0.6 + sourceAccuracy * 0.4);

        // Update performance object
        userResponse.performance.selectionsAccuracy = selectionsAccuracy;
        userResponse.performance.sourceAccuracy = sourceAccuracy;
        userResponse.performance.overallScore = overallScore;
        userResponse.performance.timeSpent = timeSpent || 0;
        userResponse.performance.feedbackGenerated = true;

        console.log("Final performance:", userResponse.performance);

        // Mark as completed and save
        await userResponse.markCompleted();

        // Generate feedback response
        const feedback = generateFeedback(userResponse, snippet, selectionsAccuracy, sourceAccuracy, overallScore);

        res.status(200).json({
            message: "Analysis submitted successfully",
            performance: userResponse.performance,
            feedback,
        });
    } catch (error) {
        console.error("Error submitting response:", error);
        res.status(500).json({
            message: "Internal server error",
            error: error.message,
        });
    }
};
```

- **`getUserHistory`**: Recupera el historial de análisis completados por un usuario, utilizando el método estático **`UserFakeNewsResponse.getUserResponses()`**.

```
/**
 * Get user's analysis history
 */
export const getUserHistory = async (req, res) => {
    try {
        const userId = req.params.userId;
        const limit = parseInt(req.query.limit) || 10;

        const responses = await UserFakeNewsResponse.getUserResponses(userId, limit);

        res.status(200).json({
            message: "History retrieved successfully",
            responses,
            count: responses.length,
        });
    } catch (error) {
        console.error("Error fetching user history:", error);
        res.status(500).json({
            message: "Internal server error",
            error: error.message,
        });
    }
};
```

La integración con AdminJS (configurada en **back/admin/** admin.config.js) fue una parte importante del desarrollo del backend para la gestión de contenido. Se definieron recursos para **FakeNewsSnippet** y **UserFakeNewsResponse**.

Para **FakeNewsSnippet**, se personalizaron las propiedades en **editProperties** y **listProperties**. Una decisión técnica clave fue el manejo de campos complejos anidados, especialmente **sourceInfo** y **expertMarkers**.

Para el campo **sourceInfo**, que es un subdocumento con campos requeridos como **name**, **domain**, **reliabilityLevel** y **checklistAnswers**, se optó por una aproximación de campos individuales en lugar de un editor JSON monolítico.

Para **expertMarkers**, se implementó una solución dual: el campo original de tipo **mixed** para almacenamiento estructurado y un campo virtual **expertMarkersJson** de tipo **textarea** como alternativa para la edición rápida en formato JSON. Este enfoque permite tanto edición granular campo por campo como edición rápida mediante JSON para usuarios avanzados.

Se implementó lógica especializada en los hooks **before** de las acciones **new** y **edit** para transformar los campos planos de AdminJS en objetos anidados que Mongoose espera:

- **Transformación de sourceInfo**: Los campos como **sourceInfo.name**, **sourceInfo.checklistAnswers** se recolectan y convierten en un objeto sourceInfo anidado con la estructura { **name: "...", checklistAnswers: { hasAuthor: true, ... }** }.
- **Procesamiento de expertMarkersJson**: Si el campo virtual **expertMarkersJson** contiene JSON válido, se parsea y se asigna al campo **expertMarkers**, eliminando después el campo virtual del payload.
- **Establecimiento de valores por defecto**: Para nuevos snippets, se aseguran valores por defecto apropiados para todos los campos requeridos, incluyendo un array vacío para **expertMarkers** y valores booleanos **false** para todos los criterios del checklist.

En los hooks **after** de las acciones **edit** y **show**, se implementó la lógica inversa para poblar el campo virtual **expertMarkersJson** con una representación JSON formateada de los **expertMarkers** existentes, facilitando la visualización y edición posterior.

```
actions: {
    new: {
        before: async (request) => {
            // Handle expertMarkersJson quick editor
            if (request.payload.expertMarkersJson && request.payload.expertMarkersJson.trim()) {
                try {
                    const parsedMarkers = JSON.parse(request.payload.expertMarkersJson);
                    request.payload.expertMarkers = parsedMarkers;
                    delete request.payload.expertMarkersJson;
                } catch (e) {
                    // If JSON is invalid, keep the field for user to fix
                }
            }

            // Convert flat sourceInfo fields to nested object
            const sourceInfoFields = {};
            const checklistAnswers = {};

            // Collect sourceInfo fields from payload
            Object.keys(request.payload).forEach((key) => {
                if (key.startsWith("sourceInfo.")) {
                    const fieldPath = key.substring("sourceInfo.".length);

                    if (fieldPath.startsWith("checklistAnswers.")) {
                        const checklistField = fieldPath.substring("checklistAnswers.".length);
                        checklistAnswers[checklistField] = request.payload[key];
                    } else {
                        sourceInfoFields[fieldPath] = request.payload[key];
                    }

                    // Remove the flat field from payload
                    delete request.payload[key];
                }
            });

            // Build the sourceInfo object
            if (Object.keys(sourceInfoFields).length > 0 || Object.keys(checklistAnswers).length > 0) {
                request.payload.sourceInfo = {
                    ...sourceInfoFields,
                    checklistAnswers: checklistAnswers,
                };
            }

            // Set default values for new snippets
            if (request.method === "post") {
                if (!request.payload.expertMarkers) {
                    request.payload.expertMarkers = [];
                }

                // Ensure sourceInfo has all required fields
                if (!request.payload.sourceInfo) {
                    request.payload.sourceInfo = {};
                }

                // Set defaults
                if (!request.payload.sourceInfo.name) {
                    request.payload.sourceInfo.name = "";
                }
                if (!request.payload.sourceInfo.reliabilityLevel) {
                    request.payload.sourceInfo.reliabilityLevel = "unreliable";
                }
                if (!request.payload.sourceInfo.checklistAnswers) {
                    request.payload.sourceInfo.checklistAnswers = {};
                }

                // Set checklist defaults
                const checklistDefaults = {…
                };

                Object.keys(checklistDefaults).forEach((field) => {
                    if (request.payload.sourceInfo.checklistAnswers[field] === undefined) {
                        request.payload.sourceInfo.checklistAnswers[field] = checklistDefaults[field];
                    }
                });
            }

            return request;
        },
    },
```

```
edit: {
    before: async (request) => {
        // Handle expertMarkersJson quick editor
        if (request.payload.expertMarkersJson && request.payload.expertMarkersJson.trim()) {
            try {
                const parsedMarkers = JSON.parse(request.payload.expertMarkersJson);
                request.payload.expertMarkers = parsedMarkers;
                delete request.payload.expertMarkersJson;
            } catch (e) {…
            }
        }
        // Convert flat sourceInfo fields to nested object (same logic as new)
        const sourceInfoFields = {};
        const checklistAnswers = {};
        Object.keys(request.payload).forEach((key) => {
            if (key.startsWith("sourceInfo.")) {
                const fieldPath = key.substring("sourceInfo.".length);
                if (fieldPath.startsWith("checklistAnswers.")) {
                    const checklistField = fieldPath.substring("checklistAnswers.".length);
                    checklistAnswers[checklistField] = request.payload[key];
                } else {
                    sourceInfoFields[fieldPath] = request.payload[key];
                }
                // Remove the flat field from payload
                delete request.payload[key];
            }
        });
        // Build the sourceInfo object
        if (Object.keys(sourceInfoFields).length > 0 || Object.keys(checklistAnswers).length > 0) {
            request.payload.sourceInfo = {
                ...sourceInfoFields,
                checklistAnswers: checklistAnswers,
            };
        }
        return request;
    },

    after: async (response) => {
        // Populate the virtual expertMarkersJson field for display
        if (response.record && response.record.params && response.record.params.expertMarkers) {
            let markers = response.record.params.expertMarkers;
            if (typeof markers === "string") {
                try {
                    markers = JSON.parse(markers);
                } catch (e) {
                    markers = [];
                }
            }
            response.record.params.expertMarkersJson = JSON.stringify(markers, null, 2);
        }
        return response;
    },
},
```

Para **UserFakeNewsResponse**, las acciones de **edit** y **new** se deshabilitaron (**isVisible: false**) ya que estas respuestas son generadas automáticamente por el sistema cuando los usuarios completan sus análisis, no administradas manualmente. Se configuraron propiedades específicas para visualización, incluyendo el uso del **jsonViewerComponent** para campos complejos como **userSelections** y **sourceAnalysis**.checklistAnswers, permitiendo a los

administradores revisar las respuestas de los usuarios en un formato legible sin posibilidad de modificación accidental.

Esta configuración de AdminJS proporciona una interfaz de administración robusta que facilita la curación de contenido por parte de expertos mientras mantiene la integridad de los datos de respuesta de los usuarios.

4.2.3 Validación y Optimización del Backend

La validación de la funcionalidad del backend se realizó principalmente mediante pruebas manuales durante el desarrollo, invocando los endpoints API a través de herramientas como Postman y directamente desde el frontend en desarrollo. Se verificó la correcta creación y recuperación de datos, el cálculo de métricas y la generación de feedback.

Para optimizar el rendimiento de las consultas, se definieron índices estratégicos en los esquemas Mongoose (ver sección 4.2.1), como en `userId` y `snippetId` en `UserFakeNewsResponse` para búsquedas frecuentes.

No se desarrolló una suite de pruebas automatizadas (unitarias o de integración) para los componentes del backend. De manera similar, no se realizaron pruebas de carga formales para cuantificar el impacto de los índices bajo alta concurrencia o grandes volúmenes de datos. La implementación de un plan de pruebas automatizadas y pruebas de carga se considera una línea de trabajo futuro esencial para asegurar la robustez y escalabilidad a largo plazo del backend.

4.3 Implementación del Frontend

La implementación del frontend se realizó utilizando SvelteKit, aprovechando su reactividad, sistema de componentes y herramientas para construir una interfaz de usuario interactiva y eficiente. El desarrollo se centró en traducir el diseño de la experiencia de usuario (flujo de cuatro pasos, gestión de estado, interacción de resaltado) en componentes Svelte funcionales y bien estructurados, escritos en TypeScript para mayor robustez.

4.3.1 Desarrollo de Stores Svelte para el Estado

Para manejar el estado global de la aplicación, especialmente la información relacionada con las selecciones de texto del usuario y el estado de la interfaz de resaltado, se optó por el sistema de stores de Svelte. Un store central en esta arquitectura es **`highlightingStore.ts`** (definido en **`front/src/lib/stores/highlightingStore.ts`**).

Este store se diseñó como un **`writable`** store que contiene un objeto **`HighlightingState`**. El estado incluye:

- **`selections`**: Un array de objetos **`UserSelection`** que representa todas las selecciones que el usuario ha realizado y clasificado.

- **`isSelectionMode`**: Un booleano que indica si el modo de selección de texto está activo.
- **`currentSelection`**: Un objeto que almacena temporalmente los detalles de una selección de texto recién hecha (texto, rango serializado por Rangy, posición en pantalla) antes de que el usuario la clasifique, utilizado para mostrar el **`SelectionPopup`**.
- **`popupVisible`**: Un booleano para controlar la visibilidad del **`SelectionPopup`**.
- **`containerElement`**: Una referencia al elemento HTML que contiene el texto del snippet, necesario para el servicio de resaltado.

```
// Highlighting state
interface HighlightingState {
    selections: UserSelection[];
    isSelectionMode: boolean;
    currentSelection: {
        text: string;
        range: string; // Rangy serialized range
        position: { x: number; y: number };
    } | null;
    popupVisible: boolean;
    containerElement: HTMLElement | null;
}

// Initial state
const initialState: HighlightingState = {
    selections: [],
    isSelectionMode: false,
    currentSelection: null,
    popupVisible: false,
    containerElement: null
};

// Main store
export const highlightingState = writable<HighlightingState>(initialState);
```

A partir de este store principal, se derivan otros stores (**`derived`**) para facilitar el acceso a partes específicas del estado desde los componentes, como **`selections`**, **`isSelectionMode`**, **`currentSelection`** y **`popupVisible`**.

El store también expone un objeto **`highlightingActions`** que agrupa todas las funciones para modificar el estado. Estas acciones son invocadas por los componentes Svelte o por el **`highlightingService`** para mantener una gestión del estado centralizada y predecible. Algunas acciones clave implementadas son:

- **`initializeContainer(element: HTMLElement):`** Configura el elemento contenedor e inicializa el modo de selección.
- **`cleanup():`** Restablece el store a su estado inicial, limpiando selecciones y desactivando el modo.
- **`showSelectionPopup(text: string, range: string, position: { x: number; y: number }):`** Muestra el popup de clasificación de marcadores.
- **`hideSelectionPopup():`** Oculta el popup.
- **`addSelection(selection: UserSelection):`** Añade una nueva selección clasificada al array `selections`. Maneja la lógica de reemplazar una selección existente si se vuelve a marcar el mismo texto o se edita.
- **`removeSelection(selectionId: string):`** Elimina una selección del store.
- **`removeSelectionComplete(selectionId: string):`** Orquesta la eliminación de la selección tanto del store como del resaltado visual en el DOM (invocando a `highlightingService`).
- **`clearAllSelectionsComplete():`** Limpia todas las selecciones del store y del DOM.
- **`loadSelections(selections: UserSelection[]):`** Permite cargar selecciones preexistentes en el store (útil si se implementara la edición de análisis previos).
- **`getSelectionsForSubmission():`** Prepara las selecciones para ser enviadas al backend.

Este diseño de store proporciona una fuente única de verdad para el estado del resaltado, desacoplando la lógica de estado de los componentes de la interfaz y facilitando la comunicación entre diferentes partes de la aplicación frontend.

4.3.2 Implementación de Componentes Svelte

La interfaz de usuario se construyó como una jerarquía de componentes Svelte, cada uno con responsabilidades bien definidas, siguiendo el diseño conceptual del Capítulo 3. El componente principal que orquesta la experiencia del módulo es **`FakeNewsAnalyzer.svelte`**. Este componente actúa como el contenedor principal y gestor del flujo de los cuatro pasos secuenciales del análisis.

`FakeNewsAnalyzer.svelte` (Componente Orquestador Principal):

Este componente es central para la funcionalidad del módulo. Sus responsabilidades clave, tal como se implementan, son:

- **Gestión del Estado del Flujo**: Mantiene el estado **`currentStep`** (de tipo **`AnalysisStep`**) que determina cuál de los componentes de paso está activo y visible. Recibe **`currentStep`** como una prop, lo que permite que

un componente padre pueda controlar el paso inicial o reaccionar a cambios.

- **Carga de Datos**: En el `onMount`, inicia la carga del primer snippet disponible a través de `fakeNewsService.getNextSnippet()`. Gestiona los estados de `isLoading` y `error` durante este proceso.

- **Orquestación de Pasos**:

    - Renderiza condicionalmente los componentes de paso (`InitialAssessmentStep`, `TextAnalysisStep`, `SourceChecklistStep`, `FeedbackDisplay`) según el valor de `currentStep`.
    - Maneja los eventos emitidos por estos componentes de paso para avanzar en el flujo. Por ejemplo, `handleInitialResponse` actualiza `initialResponse` y cambia `currentStep` a `"text-analysis"`.
    - `proceedToSourceAnalysis` valida que haya selecciones antes de pasar a `"source-evaluation"`.
    - `handleSourceAnalysis` recibe los datos del `SourceChecklistStep` y luego llama a `submitAnalysis`.

- **Interacción con Servicios y Stores**:

    - Utiliza `fakeNewsService` para obtener snippets y enviar el análisis completo.
    - Se suscribe a `highlightingState` para obtener `selections`, `popupVisible` y `currentSelection`, necesarios para la lógica de validación y para pasar al `SelectionPopup`.
    - Utiliza `highlightingActions` para limpiar el estado del resaltado (`cleanup()`) al iniciar un nuevo análisis y `highlightingService.destroy()` en `onDestroy`.
    - Obtiene el `authUserId` del `authService` para incluirlo en la petición de envío de análisis.

- **Manejo de Datos del Análisis**: Acumula los datos de cada paso (`initialResponse`, `sourceAnalysis`) y utiliza `highlightingActions.getSelectionsForSubmission()` para obtener las selecciones formateadas. Calcula `timeSpent` antes de enviar el análisis.

- **Navegación y UI**:
    - Proporciona funciones como `startNewAnalysis()` para reiniciar el proceso y `goBack()` para retroceder entre ciertos pasos.
    - Incluye un `SelectionPopup` que se muestra condicionalmente.
    - Maneja la visualización de errores y estados de carga.
    - Implementa `scrollToTop()` para mejorar la UX al cambiar de paso.
    - Despacha un evento `stepChange` para que componentes padres puedan reaccionar a los cambios de paso.
    - Expone una función `navigateToStep` para permitir la navegación programática a un paso específico, validando si la navegación está permitida.

- **Ciclo de Vida**: Utiliza **`onMount`** para la carga inicial y **`onDestroy`** para la limpieza de recursos del **`highlightingService`** y **`highlightingStore`**.

Los otros componentes clave, cuya implementación se deduce y se integra con **`FakeNewsAnalyzer.svelte`**, son:

- **Componentes de Paso**:
    - **`InitialAssessmentStep.svelte`**: Recibe **`currentSnippet`** como prop. Emite un evento **`response`** con los datos de la respuesta preliminar, que es manejado por **`handleInitialResponse`** en **`FakeNewsAnalyzer.svelte`**.
    - **`TextAnalysisStep.svelte`**: Recibe **`currentSnippet`** y **`selections`** (del store, pasadas por **`FakeNewsAnalyzer.svelte`**). Emite un evento **`continue`** (manejado por **`proceedToSourceAnalysis`**) y un evento **`back`** (manejado por **`goBack`**). Internamente, este componente sería el responsable de inicializar el **`highlightingService`** con el contenedor del texto del snippet y de mostrar el **`SelectionPopup.svelte`** cuando una selección de texto es realizada por el usuario.
    - **`SourceChecklistStep.svelte`**: Recibe **`currentSnippet`**. Emite un evento **`analysis`** con los datos de la evaluación de la fuente (manejado por **`handleSourceAnalysis`**) y un evento **`back`**. También recibe una prop **`disabled`** para controlar la interacción durante el envío.
    - **`FeedbackDisplay.svelte`**: Recibe **`feedback`** y **`selections`** como props. Emite un evento **`newAnalysis`** (manejado por **`startNewAnalysis`**).
- **Componentes de Soporte Especializados**:
    - **`SnippetDisplay.svelte`**: Es utilizado dentro de `InitialAssessmentStep.svelte`, `TextAnalysisStep.svelte` y `SourceChecklistStep.svelte`. Recibiría el `currentSnippet` y una prop `isInteractive`. En modo interactivo (en `TextAnalysisStep`), colaboraría estrechamente con `highlightingService` (inicializándolo con su elemento DOM contenedor) y `highlightingStore` (para mostrar los resaltados aplicados).
    - **`SelectionPopup.svelte`**: Se renderiza dentro de `FakeNewsAnalyzer.svelte`. Recibe `isVisible`, `position` y `selectedText` del `highlightingState`. Cuando el usuario selecciona un `MarkerType`, emite un evento `markerSelect`. `FakeNewsAnalyzer.svelte` maneja este evento, llama a `highlightingService.applyHighlight()` y, si es exitoso, a `highlightingActions.addSelection()`. Si no, oculta el popup. También maneja el evento **`close`** para ocultar el popup.

La comunicación entre estos componentes se maneja mediante props para el flujo de datos descendente y el despacho de eventos personalizados de Svelte para el flujo ascendente o acciones que modifican los stores globales.

4.3.3 Implementación de la Lógica de Highlighting con Rangy.js

La funcionalidad de resaltado interactivo es una de las implementaciones más complejas y se apoya fuertemente en la librería Rangy.js, abstraída a través del **`highlightingService.ts`** (**`front/src/lib/functions/highlightingService.ts`**) y gestionada por el **`highlightingStore.ts`**.

El **`HighlightingService`** encapsula toda la interacción directa con Rangy.js:

- **Inicialización (`initialize(containerElement: HTMLElement)`)**:
    1. Carga dinámica de Rangy.js y sus módulos (`rangy-core`, `rangy-classapplier`, `rangy-serializer`) si aún no están presentes en el objeto `window`. Esto se hace creando etiquetas `<script>` y añadiéndolas al `document.head`.
    2. Una vez cargado, inicializa Rangy (`this.rangy.init()`).
    3. Crea **`ClassApplier`** de Rangy para cada uno de los siete `MarkerType`. Cada `ClassApplier` se configura para envolver el texto seleccionado con una etiqueta `<mark>` y aplicar una clase CSS específica, por ejemplo, `highlight-epistemic-high`, y atributos de datos (`data-marker-type`, `data-highlight-id`).

```
/**
 * Create class appliers for each marker type
 */
private createClassAppliers(): void {
    const markerTypes: MarkerType[] = [
        'epistemic-high', 'epistemic-low', 'emotional-negative',
        'emotional-positive', 'bias-loaded', 'authority-claim', 'statistical-claim'
    ];

    markerTypes.forEach(markerType => {
        const className = `highlight-${markerType}`;
        this.classAppliers[markerType] = this.rangy.createClassApplier(className, {
            elementTagName: 'mark',
            elementAttributes: {
                'data-marker-type': markerType,
                'data-highlight-id': '' // Will be set dynamically
            }
        });
    });
}
```

    4. Configura listeners de eventos (**`mouseup`**, **`touchend`**) en el **`containerElement`** para detectar cuándo el usuario finaliza una selección de texto.

- **Manejo de Selección de Texto (`handleTextSelection`, `processSelection`)**:
    1. Cuando se detecta un evento de selección, y tras un breve **`setTimeout`** (expandido en la sección 4.5) para asegurar que la selección se ha estabilizado, se obtiene la selección nativa del navegador (**`window.getSelection()`**).

2. Si la selección no está vacía y tiene una longitud mínima, se utiliza Rangy para serializarla (**`this.rangy.serializeSelection()`**). Esta cadena serializada es crucial para la persistencia y reconstrucción del resaltado.
3. Se calcula la posición del popup y se invoca **`highlightingActions.showSelectionPopup()`** con el texto seleccionado, la serialización y la posición.

```
/**
 * Handle text selection
 */
private handleTextSelection(event: MouseEvent | TouchEvent): void {
    if (!this.isInitialized) {
        return;
    }

    // Clear any existing timeout
    if (this.selectionTimeout) {
        clearTimeout(this.selectionTimeout);
    }

    // Add small delay to let selection settle
    this.selectionTimeout = window.setTimeout(() => {
        this.processSelection(event);
    }, 50);
}
```

```
/**
 * Process the text selection
 */
private processSelection(event: MouseEvent | TouchEvent): void {
    const nativeSelection = window.getSelection();

    if (!nativeSelection || nativeSelection.rangeCount === 0) {
        highlightingActions.hideSelectionPopup();
        return;
    }

    const selectedText = nativeSelection.toString().trim();

    if (selectedText.length < 2) {
        highlightingActions.hideSelectionPopup();
        return;
    }

    try {
        // Get position for popup
        const nativeRange = nativeSelection.getRangeAt(0);
        const rect = nativeRange.getBoundingClientRect();
        const position = {
            x: rect.left + rect.width / 2,
            y: rect.top + window.scrollY - 10
        };

        // Serialize selection using Rangy
        const serializedSelection = this.rangy.serializeSelection();

        // Show popup
        highlightingActions.showSelectionPopup(
            selectedText,
            serializedSelection,
            position
        );

    } catch (error) {
        console.error('Error processing selection:', error);
        highlightingActions.hideSelectionPopup();
    }
}
```

- **Aplicación de Resaltado (`applyHighlight(markerType: MarkerType)`)**:
    1. Esta función es llamada cuando el usuario selecciona un tipo de marcador en el **`SelectionPopup`**.
    2. Obtiene la selección actual de Rangy (**`this.rangy.getSelection()`**).
    3. Utiliza el **`ClassApplier`** correspondiente al **`markerType`** para aplicar el resaltado visual al texto seleccionado (`applier.applyToSelection()`).
    4. Genera un **`uuidv4()`** para la nueva selección.
    5. Encuentra el elemento **`<mark>`** recién creado y le asigna el **`data-highlight-id`** con el UUID.
    6. Crea un objeto **`UserSelection`** con el **`id`**, **`text`**, **`markerType`**, **`serializedSelection`**, **`context`** (extraído usando Rangy para obtener texto antes/después y el índice del párrafo) y **`timestamp`**.
    7. Limpia la selección del navegador.
    8. Devuelve el objeto **`UserSelection`** para que sea añadido al `highlightingStore`.

```
/**
 * Apply highlight with a specific marker type
 */
applyHighlight(markerType: MarkerType): UserSelection | null {
    if (!this.isInitialized) {…
    }

    try {
        const rangySelection = this.rangy.getSelection();

        if (rangySelection.rangeCount === 0) {
            return null;
        }

        const selectedText = rangySelection.toString().trim();
        const serializedSelection = this.rangy.serializeSelection();
        const selectionId = uuidv4();

        // Apply class applier
        const applier = this.classAppliers[markerType];
        applier.applyToSelection();

        // Find and mark the created element
        const highlightElement = this.findNewestHighlightElement(selectedText);
        if (highlightElement) {
            highlightElement.setAttribute('data-highlight-id', selectionId);
        }

        // Create UserSelection object
        const userSelection: UserSelection = {
            id: selectionId,
            text: selectedText,
            markerType,
            serializedSelection,
            context: this.extractContext(rangySelection),
            timestamp: new Date()
        };

        // Clear selections
        rangySelection.removeAllRanges();
        const nativeSelection = window.getSelection();
        if (nativeSelection) {
            nativeSelection.removeAllRanges();
        }

        return userSelection;
```

- **Eliminación de Resaltado (removeHighlight(selectionId: string))**:
    1. Encuentra el elemento <mark> en el DOM que tiene el data-highlight-id correspondiente.
    2. Si se encuentra, reemplaza el elemento <mark> con su contenido textual (deshaciendo el resaltado).
    3. Normaliza el nodo padre para fusionar nodos de texto adyacentes.

- **Restauración de Resaltados (`loadExistingHighlights`, `restoreHighlight`)**:

    La implementación final adoptó una **estrategia dual** para la restauración de highlights que combina la precisión de la serialización de Rangy.js con un mecanismo de fallback robusto para garantizar la funcionalidad en todos los escenarios:

    1. **Estrategia Principal - Deserialización de Rangy**:

```
this.rangy.deserializeSelection(selection.serializedSelection)
```

        - El sistema intenta primero utilizar la deserialización nativa de Rangy.js, que proporciona la máxima precisión al reconstruir exactamente la selección original basándose en la estructura del DOM serializada.

    2. **Estrategia de Fallback - Reconstrucción Contextual:** Cuando la deserialización de Rangy falla (típicamente debido a cambios en el DOM que alteran los checksums internos), el sistema emplea un algoritmo de fallback inteligente:

```
private async recreateHighlightFromTextAndContext(selection: UserSelection): Promise<boolean>
```

    Este algoritmo involucra:

    - **Búsqueda de Texto Completa**: Recorre todos los nodos de texto del contenedor y construye un mapa completo del contenido.
    - **Detección de Múltiples Ocurrencias**: Identifica todas las instancias del texto seleccionado en el documento.
    - **Desambiguación por Contexto**: Utiliza los campos `context.before`, `context.after` y `context.paragraphIndex` del objeto `UserSelection` para determinar cuál de las múltiples ocurrencias corresponde a la selección original.
    - **Puntuación de Similitud**: Implementa un sistema de scoring que evalúa la similitud del contexto circundante para seleccionar la coincidencia más probable.
    - **Creación Manual de Highlights**: Genera elementos `<mark>` con los atributos correctos (`data-marker-type`, `data-highlight-id`) para recrear el resaltado visual.

- **Limpieza (`clearAllHighlights`, `destroy`)**:
    1. **`clearAllHighlights()`** elimina todos los elementos **`<mark>`** del contenedor.
    2. **`destroy()`** elimina los listeners de eventos y limpia los resaltados, usualmente llamado cuando el componente que usa el servicio se desmonta.

La interacción entre **`highlightingService`** y **`highlightingStore`** es fundamental: el servicio modifica el DOM y luego actualiza el store, o lee el store para reflejar cambios en el DOM.

4.3.4 Implementación de la Lógica de Página y Gestión del Flujo de Estados

La página principal del módulo "Stance & Source Check", ubicada en la ruta **`/mooc/course4/`**, se define mediante dos archivos clave: **`front/src/routes/mooc/course4/+page.server.ts`** para la lógica del lado del servidor y **`front/src/routes/mooc/course4/+page.svelte`** para la estructura y comportamiento en el lado del cliente.

- **Lógica del Lado del Servidor (`+page.server.ts`)**: El archivo **`+page.server.ts`** se ejecuta en el servidor antes de que la página se renderice. Su función **`load`** principal es:
    - **Verificar Autenticación**: Utiliza **`locals.isAuthenticated`** (establecido por **`hooks.server.ts`**) para asegurar que el usuario haya iniciado sesión. Si no está autenticado, se podría redirigir o devolver un error. En la implementación actual, devuelve un objeto que indica el error de autenticación.
    - **Proveer Datos Iniciales del Usuario**: Si está autenticado, pasa información básica del usuario como **`userId`**, **`userEmail`**, e **`isAuthenticated`** a la componente de página (**`+page.svelte`**).
    - **Indicador de Contenido Disponible**: Devuelve **`hasAvailableContent: true`** como un valor por defecto, asumiendo que hay contenido. La verificación real de si hay snippets específicos para el usuario se delega al cliente para una carga inicial más rápida de la página. El **`availableSnippet`** se establece en **`null`** ya que se cargará en el cliente.

```
export const load: PageServerLoad = async ({ locals }) => {
    try {
        // Verify authentication (already handled by hooks.server.ts)
        if (!locals.isAuthenticated) {
            return {
                availableSnippet: null,
                hasAvailableContent: false,
                error: "User not authenticated",
            };
        }

        // Get next available snippet for the user
        // Note: We use a simple approach here since fakeNewsService
        // is designed for the client, but we need data on the server

        // For server load, we only check if there's available content
        // The specific snippet will be loaded on the client for better UX
        return {
            availableSnippet: null, // Will be loaded on the client
            hasAvailableContent: true, // We assume there's content
            userId: locals.userId,
            userEmail: locals.email,
            isAuthenticated: locals.isAuthenticated,
        };
    } catch (error) {
        console.error("Error loading Course 4 data:", error);
        return {
            availableSnippet: null,
            hasAvailableContent: false,
            error: "Error loading course data",
        };
    }
};
```

Lógica y Estructura del Lado del Cliente (`+page.svelte`): El archivo `+page.svelte` define la interfaz de usuario principal para el Curso 4.

1. **Estado de la Página**: Mantiene variables de estado locales como `isLoading`, `hasAvailableContent`, `error` y `currentStep` (que se enlaza bidireccionalmente con el `FakeNewsAnalyzer`). También mantiene una referencia a la instancia del componente `analyzerComponent` (`bind:this={analyzerComponent}`).

2. **Ciclo de Vida (`onMount`)**:
    - Verifica la autenticación del lado del cliente usando el store `$isAuthenticated` del `authService`. Si no está autenticado, redirige a la página de login usando `goto("/login?redirectTo=...")`.

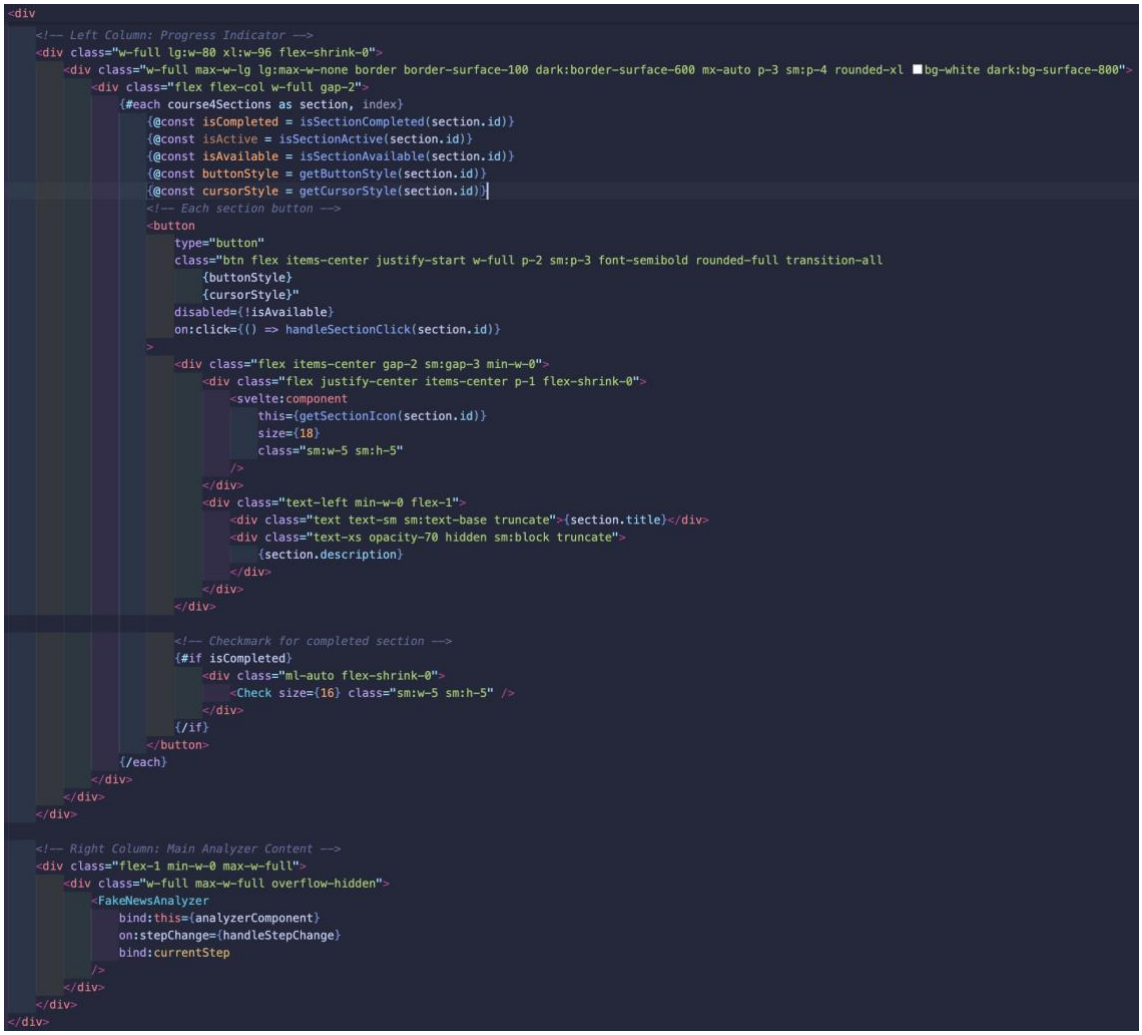
- Llama a **`fakeNewsService.hasAvailableSnippets()`** para determinar si realmente hay contenido disponible para el usuario. Actualiza **`hasAvailableContent`** y **`isLoading`** en consecuencia.

3. **Renderizado Condicional**: La UI cambia significativamente basada en los estados **`isLoading`**, **`error`**, y **`hasAvailableContent`**:
   - Muestra un **`LoadingSpinner`** mientras **`isLoading`** es **`true`**.
   - Muestra un mensaje de error si la variable error tiene contenido.
   - Si **`!hasAvailableContent`** (y no hay errores y no está cargando), muestra un mensaje de felicitación indicando que todo el contenido ha sido completado, junto con un resumen de "logros" y enlaces al dashboard y otros cursos.
   - Si hay contenido disponible, renderiza el layout principal de dos columnas.

4. **Layout de Dos Columnas (Contenido Disponible)**:

   - **Columna Izquierda (Indicador de Progreso)**:
     - Define **`course4Sections`**, un array de objetos que describe cada paso del análisis (**`initial-assessment`**, **`text-analysis`**, etc.) con título, ícono y descripción.
     - Itera sobre **`course4Sections`** para renderizar una lista de botones de navegación.
     - La apariencia y comportamiento de cada botón (activo, completado, disponible, deshabilitado) se gestiona mediante funciones como **`isSectionCompleted`**, **`isSectionActive`**, **`isSectionAvailable`**, **`getButtonStyle`**, y **`getCursorStyle`**, todas basadas en el **`currentStep`** actual.
     - Al hacer clic en un botón de sección (**`handleSectionClick`**), si la sección es navegable, llama al método **`analyzerComponent.navigateToStep(sectionId)`** para cambiar el paso dentro del `FakeNewsAnalyzer`.

   - **Columna Derecha (Contenido del Analizador)**:
     - Monta el componente **`FakeNewsAnalyzer`**.
     - Utiliza **`bind:this={analyzerComponent}`** para obtener una referencia a la instancia del analizador.
     - Utiliza **`on:stepChange={handleStepChange}`** para escuchar los cambios de paso emitidos por **`FakeNewsAnalyzer`** y actualizar el **`currentStep`** local de la página, manteniendo así sincronizado el indicador de progreso.
     - Utiliza **`bind:currentStep`** para un enlace bidireccional, permitiendo que **`FakeNewsAnalyzer`** actualice el **`currentStep`** de la página y viceversa (aunque la navegación desde el indicador de progreso se hace explícitamente con `navigateToStep`).



5. **Navegación**: La función **`goBackToMOOC()`** utiliza **`goto("/mooc")`** para volver a la página principal del MOOC.

Este componente **`+page.svelte`** actúa como el anfitrión del **`FakeNewsAnalyzer.svelte`**, proporcionando una capa de UI adicional para la navegación entre pasos y la gestión de estados de carga/error a nivel de página, mientras delega la lógica principal del análisis al **`FakeNewsAnalyzer`**. La comunicación entre la página y el analizador se logra mediante la referencia al componente (**`bind:this`**), la escucha de eventos (**`on:stepChange`**) y la invocación de métodos expuestos (**`navigateToStep`**).

4.4 Proceso de Curación e Integración del Contenido

La efectividad pedagógica depende críticamente de la calidad, autenticidad y diversidad de los snippets de noticias presentados al usuario. Por ello, la implementación del proceso de curación e integración de contenido fue una fase esencial, aplicando el diseño conceptual detallado en la sección 3.4. Este

proceso no solo implicó la creación original de snippets, sino también la adaptación de ejemplos del mundo real para construir un corpus de aprendizaje relevante y desafiante.

1. Metodología Híbrida para la Obtención de Snippets:

Para la creación del conjunto inicial de snippets, se adoptó una metodología híbrida:

- **Adaptación de Contenido Real de Plataformas de Fact-Checking**: Una parte significativa de los snippets se planificó obtener a partir del análisis de casos reales de desinformación y noticias falsas documentados por organizaciones de verificación de hechos reconocidas. Se consideraron fuentes como:
    - **Newtral.es**: para contenido en español, que luego sería cuidadosamente traducido al inglés, idioma principal de la plataforma DeStance, manteniendo la intención y los matices del mensaje original.
    - **Politifact.com**: especialmente útil para obtener ejemplos de publicaciones en redes sociales de figuras públicas, empresas o políticos que han sido verificadas.
    - **EUvsDisinfo.eu**: para casos de desinformación con un enfoque europeo.
    - **Factcheck.org**: y otras bases de datos de fake news.

    El objetivo de utilizar estos casos reales es exponer a los usuarios a los tipos de desinformación que circulan activamente, aumentando la relevancia y aplicabilidad del aprendizaje.

- **Creación Original de Snippets**: De forma complementaria, se procedió a la redacción de snippets originales. Esta vía permitió:
    - Crear ejemplos específicos que ilustraran de manera muy clara ciertos tipos de marcadores lingüísticos o tácticas de desinformación que podrían ser más difíciles de aislar en ejemplos reales.
    - Asegurar una cobertura equilibrada de las tipologías de contenido definidas en 3.4.1 (combinaciones de veracidad/sesgo, formato y temática), especialmente para llenar vacíos si los ejemplos reales no cubrían alguna categoría específica de manera adecuada.
    - Controlar la longitud y complejidad del texto para adaptarlo a diferentes niveles de dificultad previstos.

2. Aplicación Práctica de la Curación Experta:

Independientemente de si el *snippet* base provenía de una fuente real adaptada o era de creación original, cada uno pasó por el siguiente proceso de curación detallado:

- **Clasificación y Contextualización**: Cada snippet fue clasificado según su tipología, tal como se diseñó en 3.4.1.

- **Definición de `expertMarkers`**: Este fue uno de los pasos más críticos. Para cada snippet, un experto (o el rol asumido por el desarrollador del TFG en esta fase,) identificó los fragmentos de texto relevantes y los anotó como **`expertMarkers`**. Esto implicó:

    - Seleccionar el texto exacto del marcador.
    - Asignarle el **`markerType`** correspondiente de entre los siete definidos.
    - **Determinación de `startIndex` y `endIndex`**: Aunque la lógica de comparación de precisión de selecciones implementada en el backend (**`UserFakeNewsResponse.methods.calculateSelectionsAccuracy`**) actualmente se centra en el tipo de marcador y en la coincidencia textual (permitiendo cierta flexibilidad como la inclusión de cadenas), la definición precisa de **`startIndex`** y **`endIndex`** para cada **`expertMarker`** sigue siendo un paso importante en el proceso de curación. Estos índices, que indican la posición exacta del marcador en el texto del snippet (contando caracteres, incluyendo espacios, desde el inicio), son cruciales para varias razones como: la definición inequívoca del marcador experto, eliminando ambigüedades si el mismo texto aparece múltiples veces; *future-proofing* y funcionalidades avanzadas; y la validación de datos y depuración.
    - Asignar una **`explanation`** pedagógica breve si fuera necesario.

- **Completado de `sourceInfo`**: Para cada snippet, se definió la información de la fuente, incluyendo **`name`** (real o ficticio, según el origen del snippet), **`domain`** (si aplicaba), **`reliabilityLevel`** (la evaluación de la fiabilidad del snippet específico, no del source) y las **`checklistAnswers`**.

- **Redacción de `expertExplanation`**: Se elaboró una explicación global para cada snippet, detallando por qué es problemático, qué estrategias de desinformación emplea, o qué elementos clave debería identificar un análisis crítico.

3. Integración del Contenido mediante AdminJS:

Una vez curado un snippet con toda su metainformación experta, el siguiente paso fue su integración en la base de datos MongoDB. Para ello, se utilizó la interfaz de administración de AdminJS, configurada como parte del backend (ver sección 4.2.2, **`admin.config.js`**).

- **Acceso al Panel de `FakeNewsSnippet`**: Se accedió al recurso **`FakeNewsSnippet`** dentro de AdminJS.

- **Creación de Nuevos Snippets**: Utilizando la acción "New", se abrió el formulario de creación.
    - Se introdujo el **`text`** del snippet, **`imageUrl`** (si la había), y se seleccionaron los valores para **`type`**, **`format`** y **`topic`** de las listas desplegables configuradas.
    - Para los **`expertMarkers`**, se utilizó la opción de edición que permitiera ingresar el array de marcadores. La implementación de AdminJS contemplaba tanto un campo **`textarea`** para JSON (**`expertMarkersJson`**) como la posibilidad de campos anidados. El flujo implicaba asegurar que el JSON de los marcadores fuera válido y siguiera la estructura definida.
    - De manera similar, se introdujo la información de **`sourceInfo`** compuesta por el nombre, dominio, fiabilidad y las respuestas del checklist y la **`expertExplanation`**.

- **Activación del Snippet**: Finalmente, se marcó el snippet como activo (**`isActive: true`**) para asegurar que pudiera ser recuperado por el controlador **`getSnippetForUser`** y presentado a los usuarios en el frontend.

Ejemplo de un Objeto `FakeNewsSnippet` Completo:

Para ilustrar la estructura final de los datos de un snippet en formato JSON entero, es decir, que así no se introduciría directamente (salvo el contenido de **`expertMarkers`** que sí se podría introducir como JSON en su campo respectivo), sino que es meramente para dar una idea del contenido de este objeto **`FakeNewsSnippet`**,a continuación se presenta un ejemplo completo del objeto:

```
{
  "text": "BREAKING: Local authorities confirm 90% increase in crime rates in neighborhoods with high immigrant populations. Experts say this trend is 'clearly devastating' our communities. Why isn't mainstream media reporting this obvious crisis? #Immigration #Truth",
  "imageUrl": null,
  "type": "fake",
  "format": "tweet",
  "topic": "immigration",
  "expertMarkers": [
    {
      "text": "90% increase",
      "markerType": "statistical-claim",
      "startIndex": 38,
      "endIndex": 50,
      "explanation": "Unverified statistical claim without source or methodology."
    },
    {
      "text": "clearly devastating",
      "markerType": "epistemic-high",
      "startIndex": 127,
      "endIndex": 145,
      "explanation": "Expression of absolute certainty without evidence."
    },
    {
      "text": "obvious crisis",
      "markerType": "emotional-negative",
      "startIndex": 208,
      "endIndex": 222,
      "explanation": "Emotionally charged language to create urgency and fear."
    }
  ],
  "sourceInfo": {
    "name": "TruthSeekerDaily",
    "domain": "twitter.com",
    "reliabilityLevel": unreliable,
    "checklistAnswers": {
      "hasAuthor": false,
      "hasDate": true,
```

Nota: Los valores de **`startIndex`** *y* **`endIndex`** *en el ejemplo anterior no son necesarios para el funcionamiento del análisis de marcadores. Dentro del alcance de este TFG fueron utilizados para hacer debugging, pero tienen aplicaciones para posibles mejoras del módulo que quedaron fuera del alcance del mismo. Si los fuera a usar, es crucial que estos índices se calculen con precisión basándose en el conteo de caracteres (incluyendo espacios y puntuación) desde el inicio del campo text del snippet.*

Este proceso de curación e integración, aunque laborioso por su naturaleza manual y experta, es fundamental para asegurar que el módulo cuente con un banco de contenido pedagógico de alta calidad. La creación de un conjunto inicial de snippets siguiendo estos pasos permitió probar la funcionalidad completa del módulo, desde la presentación del snippet hasta la generación del feedback basado en la comparación con los datos expertos.

4.5 Desafíos Técnicos Durante la Implementación y Soluciones Aplicadas

Durante el ciclo de desarrollo del módulo "Stance & Source Check", surgieron diversos desafíos técnicos inherentes a la complejidad de sus funcionalidades y a la integración de diferentes tecnologías. A continuación, se describen algunos de los más significativos y las soluciones que se implementaron para superarlos.

1. Integración de Rangy.js con el Ciclo de Vida Reactivo de SvelteKit:

Rangy.js es una librería que manipula directamente el DOM para gestionar selecciones y resaltados. SvelteKit, por su parte, compila el código a JavaScript imperativo que actualiza el DOM directamente y utiliza un sofisticado sistema de reactividad para la interfaz. Esta diferencia en el enfoque de manipulación del DOM presentaba el riesgo de conflictos, especialmente en la creación y destrucción de componentes que contenían texto interactivo, o cuando Svelte actualizaba el DOM. Era crucial asegurar que Rangy.js operara sobre un DOM estable y que sus modificaciones no fueran sobrescritas o interferidas por las actualizaciones quirúrgicas de Svelte.

Proceso de Resolución y Solución Aplicada:

- **Encapsulación en un Servicio**: Toda la lógica de interacción con Rangy.js se centralizó en el **`highlightingService.ts`**. Este servicio se diseñó para ser el único responsable de inicializar Rangy, crear los **`ClassApplier`**, manejar los eventos de selección del DOM y aplicar/eliminar los resaltados.

- **Gestión del Ciclo de Vida**: La inicialización de Rangy.js y la configuración de los listeners de eventos (`mouseup`, `touchend` en el `containerElement`) se realizaron dentro del hook **`onMount`** del componente Svelte que contenía el texto del snippet. De manera simétrica, en el hook **`onDestroy`**, se llamó al método **`highlightingService.destroy()`** para limpiar los listeners y cualquier estado residual de Rangy, previniendo fugas de memoria o comportamientos anómalos al navegar fuera del componente.

- **Carga Dinámica y Sincronización**: Se implementó la carga dinámica de los scripts de Rangy.js (**`rangy-core.js`**, **`rangy-classapplier.js`**, **`rangy-serializer.js`**) dentro del **`highlightingService`**, asegurando que la librería y sus módulos estuvieran completamente cargados e inicializados (**`rangy.init()`**) antes de intentar utilizarlos.

- **Coordinación con Stores de Svelte**: Las acciones que modificaban el DOM a través de Rangy se coordinaron con actualizaciones al `highlightingStore.ts`. Por ejemplo, después de que `highlightingService.applyHighlight()` creara un nuevo resaltado, se generaba un objeto `UserSelection` que luego se añadía al store mediante `highlightingActions.addSelection()`. Esto mantenía el estado de la aplicación Svelte sincronizado con el estado visual del DOM.

- **Uso Estratégico de setTimeout para la Sincronización**: Un desafío particular surgió al procesar la finalización de una selección de texto por parte del usuario (`mouseup` o `touchend`). Ocasionalmente, SvelteKit podía aún estar finalizando sus propias actualizaciones del DOM (por ejemplo, si la selección de texto desencadenaba algún cambio de estado en Svelte que a su vez modificaba el DOM donde se encontraba el texto). Si Rangy.js intentaba obtener la selección (`window.getSelection()`) o serializarla (`this.rangy.serializeSelection()`) inmediatamente en este estado de transición del DOM, podía operar sobre una representación del DOM inestable o incompleta, llevando a errores o serializaciones incorrectas.

  Para mitigar esto, se introdujo un breve retardo utilizando `setTimeout`, en nuestro caso 10 milisegundos, antes de que `highlightingService` procesara la selección final del usuario (ver la implementación de `handleTextSelection` y `processSelection` en la sección 4.3.3). Este pequeño retardo generalmente proporciona tiempo suficiente para que SvelteKit complete sus actualizaciones del DOM pendientes, permitiendo a Rangy.js operar sobre una estructura DOM más estable. Esta solución fue validada mediante pruebas manuales exhaustivas en diversos escenarios de interacción, demostrando ser efectiva para la gran mayoría de los casos. Si bien el uso de `setTimeout` para sincronización no es una garantía absoluta contra condiciones de carrera en todos los escenarios teóricamente posibles, en la práctica del módulo demostró ser una solución pragmática y funcional que equilibra la reactividad con la correcta operación de Rangy.js.

Se reconoce que, para una robustez aún mayor en sistemas más complejos o con una carga de actualizaciones del DOM muy alta, se podrían explorar alternativas como el uso de `requestAnimationFrame` o mecanismos de cola de tareas más sofisticados, aunque para el alcance de este TFG, `setTimeout` resultó ser una solución adecuada y suficiente.

2. Precisión en la Serialización y Deserialización de Selecciones de Texto Complejas:

Uno de los requisitos clave era persistir las selecciones del usuario para su evaluación y posible revisión futura. Rangy.js ofrece un módulo de serialización (**`rangy-serializer.js`**) para convertir un objeto de selección en una cadena de

texto y viceversa. Sin embargo, durante la implementación se identificaron dos desafíos interrelacionados: la robustez de la serialización/deserialización con snippets complejos y, más críticamente, la **sensibilidad de los checksums de Rangy ante cambios estructurales del DOM**.

Naturaleza del Desafío Principal:

El desafío más significativo surgió del hecho de que Rangy.js genera checksums basados en la estructura específica del DOM al momento de la serialización. En el contexto del flujo de cuatro pasos del módulo, cuando el usuario navega entre `TextAnalysisStep`, `SourceChecklistStep` y regresa al análisis de texto, SvelteKit puede recrear componentes, resultando en un DOM con contenido idéntico pero checksums internos diferentes. Esto manifestaba el error:

```
Error: checksums of serialized range root node (f64b897) and target
root node (9ed5aea4) do not match
```

Este problema es particularmente desafiante porque:

- Las selecciones pueden cruzar múltiples nodos de texto y elementos HTML
- La estructura DOM puede cambiar sutilmente entre renderizados de SvelteKit
- Una deserialización incorrecta podría llevar a resaltados en posiciones incorrectas o pérdida completa de las selecciones
- El error rompía completamente la funcionalidad de persistencia de selecciones

Proceso de Resolución y Solución Aplicada:

- **Análisis de Root Cause**: Se identificó que el problema no residía en la lógica de la aplicación sino en la sensibilidad inherente de Rangy a cambios estructurales del DOM, incluso cuando el contenido textual permanecía idéntico.
- **Diseño de Estrategia Dual Robusta**: Se implementó una arquitectura de dos niveles que mantiene la precisión de Rangy cuando es posible, pero proporciona un fallback inteligente.

Se mantiene el uso completo de `rangy.serializeSelection()` y `rangy.deserializeSelection()` como estrategia primaria, preservando toda la sofisticación técnica del diseño original.

Estrategia de Fallback - Reconstrucción Contextual: Cuando la deserialización de Rangy falla, se activa un algoritmo de fallback que utiliza:

- **Información contextual enriquecida**: Los objetos `UserSelection` se ampliaron para incluir `context.before`, `context.after` y `context.paragraphIndex`

- **Algoritmo de desambiguación**: Para casos donde el mismo texto aparece múltiples veces, se implementó un sistema de scoring que evalúa la similitud del contexto circundante

- **Recreación manual de highlights**: Generación de elementos `<mark>` con atributos correctos que replican exactamente el comportamiento de Rangy

3. Gestión de la Complejidad de los `expertMarkers` en AdminJS:

El modelo `FakeNewsSnippet` requiere la entrada de `expertMarkers`, que es un array de objetos, cada uno con múltiples campos. La interfaz por defecto de AdminJS para campos de tipo `Mixed` puede no ser la más intuitiva para una entrada de datos tan estructurada y propensa a errores si se hace manualmente campo por campo para cada marcador.

Proceso de Resolución y Solución Aplicada:

- **Campo Virtual para JSON**: Como se describió en 4.2.2, se implementó un campo virtual `expertMarkersJson` de tipo `textarea` en la configuración de AdminJS para el recurso `FakeNewsSnippet`. Esto permitió a los curadores de contenido pegar directamente un array JSON completo de `expertMarkers`, lo cual es más rápido y menos propenso a errores para quienes están familiarizados con JSON.

- **Hooks de Transformación en AdminJS**:
    - En el hook `before` de las acciones `new` y `edit`, se añadió lógica para parsear el contenido del campo `expertMarkersJson`. Si el JSON era válido, se asignaba al campo real `expertMarkers` del payload y se eliminaba `expertMarkersJson` para que no intentara guardarse en la base de datos. Si el JSON era inválido, se mantenía el contenido en `expertMarkersJson` para que el usuario pudiera corregirlo.
    - En el hook `after` de las acciones `edit` y `show`, se añadió lógica para formatear el objeto `expertMarkers` y `sourceInfo` recuperado de la base de datos como una cadena JSON indentada y legible. Esta cadena se asignaba tanto al campo `expertMarkers` para su visualización en AdminJS como al campo virtual `expertMarkersJson`, facilitando la revisión y la copia para futuras ediciones.

- **Instrucciones Claras**: Se proporcionaron comentarios y placeholders detallados en la configuración de AdminJS para el campo expertMarkersJson (ver admin.config.js), explicando el formato JSON esperado, los tipos de marcadores válidos y cómo determinar los índices, para guiar al curador de contenido.

4. Flexibilidad en la Comparación de Selecciones de Usuario con Marcadores Expertos:

El cálculo de `selectionsAccuracy` en el backend (`UserFakeNewsResponse.methods.calculateSelectionsAccuracy`) requería comparar las selecciones del usuario con los expertMarkers. Una comparación estrictamente literal del texto seleccionado podría ser demasiado punitiva, ya que los usuarios podrían seleccionar fragmentos ligeramente más largos o cortos que el marcador experto, o con pequeñas variaciones, aun identificando correctamente el concepto.

Proceso de Resolución y Solución Aplicada:

- **Normalización del Texto**: Antes de cualquier comparación, tanto el texto seleccionado por el usuario como el texto del marcador experto se normalizaron convirtiéndolos a minúsculas y eliminando espacios en blanco al inicio y al final (`toLowerCase().trim()`).

- **Lógica de Coincidencia Flexible**: En lugar de una igualdad exacta, la lógica de comparación implementada en `calculateSelectionsAccuracy` (ver fragmento en 4.2.1) se diseñó para considerar una coincidencia si:

    1. El `markerType` asignado por el usuario coincidía con el del experto.
    2. Y, el texto normalizado del usuario era igual al texto normalizado del experto, O el texto del usuario contenía el texto del experto, O el texto del experto contenía el texto del usuario. Esta lógica de inclusión permitía un grado de flexibilidad, reconociendo selecciones que capturaban la esencia del marcador experto aunque no fueran delimitadas con precisión milimétrica.

- **No Penalización por Selecciones Adicionales (Pero sí por Omisiones)**: El cálculo de precisión se basó en cuántos de los `expertMarkers` fueron identificados correctamente por el usuario (`matches / totalExpertMarkers`). Esto significa que si un usuario identificaba correctamente todos los marcadores expertos pero además hacía selecciones adicionales incorrectas, su precisión en esta métrica no se veía directamente penalizada por esas selecciones extra, aunque sí podría afectar su aprendizaje general. La penalización principal venía de omitir marcadores expertos.

Estos desafíos y sus soluciones reflejan el proceso iterativo de desarrollo y la necesidad de adaptar los diseños iniciales a las realidades técnicas encontradas durante la implementación, buscando siempre un equilibrio entre la funcionalidad deseada, la robustez del sistema y la viabilidad dentro del alcance del proyecto.

4.6 Estrategia de Pruebas del Software y Validación

La validación de la funcionalidad y robustez del módulo "Stance & Source Check" fue un proceso continuo a lo largo del ciclo de desarrollo. Si bien no se implementó una suite completa de pruebas automatizadas debido al cambio de enfoque frecuente desde el plan de trabajo hasta la memoria se seguimiento, y desde entonces hasta esta última memoria, lo que conllevo restricciones temporales inesperados, se aplicó una estrategia de pruebas manuales

sistemáticas y se siguieron buenas prácticas para asegurar la calidad del software entregado.

4.6.1 Pruebas Manuales Funcionales

Durante cada iteración de desarrollo, se llevaron a cabo pruebas funcionales manuales exhaustivas para verificar que los componentes individuales y el sistema integrado cumplieran con los requisitos definidos. Estas pruebas se centraron en:

- **Flujo de Usuario Completo:** Se probó repetidamente el flujo de cuatro pasos (`InitialAssessmentStep`, `TextAnalysisStep`, `SourceChecklistStep`, `FeedbackDisplay`), asegurando que la transición entre pasos fuera lógica, que las validaciones funcionaran correctamente y que los datos se transmitieran y acumularan adecuadamente para el envío final.

- **Funcionalidad de Highlighting (Interacción con Rangy.js):** Se realizaron pruebas intensivas sobre la selección de texto, la aplicación de los siete tipos de marcadores, la correcta visualización de los resaltados (colores, atributos data-), la modificación y eliminación de selecciones, y la persistencia de estas a través de la serialización y deserialización (incluyendo la lógica de fallback contextual). Se verificó en múltiples navegadores (Chrome, Firefox, Edge, Safari, Opera, como se indica en RNF3).

- **Interacción con la API Backend:** Desde el frontend, se probó la comunicación con todos los endpoints de la API:

    - **`GET /api/fake-news/snippet/:userId`**: Verificación de la correcta recepción de snippets, la lógica de no repetición y el manejo del caso en que no hay más snippets disponibles.

    - **`POST /api/fake-news/response`**: Envío de análisis completos con diferentes combinaciones de datos (varios `userSelections`, diferentes respuestas en `sourceAnalysis` y `initialAssessment`), verificando la correcta recepción y procesamiento en el backend, el cálculo de métricas (`selectionsAccuracy`, `sourceAccuracy`, `overallScore`) y la generación del objeto de feedback.

    - **`GET /api/fake-news/history/:userId`**: Comprobación de la recuperación del historial de análisis del usuario.
    Se utilizaron las herramientas de desarrollador del navegador para inspeccionar las peticiones y respuestas, así como Postman para pruebas aisladas de la API.

- **Gestión de Estados del Frontend (Stores Svelte):** Se verificó que highlightingStore.ts y authStore (del sistema DeStance) gestionaran correctamente el estado de la aplicación, reflejando los cambios de forma reactiva en la UI.

- **Integración con DeStance MOOC:** Se probó la correcta autenticación, la navegación desde y hacia el dashboard del MOOC, y la coherencia visual con la plataforma existente.

- **Manejo de Errores:** Se simularon condiciones de error (p.ej., API no disponible, datos incorrectos) para verificar que el frontend mostrara mensajes informativos al usuario y que el backend respondiera con los códigos de estado HTTP adecuados.

Para cada funcionalidad clave, se diseñaron casos de prueba implícitos que cubrían el "camino feliz" (*happy path*) y algunas variaciones o entradas incorrectas básicas. Los resultados esperados se basaban en los requisitos funcionales y el diseño del módulo. Los defectos identificados se registraron y se priorizó su corrección antes de continuar con nuevas funcionalidades.

4.6.2 Validación de Componentes del Backend

La lógica de los controladores del backend (`fakeNewsController`) y los métodos de los modelos Mongoose (`FakeNewsSnippet`, `UserFakeNewsResponse`), como el cálculo de precisión o la generación de feedback, se validaron indirectamente a través de las pruebas de los endpoints API y mediante la inspección de los datos almacenados en MongoDB y las respuestas generadas. Se verificó la correcta persistencia de los `expertMarkers`, `sourceInfo`, `initialAssessment`, `userSelections`, `sourceAnalysis` y las métricas de performance.

4.6.3 Justificación de la Ausencia de Pruebas Automatizadas y Plan Futuro

Dada la naturaleza y el cronograma de un Trabajo de Fin de Grado, el desarrollo de una suite exhaustiva de pruebas automatizadas (unitarias, de integración y End-to-End) no fue factible. El esfuerzo se priorizó en la implementación de la funcionalidad principal y la robustez a través de pruebas manuales iterativas.

Sin embargo, se reconoce plenamente la importancia de las pruebas automatizadas para la mantenibilidad a largo plazo, la detección temprana de regresiones y la facilitación de refactorizaciones futuras. Como línea de trabajo futuro, se propone el diseño e implementación de:

- **Pruebas Unitarias:** Utilizando frameworks como Vitest (integrado en SvelteKit) o Jest, para componentes Svelte individuales, funciones de los servicios (`fakeNewsService.ts`, `highlightingService.ts`) y lógica de los stores. Para el backend, se probarían unitariamente los métodos de los controladores y modelos.
    - **Ejemplo (conceptual) para `highlightingService`**: Un test podría verificar que `applyHighlight` añade correctamente los atributos data-marker-type y data-highlight-id a un elemento <mark>.
    - **Ejemplo (conceptual) para `calculateSelectionsAccuracy`**: Un test podría pasar un conjunto de selecciones de usuario y marcadores expertos conocidos y verificar que la precisión calculada sea la esperada.

- **Pruebas de Integración:** Para el backend, estas pruebas verificarían la interacción entre los controladores, los modelos y la base de datos (utilizando una base de datos de prueba). Para el frontend, se probaría la interacción entre componentes que se comunican o dependen de stores.

- **Pruebas End-to-End (E2E):** Utilizando Playwright ya que ya es una dependencia en el código base, aunque todavía no se ha usado, se

simularían flujos completos de usuario a través de la interfaz, desde la carga de un snippet y la realización del análisis hasta la visualización del feedback.

La implementación de estas pruebas aumentaría significativamente la confianza en la estabilidad del código y reduciría el esfuerzo de prueba manual en futuras evoluciones del módulo.

5. Conclusiones y Líneas Futuras

Este capítulo final presenta una síntesis reflexiva de los resultados y experiencias acumuladas durante el desarrollo del presente Trabajo de Fin de Grado. El proyecto se ha centrado en la concepción, diseño e implementación del módulo "Stance & Source Check" para la plataforma DeStance, con el objetivo de optimizar la experiencia de usuario en la evaluación de competencias interculturales y, crucialmente, fomentar la alfabetización mediática crítica. A continuación, se evaluará el cumplimiento de los objetivos, se reconocerán las limitaciones del trabajo y se delinearán posibles vías de evolución futura para el módulo.

5.1 Conclusiones Principales y Cumplimiento de Objetivos

El desarrollo de este proyecto ha culminado con la entrega de una solución software plenamente funcional e integrada en la plataforma DeStance, cumpliendo así con el objetivo general de este TFG. Este logro se sustenta en la consecución satisfactoria de los objetivos específicos planteados al inicio del proyecto, los cuales se detallan a continuación:

1. **Diseñar la arquitectura software detallada del módulo:** Este objetivo se **cumplió plenamente**. Se diseñó y documentó una arquitectura full-stack robusta, incluyendo modelos de datos avanzados para MongoDB (`FakeNewsSnippet` y `UserFakeNewsResponse` con sus respectivos sub-esquemas como `initialAssessmentSchema` y `expertMarkerSchema`), una API RESTful granular y una estrategia clara para la integración del módulo como 'Course 4' en el MOOC existente, sentando así las bases técnicas del proyecto.

2. **Implementar los componentes del backend (Node.js/Express/Mongoose):** Este objetivo se **cumplió satisfactoriamente**. Se desarrollaron los modelos Mongoose (Sección 4.2.1), los controladores en Node.js/Express (`fakeNewsController`) que encapsulan la lógica de negocio esencial, como la selección de snippets, el cálculo de precisión de las selecciones del usuario (`calculateSelectionsAccuracy`, `calculateSourceAccuracy`), la ponderación del `overallScore` y la generación de feedback personalizado, y las rutas API necesarias para la comunicación con el frontend (Sección 4.2.2).

3. **Desarrollar los componentes interactivos del frontend (SvelteKit/TypeScript), implementando la funcionalidad de selección y resaltado de texto con Rangy.js y la gestión del estado de la interfaz:** Este objetivo se **alcanzó con éxito**. Utilizando SvelteKit, se implementaron los componentes que conforman la interfaz de usuario (Sección 4.3.2), abarcando el flujo interactivo de cuatro fases secuenciales (`InitialAssessmentStep`, `TextAnalysisStep`, `SourceChecklistStep`, `FeedbackDisplay`). Un hito particularmente relevante fue la integración y encapsulación de la librería JavaScript Rangy.js dentro del `highlightingService.ts` (Sección 4.3.3 y 4.5), permitiendo implementar la compleja funcionalidad de selección y resaltado de texto, superando las limitaciones de las APIs nativas del

navegador y asegurando la persistencia y correcta visualización de las selecciones mediante una estrategia dual de deserialización. La gestión del estado de la aplicación se realizó mediante stores reactivos de Svelte (`highlightingStore.ts`, Sección 4.3.1), y se garantizó una comunicación fluida con los servicios del backend.

4. **Integrar el módulo en la plataforma DeStance, asegurando la coherencia visual (UI/UX) y funcional:** Este objetivo se **cumplió exitosamente**. El nuevo módulo, denominado 'Course 4', se adaptó al sistema de autenticación (`checkAuth`) y a los patrones de navegación de DeStance, utilizando los estilos visuales existentes (TailwindCSS) y ofreciendo una experiencia de usuario unificada. Se implementó la integración con el dashboard del MOOC y el course4Service.ts para la adaptación de datos (Sección 3.2.4 y 4.3.4). Además, se desarrolló un panel de administración dedicado utilizando AdminJS para facilitar la gestión y curación del contenido específico del módulo (FakeNewsSnippet) y la visualización de respuestas (UserFakeNewsResponse) (Sección 4.2.2 y 3.4.3).

5. **Aplicar un plan de pruebas software (unitarias, integración, E2E) para validar la funcionalidad y robustez del módulo:** Este objetivo se **cumplió de forma parcial**. Aunque no se desarrolló una suite de pruebas automatizadas exhaustivas debido a las limitaciones de tiempo, la validación del módulo se abordó de manera continua a lo largo del ciclo de desarrollo iterativo e incremental. Se realizaron pruebas funcionales manuales sistemáticas de los componentes individuales, de los flujos de interacción principales y de los endpoints API, así como la verificación de la correcta integración entre las distintas partes del sistema, asegurando la robustez y funcionalidad del entregable (detallado en Sección 4.6). La definición de una estrategia completa de pruebas automatizadas se ha planteado como una línea de trabajo futuro crucial (Sección 5.3).

En conjunto, la arquitectura modular diseñada, la eficiente comunicación vía API RESTful y los modelos de datos cuidadosamente estructurados han permitido crear una herramienta que no solo es funcional sino que también responde a los exigentes requisitos pedagógicos del proyecto RACISMMAFF. El módulo 'Stance & Source Check' representa, por tanto, una contribución tangible y significativa, donde se han aplicado rigurosamente los conocimientos adquiridos durante el Grado para resolver desafíos de ingeniería de software en un contexto educativo y social relevante."

5.2 Limitaciones del Trabajo

A pesar de los resultados satisfactorios obtenidos, es fundamental acometer un análisis honesto de las limitaciones inherentes al desarrollo de este proyecto, circunscrito por los recursos y el marco temporal propios de un Trabajo de Fin de Grado.

- **Alcance de las funcionalidades implementadas:** Si bien el módulo cumple sus objetivos primarios, existe un vasto potencial para la incorporación de funcionalidades más avanzadas. Aspectos como la personalización adaptativa del contenido según el progreso del estudiante, o la introducción de elementos de gamificación más

elaborados para incrementar la motivación, representan áreas de expansión natural que excedían las posibilidades del presente trabajo.

- **Dependencia de librerías externas (Rangy.js):** Aunque Rangy.js ha demostrado ser una solución madura y eficaz, se ha observado que su desarrollo principal no mantiene la misma actividad que en el pasado. Esta situación podría, a largo plazo, suponer un riesgo si surgieran incompatibilidades con futuras actualizaciones de los navegadores que no fuesen atendidas por la comunidad de la librería.

- **Escalabilidad y rendimiento bajo carga extrema:** Aunque se han seguido buenas prácticas de desarrollo y se ha diseñado una arquitectura que busca la eficiencia, el proyecto no ha tenido como foco principal la optimización para un número masivo de usuarios concurrentes. Serían necesarias pruebas de estrés específicas y, posiblemente, optimizaciones adicionales para garantizar un rendimiento óptimo en escenarios de muy alta demanda.

- **Proceso de curación de contenido:** La efectividad pedagógica del módulo está intrínsecamente ligada a la calidad y diversidad de los snippets. El sistema actual depende de un proceso de curación manual realizado por expertos, lo cual, si bien garantiza una alta calidad, podría limitar la velocidad y la escala con la que se expande el banco de contenidos.

- **Validación y pruebas del software (Automatizadas):** En relación con la validación y las pruebas del software, como se detalló en la sección 4.6, aunque se realizaron pruebas funcionales manuales continuas, la implementación de un conjunto completo y exhaustivo de pruebas automatizadas (unitarias, de integración y E2E) quedó fuera del alcance de este TFG. Dicha batería de pruebas sería altamente beneficiosa para la mantenibilidad y la detección temprana de regresiones.

- **Accesibilidad (a11y):** Si bien se han considerado principios básicos, la consecución de una conformidad plena con las directrices WCAG, especialmente para las interacciones más complejas como el resaltado de texto gestionado mediante teclado, requeriría un esfuerzo adicional de evaluación y desarrollo específico*)*.

Estas limitaciones no sirven para desmerecer el trabajo realizado, sino deben entenderse como un reflejo de la complejidad del proyecto y como puntos de partida valiosos para futuras mejoras.

5.3 Propuestas de Mejoras y Líneas de Evolución del Módulo

Este módulo, en su estado actual, se concibe no como un producto finalizado e inmutable, sino como una base sólida y prometedora sobre la cual se pueden edificar futuras mejoras y extensiones funcionales. La experiencia adquirida durante su desarrollo y las limitaciones identificadas (detalladas en la Sección 5.2) permiten delinear diversas líneas de trabajo futuro que podrían enriquecer significativamente la herramienta:

1. **Implementación de un Plan de Pruebas Automatizadas Exhaustivo:** Como principal línea de trabajo técnico, se propone el desarrollo de una suite completa de pruebas automatizadas (unitarias, de integración y E2E) tal como se esbozó en la sección 4.6.3. Esto incluiría la configuración de un entorno de pruebas, la selección de herramientas adecuadas (e.g., Vitest, Playwright) y la creación de casos de prueba para las funciones críticas del backend y los componentes del frontend. Este paso es fundamental para garantizar la mantenibilidad, facilitar refactorizaciones y detectar regresiones de forma temprana.

2. **Optimización del Rendimiento y Escalabilidad:**

    - Realizar pruebas de carga formales para identificar cuellos de botella en la API y en las consultas a MongoDB bajo alta concurrencia.

    - Basado en los resultados, aplicar optimizaciones adicionales a las consultas de base de datos (más allá de los índices actuales) y, si fuera necesario, a la lógica de los controladores.

    - Monitorizar el estado de la librería Rangy.js y, en caso de problemas de compatibilidad o rendimiento a largo plazo, explorar activamente alternativas o contribuciones a la librería.

3. **Incorporación de Funcionalidades Pedagógicas Avanzadas:**

    - **Progreso Adaptativo:** Desarrollar un sistema que ajuste la dificultad de los snippets (basándose en su complejidad o en el tipo de marcadores/sesgos que contienen, y diferenciando entre snippets adaptados específicamente para el módulo y noticas reales) según el rendimiento histórico del usuario.

    - **Selección de Contenido Dirigido:** Implementar algoritmos que puedan seleccionar snippets enfocados en áreas específicas donde el usuario haya mostrado necesitar más práctica.

    - **Gamificación Extendida:** Introducir elementos como puntos, insignias por logros, como "Experto en detectar sesgos" o "Maestro de la evaluación de fuentes", y tablas de clasificación para incrementar la motivación.

    - **Dashboard de Progreso Mejorado:** Ofrecer visualizaciones más ricas y detalladas sobre la evolución de las habilidades del usuario a lo largo del tiempo, quizás mostrando tendencias en la identificación de tipos específicos de marcadores.

4. **Mejoras en la Curación y Variedad del Contenido:**

    - Explorar la integración de herramientas de Procesamiento del Lenguaje Natural (PLN) para asistir en la identificación preliminar de marcadores en nuevos snippets, agilizando el proceso de curación experta.

    - Ampliar continuamente el banco de snippets para cubrir una mayor diversidad de temas, formatos y sutilezas en las estrategias de desinformación.

5. **Explotación Avanzada de Índices de Posición (`startIndex`, `endIndex`):**

    - Implementar la capacidad de ofrecer **feedback visual preciso**, mostrando al usuario exactamente dónde se ubican los marcadores identificados por los expertos dentro del texto del snippet superponiéndolos y así empleando los índices, ahora opcionales.

    - Desarrollar un **análisis de proximidad y superposición** para las selecciones del usuario, evaluando no solo si el tipo de marcador es correcto, sino qué tan cerca estuvo de la selección experta en términos de posición y extensión. Esto permitiría un feedback más rico, especialmente en casos de ambigüedad textual.

6. **Mejora Continua de la Accesibilidad (a11y) y Uso del Teclado:**

    - Realizar una auditoría de accesibilidad exhaustiva siguiendo las directrices WCAG.

    - Asegurar la navegación y operación completas mediante teclado, especialmente en la interacción de selección de texto con Rangy.js y el popup de marcadores, lo cual podría requerir la implementación de manejadores de eventos de teclado específicos.

    - Enriquecer el uso de atributos ARIA para mejorar la experiencia con tecnologías de asistencia.

    - Ofrecer opciones de personalización de la visualización de contraste o tamaño de fuente para atender diversas necesidades visuales.

7. **Exploración de Funcionalidades Colaborativas (Opcional a más largo plazo):**

    - Considerar la posibilidad de que los estudiantes puedan, de forma anónima, comparar sus análisis con los de otros compañeros (además del análisis experto) o participar en discusiones moderadas sobre snippets particularmente ambiguos.

La implementación progresiva de estas mejoras y la exploración de las nuevas funcionalidades propuestas no solo consolidarían al módulo "Stance & Source Check" como una herramienta educativa aún más potente, inclusiva y versátil, sino que también ampliarían su potencial para generar un impacto positivo y duradero en el desarrollo de la alfabetización mediática crítica de sus usuarios.

6. Análisis de Impacto

El desarrollo del nuevo módulo para el MOOC de DeStance no solo representa la culminación de un Trabajo de Fin de Grado en Ingeniería Informática, sino que también se proyecta con un impacto significativo en diversas esferas: personal y académica para el autor, social y educativa para sus potenciales usuarios, y una posible contribución a objetivos de desarrollo más amplios.

6.1 Impacto Personal y Académico

La realización de este TFG ha supuesto una oportunidad invaluable para la consolidación y aplicación práctica de los conocimientos y competencias adquiridos durante el Grado en Ingeniería Informática. Desde una perspectiva personal y académica, el impacto se manifiesta en varios niveles:

- **Aplicación de Conocimientos Técnicos Avanzados:** El proyecto ha requerido la utilización de un stack tecnológico moderno y complejo, incluyendo SvelteKit para el frontend reactivo, Node.js con Express para el desarrollo del backend y la API RESTful, y MongoDB como base de datos NoSQL. La necesidad de implementar una funcionalidad sofisticada de interacción con texto llevó a la investigación y adopción de la librería Rangy.js, profundizando en la manipulación avanzada del DOM y la gestión de rangos y selecciones.

- **Resolución de Desafíos de Ingeniería:** El diseño e implementación del módulo presentaron retos técnicos considerables, como la interacción avanzada con texto para el resaltado (highlighting) persistente y fiable a través de múltiples nodos del DOM, y la integración de esta funcionalidad dentro del ecosistema reactivo de SvelteKit. La arquitectura full-stack del sistema también exigió una cuidadosa planificación para asegurar la cohesión, escalabilidad y mantenibilidad.

- **Desarrollo Metodológico:** Se ha seguido un enfoque de desarrollo iterativo e incremental, adaptando principios de metodologías ágiles, lo que ha permitido una gestión de proyecto dinámica y la entrega temprana de funcionalidades para prueba y retroalimentación. Esto ha incluido fases de análisis de requisitos, diseño de arquitectura y UI/UX, implementación backend y frontend, integración y documentación.

- **Contribución a un Proyecto de Investigación:** La inserción de este TFG dentro del proyecto de investigación "Estrategias de Posicionamiento en el Discurso del Racismo y la Inmigración: Análisis y Aplicaciones en Prácticas Afectivas de Aprendizaje (RACISMMAFF)" ha permitido aplicar la ingeniería informática a la resolución de un problema social y educativo relevante, enriqueciendo la formación académica con una perspectiva interdisciplinar.

- **Ciclo de Vida Completo del Software:** El autor ha transitado por todas las fases del ciclo de vida del desarrollo de un módulo software, desde la concepción inicial y el análisis de requisitos, pasando por el diseño detallado de la arquitectura y la experiencia de usuario, hasta la implementación técnica y la planificación de su integración en una plataforma existente.

- **Profundización en un Dominio Específico:** El trabajo ha requerido una inmersión en la problemática de la desinformación, las fake news y la alfabetización mediática crítica, lo que ha ampliado la comprensión sobre el impacto de la tecnología en contextos sociales sensibles.

6.2 Impacto Social y Educativo

El módulo 'Stance & Source Check' aspira a generar un impacto positivo en el ámbito social y educativo, particularmente en el contexto de la formación universitaria y la ciudadanía digital. Aunque una evaluación formal del impacto con usuarios finales excede el alcance de este TFG, su diseño y funcionalidades permiten proyectar beneficios significativos:

- **Impacto Inmediato (A través de la Funcionalidad Desarrollada):**
    - **Lucha contra la Desinformación:** En una era marcada por la proliferación de desinformación y discursos polarizantes, especialmente en temas sensibles como la inmigración y el racismo, este módulo ofrece **inmediatamente** una herramienta proactiva y estructurada. Su objetivo es equipar a los usuarios con habilidades para navegar críticamente el entorno informativo desde el primer uso.
    - **Fomento de la Alfabetización Mediática Crítica:** El proyecto responde **directamente** a la necesidad imperativa de mejorar la alfabetización mediática. Más allá de la simple detección de noticias falsas, el módulo se enfoca en enseñar a analizar y evaluar la información, identificar sesgos, comprender el posicionamiento discursivo (stance) y valorar la fiabilidad de las fuentes. Los usuarios pueden aplicar estos aprendizajes de forma práctica desde su primera interacción.
    - **Herramienta Pedagógica Innovadora:** Integrado en la plataforma DeStance, el módulo 'Stance & Source Check' se constituye como una herramienta pedagógica interactiva. A través de la interacción guiada con snippets, los estudiantes **pueden comenzar a practicar y desarrollar de forma aplicada su capacidad de análisis crítico inmediatamente.**
    - **Empoderamiento del Usuario:** Al facilitar el desarrollo de competencias para identificar marcadores lingüísticos de posicionamiento y evaluar fuentes, el módulo busca empoderar a los usuarios para que se conviertan en consumidores de información más conscientes, reflexivos y menos susceptibles a la manipulación desde el inicio de su aprendizaje con la herramienta.
    - **Mejora de una Plataforma Educativa Existente:** La incorporación de este módulo en DeStance **enriquece desde ya** la oferta formativa de la plataforma DeStance, que tiene como finalidad principal el desarrollo y la evaluación de competencias interculturales. El nuevo módulo añade una dimensión crucial relacionada con la competencia digital e informacional.

- **Aplicación Práctica de la Investigación:** El módulo traduce los objetivos del proyecto RACISMMAFF en una herramienta tangible, aplicando conceptos lingüísticos y pedagógicos en una solución software funcional orientada al aprendizaje afectivo y crítico, **disponible para su uso potencial en el marco del proyecto.**

- **Impacto Potencial a Largo Plazo y Plan de Evaluación Futura:** Para cuantificar y validar el impacto educativo y social a largo plazo, se requerirían estudios longitudinales y evaluaciones con cohortes de usuarios. **Como línea de trabajo futuro, se propone un plan para un estudio piloto que podría incluir:**

  - **Participantes:** Un grupo de entre 30-50 estudiantes universitarios, preferiblemente de disciplinas relevantes o aquellos inscritos en cursos relacionados con comunicación, estudios interculturales o ciencias sociales dentro de la plataforma DeStance.

  - **Metodología:**

    - Un diseño pre-test/post-test para medir la mejora en habilidades de identificación de marcadores y evaluación de fuentes.

    - Uso continuado del módulo "Stance & Source Check" durante un periodo determinado como un semestre académico.

    - Recogida de datos cuantitativos: Puntuaciones en los análisis de snippets, tiempo empleado, precisión en la identificación de `reliabilityLevel` (comparando `initialAssessment` con `finalVerdict` y con el `expertReliabilityLevel`).

    - Recogida de datos cualitativos: Cuestionarios de percepción sobre la utilidad del módulo, la mejora de sus habilidades de pensamiento crítico, y la usabilidad de la herramienta, por ejemplo System Usability Scale (SUS), Entrevistas semiestructuradas con una submuestra de participantes para obtener insights más profundos.

  - **Métricas de Éxito:**

    - Mejora estadísticamente significativa en las puntuaciones de análisis de snippets entre el pre-test y el post-test.

    - Una alta tasa de finalización de los análisis propuestos en el módulo.

    - Feedback positivo de los usuarios sobre la utilidad percibida y la mejora de sus competencias.

- Una puntuación SUS promedio indicativa de buena usabilidad.

Se espera que el uso continuado del módulo contribuya a una ciudadanía digital más informada y crítica, capaz de contrarrestar narrativas polarizantes y de participar de forma más constructiva en el debate público, especialmente en temas sensibles. Este impacto a largo plazo se alinea con los objetivos más amplios del proyecto RACISMMAFF de influir positivamente en el discurso de estudiantes universitarios y futuros docentes.

6.3 Alineación con los Objetivos de Desarrollo Sostenible (ODS)

Aunque este apartado es opcional en la estructura de la memoria, es pertinente reflexionar sobre cómo este proyecto podría alinearse con algunos de los Objetivos de Desarrollo Sostenible (ODS) promovidos por las Naciones Unidas:

- **ODS 4: Educación de Calidad:** El proyecto contribuye directamente a este objetivo al promover una educación que va más allá de los conocimientos técnicos, fomentando habilidades esenciales para el siglo XXI como el pensamiento crítico y la alfabetización mediática. Una ciudadanía bien informada y capaz de discernir la veracidad y el sesgo de la información es fundamental para una sociedad educada.

- **ODS 10: Reducción de las Desigualdades:** La desinformación a menudo exacerba las desigualdades al estigmatizar a colectivos vulnerables, incluyendo aquellos afectados por el racismo y las narrativas negativas sobre la inmigración. Al capacitar a los usuarios para identificar y cuestionar estos discursos, el módulo puede contribuir, indirectamente, a una mayor equidad y a la reducción de la discriminación basada en información falsa o sesgada.

- **ODS 16: Paz, Justicia e Instituciones Sólidas:** Una ciudadanía crítica y bien informada es menos susceptible a la manipulación y la polarización, elementos que pueden socavar la paz social y la confianza en las instituciones. Al fomentar la capacidad de análisis y el escepticismo saludable hacia la información, el proyecto puede ayudar a construir una sociedad más resiliente frente a las narrativas que buscan generar división o desconfianza.

La implementación del módulo "Stance & Source Check", por tanto, no solo cumple con los requisitos académicos de un TFG, sino que también se posiciona como una iniciativa con potencial para generar un impacto positivo y tangible en sus usuarios y, de forma más amplia, en la sociedad.

7. Bibliografía

[1] J. R. Martin and P. R. R. White, The language of evaluation: Appraisal in English. Palgrave Macmillan, 2005. (Referencia clave para Stance/Appraisal)

[2] M. Caulfield, Web literacy for student fact-checkers. [Libro en línea], 2017. (Referencia clave para estrategias prácticas de alfabetización mediática)

[3] J. Nielsen, "10 Usability Heuristics for User Interface Design," Nielsen Norman Group, Apr. 1994, actualizado noviembre 2020. [En línea]. Disponible en: https://www.nngroup.com/articles/ten-usability-heuristics/

[4] W3C Web Accessibility Initiative (WAI). Web Content Accessibility Guidelines (WCAG) Overview. [En línea]. Disponible en: https://www.w3.org/WAI/standards-guidelines/wcag/

[5] C. Wardle and H. Derakhshan, "Information Disorder: Toward an interdisciplinary framework for research and policy making," Council of Europe report DGI(2017)09, Sep. 2017.

[6] R. Wodak, *The politics of fear: What right-wing populist discourses mean*. London: Sage, 2015.

[7] P. Bhattacharya et al., «Rethinking stance detection: A theoretically-informed research agenda for user-level inference using language models», 2025, arXiv. doi: 10.48550/ARXIV.2502.02074.

[8] «Window: getSelection() method - Web APIs | MDN». Accedido: 20 de mayo de 2025. [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/Window/getSelection

[9] T. Down, timdown/rangy. (23 de mayo de 2025). JavaScript. Accedido: 20 de mayo de 2025. [En línea]. Disponible en: https://github.com/timdown/rangy

[10] B. A. Myers, «A brief history of human-computer interaction technology», interactions, vol. 5, n.º 2, pp. 44-54, mar. 1998, doi: 10.1145/274430.274436.

[11] «10 Usability Heuristics for User Interface Design», Nielsen Norman Group. Accedido: 20 de mayo de 2025. [En línea]. Disponible en: https://www.nngroup.com/articles/ten-usability-heuristics/

[12] N. Newman, «Overview and key findings of the 2024 Digital News Report | Reuters Institute for the Study of Journalism». Accedido: 22 de mayo de 2025. [En línea]. Disponible en: https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2024/dnr-executive-summary

[13] A. Nasuto y F. Rowe, «Understanding anti-immigration sentiment spreading on Twitter», PLoS One, vol. 19, n.º 9, p. e0307917, sep. 2024, doi: 10.1371/journal.pone.0307917.

[14] «ABOUT THE PROJECT | STANCE STRATEGIES IN IMMIGRATION AND RACISM-RELATED DISCOURSE: ANALYSIS AND APPLICATIONS IN AFFECTIVE LEARNING PRACTICES». Accedido: 25 de mayo de 2025. [En línea]. Disponible en: https://www.ucm.es/racismmaff/about-the-project

[15] G. Piatti, «Vigilancia epistémica y marcación discursiva en la coconstrucción interaccional», PAL, n.º 15, p. 077, oct. 2024, doi:

10.24215/18536212e077.

[16] «Project Look Sharp». Accedido: 3 de junio de 2025. [En línea]. Disponible en: https://www.projectlooksharp.org/

[17] E. Oliveros y M. S. Serrano Moreno, «Modalidad epistémica y evidencialidad: el caso de los marcadores en la sección conclusiones del artículo de investigación», Letras, vol. 63, n.º 103, pp. 81-108, 2023.

[18] S. L. Roozenbeek J, van der Linden, "Fake news game confers psychological resistance against online misinformation," Palgrave Commun., vol. 5, art. 65, jun. 2019. doi: 10.1057/s41599-019-0279-9.

[19] S. M. J. H. Shahsavari, A. P. Holur, T. Wang, R. R. R. P. T. Tang, and M. A. Cheema, "A Critical Review of the State-of-the-Art Fake News Detection Systems," in 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, Dec. 2023, pp. 3733-3742. DOI: 10.1109/BigData59044.2023.10386363.

8. Anexos

8.1 Anexo 1: Informe de originalidad

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	DANIEL RAMON ROBERTSON
Assignment title:	Turnitin Memoria Final
Submission title:	TFG_II_MemoriaFinal_DanielRamonRobertson.pdf
File name:	33441_DANIEL_RAMON_ROBERTSON_TFG_II_MemoriaFinal_Da...
File size:	4.47M
Page count:	112
Word count:	28,792
Character count:	176,453
Submission date:	04-Jun-2025 10:40PM (UTC+0200)
Submission ID:	2692248349

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Memoria de seguimiento

Desarrollo del Módulo Interactivo 'Stance & Source Check' para el Fomento de la Alfabetización Mediática en la Plataforma DeStance

Autor: Daniel Ramón Robertson
Tutor(a): Jelena Bobkina, Elena Domínguez Romero

Madrid, junio 2025

1% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text

Top Sources

0% Internet sources
0% Publications
1% Submitted works (Student Papers)