Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

Creación de una IA optimizada para la generación de preguntas y ejercicios sobre Fundamentos de Programación

Realizado por **Marco Antonio Ortega Rojas**

✠✠✠

Universidad de Valladolid

18 de septiembre de 2025

Tutores:

Dr. Carlos Vivaracho Pascual

Dr. Alejandro Ortega Arranz

Resumen

Este Trabajo Fin de Máster aborda el diseño, desarrollo y evaluación de una plataforma orientada a la generación automática de preguntas y ejercicios vinculados con los contenidos de la asignatura de *Fundamentos de Programación*. La propuesta surge ante el problema creciente de que muchos estudiantes recurren a herramientas de inteligencia artificial externas que, en ocasiones, ofrecen respuestas incorrectas o poco adaptadas al temario. Frente a ello, se plantea una solución que permita, de manera controlada, elaborar materiales de práctica basados en los contenidos definidos por el profesorado, garantizando así una mayor fiabilidad y coherencia con los objetivos de la asignatura.

La plataforma incorpora un sistema de gestión que diferencia los roles de docente y estudiante: los docentes pueden preparar y publicar ejercicios a partir de materiales propios, mientras que los estudiantes acceden a un entorno en el que resuelven las actividades y reciben retroalimentación inmediata. Con este enfoque, se busca disminuir la carga de trabajo de los docentes en la elaboración y corrección de tareas, al mismo tiempo que se proporciona a los estudiantes un recurso adicional de práctica con retroalimentación de calidad.

En conjunto, el trabajo constituye un primer paso hacia el desarrollo de herramientas educativas que integran inteligencia artificial de forma supervisada, con el objetivo de apoyar el aprendizaje y contribuir a una experiencia más alineada con los contenidos y metodologías de la asignatura.

Palabras clave

Inteligencia artificial, modelos de lenguaje, programación, educación, generación de ejercicios, retroalimentación automática

Abstract

This Master's Thesis addresses the design, development, and evaluation of a platform aimed at the automatic generation of questions and exercises linked to the contents of the *Fundamentals of Programming* course. The proposal arises from the growing problem that many students rely on external artificial intelligence tools, which sometimes provide incorrect answers or ones not well aligned with the syllabus. To address this, the work proposes a solution that, in a controlled way, enables the preparation of practice materials based on the contents defined by the teaching staff, thus ensuring greater reliability and coherence with the course objectives.

The platform incorporates a management system that distinguishes the roles of instructors and students: instructors can prepare and publish exercises from their own materials, while students access an environment in which they solve activities and receive immediate feedback. With this approach, the aim is to reduce the workload of instructors in preparing and correcting assignments, while providing students with an additional resource for practice with high-quality feedback.

Overall, this work constitutes a first step towards the development of educational tools that integrate artificial intelligence under supervision, with the goal of supporting learning and contributing to an experience more closely aligned with the contents and methodologies of the course.

Keywords

Artificial intelligence, language models, programming, education, automatic exercise generation, automatic feedback

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación

La asignatura de *Fundamentos de Programación* afronta desde hace años un reto ampliamente documentado: grupos numerosos, perfiles heterogéneos y bajas tasas de aprobados persistentes en los primeros cursos [1, 40, 2]. En este escenario, la disponibilidad de práctica frecuente y de retroalimentación oportuna resulta decisiva: la literatura reporta efectos positivos del *feedback* específico y a tiempo sobre el aprendizaje [11].

Ahora bien, a medida que crece el volumen de entregas también lo hace la carga de corrección para los docentes, de modo que las estrategias de evaluación automatizada y el soporte a la generación de ejercicios han ido ganando importancia en la educación en programación [27, 25]. Es por eso que los modelos de lenguaje de inteligencia artificial pueden ayudar a solventar dichas cargas de trabajo del profesorado.

1.2. Modelos de lenguaje: oportunidades y límites en programación

La irrupción de los modelos de lenguaje de gran tamaño (LLM) ha abierto oportunidades para fomentar la práctica autónoma y ayudar en la redacción de documentos, pero también introduce riesgos de sesgo, alucinaciones y desalineación con los objetivos de un curso [18].

En programación, los modelos de lenguaje entrenados específicamente con código muestran una capacidad creciente para producir o transformar fragmentos funcionales (Code Llama[1], StarCoder2[2], entre otros). Con todo, evaluaciones sistemáticas —como las realizadas en torno a Codex— recuerdan que la corrección de las soluciones no puede asumirse y siempre requiere verificación [3].

[1] https://ollama.com Último acceso: 13 de septiembre de 2025

[2] https://huggingface.co/bigcode/starcoder2 Último acceso: 13 de septiembre de 2025

En un contexto introductorio, este desajuste puede amplificarse: un LLM genérico tiende a proponer soluciones plausibles, pero no siempre alineadas con el temario, especialmente cuando el curso prioriza el enfoque imperativo de Java en etapas iniciales. En consecuencia, el sistema que se presenta evita exponer soluciones completas al estudiante, priorizando el diagnóstico con alternativas bajo supervisión docente [18, 3].

1.3. Problema de contextualización y alineación con el temario

El núcleo del problema está en asegurar que los ejercicios y las preguntas prácticas que los estudiantes consultan a grandes modelos de lenguaje, como los que están detrás de ChatGPT, se ajusten a los objetivos de *Fundamentos de Programación* (Java, programación estructurada, paradigma imperativo y buenas prácticas en programación) y que el *feedback* esté contextualizado en los materiales del curso.

Una forma prometedora para lograrlo consiste en anclar la generación de respuestas a contenidos supervisados por los docentes. Actualmente, existen diferentes formas de contextualizar dichas respuestas, como el *prompt engineering*, la Recuperación Aumentada de Información (RAG) o combinaciones híbridas, con el propósito de reducir desalineaciones y mejorar la pertinencia del contenido generado [23, 12].

1.4. Objetivos del trabajo

El proyecto persigue cuatro metas complementarias:

1. Disminuir la carga de los docentes en la preparación de práctica básica y en la corrección de intentos.
2. Aumentar la disponibilidad de ejercicios alineados con el temario (Java y paradigma imperativo en etapas iniciales).
3. Proporcionar retroalimentación inmediata y pertinente a los estudiantes, preservando la supervisión docente y la posibilidad de revisión.
4. Ofrecer diagnóstico localizado que señale dónde y por qué una solución es incorrecta y proponga alternativas coherentes con el temario, evitando la entrega de la solución completa.

1.5. Solución propuesta y alcance

Se propone una plataforma con el siguiente flujo:

1. Subida de materiales por parte del docente.

2. Limpieza y síntesis para extraer conceptos y enunciados base.
3. Generación de ejercicios teóricos y prácticos a partir de esos materiales, con pautas de corrección internas y soluciones completas reservadas al uso docente.
4. Realización de intentos por los estudiantes con calificación automática y diagnóstico textual que indica dónde y por qué una respuesta es incorrecta, proponiendo alternativas alineadas con el temario.
5. Revisión y retroalimentación adicional por parte del docente, de manera optativa.

De forma predeterminada, el estudiante no recibe la solución final, sino orientación y criterios. El diseño enfatiza el ajuste al subconjunto imperativo de Java en etapas iniciales y las buenas prácticas de la asignatura, con un enfoque *local-first* para reducir dependencia de terceros y facilitar el control de datos y modelos; se emplean componentes que permiten ejecutar modelos en local e integrarlos en flujos de preprocesamiento y generación (Ollama[3] y LangChain[4]). La propuesta se apoya en prácticas consolidadas de autoevaluación en programación para articular la evaluación [27, 25].

1.6. Estructura del documento

Este TFM se organiza como sigue:

- **Capítulo 1 – Introducción:** presenta el contexto, motivación, problemas detectados y los objetivos del trabajo.
- **Capítulo 2 – Objetivos del proyecto:** define el propósito general y los objetivos específicos de la propuesta.
- **Capítulo 3 – Conceptos teóricos:** expone la literatura y el marco teórico en torno a educación en programación, retroalimentación y LLM aplicados a código.
- **Capítulo 4 – Planificación del proyecto:** describe la organización temporal, fases y entregables previstos.
- **Capítulo 5 – Técnicas y herramientas:** detalla las decisiones de diseño, metodologías empleadas y las tecnologías utilizadas.
- **Capítulo 6 – Evaluación y pruebas:** explica la metodología de evaluación, los experimentos y los resultados obtenidos.
- **Capítulo 7 – Conclusiones y líneas de trabajo futuras:** sintetiza los logros, limitaciones y propuestas de mejora.

[3] https://ollama.com Último acceso: 13 de septiembre de 2025
[4] https://python.langchain.com Último acceso: 13 de septiembre de 2025

- **Anexos:** incluyen manuales, ejemplos de prompting y documentación técnica complementaria.

Capítulo 2

Objetivos del proyecto

2.1. Propósito y alcance

Este proyecto persigue un doble propósito: **(i)** mejorar el aprendizaje de los estudiantes mediante *práctica alineada con el temario* y *retroalimentación inmediata*, y **(ii)** reducir la carga de preparación y corrección de los docentes.

2.2. Objetivo general

Diseñar, implementar y validar una plataforma que, a partir de materiales supervisados por los docentes, genere y evalúe ejercicios de programación alineados con el enfoque imperativo temprano en Java, proporcionando retroalimentación inmediata y contextualizada al estudiante y, en paralelo, optimizando el esfuerzo de preparación y corrección del profesorado.

2.3. Objetivos específicos

Estos objetivos específicos desarrollan y concretan las metas generales presentadas en la Introducción, de manera que su consecución permite dar por alcanzado el objetivo general:

1. Disminuir la carga de los docentes en la preparación y corrección de práctica básica.
2. Aumentar la disponibilidad de ejercicios alineados con el temario (Java, paradigma imperativo inicial y buenas prácticas).
3. Proporcionar a los estudiantes retroalimentación inmediata, comprensible y contextualizada, evitando la exposición directa de la solución completa.

4. Ofrecer diagnósticos localizados que indiquen dónde y por qué falla una solución, proponiendo alternativas coherentes con el temario.
5. Analizar y seleccionar modelos de lenguaje y técnicas de recuperación aumentada (*RAG*) adecuados al contexto de la asignatura.
6. Implementar una herramienta que integre generación y evaluación de ejercicios de forma controlada, con trazabilidad y supervisión docente.
7. Evaluar la herramienta en un entorno real de aula para comprobar su eficacia pedagógica y técnica.
8. Diseñar el sistema con criterios de sencillez de uso y extensibilidad, incluyendo la posibilidad de integración en LMS institucionales.

Capítulo 3

Conceptos teóricos y estado del arte

Resumen del capítulo

Este capítulo establece la base conceptual del proyecto y sitúa la propuesta en el estado del arte. Se definen los conceptos clave de inteligencia artificial y modelos de lenguaje de gran tamaño, se revisa la evidencia sobre el uso de estas herramientas en educación —con especial foco en el aprendizaje de la programación— y se identifican sus principales limitaciones. Posteriormente, se presentan los métodos de contextualización más relevantes y se justifica la elección aplicada en este trabajo. Finalmente, se describe cómo se preparan los materiales docentes para integrarlos en el sistema y se revisan trabajos previos relacionados, lo que permite ubicar la contribución de este TFM.

3.1. Terminología: IA, IA generativa y LLM

En sentido amplio, *inteligencia artificial* (IA) designa técnicas para resolver tareas cognitivas mediante máquinas. La *IA generativa* hace referencia a modelos capaces de producir contenido nuevo (texto, imagen, código). Los *modelos de lenguaje de gran tamaño* (LLMs) son redes neuronales basadas en la arquitectura *Transformer*—introducida por Vaswani et al.—que predicen la siguiente unidad lingüística condicionada al contexto [39]. Esta capacidad de modelado contextual explica tanto su versatilidad como su dependencia de los datos que se les suministran.

3.2. LLMs en educación de programación: oportunidades y riesgos

La literatura reciente identifica oportunidades de los LLMs para apoyar práctica autónoma, escribir explicaciones o proponer variantes de ejercicios, junto con riesgos

como sesgos, alucinaciones y desalineación con los objetivos de curso, lo que exige un uso responsable y supervisado [18].

En el ámbito de la programación, existen evidencias empíricas de que los asistentes basados en LLM pueden influir en el proceso de resolución. Por ejemplo, mostrar directamente código "correcto" puede reducir la exploración y el aprendizaje activo, mientras que ofrecer diagnósticos con pistas (sin revelar la solución completa) favorece mejores resultados [19, 3]. Estas observaciones resultan especialmente relevantes en la asignatura de *Fundamentos de Programación*, donde se prioriza la programación estructurada, el paradigma imperativo y la construcción paulatina de hábitos de diseño.

3.3. Limitaciones relevantes: alucinaciones y falta de contextualización

Dos limitaciones afectan de manera crítica a un curso introductorio:

- **Alucinaciones**: el modelo puede producir salidas plausibles pero incorrectas; la literatura ha documentado su carácter sistemático en generación de lenguaje [16]. En un curso de programación, este problema puede inducir al estudiante a aceptar explicaciones o fragmentos de código erróneos como válidos.
- **Falta de contextualización curricular**: los LLMs generales tienden a proponer soluciones acordes a su preentrenamiento (por ejemplo, introducir orientación a objetos prematuramente) aunque la asignatura delimite un subconjunto imperativo de Java. Esto contradice la secuencia pedagógica establecida por el profesorado y puede confundir al alumnado.

Estas limitaciones motivan que la retroalimentación que reciba el estudiante esté anclada a materiales supervisados por el profesorado y que, por defecto, el sistema ofrezca diagnóstico y alternativas en lugar de la solución completa.

3.4. Métodos de contextualización: *prompting*, RAG y *fine-tuning*

Para alinear un LLM con un contexto docente concreto existen tres enfoques complementarios:

Plantillas y *prompting* estructurado

Consiste en definir instrucciones y ejemplos en el *prompt* (criterios, estilo de respuesta, nivel del curso). Su principal ventaja es la sencillez y el bajo coste computacional, ya que no requiere modificar los parámetros del modelo. Sin embargo, no garantiza la recuperación

de contenidos específicos del temario si no se le proporcionan explícitamente, lo que limita su eficacia en cursos con materiales cambiantes [24, 41].

En este TFM se emplea como apoyo a RAG, con el fin de reforzar el estilo y el formato de las salidas, pero no como técnica aislada.

RAG (*Retrieval-Augmented Generation*)

RAG introduce una capa de recuperación que selecciona fragmentos de los materiales del curso y los inserta en el contexto del modelo antes de generar [23]. Diversas revisiones muestran que inyectar evidencia externa mejora la pertinencia, reduce alucinaciones y permite actualizar contenidos sin reentrenar [42, 9].

En el plano educativo, se han reportado beneficios al mantener las respuestas alineadas con el temario y adaptadas al nivel de los estudiantes [12]. En este proyecto, RAG constituye la técnica principal de contextualización.

Fine-tuning y adaptación eficiente (PEFT/LoRA)

El *fine-tuning* ajusta los parámetros del modelo con datos específicos del dominio. Las técnicas de adaptación eficiente, como LoRA o QLoRA, reducen costes y memoria insertando matrices de bajo rango o aplicando cuantización [13, 7]. Aunque ofrecen un ajuste profundo, requieren corpus de calidad y un mantenimiento costoso ante cambios en el temario. Por este motivo, no se consideran adecuadas en este trabajo, donde prima la flexibilidad.

Comparativa y elección de enfoque

En la Tabla 3.1 se presenta una síntesis de las tres aproximaciones.

En este proyecto se prioriza la combinación de RAG con *prompting* estructurado: permite mantener el control curricular, deja rastro de las fuentes y evita ciclos de reentrenamiento, encajando con la naturaleza dinámica de los materiales docentes.

3.5. Representación de materiales docentes y preparación

Para contextualizar el modelo con los recursos de la asignatura se requiere una canalización que convierta los materiales en evidencia utilizable. Los pasos a seguir son los siguientes:

1. **Extracción y limpieza**: homogeneizar textos eliminando artefactos y encabezados irrelevantes, conservando enunciados y definiciones clave.

	Prompting	**RAG**	**Fine-tuning/PEFT**
Coste computacional	Bajo	Medio (indexado + búsqueda)	Medio/Alto (entrenamiento)
Actualización de contenidos	Manual en prompts	Reindexar documentos	Reentrenar/adaptar
Ajuste a temario	Limitado	Alto (evidencia explícita)	Alto si hay datos
Trazabilidad	Media	Alta (citas/fragmentos)	Baja (implícito en pesos)
Riesgo de alucinación	Medio	Bajo–medio	Medio (si datos escasos)

Tabla 3.1: Comparativa de técnicas de contextualización aplicables a un curso de programación.

2. **Fragmentación razonada**: dividir los textos en unidades coherentes que equilibren completitud y presupuesto de contexto.
3. **Indexación densa**: codificar fragmentos en *embeddings* y construir un índice para recuperación semántica [17].
4. **Inyección en el contexto**: recuperar fragmentos relevantes ante cada consulta y suministrarlos al LLM junto con instrucciones pedagógicas (nivel, estilo imperativo, no revelar solución completa).

3.6. Banco de ejercicios y retroalimentación pedagógica

La evidencia en educación sugiere que la retroalimentación específica y a tiempo mejora el aprendizaje [11]. En programación introductoria, conviene evitar mostrar la solución completa: el diagnóstico localizado con alternativas promueve mejor comprensión que la mera entrega del código final [19]. Por ello, el sistema prioriza explicar dónde y por qué una respuesta es incorrecta y sugiere reparaciones coherentes con el temario; la supervisión docente sigue siendo esencial para garantizar calidad.

3.7. Trabajos relacionados

Antes del despliegue de modelos generativos, el *autograding* en programación se apoyaba en entornos que compilaban y comparaban salidas frente a oráculos. CodeRunner es el

exponente más citado en el ecosistema Moodle [25], y revisiones recientes sistematizan sus límites (coste de autoría, feedback limitado) [27].

Con los LLM surgieron proyectos que generan cuestionarios a partir de materiales fuente:

- Generación de *quizzes* desde transcripciones de YouTube con LangChain y un modelo local (Mistral) [8].
- Creación de preguntas de opción múltiple a partir de PDF con LangChain+OpenAI [15].

En docencia universitaria, Slade et al. proponen un tutor de Psicología basado en RAG (Qdrant) [36], mientras que Chudziak y Kostka plantean una plataforma de matemáticas multiagente con GraphRAG [4].

En el ámbito hispanohablante, iniciativas como *thinkö AI* (Tekman) se centran en productividad docente y preparación de materiales [38].

Este TFM se sitúa en la intersección de estas líneas con tres decisiones técnicas:

1. Ejecución local con Ollama para reducir costes y gobernar datos.
2. Una **pipeline end-to-end** (limpieza → generación → entrega → diagnóstico → revisión) que prioriza feedback localizado sobre mostrar soluciones completas.
3. Un ciclo de evaluación con intento único, calificación automática trazable y cierre manual opcional, alineado con literatura de *feedback* efectivo [11].

Frente a los generadores de ítems y tutores RAG, la contribución de este trabajo no es solo "producir preguntas", sino integrar generación y evaluación en un flujo auditable y adaptado a la asignatura.

Capítulo 4

Planificación del proyecto

Resumen

Este capítulo describe cómo se organizó el trabajo: metodología de desarrollo y evaluación, planificación con hitos y carga en horas, gestión de riesgos y presupuesto. Para evitar redundancias, los detalles de entorno y despliegues se remiten al Capítulo 5.

4.1. Metodología de trabajo

Se adoptó un enfoque **iterativo e incremental** inspirado en prácticas ágiles (revisión frecuente con los tutores y entregables verificables por iteración). La coordinación se articuló en: **(i)** una lista priorizada de trabajo (*backlog*[1]); **(ii)** iteraciones cortas con objetivos concretos y revisión conjunta al cierre; y **(iii)** criterios de aceptación claros por entregable (resultado observable y evidencia registrada).

Para la evaluación se definieron indicadores alineados con los **objetivos del Capítulo 2** y se desarrollan en el apartado 4.4. La selección combina métricas técnicas del *backend* y apreciaciones pedagógicas fundamentadas en la literatura sobre *feedback* [11] y corrección automática en programación [27].

4.2. Plan de trabajo e hitos

La planificación se estructuró en iteraciones con entregables verificables. En la **Tabla 4.1** se presenta el *calendario inicial* y, para cada hito, su entregable. La carga total planificada es de **150 horas** (6 ECTS × 25 h/ECTS), distribuida por iteraciones.

[1] *Backlog*: listado priorizado de tareas/historias abiertas que guía el trabajo de cada iteración.

Tabla 4.1: Iteraciones, entregables y horas planificadas (calendario inicial).

Iteración / Hito	**Semanas**	**Entregable verificable**	**Horas**
I. Revisión y alcance	1–2	Estado del arte acotado a educación en programación y LLM; selección inicial de modelos locales y enfoque de contextualización (RAG/FT).	20
II. Arquitectura base	3–4	Esqueleto de API (FastAPI), servicio de inferencia (Ollama) y frontend (Laravel) con gestión de roles.	20
III. Limpieza de materiales	5–6	*Pipeline* de preprocesamiento y persistencia (material original y limpio).	25
IV. Generación de ejercicios	7–10	Flujo de generación (teórico/práctico) desde materiales; almacenamiento y consulta.	35
V. Calificación y diagnóstico	11–12	Intentos de estudiante, calificación automática y diagnóstico textual; revisión docente.	20
VI. Evaluación técnica	13–14	*Benchmark* local de modelos (TTFT y *throughput*); exportación de resultados.	20
VII. Ajustes y memoria	15–16	Correcciones, validación cruzada y cierre de documentación.	10

4.3. Entorno y despliegues (referencia)

Las decisiones de entorno y despliegue condicionaron la planificación (p. ej., disponibilidad de GPU y límites de publicación temporal). El **detalle técnico** (equipos, configuraciones y *pipeline* operativo) se documenta en el **Capítulo 5**.

4.4. Plan de evaluación

La evaluación se diseñó para comprobar el grado de cumplimiento de los **objetivos del Capítulo 2** en dos niveles complementarios:

1. **Nivel técnico (modelos en local):** latencia al primer token (TTFT) y velocidad de generación (tokens/s), junto al consumo de VRAM/RAM y estabilidad del servicio. Las pruebas se automatizaron para obtener trazas repetibles y comparables.
2. **Nivel pedagógico:** utilidad y claridad de la retroalimentación; aprovechamiento de los ejercicios tras revisión docente; y acuerdo IA–docente por tramos de calificación. Estos indicadores se apoyan en la literatura sobre *feedback* [11] y corrección automática en programación [27].

Los **resultados y el método de recogida** (instrumentos, muestras y análisis) se presentan en el **Capítulo 6**.

4.5. Riesgos y mitigación

En la **Tabla 4.2** se muestran los riesgos principales identificados y el plan de mitigación aplicado.

Tabla 4.2: Riesgos principales y mitigación.

Riesgo	**Prob./Impacto**	**Mitigación aplicada**
Limitaciones de cómputo (VRAM/CPU)	Alta/Alta	Modelos cuantizados, reducción de contexto y ejecución local en equipo con GPU; planificación de pruebas acorde a recursos.
Publicación temporal y *timeouts*	Media/Media	Sesiones de prueba acotadas; reducción de tiempos por petición; *polling* para operaciones largas; evitar exponer puertos sensibles.
Calidad pedagógica irregular de ítems	Media/Media	Generación condicionada a materiales del curso; revisión y curación docente; guías internas de corrección.
Desalineación con el temario	Media/Alta	Plantillas de *prompt* ancladas a materiales y estilo de la asignatura; validación muestreada con tutores.

4.6. Presupuesto y costes

Costes directos

Tabla 4.3: Costes directos de software/infraestructura.

Concepto	**Coste**
Dominio anual	$\sim 5\,€$
Cloudflare Tunnel (plan gratuito)	$0\,€$
Software empleado (FastAPI, Laravel, Spatie, Ollama, LangChain)	$0\,€$
Total costes directos	$\sim$ **5 €**

Coste de personal (referencia)

La carga del TFM es de **6 ECTS** (**150 h**). Tomando como referencia pública la horquilla salarial de perfiles *junior* en España (14–17 €/h, a partir de 24–30 k€/año y 1 720 h/año), el coste estimado es:

$$C \approx 14\text{–}17\,€/\text{h} \times 150\,\text{h} \;=\; 2{,}100\text{–}2{,}550\,€.$$

La suma total (costes directos + personal de referencia) queda, por tanto, en el entorno de **2,105–2,555 €**.

Capítulo 5

Técnicas y herramientas

Resumen

Este capítulo presenta los requisitos de diseño y las decisiones tecnológicas que sustentan el sistema. Se describe la arquitectura, el modelo de datos, las interfaces de usuario y los *pipelines* operativos, junto con la justificación de las tecnologías seleccionadas. El objetivo es mostrar cómo se integran los componentes para responder a los problemas de la Introducción y a los objetivos del proyecto.

5.1. Requisitos del sistema

Los requisitos funcionales y no funcionales se definieron en la fase de diseño y constituyen la base sobre la que se evaluará la validez del sistema.

Requisitos funcionales

- **Interfaz de estudiante:** acceso a exámenes generados a partir de materiales del curso, con retroalimentación inmediata en cada intento.
- **Interfaz de docente:** subida y organización de materiales, generación de ejercicios y revisión opcional de intentos.
- **Gestión de roles:** diferenciación clara entre permisos de docencia y alumnado.
- **Procesamiento automático:** generación de ejercicios a partir de materiales y calificación con *feedback*.
- **Diagnóstico localizado:** identificación de errores en las respuestas y propuesta de alternativas alineadas con el temario.
- **Revisión docente:** posibilidad de revaluación y comentarios finales sobre intentos.

Requisitos no funcionales

- **Facilidad de uso** para perfiles docente y estudiantil.
- **Ejecución en hardware accesible**, con soporte opcional de GPU.
- **Extensibilidad**, con opción de integración futura en LMS institucionales.
- **Compatibilidad multiplataforma** (Windows, Linux, macOS).

5.2. Arquitectura del sistema

La plataforma sigue una arquitectura cliente–servidor en tres capas:

1. **Interfaz y gestión (Laravel):** vistas para docente/estudiante, gestión de usuarios y materiales, y orquestación de colas de trabajo.
2. **Procesamiento de IA (FastAPI):** recibe peticiones de Laravel, construye *prompts*, invoca modelos y devuelve resultados JSON.
3. **Modelos de lenguaje (Ollama + LangChain):** limpieza de materiales y generación de ejercicios, con *Retrieval-Augmented Generation* (RAG).

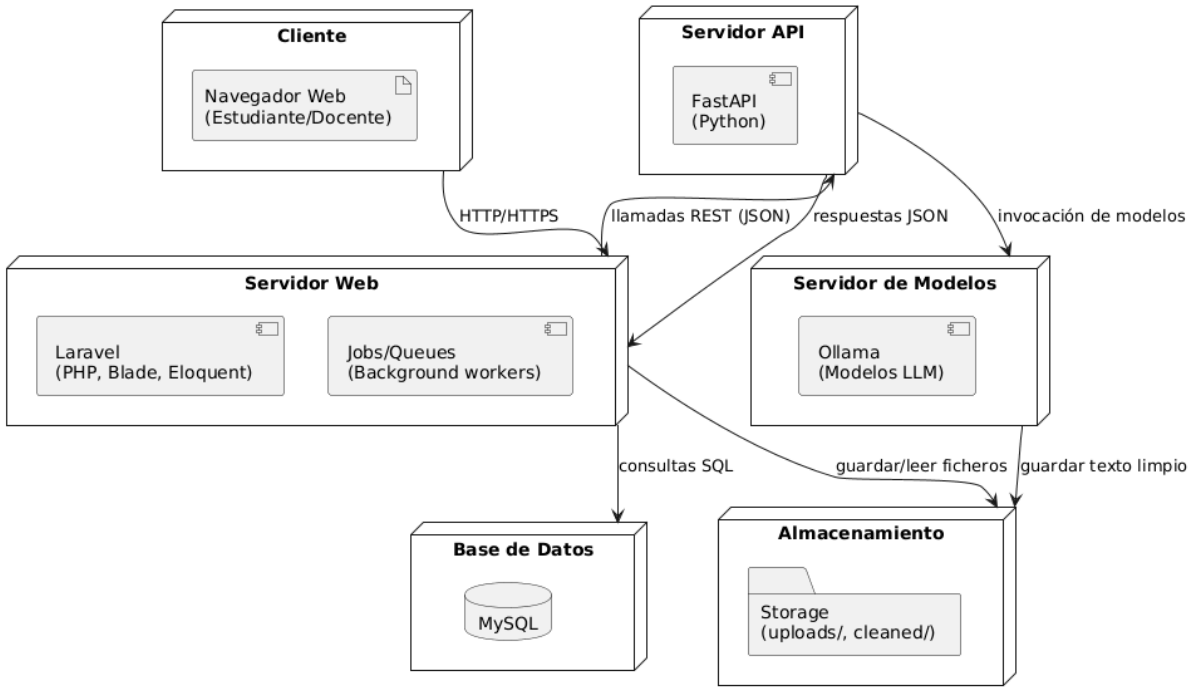

Figura 5.1: Arquitectura y despliegue lógico de la plataforma.

5.3. Pipelines operativos

La solución se organiza en tres *pipelines* principales, que aseguran trazabilidad extremo a extremo:

Ingesta y limpieza de materiales

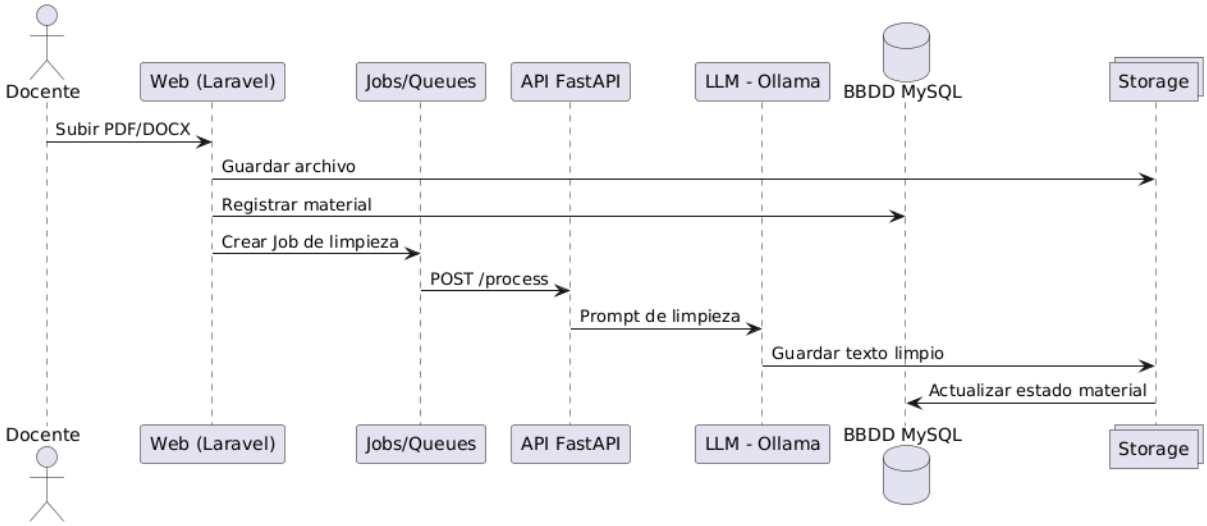

Figura 5.2: Pipeline de ingesta y limpieza de materiales.

Generación de ejercicios

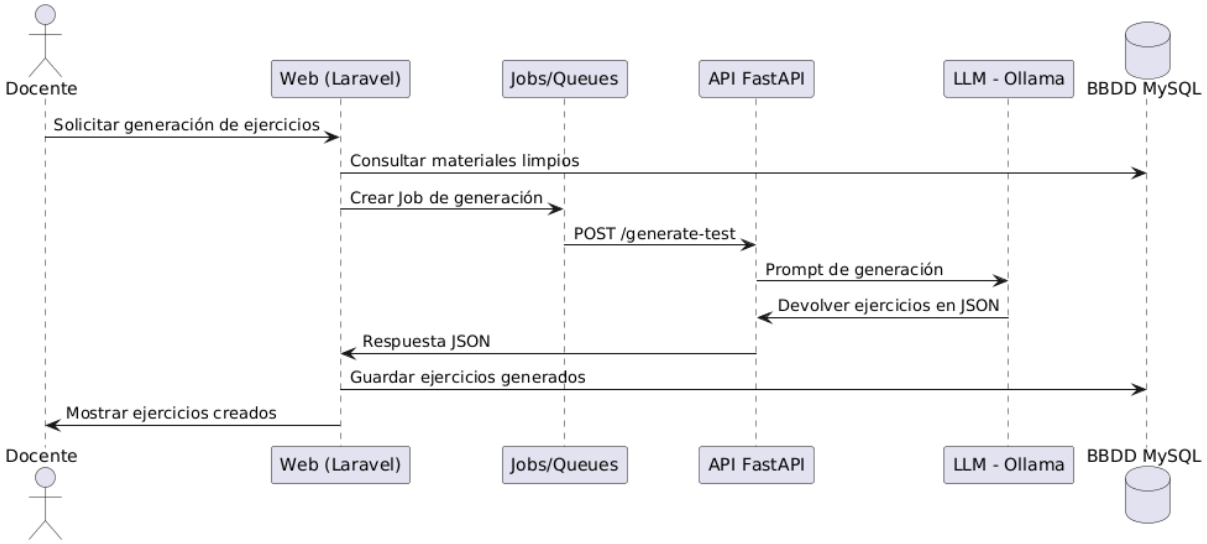

Figura 5.3: Pipeline de generación de ejercicios.

Intento de estudiante y evaluación

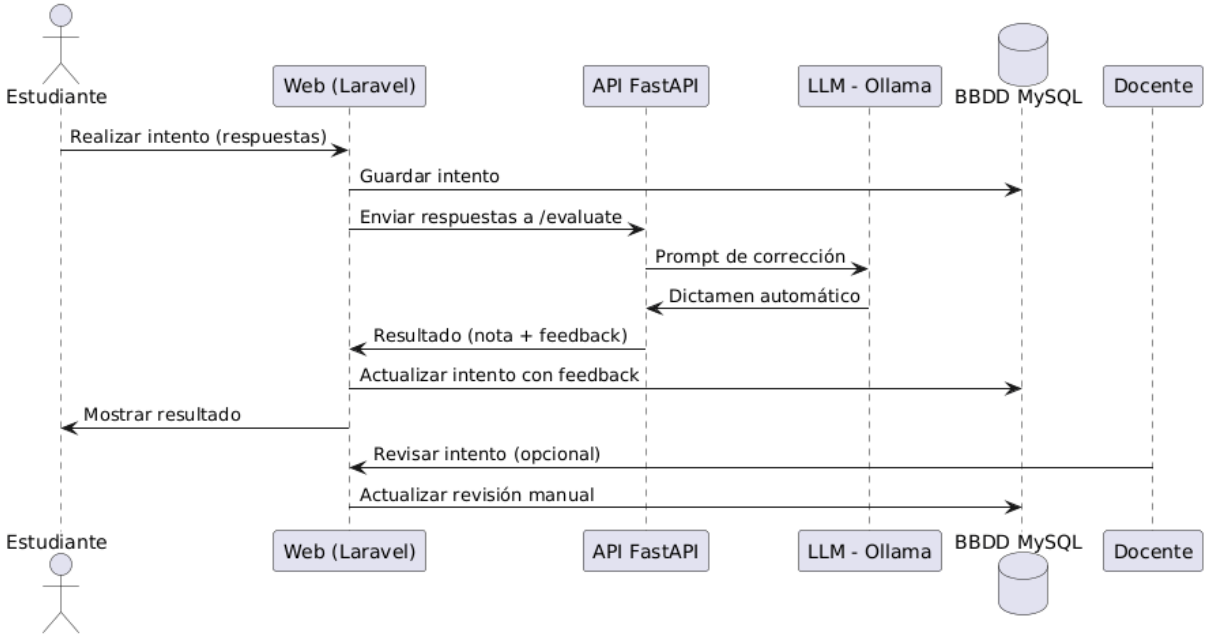

Figura 5.4: Pipeline de intento de estudiante y evaluación.

5.4. Diagrama de secuencia general

El siguiente diagrama sintetiza el intercambio entre actores y servicios a lo largo de las fases previas.

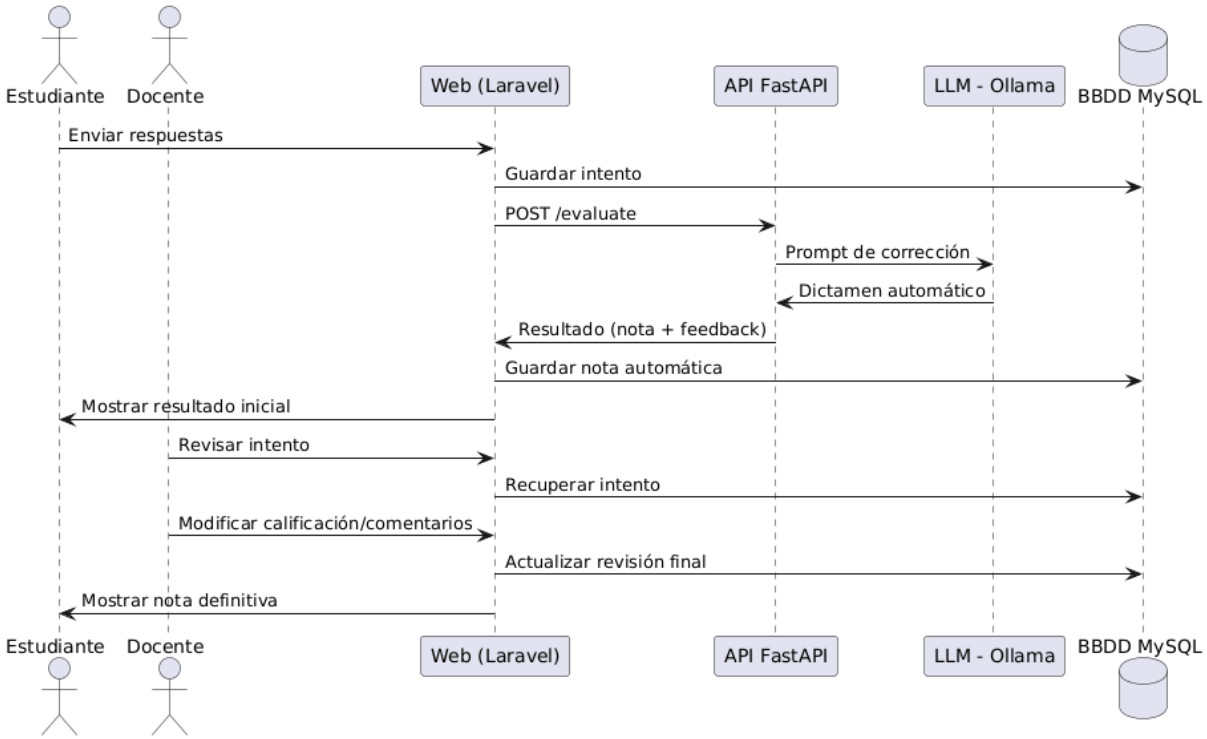

Figura 5.5: Diagrama de secuencia general del sistema.

5.5. Modelo de datos

Se emplea **MySQL** por su integración con Laravel (Eloquent), soporte amplio y despliegue sencillo en entornos académicos. Las entidades principales son: *usuarios*, *temas*, *materiales*, *exámenes/preguntas*, *intentos/respuestas* y *revisiones*.

Modelo relacional

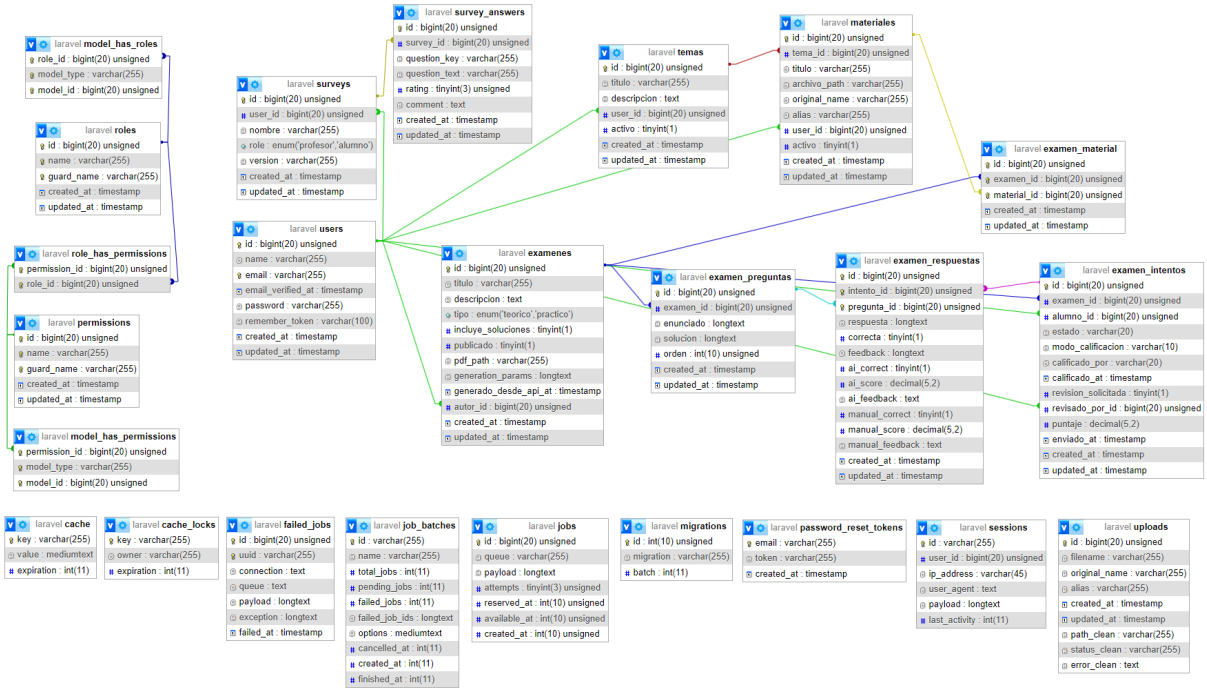

Figura 5.6: Modelo relacional del sistema.

5.6. Interfaces de usuario

Existen dos áreas diferenciadas:

- **Docente:** creación de temas, subida de materiales, generación de exámenes, revisión de intentos y encuestas.
- **Estudiante:** acceso a exámenes publicados, realización de intentos, visualización de resultados y encuestas finales.

5.7. Operación y observabilidad

La ejecución asíncrona mediante colas de Laravel desacopla tareas costosas (limpieza y generación). Se instrumentan *logs* informativos y de error en Laravel y FastAPI, y tiempos de respuesta por tramo del pipeline, con política de reintentos ante fallos transitorios. Las métricas de rendimiento (TTFT, *throughput*, uso de VRAM/RAM y estabilidad) se reportan en el Capítulo 6.

5.8. Tecnologías

La selección se realizó por facilidad de uso, adecuación académica y ejecución local.

- **Ollama:** ejecución local de LLMs modernos y control de datos.
- **FastAPI:** backend ligero en Python con validación y OpenAPI.
- **LangChain:** orquestación de *prompts* y soporte RAG.
- **Laravel + Spatie Permission:** vistas (Blade), persistencia (Eloquent) y autorización basada en roles/permisos.
- **Librerías auxiliares:** `Smalot/PdfParser`, `PhpOffice/PhpWord`, `Axios/Http` y `Storage` de Laravel.

Capítulo 6

Evaluación y pruebas

Resumen

Este capítulo comprueba en qué medida la plataforma cumple los objetivos definidos en el Capítulo 2 desde la perspectiva de sus usuarios. La evaluación se basa exclusivamente en valoraciones de *docente* y *estudiante* recogidas dentro de la propia aplicación mediante escalas tipo Likert (1–5). Los resultados muestran fortalezas en *facilidad de uso*, *flujo de trabajo*, *tiempos de generación* y *feedback*; y señalan como prioridades de mejora la *coherencia con el temario* (lado docente) y la *justicia de la evaluación* (lado estudiante).

6.1. Metodología de recogida

La plataforma integra formularios de valoración diferenciados por rol. Se analizaron **4** respuestas de **docente** y **6** de **estudiante**. Se descartaron posibles duplicados cuando todas las respuestas coincidían exactamente.[1]

6.2. Resultados

Docente

Aspectos evaluados: interfaz, flujo, tiempos (limpieza/generación), coherencia con el temario, adecuación de enunciados, utilidad del *feedback* y satisfacción global.

[1] Cuando un registro presentaba el mismo patrón de respuestas que otro, se consideró duplicado y no se incluyó en el cómputo.

Tabla 6.1: Docente: medias por aspecto (1–5), $n = 4$.

Aspecto	**Media**
Interfaz intuitiva	4.25
Flujo de trabajo claro	4.75
Tiempo de limpieza de materiales	4.00
Tiempo de generación de ejercicios	**5.00**
Coherencia con el temario	**3.25**
Adecuación de los enunciados	3.75
Utilidad del *feedback* automático	4.50
Satisfacción global (docente)	4.00

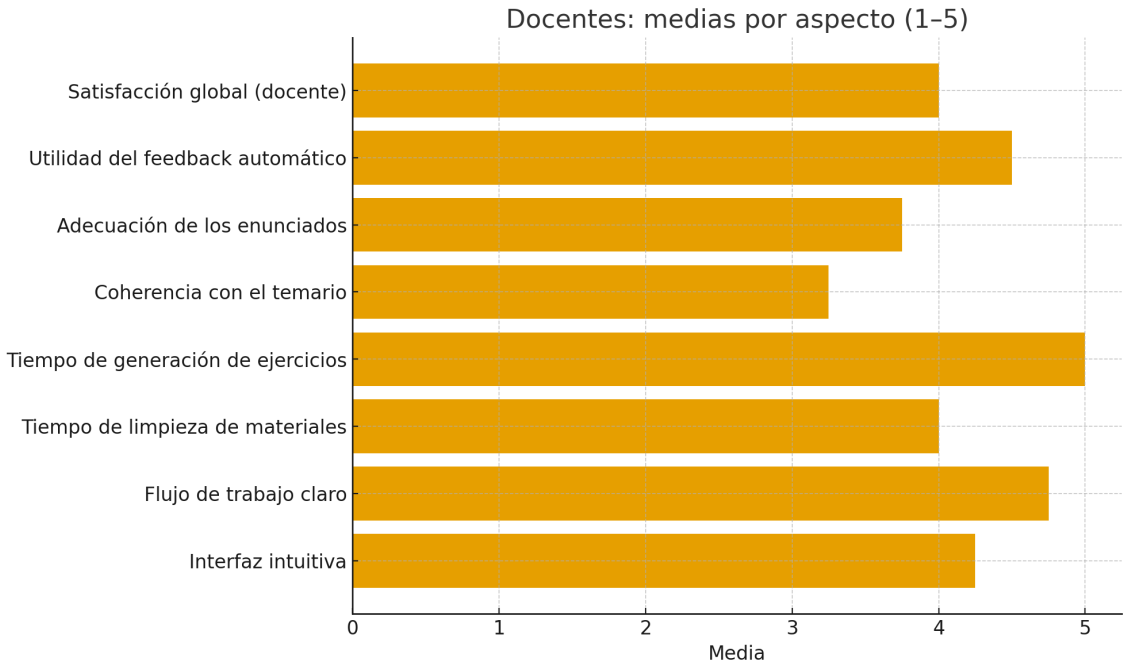

Figura 6.1: Docentes: medias por aspecto (1–5).

Lectura. Se valora muy positivamente el *tiempo de generación* y el *flujo de trabajo*. La *coherencia con el temario* aparece como punto débil. Acción propuesta: reforzar la recuperación de evidencias (RAG) y las plantillas de *prompt*, con una validación rápida por parte del docente antes de publicar.

Estudiante

Aspectos evaluados: facilidad de inicio, claridad de preguntas, coherencia, tiempos de respuesta, *feedback*, estabilidad, justicia de la evaluación y satisfacción.

Tabla 6.2: Estudiantes: medias por aspecto (1–5), $n = 6$.

Aspecto	**Media**
Facilidad de inicio y acceso	**5.00**
Claridad de las preguntas	4.17
Coherencia con el temario	4.50
Tiempos de respuesta	4.00
Feedback automático	4.67
Estabilidad del sistema	4.17
Justicia de la evaluación automática	**3.83**
Satisfacción global (estudiante)	4.17

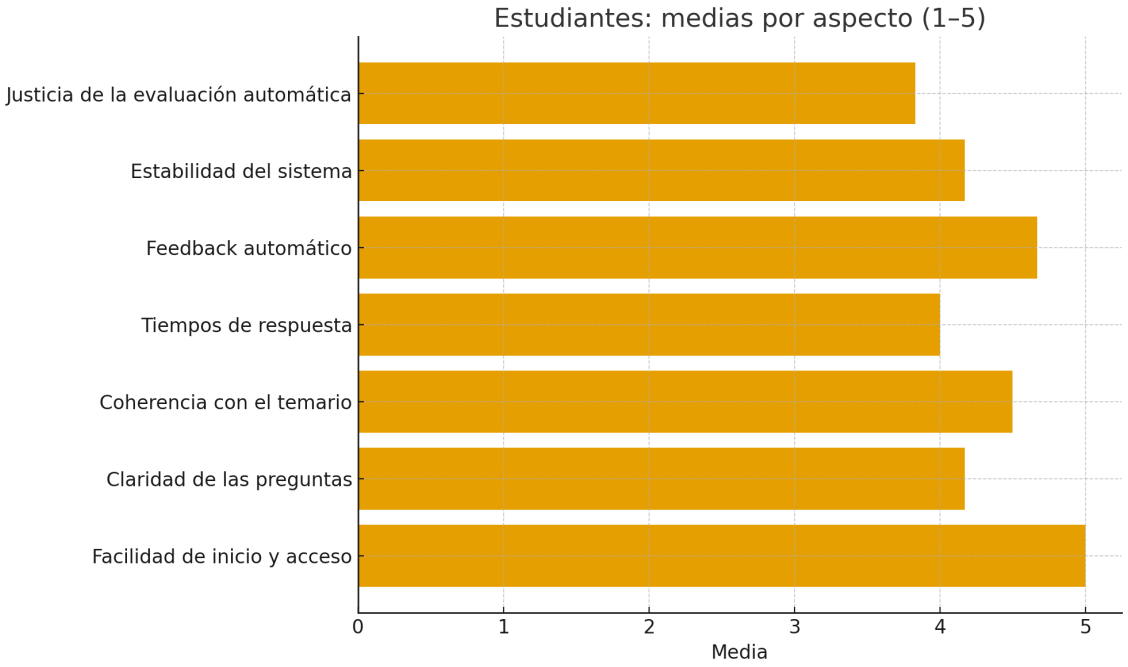

Figura 6.2: Estudiantes: medias por aspecto (1–5).

Lectura. *Facilidad de inicio* y *feedback* obtienen medias altas. La *justicia de la evaluación* es el indicador con menor media. Acción propuesta: mostrar criterios de corrección visibles (rúbrica breve por tipo de ítem) y ejemplos de respuestas parcialmente correctas con su puntuación.

6.3. Comparativa docente–estudiante

Para tres aspectos transversales se observa el siguiente resumen:

Tabla 6.3: Comparativa de medias por rol (1–5).

Aspecto	**Docente**	**Estudiante**
Coherencia con el temario	3.25	4.50
Feedback automático	4.50	4.67
Satisfacción global	4.00	4.17

Lectura. El alumnado percibe mayor coherencia con el temario que el profesorado. El *feedback* es bien valorado por ambos roles y la satisfacción global es positiva en los dos casos.

6.4. Vinculación con los objetivos

- **Obj. 1 (disminuir carga docente):** la valoración del *tiempo de generación* y del *feedback* respalda la reducción de esfuerzo en preparación/corrección.
- **Obj. 2 (disponibilidad y alineación):** la coherencia es adecuada según el alumnado; se propone reforzarla del lado docente ajustando RAG y plantillas.
- **Obj. 3 (retroalimentación inmediata y comprensible):** el *feedback* destaca en ambos roles.
- **Obj. 4 (diagnóstico sin revelar soluciones):** para aumentar la percepción de *justicia*, se recomienda hacer visibles criterios y ejemplos de corrección.

6.5. Pruebas del sistema

Además de la valoración de la interfaz y de la utilidad didáctica (véase Sección 6), se recopilaron impresiones de docentes y estudiantes respecto al **rendimiento del sistema**. El objetivo fue comprobar si las elecciones técnicas (modelo de lenguaje, despliegue local mediante túnel, etc.) resultaban adecuadas desde la práctica de uso.

Percepción de rendimiento

En general, la respuesta del sistema fue considerada **fluida** en la mayoría de las pruebas: la generación de ejercicios se completaba en tiempos razonables y sin bloqueos perceptibles. El modelo *llama3.2:3b* fue evaluado como la opción más equilibrada, al ofrecer velocidad aceptable y coherencia con el temario.

Limitaciones observadas

Se identificaron dos aspectos relevantes:

- **Capacidad de usuarios simultáneos:** debido a que el despliegue se realizó mediante un túnel gratuito, la concurrencia quedó limitada a un máximo de 5 usuarios. Cuando se superaba este número, la sesión podía interrumpirse, lo que fue percibido como una caída del servicio.
- **Variabilidad en tiempos de respuesta:** algunos docentes señalaron que la primera interacción tras iniciar el modelo era más lenta, algo atribuible a la carga inicial en memoria. Esta percepción coincide con el comportamiento esperado de los motores locales.

Conclusión de las pruebas de sistema

Los resultados sugieren que la **selección de modelo y configuración actual** resultan suficientes para un entorno académico controlado (pequeños grupos de estudiantes, pruebas locales). No obstante, para un uso intensivo o con más de cinco usuarios concurrentes, sería necesario evaluar opciones de despliegue con mayor capacidad y estabilidad.

El detalle técnico de las pruebas de humo y del *benchmark* operativo se recoge en el **Apéndice: Documentación del programador**.

Capítulo 7

Conclusiones y líneas de trabajo futuras

Resumen

Este trabajo ha mostrado que es viable integrar modelos de lenguaje, de forma controlada, en una plataforma docente para *Fundamentos de Programación*. A partir de materiales aportados por el profesorado, el sistema genera ejercicios alineados con el temario y proporciona retroalimentación inmediata. La evaluación con usuarios (Cap. 6) confirma fortalezas en *facilidad de uso*, *flujo de trabajo*, *tiempos de generación* y *feedback*; y señala como prioridades de mejora la *coherencia con el temario* en la percepción del profesorado y la *justicia de la evaluación* en la del alumnado.

7.1. Resultados respecto a los objetivos

Se sintetiza a continuación el grado de cumplimiento de los objetivos definidos en el Capítulo 2, apoyando cada punto en evidencias del Capítulo 6:

- **Obj. 1 — Disminuir la carga docente en preparación/corrección:** logrado. Las medias altas en *tiempo de generación* (5.00) y *utilidad del feedback* (4.50) indican reducción efectiva de esfuerzo.
- **Obj. 2 — Aumentar la disponibilidad de práctica alineada:** parcialmente logrado. El alumnado percibe alta coherencia (4.50), mientras que el profesorado la valora de forma más desigual (3.25), lo que sugiere reforzar la alineación curricular.
- **Obj. 3 — Ofrecer retroalimentación inmediata y comprensible:** logrado. El *feedback* obtiene medias elevadas en ambos roles (4.50–4.67).
- **Obj. 4 — Diagnóstico localizado sin revelar la solución completa:** parcialmente logrado. La *justicia de la evaluación* (3.83, alumnado) apunta a hacer más visibles los criterios de corrección.

- **Obj. 5 — Analizar y seleccionar técnicas/modelos adecuados:** logrado. Se implementó una solución basada en RAG y *prompting* estructurado, ejecutada en local.
- **Obj. 6 — Implementar una herramienta trazable y supervisada:** logrado. Gestión de roles diferenciada y revisión docente opcional por intento.
- **Obj. 7 — Evaluar en contexto real de aula:** logrado. Se recogieron valoraciones de 4 docentes y 6 estudiantes integradas en la plataforma.
- **Obj. 8 — Sencillez de uso/extensibilidad:** logrado en uso; extensibilidad prevista. Interfaz valorada positivamente (docencia 4.25; alumnado 5.00) y arquitectura preparada para integración con LMS.

7.2. Conclusiones

Utilidad pedagógica

La plataforma facilita al profesorado la preparación de práctica y la revisión de intentos, manteniendo el control sobre lo generado. Para el alumnado, la combinación de ejercicios y *feedback* inmediato favorece la autoevaluación guiada. La diferencia de percepción en *coherencia con el temario* (más crítica en docencia) y en *justicia* (más sensible en alumnado) orienta con claridad las siguientes mejoras.

Diseño y tecnología

El enfoque *local-first* con ejecución de modelos en local y la combinación de RAG + *prompting* estructurado han resultado adecuados para: (i) anclar las respuestas a materiales del curso con trazabilidad, (ii) modularizar el sistema (interfaz, servicios y motor de IA), y (iii) permitir la supervisión docente sin exponer la solución completa.

7.3. Líneas de trabajo futuras

Las acciones priorizadas a corto plazo se definen como sigue:

- **Alineación curricular reforzada.** Ajustar la recuperación (RAG) con mejores fragmentaciones y filtros por tema/unidad; revisar plantillas de *prompt* con ejemplos canónicos de la asignatura antes de publicar.
- **Criterios visibles para aumentar la justicia percibida.** Mostrar rúbricas breves por tipo de ítem y ejemplos de respuestas parcialmente correctas con su puntuación.
- **Calidad de ítems y banco de preguntas.** Añadir metadatos (nivel, competencia, resultado de aprendizaje, dificultad), control docente de versiones y un flujo de validación rápida.

- **Integración con LMS.** Incorporar LTI para publicar y recoger calificaciones desde plataformas institucionales (p. ej., Moodle), evitando duplicidades.
- **Analítica de uso y mejora continua.** Registrar métricas de adopción (intentos, tiempo en tarea, revisiones docentes) y cerrar el bucle de mejora con iteraciones cortas.
- **Extensión funcional.** Explorar generación de exámenes con variantes y rúbricas exportables; estudiar compilación segura en *sandbox* para enriquecer el *feedback* en ejercicios de código.

Cierre

El prototipo desarrollado demuestra que es posible ofrecer práctica alineada con el temario y retroalimentación inmediata bajo supervisión docente. La evidencia recogida respalda la utilidad de la herramienta y acota con precisión las mejoras que incrementarán su impacto en docencia: reforzar la alineación curricular y hacer más transparentes los criterios de corrección. Estas líneas de trabajo permiten proyectar una evolución natural hacia un despliegue sostenido en asignaturas de primer curso.

Apéndices

Apéndice A

Documentación del programador

Resumen

Este apéndice reúne la información técnica necesaria para comprender la arquitectura de la solución, mantener el código y desplegarlo en entornos locales de desarrollo. Se describen la estructura de directorios, los flujos internos (ingesta/limpieza, generación, intento y diagnóstico), la configuración de entornos (`.env`), los pasos de despliegue y las pruebas de humo. Las decisiones de selección de modelos y fundamentos tecnológicos se tratan en **Cap. 5**; la evaluación pedagógica/UX del sistema se recoge en **Cap. 6**, que remite aquí para las pruebas técnicas.

A.1. Estructura de directorios

El proyecto se organiza en dos bloques: *frontend* (Laravel) y *backend* (FastAPI).

Frontend (Laravel)

- `/app`: lógica de negocio (modelos Eloquent, *policies*, servicios).
- `/app/Http/Controllers`: controladores por contexto (Temas, Materiales, Exámenes, Intentos).
- `/routes`: definición de rutas (`web.php`, `api.php`).
- `/resources/views`: vistas Blade (UI docente/estudiante).
- `/database/migrations`: migraciones; `/database/seeders` (si corresponde).
- `/public`: ficheros estáticos; symlink a `storage` mediante `php artisan storage:link`.
- `/storage/app/public`: subidas de materiales y artefactos derivados (texto limpio, etc.).

Backend (FastAPI)

- `/api_ia.py`: aplicación principal FastAPI (`/process`, `/generate`, `/grade`).
- `/prompts/`: plantillas de *prompting* (clean → generate → grade).
- `/tests`: pruebas unitarias/ejemplos.
- `/.env`: variables del servicio (puerto, modelo por defecto, rutas).

Utilidades y resultados

- `/bench_out`: métricas de *benchmark* de humo en local.
- `/scripts/llm_bench_win_v5.sh`: script de verificación rápida de modelos locales en Ollama.

A.2. Arquitectura y flujos

La plataforma adopta una arquitectura de dos servicios: Laravel (aplicación/presentación) y FastAPI (servicios IA), orquestados localmente con Ollama como motor de modelos. A continuación se listan los diagramas obligatorios; inclúyanse con `\caption{}` y `\label{}` y referencia en el texto:

- **Ingesta y limpieza de materiales** (`img/ingesta_limpieza.png`)
- **Generación de ejercicios** (`img/generacion.png`)
- **Intento de estudiante y diagnóstico/revisión** (`img/intento_diagnostico.png`)
- **Extensiones/plug-ins (secuencia)** (`img/secuencia_extension.png`)
- **Despliegue** (`img/despliegue.png`)
- **Modelo relacional** (`img/modelo_relacional.png`), en apaisado a página completa con `pdflscape`.

Flujo interno resumido

1. El **docente** sube material (PDF/DOCX) en Laravel → se almacena en `storage/app/public`.
2. Laravel encola un trabajo o invoca FastAPI para **limpieza** (normalización/OCR) y genera `.txt` limpio.

3. Con texto limpio, Laravel solicita a FastAPI la **generación** (teórico/práctico/mixto) con salida JSON validada.
4. Publicación de exámenes, intentos del **estudiante**, corrección automática y revisión docente.

A.3. Configuración del entorno

Requisitos

- PHP 8.2+, Composer, Laravel 10.x.
- Node.js (si se usa Vite para *assets*).
- Python 3.11+, FastAPI, Pydantic, Uvicorn.
- Ollama instalado y modelos locales (p. ej., `ollama pull llama3.2:3b`).
- MySQL/MariaDB o PostgreSQL.

Variables de entorno (`.env`)

Laravel (`.env`)

```
APP_URL=http://localhost:8001
APP_ENV=local
APP_KEY=base64:GENERADA_CON_KEY:GENERATE
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ia_creacionejercicios
DB_USERNAME=usuario
DB_PASSWORD=******
FASTAPI_BASE_URL=http://localhost:8000
QUEUE_CONNECTION=database   # o sync si no usas colas
```

FastAPI (`.env`)

```
API_PORT=8000
OLLAMA_BASE_URL=http://localhost:11434
DEFAULT_MODEL=llama3.2:3b
MAX_TOKENS=256
TIMEOUT_S=120
```

A.4. Despliegue y arranque de servicios

Laravel

1. `composer install`
2. `cp .env.example .env && php artisan key:generate`
3. Configurar variables en `.env` (DB, `FASTAPI_BASE_URL`).
4. `php artisan migrate` (`php artisan migrate -seed` si procede).
5. `php artisan storage:link`
6. (Opcional) `npm install && npm run build` (o `npm run dev`).
7. `php artisan serve -port=8001`

FastAPI

1. Crear/activar entorno Python e instalar dependencias (`pip install -r requirements.txt`).
2. Configurar `.env` del servicio.
3. `uvicorn api_ia:app -reload -port=8000`

Ollama (modelos locales)

1. `ollama pull llama3.2:3b`
2. (Opcional) `ollama pull llama3.1:8b`, `gemma3:12b`, etc.
3. (Sugerido) Definir `OLLAMA_KEEP_ALIVE` para reducir TTFT tras la primera carga.

A.5. Pruebas del sistema (humo y funcionales)

Desde la perspectiva de uso (docente/estudiante) se ejecutan **pruebas funcionales y de humo** sobre los flujos clave. Las métricas de inferencia se usan únicamente como verificación operativa del entorno local. La evaluación pedagógica/UX y las tablas de encuestas se reportan en **Cap. 6**.

Checklist funcional

- **Ingesta y limpieza**: subida de PDF/DOCX y obtención de `.txt` limpio vía FastAPI; persistencia correcta en Laravel.
- **Generación**: creación de ejercicios a partir de material limpio y validación del *esquema JSON* (campos obligatorios, `pauta_correccion` en teórico, `codigo_java` en práctico).
- **Roles y permisos**: separación docente/estudiante con Spatie; respuestas HTTP adecuadas ante accesos no autorizados.
- **Trazabilidad y RAG (humo)**: intento de estudiante + revisión docente; vínculos *Intentos*/*Revisiones* y correspondencia temática de enunciados.
- **Estabilidad y despliegue mínimo**: lotes cortos de limpieza/generación con `HTTP 200` y migraciones/arranque limpio (`php artisan migrate`, colas/API).

Benchmark de humo (opcional)

Ejecución de `scripts/llm_bench_win_v5.sh`[1] con `num_predict`=256 y 3 iteraciones/modelo para comprobar servicio estable; se recogen TTFT, TPS y VRAM [29, 21, 30].

Tabla A.1: Benchmark de humo en local (3 iteraciones/modelo, `num_predict`=256).

Modelo	n	TTFT (ms)	TPS (tokens/s)	VRAM pico (MB)	Éxito (%)
llama3.2:3b	3	1869	59.12	0	100.0
llama3.1:8b	3	2329	26.29	0	100.0
gemma3:12b	3	4116	11.65	0	100.0
phi4:14b	3	3438	8.42	0	100.0
deepseek-r1:14b	3	3650	8.23	3717	100.0
qwen2.5:14b-instruct	3	5158	8.14	1048	100.0

Glosario de métricas

- **n**: iteraciones por modelo (consistencia).
- **TTFT (ms)**: tiempo hasta primer token (*responsividad*).
- **TPS (tokens/s)**: ritmo de generación (*fluidez*).
- **VRAM pico (MB)**: memoria máxima de GPU (*viabilidad en hardware*).
- **Éxito (%)**: ejecuciones sin error (estabilidad del servicio).

[1] Repositorio: https://gitlab.inf.uva.es/ia_docencia/ia_creacionejercicios. Script adaptado a modelos locales en Ollama; imprime métricas por iteración.

Lectura breve *llama3.2:3b* ofrece mayor **velocidad** ($\approx$ 59 tok/s) con TTFT contenido; modelos de 12–14B (*gemma/phi4/deepseek/qwen*) incrementan **riqueza** a costa de **latencia** y, en algún caso, **VRAM** (*deepseek-r1:14b* $\sim$ 3,7 GB). Configurar `OLLAMA_KEEP_ALIVE` puede reducir TTFT tras la primera carga [30]. Los valores dependen de hardware y cuantización [29]. La orquestación local se realiza con LangChain/Ollama [21].

A.6. Resolución de problemas (FAQ técnica)

- **Laravel no sirve archivos de `storage`** → ejecutar `php artisan storage:link`.
- **Errores 500 al subir PDF** → revisar permisos de `storage/` y tamaño máximo en `php.ini`.
- **FastAPI no responde** → comprobar puerto `8000`, CORS y `FASTAPI_BASE_URL` en `.env` de Laravel.
- **TTFT alto en primera llamada** → activar `OLLAMA_KEEP_ALIVE` y mantener modelos pre-cargados.
- **Colas atascadas** → revisar `QUEUE_CONNECTION`, `php artisan queue:work` y logs.

A.7. Seguridad y privacidad

- Datos y materiales se procesan **localmente**; no se envían a terceros.
- Revisar `APP_DEBUG=false` en despliegues de demostración.
- Roles y permisos gestionados con Spatie; validar políticas antes de publicar.

A.8. Mantenimiento y actualizaciones

- Actualizar dependencias de forma controlada (`composer update`, `pip install -U`).
- Versionado de prompts (`/prompts`) y `/bench_out` para trazabilidad.
- Scripts de utilidad: `scripts/llm_bench_win_v5.sh`.

A.9. Referencias internas y repositorio

Código y scripts: https://gitlab.inf.uva.es/ia_docencia/ia_creacionejercicios. Elección de modelo y justificación tecnológica: **Cap. 5**. Evaluación pedagógica/UX: **Cap. 6** (con remisión a A.5 para el detalle técnico).

Apéndice B

Documentación de usuario

Resumen

Este apéndice describe cómo utilizar la plataforma desde el punto de vista del usuario final. Incluye los requisitos mínimos, un arranque rápido y un manual básico de uso para los perfiles de **docente** y **estudiante**. La preparación completa del entorno, dependencias y despliegue técnico se detalla en el *Apéndice: Documentación del programador*.

B.1. Requisitos de usuarios

El sistema está pensado para ejecutarse en entorno local.

- Navegador web actualizado (Chrome, Firefox o equivalente).
- Conocimientos básicos de navegación web.
- Para el rol **docente**: disponer de materiales en PDF o DOCX.

Requisitos técnicos mínimos (alto nivel)

- Ordenador con al menos 8 GB de RAM (recomendado 16 GB).
- Windows, Linux o macOS.

Nota. La instalación de herramientas (PHP/Laravel, Python/FastAPI, Ollama y base de datos) y la configuración de `.env` forman parte del **Apéndice del programador**.

B.2. Instalación (arranque rápido)

Con el entorno ya preparado por el responsable técnico:

1. Iniciar la aplicación web:

   ```
   php artisan serve --port=8001
   ```

2. Iniciar el servicio de backend:

   ```
   uvicorn api_ia:app --reload --port=8000
   ```

3. (Si procede) Asegurar que los modelos locales están disponibles en Ollama.
4. Acceder desde el navegador a http://localhost:8001.

Si el entorno no está configurado, siga el **Apéndice: Documentación del programador** (instalación completa y variables `.env`).

B.3. Manual del usuario

Acceso rápido (demo)

URL: https://tooltfm.pagekite.me/login

Credenciales de prueba:

- **Docente**: `docente@ejemplo.com` / `docente`
- **Estudiante**: `estudiante@ejemplo.com` / `estudiante`

Tras iniciar sesión, se muestran los *dashboards* específicos de cada rol.

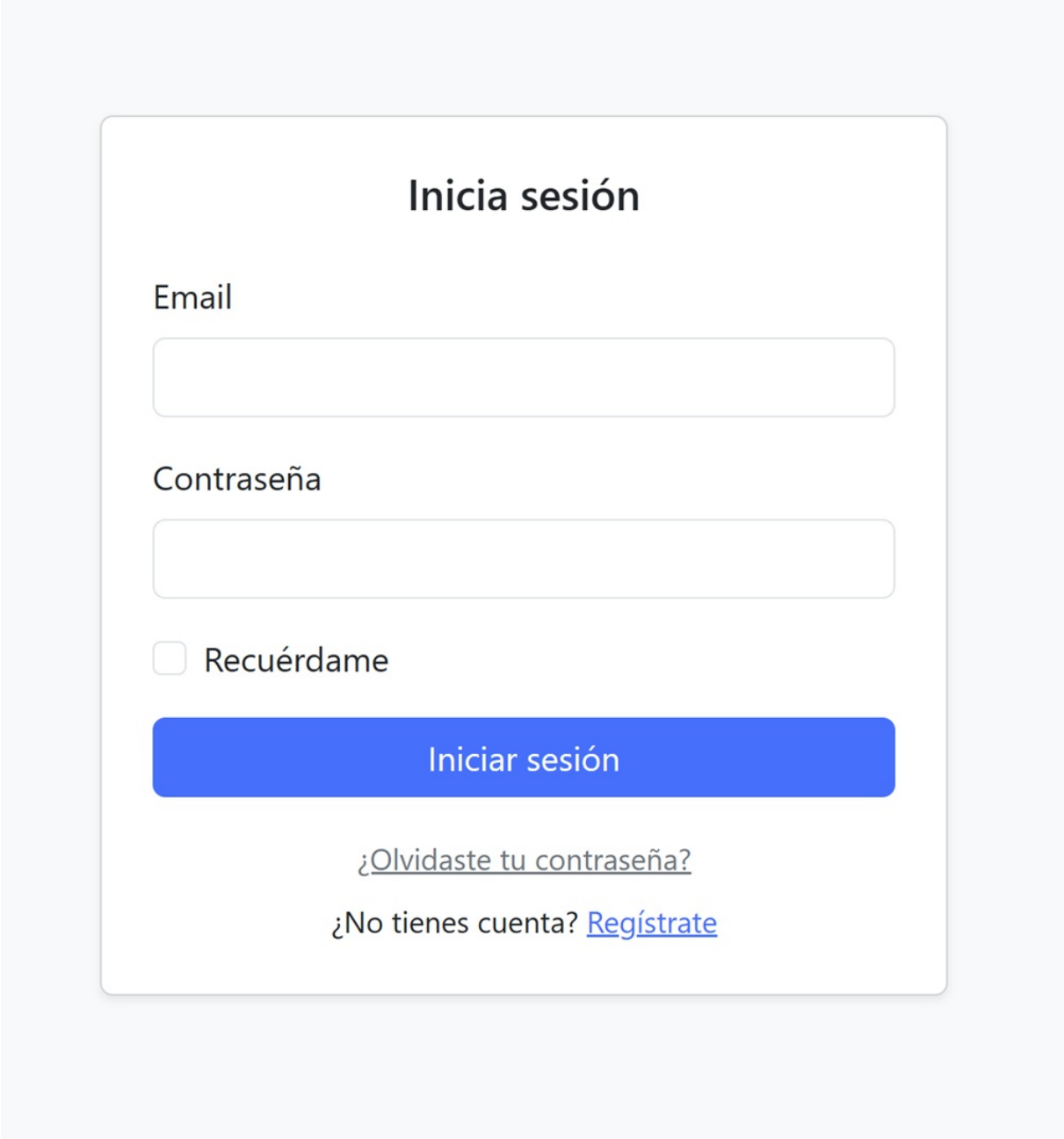

Figura B.1: Pantalla de acceso (`/login`).

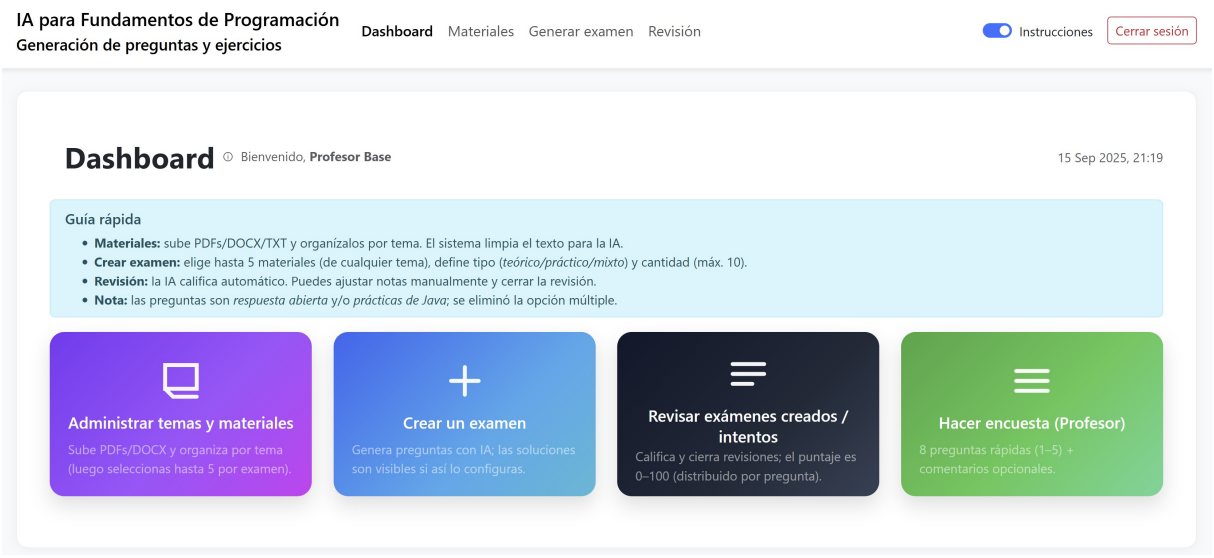

Figura B.2: Dashboard del **docente** (Temas, Materiales, Exámenes, Revisión, Encuesta).

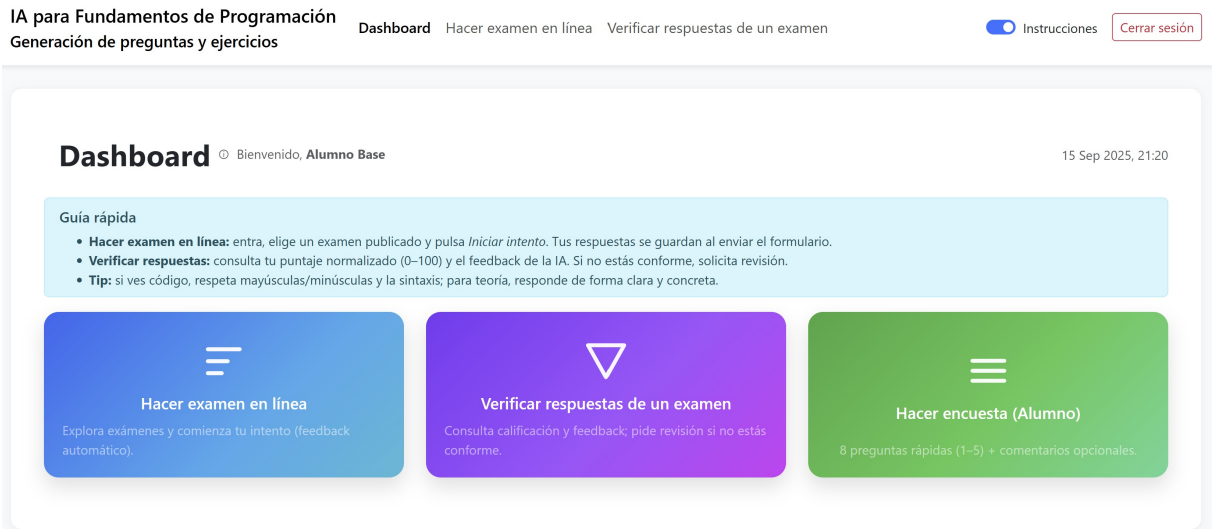

Figura B.3: Dashboard del **estudiante** (exámenes disponibles, intentos y encuesta).

Flujo para docencia

1) Crear tema

1. Ir a `Temas` → `Crear tema`.
2. Completar *Título* (y opcionalmente *Descripción*). Guardar.

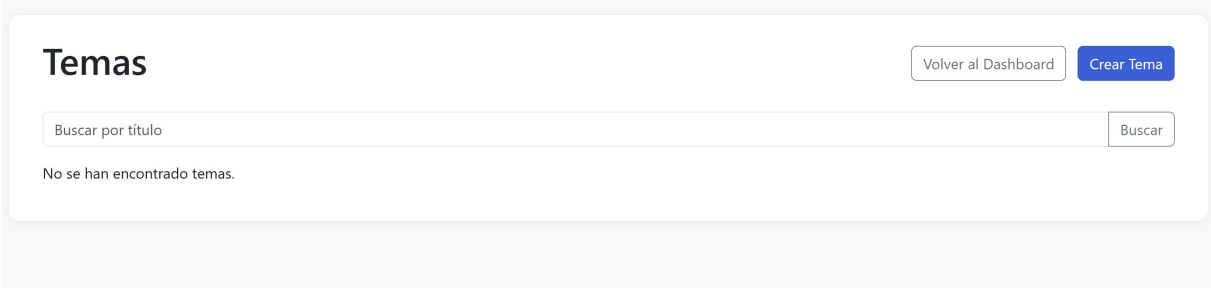

Figura B.4: Listado de temas del docente.

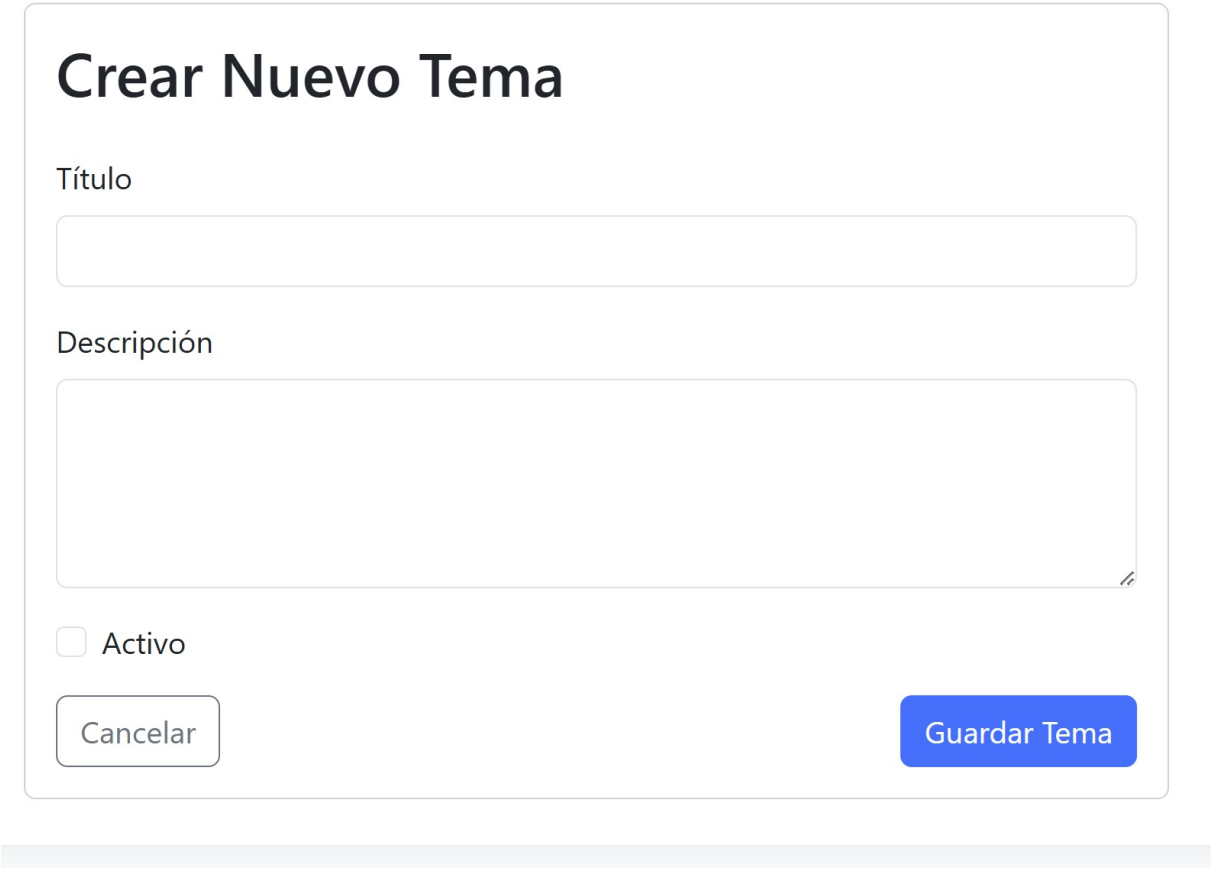

Figura B.5: Formulario de creación de tema.

2) Subir material (PDF/DOCX)

1. En el tema: `Materiales` → `Subir PDF/DOCX`.
2. Seleccionar archivo y (opcional) *Alias*. Guardar.

3. El sistema extrae y limpia el texto automáticamente.

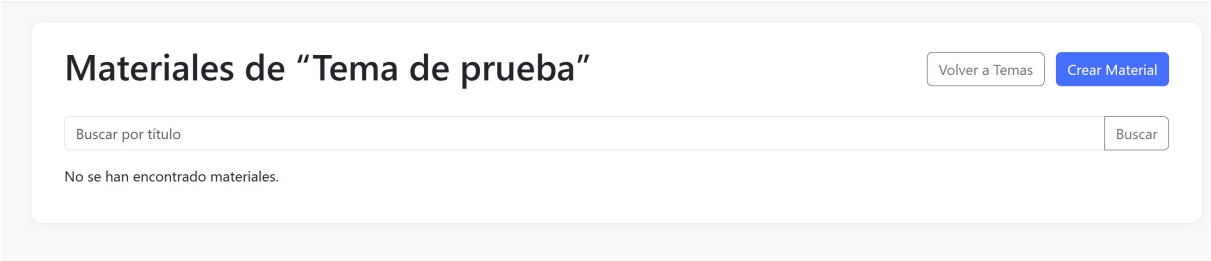

Figura B.6: Materiales del tema (listado).

Figura B.7: Subida de material (con alias opcional).

3) Generar examen

1. Ir a `Exámenes` → `Crear`.
2. Elegir hasta 5 materiales.
3. Seleccionar *Tipo* (teórico/práctico/mixto) y *n.º de preguntas* (1–10).

4. (Opcional) Incluir soluciones y/o marcar como publicado. Guardar.

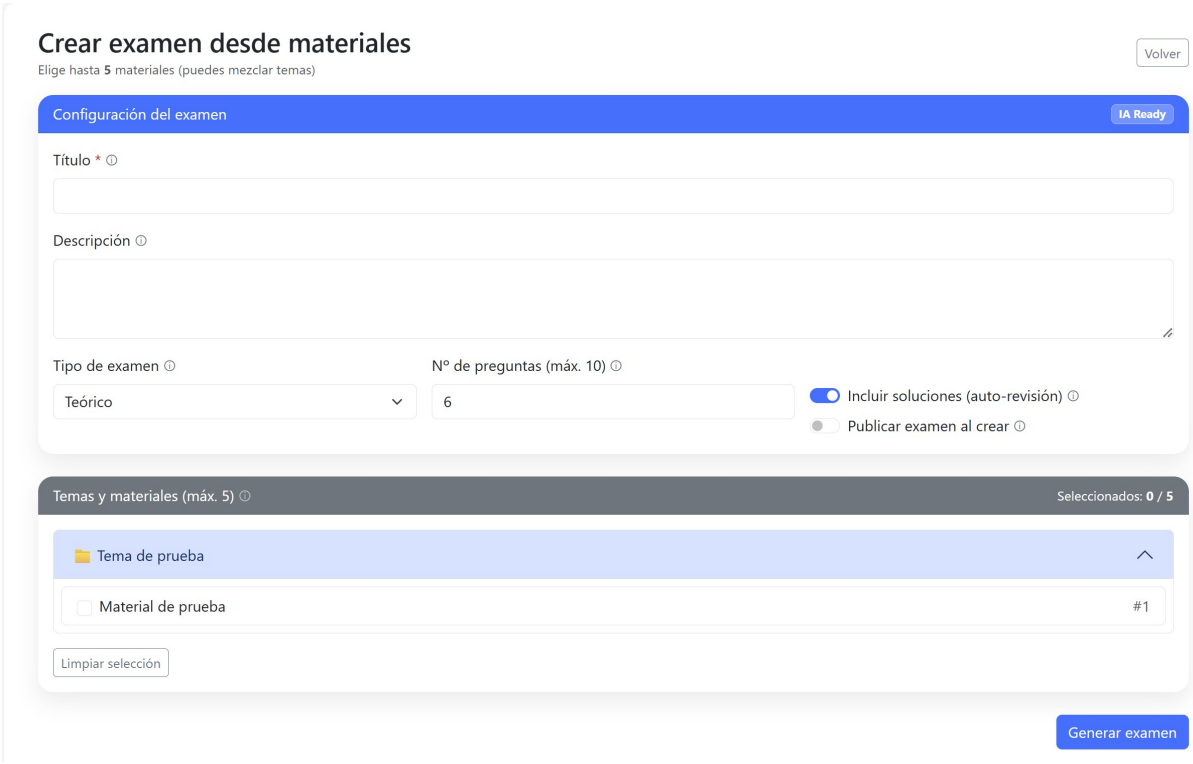

Figura B.8: Formulario de creación de examen (selección de materiales y parámetros).

4) Publicar y revisar

1. En `Exámenes` → `Detalle`: activar *Publicar*.
2. Abrir `Intentos` para ver envíos de estudiantes.
3. Entrar a un intento, ajustar **puntuaciones**/comentarios y, si procede, *Finalizar revisión*.

Figura B.9: Detalle del examen.

Figura B.10: Listado de intentos de un examen (estado, estudiante, **puntuación**).

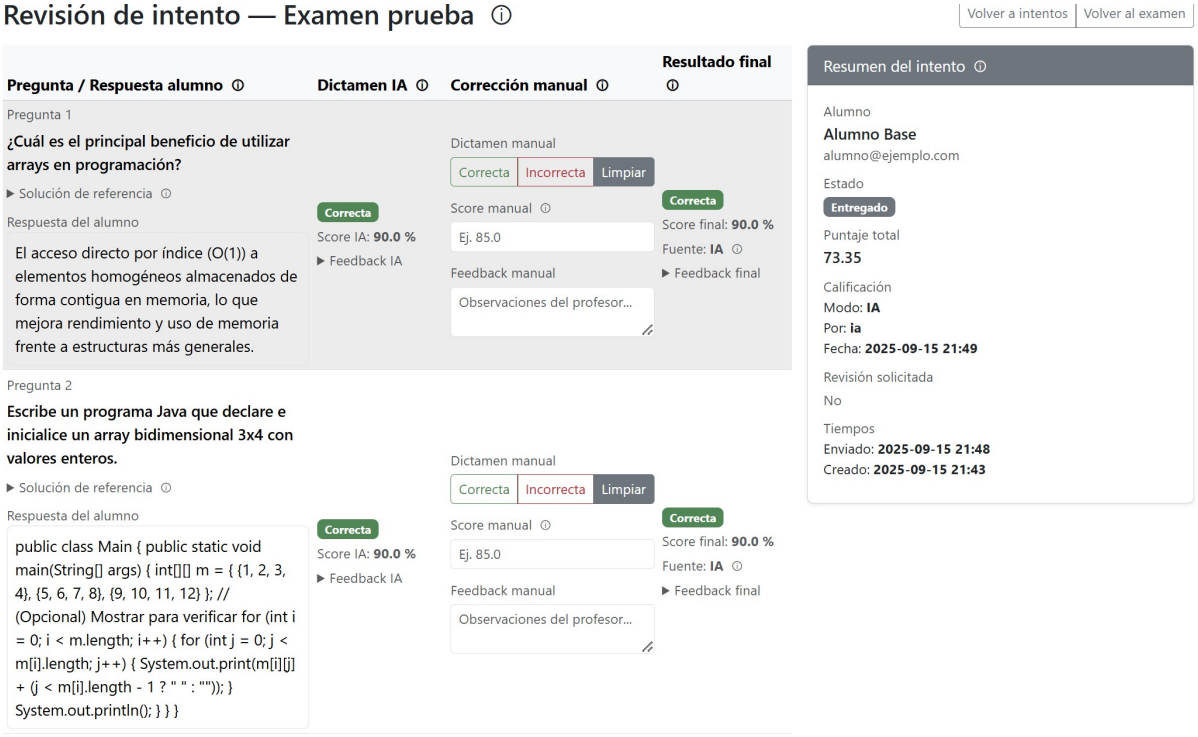

Figura B.11: Vista de intento y corrección manual (edición de dictamen y cierre de revisión).

5) Encuesta del profesorado

1. Acceder a `/encuesta/docente`.
2. Responder los 8 ítems (escala 1–5) y enviar.

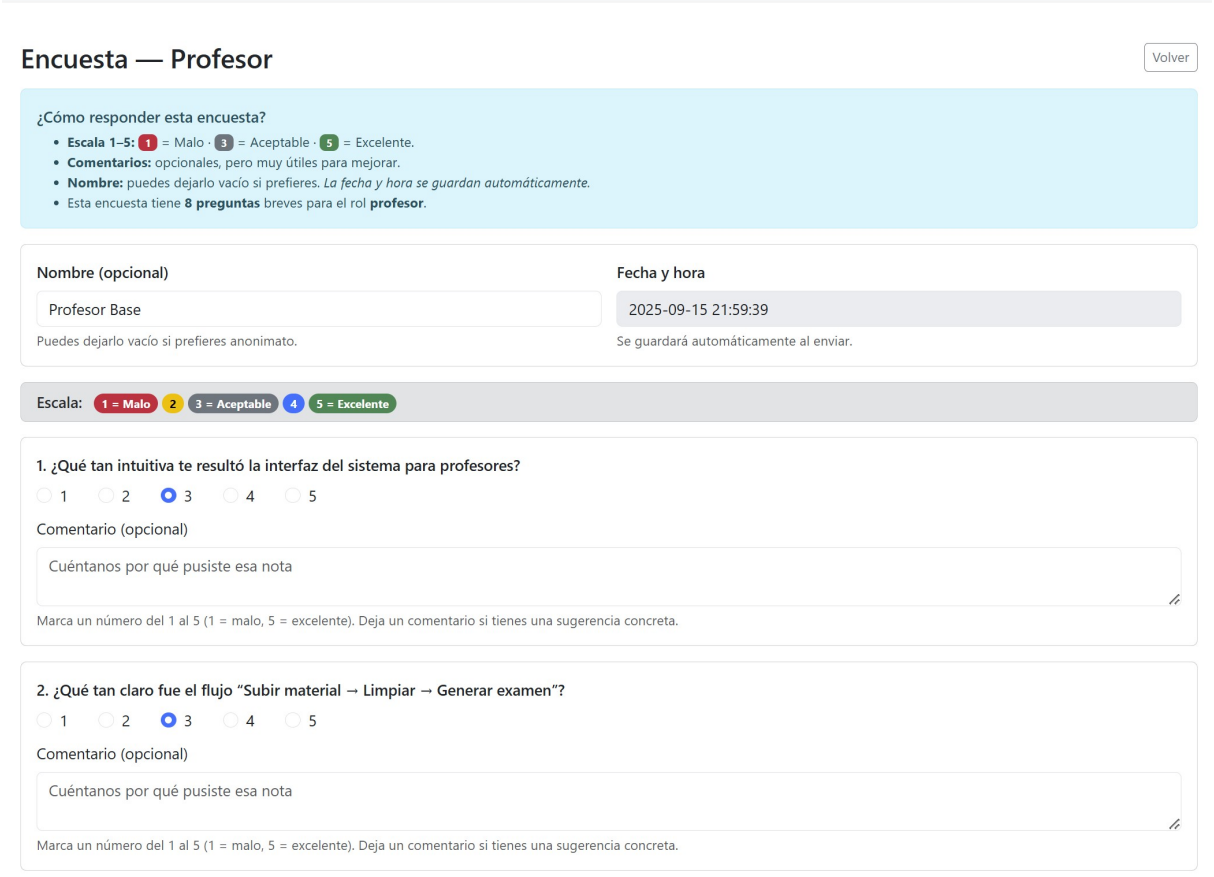

Figura B.12: Formulario de encuesta para docencia.

Flujo para estudiantado

1) Ver exámenes disponibles

1. Ir a `Estudiante` → `Exámenes`.
2. Filtrar o buscar si es necesario; abrir un examen publicado.

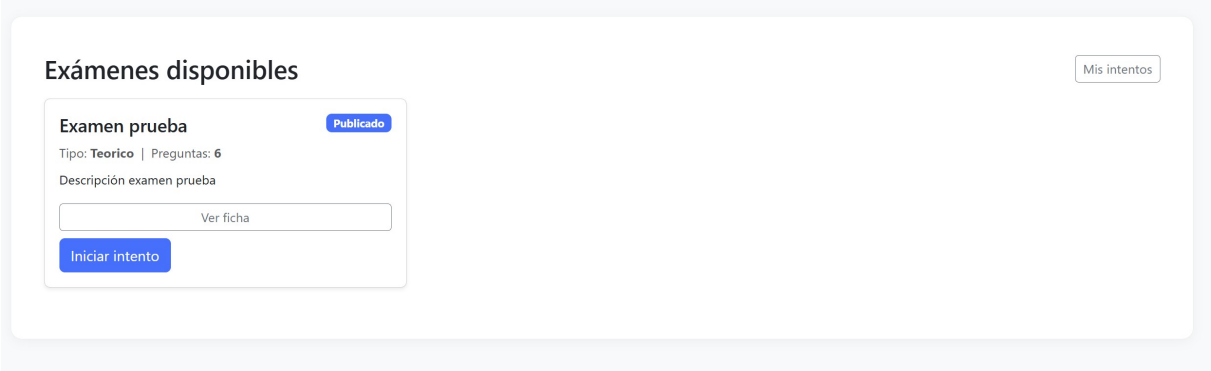

Figura B.13: Exámenes disponibles para el estudiante.

2) Ficha del examen e inicio del intento

1. En la ficha del examen, pulsar `Iniciar intento`.
2. Si ya existe un intento en curso, aparecerá `Continuar intento`.
3. Si ya se entregó, aparecerá `Ver resultado`.

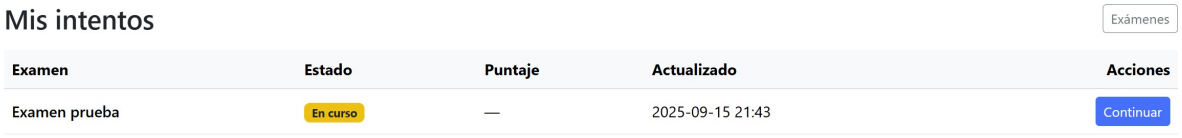

Figura B.14: Ficha del examen con CTA (iniciar/continuar/ver resultado).

3) Responder y enviar

1. Completar las respuestas en el formulario del intento.
2. Pulsar `Enviar examen`. El sistema califica automáticamente (IA) si hay solución de referencia.

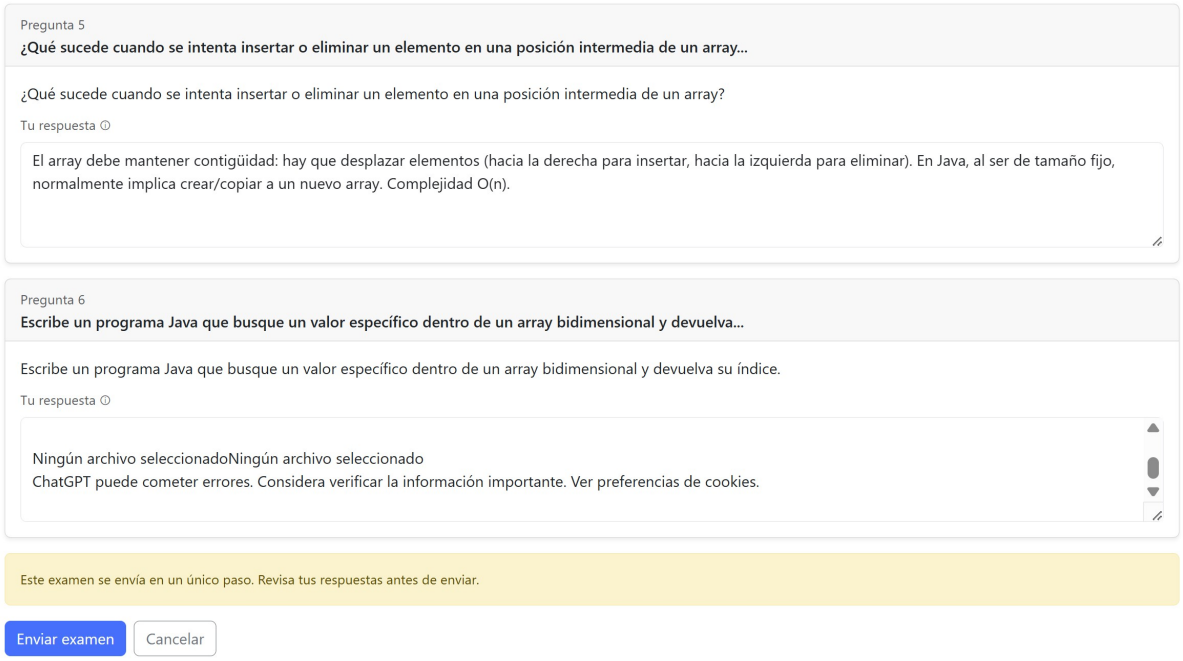

Figura B.15: Intento en curso (formulario de respuestas).

4) Resultado y solicitud de revisión

1. Tras el envío, se muestra el **resultado**: **puntuación** total, detalle por pregunta y *feedback* de la IA.
2. Si no está conforme, pulsar `Solicitar revisión` para que un docente ajuste la nota.

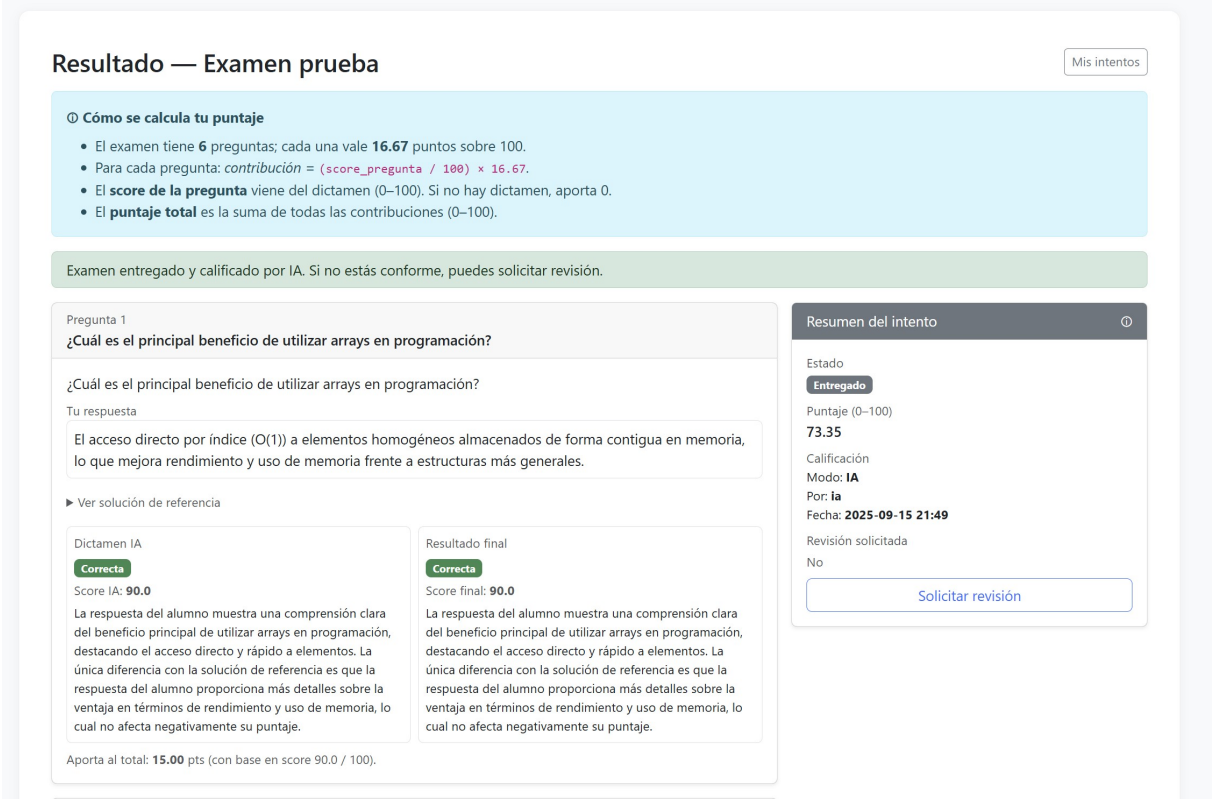

Figura B.16: Resultado del intento (puntuación total y por ítem, *feedback* IA).

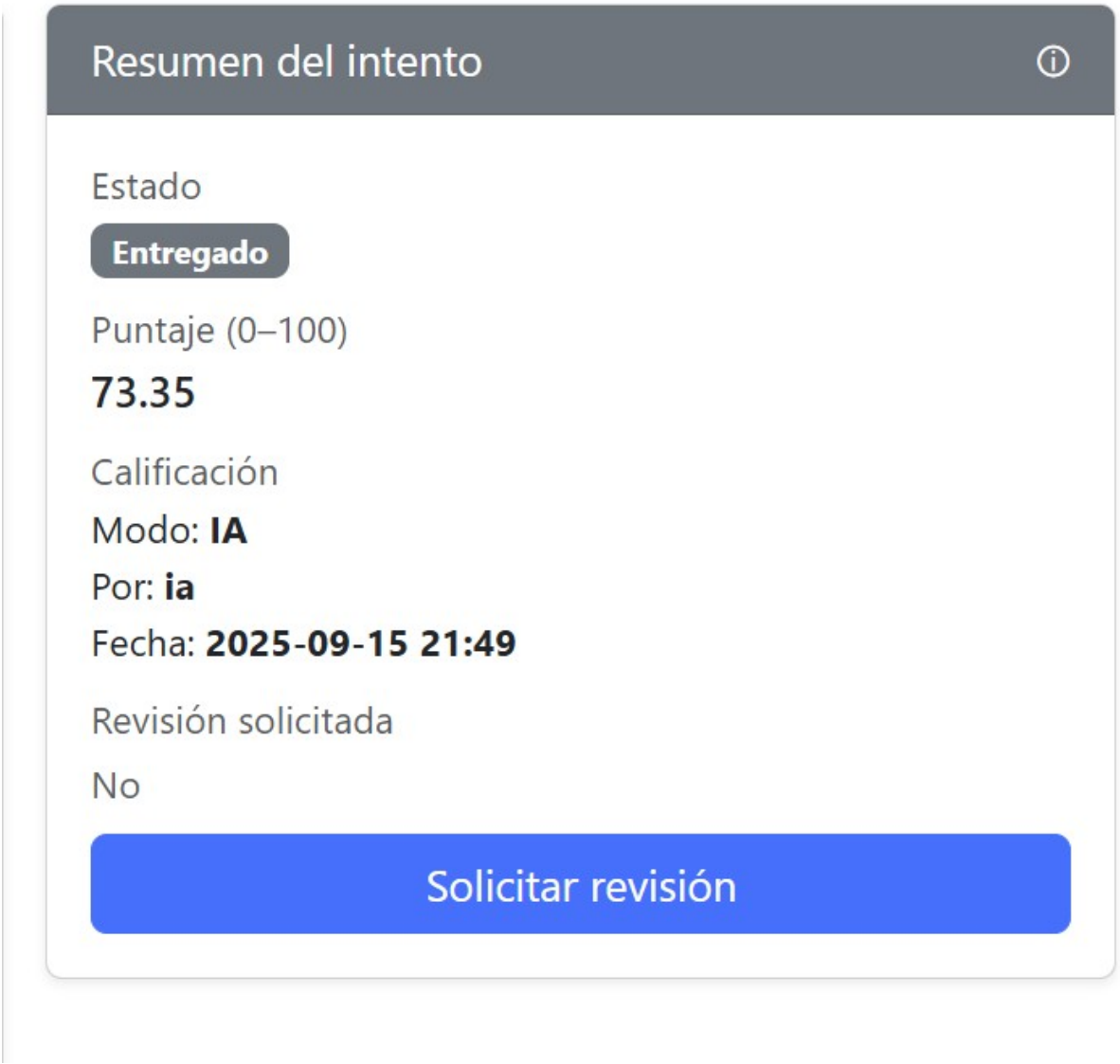

Figura B.17: Envío de solicitud de revisión del intento.

5) Historial de intentos

1. Ir a `Estudiante` → `Mis intentos` para ver todos los envíos y su estado.

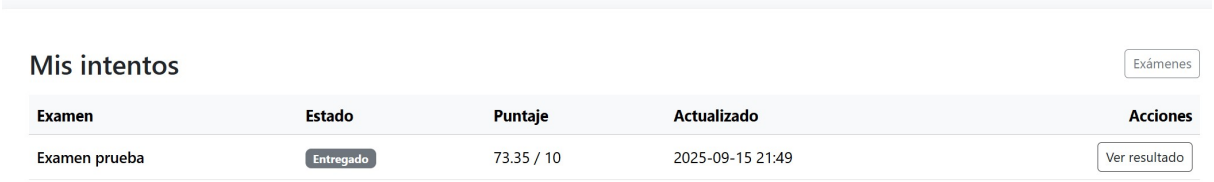

Figura B.18: Historial de intentos del estudiante.

6) Encuesta del alumnado

1. Acceder a `/encuesta/estudiante`.
2. Responder los 8 ítems (escala 1–5) y enviar.

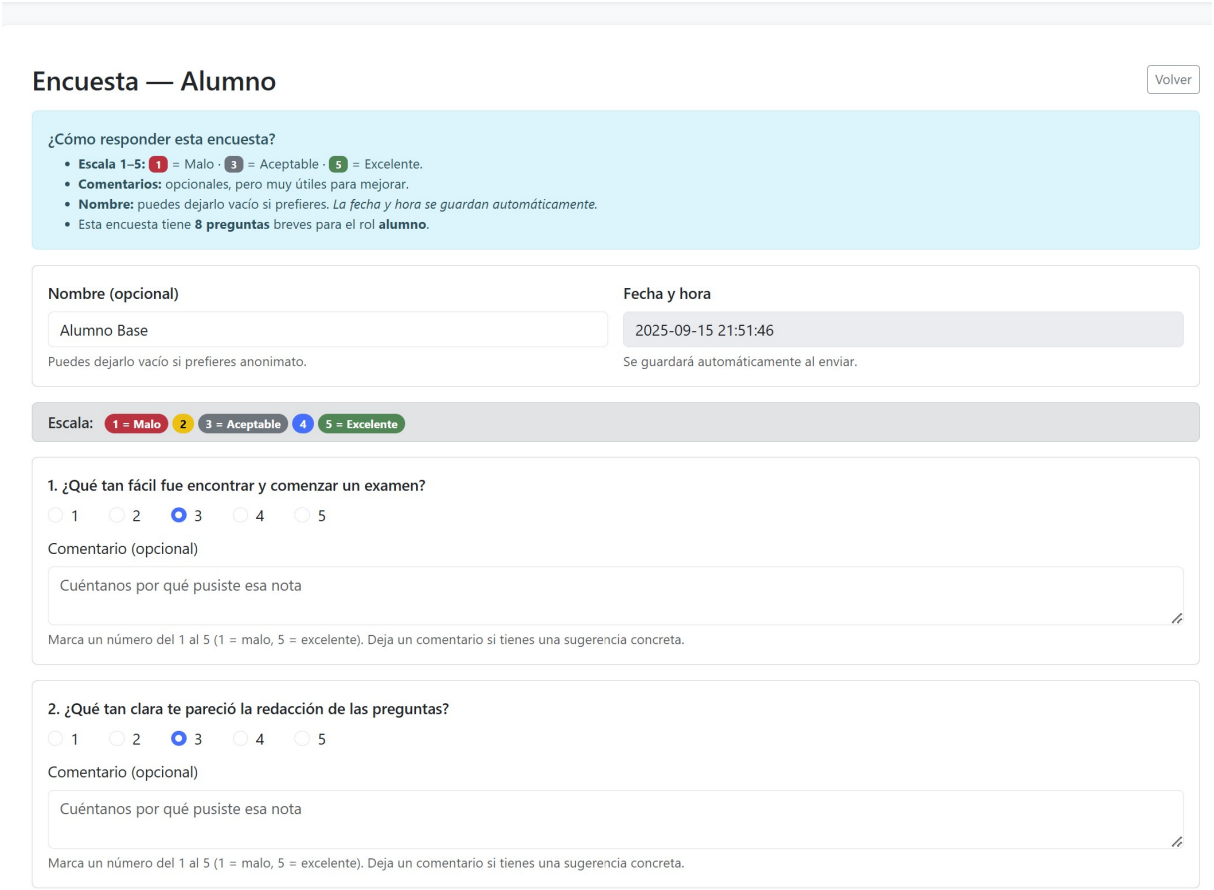

Figura B.19: Formulario de encuesta para estudiantado.

Cerrar sesión

1. En la barra superior, abrir el menú del usuario (avatar/nombre).
2. Seleccionar `Cerrar sesión`.

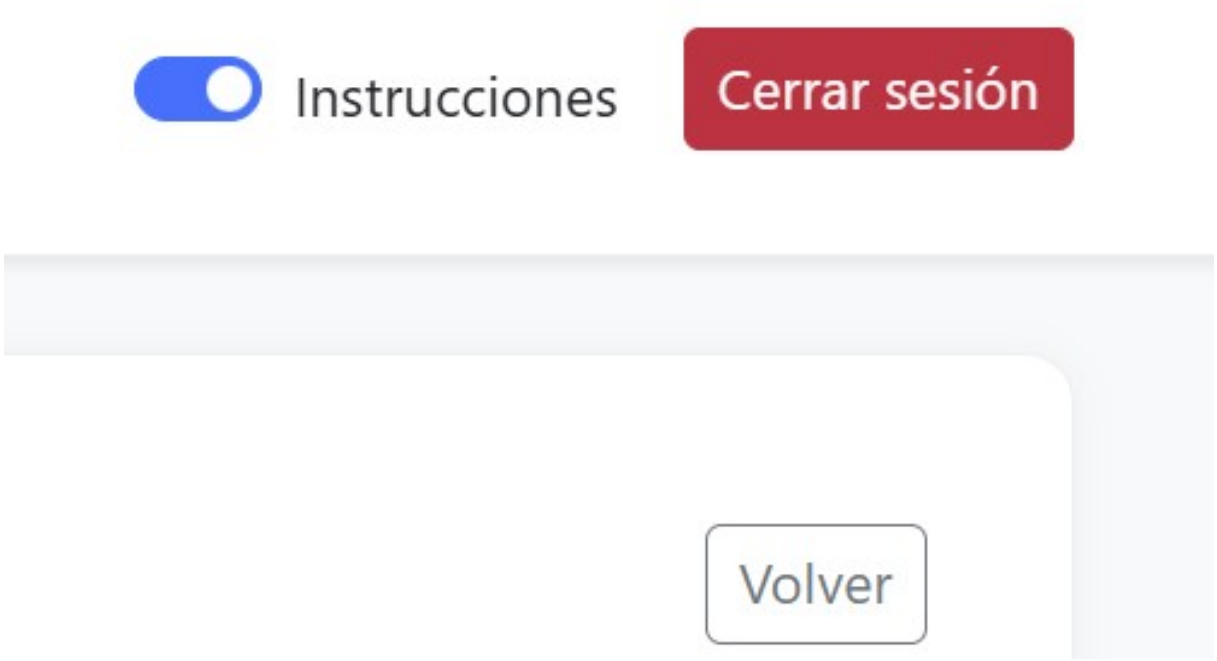

Figura B.20: Menú de usuario: opción *Cerrar sesión*.

Notas importantes

- El tiempo de generación depende del modelo y del hardware disponibles.
- La ejecución es local: no se requiere conexión a servicios externos.
- Los materiales procesados se almacenan internamente y no se comparten fuera del entorno local.

En resumen, el flujo de uso es: *subir material* → *procesar* → *generar ejercicios* → *consultar resultados*.

Apéndice C

Prompting

Resumen

Este anexo documenta las decisiones de *prompting* empleadas para alinear la generación y la evaluación con los objetivos docentes de *Fundamentos de Programación*. Se adopta un enfoque *schema-first* (salidas en JSON válido) y se combinan instrucciones explícitas con recuperación de evidencia (*RAG*) para reducir alucinaciones y variabilidad. La organización y las pautas siguen las recomendaciones recogidas en encuestas y catálogos de patrones de *prompting* [24, 41].

C.1. Motivación y contexto

Los modelos de lenguaje muestran sensibilidad a cómo se formulan las instrucciones. La literatura sistematiza técnicas para mejorar control, formato y estabilidad (instrucciones positivas/negativas, ejemplos mínimos, plantillas y validación de salida) [24, 41]. En este proyecto, el *prompting* no se concibe como "magia", sino como ingeniería de interfaces: especificar tarea, restricciones, formato y criterios de corrección de forma inequívoca, y proporcionar al modelo el contexto pertinente.

C.2. Problemas sin contexto y mitigación

En ensayos tempranos, sin materiales de referencia, se observaron respuestas plausibles pero incorrectas (alucinaciones), oscilación entre ejecuciones y errores de formato. Estos fenómenos están descritos de forma amplia en la bibliografía [16]. La mitigación adoptada combina: (i) limpieza/síntesis de materiales, (ii) RAG para anclar la generación en evidencias y (iii) plantillas con salida estructurada.

C.3. Diseño de *prompts* en la herramienta

El sistema implementa tres haces coherentes con la canalización del Cap. 5 (*clean* → *generate* → *grade*). Las plantillas residen en el servicio `api_ia.py`[1] y se parametrizan por tipo de ejercicio.

Limpieza y síntesis (`build_clean_prompt`)

Propósito. Homogeneizar materiales (PDF/DOCX) en un texto docente conciso, eliminando ruido (marcas, numeraciones, soluciones resueltas) y extrayendo definiciones, enunciados base y términos clave.

Rationale. Mejorar la calidad de la entrada reduce ambigüedad e incrementa la utilidad por token, lo que se traduce en salidas más estables y verificables [24].

Fragmento ilustrativo.

```
TAREA: Reescribe el material para uso docente.
INSTRUCCIONES:
- Mantén solo definiciones, ejemplos y enunciados base.
- Elimina soluciones y pasos resueltos.
- Devuelve texto limpio en secciones con títulos claros.
SALIDA: texto plano (sin listas de respuestas).
```

Generación de ejercicios (`build_test_prompt`)

Propósito. Producir ítems teóricos y prácticos alineados con el temario, con control estricto de formato.

Reglas.

- **Teórico (respuesta abierta).** Prohibida la opción múltiple. Incluir una pauta breve de corrección.
- **Práctico (Java, imperativo).** Exigir código funcional completo y autocontenido. Evitar salidas "tipo datos".
- **Mixto.** Combinar ambos, preservando las restricciones.
- **Formato.** *Solo JSON válido* con campos documentados.

Fragmento ilustrativo (JSON esperado).

[1] Código en el repositorio del proyecto.

```
{
  "type": "teorico|practico|mixto",
  "items": [
    {
      "enunciado": "...",
      "pauta_correccion": "... (breve, sin solución completa)",
      "codigo_java": "...",          // solo en "practico" o "mixto"
      "tema": "Estructuras de control",
      "nivel": "basico|intermedio|avanzado"
    }
  ]
}
```

Este diseño aplica patrones de *prompting* recomendados: tareas explícitas, instrucciones negativas y esquema de salida obligatorio [41, 24].

Evaluación de respuestas (`build_grade_prompt`)

Propósito. Devolver una calificación trazable sin revelar la solución completa.

Criterios. Cobertura semántica para respuestas abiertas; criterios estáticos (nombres, entradas/salidas, control de flujo) para prácticas.

Fragmento ilustrativo (JSON esperado).

```
{
  "score_pct": 0-100,
  "diagnostico": "... (dónde falla y por qué)",
  "sugerencia": "... (pista o alternativa sin dar la solución completa)"
}
```

La salida exclusiva en JSON minimiza verborrea y facilita el procesado automático en la plataforma.

C.4. Integración con recuperación (*RAG*)

Antes de generar o evaluar, el modelo recibe fragmentos relevantes del material limpio (evidencia). RAG introduce una memoria no paramétrica que mejora pertinencia y reduce alucinaciones, además de permitir actualizar contenidos sin reentrenar [23, 42, 9].

Efecto observado.

1. **Sin RAG.** Mayor deriva entre ejecuciones y mayor tasa de afirmaciones inventadas.
2. **Con RAG + limpieza.** Enunciados más fieles al temario y menor dispersión, especialmente en ejercicios teóricos.

C.5. Prácticas operativas

- **Especificidad de rol y público.** Instrucciones enmarcadas como "herramienta para *Fundamentos de Programación* (Java, enfoque imperativo)", con nivel de curso explícito.
- **Ejemplificación mínima.** Uno o dos ejemplos breves (cuando procede) mejoran estabilidad sin "anclar" en exceso [24].
- **Instrucciones negativas claras.** "No revelar la solución completa", "no usar opción múltiple en teórico".
- **Control de formato.** Esquema JSON declarado en el propio *prompt*; validación posterior en servidor.
- **Cierre de longitud.** Límites razonables en explicaciones para promover concisión y facilitar la lectura.

C.6. Generalización a otras asignaturas

El flujo *clean* → *generate* → *grade* es independiente del dominio. Cambiando instrucciones de ámbito y criterios de corrección, y proporcionando el corpus correspondiente a RAG, la misma estrategia resulta aplicable a otras materias (bases de datos, estructuras de datos, etc.), conservando trazabilidad y control de formato [42, 9].

Conclusión

La combinación de **calidad de entrada** (limpieza/síntesis), **prompting estructurado** (tarea, restricciones y esquema) y **RAG** ha sido determinante para estabilizar la generación y reducir alucinaciones en este proyecto. Este *stack* ofrece una base sólida y extensible para iteraciones futuras centradas en mejorar la alineación curricular y la transparencia de la evaluación.

Bibliografía

[1] Bennedsen, J., and Caspersen, M. E. Failure rates in introductory programming. *ACM SIGCSE Bulletin 39*, 2 (2007), 32–36.

[2] Bennedsen, J., and Caspersen, M. E. Failure rates in introductory programming: 12 years later. *ACM Inroads 10*, 2 (2019), 30–36.

[3] Chen, M., Tworek, J., Jun, H., et al. Evaluating large language models trained on code. *arXiv 2107.03374* (2021).

[4] Chudziak, J. A., and Kostka, A. AI-Powered Math Tutoring: Platform for Personalized and Adaptive Education. *arXiv preprint arXiv:2507.12484* (2025). Open-source code available at https://github.com/feilaz/AI_Powered_Math_Tutoring.

[5] Cloudflare. Cloudflare tunnel documentation. https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/, 2025.

[6] Cloudflare. Timeouts and limits for cloudflare proxies. https://developers.cloudflare.com/support/troubleshooting/cloudflare-settings/maximum-upload-size-timeouts-and-other-limits/, 2025.

[7] Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. arXiv:2305.14314, 2023.

[8] Ellaithy, A. Langchain-powered youtube transcript quiz generator with mistral llm. https://github.com/abdallaellaithy/LangChain-and-Mistral-LLM-Powered-YouTube-Quiz-Generator, 2024. GitHub repository.

[9] Fan, Y., He, J., Chen, D., et al. Retrieval-augmented generation for large language models: A survey. *ACM Computing Surveys* (2024).

[10] Glassdoor. Junior software engineer salaries in madrid, spain. https://www.glassdoor.com/Salaries/

madrid-spain-junior-software-engineer-salary-SRCH_IL.0,12_IM1030_KO13,37.htm, 2025.

[11] Hattie, J., and Timperley, H. The power of feedback. *Review of Educational Research 77*, 1 (2007), 81–112.

[12] Henkel, M., Limniotis, K., Szajda, I., Horlebein, L., Rogalla, M., Herder, E., and Lingnau, A. Large language models for mathematics education: A case for rag. In *Proceedings of the 17th International Conference on Educational Data Mining (EDM 2024), Short Paper* (Rennes, France, 2024).

[13] Hu, E. J., Shen, Y., Wallis, P., et al. Lora: Low-rank adaptation of large language models. arXiv:2106.09685, 2021.

[14] HuffPost. Esto es lo que cobra un programador en españa en 2024. https://www.huffingtonpost.es/sociedad/esto-cobra-programador-espana-2024.html, 2024.

[15] Jaiswal, G. Mcq generator using langchain and openai. https://github.com/Gaurav-Jaiswal-1/MCQ-Generator-Using-Langchain-and-OpenAI, 2024. GitHub repository.

[16] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Madotto, A., and Fung, P. Survey of hallucination in natural language generation. *ACM Computing Surveys* (2023).

[17] Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W. Dense passage retrieval for open-domain question answering. In *EMNLP 2020* (2020).

[18] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences 103* (2023), 102274.

[19] Kazemitabaar, M., Zhang, L., Salman, H., et al. Codeaid: Clarifying student intent in the presence of ai-powered code assistants. In *CHI 2024* (2024).

[20] LangChain. Ollama integration in langchain. https://python.langchain.com/docs/integrations/llms/ollama, 2025.

[21] LangChain. Run and manage models locally. https://python.langchain.com/docs/integrations/platforms/local, 2025.

[22] Laravel. Laravel documentation. https://laravel.com/docs, 2025.

[23] Lewis, P., Perez, E., Piktus, A., et al. Retrieval-augmented generation for knowledge-intensive nlp. In *Advances in Neural Information Processing Systems (NeurIPS 2020)* (2020).

[24] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys 55*, 9 (2023), 1–35.

[25] Lobb, R., and Harlow, J. Coderunner: A tool for assessing computer programming skills. *ACM Inroads 7*, 1 (2016), 47–51.

[26] Lozhkov, L., Allal, L. B., et al. Starcoder2 and the stack v2: The next generation. arXiv:2402.19173, 2024.

[27] Messer, V., Böttinger, K., Albrecht, A., et al. Automated grading for programming education: A review. *ACM Computing Surveys* (2024).

[28] NVIDIA. Nvidia vgpu software documentation. https://docs.nvidia.com/grid/index.html, 2024. Perfiles vGPU y asignación de VRAM.

[29] Ollama. Ollama documentation. https://ollama.com, 2025.

[30] Ollama. Server options: Ollama_keep_alive. https://github.com/ollama/ollama/blob/main/docs/env.md, 2025.

[31] Project Management Institute. A guide to the project management body of knowledge (pmbok®guide), 7th ed. — risk principles overview. https://www.pmi.org/pmbok-guide-standards/foundational/pmbok, 2021.

[32] Ramírez, S. Fastapi documentation. https://fastapi.tiangolo.com/, 2025.

[33] Rozière, B., et al. Code llama: Open foundation models for code. arXiv:2308.12950, 2023.

[34] Schwaber, K., and Sutherland, J. The scrum guide (2017 update). https://www.scruminc.com/wp-content/uploads/2016/01/2017-Scrum-Guide-US.pdf, 2017.

[35] Schwaber, K., and Sutherland, J. The scrum guide (2020 update). https://scrumguides.org/scrum-guide.html, 2020.

[36] Slade, J. J., Hyk, A., and Gurung, R. A. R. Transforming learning: Assessing the efficacy of a retrieval-augmented generation system as a tutor for introductory psychology. In *Proceedings of the 68th International Annual Meeting of the HFES* (2024), vol. 68, pp. 1827–1830.

[37] Spatie. Laravel permission. https://spatie.be/docs/laravel-permission, 2025.

[38] Tekman Education. thinköai: Inteligencia artificial diseñada por y para docentes. https://www.thinkoai.com, 2023. Plataforma web de generación de contenido educativo con IA.

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS 2017)* (2017).

[40] Watson, C., and Li, F. W. B. Failure rates in introductory programming revisited. In *Proceedings of ITiCSE 2014* (2014), pp. 39–44.

[41] White, J., Fu, Y., and Hu, Q. M. A prompt pattern catalog to enhance prompt engineering with chatgpt. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)* (2023), pp. 1–16.

[42] Zhao, W. X., Chen, J., Zhang, S., Fan, Z., Wang, X., et al. A survey of retrieval-augmented generation for large language models. arXiv:2402.19473, 2024.