



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un sistema para la evaluación
automática de cuestiones sobre arquitectura de
software

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Magaña Mascuñana, Miguel

Tutor/a: Pelechano Ferragud, Vicente

CURSO ACADÉMICO: 2022/2023

Resumen

Actualmente, en los procesos de selección para perfiles de nivel medio en el campo de la Ingeniería de Software, se suele incluir al menos una fase dedicada a evaluar los conocimientos del candidato sobre arquitectura de software. Normalmente, el proceso suele requerir de la explicación y el diseño de un sistema para cierto escenario dadas unas premisas técnicas.

Una de las principales desventajas que se identifican en el proceso es que el resultado no es determinista, es decir, dos soluciones iguales pueden obtener calificaciones diferentes dependiendo del evaluador, entre otros factores, por tanto, no se puede garantizar la igualdad de condiciones entre los candidatos.

Otro de los problemas identificados es la gran cantidad de recursos técnicos que se destinan al proceso de entrevistar y evaluar al candidato.

Identificados los principales problemas se propone investigar y tratar de implementar una solución al problema de la calificación automática de este tipo de cuestiones. La aplicación de los últimos avances en inteligencia artificial y aprendizaje automático puede reducir o eliminar el componente humano en esta fase del proceso de selección y aumentar así la fiabilidad, eficiencia e igualdad en los procesos de selección, además de reducir los costes asociados a los recursos técnicos empleados en el proceso.

Palabras clave: Arquitectura de software, inteligencia artificial, aprendizaje automático, igualdad, procesos de selección.

Abstract

Nowadays, on most of Software Engineering interview processes for mid-level individuals and above there is at least a Software Architecture question included. This question usually requires the design and explanation of a System Architecture for a given scenario.

One of the main disadvantages of this process is that the result isn't deterministic, two identical solutions might be differently evaluated. Therefore, it's impossible to guarantee that the same terms and conditions apply to every candidate.

Another problem is the number of technical resources that the process requires, usually engineers that interview, and then evaluate the candidate.

Having the main problems identified, an investigation process will be carried out to implement a solution to automatically evaluate these questions. Applying state of the art artificial intelligence and machine learning technologies to reduce and possibly eliminate human participation in the evaluation process. This will increase the reliability, efficiency and fairness of the selection processes and will also reduce the technical resources' cost on the process.

Keywords: Software Architecture, artificial intelligence, machine learning, equality, selection processes.

Glosario

- **Arquitectura de software:** La arquitectura de software se refiere a la estructura y diseño de un programa o sistema informático. Define cómo los componentes del software interactúan entre sí y con el hardware.
- **Aprendizaje automático:** Es una rama de la inteligencia artificial que permite a las computadoras aprender y mejorar automáticamente a partir de datos y experiencias previas.
- **CDK (*Cloud Development Kit*):** Es una herramienta que permite a los desarrolladores definir infraestructura en la nube utilizando lenguajes de programación familiares, como TypeScript o Python.
- **Chatbot conversacional:** Un chatbot conversacional es un programa informático diseñado para interactuar con usuarios a través de conversaciones naturales. Puede responder preguntas y llevar a cabo tareas automatizadas.
- **Clave de API:** Una clave de API es una cadena de caracteres utilizada para autenticar y acceder a servicios web y APIs. Funciona como una contraseña para la comunicación entre aplicaciones.
- **Clustering:** Se refiere a la técnica de agrupar datos similares en conjuntos o clusters para su análisis. Es común en estadísticas y aprendizaje automático.
- **Dirección URL:** Una dirección URL es una cadena de caracteres que se utiliza para localizar recursos en la web, como sitios web o archivos. Es lo que escribimos en la barra de direcciones del navegador.
- **Escalado horizontal:** Es el proceso de aumentar la capacidad de un sistema distribuyendo la carga entre múltiples servidores o máquinas. Se agrega más hardware para manejar la carga.
- **Escalado vertical:** Se refiere a aumentar la capacidad de un sistema mejorando los recursos de una única máquina, como agregar más RAM o CPU.
- **Explicabilidad (IA):** La explicabilidad en inteligencia artificial se refiere a la capacidad de comprender y explicar cómo un algoritmo o modelo de IA toma decisiones. Es importante para entender el razonamiento detrás de las predicciones de la IA.
- **Handler:** En programación, un "handler" es una función o rutina que se utiliza para manejar eventos o solicitudes específicas.
- **HTTP (*Hypertext Transfer Protocol*):** Es el protocolo utilizado para la transferencia de datos en la web. Define cómo los mensajes se formatean y transmiten entre servidores y navegadores web.
- **Inteligencia artificial:** La inteligencia artificial se refiere a la capacidad de las computadoras para realizar tareas que normalmente requieren inteligencia humana, como el aprendizaje, la percepción y la toma de decisiones.
- **Ingeniería de software:** La ingeniería de software es la disciplina que se ocupa del diseño, desarrollo y mantenimiento de sistemas de software de manera sistemática y eficiente.
- **JSON (*JavaScript Object Notation*):** JSON es un formato ligero de intercambio de datos que se utiliza para representar información estructurada en forma de objetos y listas.
- **Modelo generativo pre-entrenado (GPT):** Se refiere a un modelo de inteligencia artificial que ha sido entrenado previamente en un gran conjunto de datos y se puede ajustar o refinar para tareas específicas.



Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

- **Modelos de aprendizaje:** Los modelos de aprendizaje se refieren a las representaciones matemáticas utilizadas en el aprendizaje automático para realizar predicciones o tomar decisiones basadas en datos.
- **Procesamiento de lenguaje natural (NLP):** Es una rama de la inteligencia artificial que se ocupa de la interacción entre las computadoras y el lenguaje humano, permitiendo a las máquinas entender, interpretar y generar lenguaje humano.
- **Prompt:** En inteligencia artificial, un "prompt" es una solicitud o entrada que se le da a un modelo de lenguaje para generar una respuesta o texto continuo.
- **Prompt engineering:** Se refiere al proceso de diseñar y ajustar cuidadosamente los *prompts* para obtener respuestas específicas de modelos de lenguaje, como GPT-3 o Bard.
- **RESTful (Representational State Transfer):** Es un estilo arquitectónico para el diseño de sistemas web que se basa en representar recursos como URLs y utilizar métodos HTTP para interactuar con ellos.
- **XML (Extensible Markup Language):** XML es un lenguaje de marcado que se utiliza para estructurar datos de manera legible tanto por humanos como por máquinas. Se usa comúnmente para intercambio de datos.

Índice de figuras

Figura 1. Distribución estimada de tiempos fase de investigación.	13
Figura 2. Distribución estimada de tiempos fase de diseño de la solución.....	14
Figura 3. Ventajas y desventajas del EDD.	14
Figura 4. Principales ventajas y desventajas de utilizar TDD	16
Figura 5. Estimación de tiempos fase de implementación.....	16
Figura 6. Predicción del valor de mercado de la IA [2].	19
Figura 7. Porcentaje de puestos de trabajos en riesgo por la automatización [3].....	20
Figura 8. ATAM pasos y distintas fases. [13]	25
Figura 9. Ventajas y desventajas del ISO/IEC 42010.....	27
Figura 10. Metodología de evaluación propuesta.....	27
Figura 11. Comparación de las API de GPT-4 y Bard [15].	29
Figura 12. Ejemplo de diagrama de arquitectura de software.....	30
Figura 13. Diagrama de la solución propuesta	31
Figura 14. Diagrama de contexto del sistema diseñado.	39
Figura 15. Diagrama de contenedores del sistema diseñado.	41
Figura 16. Diagrama de componentes del sistema diseñado.	43
Figura 17. Ejemplo de diagrama de código inicial del sistema diseñado.	44
Figura 18. Diseño de la API en Swagger.	45
Figura 19. Código encargado del despliegue de la infraestructura en la nube.	48
Figura 20. Script en Node para desplegar la infraestructura en la nube.	49
Figura 21. Ejemplo de código de un handler en el contexto de una Lambda de AWS. ...	50
Figura 22. Ejemplo de casos de prueba con Postman.	52
Figura 23. Ejemplo de grafo descrito en GraphSON [25].	54
Figura 24. Parámetros a evaluar de un diagrama de arquitectura de software.	59
Figura 25. Escala de evaluación utilizada en el sistema.	59
Figura 26. Código del generador de consultas.	60
Figura 27. Ejemplo de caso de prueba sencillo.	64
Figura 28. Solución propuesta por el candidato.	64
Figura 29. Resultado de la evaluación de una arquitectura correcta.	65
Figura 30. Diagrama deficiente para evaluar un ingeniero Junior	66
Figura 31. Ejemplo de evaluación de un diagrama deficiente para un ingeniero junior.	67
Figura 32. Diagrama mejorado para evaluar un ingeniero Junior.	67
Figura 33. Evaluación de un diagrama mejorado para un ingeniero Junior.	68
Figura 34. Caso de prueba para evaluación de un candidato de nivel intermedio.	69
Figura 35. Diagrama de ejemplo para la evaluación de un candidato de nivel intermedio.	69
Figura 36. Evaluación del sistema proporcionado para un ingeniero de nivel intermedio.	71
Figura 37. Diagrama complejo proporcionado para evaluar las capacidades de la herramienta [26].	73
Figura 38. Evaluación del sistema para un diagrama de arquitectura más complejo. ...	75



Tabla de contenidos

1	Introducción	9
1.1	Motivación	9
1.2	Objetivos	10
1.3	Impacto esperado	11
1.4	Metodología.....	12
1.5	Estructura	17
2	Contexto tecnológico	19
3	Análisis e investigación	23
3.1	Evaluación de arquitecturas de software.....	23
3.2	Análisis de las herramientas actuales	28
3.3	Solución propuesta	31
4	Requisitos del sistema	33
4.1	Usuarios del sistema	33
4.2	Funciones del Sistema	33
4.3	Restricciones.....	34
4.4	Suposiciones y dependencias.....	34
4.5	Requisitos de rendimiento.....	34
4.6	Atributos del sistema	34
5	Diseño del sistema	37
5.1	Diagrama de Contexto	37
5.2	Diagrama de contenedores	39
5.3	Diagrama de componentes	42
5.4	Diagrama de código	44
5.5	Diseño de la API.....	45
6	Fase de implementación	47
6.1	Tecnologías utilizadas.....	47
6.2	Desafíos presentados	53
7	Pruebas	63
7.1	Evaluación del diagrama de un sistema simple.....	64
7.2	Evaluación de un diagrama de complejidad intermedia	68
7.3	Evaluación del diagrama de un sistema complejo.....	72
8	Relación con los estudios de máster	77
9	Conclusiones	79
10	Futuras mejoras.....	81



11	Referencias	83
12	Anexo	85
	I. Cuestiones utilizadas en las pruebas de aceptación para ingenieros Junior	85
	II. Ejemplo de <i>prompt</i> generado para la evaluación de ingenieros Junior	86
	III. Cuestiones utilizadas para la evaluación de un ingeniero de software de nivel intermedio.....	87
	IV. <i>Prompt</i> generado para evaluar a un candidato de nivel intermedio	88
	V. Consulta generada para evaluación de un sistema complejo	89
	VI. Código de especificación de la API.....	92
	VII. Objetivos de desarrollo sostenible.....	95

1 Introducción

En un mundo donde la tecnología avanza a una velocidad abrumante, las empresas e instituciones han de invertir en mejorar los procesos para que sean más ágiles y se adapten a los nuevos tiempos, donde cada vez más, los usuarios están acostumbrados a la rapidez y a la agilidad que ofrece la tecnología.

En el sector del desarrollo de software, en la mayoría de los casos, no se utilizan las últimas tecnologías en el proceso de selección, o se utilizan mínimamente.

En los casos en los que se utilizan estas tecnologías, suelen hacerse para evaluar las aptitudes técnicas del candidato de una manera práctica, suelen incluir pruebas de código, algorítmicas o de proyectos reales, y en los casos donde el candidato tiene un nivel más avanzado se incluye una prueba de diseño de sistemas.

Las pruebas de código se pueden evaluar automáticamente haciendo uso de casos de prueba unitarios, funcionales, cumpliendo criterios de aceptación, utilizando herramientas para analizar la calidad del código, etc.

Para las pruebas de diseño de sistemas la evaluación es más compleja, ya que, al ser una cuestión de respuesta abierta, cada candidato puede diseñar distintos sistemas que sean correctos dependiendo de las premisas técnicas y los requisitos que se hayan especificado, por tanto, se requiere la intervención humana en el proceso de evaluación.

Este proyecto proporcionará una forma de evaluar automáticamente este tipo de cuestiones garantizando la objetividad y la igualdad en la selección de personal, además de reducir los recursos destinados al proceso de selección.

1.1 Motivación

Como ingeniero de software, he hecho tanto de candidato como de entrevistador en los procesos de selección numerosas veces. Desde hace unos años, he participado activamente como entrevistador para perfiles de distintos niveles y evaluando distintas aptitudes.

Durante todo este tiempo, la mayor dificultad a la que me he enfrentado como entrevistador es conseguir evaluar el trabajo de un candidato de una forma objetiva, es decir, intentar no comparar el resultado de la entrevista con otras evaluaciones anteriores.

Adicionalmente, en el proceso de evaluación, dependiendo de la complejidad de la entrevista se suele tardar de entre diez a treinta minutos en calificar el resultado y en reflejarlo en algún medio digital para su posterior consulta, a este tiempo hay que sumarle el tiempo que dure la entrevista donde se interactúa con el candidato. Cuando se contabiliza el tiempo que los recursos técnicos dedican a este proceso, teniendo en cuenta el tamaño de la compañía, normalmente es muy superior a lo que se suele pensar, y, por tanto, se encuentran con que sus recursos técnicos dedican más capacidad de la que deberían a este proceso.

En la mayoría de las compañías en las que he entrevistado o he sido entrevistado, me ha sorprendido la falta de un proceso de evaluación estandarizado, en los mejores casos, para estas cuestiones, se utilizan rúbricas que definen unos aspectos a evaluar con unos rangos de puntuación para cada aspecto dependiendo de ciertos requisitos.

Estas rúbricas son aplicadas por el entrevistador, por tanto, el componente humano no garantiza que vayan a ser aplicadas de la misma forma dos veces, por muy profesional que sea el entrevistador. Este proyecto pretende desafiar a los procesos tradicionales e implementar una forma de evaluación completamente automatizada para este tipo de cuestiones utilizando la inteligencia artificial (IA) y el aprendizaje automático (AA).

Personalmente, considero que se ha infravalorado el potencial de la inteligencia artificial durante mucho tiempo. Con este proyecto quiero explorar el potencial de la inteligencia artificial actual utilizándola para resolver un problema real, además de explorar hasta qué punto la inteligencia artificial puede ayudarnos a automatizar procesos que, hasta ahora, se consideraban demasiado complejos.

1.2 Objetivos

Actualmente, la totalidad de las pruebas de diseño de software que he realizado tanto en mi puesto actual como las que realizan nuestros clientes, se evalúan de forma manual, siguiendo unos criterios de calificación proporcionados en rúbricas de evaluación. En consecuencia, el criterio del entrevistador tiene un papel muy importante en la calificación del resultado.

Haciendo uso de datos internos de la compañía, se ha determinado que el tiempo empleado en cada entrevista de diseño de software varía de entre cuarenta y cinco minutos a varios días dependiendo de la complejidad del problema presentado. Consecuentemente, el posterior proceso de evaluación suele variar de entre diez minutos a varias horas.

A continuación, se listan los objetivos principales del proyecto.

- Diseñar e implementar una solución automatizada para evaluar y calificar los conocimientos de arquitectura de software de los candidatos en procesos de selección de Ingeniería de Software.
- Aplicar técnicas de inteligencia artificial y aprendizaje automático para reducir o eliminar la subjetividad en la calificación de las soluciones propuestas por los candidatos.
- Mejorar la igualdad de condiciones entre los candidatos al eliminar posibles sesgos humanos en el proceso de evaluación.
- Aumentar la fiabilidad y eficiencia del proceso de selección al utilizar una solución automatizada, minimizando así la variabilidad en los resultados y reduciendo el tiempo requerido para evaluar a cada candidato.
- Reducir los costes asociados a los recursos técnicos utilizados en el proceso de selección al sustituir o complementar la evaluación humana con una herramienta automatizada.

Los siguientes puntos se consideran importantes en cuanto a tratar de cumplirlos, pero su cumplimiento no está totalmente garantizado, por tanto, se consideran secundarios:

- Investigar y analizar las técnicas existentes en inteligencia artificial y aprendizaje automático para evaluar soluciones de arquitectura de software.
- Validar y evaluar la precisión y efectividad de la solución automatizada mediante pruebas y comparaciones con la evaluación tradicional.
- Identificar las limitaciones y desafíos en la implementación de dicha solución automatizada y proponer soluciones o mejoras para superarlos.
- Evaluar el impacto y la aceptación de la solución automatizada en el proceso de selección, tanto desde la perspectiva de los candidatos como de los evaluadores y la organización.

En el siguiente apartado, se detalla el impacto esperado tras el desarrollo del proyecto y su implementación.

1.3 Impacto esperado

Se espera que la implementación del proyecto tenga un impacto positivo en el proceso de selección de perfiles de ingeniería de software, a continuación, se detalla el impacto que se espera que el proyecto tenga sobre distintas áreas.

Se espera un aumento en la precisión y consistencia de las evaluaciones, al utilizar técnicas de inteligencia artificial y aprendizaje automático, se espera reducir la subjetividad y la variabilidad en la calificación de las soluciones propuestas por los candidatos. Por lo tanto, conlleva una evaluación más precisa y consistente de las habilidades y conocimientos de arquitectura de software.

Adicionalmente, se espera mejorar la eficiencia, el proyecto permitirá acelerar el proceso de selección, lo que resulta en una mayor eficiencia. Como consecuencia, se espera reducir los costes del proceso debido a la automatización de la fase de evaluación, al reducir o eliminar la necesidad de recursos técnicos humanos para esta fase, se reducen los costes asociados al proceso de selección.

Implementar una solución automatizada permite eliminar posibles sesgos humanos, se espera garantizar la igualdad de condiciones entre los candidatos. Se espera garantizar que cada candidato pueda ser evaluado de manera objetiva y justa, independientemente de factores como el evaluador, la experiencia del candidato, el *curriculum*, o el contexto de la evaluación.

Además, mediante la aplicación de técnicas de IA y AA se espera identificar patrones y características en las soluciones propuestas por los candidatos que pudieran pasar desapercibidas en la evaluación humana para descubrir habilidades que no son habituales y que pudieran ser relevantes al puesto de trabajo.

Para finalizar, se espera que, al tener un sistema de evaluación objetivo y único, la toma de decisiones a la hora de contratar un perfil sea más sencilla, por tanto, se ayude en la toma de



decisiones al conocer que todos los candidatos han seguido exactamente el mismo proceso de evaluación, con el mismo margen de error.

No se espera que la solución automatizada reemplace por completo al perfil del entrevistador, ya que esta solución no sabrá medir las habilidades blandas como la comprensión, el liderazgo, la capacidad de pensar en distintas soluciones, etc.

En resumen, se espera que la implementación de esta solución automatizada mejore la calidad y la eficacia del proceso de selección, beneficiando tanto a los candidatos como a las organizaciones que buscan perfiles en el campo de la ingeniería de software.

1.4 Metodología

Para abordar el proyecto de una forma estructurada y eficiente, se ha decidido dividirlo en tres fases.

En la primera fase, se analiza el problema desde un punto de vista técnico, se estudian y analizan las tecnologías actuales, incluyendo los últimos avances en inteligencia artificial y se plantean distintas soluciones que pudieran ser adecuadas para resolver cada problema.

En la segunda fase, una vez concluido el análisis inicial, se procede a evaluar una o varias soluciones al problema propuesto y se considera la mejor solución, proponiendo un diseño conceptual.

A continuación, en la última fase, una vez seleccionada la solución más adecuada para la resolución del problema presentado, se define un producto mínimo viable con el objetivo de validar dicha solución.

1.4.1 Análisis e investigación

En esta primera fase, el objetivo principal es comprender los últimos avances en inteligencia artificial y aprendizaje automático para evaluar su posible aplicación a la resolución total o parcial del problema expuesto.

En esta fase de investigación se pretende dar respuesta a las siguientes preguntas:

- ¿Qué aspectos se necesitan evaluar de una arquitectura de software?
- ¿Qué metodología se sigue para evaluar este tipo de cuestiones?
- ¿Cómo puede un sistema interpretar un diseño de arquitectura?
- ¿Existen herramientas capaces de evaluar automáticamente un diseño de arquitectura de software?
- ¿Se puede implementar un sistema que interprete y evalúe una arquitectura de software de forma automática?
- ¿Qué limitaciones o riesgos técnicos puede presentar el sistema diseñado?

Se estima que esta primera fase puede durar un total de tres a cuatro semanas, la distribución de tiempos para las distintas etapas de esta fase se puede observar en la siguiente figura.

Figura 1. Distribución estimada de tiempos fase de investigación.

	1-mayo	8-mayo	15-mayo	21-mayo
Análisis inicial				
Investigación para determinar la solución a cada problema				
Viabilidad de la solución global				

En la primera fase de análisis inicial, se plantea de forma técnica que problema se necesita resolver, dividiendo el problema de la evaluación automática en pequeños subproblemas, con el objetivo de intentar resolver cada problema individualmente.

En la segunda fase, una vez identificados los problemas que se han de resolver, se lleva a cabo una investigación para determinar cómo solucionar cada problema de la mejor manera posible. En esta fase, se debe intentar recurrir a herramientas o soluciones en el contexto individual de cada problema, ya que no se desea invertir demasiado esfuerzo en solucionar un problema que ya se haya resuelto con anterioridad.

A continuación, una vez planteadas las posibles soluciones, se procede a evaluar la viabilidad de obtener una solución para el problema global y se documentan las respuestas a cada pregunta planteada.

Una vez finalizada esta fase, se puede diseñar una solución viable para el problema presentado.

1.4.2 Diseño de la solución

Una vez concluida la fase de análisis e investigación se espera tener la información y el conocimiento suficiente sobre cómo se puede diseñar un sistema viable para dar solución al problema planteado.

Los objetivos específicos de esta fase del proyecto son los siguientes:

- Definir a grandes rasgos que requisitos debe cumplir el sistema.
- Proporcionar un diseño conceptual concreto del sistema.

Para definir los requisitos que debe cumplir el proyecto se ha pensado utilizar el estándar IEEE 830 ^[5], es un estándar que sirve como guía para documentar de manera clara y precisa los objetivos, funciones y características de un sistema de software, pero se considera demasiado completo para este proyecto, ya que la complejidad del proyecto reside en la investigación y el diseño más que en la implementación.

Para definir el diseño conceptual del sistema se utiliza el modelo C4 ^[6], es un enfoque de modelado arquitectónico que se utiliza para representar visualmente la arquitectura de software. Se basa en cuatro niveles de abstracción: Contexto, Contenedores, Componentes y Código.



Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

Utilizando metodologías estándar se busca proporcionar una base sólida para la comunicación y comprensión entre los interesados y los desarrolladores del software.

A continuación, se presenta la estimación iniciales estimados para esta fase.

Figura 2. Distribución estimada de tiempos fase de diseño de la solución

	27-mayo	3-junio	10-junio	17-junio
Establecer requisitos				
Diseño inicial del sistema utilizando C4				

Cabe destacar que como resultado de esta fase no se esperan obtener documentos inmutables. Durante el desarrollo del proyecto existe la posibilidad de que algunos requisitos deban cambiar, se tomen decisiones para cambiar la arquitectura del sistema o la implementación, y, por tanto, se necesiten cambios en el diseño presentado o en los requisitos.

En cuanto al diseño inicial del sistema, en esta fase, solamente se esperan obtener los diagramas más generales del modelo C4.

Para esta fase, se explora la opción de utilizar Event-Driven Design (EDD), es un enfoque de diseño de sistemas en el que la estructura y la interacción de los componentes se definen en torno a eventos y las acciones que se desencadenan como respuesta.

A diferencia de los enfoques más tradicionales basados en el flujo de control, en el diseño basado en eventos, el sistema responde a eventos externos o internos de manera asincrónica y desencadena acciones específicas en función de ellos.

Figura 3. Ventajas y desventajas del EDD.

Ventajas	Desventajas
Flexibilidad y adaptabilidad: El diseño basado en eventos permite una mayor flexibilidad y adaptabilidad en el sistema, ya que los componentes pueden responder dinámicamente a eventos y condiciones cambiantes.	Complejidad: El diseño basado en eventos puede aumentar la complejidad del sistema, especialmente cuando hay una gran cantidad de eventos y una interacción compleja entre los componentes. La comprensión de la propagación de eventos y la gestión adecuada puede requerir una planificación cuidadosa y una mayor experiencia en diseño.
Modularidad: El diseño basado en eventos fomenta la modularidad y la separación de preocupaciones. Los componentes del sistema pueden desarrollarse y probarse de forma independiente, lo que facilita la colaboración y la evolución del proyecto de investigación.	Curva de aprendizaje: El diseño basado en eventos requiere una curva de aprendizaje más pronunciada para el equipo de desarrollo, especialmente si no se está familiarizado con este enfoque. Comprender los conceptos y las mejores prácticas del EDD puede llevar tiempo y esfuerzo adicional.
Reutilización: Al diseñar componentes que se comunican mediante eventos, se fomenta la reutilización. Los componentes pueden ser utilizados en diferentes contextos y proyectos,	Depuración: La depuración puede ser más complicada debido a la naturaleza asincrónica y distribuida de las interacciones entre los componentes. Detectar y solucionar

lo que ahorra tiempo y esfuerzo en el desarrollo.	problemas de lógica y flujo de eventos requiere herramientas y técnicas específicas.
---	--

En conclusión, utilizar EDD en la fase de diseño, puede resultar beneficioso en el proyecto de investigación debido principalmente a la flexibilidad y extensibilidad. Sin embargo, también puede introducir complejidad y requerir una planificación cuidadosa, especialmente en sistemas con interacciones complejas.

1.4.3 Implementación

Una vez completada la fase de diseño, se obtiene el documento de especificación de requisitos (ERS) y el diseño conceptual de la arquitectura del sistema. En el ERS se detallan los requisitos que debe de cumplir el sistema, por tanto, se puede comenzar la fase de implementación teniendo en cuenta los requisitos que debe cumplir el producto desarrollado.

En la fase de implementación también se cuenta con un plan de pruebas y validación que permiten determinar si el sistema cumple con los requisitos establecidos.

La arquitectura del sistema dependerá de los resultados obtenidos en la primera fase de análisis, pero inicialmente se pretende utilizar TypeScript o Python para desarrollar un servicio alojado en la nube que sea capaz de aceptar peticiones y responder en un tiempo razonable.

En esta fase se valora la utilización de Test-Driven Development (TDD), una práctica de desarrollo en la que se escriben pruebas automatizadas antes de escribir el código de producción. El ciclo de desarrollo en TDD consta de tres fases: fase prueba (Red), fase de implementación (Green) y fase de refactorización.

En la siguiente figura se muestra una comparativa con las principales ventajas y desventajas de utilizar TDD en la fase de implementación.



Figura 4. Principales ventajas y desventajas de utilizar TDD

Ventajas	Desventajas
Código fiable: Las pruebas en TDD actúan como una garantía de que el código funciona como se espera y ayuda a reducir la introducción de errores.	Mayor esfuerzo y tiempo inicial: Escribir pruebas antes de escribir el código puede llevar más tiempo y esfuerzo inicialmente.
Diseño modular y bajo acoplamiento: TDD fomenta el diseño de código modular y desacoplado, lo que facilita la mantenibilidad y la reutilización.	Requiere habilidades de diseño de pruebas: Es necesario tener habilidades para diseñar pruebas efectivas, lo que puede ser un desafío para los desarrolladores menos experimentados.
Mejora de la productividad: Aunque puede llevar más tiempo inicialmente, TDD puede mejorar la productividad a largo plazo al reducir el tiempo dedicado a la depuración y corrección de errores.	Puede limitar la flexibilidad: En algunos casos, el enfoque de TDD puede limitar la flexibilidad para experimentar con diferentes soluciones o cambios en los requisitos.

Se profundiza en el TDD y en sus fases de desarrollo en la correspondiente sección de la fase de implementación. En la siguiente figura se muestra la estimación de tiempos de la fase de implementación.

Figura 5. Estimación de tiempos fase de implementación.

	10-junio	17-junio	24-junio	1-julio	8-julio	-	-	-
Primer esprint								
Segundo esprint								
Tercer esprint								
Cuarto esprint								
Validación y pruebas								

Como se puede observar en la figura anterior, se ha seguido una metodología ágil de desarrollo por esprints. Se ha decidido configurar los esprints de una semana debido a la naturaleza del proyecto, con esto se espera poder actuar rápidamente frente a cambios o imprevistos, además de entregar valor cada esprint.

1.5 Estructura

En esta sección se describe a grandes rasgos la estructura general del documento, la memoria contiene un total de 11 puntos.

El primero de ellos es la **introducción**, donde se realiza una descripción del sistema que se desea desarrollar junto con la motivación profesional que ha generado la idea del producto. Se establecen los objetivos que se esperan cumplir con el desarrollo del proyecto junto con la metodología que se utiliza para garantizar la mejor ejecución posible de este.

En el segundo apartado, se incluye el **contexto tecnológico** en el que se desarrolla el proyecto, que es en el momento donde los grandes modelos de lenguaje se hacen públicos, en concreto, **GPT-3.5 y Bard**.

Seguidamente, en la tercera sección se presenta la fase de **análisis e investigación** donde se presentan las tres principales áreas de conocimiento técnico que se deben abordar para ser capaz de desarrollar y validar la idea: evaluación de arquitecturas software, análisis de las herramientas actuales y la solución propuesta.

A continuación, en la cuarta sección se presentan los **requisitos del sistema**, se ha utilizado como base el estándar IEEE 830/1998 simplificado para adecuarlo al tamaño del proyecto. Este apartado define los requisitos funcionales y no funcionales que debe de cumplir el sistema diseñado.

En el quinto apartado se presenta el **diseño del sistema** utilizando el modelo C4, mediante el uso del **modelo C4** se especifican cuatro diagramas con distintos niveles de abstracción destinados a público con distintos niveles de conocimiento técnico y, por tanto, presentan una visión completa del sistema. Esta sección finaliza mostrando el diseño de la API desarrollada utilizando el estándar **OpenAPI 3.0**.

La sexta sección se corresponde con la fase de **implementación**, donde se presentan las distintas tecnologías utilizadas a lo largo del desarrollo del proyecto. Se han nombrado las tecnologías más relevantes ya que a lo largo del proyecto se han utilizado numerosas tecnologías y no se le puede dedicar un apartado a cada una de ellas. Además, se detallan las principales barreras técnicas que se han encontrado a lo largo del desarrollo del proyecto y como se han superado.

En el siguiente apartado, se presentan las **pruebas** del sistema, estas pruebas son cruciales para evaluar la validez del sistema desarrollado y decidir si se cumplen los requisitos especificados. Se han hecho pruebas dependiendo del nivel de candidato, siguiendo un proceso iterativo y aumentando de dificultad en cada fase del proceso.

El octavo punto describe la **relación** que existe entre los **estudios del máster con el proyecto** realizado, se hace una mención a las principales asignaturas y competencias transversales que se han utilizado en el desarrollo del proyecto.

La novena sección se exponen las **conclusiones** del proyecto, se hace un repaso de los objetivos definidos para el proyecto y se menciona si se consideran cumplidos o no. Además, se incluye una valoración sobre las características del proyecto.

En la penúltima sección, se detallan las **futuras mejoras** que se han identificado una vez finalizado el proyecto y habiendo hecho pruebas de rendimiento. Además, se realizan consideraciones sobre si es preciso sustituir la herramienta utilizada por otra de mejor rendimiento.

Para finalizar, se muestran las **referencias bibliográficas** del proyecto, utilizando el estándar UNE-ISO 690 para las referencias bibliográficas siguiendo los formatos establecidos en [18].

2 Contexto tecnológico

En este apartado se presenta el contexto tecnológico en el que se aborda el proyecto. Se realiza un breve análisis de lo que ha supuesto la IA y el AA en los últimos tiempos y se dan ejemplos de aplicación en el mundo real y los desafíos a los que se enfrentan actualmente.

Los últimos avances en inteligencia artificial y el aprendizaje automático han causado un impacto significativo tanto en el campo de la ingeniería de software como en otros sectores. Estas disciplinas han experimentado un rápido avance y han despertado un gran interés debido a su potencial para revolucionar distintas industrias y sectores no necesariamente digitalizados.

En este punto cabe diferenciar entre la IA, que se refiere a la capacidad de las máquinas para realizar tareas que normalmente requieren rasgos de la inteligencia humana, como el razonamiento, la percepción y la toma de decisiones. Por otro lado, el AA se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender y mejorar a partir de los datos disponibles.

Actualmente, la IA y el AA han encontrado aplicaciones en diversos sectores, por ejemplo:

- En la industria médica, se han desarrollado sistemas de diagnóstico asistido por IA que son capaces de analizar grandes volúmenes de datos médicos y ayudar a los médicos a realizar diagnósticos más precisos y tempranos.
- En el sector financiero, los algoritmos de AA se utilizan ampliamente para el análisis de riesgos y la detección de fraudes, permitiendo a las instituciones financieras tomar mejores decisiones y mitigar los riesgos asociados con las transacciones y las inversiones.
- En el ámbito de la conducción autónoma, la IA desempeña un papel fundamental para desarrollar sistemas inteligentes capaces de tomar decisiones en tiempo real y garantizar la seguridad vial.

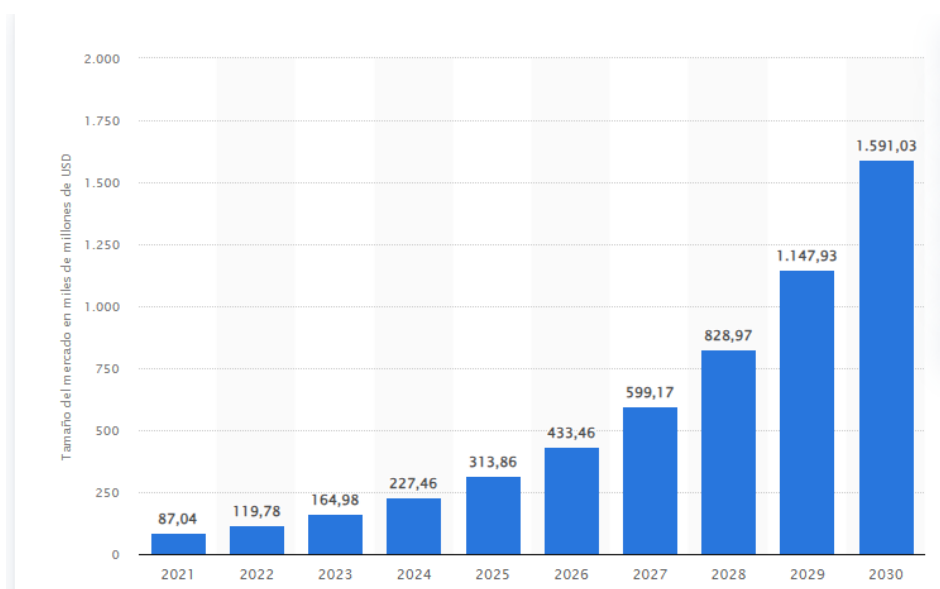


Figura 6. Predicción del valor de mercado de la IA [2].

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

En la figura anterior se muestra el crecimiento del valor de mercado de la IA desde 2021 hasta 2023 y se hace una predicción del valor que podría alcanzar en 2030. Se puede observar que se espera un crecimiento exponencial en los próximos años.

No obstante, a pesar de los avances logrados, la IA y el AA todavía se enfrentan a desafíos y problemas muy complejos. Uno de los principales desafíos es la dificultad para interpretar los modelos de IA, pues a medida que la IA avanza, los algoritmos se vuelven más complejos y se utilizan redes neuronales más profundas y complejas, por tanto, aumenta la dificultad para comprender el proceso de toma de decisiones y el razonamiento que se ha seguido para llegar a ciertas conclusiones.

Como consecuencia, especialmente en aplicaciones críticas como la medicina y la conducción autónoma, se plantean cuestiones éticas y de responsabilidad. En estos campos de aplicación, donde la transparencia y la explicabilidad de la IA son fundamentales para generar confianza en los usuarios y garantizar la seguridad.

Además, la privacidad y la ética son preocupaciones importantes en el desarrollo de sistemas de IA y AA. La recopilación y el uso de grandes cantidades de datos personales plantean desafíos de seguridad y protección de la privacidad. Es necesario establecer políticas y regulaciones sólidas para garantizar que los datos se utilicen de manera ética y responsable, respetando los derechos y la privacidad de los usuarios.

Otro desafío clave es abordar la falta de diversidad y los sesgos en los conjuntos de datos utilizados para entrenar los modelos de IA y AA. Si los datos utilizados para el entrenamiento están sesgados o carecen de diversidad, los modelos resultantes pueden generar resultados injustos o discriminatorios. Por lo tanto, es crucial fomentar la inclusión y la diversidad en la recopilación y el procesamiento de datos, así como en la formación de los equipos de desarrollo.

En cuanto al futuro de la IA, se prevé un continuo avance y un mayor impacto tanto en la sociedad como en la industria. Los avances en el procesamiento de grandes volúmenes de datos, la computación en la nube y la capacidad de cálculo permitirán el desarrollo de sistemas de IA más potentes y sofisticados.

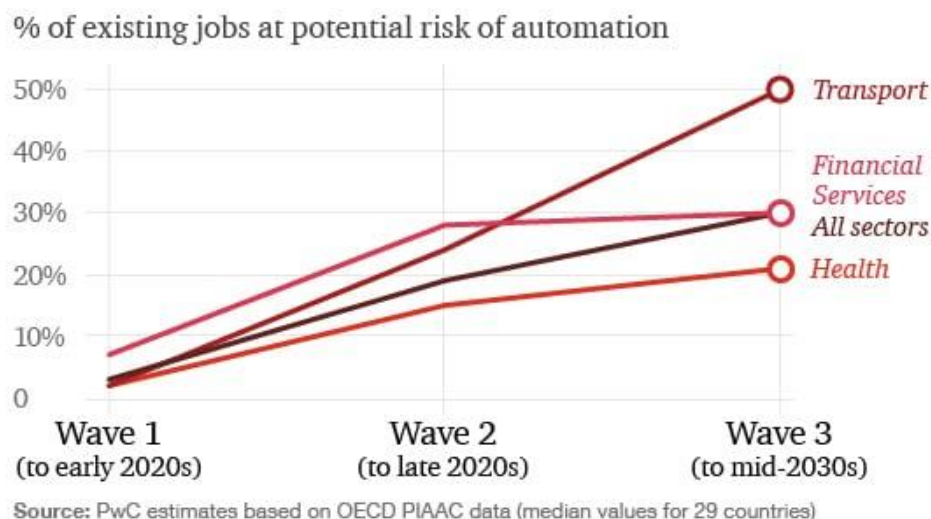


Figura 7. Porcentaje de puestos de trabajos en riesgo por la automatización [3].

Como se puede observar en la figura anterior, se espera que la IA tenga un impacto significativo en la automatización de tareas rutinarias y repetitivas, reduciendo carga a los profesionales para permitir que se enfoquen en tareas más creativas y estratégicas. Además, se espera que la IA juegue un papel clave en la mejora de la eficiencia y la productividad en una variedad de sectores, desde la fabricación y la logística hasta el cuidado de la salud y los servicios financieros.

En conclusión, se ha demostrado que la IA y el AA han alcanzado un estado del arte notorio en la ingeniería de software, con aplicaciones en diversas áreas del mundo real. A pesar de los desafíos actuales, se espera que la IA continúe evolucionando y transformando nuestra sociedad. Para ello, es fundamental abordar cuestiones como la interpretabilidad de los modelos, la privacidad, la ética y los sesgos en los datos para garantizar que la IA se desarrolle y aplique de manera responsable y equitativa. Teniendo en cuenta los avances continuos en las tecnologías y la colaboración entre la comunidad académica, las industrias y los responsables de la formulación de políticas, el futuro de la IA se presenta prometedor y lleno de oportunidades para impulsar la innovación y mejorar la calidad de vida.

3 Análisis e investigación

En esta **fase de análisis e investigación** se divide en tres fases distintas, con el objetivo de organizar la investigación y tener **resultados tangibles** en cada fase para ayudar a la toma de decisiones del proyecto.

En la primera fase, se investigan los **métodos actuales para evaluación de arquitecturas** de software, determinando los aspectos a evaluar y como evaluar un diagrama de arquitectura software.

Una vez conocidos **los aspectos que se han de evaluar y como se evalúan**, en la siguiente fase se investiga si ya existen herramientas actuales que puedan realizar las funciones que necesita el proyecto, y también, conocer cómo puede un sistema interpretar un diseño de arquitectura de software.

La fase final de la investigación se centra en estudiar las tecnologías actuales e intentar definir una posible aproximación que pudiera dar solución al problema propuesto y analizar las posibles limitaciones técnicas o riesgos a los que puede enfrentarse la solución propuesta.

3.1 Evaluación de arquitecturas de software

En esta sección se exploran las entrevistas de arquitectura de software con el fin de conocer qué aspectos se valoran normalmente en una entrevista de arquitectura de software y cómo se puede implementar el método de evaluación manual en un sistema para automatizarlo.

3.1.1 Aspectos a evaluar en una entrevista de arquitectura de software

En esta sección se analiza **cómo se evalúan** actualmente las arquitecturas de software en los procesos de selección y **qué metodología se utiliza** actualmente para calificar este tipo de cuestiones.

Las cuestiones sobre arquitectura de software no son exactamente iguales entre sí, sino que varían dependiendo de muchos factores, algunos de estos factores son los siguientes.

- El **nivel de experiencia del candidato**, en los procesos de selección para perfiles senior no se evalúa con la misma exigencia que en los procesos para candidatos menos experimentados se tienen en cuenta distintos aspectos. Mientras que de un perfil senior se evalúan características como la modularidad, escalabilidad, rendimiento, mantenibilidad, seguridad, interoperabilidad o flexibilidad, para un perfil menos experimentado, el rendimiento y la escalabilidad podrían ser suficientes.
- **Las necesidades del puesto de trabajo**, dependiendo del puesto que vaya a desempeñar el candidato se pueden tener en cuenta distintos factores, puede que para el puesto a desempeñar solamente sea necesario tener nociones básicas sobre



Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

arquitectura de software, mientras que para otros puestos se necesiten conocimientos avanzados.

- La propia compañía puede establecer distintos aspectos que pueden ser cruciales para el puesto y relajar la evaluación de otros aspectos que pueden no ser tan importantes.
- El nivel de detalle de la arquitectura, hay características de la arquitectura como la seguridad, que resultan imposibles de evaluar sin un alto grado de detalle y posiblemente se necesiten documentos adicionales al diagrama de arquitectura.

Como se puede observar, los puntos a evaluar pueden ser muy variados dependiendo de muchos factores, por tanto, el sistema desarrollado ha de ser flexible en cuanto al nivel de exigencia y ha de ser capaz de calificar distintos aspectos que puedan ser relevantes para cada cuestión.

Los factores anteriormente mencionados son factores técnicos que se pueden evaluar mediante el análisis de un diagrama de arquitectura, como ya se ha mencionado, dependiendo del nivel de detalle, hay muchos otros factores en una entrevista que no pueden ser analizados automáticamente como son las denominadas habilidades blandas.

Las habilidades blandas son habilidades que ayudan a mejorar la efectividad en el trabajo pero que no son específicamente técnicas para el puesto de trabajo, algunos ejemplos de habilidades blandas son las siguientes.

- La capacidad de comunicarse de manera clara y efectiva con personal de distintos departamentos. Esto implica expresar ideas de manera concisa, escuchar activamente, hacer preguntas relevantes y transmitir información técnica de manera comprensible para personas con distintos niveles de conocimientos técnicos.
- Resolución de problemas: ser capaz de abordar problemas complejos de manera sistemática y encontrar soluciones efectivas, disponer de distintas habilidades analíticas y ser capaz de descomponer problemas en partes más pequeñas, identificar patrones y tendencias, y aplicar métodos de resolución de problemas.
- Pensamiento crítico: Ser capaz de analizar de manera objetiva y reflexiva los problemas y situaciones, así como evaluar la información disponible, cuestionar suposiciones, identificar posibles implicaciones y consecuencias, y tomar decisiones informadas basadas en la lógica y la evidencia.
- Adaptabilidad: La industria del software está en constante evolución, por lo que es importante ser adaptable y estar dispuesto a aprender y adaptarse a nuevos lenguajes de programación, tecnologías y metodologías. Ser capaz de gestionar el cambio y mantener una mentalidad abierta es fundamental en la ingeniería del software.
- La capacidad de gestionar eficientemente el tiempo y los recursos para cumplir con los plazos y objetivos establecidos. Requiere habilidades como la planificación, establecer prioridades, administrar tareas de manera efectiva y ser capaz de manejar múltiples proyectos simultáneamente.
- El cumplimiento de los requisitos, en una entrevista de arquitectura se presenta una tarea con descripción y unos requisitos a cumplir, estos requisitos son variables y dependientes de la tarea en sí, por tanto, es muy difícil establecer unos requisitos comunes.

Una vez conocidas las características técnicas a evaluar que normalmente se evalúan en una entrevista de arquitectura software, se ha de determinar una estrategia de evaluación flexible y determinada para poder evaluar un amplio rango de diagramas con distintos requisitos.

3.1.2 Métodos de evaluación de arquitecturas software

En esta sección se analizan distintas opciones para evaluar las arquitecturas software, se analizan estándares y métodos utilizados en la industria con el fin de conocer las mejores prácticas para evaluar arquitecturas de software.

3.1.2.1 Architecture Tradeoff Analysis Method

El **ATAM** (*Architecture Tradeoff Analysis Method*) es un método utilizado para evaluar arquitecturas de software en función de **atributos de calidad específicos**. Su objetivo es identificar riesgos y tomar decisiones informadas sobre los compromisos arquitectónicos.

La aplicación del **ATAM** para analizar diagramas de arquitectura sigue la siguiente estructura.

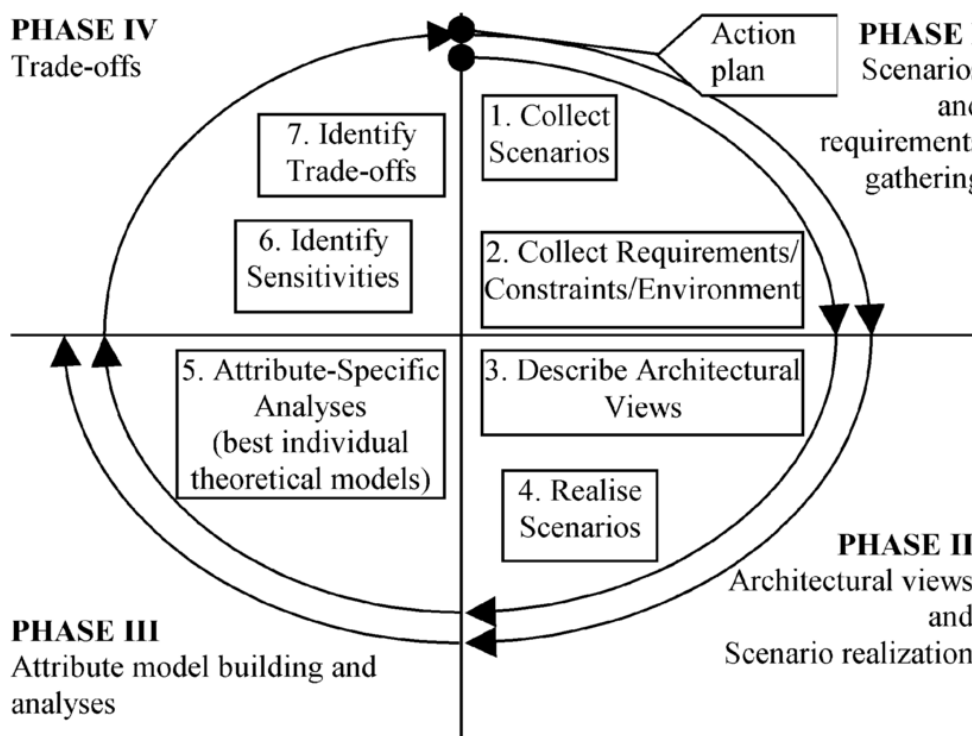


Figura 8. ATAM pasos y distintas fases. [13]

1. En la primera fase se definen los objetivos del análisis, se deben establecer los objetivos claros del análisis de la arquitectura y determinar qué atributos y requisitos de calidad se evaluarán. Por ejemplo, podrían ser rendimiento, mantenibilidad, acoplamiento, flexibilidad, entre otros.
2. En la segunda fase se realiza una presentación detallada de la arquitectura utilizando diagramas y documentación pertinente. Además de explicar la estructura, los

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

componentes, las interacciones y los mecanismos utilizados. Posteriormente se procede a definir y describir escenarios relevantes que representan situaciones o casos de uso en los cuales se espera que la arquitectura cumpla con los atributos de calidad deseados. Por ejemplo, un escenario podría ser la respuesta de la arquitectura ante un alto volumen de tráfico.

3. En la tercera fase se evalúan los atributos seleccionados, se analiza cómo la arquitectura satisface los atributos de calidad descritos en los escenarios, en esta fase se identifican los puntos fuertes y débiles de la arquitectura en relación con cada punto a evaluar.
4. La cuarta fase consiste en identificar los riesgos y los compromisos arquitectónicos asociados con los atributos de calidad evaluados. Además, se han de analizar las implicaciones y consecuencias de las decisiones arquitectónicas.
5. En la fase final se proponen recomendaciones para mitigar los riesgos identificados y mejorar la arquitectura. Estas recomendaciones pueden incluir cambios en la arquitectura, ajustes en los componentes o consideraciones adicionales para el diseño.

Cabe destacar que el *ATAM* es un enfoque flexible y los pasos pueden adaptarse según las necesidades y requisitos del proyecto. La estructura básica proporcionada por el *ATAM* ayuda a guiar el análisis y la evaluación de la arquitectura, permitiendo tomar decisiones informadas y mejorar la calidad arquitectónica en base a los resultados obtenidos.

3.1.2.2 ISO/IEC 42010

El ISO/IEC 42010 es un estándar internacional que establece los principios y conceptos fundamentales para describir, analizar y evaluar arquitecturas de software. Sus principales objetivos son:

- Proporcionar un **lenguaje común** y una terminología precisa para facilitar la comunicación entre los profesionales de la arquitectura de software.
- Establecer un **marco de referencia** para el desarrollo y evaluación de arquitecturas de software.
- Guiar la descripción y documentación de arquitecturas de software de manera coherente y completa.
- **Facilitar el análisis y la evaluación sistemática** de las arquitecturas de software en términos de requisitos, calidad y riesgos.

En la siguiente figura se analizan las principales ventajas y desventajas que conlleva la aplicación del estándar.

Figura 9. Ventajas y desventajas del ISO/IEC 42010

Ventajas	Desventajas
Establece una base común de conocimientos y terminología, lo que facilita la comunicación y comprensión mutua entre profesionales.	Puede requerir una curva de aprendizaje inicial para comprender y aplicar plenamente los conceptos y terminología del estándar.
Proporciona un marco estructurado para describir, analizar y evaluar arquitecturas, lo que ayuda a realizar evaluaciones más sistemáticas y fundamentadas.	La aplicación del estándar puede resultar compleja en contextos de proyectos muy pequeños o no tan estructurados.
Promueve la documentación coherente y completa de las arquitecturas, lo que facilita su comprensión y mantenimiento a largo plazo.	Al ser un estándar amplio, no proporciona una guía detallada sobre cómo realizar ciertos tipos específicos de análisis o evaluaciones arquitectónicas.

El ISO/IEC 42010 es un estándar que establece los principios y conceptos fundamentales para la descripción, análisis y evaluación de arquitecturas de software. Ofrece una terminología común, un marco de referencia y pautas para mejorar la comunicación, documentación y evaluación de las arquitecturas.

No obstante, tras analizar el estándar se considera que su aplicación en arquitecturas simples puede resultar excesiva, se considera que requiere una cantidad de tiempo y recursos que no pueden justificarse en el contexto de una entrevista de software, además de añadir complejidad innecesaria y dificultar el proceso de análisis.

3.1.2.3 Enfoque Práctico para el Análisis de Arquitecturas de Software.

Tras revisar los estándares y metodologías existentes se considera que pueden resultar excesivos para el contexto del proyecto. Salvo ciertas excepciones, el resultado de una entrevista de arquitectura software suele ser menos complejo debido a las limitaciones temporales.

Teniendo en cuenta dichas limitaciones, se propone un plan de evaluación con un enfoque ligero y adaptado al contexto, en la siguiente figura se puede observar los pasos que sigue el proceso propuesto.



Figura 10. Metodología de evaluación propuesta.

- 1. Análisis de requisitos:** En esta fase se verifica si la arquitectura cumple con los requisitos y objetivos establecidos para el sistema, se revisa si se abordan adecuadamente los aspectos funcionales y no funcionales definidos en los requisitos. Adicionalmente en esta fase se identifican posibles brechas y posibles ajustes en la arquitectura si procediera.
- 2. Análisis heurístico:** En la segunda fase de análisis se realiza una lista de características para evaluar como pueden ser la modularidad, escalabilidad, acoplamiento, cohesión, separación de responsabilidades, etc. En esta fase se pueden incluir atributos a evaluar que no se consideran como heurísticas, pero puedan ser importantes dentro del contexto como la disponibilidad o la consistencia.
- 3. Análisis de dependencias:** En esta fase se realiza un análisis de las dependencias entre los componentes de la arquitectura. Identifica las dependencias innecesarias o no deseadas que puedan generar acoplamiento excesivo y dificultar la mantenibilidad. Se pueden considerar opciones para reducir las dependencias y promover una arquitectura más modular y flexible.
- 4. Patrones arquitectónicos conocidos:** En la última fase se considera si la arquitectura sigue patrones arquitectónicos conocidos y ampliamente aceptados para problemas similares. Por ejemplo, puedes analizar si se utiliza el patrón de capas, el patrón cliente-servidor o el patrón de microservicios, según corresponda. Evalúa cómo se aplica el patrón y si cumple con los principios y beneficios esperados.

Una vez conocido el enfoque que se desea utilizar para analizar las características de una arquitectura software, se procede a analizar las herramientas actuales que pueden ser utilizadas para alcanzar los objetivos del proyecto.

3.2 Análisis de las herramientas actuales

En esta sección se analizan **las herramientas actuales en el ámbito de la IA y de los grandes modelos de lenguaje (LLM)** que pueden ser de utilidad para evaluar arquitecturas software. Concretamente se analizan las dos herramientas de inteligencia artificial más populares *Google Bard* y *Open AI GPT-4*, capaces de generar texto, traducir idiomas, escribir diferentes tipos de contenido creativo y responder a preguntas de manera informativa [14].

Por tanto, es posible que ambas herramientas puedan utilizarse para automatizar la evaluación de diagramas de arquitectura de software.

- Google Bard fue anunciado por **Google** en enero de 2023. Está basado en el modelo LaMDA (*Language Model for Dialogue Applications*), que fue entrenado en un conjunto de datos de texto y código de 1,56T de palabras.

- GPT-4 fue anunciado por **OpenAI** en noviembre de 2022. Es la última versión del modelo GPT (Generative Pre-trained Transformer), GPT-4 está entrenado en un conjunto de datos de 175T de palabras.

Ambos modelos serían capaces de cumplir con precisión las exigencias del proyecto, por tanto, para decidir que herramienta se selecciona para el proyecto se van a comparar las características de las API de ambas herramientas.

Comparativa entre Bard y OpenAI

En términos de rendimiento, la API de OpenAI utiliza un **modelo de lenguaje** de 175 mil millones de parámetros, mientras que la API de Bard utiliza un modelo de lenguaje de 1.56 mil millones de parámetros. Esto significa que el modelo de lenguaje de la API de OpenAI es aproximadamente **11 veces más grande** que el modelo de lenguaje de la API de Bard. Los modelos de lenguaje más grandes suelen ser **más rápidos** porque pueden procesar un mayor flujo de información de forma paralela.

Las dos API utilizan diferentes técnicas para mejorar la precisión de sus respuestas. La API de OpenAI utiliza un enfoque de **aprendizaje supervisado**, que significa que se entrena en un conjunto de datos de texto y código etiquetado. La API de Bard, por otro lado, utiliza un enfoque de **aprendizaje no supervisado**, que significa que se entrena en un conjunto de datos de texto sin etiquetas, en este apartado, ambas API son prácticamente equivalentes, sin embargo, la API de OpenAI puede ser más precisa para tareas que requieren un conocimiento específico, como la traducción de idiomas o la escritura de código.

En cuanto a las principales diferencias Bard al ser diseñado como un **chatbot conversacional**, es capaz de comprender mejor las **preguntas abiertas y que incluyen gran variedad de matices** como puede ser requisitos de arquitectura de software, además de ser capaz de acceder a información en tiempo real.

A continuación, se puede observar una figura que describe la comparativa realizada entre la API de GPT-4 y Bard comparando características como los precios, precisión, velocidad, diseño y las principales ventajas de un modelo sobre el otro.

Figura 11. Comparación de las API de GPT-4 y Bard [15].

Característica	API de OpenAI	API de Bard
Tamaño del conjunto de datos	175T palabras	1,56T palabras
Rendimiento	Media de 25s para consultas complejas	Media de 35s para consultas complejas
Precisión	Mayor precisión	Menor precisión
Precios	20\$/mes	Gratuito
Reconocimiento de imágenes	Si	Si
Diseño	Transformador generativo	Chatbot conversacional
Ventajas	Generar diferentes formatos	Mejor capacidad de respuesta

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

	de texto creativo, como poemas, código, guiones, piezas musicales, correo electrónico, cartas, etc.	frente a preguntas abiertas, desafiantes o extrañas. Además, puede acceder a internet.
--	---	--

Para elegir correctamente la herramienta, se analizan las capacidades de ambos modelos para la evaluación de un diagrama de arquitectura del software, imaginemos que se debe diseñar un sistema de procesamiento de pagos ser seguro, escalable y eficiente y la solución proporcionada es la siguiente.

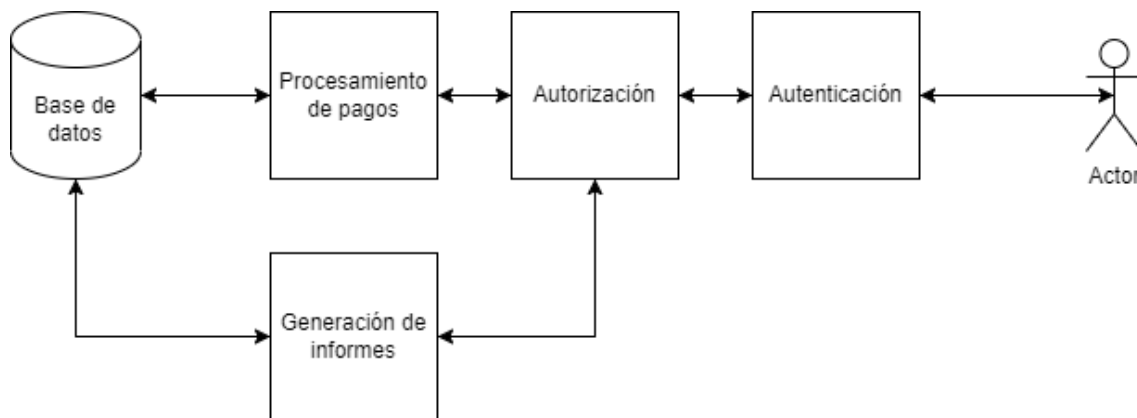


Figura 12. Ejemplo de diagrama de arquitectura de software.

Ambos modelos son capaces de evaluar el diagrama de componentes y determinar si es consistente con los requisitos del sistema, si incluye todos los componentes necesarios, y si los componentes cumplen con los requisitos de seguridad, escalabilidad y eficiencia.

Ambas herramientas son capaces de evaluar el diagrama en términos de modularidad, escalabilidad, acoplamiento, cohesión, separación de responsabilidades y seguridad.

También pueden identificar las dependencias entre los componentes del diagrama. Por ejemplo, el diagrama tiene una dependencia alta entre el componente de procesamiento de pagos y el componente de autorización. Esta dependencia podría dificultar la mantenibilidad del sistema, ya que cualquier cambio en el componente de procesamiento de pagos podría requerir un cambio en el componente de autenticación o viceversa.

En diagramas más complejos, también se podría determinar si el diagrama utiliza el patrón de capas, o el patrón de microservicios, si consta de capas de presentación, lógica empresarial y datos, etc.

En general, tanto Bard como GPT-4 podrían proporcionar evaluaciones útiles de un diagrama de arquitectura de software. Bard sería más adecuado para evaluar la consistencia, calidad y problemas de diseño de los diagramas, mientras que GPT-4 parece más adecuado para evaluar la claridad, precisión y creatividad de los diagramas.

Teniendo en cuenta que ambos modelos son perfectamente válidos y la naturaleza experimental del proyecto, se considera que Bard es la herramienta más adecuada, ya que permite el acceso a

información en tiempo real además de ser gratuito. La penalización en el rendimiento no es un problema para el proyecto, además la diferencia no es suficiente para justificar el coste extra de 20\$ mensuales de la licencia de uso de GPT-4.

Habiendo analizado las diferentes opciones que pueden utilizarse para conseguir el objetivo del proyecto, se procede a definir una posible solución a alto nivel, identificando los posibles riesgos y limitaciones técnicas.

3.3 Solución propuesta

Una vez analizadas las herramientas de software capaces de dar solución al problema del proyecto, que es la **automatización de la evaluación** de diagramas de arquitectura de software, se procede a diseñar a alto nivel una solución de software que pueda explotar el potencial de estas herramientas y proporcionar la solución como servicio.

Debido a la naturaleza del servicio que ofrece el proyecto, un cliente debería de ser capaz de realizar peticiones aisladas utilizando un mecanismo de autenticación, pero el servidor no tiene la necesidad de almacenar información contextual relativa a la petición, por lo tanto, se decide implementar un servicio denominado *Stateless Rest API* [16].

Las principales ventajas que ofrece una *Stateless Rest API* son principalmente la escalabilidad y la seguridad. Al no requerir que el servidor mantenga el estado de la sesión del cliente, las solicitudes de un cliente pueden ser procesadas por cualquier servidor de la aplicación, facilitando el balanceo de carga. Además, al no almacenar información confidencial del cliente en el servidor como contraseñas o tarjetas de crédito, dificulta que un atacante pueda obtener ventajas al atacar estos servicios.

A continuación, se muestra un diagrama de alto nivel sobre cómo podría implementarse la solución software para el proyecto.



Figura 13. Diagrama de la solución propuesta

Como se muestra en la figura anterior, el servicio a implementar en el proyecto sería la API, el flujo del diagrama sería el siguiente.

1. Un usuario proporciona una tarea de arquitectura de software con distintos parámetros como la descripción, rúbrica de evaluación y la solución de un candidato.
2. El cliente hace uso de una API o un servicio para transformar la descripción de la tarea, el diagrama y la rúbrica de evaluación al formato que la API especifique, además probablemente deberá acompañar la petición de un identificador.
3. La API implementada recibe la petición del cliente, y utilizando técnicas de *prompt engineering* transforma la petición en un *prompt* adecuado para Google Bard.

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

4. La API implementada hace la petición a la API de Google Bard.
5. Una vez recibida la respuesta de Google Bard, la API realiza las transformaciones pertinentes para devolver la respuesta al cliente en el formato especificado.
6. El cliente recibe la respuesta.

Limitaciones técnicas y posibles soluciones

En cuanto a las limitaciones técnicas de la solución nos encontramos las siguientes:

- Bard puede **cometer errores y proporcionar información incorrecta** sobre los componentes y relaciones de un diagrama de arquitectura, o puede no ser capaz de comprender correctamente los parámetros de evaluación.
- Bard **puede ser lento**, es una herramienta computacionalmente intensiva. Si el diagrama de arquitectura es complejo o los parámetros de evaluación son numerosos, Bard puede tardar un tiempo en completar la evaluación.
- El plan gratuito actual, tiene un **límite de 10,000 solicitudes por mes y 10 solicitudes por segundo**.
- Es posible que los **planes de suscripción cambien** en el futuro y que Bard no sea la mejor opción en el futuro.
- La API de Google Bard está en desarrollo, por tanto, **la información obtenida siempre debe de ser verificada**.

Para mitigar estas limitaciones, es importante proporcionar un diagrama de arquitectura completo y bien documentado junto con parámetros de evaluación claros y concisos.

- **Proporcionar información clara y concisa** sobre los requisitos, restricciones, así como los parámetros de evaluación del sistema.
- **Realizar pruebas de evaluación** para asegurarse de que Bard está funcionando correctamente.
- **Verificar las respuestas** de la API utilizando a un experto en la materia.
- Desarrollar el sistema teniendo en cuenta que es posible que los planes de suscripción cambien o que surja un nuevo modelo más adecuado para el proyecto en el futuro.

Se considera la utilización de servicios en la nube para desplegar la infraestructura de la aplicación, ya que abstrae al programador de la mayor parte de los problemas sobre escalabilidad, seguridad, despliegue, etc.

En cuanto a los proveedores en la Nube, se ha escogido *Amazon Web Services* ya que ofrece un plan gratuito flexible y adecuado al proyecto, se utilizarán los siguientes servicios de AWS.

Principalmente, se utiliza AWS Lambda para proporcionar la infraestructura que se encarga de atender, autenticar, autorizar las peticiones y proporcionar las respuestas.

Utilizar servicios en la nube proporciona una solución fiable para problemas como la escalabilidad, seguridad, disponibilidad e infraestructura del proyecto, entre otros factores.



4 Requisitos del sistema

En esta sección se definen a grandes rasgos **los requisitos que debe cumplir el sistema** para considerarlo viable y funcional. Normalmente, en esta sección se haría uso de un estándar de definición de requisitos como el **IEEE 830**. No obstante, dado el grado de incertidumbre del proyecto, así como teniendo en cuenta la alta variabilidad, se ha decidido establecer los requisitos de una forma menos estricta y con un alto grado de flexibilidad.

4.1 Usuarios del sistema

Se ha identificado **un único usuario en el sistema**, el sistema está pensado para ser utilizado por cualquier usuario, y cualquier usuario puede registrarse. La gestión de los usuarios no será competencia directa del sistema diseñado, se utilizará un sistema externo para la autenticación y autorización. El Usuario se refiere a cualquier persona que utiliza la aplicación.

En cuanto a las características del usuario se destacan las siguientes.

- Nivel de conocimientos técnicos de arquitectura de software: El usuario debe de ser capaz de entender los requisitos del diagrama, y evaluar los resultados.
- Capacidad para realizar peticiones al sistema: El sistema es una API, por tanto, el usuario debe de ser capaz de utilizar la API haciendo uso de las herramientas de software que considere.
- Capacidad para obtener las respuestas y procesarlas debidamente: El usuario debe de ser capaz de procesar la respuesta de la API.

4.2 Funciones del Sistema

En este apartado, se listan las **principales funciones** que debe de ofrecer el sistema, las funciones se definen teniendo en cuenta que debe ser capaz de realizar el sistema, pero no se describe como debe de hacerlo.

- **Registrar usuario:** Se debe de permitir el registro de un usuario mediante algún método de autenticación para que haga uso del sistema diseñado.
- **Eliminar usuario:** Un usuario registrado puede, si así lo desea, dejar de estar registrado en el sistema.
- **Autenticar usuario:** El sistema debe de ser capaz de autenticar a un usuario registrado.
- **Autorizar usuario:** El sistema debe autorizar las peticiones del usuario tras ser autenticado.
- **Procesar diagramas** de arquitectura de software: El sistema debe de ser capaz de reconocer y evaluar diagramas de arquitectura de software.
- **Aceptar y procesar peticiones** de usuarios autorizados: El sistema debe de proporcionar un punto de acceso para que los usuarios identificados puedan realizar peticiones y obtener su respuesta deseada.
- **Limitar el uso del servicio:** Se debe de limitar el uso del servicio por usuario para no exceder los límites de los servicios consumidos.



4.3 Restricciones

El servicio implementado utilizará el protocolo de comunicación *http*, por tanto, el consumidor debe tener **acceso a internet** y sus peticiones deben ajustarse al formato definido por el servicio.

Se consideran las siguientes restricciones, sin tener en cuenta las posibles restricciones de la infraestructura utilizada.

- La longitud máxima de las peticiones es de 2048 caracteres.
- El número de posibles resultados es de 100 resultados por petición.
- El tamaño máximo de la respuesta es de 1MB.
- Se pueden hacer un máximo de 10 peticiones por segundo.
- La API de Bard únicamente soporta el tipo de contenido “*application/json*”, por tanto, el formato utilizado en las peticiones debe de ser JSON.
- Debido a las limitaciones de la infraestructura, el consumidor debe proporcionar el diagrama en formato de dirección *url* accesible públicamente.

4.4 Suposiciones y dependencias

Como se ha descrito anteriormente, existe una alta dependencia técnica con el modelo utilizado, en este caso, Bard. Por lo tanto, el sistema debe desarrollarse de una forma desacoplada permitiendo el cambio a otro modelo fácilmente.

4.5 Requisitos de rendimiento

No se especifican requisitos de rendimiento, el sistema debe responder con la mayor celeridad posible a todas las peticiones. Sin embargo, las API de los modelos tienen un tiempo medio de respuesta de 30 segundos. Teniendo en cuenta que las peticiones que se van a realizar requieren un alto rendimiento computacional, se puede establecer un tiempo máximo de respuesta inferior a 2 minutos.

Es posible, que, en periodos con un alto volumen de peticiones, el tiempo máximo de respuesta sea superior al descrito o incluso que la API falle, en este caso, el consumidor deberá de intentar realizar la petición en otro momento o reformular la petición para que se ajuste mejor a los parámetros requeridos.

4.6 Atributos del sistema

En este apartado se describen las **características** que ha de cumplir el sistema. Las características principales son: **fiabilidad, escalabilidad y seguridad**.

- El sistema debe ofrecer una correcta implementación de las funcionalidades descritas anteriormente, es decir, debe ser fiable.

- El producto desarrollado debe adaptarse a las demandas del número de usuarios que puedan acceder a él simultáneamente y realizar peticiones simultaneas dentro de los límites establecidos por la API.
- El sistema ha de implementar una política de seguridad para no permitir accesos no autorizados y proporcionar un sistema de persistencia de datos. La seguridad se ha de implementar mediante el uso de claves de API. El sistema ha de validar la clave de API en cada petición y confirmar que es correcta y que el consumidor tiene acceso a los servicios.

A continuación, una vez conocidos a alto nivel los requisitos del sistema, se procede a diseñar el sistema haciendo uso del modelo de C4.



5 Diseño del sistema

En esta sección se especifica el diseño del sistema siguiendo el estándar ya mencionado **modelo C4**. La arquitectura de software es el arte y la ciencia de diseñar y construir sistemas de software complejos, por tanto, es un proceso que requiere una cuidadosa planificación y documentación para garantizar que los sistemas sean robustos, escalables y adaptables. El modelo C4 es una técnica de **notación gráfica** que se utiliza para modelar la arquitectura de sistemas de software. Fue desarrollado por Alistair Cockburn y fue publicado por primera vez en 2011 [17].

La principal ventaja de C4 es que el estándar está pensado para audiencia tanto técnica como general, se puede pensar en C4 como si fuera un mapa geográfico, en el nivel más general estaría el país, y conforme se profundiza en el país surgen las comunidades, provincias, ciudades, por tanto, cuanto más se profundiza mayor es el nivel de conocimientos técnicos requeridos para comprender los diagramas.

El modelo C4 consta de cuatro diagramas: contexto, contenedores, componentes y código.

- **El diagrama de contexto** es el diagrama más general del modelo C4.
- **El diagrama de contenedores** muestra los contenedores que componen un sistema, junto con sus relaciones.
- **El diagrama de componentes** muestra los componentes que componen un contenedor, junto con sus relaciones.
- **El diagrama de código** muestra el código fuente de un componente, junto con sus dependencias.

Finalmente, se incluye un apartado final con el diseño de la API según el estándar *OpenAPI 3.0*.

5.1 Diagrama de Contexto

En esta sección se presenta el **diagrama de contexto**, este diagrama proporciona una **visión general de un sistema de software y su entorno**. Muestra el sistema en sí, los actores que interactúan con él y los límites entre el sistema y su entorno. El diagrama de contexto es el primer diagrama del modelo C4. Se utiliza normalmente por arquitectos, ingenieros y gerentes de producto, aunque también puede ser interesante para analistas de negocios y probadores.

El diagrama de contexto se utiliza para:

- **Comunicar el alcance del sistema** a los interesados: muestra los límites del sistema, lo que ayuda a los interesados a comprender qué está dentro del sistema y qué está fuera del sistema.
- **Identificar los actores** que interactúan con el sistema: representa los actores que utilizan el sistema, así como los actores que se ven afectados por el sistema. Esta información normalmente se utiliza para priorizar los requisitos y para diseñar las interfaces del sistema.

- **Comprender las dependencias del sistema con su entorno:** define los límites entre el sistema y su entorno. Se puede utilizar para identificar riesgos y para tomar decisiones sobre cómo desplegar y mantener el sistema.

El diagrama de contexto consta de los siguientes elementos:

- El sistema de software que se está documentando, se representa como un rectángulo con el nombre del sistema en el centro.
- Los actores u otros sistemas que interactúan con el sistema, estos se representan como personas o iconos, y se conectan al sistema con líneas de comunicación.
- Los límites que separan el sistema de su entorno, los límites se representan como una línea discontinua que encierra el sistema y sus actores.

A continuación, se muestran unos ejemplos de cómo se podría utilizar el diagrama de contexto en un proyecto real.

- Un equipo de desarrollo que está construyendo un nuevo sistema de gestión para una compañía podría utilizar el diagrama de contexto para identificar a los empleados que utilizarán el sistema.
- Un gerente de proyecto que está a cargo de desarrollar un nuevo sistema web podría utilizar el diagrama de contexto para identificar a los usuarios que utilizarán el sitio web, como los clientes, los empleados y los socios.
- El equipo de marketing que está lanzando un nuevo producto *SaaS* podría utilizar el diagrama de contexto para explicar a los clientes cómo el producto podría integrarse con sus sistemas existentes.

En la siguiente figura se presenta el diagrama de contexto del sistema diseñado.

System Context Diagram

This diagram shows the relationship between a software system and its environment.

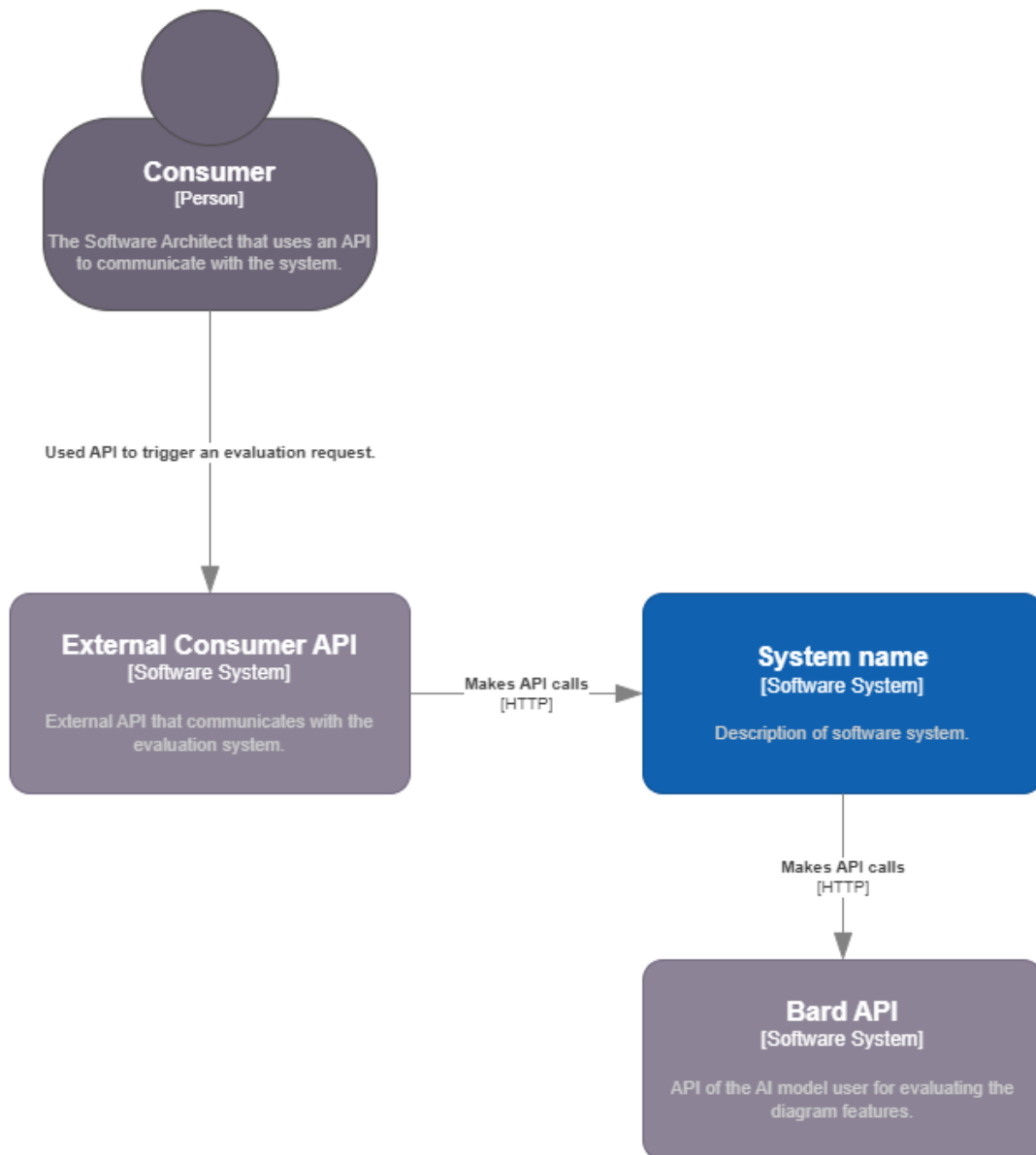


Figura 14. Diagrama de contexto del sistema diseñado.

A continuación, en la siguiente sección se presenta el segundo de los cuatro diagramas del modelo C4, el diagrama de contenedores.

5.2 Diagrama de contenedores

En esta sección, se presenta el **diagrama de contenedores**, los contenedores pueden representar diferentes niveles de abstracción, desde aplicaciones individuales hasta sistemas completos. Este diagrama está dirigido a una **audiencia más técnica**, pero puede ser comprendido por un público no especializado en ingeniería de software, en este diagrama se muestra cómo los diferentes contenedores se relacionan entre sí.

El diagrama de contenedores es **el segundo nivel** de un conjunto de diagramas de arquitectura de software. Proporciona una **vista de alto nivel de la estructura del sistema** de software, mostrando cómo se distribuyen las responsabilidades entre los contenedores.

Este diagrama es típicamente utilizado por arquitectos, ingenieros y gerentes de producto. El diagrama se compone de los siguientes elementos:

- **Los contenedores:** Los contenedores son unidades de software que ejecutan código o almacenan datos. Pueden ser aplicaciones, servicios, bases de datos, máquinas virtuales, etc.
- Las **relaciones entre contenedores:** Las relaciones entre contenedores muestran cómo los contenedores interactúan entre sí. Las relaciones pueden ser de entrada, salida o ambas.

El diagrama de contenedores se utiliza para:

- Comunicar la estructura del sistema a los interesados: define cómo se dividen las responsabilidades del sistema en contenedores, por tanto, sirve para comprender cómo funciona el sistema.
- Identificar los contenedores que componen el sistema: en este diagrama se muestran los contenedores que componen el sistema, así como las relaciones entre ellos.
- Entender las dependencias entre los contenedores: al representar las dependencias entre los contenedores, puede ser útil para identificar riesgos y a tomar decisiones sobre el orden de despliegue de los distintos contenedores del sistema.

En cuanto a la representación, una forma común es utilizar un diagrama de bloques. En un diagrama de bloques, los contenedores se representan como bloques rectangulares y las relaciones entre contenedores se representan como flechas.

Otra forma común de representar el diagrama de contenedores es utilizar un diagrama de composición. En un diagrama de composición, los contenedores se representan como rectángulos con un borde sólido y las relaciones entre contenedores se representan como líneas que conectan los contenedores.

La forma de representar el diagrama de contenedores depende de las preferencias del equipo de arquitectura, pero normalmente se siguen una serie de reglas:

- El nombre de los contenedores debe ser breve y descriptivo.
- Las relaciones entre contenedores deben ser claras y concisas.
- El diagrama de contenedores debe ser lo suficientemente detallado para ser útil para su público objetivo, pero no demasiado detallado para ser complejo o confuso.

A continuación, se puede ver el diagrama de contenedores del sistema diseñado.

AICHitect Container Diagram

The container diagram for the architecture diagram evaluation system.

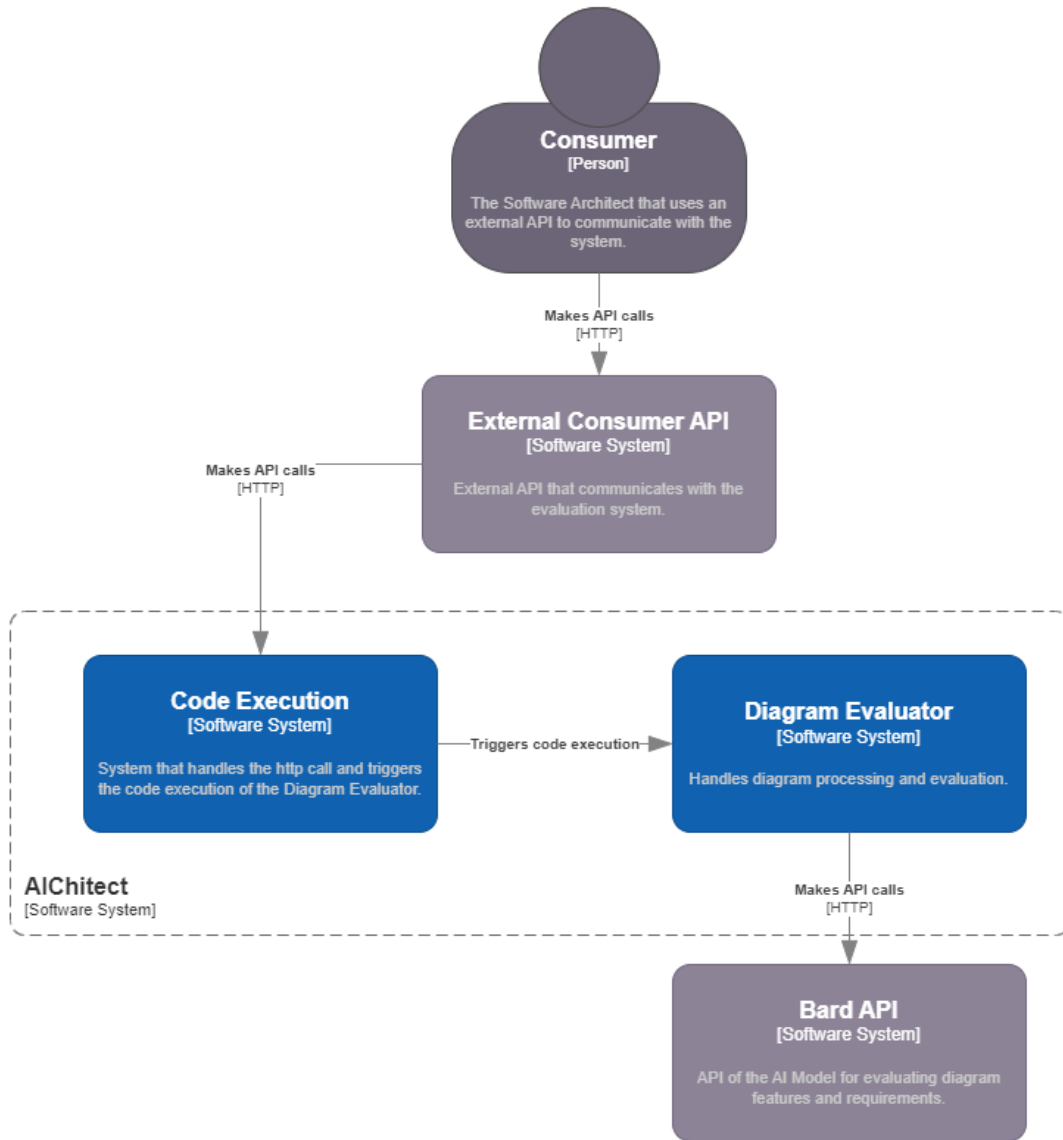


Figura 15. Diagrama de contenedores del sistema diseñado.

Como se puede observar, el diagrama de contenedores es una herramienta importante para comunicar la estructura del sistema de software, se utiliza principalmente por arquitectos, ingenieros y otros interesados para comprender y comunicar la arquitectura del sistema.

5.3 Diagrama de componentes

En esta sección se presenta el **diagrama de componentes**, es un diagrama de arquitectura de software que representa los componentes de un sistema de software y sus relaciones. Este diagrama es el tercero de los cuatro diagramas del modelo C4, y se centra en el nivel de abstracción intermedio de la arquitectura de un sistema.

El diagrama de componentes del modelo C4 se compone de los siguientes elementos:

- **Componentes:** Los componentes son unidades de software que tienen una función independiente. Pueden ser módulos, clases, paquetes, servicios, contenedores, etc.
- **Relaciones:** Las relaciones entre componentes representan las dependencias, asociaciones o composición entre ellos.
- **Interfaces:** Las interfaces son los puntos de entrada y salida de un componente.
- **Tecnología:** La tecnología utilizada para implementar un componente.

El diagrama de componentes se utiliza para **comunicar la estructura** de un sistema de software a una **audiencia técnica**, puede ayudar a comprender cómo se relacionan los componentes entre sí, y cómo se implementan. Por lo tanto, sus principales beneficios son los siguientes:

- El equipo de desarrollo que está construyendo un nuevo sistema de aprendizaje automático podría utilizar el diagrama de componentes para diseñar la arquitectura de los componentes del sistema, los componentes de procesamiento de datos, los componentes de aprendizaje automático y los componentes de visualización.
- El equipo de pruebas que está probando un nuevo sistema podría utilizar el diagrama de componentes para identificar tanto los componentes que deben probarse como los componentes relacionados.
- Un equipo de mantenimiento que está solucionando un problema con un sistema de gestión de riesgos podría utilizar el diagrama de componentes para identificar los componentes que podrían estar involucrados en el problema.

Como se ha mencionado, el diagrama de componentes proporciona más detalles técnicos sobre el sistema, sus componentes y las interacciones entre los distintos componentes. La representación del sistema de componentes implica bajar un nivel de abstracción y representar la estructura interna de los contenedores del diagrama anterior.

En la siguiente figura se puede observar el diagrama de componentes del sistema diseñado, se muestran los componentes de los contenedores denominados **Code Execution** y **Diagram Evaluator**.

AIChitect Component Diagram

This shows the component diagram of the diagram evaluation system.

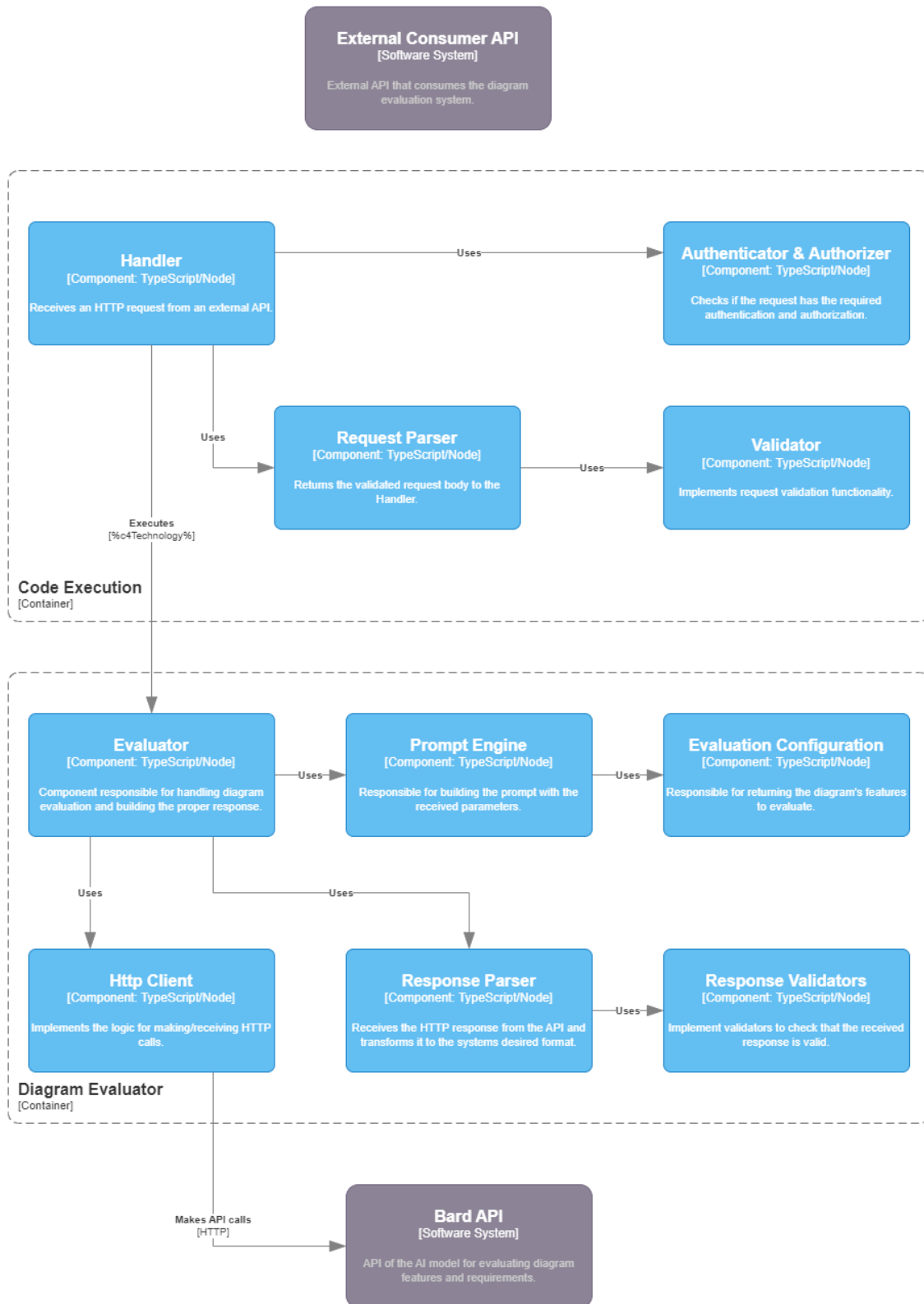


Figura 16. Diagrama de componentes del sistema diseñado.

5.4 Diagrama de código

El **diagrama de código** es un diagrama de arquitectura de software que representa la estructura del código de un sistema. Este diagrama es el cuarto y último de los diagramas del modelo C4, y se centra **en el nivel de abstracción más bajo** de la arquitectura de un sistema. Se compone de los siguientes elementos:

- **Elementos de código:** Los elementos de código son los bloques de construcción básicos del sistema. Pueden ser clases, métodos, funciones, variables, constantes, etc.
- **Relaciones** entre elementos de código: Las relaciones entre elementos de código representan las dependencias, asociaciones o composición entre ellos.
- **Comentarios:** Los comentarios proporcionan información adicional sobre el código.

El diagrama de código se utiliza para **documentar el código** de un sistema. Puede ayudar a **comprender la estructura y el funcionamiento del código**, así como identificar posibles problemas o errores. Este diagrama es utilizado principalmente por los desarrolladores, que necesitan comprender la estructura y el funcionamiento del código de un sistema. Puede ser útil para desarrolladores de todos los niveles de experiencia, desde principiantes hasta expertos.

Los principales beneficios del diagrama de código son los siguientes:

- Mejorar la comprensión del código ya que ayuda a los desarrolladores a comprender cómo se relacionan entre sí los diferentes elementos del código, facilitando la depuración, el mantenimiento y la evolución del sistema.
- Identificar posibles problemas o errores en el código, puede ayudar a prevenir fallas o problemas de rendimiento.
- Puede ayudar a documentar el código del sistema, útil para los futuros desarrolladores que necesiten comprender el código.

Cabe destacar que este diagrama es **opcional** y se suele generar **bajo demanda**, utilizando herramientas como entornos de desarrollo integrados (**IDE**).

Un ejemplo de diagrama de código simplificado es el siguiente:

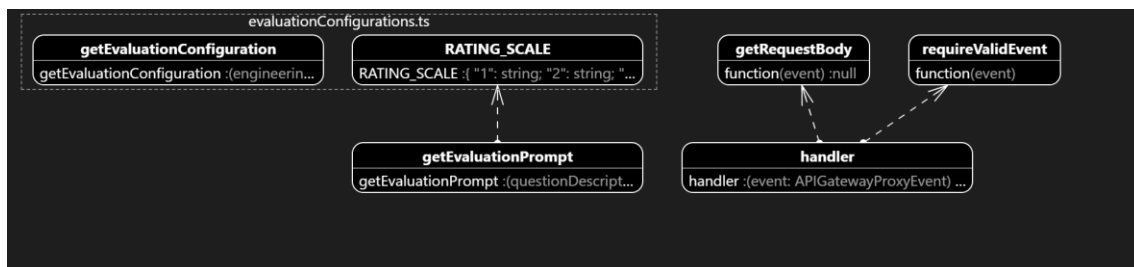


Figura 17. Ejemplo de diagrama de código inicial del sistema diseñado.

Una vez presentados los diagramas del modelo C4 del sistema diseñado, se procede a describir el diseño de la API del sistema utilizando los estándares más comunes para API webs.

5.5 Diseño de la API

En esta sección se presenta el **diseño de la API**, se ha utilizado la herramienta *Swagger* [27] y la especificación **OpenAPI 3.0** [28]. **Swagger y OpenAPI 3.0** son dos tecnologías relacionadas que se utilizan en el desarrollo de API web para documentar y definir la estructura y comportamiento de dichas API.

- **Swagger** es un conjunto de herramientas y estándares que se utiliza para diseñar, documentar y probar API web de manera sencilla y eficiente. Proporciona una especificación que describe los puntos finales de la API, sus parámetros, formatos de datos admitidos y otros detalles importantes.
- **OpenAPI 3.0** es la tercera versión de la especificación **OpenAPI**, que se utiliza para definir de manera estándar la estructura de una API web. **OpenAPI 3.0** se basa en **JSON** o **YAML** y proporciona una descripción detallada de los puntos finales de la API, los tipos de datos utilizados, la autenticación requerida y otros aspectos importantes. Esta especificación es independiente del lenguaje de programación y la plataforma, lo que la hace muy versátil y útil para la interoperabilidad entre sistemas.

A continuación, se muestra una imagen donde se puede ver el punto de acceso definido, los parámetros que acepta y las respuestas que devuelve.

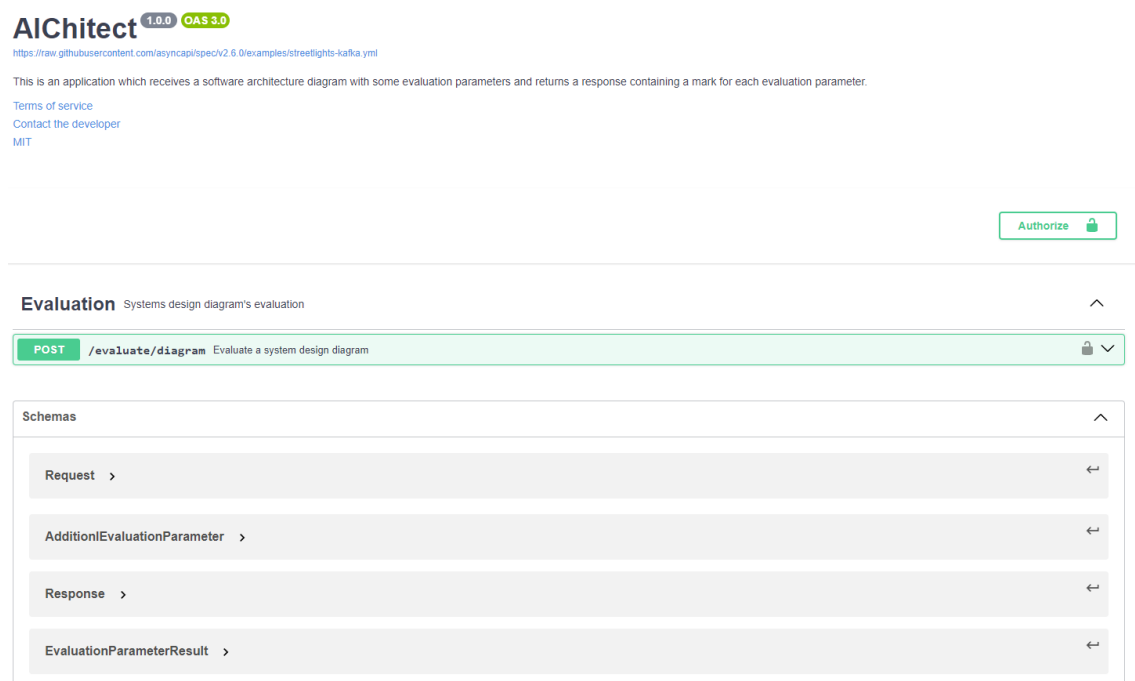


Figura 18. Diseño de la API en Swagger.

En la figura anterior se puede observar que la API dispone de una ruta */evaluate/diagram* que responde a una petición **HTTP** de tipo **POST**. Los parámetros de la petición y los formatos requeridos cuerpo de esta se pueden explorar con mayor detalle utilizando el código proporcionado en el **anexo VI** en un editor en línea de **Swagger** como <https://editor-next.swagger.io/>.

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

La especificación de la API de Bard es actualmente pública, únicamente requiere una cuenta de **Google Cloud Platform** y cualquiera puede utilizarla. Este plan gratuito cambiará en octubre de 2023 y pasará a ser un plan con características más reducidas [29].

En el siguiente apartado se especifica la fase de implementación, donde se hace una mención junto a una breve explicación de las tecnologías utilizadas y se mencionan los desafíos a los que se ha enfrentado el proyecto.

6 Fase de implementación

En la fase de implementación se explican las tecnologías que se han utilizado desde el inicio del proyecto para su desarrollo, diseño o implementación.

En la primera sección se describen las tecnologías que se consideran relevantes. Se utilizan numerosas librerías y herramientas cuya importancia no es determinante para el proyecto ni su desarrollo y, por tanto, quedan excluidas de esta sección.

En la segunda sección se presentan las dificultades que ha sufrido el proyecto en la fase de implementación y como se han hecho frente a dichas dificultades.

6.1 Tecnologías utilizadas

A continuación, se describen las principales tecnologías utilizadas para el desarrollo del proyecto. Como ya se ha mencionado anteriormente, se destacan las principales tecnologías y se excluyen aquellas que se consideran menos relevantes.

6.1.1 TypeScript

TypeScript es un **lenguaje de programación** que proviene de JavaScript, desarrollado y mantenido por **Microsoft**. La principal diferencia de TypeScript y JavaScript es la capacidad de definir objetos basados en clases y tipado estático, añadiendo seguridad a las aplicaciones que lo utilizan debido a que se ofrece la posibilidad de indicar el tipo de la variable en tiempo de compilación, por tanto, se dice que TypeScript es un superconjunto de JavaScript.

TypeScript puede transformarse a JavaScript bien en tiempo de ejecución o en tiempo de compilación. Normalmente el compilador se encarga de realizarlo automáticamente dependiendo de las opciones de configuración que se especifiquen.

Se ha decidido utilizar TypeScript ya que **AWS Lambda** ofrece soporte para Node (JavaScript) y es uno de los lenguajes más utilizados actualmente. Cabe destacar que se consideró la opción de utilizar Python por la naturaleza del proyecto, pero no se poseen conocimientos avanzados y de haber elegido **Python** podría haber resultado en **limitaciones técnicas adicionales para el proyecto**. Por tanto, en esta ocasión los riesgos superaban a los posibles beneficios obtenidos mediante el uso de este lenguaje.

A continuación, se presenta un ejemplo de código en TypeScript que se utiliza para crear los recursos en la nube utilizados en el proyecto.


```
import { Stack, StackProps } from "aws-cdk-lib";
import * as lambda from "aws-cdk-lib/aws-lambda";
import { Construct } from "constructs";

export default class AIchitectStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    new lambda.Function(this, "AIchitectLambda", {
      code: lambda.Code.fromAsset("handler"),
      runtime: lambda.Runtime.NODEJS_16_X,
      handler: "query",
    });
  }
}
```

Figura 19. Código encargado del despliegue de la infraestructura en la nube.

En la figura anterior, se puede apreciar cómo se genera el recurso de AWS Lambda para desplegar en la nube utilizando el cdk de aws, el proceso de despliegue se detalla en la siguiente sección.

6.1.2 CDK de AWS

El **CDK de AWS** es un marco de **código abierto** que utiliza las mejores prácticas de la industria para **crear infraestructura en la nube**, está escrito en TypeScript y se puede utilizar con cualquier editor de código.

Proporciona una serie de funcionalidades que permiten crear la infraestructura que necesaria dentro de la nube de Amazon. Cada recurso ofrecido por el CDK está mapeado a un recursos de AWS, por lo que no es necesario dedicar esfuerzos a mantener la infraestructura actualizada.

Además de poseer experiencia previa en el entorno de la nube de AWS, se ha decidido trabajar con el CDK de AWS por los siguientes motivos:

- **Facilidad de uso**, es relativamente fácil de aprender y utilizar
- **Flexibilidad**: Puede ser utilizado para crear la infraestructura dentro de AWS y configurarla de forma dinámica, esto proporciona flexibilidad a la hora de crear entornos de despliegue distintos.
- **Escalabilidad**: Es ligero y puede ser utilizado incluso en proyectos de menor tamaño, como es el caso.
- **Comunidad**: la comunidad de usuarios y desarrolladores es muy activa y existen numerosos recursos de aprendizaje disponibles de forma gratuita.

Para comenzar a utilizar el CDK es tan sencillo como realizar los siguientes pasos:

1. Instalar el cdk, cabe destacar que se ha de asociar el cdk a una cuenta de aws. Para instalarlo basta con ejecutar el siguiente comando: `npm install aws-cdk-lib`.
2. Inicializar el proyecto en TypeScript ejecutando el siguiente comando: `cdk init app --language typescript`.

3. Customizar el proyecto creado a las necesidades específicas de infraestructura del servicio que se necesita.
4. Ejecutar `cdk synth`: Genera una **plantilla de AWS CloudFormation** desde uno o más Stacks generados en la aplicación de CDK de AWS. Esto ayuda a comprobar si existen cambios con respecto a despliegues previos y evita el despliegue innecesario de infraestructura si no se han realizado cambios.
5. Ejecutar `cdk deploy`: **Despliega los recursos definidos** en la aplicación de CDK en la nube de Amazon.

A continuación, se puede observar un ejemplo de cómo se ha generado el script ejecutable en **Node** para desplegar la infraestructura en la nube del proyecto diseñado.

```
#!/usr/bin/env node
import * as cdk from "aws-cdk-lib";
import AIchitectStack from "../lib/a_ichitect-stack";

const app = new cdk.App();
new AIchitectStack(app, "AIchitectStack");
```

Figura 20. Script en Node para desplegar la infraestructura en la nube.

Como se puede observar el proceso es relativamente sencillo si ya se posee experiencia trabajando con recursos en la nube, es muy fácil poner en marcha un proyecto desde cero, además el plan gratuito de AWS es muy completo y flexible, permitiendo validar una idea sin generar gastos económicos extra.

6.1.3 AWS Lambda

AWS Lambda es un **servicio de computación en la nube** que permite a los desarrolladores ejecutar código **sin tener que preocuparse por la infraestructura** subyacente. Esto significa que los desarrolladores no tienen que dedicar esfuerzos a crear o administrar instancias de servidor, escalar el código o gestionar el tiempo de ejecución.

AWS Lambda es una **solución óptima** para los siguientes casos de uso:

- Es ideal para **procesar eventos en tiempo real**, como eventos de clics en un sitio web, eventos de inicio de sesión, eventos de ficheros añadidos a una base de datos, etc.
- Se puede utilizar para crear API de forma rápida y sencilla, ya que cada Lambda expone un punto de acceso para ser ejecutada mediante un evento o una llamada HTTP.
- Debido a su **capacidad computacional**, AWS Lambda se puede utilizar para analizar datos de forma paralela y escalable, aunque también hay servicios dedicados para este caso de uso.
- Lambda también se puede utilizar **para automatizar tareas**, como el envío de correos electrónicos o la actualización de bases de datos.

Entre las principales ventajas destacan su **facilidad de uso**, ya que no se requiere de conocimientos específicos de infraestructura. Además, es un servicio escalable, por lo que los desarrolladores pueden ejecutar sus funciones de Lambda sin tener que preocuparse por el rendimiento o la capacidad.

Una de las principales desventajas en el pasado es que la Lambda podía sufrir el denominado **cold start** cuando no había sido utilizada en un periodo de tiempo. El **cold start** es una **ralentización de la primera ejecución** debido al tiempo que tarda el proveedor en preparar la infraestructura necesaria para la ejecución de esta. Actualmente, se puede reservar una **provisión de concurrencia**, es decir, un número mínimo de entornos que estarán disponibles en todo momento para la ejecución de la función. Además, es posible definir reglas de escalado y desescalado tanto horizontal como vertical para que los usuarios no experimenten retrasos en momentos donde el volumen de peticiones es alto.

El código específico que se ha de desarrollar para hacer uso de una Lambda es el denominado *handler*, que no es más que el código que se encarga de responder a una petición o evento que haya invocado la función Lambda. En la siguiente figura, se muestra un ejemplo de código de un *handler* sencillo.

```
import { APIGatewayProxyEvent, APIGatewayProxyResult } from "aws-lambda";
import axios from "axios";

export const handler = async (
  event: APIGatewayProxyEvent
): Promise<APIGatewayProxyResult> => {
  if (!event.body) {
    return {
      statusCode: 400,
      body: "No data to process",
    };
  }

  const body = JSON.parse(event.body);

  const url = "https://api.google.com/query";
  const params = {
    query: body.query,
  };

  const response = await axios.get(url, {
    params,
  });

  return {
    statusCode: 200,
    body: response.data,
  };
};
```

Figura 21. Ejemplo de código de un handler en el contexto de una Lambda de AWS.

Como se puede observar, el *handler* es el punto de entrada a la función lambda, desde el handler se ha de realizar el proceso requerido para cada petición, y después, se devuelve la respuesta correspondiente.

Uno de los principales inconvenientes de las funciones Lambda es que no tienen soporte para muchos lenguajes de programación, y en ese aspecto limitan bastante a los desarrolladores que no estén actualizados a las últimas tecnologías, admite los siguientes lenguajes de programación: Java, Go, PowerShell, Node.js, C#, Python y Ruby [19].

6.1.4 Postman

Postman es una **herramienta GUI** (interfaz gráfica de usuario) que permite a los usuarios **crear, enviar y recibir solicitudes HTTP/S**. También permite a los usuarios crear y guardar colecciones de solicitudes, que pueden ser compartidas con otros usuarios. [20]

Postman se suele utilizar para **pruebas de API**, ya que ofrece las siguientes funcionalidades que permiten probar un proyecto prácticamente desde el inicio de este.

- **Creación de solicitudes HTTP/S**
- **Envío y recepción de solicitudes** a APIs
- **Almacenamiento de colecciones** de solicitudes
- **Compartición de colecciones** de solicitudes
- **Automatización de pruebas** de API

Postman se ha utilizado para realizar pruebas de aceptación y validación de la API diseñada. Las pruebas de aceptación se utilizan para verificar que la API cumple con los requisitos del cliente, mientras que las pruebas de validación se utilizan para verificar que la API funciona correctamente, se especifica cada tipo de prueba en la correspondiente sección de pruebas.

Para ello, se han creado colecciones de prueba, que, a grandes rasgos, contienen los siguientes datos:

- URL con un diagrama de prueba.
- Requisitos que debe de cumplir el sistema.
- Contexto del sistema.
- Rubrica de evaluación.

Una vez creados los datos de prueba, para cada prueba se puede especificar la respuesta que se espera obtener de la API.

Adicionalmente, Postman tiene integración con la herramienta de test **Chai** y permite escribir scripts para validar la respuesta de cada caso de prueba como se puede observar en la siguiente figura.

Recientemente Postman ha añadido la funcionalidad de escribir **casos de prueba automáticos mediante el uso de *PostBot***, no se ha hecho uso de esta herramienta porque los casos de prueba generados eran demasiado sencillos comparados con lo que se debía de probar en algunos casos extremos, como se puede comprobar en la sección dedicada a las pruebas.



```
pm.test("response should not be valid", function () {
  pm.response.to.have.status(400);
  pm.response.to.have.jsonBody({
    statusCode: 400,
    errorMessage: "Invalid request, parameters must contain a diagram public Url",
  });
});

pm.test("response should be valid and have the expected result", function () {
  pm.response.to.not.be.error;
  pm.response.to.not.have.jsonBody("error");
  pm.response.to.have.jsonBody({
    meetsRequirements: 1,
    doesNotHaveSPOF: 1,
    isScalable: 0.5,
    isComplete: 0.5,
    isCreative: -1,
  });
});
```

Figura 22. Ejemplo de casos de prueba con Postman.

Como se puede observar, Postman es una herramienta muy útil para definir tanto **casos de prueba aislados** como utilizar **colecciones de casos de prueba** que pueden ser ejecutados rápidamente y compartidos entre el equipo de pruebas, esto mejora la colaboración y comunicación entre los miembros del equipo de pruebas.

6.1.5 Jest

Jest es una **herramienta de pruebas para entornos basados en JavaScript** que se utiliza para escribir pruebas unitarias, de integración y de e2e. Esta herramienta fue creada por **Facebook** en 2013 y se ha utilizado en muchos proyectos de **JavaScript** de **Facebook**, independientemente de la tecnología. Jest ha sido adoptado por muchos otros proyectos de JavaScript, como Angular, Vue.js y Node.js y tiene una gran comunidad de desarrolladores [21].

Entre las principales ventajas, se incluye su **facilidad de uso** y su **velocidad**, Jest tiene una sintaxis simple y fácil de entender, se puede empezar a escribir pruebas casi de inmediato, acertando significativamente la curva de aprendizaje.

Gracias al **aislamiento** y la capacidad para ejecutar **casos de prueba en paralelo**, es una de las herramientas más rápidas disponibles de forma gratuita. Adicionalmente, **Jest** viene equipado con una librería de aserciones y un corredor de pruebas integrados, por tanto, no es necesario utilizar ni configurar otras herramientas externas.

Jest también ofrece reportes sobre la **cobertura del código** y se puede determinar qué áreas del código están completamente cubiertas por los casos de prueba y en qué áreas se debe aumentar esa cobertura, esto permite **la toma de decisiones informada** para mejorar la calidad del sistema.

No obstante, Jest suele tener un **tamaño de archivo relativamente grande**, lo que puede afectar el tiempo que lleva iniciar las pruebas, por tanto, puede ser poco adecuado para proyectos con recursos limitados.

No es compatible con todos los *frameworks* y librerías basados en JavaScript, por tanto, integrarlo con tecnologías específicas puede requerir un esfuerzo adicional.

Por último, Jest puede tener una **curva de aprendizaje moderada**, especialmente para aquellos que son nuevos en las pruebas unitarias y las metodologías de prueba. Sin embargo, existen numerosos recursos de aprendizaje gratuitos y de gran calidad.

6.2 Desafíos presentados

En esta sección se presentan los desafíos a los que se ha hecho frente durante el proceso de diseño e implementación, estos desafíos han sido clave a la hora de elegir las tecnologías utilizadas, definir el alcance del proyecto, elegir la infraestructura necesaria, etc.

6.2.1 Reconocer y procesar diagramas de arquitectura de software

En esta sección se presenta el desafío que ha supuesto para el proyecto el **reconocimiento y procesamiento de diagramas** de arquitectura de software. Al inicio del proyecto, cuando se comienza la fase de investigación, la primera barrera técnica que se hizo frente fue precisamente como proporcionar a la herramienta de IA el diagrama.

En ese momento, como ya se ha mencionado con anterioridad, se analiza la viabilidad de utilizar Bard o GPT, Bard proporciona conexión a internet y GPT no la proporcionaba todavía. Inicialmente, cabría esperar que Bard fuera la herramienta elegida por ser capaz de **compartir recursos mediante direcciones web**, pero no se deseaba limitar la elección de la herramienta únicamente por este factor.

Por lo tanto, se comienza una **fase de investigación** sobre como distintas herramientas analizan estructuras similares, en este caso, grafos. En este momento se descubre **GraphSON**, un formato de archivo basado en *JSON* que se utiliza para describir grafos y sus propiedades. En la siguiente figura se puede observar un ejemplo de un grafo simple descrito en **GraphSON**.

```
{
  "graph": {
    "mode": "NORMAL",
    "vertices": [
      {
        "name": "Alice",
        "age": 31,
        "_id": "1",
        "_type": "vertex"
      },
      {
        "name": "Bob",
        "age": 27,
        "_id": "2",
        "_type": "vertex"
      }
    ],
    "edges": [
      {
        "type": "friends",
        "_id": "3",
        "_type": "edge",
        "_outV": "1",
        "_inV": "2",
        "_label": "knows"
      }
    ]
  }
}
```

Figura 23. Ejemplo de grafo descrito en GraphSON [25].

Como se puede observar, en *GraphSON* se ha de definir cada vértice o nodo y posteriormente definir las aristas del grafo. En este momento se considera crear una herramienta capaz de recibir un diagrama y transformarlo en el formato descrito, sería una herramienta compleja, pero factible.

No obstante, se siguen buscando alternativas destinadas a reducir el esfuerzo en esta parte, ya que esta herramienta sería imposible de probar en todos los escenarios posibles, que pueden ser infinitos.

Otra alternativa que se estudia es la capacidad de las herramientas de IA de reconocer diagramas exportados en *XML* de las principales herramientas utilizadas para diseñar diagramas: *Miro*, *diagrams.net*, *Figma*, etc. La herramienta es capaz de reconocer dichos formatos *XML* prácticamente con exactitud, por tanto, se posiciona como una alternativa viable a *GraphSON*.

Finalmente, una vez explorados los formatos que son capaces de interpretar ambas

herramientas, se presenta el segundo obstáculo. Las herramientas tienen un tamaño máximo de consulta que no puede exceder ciertos límites, por tanto, en muchos casos, generar la consulta con el diagrama incluido es inviable. No obstante, existen alternativas:

- Se puede enviar la consulta por partes, pero cada parte ha de tener sentido, de lo contrario la herramienta puede perder capacidad de interpretar la consulta y su contexto devolviendo una respuesta inválida.
- Se puede enviar directamente la dirección url pública del diagrama.
- Estudiar la capacidad de reconocimiento de imágenes de la herramienta.

Se decide escoger la segunda alternativa ya que reduce la intervención del sistema entre los parámetros recibidos y enviados a la herramienta. Además, estudiar la precisión del reconocimiento de imagen teniendo en cuenta la gran cantidad de escenarios posibles se considera inviable para el proyecto. Por tanto, se decide que la mejor alternativa es proporcionarle el diagrama al sistema mediante una **dirección url**, y estudiar si la herramienta es capaz de reconocer los diagramas proporcionados y evaluar sus características.

La principal desventaja de este sistema es que la dirección url debe de ser **pública** para que sea accesible por el sistema, por tanto, el diagrama no debería de contener datos de carácter privado. También existe la opción de limitar el acceso a la url durante un tiempo determinado, pero se considera fuera del alcance del proyecto.

No existen restricciones en cuanto a la herramienta a utilizar, en este proceso, los candidatos pueden utilizar cualquier pizarra que permita exportar el diagrama en un formato **XML**. Se han utilizado para las pruebas las herramientas **diagrams.net, Miro, Figma y Excalidraw** y el sistema es capaz de reconocer los archivos exportados de todas ellas, lo más importante es que el candidato etiquete los nodos correctamente y el sistema sea capaz de identificar qué es cada nodo.

6.2.2 Evaluación de diagramas de arquitectura de software

En esta sección se describe el proceso que se ha seguido para definir los parámetros que se evalúan de cada diagrama de arquitectura. Uno de los mayores desafíos de esta parte ha sido definir qué parámetros se evalúan para cada diagrama, además, permitiendo flexibilidad para que cada consumidor pueda evaluar los parámetros que considere más adecuados para sus intereses.

En esta fase del proyecto, se decidió que la mejor forma de resolver el problema es definiendo qué características se esperan dependiendo del nivel del candidato. No es lo mismo evaluar a un candidato que acaba de iniciarse en el mundo laboral y tiene los estudios básicos que a un ingeniero sénior con una década de experiencia.

Por lo tanto, un mismo diagrama puede ser completo o insuficiente dependiendo del nivel del candidato evaluado, es conveniente ofrecer **flexibilidad** en las evaluaciones. Este problema se ha resuelto definiendo primeramente qué aspectos se pueden evaluar de un diagrama de



Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

software, y después se han dividido esos aspectos dependiendo del nivel de experiencia del candidato. Se han considerado los siguientes niveles: Junior, Intermedio y Senior.

Los parámetros a evaluar elegidos, su descripción y la metodología de evaluación se pueden observar en la siguiente figura.

Parámetro	Descripción	Metodología de evaluación
Identificación de Componentes	Hace referencia a la capacidad del candidato de identificar los componentes necesarios en el diagrama de arquitectura de software.	Rate the candidate's ability to identify and include all necessary components in the architecture diagram. Consider whether the key components of the platform are correctly identified and depicted.
Requisitos	Los requisitos hacen referencia a las cualidades o características que ha de implementar un sistema, pueden ser de tipo funcional y no funcional.	Rate how the candidate has addressed all the requirements mentioned in the question description.
Interacciones	Las interacciones son las relaciones entre los distintos componentes del diagrama.	Rate the clarity and logical coherence of interactions between components in the architecture diagram. Consider whether the candidate has effectively illustrated how different parts of the system communicate.
Consistencia de Datos	Hace referencia refiere a la integridad y precisión de los datos almacenados en un sistema o base de datos	Rate the candidate's approach to addressing data consistency and security concerns in the design. Consider whether the candidate has described mechanisms for ensuring data integrity and security.
Escalabilidad	La capacidad de un sistema o aplicación para crecer y manejar un aumento en la carga o la demanda de manera eficiente, sin comprometer su rendimiento o funcionalidad	Rate the candidate's consideration of scalability in the architecture, especially for peak requests volume. Assess whether the candidate has included components or strategies to handle increased load.

Tolerancia a fallos	Se refiere a la capacidad de un sistema informático o una aplicación para continuar funcionando de manera adecuada y proporcionar un servicio esencial incluso cuando ocurren errores, fallas o interrupciones inesperadas	Rate the candidate's inclusion of fault tolerance mechanisms to ensure system availability. Consider whether the candidate has accounted for potential failures and recovery strategies.
Optimización de rendimiento	Hace referencia al proceso de mejorar la eficiencia y velocidad de un sistema, aplicación o proceso informático con el objetivo de lograr un mejor rendimiento y tiempo de respuesta. Esto implica identificar y eliminar cuellos de botella, mejorar algoritmos, ajustar la configuración y utilizar recursos de manera más eficaz para lograr un funcionamiento más rápido y eficiente.	Rate the candidate's strategies for optimizing performance, such as reducing latency in processing requests. Evaluate whether the candidate has addressed performance bottlenecks.
Autenticación de usuarios	Proceso de verificar la identidad de un usuario que intenta acceder a un sistema, aplicación o recurso digital	Rate the candidate's approach to user authentication and its inclusion in the architecture. Assess whether the candidate's design ensures secure user authentication.
Claridad en la representación	Hace referencia a la capacidad de representar el diagrama y sus componentes de una manera clara y concisa.	Rate the clarity and structure of the candidate's representation of the software architecture. Assess whether the candidate effectively communicates their design choices and rationale.
Disponibilidad y redundancia	<p>La disponibilidad se refiere a la capacidad de un sistema o servicio para estar en funcionamiento y accesible para los usuarios durante un período de tiempo determinado.</p> <p>La redundancia implica la duplicación de componentes críticos en un sistema con el fin de garantizar la continuidad del servicio en caso de que uno de los</p>	Rate the candidate's approach to ensuring high availability, including strategies for service redundancy, global load balancing, and minimizing single points of failure.

	componentes falle.	
Control de acceso y autorización	<p>El control de acceso se refiere al proceso de decidir quién tiene permiso para acceder a un recurso o sistema informático y en qué condiciones.</p> <p>La autorización es la etapa posterior al control de acceso y se refiere a la asignación de permisos y privilegios específicos a usuarios autenticados</p>	Rate the candidate's representation of authorization and access control mechanisms, considering role-based access control (RBAC), attribute-based access control (ABAC), and least privilege principles.
Particiones de sistemas Distribuidos	<p>En sistemas distribuidos, las particiones son divisiones lógicas o geográficas que separan los componentes de un sistema en grupos o segmentos separados.</p> <p>La tolerancia a particiones (Partition Tolerance) es uno de los tres aspectos del teorema CAP (Consistency, Availability, Partition Tolerance) que los diseñadores de sistemas distribuidos deben tener en cuenta.</p>	Rate the candidate's ability to design for and manage distributed system partitions, handling scenarios involving network partitions, data partitioning, and eventual consistency.
Patrones arquitectónicos avanzados	<p>Son enfoques de diseño de sistemas de software que abordan desafíos más complejos y específicos en la arquitectura de software. Estos patrones se utilizan para resolver problemas que van más allá de las estructuras arquitectónicas básicas y pueden incluir estrategias de diseño para aspectos como escalabilidad, seguridad, rendimiento y gestión de datos.</p>	Rate the candidate's incorporation of advanced architectural patterns such as event sourcing, Command Query Responsibility Segregation, micro-frontends, etc.

Figura 24. Parámetros a evaluar de un diagrama de arquitectura de software.

Adicionalmente, es necesario proporcionar flexibilidad para que los consumidores sean capaces de evaluar distintos parámetros que no se hayan tenido en cuenta, por lo tanto, es posible especificar nuevos parámetros específicos para la consulta incluyendo un campo en la petición. Si el sistema recibe campos adicionales, los incluirá en la petición, es importante que los consumidores del sistema especifiquen los parámetros con un alto grado de precisión.

Otro aspecto importante es la **escala de evaluación** utilizada, se ha demostrado mediante numerosas pruebas que los modelos son más precisos cuanto menor es la escala de evaluación. Por lo tanto, si se utiliza una escala de valores comprendidos en el siguiente rango [1-5] se obtiene una respuesta más precisa que si se especificara un rango del [1-10], esto es debido a la **simplificación de la evaluación** y **reducción de la ambigüedad**. En la siguiente figura se puede observar la escala de evaluación utilizada.

```
export const RATING_SCALE = {
  "1": "Lowest",
  "2": "Below Average",
  "3": "Average",
  "4": "Above Average",
  "5": "Highest",
};
```

Figura 25. Escala de evaluación utilizada en el sistema.

La escala de evaluación se transforma para dar a la herramienta un grado de flexibilidad en las evaluaciones, para evaluar el diagrama, se le especifica a cada característica su método de evaluación y se le pide al sistema que evalúe cada característica **independientemente**.

A continuación, se describe el proceso para generar consultas y como se ha conseguido aumentar la precisión de las respuestas obtenidas utilizando técnicas de generación de consultas.

6.2.3 La precisión de las respuestas obtenidas

En este apartado se describe el generador de consultas utilizado para aumentar la precisión de las evaluaciones y pedir a la herramienta que devuelva la respuesta en un formato determinado.

Para ello se ha diseñado un componente en TypeScript denominado **promptEngine**, este componente recibe los parámetros de la petición del cliente y genera una consulta válida y precisa para la tarea que se pretende realizar. Un ejemplo del código fuente de dicho programa se puede visualizar en la siguiente figura.

```
export const getEvaluationPrompt = (
  questionDescription: string,
  diagramUrl: string,
  candidateLevel: EngineeringLevel
```

```
) => `
You are a top-level Software Architect at one of the best companies of the
world, you just conducted an interview and the question that you asked was
the following:

${questionDescription}

Now, you will evaluate the software architecture design provided by the
candidate here:"${diagramUrl}".

Each feature should be assessed with this rating scale ${JSON.stringify(
  RATING_SCALE
)}, based on the following criteria:

${getFeaturesToEvaluatePrompt(candidateLevel)}

Let's work this out step-by-step to be sure we have the right answer, think
carefully about the assignment and the answers you will provide.
Please provide detailed feedback for each feature, explaining the strengths
and areas for improvement in the candidate's software architecture design.
Use the rating scale and criteria provided to guide your evaluation.
Please, before providing the answers and the feedback, think carefully if
the answers that you are just about to provide are the best ones based on
the description and make sense.

Your response should be in a JSON format with the following structure:
{
  result:[{
    "feature": "...",
    "explanation": "...",
    "mark": ...
  }]
}
`;

const getFeaturesToEvaluatePrompt = (candidateLevel: EngineeringLevel) => {
  const featuresToEvaluate = getEvaluationConfiguration(candidateLevel);
  return featuresToEvaluate
    .map(({ feature, criteria }) => `t- ${feature}:${criteria}`)
    .join("\n");
};
```

Figura 26. Código del generador de consultas.

En el código anterior se pueden observar distintas técnicas y características que aumentan la precisión de las consultas para las herramientas de **GPT** [22][23].

Uno de los principales factores que aumenta la precisión es el **contexto**. El contexto puede ayudar a GPT a comprender mejor la consulta. Por ejemplo, en el código anterior se le da a la

herramienta el contexto en el que está englobada la consulta, que es una entrevista con un candidato, que le ha hecho una pregunta y ahora es el momento de evaluar la solución propuesta.

Otro de los factores que ayuda a que la respuesta sea más relevante es el denominado **juego de roles**, que es la acción de dar instrucciones a un modelo de IA para que asuma un rol específico. Esto puede mejorar la precisión y pertinencia de sus respuestas en ciertos contextos. Al especificar un rol, proporcionas al modelo de IA un contexto y restricciones que pueden ayudar a generar respuestas más adecuadas desde el punto de vista contextual.

Sin embargo, la precisión de las respuestas de la IA todavía depende de varios factores, como la calidad de las instrucciones, la complejidad de la tarea y los datos de entrenamiento del modelo. Si bien el juego de roles puede ser efectivo para tareas que requieren una persona o conocimientos específicos en un dominio, no siempre garantiza una precisión perfecta, y aún puede haber casos en los que las respuestas de la IA sean inexactas o incompletas.

Por ejemplo, en el código anterior, se puede observar cómo se le pide a la IA que actúe como un Arquitecto de Software que trabaja para una de las mejores compañías del mundo, por tanto, el modelo ya comienza a valorar las respuestas que va a dar como si fuera un Arquitecto de Software y proporciona respuestas enmarcadas en ese contexto.

Otra de las técnicas que aumenta la precisión de las respuestas es la **autorreflexión**. La autorreflexión es una técnica que permite al modelo evaluar su propia respuesta y detectar posibles errores. Esto se consigue mediante el uso de un mecanismo de razonamiento que analiza la respuesta en comparación con la consulta original y, si se detecta un error, el modelo puede corregirlo de forma autónoma, esto puede aumentar la precisión de varias maneras.

En primer lugar, permite al modelo identificar y corregir sus propios errores, especialmente importante en casos de respuestas complejas o de larga duración, donde es más probable que se produzcan errores.

En segundo lugar, la autorreflexión puede ayudar al modelo a aprender de sus errores, por tanto, cuando el modelo identifica un error, puede almacenar esta información para evitar cometerlo en el futuro, y estaría retroalimentándose automáticamente.

Otro de los factores que aumenta la precisión de futuras respuestas es la **retroalimentación**. Cuando GPT genera una respuesta, es posible proporcionar retroalimentación sobre la precisión de la respuesta, pero esto aumenta la precisión a futuro, no de la consulta actual.

Según un estudio reciente realizado por el equipo de *OpenAI*, los creadores de *GPT*, se demostró que la autorreflexión puede mejorar significativamente la precisión de las respuestas del modelo. En el estudio, GPT-3 con autorreflexión obtuvo un **88%** de precisión en una tarea de respuesta a preguntas, frente al **67%** de precisión de GPT-3 sin autorreflexión [23].

Un ejemplo de autorreflexión en el código proporcionado es el siguiente: “Please, before providing the answers and the feedback, think carefully if the answers that you are just about to provide are the best ones based on the description and make sense.”. Está forzando al modelo a pensar si las respuestas que está a punto de dar son las mejores para la descripción de la tarea proporcionada y si tienen sentido en el contexto que se engloba la consulta.



7 Pruebas

En esta sección se describen las pruebas que se han realizado para **validar el proyecto realizado**, el apartado se centra en las pruebas de validación ya que se consideran las más interesantes en el proyecto, cuyo objetivo principal es validar la idea de que se puede automatizar la evaluación de diagramas de arquitectura de software.

Las **pruebas de validación** son un tipo de pruebas de software que se realizan para determinar si el software cumple con los requisitos especificados por los usuarios y los clientes. Estas pruebas se suelen realizar en la fase final del desarrollo del software, y antes de su puesta en producción.

El objetivo de este tipo de pruebas es asegurar que el software sea **capaz de cumplir con los requisitos del usuario**. Estas pruebas ayudan a identificar errores y defectos que pueden afectar negativamente a la experiencia del usuario, y que pueden provocar el rechazo del software por parte de los usuarios.

Las pruebas de validación se suelen realizar utilizando una variedad de técnicas, incluyendo:

- **Pruebas de aceptación del usuario (UAT):** Estas pruebas se realizan por los usuarios finales del software, para asegurarse de que cumple con sus necesidades y expectativas.
- **Pruebas de usabilidad:** Estas pruebas se realizan para evaluar la facilidad de uso del software.
- **Pruebas de seguridad:** Estas pruebas se realizan para identificar vulnerabilidades de seguridad en el software.
- **Pruebas de rendimiento:** Estas pruebas se realizan para evaluar el rendimiento del software en diferentes escenarios.

Esta sección se va a centrar en las pruebas de aceptación del usuario, ya que el objetivo de este apartado es validar si el sistema diseñado es capaz de resolver los problemas planteados al inicio del proyecto.

En cuanto a las pruebas, se han planteado siguiendo una forma iterativa, primeramente, se valida la idea con un sistema simple y se va aumentando la complejidad mientras el sistema siga proporcionando resultados fiables.

Estas pruebas **se realizan de forma manual**, por tanto, la validación se hace teniendo en cuenta tanto la puntuación obtenida en cada parámetro como la explicación que da la herramienta para detectar posibles fallos o alucinaciones del sistema. En IA, la alucinación hace referencia a cuando la IA proporciona una respuesta que no se identifica con la consulta proporcionada, por tanto, no tiene sentido en el contexto de esta.

En un futuro, las pruebas deberían **de ser automatizadas** utilizando una herramienta como **Postman**, anteriormente descrita. Estas pruebas automatizadas son necesarias para comprobar la **precisión y eficiencia** del sistema en el futuro **tras cada desarrollo** y seguir proporcionando un sistema fiable y válido para el propósito que se ha considerado.



7.1 Evaluación del diagrama de un sistema simple

En esta sección se detallan las pruebas realizadas para validar el sistema con **arquitecturas de software simples**. Se han hecho pruebas con cinco cuestiones simples y tres respuestas por cuestión, una respuesta positiva y una respuesta negativa, para evaluar la capacidad del sistema de proponer mejoras e identificar problemas en el diagrama presentado por el usuario. Las cuestiones utilizadas se pueden ver en el anexo I.

A continuación, se muestra **el caso de prueba** con una solución que debería de ser buena, se incluye la descripción de la tarea que el candidato ha de resolver, el nivel esperado del candidato y la solución del candidato.

```
const input = {
  questionDescription: `
    You are tasked with designing the architecture for an e-commerce platform.
    This platform should support user registration, product catalog management,
    shopping cart functionality, order processing, and payment processing.
    Please provide a high-level software architecture diagram that illustrates the key components and their interactions.
    Highlight how you would ensure data consistency and security.
  `,
  engineeringLevel: 0, // 0 is junior, 1 is intermediate, 2 is senior.
  candidateSolution:
    "https://drive.google.com/file/d/1scIYW3XzXK2lkCwkpkokAjoF8x0iaqtPi/view?usp=sharing",
};
```

Figura 27. Ejemplo de caso de prueba sencillo.

El diagrama proporcionado por el candidato es el siguiente.

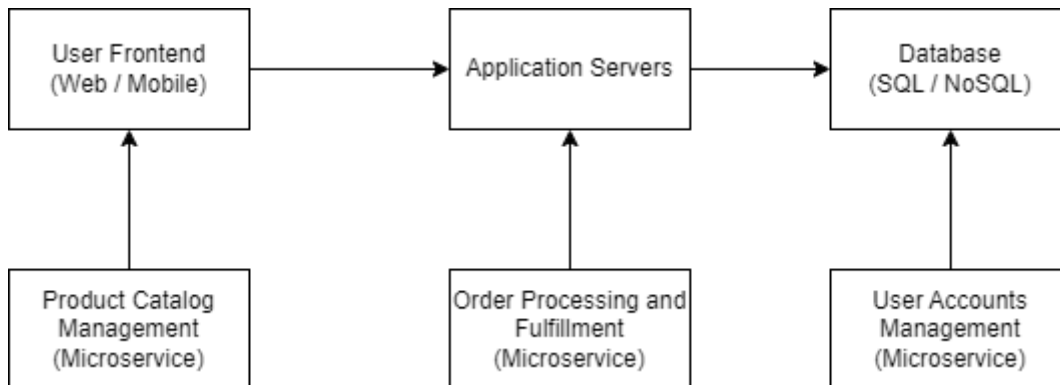


Figura 28. Solución propuesta por el candidato.

Utilizando la estrategia de generación de *prompts* basada en el nivel del candidato presentada anteriormente, se ha obtenido el *prompt* visible en el anexo II, y la respuesta de la herramienta es la siguiente.

```
{
  "result": [
    {
      "feature": "Component Identification",
      "explanation": "The candidate effectively identified key components such as User Frontend, Application Servers, Product Catalog Management, Order Processing and Fulfillment, User Accounts Management, and Databases. The microservices within Application Servers are also
```

```

recognized.",
  "mark": 5
},
{
  "feature": "Interactions",
  "explanation": "The interactions between components are clearly
defined and well-structured in the architecture diagram. The candidate
illustrates how different parts of the system communicate, indicating a
strong understanding of system interactions.",
  "mark": 5
},
{
  "feature": "Data Consistency",
  "explanation": "The candidate demonstrates a robust approach to
data consistency and security. They mention the use of both SQL and
NoSQL databases, which suggests an understanding of data diversity.
However, specific mechanisms for ensuring data integrity and security
could be elaborated further.",
  "mark": 4
},
{
  "feature": "User Authentication",
  "explanation": "User authentication is considered and included in
the architecture. While it's mentioned, additional details about
authentication methods and security protocols could enhance the
design.",
  "mark": 4
},
{
  "feature": "Clarity of Representation",
  "explanation": "The candidate's representation of the software
architecture is clear and well-structured. They effectively communicate
their design choices and rationale, making it easy to understand the
proposed solution.",
  "mark": 5
}
]
}

```

Figura 29. Resultado de la evaluación de una arquitectura correcta.

A continuación, se va a generar el mismo *prompt* pero con una arquitectura que no debería ser evaluada como correcta, ya que tiene serias deficiencias, la arquitectura proporcionada es la siguiente.

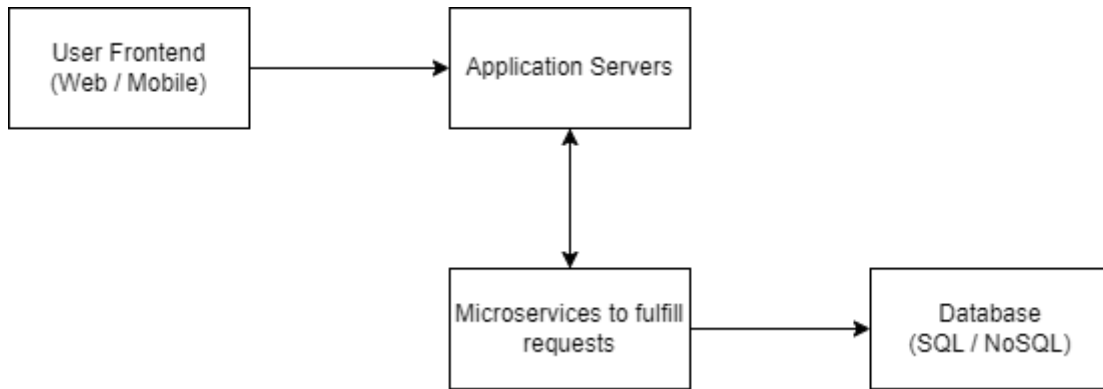


Figura 30. Diagrama deficiente para evaluar un ingeniero Junior

La evaluación de la herramienta se muestra en la siguiente figura, se puede observar como el sistema es capaz de identificar correctamente las deficiencias del diagrama evaluado, como la falta de un componente encargado de autenticar al usuario.

```
{
  {
    "result": [
      {
        "feature": "Component Identification",
        "explanation": "The candidate's diagram lacks clarity in component identification. It includes only the most basic components and crucial parts such as a database and payment processing are missing.",
        "mark": 2
      },
      {
        "feature": "Interactions",
        "explanation": "Interactions between components are not clearly defined in the candidate's diagram. Nevertheless, there are some interactions that fail to illustrate data flow and communication pathways, leading to a lack of coherence.",
        "mark": 3
      },
      {
        "feature": "Data Consistency",
        "explanation": "The candidate's design completely overlooks data consistency and security concerns. There is no mention of databases or security mechanisms, indicating a critical deficiency.",
        "mark": 2
      },
      {
        "feature": "User Authentication",
        "explanation": "User authentication is entirely absent from the candidate's architecture. There is no consideration or inclusion of authentication methods or security protocols.",
        "mark": 2
      }
    ]
  }
}
```

```

{
  "feature": "Clarity of Representation",
  "explanation": "The representation lacks detail and clarity. Components are not labeled, and there is no structure to explain the design choices. It fails to effectively communicate the architecture.",
  "mark": 2
}
]
}

```

Figura 31. Ejemplo de evaluación de un diagrama deficiente para un ingeniero junior.

Después de identificar las deficiencias de la arquitectura, se desea comprobar si es capaz de identificar una pequeña mejora sobre la arquitectura deficiente, por ejemplo, añadiendo una base de datos y especificando los microservicios, pero la base de datos no se conecta con ningún componente.

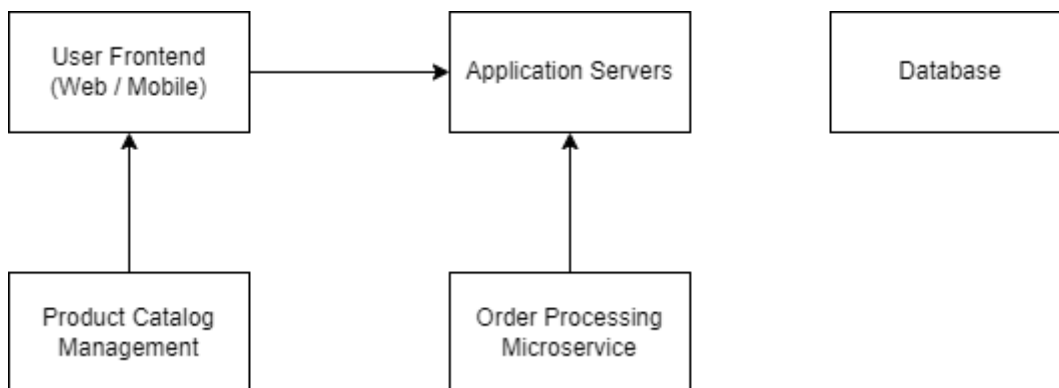


Figura 32. Diagrama mejorado para evaluar un ingeniero Junior.

En este caso, **la herramienta es capaz de identificar las mejoras**, no obstante, al evaluación no se basa en el diagrama anterior y es capaz de aislar cada evaluación, eliminando, por tanto, la necesidad de comparar soluciones para calificar los resultados y evaluando a cada candidato de manera independiente, proporcionando el siguiente resultado.

```

{
  "result": [
    {
      "feature": "Component Identification",
      "explanation": "The candidate's diagram effectively identifies key components, including User Frontend, Application Servers, Microservices, and a Database. These components cover the core functionalities of the e-commerce platform.",
      "mark": 4
    },
    {
      "feature": "Interactions",
      "explanation": "The interactions between User Frontend, Application Servers, Microservices, and the Database are not explicitly illustrated. While the components are present, there is a lack of clarity regarding

```

```
how they communicate.",
  "mark": 2
},
{
  "feature": "Data Consistency",
  "explanation": "The candidate's design mentions the use of a
database, which is a positive step toward addressing data consistency.
However, it lacks specific mechanisms for ensuring data integrity and
security. Providing more details on these aspects would be beneficial.",
  "mark": 3
},
{
  "feature": "User Authentication",
  "explanation": "User authentication is not explicitly addressed in
the candidate's architecture. Including authentication methods and
security protocols is crucial for a secure e-commerce platform.",
  "mark": 2
},
{
  "feature": "Clarity of Representation",
  "explanation": "The representation of the software architecture
lacks clarity in illustrating interactions between components.
Components are identified but need to be more clearly connected to show
how data flows and how different parts of the system communicate.",
  "mark": 2
}
]
}
```

Figura 33. Evaluación de un diagrama mejorado para un ingeniero Junior.

Como se puede observar, la herramienta proporciona una evaluación clara y concisa de diagramas sencillos, una vez validada la idea para cuestiones simples, se procede a aumentar la dificultad para ver si la herramienta sigue siendo acertada con un mayor número de parámetros y de complejidad.

7.2 Evaluación de un diagrama de complejidad intermedia

En esta sección, se explica el proceso seguido para validar como la herramienta evalúa un diagrama de arquitectura software de complejidad media. Se sigue el mismo proceso iterativo que en la cuestión anterior, pero eliminando la opción de presentar un diagrama parcialmente válido.

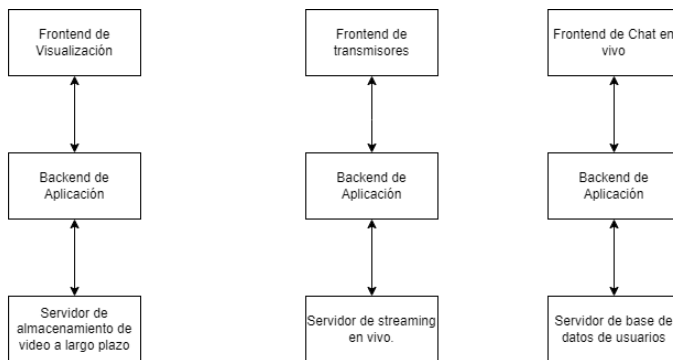
Primeramente se proporciona a la herramienta un diagrama completo y con anotaciones, se evalúa si la herramienta es capaz de reconocer el diagrama, sus componentes, interacciones y si es capaz de evaluar los requisitos para un ingeniero de nivel intermedio.

Uno de los casos de prueba utilizado para evaluar candidatos de nivel intermedio se puede observar en la siguiente figura. No obstante, en el anexo III se encuentran todas las cuestiones utilizadas para esta fase de las pruebas de validación.

```
const input = {
  questionDescription: `
  You are tasked with designing the architecture for a live streaming platform like Twitch.
  This platform should support the following key components and their interactions:
  • User Registration and Authentication
  • Live Streaming
  • Chat and Interaction Features
  • Video Storage and Playback
  • Recommendation System
  Please provide a high-level software architecture diagram that illustrates these key components and their interactions.
  Highlight how you would ensure scalability, performance, and high availability in your design.
  `,
  engineeringLevel: 1, // 0 is junior, 1 is intermediate, 2 is senior.
  candidateSolution:
  "https://drive.google.com/file/d/1scIVw3XzXK2lkWkCkopkA0jf8x0iaqTpi/view?usp=sharing",
};
```

Figura 34. Caso de prueba para evaluación de un candidato de nivel intermedio.

El diagrama proporcionado para la evaluación positiva se puede observar en la siguiente figura, el *prompt* generado se puede ver en el anexo IV, la principal diferencia que se puede observar es que para un ingeniero de nivel intermedio se han introducido parámetros nuevos de evaluación como la escalabilidad, tolerancia a fallos y optimización de rendimiento.



Anotaciones:

- **Frontend de Visualización:** Interfaz de usuario para visualizar transmisiones en vivo y chats.
- **Frontend de Transmisores:** Interfaz de usuario para los transmisores que realizan las transmisiones en vivo.
- **Chat en Vivo:** Interfaz de usuario para participar en chats en tiempo real durante las transmisiones.
- **Backend de Aplicación:** Lógica de aplicación que gestiona el enrutamiento de video, el chat y la autenticación.
- **Servidor de Almacenamiento de Video a Largo Plazo:** Almacena grabaciones de transmisiones en vivo para su visualización posterior.
- **Servidor de Streaming en Vivo:** Gestiona la transmisión en tiempo real de video y chats durante las transmisiones en vivo.
- **Base de Datos de Usuarios:** Almacena perfiles de usuarios, detalles de inicio de sesión y otra información relacionada con usuarios.
- **Conexiones de Interfaz de Usuario:** Representa las conexiones entre los frontends y el backend de la aplicación.
- **Conexiones de Backend:** Representa las conexiones entre el backend de la aplicación y los servidores de almacenamiento y la base de datos.
- **Conexiones de Transmisión:** Representa las conexiones entre el backend de la aplicación y el servidor de streaming en vivo durante las transmisiones.
- **Conexiones de Chat:** Representa las conexiones entre el frontend de chat en vivo y el backend de la aplicación para chats en tiempo real.

Figura 35. Diagrama de ejemplo para la evaluación de un candidato de nivel intermedio.

La evaluación que el sistema ha hecho sobre este diagrama es la siguiente. Como se puede observar el sistema es capaz de analizar las fortalezas y debilidades del candidato, además de sugerir mejoras que podrían mejorar la evaluación final.



```
{
  "result": [
    {
      "feature": "Component Identification",
      "explanation": "The candidate has successfully identified and included all the necessary components in the architecture diagram. The key components, including User Registration and Authentication, Live Streaming, Chat and Interaction Features, Video Storage and Playback, and the Recommendation System, are correctly identified and depicted.",
      "mark": 5
    },
    {
      "feature": "Interactions",
      "explanation": "The interactions between components are logically coherent and well-illustrated. The candidate has effectively depicted how different parts of the system communicate, such as users interacting with the Chat and Interaction Features during live streaming.",
      "mark": 5
    },
    {
      "feature": "Data Consistency",
      "explanation": "The candidate's approach to data consistency and security is well-considered. They have described mechanisms for ensuring data integrity and security, especially important in a live streaming platform where user data and content need protection.",
      "mark": 4
    },
    {
      "feature": "Scalability",
      "explanation": "The candidate has considered scalability in the architecture. While they have included components that suggest scalability, such as load balancers, they could provide more specific details on how the system handles peak request volume.",
      "mark": 3
    },
    {
      "feature": "Fault Tolerance",
      "explanation": "Fault tolerance mechanisms are somewhat included in the design, with redundant components like load balancers and multiple servers. However, the candidate could elaborate more on how the system recovers from potential failures.",
      "mark": 3
    },
    {
      "feature": "Performance Optimization",
      "explanation": "Performance optimization strategies are somewhat addressed, but the candidate could provide more specific details on how they plan to reduce latency and address potential performance
```

```

bottlenecks, especially in live streaming scenarios.",
  "mark": 3
},
{
  "feature": "User Authentication",
  "explanation": "User authentication is appropriately included in
the architecture, ensuring secure user access to the platform.",
  "mark": 4
},
{
  "feature": "Clarity of Representation",
  "explanation": "The architecture diagram is well-structured and
effectively communicates the design choices and rationale. However,
adding labels or descriptions to components and connections could
further improve clarity.",
  "mark": 4
}
]
};

```

Figura 36. Evaluación del sistema proporcionado para un ingeniero de nivel intermedio.

Como se puede observar, la herramienta **es capaz de reconocer los componentes y las interacciones entre los componentes**, además de **evaluar la escalabilidad**, es capaz de **sugerir posibles mejoras** que incrementarían la calidad del diagrama evaluado. Actualmente no se están teniendo en cuenta las mejoras para el consumidor, pero resulta interesante considerarlas en la fase de pruebas para evaluar si el sistema es capaz de identificar las mejoras y sugerirlas en un futuro.

Algunas de las mejoras sugeridas para mejorar la escalabilidad proporcionadas por el sistema son las siguientes:

- **Arquitectura de microservicios:** Puedes agregar un nuevo componente que represente la arquitectura de microservicios. Esto podría incluir microservicios separados para Live Streaming, Chat y User Interactions. Cada microservicio es escalable de forma independiente.
- **Autoescalado:** En la capa de Servidores de Aplicación, puedes añadir una capa adicional de "Auto-Scaling Group" que representa la capacidad de escalar automáticamente según la demanda. Conecta esta capa con los componentes relevantes, como Live Streaming y Chat.
- **CDN (Content Delivery Network - Red de entrega de contenido):** Agrega un nuevo componente "CDN" conectado entre Frontend de Usuario y Almacenamiento de Video. Esto representa la distribución de contenido en una red de entrega de contenido para reducir la carga en los servidores.

En cuanto a la mejora de rendimiento, el sistema ha sugerido las siguientes mejoras para el diagrama proporcionado.

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

- **Caché:** Introduce componentes de caché en la capa de Servidores de Aplicación y Almacenamiento de Video. Utiliza etiquetas para identificar qué datos se almacenan en caché y dónde. Conecta estos componentes con las partes relevantes del sistema.
- **Compresión de contenido:** Agrega un nuevo componente "Content Compression" en la capa de Almacenamiento de Video para representar la compresión de video y audio. Conecta este componente con la transmisión de video.
- **Balanceo de carga:** Añade un componente "Load Balancer" en la capa de Servidores de Aplicación y conéctalo a los microservicios correspondientes. Etiqueta los enlaces para indicar el equilibrio de carga.
- **Precarga de contenido:** Agrega una nueva función de "Content Pre-fetching" en la capa de Frontend de Usuario y conéctala con las partes del sistema que precargan contenido.
- **Optimización de la transmisión de contenido:** En la capa de Almacenamiento de Video, incluye un nuevo componente para representar la optimización de la transmisión de video y conecta esto con la transmisión de video.

Como se puede intuir con la prueba que se ha descrito, en esta fase de pruebas el sistema es completamente válido y proporciona resultados

7.3 Evaluación del diagrama de un sistema complejo

En esta sección, se presenta **la última fase de pruebas**, habiendo validado el sistema para diagramas de arquitectura simples y de nivel intermedio, esta prueba pretende demostrar si el sistema es capaz de reconocer y evaluar los diagramas de arquitectura más complejos.

Para ello se ha elegido una de las preguntas más comunes en el proceso de entrevistas de diseño de sistemas entre las compañías más grandes, que es diseñar un sistema de mensajería como *WhatsApp*. Los requisitos que se le dan al candidato son los siguientes:

- Soportar mensajes privados entre usuarios.
- Soportar chats grupales.
- Mostrar al usuario si su mensaje se ha recibido y leído.
- Ofrecer la funcionalidad de última conexión.
- Implementar notificaciones push.
- Ofrecer compartir y visualizar documentos, imágenes, grabaciones y videos.
- Proporcionar **encriptación punto a punto**.
- Permitir al usuario configurar opciones de privacidad.

Como solución se ha proporcionado el siguiente diagrama extraído de una conocida página web que ofrece cursos para preparar este tipo de entrevistas.

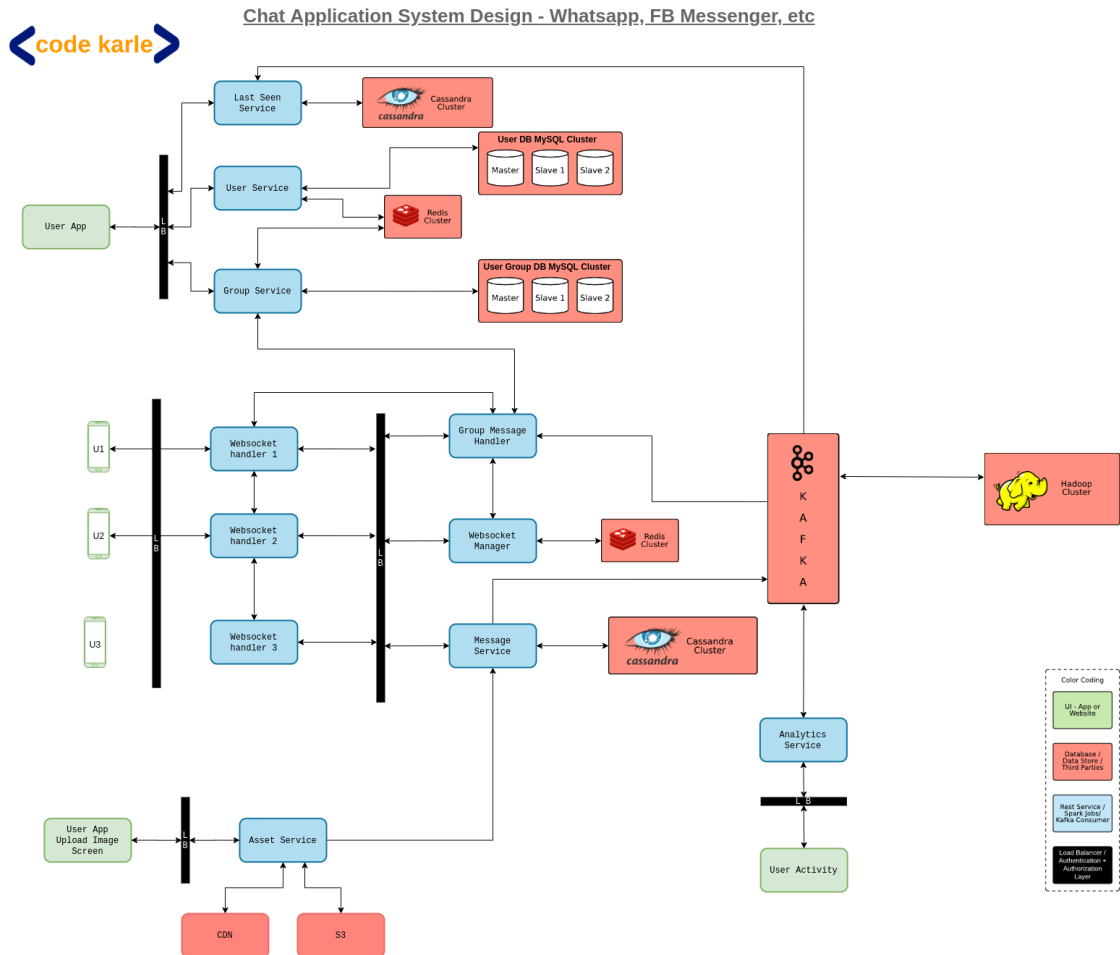


Figura 37. Diagrama complejo proporcionado para evaluar las capacidades de la herramienta [26].

La consulta generada mediante la herramienta puede observarse en el anexo V, se ha añadido un nuevo requisito a evaluar para este diagrama, se le pide al sistema que califique si el candidato ha cumplido con los requisitos de la descripción de la tarea.

En este caso como se puede observar en la siguiente figura, el sistema es capaz de evaluar las distintas capacidades que se le han exigido, y además proporciona detalles sobre la evaluación de cada requisito.

```

{
  "result": [
    {
      "feature": "Component Identification",
      "explanation": "The candidate's architecture diagram identifies most of the key components required for a chat app like WhatsApp, including clients, servers, databases, and external services. However, some components like load balancers and authentication services are missing.",
      "mark": 4
    },
    {
      "feature": "Interactions",

```



```
"explanation": "The interactions between components are reasonably clear, illustrating how clients communicate with servers and databases. However, there could be more detail on how the load balancers distribute requests to the servers for improved clarity.",
  "mark": 4
},
{
  "feature": "Data Consistency",
  "explanation": "The candidate mentions end-to-end encryption, which is a crucial security feature. However, they should include mechanisms for data consistency, such as database replication and backup strategies, to ensure data integrity and security.",
  "mark": 4
},
{
  "feature": "Scalability",
  "explanation": "The candidate mentions horizontal scaling for servers, which is a good start. However, they should provide more details on how this scaling would be achieved, including load balancing strategies, to handle increased load effectively.",
  "mark": 4
},
{
  "feature": "Fault Tolerance",
  "explanation": "The candidate includes multiple servers, which can enhance fault tolerance. However, there's no explicit mention of redundancy for critical services like databases or how the system handles server failures and recovers.",
  "mark": 3
},
{
  "feature": "Performance Optimization",
  "explanation": "There's no specific mention of performance optimization strategies in the diagram. The candidate should have considered techniques like caching, content delivery networks (CDNs), and database optimization for better performance.",
  "mark": 2
},
{
  "feature": "User Authentication",
  "explanation": "User authentication is mentioned, but the candidate should have included a dedicated authentication service for added security. Secure authentication mechanisms like OAuth2 or JWT should also be considered.",
  "mark": 3
},
{
  "feature": "Clarity of Representation",
  "explanation": "The diagram is relatively clear and well-
```

```

structured but lacks some detail in certain areas. The candidate should
improve labeling and use consistent symbols for components. Additional
documentation may be necessary to explain certain choices.",
  "mark": 4
},
{
  "feature": "Availability and Redundancy",
  "explanation": "The candidate mentions multiple servers, which can
contribute to availability. However, they should consider strategies for
service redundancy, global load balancing, and minimizing single points
of failure for a more robust architecture.",
  "mark": 3
},
{
  "feature": "Authorization and Access Control",
  "explanation": "There's no explicit representation of
authorization and access control mechanisms in the diagram. The
candidate should have considered RBAC, ABAC, and least privilege
principles for user permissions on the group chats.",
  "mark": 2
},
{
  "feature": "Distributed System Partitions",
  "explanation": "The candidate does not address distributed system
partitions or network partition scenarios. They should have considered
strategies for data partitioning and eventual consistency.",
  "mark": 1
},
{
  "feature": "Advanced Architectural Patterns",
  "explanation": "The candidate does not incorporate advanced
architectural patterns such as event sourcing, CQRS, or micro-frontends.
These patterns could enhance system scalability and flexibility.",
  "mark": 1
},
{
  "feature": "Requirements",
  "explanation": "The candidate's architecture design addresses most
of the functional and non-functional requirements mentioned in the
question description. However, improvements are needed in data
consistency, fault tolerance, performance optimization, and
authorization.",
  "mark": 3
}
]
}

```

Figura 38. Evaluación del sistema para un diagrama de arquitectura más complejo.

Desarrollo de un sistema para la evaluación automática de cuestiones sobre arquitectura de software

No obstante, el sistema no se puede probar con todos y cada uno de los numerosos escenarios y se ha de tener en cuenta que una validación humana de los resultados es necesaria, ya que en algunas ocasiones podría dar una evaluación errónea en cualquiera de los parámetros, especialmente en aquellos parámetros que sean especificados por el consumidor, esto se considerará en la sección de futuras mejoras.

Una vez validada la idea para sistemas de mayor complejidad, se concluyen las pruebas de validación y se concluye que el sistema realiza las funciones para las que se ha diseñado, el siguiente paso es identificar las áreas de mejora y preparar el sistema para la puesta a producción.

8 Relación con los estudios de máster

En este apartado se realiza una relación entre los estudios de máster, concretamente las asignaturas, y la adquisición de conocimientos que han contribuido en mayor medida al desarrollo del proyecto.

La principal asignatura que mayor relación tiene con el proyecto es la asignatura de **Sistemas inteligentes** de primer curso, esta asignatura supone el primer contacto con sistemas inteligentes más avanzados que los estudiados en el grado de Ingeniería Informática, ya que en el mundo laboral todavía no se había trabajado con sistemas inteligentes avanzados.

Otra asignatura relacionada en gran medida con el proyecto es **Ciencia de datos**, en esta asignatura, se estudian técnicas de aprendizaje automático para generar **modelos de regresión** utilizando técnicas como el *clustering*. En esa asignatura, se decide desarrollar como proyecto final un sistema que utiliza técnicas de aprendizaje automático y análisis de datos para generar recomendaciones musicales a partir del análisis de atributos de cada canción, este proyecto ha sido el primer contacto con modelos de aprendizaje automático.

En cuanto a la parte de infraestructura del proyecto, cabe destacar la asignatura **Servicios y aplicaciones distribuidas**, donde se estudia **el diseño y la arquitectura** para diseñar aplicaciones distribuidas en la nube. En este caso, ya se posee experiencia laboral previa en entornos *cloud*, no obstante, se considera que los contenidos presentados en la asignatura, en especial, en la parte de prácticas son un buen punto de contacto para comenzar a diseñar sistemas teniendo en cuenta su infraestructura en la nube.

Una de las fases más complicadas en un proyecto de esta naturaleza, de investigación, es la planificación y la gestión del proyecto. En este sentido, la asignatura de **Planificación y Dirección de proyectos TI** ha contribuido en gran medida a la finalización de este proyecto. Siendo capaces de identificar los riesgos asociados al proyecto, establecer planes de mitigación o contingencia para cada riesgo y siendo capaces de analizar y establecer planes alternativos en caso de que ese riesgo se materializara.

Por último, cabe mencionar la asignatura de **Ciberseguridad**, que, aunque no haya tenido un papel fundamental en el desarrollo del proyecto por su temática, ha proporcionado **conocimientos y competencias transversales necesarias** en el desarrollo de este. Por su metodología de evaluación, incita al alumnado a realizar trabajos de investigación y validar ideas, buscar y probar distintas herramientas para resolver un problema de seguridad.

En general, cabe destacar que cada asignatura aporta una serie de competencias transversales y habilidades blandas que, en algunas ocasiones, son incluso más importantes que los conocimientos de la propia asignatura. Por ejemplo, **el trabajo en equipo, la investigación, el desarrollo de un proyecto y la propia comunicación** tanto entre miembros del grupo de trabajo como en las presentaciones de proyectos y trabajos, que enseñan al alumno tratar con colaboradores de distinto nivel técnico, muy útil para comunicarse en el mundo laboral, donde no todos los compañeros de equipo poseen los mismos conocimientos técnicos.

9 Conclusiones

El objetivo principal del proyecto era diseñar e implementar una solución automatizada para calificar los conocimientos de arquitectura de software de los candidatos en procesos de selección de Ingeniería de Software. Por tanto, se considera que el objetivo principal del proyecto se ha cumplido.

Otro de los objetivos era reducir la subjetividad en la evaluación de las soluciones propuestas por los candidatos utilizando técnicas de inteligencia artificial y aprendizaje automático, es una realidad que no se ha implementado directamente en el proyecto aprendizaje automático o inteligencia artificial, pero se han utilizado tecnologías desarrolladas con estas técnicas como Bard.

Se ha conseguido también mejorar la igualdad de condiciones entre los candidatos ya que la solución automatizada no distingue entre los candidatos y solamente valora sus respuestas sin tener en cuenta ninguna característica identificativa de los candidatos, por tanto, la fiabilidad y eficiencia del proceso se han aumentado al automatizar el proceso.

El último objetivo principal trataba de reducir los costes asociados a los recursos técnicos utilizados en el proceso de selección. La herramienta todavía no puede sustituir completamente el componente humano en el proceso de selección ya que es necesario que un experto revise los resultados de la calificación proporcionados por el sistema, pero se puede complementar la evaluación automática con la verificación del experto, y, por tanto, se reduce el tiempo asociado al proceso.

Además de los objetivos principales, se consideraron distintos objetivos secundarios que en gran medida también se han conseguido cumplir.

El principal objetivo secundario era investigar las técnicas existentes para evaluar las soluciones de arquitectura de software. Este objetivo se ha conseguido ya que se ha utilizado nuevas tecnologías para implementar el proyecto. Además, la solución también se ha validado inicialmente para diagramas y soluciones de distinta complejidad, obteniendo resultados muy favorables.

Un objetivo secundario que no se ha podido cumplir por motivos externos es evaluar el impacto y la aceptación de la solución automatizada para el proceso de selección, ya que el proyecto no se encuentra en la fase de producción y no se ha probado todavía en sistemas reales.

Para concluir, cabe considerar que la mayor dificultad del proyecto ha radicado en el proceso de análisis e investigación más que en la propia implementación. Cabe considerar que, desde el inicio del proyecto, hasta su finalización, las tecnologías investigadas han mejorado a una velocidad increíble y, en muchas ocasiones, se ha propiciado un replanteamiento de ciertas decisiones.

10 Futuras mejoras

Una vez concluido el proyecto, se analiza de una forma objetiva **cuáles son sus carencias** y se proponen ciertas **mejoras para** aumentar la calidad, eficiencia, usabilidad y viabilidad del sistema diseñado.

La principal mejora que se considera para el proyecto es la **capacidad de recibir retroalimentación por parte de los consumidores** para ayudar al entrenamiento del modelo y aumentar la precisión y calidad de las respuestas, esto incrementaría en un futuro la precisión de las respuestas obtenidas y permitiría la etiquetación de los resultados, este sería el primer paso para la siguiente mejora que se desea implementar.

Uno de los mayores problemas que actualmente tiene el proyecto es el **rendimiento**, el modelo tarda una **media de 30 segundos en responder** a cada petición. El sistema es escalable gracias al diseño y la infraestructura, no obstante, se pretende entrenar un modelo de aprendizaje automático capaz de calcular el grado de similitud entre los diagramas, la tarea descrita y el nivel del candidato y que actúe como una caché. Esta solución eliminaría la dependencia del proyecto en Bard y sus evaluaciones, respondiendo de manera más rápida.

Otra de las mejoras contempla la posibilidad de **sustituir el modelo** con el que se integra el sistema, que es **Bard, por GPT-4 plus**, que ofrece mayores ventajas. Uno de los motivos por los que se descartó GPT-4 es porque **no proporcionaba conectividad** a internet en el momento en el que se estaba implementando el proyecto, hoy por hoy, con GPT-4 plus este impedimento se ha resuelto y además actualmente ofrece mejores características que Bard.

Actualmente, se utiliza una sola instancia de Bard para evaluar al candidato, cuando se haga la migración a GPT-4 plus, se puede **explorar la opción de generar un comité de evaluación conjunto** con distintas instancias de la IA. Una idea de implementación sería lanzar la misma pregunta a, por ejemplo, tres instancias de la IA, y conseguir que las IA debatan sobre los resultados obtenidos hasta que converjan en una decisión.

A parte de estas mejoras, el proyecto necesita trabajo para permitir su puesta en producción. A continuación, se listan las mejoras necesarias para su puesta en producción.

- La infraestructura es básica para validar la idea, pero no sería capaz de soportar un gran volumen de peticiones ya que se utiliza la versión gratuita de AWS. Habría que **implementar una infraestructura más avanzada** con *routing*, utilizar *API Gateway* y distintas subredes para no exponer la API al público, etc.
- **Proporcionar mecanismos de autenticación y autorización más avanzados** para los consumidores, permitir que sean capaces de crear y eliminar sus propias claves de API de forma automática.
- **Implementar mecanismos de monitorización de uso** para los distintos consumidores, necesario para conseguir establecer planes de pago por uso de la herramienta.

En general, el proyecto se ha considerado un producto mínimo viable para validar una idea, el siguiente paso es trabajar para mejorar aspectos clave de la implementación y preparar el producto para su puesta en producción, aparte de realizar distintas mejoras sustanciales.



11 Referencias

- [1] Peña, M. T., Carballal, A., Rodriguez-Fernandez, N., Santos, I., & Romero, J. a. P. (2021). *Artificial intelligence applied to conceptual design. A review of its use in architecture. Automation in Construction*, 124, 103550. <https://doi.org/10.1016/j.autcon.2021.103550>
- [2] Statista. (2023, June 14). *Inteligencia artificial: valor de mercado mundial 2021-2030*. <https://es.statista.com/estadisticas/1139768/inteligencia-artificial-valor-de-mercado/>
- [3] PricewaterhouseCoopers. (n.d.). How will automation impact jobs? PwC. <https://www.pwc.co.uk/services/economics/insights/the-impact-of-automation-on-jobs.html>
- [4] De Zárate Alcarazo, L. O. (2022). *Explicabilidad (de la inteligencia artificial)*. *Eunomía*, 22, 328–344. <https://doi.org/10.20318/eunomia.2022.6819>
- [5] Sorte, B. W., Joshi, P., & Jagtap, V. (2015). *Use of Artificial Intelligence in Software Development Life Cycle: A state of the Art Review*. ResearchGate. https://www.researchgate.net/publication/274254538_Use_of_Artificial_Intelligence_in_Software_Development_Life_Cycle_A_state_of_the_Art_Review
- [6] Saini, D. (2016). *Applications of Various Artificial Intelligence Techniques in Software Engineering*.
- [7] *IEEE Recommended Practice for Software Requirements Specification*. <https://personal.utdallas.edu/~chung/RE/IEEE830-1993.pdf>
- [8] Ramseyjiang. (2023, June 11). *An Overview of TDD, EDD, BDD, and DDD | Level Up Coding*. Medium. <https://levelup.gitconnected.com/an-overview-of-tdd-edd-bdd-and-ddd-e1c15e7e1e64>
- [9] Beck, K. (2002). *Test-Driven Development: By Example*. Addison-Wesley Professional.
- [10] Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.
- [11] Hanes, M. (2023). *Why Test-Driven Development (TDD)*. Marsner Technologies. <https://marsner.com/blog/why-test-driven-development-tdd/>
- [12] O. Agudelo, F. Riveros-Sanabria, S. Valbuena (2021) *Evaluación de una Arquitectura de Software, Prospectiva, Vol 19, N° 2, 2021*.
- [13] ATAM activities that consider scenarios. ATAM phases. (n.d.). ResearchGate. https://www.researchgate.net/figure/ATAM-activities-that-consider-scenarios-Fig-6-ATAM-phases-29_fig3_3188246
- [14] D. Khurana, A. Koli, K. Khatter, and S. Singh (2023). Natural language processing: state of the art, current trends and challenges. <https://link.springer.com/10.1007/s11042-022-13428-4>



- [15] Ahmed, I., Kajol, M., Hasan, U., Datta, P. P., Roy, A., & Reza, M. R. (2023). ChatGPT vs. Bard: A Comparative Study. TechRxiv, 23536290. https://www.techrxiv.org/articles/preprint/ChatGPT_vs_Bard_A_Comparative_Study/23536290
- [16] Robin y Mohammed Amine. (2023, 2 de marzo). Stateless REST APIs - Advantages of Statelessness in REST. <https://restfulapi.net/statelessness/>
- [17] Cockburn, A. (2011). The C4 model: A visual approach to software architecture. Pragmatic Programmers. <https://c4model.com/>
- [18] Referencias APA – Normas APA. (n.d.). <https://normas-apa.org/referencias/>
- [19] AWS Lambda – Preguntas frecuentes. (n.d.). Amazon Web Services, Inc. <https://aws.amazon.com/es/lambda/faqs/>
- [20] Overview | Postman Learning Center. (2023, September 6). Postman Learning Center. <https://learning.postman.com/docs/introduction/overview/>
- [21] Jest. (n.d.). <https://jestjs.io/docs/getting-started>
- [22] Li, Y., Wang, Z., & Li, J. (2022). Improving the accuracy of GPT-3 queries with context and feedback. <https://arxiv.org/pdf/2203.02155.pdf>
- [23] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. <https://arxiv.org/pdf/2005.14165.pdf>
- [24] Shinn, N., Li, Y., Wang, Z., Li, J., & Zhou, Q. (2023). Reflexion: an autonomous agent with dynamic memory and self-reflection. <https://www.arxiv-vanity.com/papers/2303.11366/>
- [25] March2021. (n.d.). Graph Developer's Guide for Property Graph. Oracle Help Center. <https://docs.oracle.com/en/database/oracle/property-graph/21.1/spgdg/graphson-data-format.html#GUID-8F01B553-22F0-429E-9018-A1C6F3655508>
- [26] CodeKarle: WhatsApp System Design | Slack/FB Messenger System Design. (n.d.). <https://www.codekarle.com/system-design/Whatsapp-system-design.html>
- [27] API Documentation & Design Tools for Teams | Swagger. (n.d.). <https://swagger.io/>
- [28] OpenAPI Specification - Version 3.0.3 | Swagger. (n.d.). <https://swagger.io/specification/v3/>
- [29] Aibard. (2023, August 28). Google Bard API Documentation 2023: Developers Resources Guide. AIBard. <https://aibard.online/bard-api-documentation/>

12 Anexo

I. Cuestiones utilizadas en las pruebas de aceptación para ingenieros Junior

E-commerce Platform

You are tasked with designing the architecture for an e-commerce platform. This platform should support user registration, product catalog management, shopping cart functionality, order processing, and payment processing. Please provide a high-level software architecture diagram that illustrates the key components and their interactions. Highlight how you would ensure data consistency and security.

Social Media Analytics

Imagine you are building a social media analytics tool that collects data from various social media platforms, processes it, and provides insights to users. Design the software architecture for this tool, considering data collection, storage, analysis, and user interaction. Provide a detailed architecture diagram and explain how you would handle scalability and real-time data updates.

Internet of Things (IoT) Home Automation

You are developing a home automation system that connects various IoT devices (e.g., smart lights, thermostats, cameras) to a central control hub accessible through a mobile app. Create a software architecture diagram that demonstrates how these devices communicate with the hub, the app's user interface, and how you would ensure responsiveness and security.

Content Delivery Network (CDN)

Design the architecture for a Content Delivery Network (CDN) service that efficiently serves static and dynamic content to users worldwide. Your solution should include edge servers, content caching, load balancing, and failover mechanisms. Provide a comprehensive architecture diagram and describe how you would optimize content delivery.

Online Collaborative Document Editing

You are building an online collaborative document editing platform similar to Google Docs. Design the software architecture, considering real-time collaboration, version control, user authentication, and data synchronization. Present a clear architecture diagram and discuss how you would handle concurrent edits while maintaining data integrity.



II. Ejemplo de *prompt* generado para la evaluación de ingenieros Junior

```
You are a top-level Software Architect at one of the best companies of the world, you just conducted an interview and the question that you asked was the following:

    You are tasked with designing the architecture for an e-commerce platform.
    This platform should support user registration, product catalog management, shopping cart functionality, order processing, and payment processing.
    Please provide a high-level software architecture diagram that illustrates the key components and their interactions.
    Highlight how you would ensure data consistency and security.

Now, you will evaluate the software architecture design provided by the candidate here:"https://drive.google.com/file/d/1scIYW3XzXK2lkCWkpokAjOf8xOiaqtPi/view?usp=sharing".
Each feature should be assessed with this rating scale {"1":"Lowest","2":"Below Average","3":"Average","4":"Above Average","5":"Highest"}, based on the following criteria:

    - Component Identification:Rate the candidate's ability to identify and include all necessary components in the architecture diagram. Consider whether the key components of the platform are correctly identified and depicted.
    - Interactions:Rate the clarity and logical coherence of interactions between components in the architecture diagram. Consider whether the candidate has effectively illustrated how different parts of the system communicate.
    - Data Consistency:Rate the candidate's approach to addressing data consistency and security concerns in the design. Consider whether the candidate has described mechanisms for ensuring data integrity and security.
    - User Authentication:Rate the candidate's approach to user authentication and its inclusion in the architecture. Assess whether the candidate's design ensures secure user authentication.'
    - Clarity of Representation:Rate the clarity and structure of the candidate's representation of the software architecture. Assess whether the candidate effectively communicates their design choices and rationale.

Let's work this out step-by-step to be sure we have the right answer, think carefully about the assignment and the answers you will provide. Please provide detailed feedback for each feature, explaining the strengths and areas for improvement in the candidate's software
```

architecture design. Use the rating scale and criteria provided to guide your evaluation.

Please, before providing the answers and the feedback, think carefully if the answers that you are just about to provide are the best ones based on the description, have a good quality and will give a clear sense of the candidate's strengths and weaknesses.

Your response should be in a JSON format with the following structure:

```
{
  result:[{
    "feature": "...",
    "explanation": "...",
    "mark": ...
  }]
}
```

III. Cuestiones utilizadas para la evaluación de un ingeniero de software de nivel intermedio

1. Design the architecture for a social media platform where users can create profiles, post text and multimedia content, follow other users, and receive real-time notifications. Illustrate the key components and how they interact to support millions of users and high user engagement.
2. You are building a ride-sharing application similar to Uber. Create a system diagram that outlines the components and their interactions for matching drivers and riders, tracking real-time locations, handling payments, and ensuring data security.
3. Imagine you are developing a content streaming service like Netflix. Draw a system diagram that depicts how the application serves video content to users, ensures smooth streaming across devices, handles user preferences, and manages content recommendations.
4. You are tasked with building a cloud-based document collaboration platform like Google Docs. Create a system diagram illustrating how users can collaborate on documents in real-time, with features like version control, concurrent editing, and offline access.
5. You are tasked with designing the architecture for a live streaming platform like Twitch. This platform should support the following key components and their interactions:
 - User Registration and Authentication
 - Live Streaming
 - Chat and Interaction Features
 - Video Storage and Playback
 - Recommendation System

Please provide a high-level software architecture diagram that illustrates these key components and their interactions. Highlight how you would ensure scalability, performance, and high availability in your design.

IV. Prompt generado para evaluar a un candidato de nivel intermedio

You are a top-level Software Architect at one of the best companies of the world, you just conducted an interview and the question that you asked was the following:

You are tasked with designing the architecture for a live streaming platform like Twitch. This platform should support the following key components and their interactions:

- User Registration and Authentication
- Live Streaming
- Chat and Interaction Features
- Video Storage and Playback
- Recommendation System

Please provide a high-level software architecture diagram that illustrates these key components and their interactions.

Highlight how you would ensure scalability, performance, and high availability in your design.

Now, you will evaluate the software architecture design provided by the candidate

here: "<https://drive.google.com/file/d/1scIYW3XzXK2lkWCkopkA0jf8xOiaqtPi/view?usp=sharing>".

Each feature should be assessed with this rating scale

{"1": "Lowest", "2": "Below Average", "3": "Average", "4": "Above Average", "5": "Highest"}, based on the following criteria:

- Component Identification: Rate the candidate's ability to identify and include all necessary components in the architecture diagram. Consider whether the key components of the platform are correctly identified and depicted.

- Interactions: Rate the clarity and logical coherence of interactions between components in the architecture diagram. Consider whether the candidate has effectively illustrated how different parts of the system communicate.

- Data Consistency: Rate the candidate's approach to addressing data consistency and security concerns in the design. Consider whether the candidate has described mechanisms for ensuring data integrity and security.

- Scalability: Rate the candidate's consideration of scalability in the architecture, especially for peak requests volume. Assess whether

the candidate has included components or strategies to handle increased load.

- Fault Tolerance:Rate the candidate's inclusion of fault tolerance mechanisms to ensure system availability. Consider whether the candidate has accounted for potential failures and recovery strategies.

- Performance Optimization:Rate the candidate's strategies for optimizing performance, such as reducing latency in processing requests. Evaluate whether the candidate has addressed performance bottlenecks.

- User Authentication:Rate the candidate's approach to user authentication and its inclusion in the architecture. Assess whether the candidate's design ensures secure user authentication.'

- Clarity of Representation:Rate the clarity and structure of the candidate's representation of the software architecture. Assess whether the candidate effectively communicates their design choices and rationale.

Let's work this out step-by-step to be sure we have the right answer, think carefully about the assignment and the answers you will provide. Please provide detailed feedback for each feature, explaining the strengths and areas for improvement in the candidate's software architecture design. Use the rating scale and criteria provided to guide your evaluation.

Please, before providing the answers and the feedback, think carefully if the answers that you are just about to provide are the best ones based on the description and make sense

Your response should be in a JSON format with the following structure:

```
{
  result:[{
    "feature": "...",
    "explanation": "...",
    "mark": ...
  }]
}
```

V. Consulta generada para evaluación de un sistema complejo

You are a top-level Software Architect at one of the best companies of the world, you just conducted an interview and the question that you asked was the following:

You are tasked with designing the architecture for a chat app like Whatsapp.

This platform should mee the following functional requirements:

- Should support one-on-one chats
- Should support group chats

- Should have image, video and file-sharing capabilities
- Should indicate read/receipt of messages
- Show indicate last seen time of users
- Implement end-to-end encryption.
- Customizable privacy settings.

There are also non-functional requirements:

- Should have very very low latency
- Should be always available
- Users should not experience any lags
- Should be highly scalable

Please provide a high-level software architecture diagram that illustrates how the design on this app should be.

Now, you will evaluate the software architecture design provided by the candidate here: "<https://www.codekarle.com/images/WhatsApp.png>".

Each feature should be assessed with this rating scale

{ "1": "Lowest", "2": "Below Average", "3": "Average", "4": "Above Average", "5": "Highest" }, based on the following criteria:

- Component Identification: Rate the candidate's ability to identify and include all necessary components in the architecture diagram. Consider whether the key components of the platform are correctly identified and depicted.

- Interactions: Rate the clarity and logical coherence of interactions between components in the architecture diagram. Consider whether the candidate has effectively illustrated how different parts of the system communicate.

- Data Consistency: Rate the candidate's approach to addressing data consistency and security concerns in the design. Consider whether the candidate has described mechanisms for ensuring data integrity and security.

- Scalability: Rate the candidate's consideration of scalability in the architecture, especially for peak requests volume. Assess whether the candidate has included components or strategies to handle increased load.

- Fault Tolerance: Rate the candidate's inclusion of fault tolerance mechanisms to ensure system availability. Consider whether the candidate has accounted for potential failures and recovery strategies.

- Performance Optimization: Rate the candidate's strategies for optimizing performance, such as reducing latency in processing requests. Evaluate whether the candidate has addressed performance bottlenecks.

- User Authentication: Rate the candidate's approach to user authentication and its inclusion in the architecture. Assess whether the candidate's design ensures secure user authentication.'

- Clarity of Representation: Rate the clarity and structure of

the candidate's representation of the software architecture. Assess whether the candidate effectively communicates their design choices and rationale.

- Availability and Redundancy:Rate the candidate's approach to ensuring high availability, including strategies for service redundancy, global load balancing, and minimizing single points of failure.

- Authorization and Access Control:Rate the candidate's representation of authorization and access control mechanisms, considering role-based access control (RBAC), attribute-based access control (ABAC), and least privilege principles.

- Distributed System Partitions:Rate the candidate's ability to design for and manage distributed system partitions, handling scenarios involving network partitions, data partitioning, and eventual consistency.

- Advanced Architectural Patterns:Rate the candidate's incorporation of advanced architectural patterns such as event sourcing, Command Query Responsibility Segregation, micro-frontends, etc.

Let's work this out step-by-step to be sure we have the right answer, think carefully about the assignment and the answers you will provide. Please provide detailed feedback for each feature, explaining the strengths and areas for improvement in the candidate's software architecture design. Use the rating scale and criteria provided to guide your evaluation.

Please, before providing the answers and the feedback, think carefully if the answers that you are just about to provide are the best ones based on the description and make sense

Your response should be in a JSON format with the following structure:

```
{
  result:[{
    "feature": "...",
    "explanation": "...",
    "mark": ...
  }]
}
```



VI. Código de especificación de la API

```
openapi: 3.0.3
info:
  title: AIChitect
  description: |-
    This is an application which receives a software architecture
    diagram with some evaluation parameters and returns a response
    containing a mark for each evaluation parameter.

  termsOfService: http://swagger.io/terms/
  contact:
    email: miguelangelmagana27@gmail.com
  license:
    name: MIT
    url: https://opensource.org/licenses/mit/
  version: 1.0.0
tags:
  - name: AIChitect
  - name: Evaluation
  description: Systems design diagram's evaluation
paths:
  /evaluate/diagram:
    post:
      tags:
        - Evaluation
      summary: Evaluate a system design diagram
      operationId: evaluateDiagram
      requestBody:
        description: Evaluate the provided diagram depending on the
        candidate's expertise
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Request'
            required: true
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Response'
        '400':
          description: Invalid parameters supplied
        '404':
          description: Diagram not found
        '405':
```

```

        description: Invalid diagram
    security:
      - api_key: []
components:
  schemas:
    Request:
      type: object
      properties:
        questionDescription:
          type: string
          description: The question description
          example: You are tasked with designing the architecture for an
e-commerce platform [...]
        engineeringLevel:
          type: integer
          description: The candidate's engineering level, 0 stands for
Junior, 1 stands for intermediate and 2 stands for Senior.
          example: 0
          enum:
            - 0
            - 1
            - 2
        candidateSolution:
          type: string
          description: The candidate's solution url publicly accessible
and in an xml format.
          format: url
          example: http://www.example-solution.com
        additionalEvaluationParameters:
          type: array
          optional: true
          description: Additional evaluation parameters provided by the
items:
          $ref: '#/components/schemas/AdditionLEvaluationParameter'
    AdditionLEvaluationParameter:
      type: object
      properties:
        parameterName:
          type: string
        evaluationMethod:
          type: string
    Response:
      type: array
      items:
        $ref: '#/components/schemas/EvaluationParameterResult'
    EvaluationParameterResult:
      type: object
      properties:
        feature:

```

```
    type: string
    description: Feature or parameter evaluated
  mark:
    type: integer
    description:
      enum:
        - 0
        - 1
        - 2
        - 3
        - 4
        - 5
    explanation:
      type: string
      description: The system's explanation for the mark provided

  requestBodies:
    Evaluation:
      description: Diagram that needs to be evaluated with its question
        description and candidate level.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Request'
  securitySchemes:
    api_key:
      type: apiKey
      name: x-api-key
      in: header
```

VII. Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				✓
ODS 2. Hambre cero.				✓
ODS 3. Salud y bienestar.				✓
ODS 4. Educación de calidad.				✓
ODS 5. Igualdad de género.	✓			
ODS 6. Agua limpia y saneamiento.				✓
ODS 7. Energía asequible y no contaminante.				✓
ODS 8. Trabajo decente y crecimiento económico.			✓	
ODS 9. Industria, innovación e infraestructuras.		✓		
ODS 10. Reducción de las desigualdades.	✓			
ODS 11. Ciudades y comunidades sostenibles.				✓
ODS 12. Producción y consumo responsables.				✓
ODS 13. Acción por el clima.				✓
ODS 14. Vida submarina.				✓
ODS 15. Vida de ecosistemas terrestres.				✓
ODS 16. Paz, justicia e instituciones sólidas.			✓	
ODS 17. Alianzas para lograr objetivos.				✓

El trabajo se puede relacionar con varios de los ODS, principalmente con aquellos relacionados con la igualdad de condiciones. Al implementar una solución automatizada para evaluar las soluciones de arquitectura de software propuestas por candidatos, se espera eliminar y reducir la subjetividad en esta fase del proceso de selección.

La herramienta automatizada no recibe datos privados de quien ha proporcionado la solución, únicamente recibe el nivel de experiencia de la persona que lo ha resuelto para ser capaz de evaluar las características correctas, esto contribuye en gran medida a la **reducción de las desigualdades** y a la mejorar la **igualdad de género** a la hora de obtener un puesto de trabajo,



ya que todos los candidatos se evalúan bajo el mismo criterio sin tener en cuenta datos como el sexo, la raza, la edad o la localización.

Adicionalmente, dado que es un proyecto de investigación utilizando tecnologías innovadoras, como son los grandes modelos de lenguaje, relacionados con la inteligencia artificial se considera que contribuye en menor grado al objetivo relacionado con la **industria, innovación e infraestructura**. Indirectamente, desarrollar tecnologías y validar ideas utilizando últimas tecnologías abre las puertas a que se realicen otros desarrollos relacionados, creando un conjunto de herramientas o tecnologías que pueden relacionarse con otros ODS.

Otro de los ODS relacionado en menor medida con el proyecto, es el objetivo de desarrollo sostenible denominado **trabajo decente y crecimiento económico**. Se considera que una compañía que considere valorar la igualdad de condiciones en sus procesos de selección tiene mayores probabilidades de proporcionar unas condiciones laborales y salariales de mayor calidad que una empresa que no considera que haya desigualdades en los procesos de selección.

Como último objetivo de desarrollo sostenible relacionado con el trabajo se considera el denominado **ODS 16. Paz, justicia e instituciones sólidas**. La justicia es un término difícil de definir, no obstante, existen las siguientes definiciones:

- Principio moral que lleva a dar a cada uno lo que le corresponde o pertenece.
- Derecho, razón, **equidad**.

Como se puede observar, la **justicia** por definición implica **equidad** y la equidad se define como la cualidad que consiste en dar a **cada uno lo que se merece en función de sus méritos** o condiciones. Esta definición concuerda con los objetivos del proyecto relacionados con eliminar el componente humano en la evaluación del diagrama de arquitectura de software, por lo tanto, la solución es evaluada de manera equitativa para cada candidato.

Cabe mencionar que el proyecto no contribuya a los restantes objetivos de desarrollo sostenible no implica que se consideren de menor importancia. Principalmente, debido a la naturaleza del proyecto y los objetivos planteados al inicio de este, no es posible relacionarlo directamente con objetivos de carácter medioambiental, social o humanitario.

Para concluir, se espera que a medida que las tecnologías avancen, el ser humano sea capaz de utilizarlas para eliminar las diferencias sociales, mejorar el estado del planeta y restaurar los ecosistemas que están próximos a su desaparición, defender la igualdad, la justicia y propiciar una sociedad que defienda el valor de cada ser vivo, mejorando la calidad de vida de todos y cada uno de los individuos que formamos parte de esta sociedad.