Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

ARTÍCULO
Original

Framework para la documentación automática de procesos de negocio mediante un modelo transaccional en JavaScript

A JavaScript Framework for the Automatic Documentation of Business Processes Using a Transactional Model

https://doi.org/10.61325/pj992973

Erick Medel Galindo (autor corresponsal)
me440144@uaeh.edu.mx
ORCID: 0009-0009-8008-6310
Universidad Autónoma del Estado de Hidalgo

Gabriel Sánchez Bautista
gabriel_sanchez@uaeh.edu.mx
ORCID: 0000-0002-9955-8711
Universidad Autónoma del Estado de Hidalgo

Recepción	Aceptación	Publicación
19 de agosto de 2026	25 de septiembre de 2026	30 de septiembre de 2026

Resumen

La documentación respecto a la creación de sistemas de software puede quedar desactualizada frente a la implementación, dificultando la comprensión y el mantenimiento de los procesos de negocio. En esta investigación se desarrolló un *framework* para JavaScript con base en un modelo transaccional utilizado como fuente para elaborar documentación estructurada en HTML. El objetivo fue diseñar, implementar y evaluar su capacidad para generar documentación automática utilizando transacciones y rutinas. La investigación aplicada estuvo orientada al desarrollo tecnológico, con alcance descriptivo-evaluativo y empleó un enfoque cuantitativo, con diseño no experimental y estudio de caso único. Se evaluaron 34 escenarios de aceptación, integrados por 236 pasos e implementados mediante Cucumber, correspondientes al backend de una plataforma financiera. Los datos se recolectaron a través de pruebas de aceptación y comparación estructurada entre las definiciones del código y el HTML generado. Los 34 escenarios y 236 pasos se ejecutaron sin fallos en 4.75 segundos. Asimismo, se comprobó la correspondencia de 12 transacciones, 23 usos de rutinas, sus secuencias e indicadores de reverso y una rutina reutilizada, sin discrepancias. Se concluyó que el *framework* fue viable para el caso estudiado y permitió vincular de forma directa y reproducible la documentación con la implementación.

Palabras clave: Documentación Automática; Proceso de Negocio; Modelo Transaccional; Patrón SAGA; JavaScript; Rutina; Mantenimiento de Software; Prueba de Aceptación

Abstract

Documentation related to software system development may become outdated with respect to the actual implementation, making it more difficult to understand and maintain business processes. In this study, a JavaScript framework was developed based on a transactional model used as the source for generating structured HTML documentation. The objective was to design, implement, and evaluate its ability to automatically generate documentation from transactions and routines. This applied research was oriented toward technological development, had a descriptive-evaluative scope, and employed a quantitative approach with a non-experimental design and a single-case study. A total of 34 acceptance scenarios comprising 236 steps were evaluated using Cucumber; these scenarios corresponded to the backend of a financial platform. Data were collected through acceptance testing and a structured comparison between code definitions and the generated HTML. All 34 scenarios and 236 steps were executed successfully in 4.75 seconds. In addition, the correspondence of 12 transactions, 23 routine uses, their sequences and reversal indicators, and one reused routine was verified, with no discrepancies identified. It was concluded that the framework was viable for the case studied and enabled a direct and reproducible linkage between the documentation and the implementation.

Keywords: Automatic Documentation; Business Process; Transactional Model; SAGA Pattern; JavaScript; Routine; Software Maintenance; Acceptance Test

Framework para la documentación automática de procesos de negocio mediante un modelo transaccional en JavaScript
Erick Medel Galindo ORCID: 0009-0009-8008-6310
Gabriel Sánchez Bautista ORCID:0000-0002-9955-8711
https://doi.org/10.61325/pj992973

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Introducción

En los sistemas de software de larga duración, la evolución continua de los requerimientos, el crecimiento del código fuente y los cambios en los equipos de desarrollo pueden dificultar la conservación del conocimiento sobre el comportamiento y la estructura de las aplicaciones. Esta situación cobra especial relevancia cuando la documentación no se mantiene sincronizada con el código, debido a que puede dificultar la comprensión de los componentes, sus dependencias y los procesos que conforman el sistema (Rai et al., 2022; Hovhannisyan et al., 2024).

Entre las principales consecuencias asociadas a estas dificultades se encuentran un mayor esfuerzo de mantenimiento porque los desarrolladores requieren más tiempo para comprender el código, identificar dependencias y evaluar los posibles efectos de los cambios (Rai et al., 2022); mayor dificultad en la incorporación de nuevos desarrolladores, debido a que la documentación constituye uno de los principales recursos utilizados durante el proceso de incorporación técnica, por lo que su desactualización puede dificultar la comprensión inicial del sistema (Padoan et al., 2024); y, mayor dificultad para identificar dependencias entre componentes, dado que la divergencia entre el código y su documentación puede limitar la comprensión de las relaciones existentes entre los elementos del sistema y de los efectos asociados a sus modificaciones (Hovhannisyan et al., 2024).

Aunque la documentación constituye la herramienta base para mitigar este problema, en la práctica puede llegar a mantenerse de forma manual y, al no estar acoplada al código que describe, puede quedar desactualizada a medida que evoluciona el código. La metodología incorrecta en el mantenimiento de la documentación constituye un problema crítico, especialmente dado el crecimiento continuo de sistemas robustos que requieren esfuerzos considerables por parte de los desarrolladores (Rai et al., 2022). Por ejemplo, en un corpus de 3000 repositorios de GitHub se detectaron referencias obsoletas a elementos de código de forma sistemática (Tan et al., 2024). También se ha encontrado que la falta de sincronía entre los cambios del código y las actualizaciones de su documentación puede dificultar la comprensión y el mantenimiento del software (Hovhannisyan et al., 2024). En consecuencia, la documentación desactualizada en lugar de cumplir una función informativa puede inducir a interpretaciones erradas sobre el comportamiento del sistema. Tal como se observa en un estudio que mediante encuestas y entrevistas a 41 profesionales de la industria del software, identificó que la documentación estática es el recurso más utilizado durante el proceso de incorporación técnica, pero que su constante desactualización genera desventajas a largo plazo. Además, los propios profesionales reconocieron no contar con un responsable claro de mantenerla vigente, lo que agrava el impacto del problema en equipos que heredan sistemas en producción (Padoan et al., 2024).

Frente a este vacío, la industria ha adoptado especificaciones que funcionan simultáneamente como contrato ejecutable y como fuente primaria de documentación, siendo OpenAPI el precedente más consolidado (Karlsson et al., 2020); sin embargo, dicho patrón se ha limitado principalmente al dominio de las APIs REST y no se ha extendido de forma sistemática a la composición de procesos de negocio completos. En el terreno del modelado de estos procesos, se ha propuesto MicroArc, un método que emplea notaciones como la cadena de procesos dirigida por eventos (EPC) para identificar microservicios candidatos en fases tempranas del desarrollo (Yıldız & Demirörs, 2025); también se ha desarrollado un modelo de decisión que mapea 72 características de 23 lenguajes de modelado de procesos de negocio, concluyendo que ninguno de ellos resuelve de forma integrada la consistencia entre el modelo elaborado y su implementación (Farshidi et al., 2024). En ambos casos, dichos modelos se elaboran manualmente y no permanecen acoplados a la implementación, por lo que quedan expuestos a la misma desactualización que afecta a la documentación tradicional. Desde una perspectiva distinta, un método de generación de documentación basado en diseño modular, que sigue un paradigma de "configuración-plantilla-renderizado", ha demostrado que la documentación estructurada puede generarse de forma eficiente (Qiao et al., 2025).

Otra alternativa a la documentación manual es el uso de grandes modelos de lenguaje (LLM), para los cuales se han reportado mejoras en la generación automática de documentación (Khan & Uddin, 2022; Dvivedi et al., 2023). No obstante, estos enfoques presentan limitaciones relacionadas con la generación de información no sustentada en la implementación y con la falta de coherencia a nivel de sistema completo, especialmente cuando los modelos deben documentar relaciones complejas entre componentes (Fan et al., 2023; Gao et al., 2025). En esta misma línea, pruebas con GPT-4.1-mini ponen de manifiesto la volatilidad que pueden presentar los enfoques basados exclusivamente en LLM, lo que evidencia la necesidad de mecanismos adicionales de validación y consistencia (Kiecker et al., 2026).

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista ORCID:0000-0002-9955-8711
Erick Medel Galindo ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Frente a estas limitaciones, surge DocAgent, un sistema multiagente que intenta superarlas mediante procesamiento topológico del código para construir contexto incremental. El sistema incluye agentes especializados (Reader, Searcher, Writer, Verifier y Orchestrator) que colaboran en la generación de documentación y presenta mejoras significativas respecto a los métodos de referencia, aunque requiere una coordinación compleja entre los distintos agentes y múltiples iteraciones (Yang et al., 2025). Sin embargo, persiste la necesidad de mecanismos que permitan utilizar una misma definición estructurada para ejecutar procesos de negocio y generar su documentación, manteniendo la correspondencia entre ambos componentes.

La justificación de esta investigación radica en su aporte teórico y práctico: en el plano teórico, extiende el patrón consolidado por especificaciones como OpenAPI más allá del dominio de las APIs REST, aplicándolo a la composición de procesos de negocio completos; en el plano práctico, ofrece a los equipos de desarrollo un mecanismo que puede contribuir a reducir el mantenimiento manual independiente de la documentación y ofrece una alternativa directa que no depende de la generación documental mediante LLM.

En este sentido surge la pregunta, ¿en qué medida un *framework* para JavaScript basado en un modelo transaccional permite definir, ejecutar y documentar automáticamente procesos de negocio, manteniendo la correspondencia entre la documentación y la implementación?

Por consiguiente, el objetivo general de la investigación es diseñar, implementar y evaluar un *framework* para JavaScript basado en un modelo transaccional compuesto por rutinas, capaz de generar automáticamente documentación estructurada de los procesos de negocio. Como objetivos específicos, se plantean: (1) diseñar un modelo transaccional compuesto por rutinas que permita representar procesos de negocio mediante composiciones explícitas y un mecanismo de reverso inspirado en el patrón SAGA; (2) implementar el *framework* y validar su capacidad de ejecución mediante pruebas de aceptación en notación Gherkin sobre un caso de estudio funcional; (3) evaluar la fidelidad de la documentación HTML generada automáticamente respecto a la composición de las transacciones, las rutinas y su grado de reutilización; y (4) identificar las limitaciones observadas durante la aplicación del enfoque propuesto.

Método

Enfoque Metodológico

La investigación se desarrolla bajo un enfoque cuantitativo. El estudio se centra en el diseño, la implementación y la evaluación de un *framework* para JavaScript que permite definir procesos de negocio mediante un modelo transaccional. La validación funcional del *framework* se realiza mediante pruebas automatizadas de aceptación, mientras que la documentación HTML generada se examina mediante inspección directa, verificando su correspondencia con las definiciones transaccionales en cuanto al listado y orden de las transacciones y rutinas, la disponibilidad de operaciones de reverso y el grado de reutilización de las rutinas.

Diseño de Estudio

El estudio corresponde a un diseño no experimental, desarrollado mediante la estrategia metodológica de estudio de caso único. Es no experimental debido a que el *framework* se evalúa en el contexto funcional definido para el caso de estudio, sin manipulación de variables externas. Se considera un estudio de caso aplicado debido a que analiza específicamente la implementación del *framework* en el *backend* de una plataforma de gestión financiera personal.

Tipo de Estudio

La investigación es de tipo aplicado y de alcance descriptivo-evaluativo. Es aplicada porque busca resolver una problemática concreta relacionada con la desactualización de la documentación de software mediante el diseño de un *framework* funcional; y es descriptivo-evaluativa porque describe la composición y ejecución de los procesos de negocio modelados y evalúa la correspondencia de la documentación generada automáticamente con las definiciones implementadas.

Instrumentos

Los instrumentos y recursos utilizados fueron los siguientes:

- Conjunto de 34 escenarios de aceptación, redactados en notación Gherkin mediante Feature, Background, Scenario, Given, When y Then, utilizados para evaluar los comportamientos

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista ORCID:0000-0002-9955-8711
Erick Medel Galindo ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

funcionales definidos.
- Cucumber, empleado como herramienta para la ejecución automatizada de los escenarios de aceptación.
- Módulo de generación de documentación HTML, incorporado al framework y utilizado para producir la representación estructurada de las transacciones y rutinas.
- Matriz de verificación de la fidelidad de la documentación HTML, utilizada para comparar sistemáticamente los elementos definidos en el código con los mostrados en la documentación generada.

Tabla 1
Integración descriptiva de los resultados cuantitativos y cualitativos sobre las prácticas didácticas y la diferenciación pedagógica

Elemento evaluado	Fuente de referencia	Verificación realizada en el HTML	Criterio de correspondencia
Transacciones	Definiciones transaccionales del código	Comprobación del nombre y presencia de cada transacción	Coincidencia total de nombres y cantidad
Rutinas	Rutinas declaradas en cada transacción	Comprobación de las rutinas mostradas para cada proceso	Presencia de todas las rutinas definidas
Orden de ejecución	Secuencia del arreglo de ejecución	Comparación con el orden presentado en el HTML	Conservación exacta de la secuencia
Operaciones de reverso	Operación de reverso declarada por rutina	Verificación del indicador de disponibilidad de reverso	Coincidencia de la presencia o ausencia del reverso
Reutilización	Frecuencia de aparición de cada rutina en las transacciones	Comparación con el conteo de reutilización mostrado	Igualdad entre ambas frecuencias
Omisiones o elementos adicionales	Conjunto total de elementos definidos	Identificación de componentes faltantes o no declarados	Ausencia de omisiones y elementos adicionales

En la Tabla 1 se presentan los criterios utilizados para verificar la correspondencia entre las definiciones transaccionales del código fuente y la documentación HTML generada. La matriz contempla la presencia y cantidad de transacciones y rutinas, el orden de ejecución, la disponibilidad de operaciones de reverso, la reutilización y la existencia de posibles omisiones o elementos adicionales.

Procedimiento

El procedimiento se desarrolló en cinco etapas. En primer lugar, se definieron los requerimientos funcionales y no funcionales del caso de estudio tomando como referencia el trabajo de Wu (2024). En segundo lugar, los procesos de negocio del *backend* se modelaron mediante transacciones y rutinas, incorporando el mecanismo de reverso inspirado en el patrón SAGA. En tercer lugar, se elaboraron los 34 escenarios de aceptación en notación Gherkin y se ejecutaron mediante Cucumber para verificar el comportamiento funcional del *framework*. En cuarto lugar, se levantó el servidor de documentación y se generó automáticamente el HTML correspondiente a las transacciones implementadas. Finalmente, el contenido generado se contrastó con las definiciones del código fuente mediante la matriz de verificación establecida en la Tabla 1.

Modelo Transaccional y Mecanismo de Reverso

En el contexto del *framework* desarrollado, el término transacción se emplea con un alcance distinto al asociado estrictamente a las transacciones de bases de datos relacionales. Una transacción representa un proceso de negocio completo compuesto por una secuencia ordenada de rutinas y cumple una doble función: organiza la ejecución del proceso y constituye la definición estructurada a partir de la cual se genera la documentación. Las rutinas corresponden a unidades de procesamiento que encapsulan operaciones específicas dentro de dicha composición, en concordancia con principios de descomposición funcional revisados por Velepucha y Flores (2023).

Dentro del *framework*, cada transacción representa un proceso de negocio completo y se compone de una secuencia de rutinas, entendidas como transacciones parciales siguiendo el patrón SAGA (Daraghmi et al., 2022; Haapakoski, 2023; Lee et al., 2025), cuyas implementaciones se han orientado

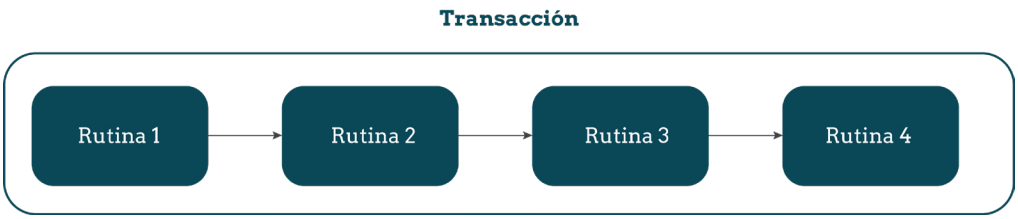
Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

principalmente a garantizar la consistencia de datos, sin extenderse a la generación automática de documentación a partir de la propia definición transaccional.

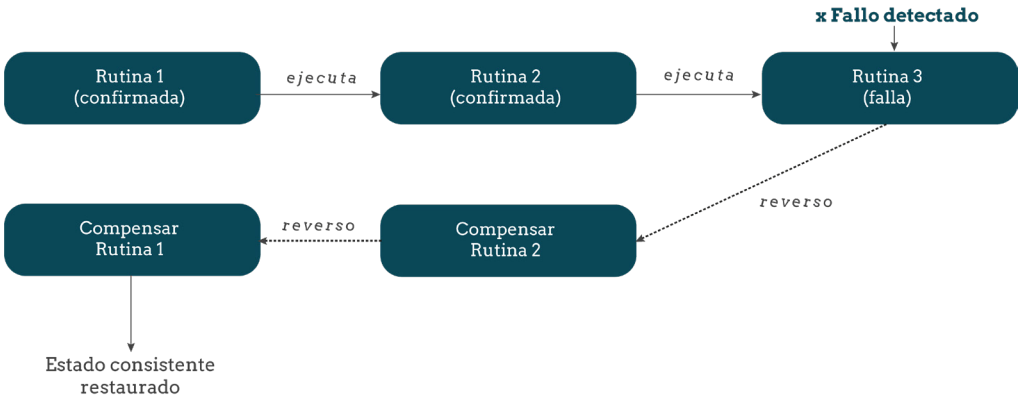
Figura 1
Composición de una transacción a partir de rutinas ejecutadas secuencialmente



En la Figura 1 se muestra la composición de una transacción a partir de una secuencia de rutinas ejecutadas de manera ordenada. Cada rutina representa una unidad de procesamiento dentro del proceso de negocio y, en conjunto, conforman la estructura transaccional utilizada tanto para la ejecución como para la generación de la documentación.

Al confirmarse cada rutina de forma independiente, un fallo en una etapa intermedia de la secuencia no puede deshacerse mediante un *rollback* convencional (reversión de los cambios de una transacción), debido a que las rutinas anteriores ya fueron confirmadas. Por ello, cada rutina puede declarar una operación de reverso que compense semánticamente sus efectos. Ante un fallo, el *framework* ejecuta las compensaciones de las rutinas previamente confirmadas en orden inverso al de ejecución.

Figura 2
Composición de una transacción a partir de rutinas ejecutadas secuencialmente



Nota. Ante un fallo en la Rutina 3, se ejecutan las operaciones compensatorias de las rutinas previamente confirmadas en orden inverso.

En la Figura 2 se muestra el mecanismo de compensación aplicado cuando ocurre un fallo durante la ejecución de una transacción. Las rutinas previamente confirmadas son compensadas en orden inverso respecto de la secuencia original, de modo que las operaciones de reverso se ejecutan desde la rutina más recientemente confirmada hacia las anteriores.

Definición e Implementación de las Transacciones

En código, las transacciones se definen como funciones que retornan dos componentes principales: los metadatos de la transacción, que incluyen su nombre y descripción, y el arreglo execute, que contiene la secuencia ordenada de rutinas que conforman la transacción, junto con sus respectivos metadatos y operaciones de reverso.

Listado 1
Estructura de la transacción "create-account", incluye la rutina "insert-account" y su reverso

```
function buildCreateAccountTx({ accountId, userId, input, result }: CreateAccountParams):
TransactInput {
 return {
```

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

```
  metadata: { name: 'create-account', description: 'Crea una cuenta para un usuario' },
  execute: [
   routine({
    metadata: { name: 'insert-account' },
    execute: async () => {
     result.account = accountRepository.create({
      id: accountId,
      userId,
      name: input.name,
      type: input.type,
      currency: input.currency ?? 'USD',
     });
    },
    rollback: async () => {
     accountRepository.delete(accountId);
    },
   }),
  ],
 };
}
```

En el Listado 1 se muestra la implementación de la transacción create-account, compuesta por la rutina insert-account. La definición incluye los metadatos de la transacción y de la rutina, así como la operación principal y la operación de reverso asociada, lo que permite representar en una misma estructura tanto la ejecución del proceso como el mecanismo de compensación correspondiente.

Esta estructura permite separar la definición de una transacción de su ejecución. Para ejecutar una transacción, el objeto retornado por su función constructora se pasa a una instancia de Transact y se invoca el método .run().

Listado 2

Estructura de la transacción "create-account", incluye la rutina "insert-account" y su reverso

```
class AccountService {
 async update(userId: string, accountId: string, fields: { name?: string; type?: string }) {
  const previous = assertOwned(accountRepository.findById(accountId), userId);
  const result: { account?: AccountRow } = {};

  await new Transact(buildUpdateAccountTx({ accountId, fields, previous, result })).run();

  return result.account!;
 }
}
```

En el Listado 2 se muestra la ejecución de la transacción update-account. El objeto retornado por su función constructora se proporciona a una instancia de Transact y posteriormente se ejecuta mediante el método .run(), lo que permite mantener separadas la definición estructural de la transacción y su ejecución.

Unidad de Análisis

La unidad de análisis corresponde al *backend* de un caso de estudio funcional desarrollado por los autores de esta investigación, no desplegado en producción, tomando como referencia los requerimientos descritos por Wu (2024):

Requerimientos funcionales

- Permitir a los nuevos usuarios registrarse en la plataforma.
- Autenticar a los usuarios mediante JWT (JSON Web Tokens).
- Permitir a los usuarios visualizar y editar su perfil.
- Crear, editar y eliminar cuentas.
- Mostrar el saldo y las transacciones de cada cuenta.
- Registrar ingresos y gastos en diferentes cuentas.
- Categorizar automáticamente las transacciones según su tipo.
- Establecer y seguir objetivos financieros personales.
- Generar informes de transacciones detallados.

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

- Proporcionar datos financieros.

Requerimientos no funcionales

- Implementar autenticación y autorización seguras, y cifrado de datos.
- Asegurar una respuesta rápida del sistema y escalabilidad para soportar múltiples usuarios concurrentes.
- Incorporar mecanismos de recuperación ante fallos mediante operaciones compensatorias.

Los requerimientos no funcionales se utilizaron como referencia para la construcción del caso de estudio; sin embargo, su evaluación específica no formó parte del alcance de esta investigación, centrada en la ejecución de los procesos funcionales y en la correspondencia entre su definición y la documentación generada.

Tabla 2
Escenarios de aceptación implementados con Cucumber

Feature: Autenticación y perfil	
1	Registro exitoso crea usuario, cuenta principal y *token*
2	No se puede registrar dos veces el mismo correo
3	El registro rechaza contraseñas demasiado cortas
4	*Login* con credenciales correctas devuelve *token*
5	*Login* con contraseña incorrecta es rechazado
6	*Login* con correo inexistente es rechazado
7	Acceder al perfil sin *token* es rechazado
8	Acceder al perfil con un *token* corrupto es rechazado
9	Ver y editar el perfil propio
Feature: Cuentas	
10	Crear una cuenta adicional
11	Listar cuentas incluye la cuenta principal creada en el registro
12	Editar una cuenta propia
13	No se puede eliminar una cuenta con transacciones registradas
14	Eliminar una cuenta sin movimientos
15	No se puede ver ni editar la cuenta de otro usuario
Feature: Transacciones (ingresos, gastos, categorización automática y reverso)	
16	Registrar un ingreso actualiza el saldo y lo categoriza automáticamente
17	Registrar un gasto lo categoriza automáticamente según palabras clave
18	Un gasto sin palabras clave reconocidas cae en la categoría por defecto
19	Un gasto que excede el saldo se rechaza y no modifica el saldo (*rollback* automático)
20	Revertir una transacción restaura el saldo mediante un asiento compensatorio
21	No se puede revertir dos veces la misma transacción
22	No se puede revertir un asiento que ya es un reverso
23	Listar transacciones de una cuenta con filtro por tipo
24	No se puede ver el detalle de una transacción de otra cuenta
Feature: Objetivos financieros	
25	Crear un objetivo financiero
26	Aportar a un objetivo retira saldo de la cuenta y avanza el progreso
27	Un aporte que excede el saldo disponible se rechaza sin afectar el objetivo
28	Alcanzar el monto objetivo marca el objetivo como logrado
29	Editar y luego eliminar un objetivo
Feature: Informes financieros	
30	El resumen financiero refleja ingresos, gastos y saldo neto
31	El informe detallado de transacciones incluye nombres de cuenta y categoría
32	El informe detallado se puede filtrar por tipo
33	Categorías disponibles incluyen las globales por defecto
34	Se puede crear una categoría personalizada

En la Tabla 2 se presentan los 34 escenarios de aceptación utilizados para evaluar los requerimientos funcionales definidos para el *backend* del caso de estudio. Los escenarios se agruparon en cinco grupos funcionales: autenticación y perfil, cuentas, transacciones, objetivos financieros e informes financieros.

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Técnica de Recolección de Datos

Se emplearon dos técnicas de recolección de datos. La primera correspondió a la ejecución de pruebas automatizadas de aceptación mediante Cucumber, con el propósito de verificar el comportamiento funcional del *framework* frente a los requerimientos definidos para el caso de estudio. La segunda consistió en la inspección directa de la documentación HTML generada, mediante la comparación de su contenido con las transacciones y rutinas declaradas en el código fuente.

Los escenarios de aceptación se redactaron en notación Gherkin mediante las cláusulas *Given* (estado inicial), *When* (acción) y *Then* (resultado esperado). Esta estructura permite expresar los requerimientos mediante un formato legible y comprobable que puede ejecutarse como prueba automatizada mediante Cucumber (Farooq et al., 2023). El desarrollo guiado por comportamiento (*Behavior-Driven Development* [BDD]) emplea este tipo de escenarios para especificar y verificar el comportamiento esperado del sistema. Su aplicación también se ha extendido a la automatización de requerimientos no funcionales, incluidos aspectos relacionados con la eficiencia del rendimiento (Santos et al., 2024).

Listado 3

Escenario de aceptación en notación Gherkin correspondiente al registro de un ingreso

```
Feature: Transacciones (ingresos, gastos, categorización automática y reverso)

 Background:
  Given un usuario nuevo con nombre "Diego Marin", correo "diego+{{uuid}}@example.com" y
contraseña "clave12345", autenticado

 Scenario: Registrar un ingreso actualiza el saldo y lo categoriza automáticamente
  When hace una petición POST a "/transactions" con cuerpo:
   """
   {"accountId":"{{defaultAccountId}}","type":"income","amount":1000,"description":"pago de salario
mensual"}
   """
  Then la respuesta debe tener código 201
  And el cuerpo debe tener "categoryId" igual a "cat-income-salary"
  When hace una petición GET a "/accounts/detail?id={{defaultAccountId}}"
  Then la respuesta debe tener código 200
  And el cuerpo debe tener "balance" igual a "1000"
```

En el Listado 3 se presenta uno de los escenarios de aceptación implementados mediante Cucumber y redactados en notación Gherkin. El escenario corresponde al registro de un ingreso y verifica tanto la respuesta obtenida tras realizar la solicitud como la asignación automática de la categoría y la actualización posterior del saldo de la cuenta.

Análisis de Datos

- Evaluación funcional: Los resultados de las pruebas de aceptación se analizaron mediante el número de escenarios y pasos ejecutados satisfactoriamente y la identificación de posibles fallos durante la ejecución. Para las operaciones compensatorias, se registró además la secuencia de ejecución de los reversos con el propósito de verificar su orden dentro de cada saga.
- Fidelidad documental: La documentación HTML generada se analizó mediante inspección directa conforme a la matriz de verificación presentada en la Tabla 1. Se compararon las transacciones y rutinas definidas en el código con las mostradas en el HTML, considerando su presencia, orden de ejecución, disponibilidad de operaciones de reverso, reutilización y existencia de omisiones o elementos adicionales.

Consideraciones éticas

La investigación no involucra la participación de sujetos humanos ni el uso de datos personales reales, dado que el caso de estudio se ejecuta sobre datos de prueba generados de forma sintética. Por tal motivo, no se requiere aprobación de un comité de ética. Se declara que la información técnica y los resultados presentados corresponden fielmente a la ejecución del *framework*.

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Resultados

Los resultados obtenidos a partir del caso de estudio permitieron responder a los objetivos planteados.

Capacidad del Modelo Transaccional para Definir y Ejecutar Procesos de Negocio

El *framework* permitió modelar la totalidad de los requerimientos funcionales definidos para el *backend* del caso de estudio de gestión financiera personal mediante composiciones de transacciones y rutinas. La correcta ejecución de los procesos definidos se validó mediante 34 escenarios de aceptación (236 pasos), los cuales se ejecutaron satisfactoriamente en 4.75 segundos, sin fallos. Los escenarios relacionados con el mecanismo de reverso permitieron comprobar el comportamiento de las operaciones compensatorias. Para verificar el orden de ejecución de dichas compensaciones, se creó una tabla en PostgreSQL con los campos que se presentan en la Tabla 3. Los registros correspondientes a una misma saga se agruparon y ordenaron según seq. Posteriormente, se comprobó que, conforme aumentaba seq, el valor de step disminuyera, lo que permitió verificar que las operaciones compensatorias se ejecutaron en orden inverso respecto de la secuencia original.

Tabla 3
Estructura de la tabla saga_rollback para verificación del mecanismo de reverso

Columna	Tipo	Criterio de correspondencia
seq	BIGSERIAL	Número correlativo asignado por el gestor de base de datos al insertar la fila. Refleja el orden real en que se ejecutaron las compensaciones: el *framework* ejecuta los manejadores de compensación de forma secuencial, esperando la finalización de cada uno antes de iniciar el siguiente, por lo que la escritura de una fila concluye antes de que comience la compensación siguiente. Es la columna sobre la que se verifica el orden inverso.
run_id	TEXT	Identificador único generado al iniciar el servidor, utilizado para distinguir los registros correspondientes a diferentes ejecuciones. Como la base de datos conserva el historial entre ejecuciones, este campo permite aislar la traza asociada a una ejecución específica y analizar únicamente las compensaciones registradas en ella.
saga	TEXT	Nombre del proceso de negocio al que pertenece la compensación: create-transaction, reverse-transaction, goal-contribution o user-signup. Identifica la definición de la saga, no una ejecución concreta.
routine	TEXT	Identificador de la rutina cuya operación compensatoria fue ejecutada. Corresponde al nombre definido para dicha rutina dentro de la saga, por ejemplo, apply-balance o insert-transaction.
step	INTEGER	Posición fija de ese paso dentro de la definición de la saga, empezando en cero. No varía entre ejecuciones. Contrastada con seq, es lo que evidencia el orden inverso de compensación: conforme seq aumenta, step disminuye, lo que significa que cada compensación deshizo un paso anterior al ya deshecho.
description	TEXT	Enunciado en lenguaje llano de qué deshizo la compensación, incluyendo los identificadores y montos afectados (p. ej. saldo de la cuenta febd9066… ajustado en –500). Permite identificar la acción realizada por cada operación compensatoria y facilitar el contraste de la traza con el estado registrado en la base de datos de dominio.

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

at	TIMESTAMPTZ	Instante en que se registra la compensación. Se utiliza como información temporal complementaria de la ejecución, mientras que seq constituye el campo principal empleado para verificar el orden.

En la Tabla 3 se describen los campos utilizados para registrar la ejecución de las operaciones compensatorias. El campo seq permite identificar el orden real de ejecución, mientras que step representa la posición de cada rutina dentro de la saga. Asimismo, run_id permite distinguir las ejecuciones registradas, y los campos saga, routine, description y at aportan información sobre el proceso, la rutina compensada, la acción realizada y el momento de ejecución.

Listado 4

Consulta SQL para verificar el orden de ejecución de las operaciones compensatorias

```
WITH ultima AS (
  SELECT run_id
  FROM saga_rollback
  ORDER BY seq DESC
  LIMIT 1
),
comparacion AS (
  SELECT
    saga,
    seq,
    routine,
    step,
    LAG(step) OVER (
      PARTITION BY run_id, saga
      ORDER BY seq
    ) AS step_anterior
  FROM saga_rollback
  WHERE run_id = (SELECT run_id FROM ultima)
)
SELECT
  saga,
  count(*) AS compensaciones,
  string_agg(
    routine || ' (' || step || ')',
    ' -> ' ORDER BY seq
  ) AS orden_ejecutado,
  count(*) FILTER (
    WHERE step_anterior IS NOT NULL
    AND step >= step_anterior
  ) AS violaciones,
  CASE
    WHEN count(*) = 1
      THEN 'NO APLICA: UNA ÚNICA COMPENSACIÓN'
    WHEN count(*) FILTER (
      WHERE step_anterior IS NOT NULL
      AND step >= step_anterior
    ) = 0
      THEN 'ORDEN INVERSO OK'
    ELSE 'VIOLACION LIFO'
  END AS veredicto
FROM comparacion
GROUP BY saga
ORDER BY saga;
```

En el Listado 4 se presenta la consulta SQL utilizada para comprobar el orden de ejecución de las operaciones compensatorias. Para las sagas con dos o más compensaciones, la consulta verifica que los valores de step sean estrictamente decrecientes respecto de seq; se contabiliza como violación

Framework para la documentación automática de procesos de negocio mediante un modelo transaccional en JavaScript
Gabriel Sánchez Bautista ORCID:0000-0002-9955-8711
Erick Medel Galindo ORCID: 0009-0009-8008-6310
https://doi.org/10.61325/pj992973

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

cualquier caso en el que el valor de step sea igual o superior al de la compensación anterior. Cuando una saga registra una única compensación, no es posible evaluar un orden relativo entre múltiples operaciones, por lo que el resultado se clasifica como "NO APLICA: UNA ÚNICA COMPENSACIÓN".

Listado 5
Resultado de la ejecución del *rollback* de cada una de las rutinas


```
[
 {
  "saga": "create-transaction",
  "compensaciones": 2,
  "orden_ejecutado": "apply-balance (3) -> insert-transaction (2)",
  "violaciones": 0,
  "veredicto": "ORDEN INVERSO OK"
 },
 {
  "saga": "goal-contribution",
  "compensaciones": 2,
  "orden_ejecutado": "record-transaction (2) -> withdraw-from-account (1)",
  "violaciones": 0,
  "veredicto": "ORDEN INVERSO OK"
 },
 {
  "saga": "reverse-transaction",
  "compensaciones": 2,
  "orden_ejecutado": "adjust-balance (2) -> create-compensating-entry (1)",
  "violaciones": 0,
  "veredicto": "ORDEN INVERSO OK"
 },
 {
  "saga": "user-signup",
  "compensaciones": 1,
  "orden_ejecutado": "create-user (0)",
  "violaciones": 0,
  "veredicto": "NO APLICA: UNA ÚNICA COMPENSACIÓN"
 }
]
```


En el Listado 5 se presentan los resultados obtenidos para las sagas evaluadas mediante la ejecución de las operaciones compensatorias. En create-transaction, goal-contribution y reverse-transaction se registraron dos compensaciones en cada caso, ejecutadas en orden inverso y sin violaciones. En user-signup se registró una única compensación, por lo que no fue posible evaluar un orden inverso entre múltiples operaciones; en este caso, el resultado únicamente confirmó la ejecución de la operación compensatoria correspondiente.

Tabla 4
Resultados de la ejecución de pruebas de aceptación

Escenarios	Pasos	Tiempo de ejecución	Resultado
34 (34 aprobados)	236 (236 aprobados)	4.75 s	sin fallos registrados

En la Tabla 4 se muestran los resultados de la ejecución de las pruebas de aceptación. Los 34 escenarios y los 236 pasos que integraron la suite se ejecutaron satisfactoriamente en 4.75 segundos, sin registrarse fallos durante la ejecución.

Fidelidad de la Documentación Generada

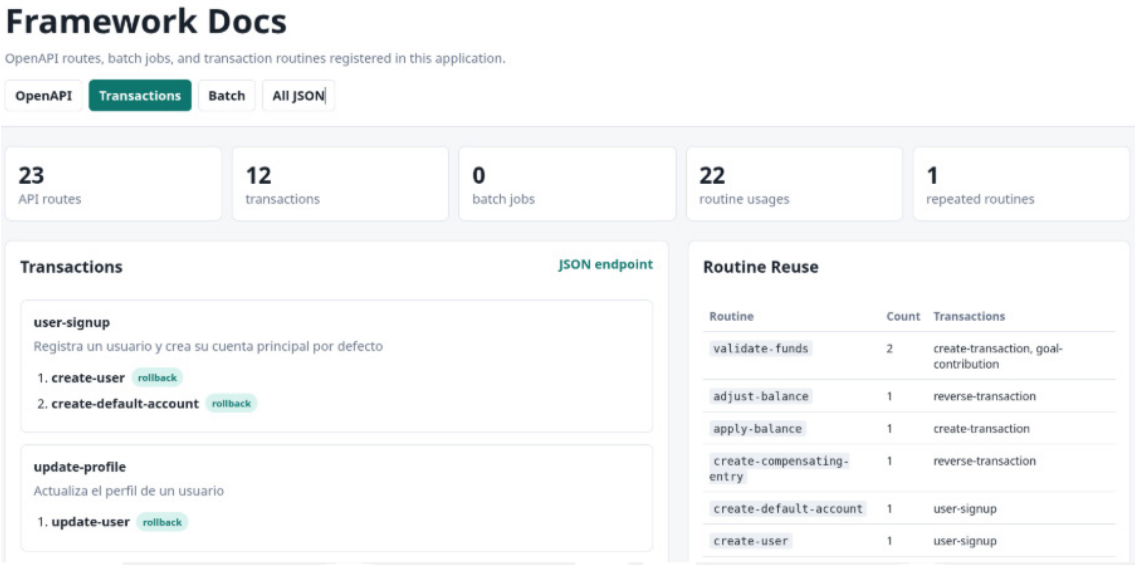
La documentación HTML generada automáticamente al levantar el servidor reflejó de forma directa la estructura definida en las transacciones, sin requerir una actualización manual independiente de dicha estructura documental.

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Figura 3
Vista general de la documentación generada: cada transacción del *backend* se presenta junto con la secuencia ordenada de rutinas que la conforman



En la Figura 3 se observa el *dashboard* generado por el *framework*, en el que se presenta el listado completo de transacciones junto con las rutinas que las componen, el indicador de disponibilidad de una operación de reverso para cada rutina y el grado de reutilización de estas entre distintas transacciones. Este último elemento permite identificar dependencias compartidas entre procesos de negocio, información que, en un esquema de documentación manual, puede perderse o quedar desactualizada.

Tabla 5
Correspondencia entre las definiciones transaccionales del código fuente y la documentación HTML generada

Elemento evaluado	Criterio de verificación	Definido en el código	Mostrado en el HTML	Discrepancias
Transacciones	Coincidencia de nombres y cantidad total	12	12	0
Usos de rutinas	Coincidencia de las rutinas asociadas con cada transacción	23	23	0
Orden de las rutinas	Conservación de la secuencia definida en cada transacción	12 secuencias	12 secuencias coincidentes	0
Operaciones de reverso	Correspondencia entre la operación declarada y el indicador mostrado	17 de 23 usos de rutinas	17 de 23 usos de rutinas	0
Rutinas reutilizadas	Coincidencia de rutinas presentes en más de una transacción	1 (validate-funds)	1 (validate-funds)	0
Omisiones o elementos adicionales	Ausencia de componentes faltantes o no definidos en el código	No aplica	Ninguno identificado	0

Nota: Un uso de rutina corresponde a cada aparición de una rutina dentro de una transacción. La rutina validate-funds se reutilizó en dos transacciones: create-transaction y goal-contribution.

En la Tabla 5 se presenta la correspondencia entre las definiciones transaccionales del código fuente y la documentación HTML generada. Se identificó coincidencia en las 12 transacciones, los 23 usos de rutinas y las 12 secuencias evaluadas. Asimismo, los 17 usos de rutinas con operaciones de reverso y la rutina reutilizada validate-funds coincidieron entre el código y el HTML. No se

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

identificaron omisiones, elementos adicionales ni discrepancias estructurales en los aspectos evaluados.

Discusión

Los resultados mostraron que el *framework* fue capaz de representar los procesos de negocio analizados mediante una estructura basada en transacciones y rutinas, utilizando las mismas definiciones tanto para su ejecución como para la generación automática de la documentación HTML. La correspondencia observada entre ambas representaciones comprendió las 12 transacciones, los 23 usos de rutinas y las 12 secuencias evaluadas, además de los indicadores de reverso y la rutina reutilizada validate-funds, sin identificarse omisiones, elementos adicionales ni discrepancias estructurales en los aspectos considerados. En este sentido, la relevancia del resultado no radica únicamente en la ejecución de los procesos, sino en que la estructura utilizada para ejecutarlos constituye simultáneamente la fuente de su representación documental.

Este resultado adquirió importancia frente al problema de divergencia entre código y documentación señalado previamente en la literatura. Aghajani et al. (2020) identificaron la inconsistencia y la falta de automatización entre los problemas relevantes asociados con la documentación de software. De manera complementaria, Tan et al. (2024), al analizar más de 3.000 proyectos de GitHub, encontraron que el 28.9 % de los proyectos más populares contenía al menos una referencia desactualizada al momento del análisis y que el 82.3 % había presentado documentación desactualizada en algún momento de su historia; además, estas referencias podrían permanecer sin corregirse durante varios años. Estos antecedentes mostraron que la sincronización entre implementación y documentación constituye un problema persistente, por lo que derivar la estructura documental de las mismas definiciones utilizadas durante la ejecución puede disminuir la posibilidad de que ambos componentes evolucionen de manera independiente.

La utilización de una representación estructurada como fuente documental también guarda relación con lo reportado por Ciurumelea et al. (2023). Los autores analizaron prácticas de documentación en más de 18.000 proyectos desarrollados en Python y Java y encontraron un uso extendido de elementos estructurados en los comentarios; además, observaron mejoras en la precisión de una herramienta que sugiere automáticamente palabras durante la redacción de comentarios de código cuando se incorporó información estructural al modelo. Asimismo, señalaron que las estructuras documentales formales facilitan el procesamiento automatizado y la generación de documentación. En el presente estudio, este principio se trasladó desde la estructuración de comentarios y metadatos hacia la composición de procesos de negocio, dado que las transacciones, las rutinas y sus relaciones constituyen la estructura desde la cual se genera la documentación HTML.

Otro resultado relevante correspondió al mecanismo de reverso. Su incorporación, inspirada en el patrón SAGA, permitió representar explícitamente las operaciones compensatorias asociadas a determinadas rutinas y verificar su ejecución en orden inverso cuando una saga contenía múltiples compensaciones. Este planteamiento presentó coherencia con Daraghmi et al. (2022) y Haapakoski (2023), quienes abordaron la descomposición de transacciones distribuidas en unidades que pueden ser compensadas cuando no resulta posible recuperar el estado mediante mecanismos transaccionales convencionales. La información obtenida del *framework* evaluado en la presente investigación, no se limitó al comportamiento interno de ejecución, sino que formó parte de la definición transaccional y pudo reflejarse en la documentación generada, visibilizando qué rutinas disponen de mecanismos de reverso.

En lo referente a la automatización de la documentación, Ghale y Dabbagh (2025), tras evaluar de forma empírica T5 y BART para la generación automática de comentarios de código, manifestaron la necesidad de considerar la precisión, coherencia y relevancia contextual de los contenidos producidos y resaltaron que la documentación manual puede ser costosa en tiempo y presenta el riesgo de quedar incompleta o desactualizada a medida que aumenta la complejidad del software. Del mismo modo, el presente estudio buscó reducir la dependencia de procesos documentales manuales, aunque la estructura de la documentación no se generó de manera probabilística, sino que se obtuvo directamente de las definiciones transaccionales implementadas en el código. Cabe resaltar que esto no excluye el uso complementario de LLM para enriquecer aspectos descriptivos, sobre todo en los que todavía están sujetos a los metadatos proporcionados por el desarrollador.

Framework para la documentación automática de procesos de negocio mediante un modelo transaccional en JavaScript
Erick Medel Galindo
ORCID: 0009-0009-8008-6310
Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
https://doi.org/10.61325/pj992973

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Por otro lado, la relación entre ejecución y documentación fue acorde con la investigación de Fantino et al. (2026) que exploró un enfoque sensible a la ejecución a través de la captura de transformaciones producidas sobre los datos después de ejecutar el código y que utilizó dichas transformaciones como señales estructuradas complementarias para generar comentarios de forma automática, pero sin mejoras uniformes en todos los escenarios, aportando evidencia sobre la viabilidad de utilizar información derivada de la ejecución para producir representaciones documentales en función del contexto. En contraste con el presente estudio, la vinculación se da por medio de un mecanismo diferente, es decir, la documentación estructural no se infiere a partir del comportamiento observado posteriormente, sino que se deriva de manera directa de las mismas definiciones transaccionales empleadas para ejecutar los procesos.

En la praxis, los hallazgos obtenidos sugieren que emplear una definición común para la ejecución y la documentación puede reducir el riesgo de divergencia estructural entre ambos componentes. Al respecto, Hassan et al. (2024) destacaron la importancia de contar con documentación, métricas explícitas y suites de pruebas verificables para favorecer la reproducibilidad y sostenibilidad de las herramientas de software. En este trabajo, el *framework* facilitó la visualización en una misma estructura para entender cómo se compone y funciona cada proceso del negocio, mostrando sus transacciones, el orden de las rutinas, su reutilización y las operaciones de reverso, sin necesidad de mantener esa información de forma independiente en otro documento, pero no se puede asegurar que este reduzca efectivamente el tiempo o el esfuerzo de mantenimiento, puesto que las variables utilizadas no se midieron, por lo que su aporte se limitó a mostrar que la estructura documental evaluada permaneció vinculada con las definiciones implementadas.

Entre las fortalezas del estudio se incluyen la evaluación del *framework* sobre un caso funcional que abarcó los requerimientos definidos para el *backend*, su verificación mediante 34 escenarios de aceptación y la comparación estructurada entre las definiciones implementadas y la documentación HTML generada. No obstante, al evaluarse un único caso de estudio desarrollado por los propios autores, no desplegado en producción y ejecutado con datos sintéticos, los resultados deben interpretarse en función a ello. De acuerdo con Fantino et al. (2026), los resultados de una prueba de concepto en un contexto limitado no son evidencia suficiente de aplicabilidad inmediata a otros lenguajes o ecosistemas, es decir, la generalización del enfoque propuesto tiene que aplicarse en otros dominios, arquitecturas y niveles de complejidad.

Respecto a las limitaciones del estudio, la interfaz HTML generada fue únicamente de lectura y no permitió ejecutar directamente las transacciones desde el *dashboard*, por lo que no pudo utilizarse como herramienta interactiva durante las pruebas. Asimismo, aunque la estructura de la documentación se generó automáticamente, la calidad semántica de los nombres y descripciones continuó dependiendo de los metadatos incorporados por el desarrollador. Por ello, una línea de desarrollo futura consistiría en incorporar mecanismos que permitan validar o mejorar automáticamente estos contenidos descriptivos. En este contexto, la propuesta de Banias et al. (2021) sobre la posibilidad de trasladar casos hacia *frameworks* de pruebas podría explorarse como una alternativa para futuras extensiones.

Conclusiones

Se concluye que el *framework* para JavaScript basado en un modelo transaccional compuesto por rutinas permitió, en el caso estudiado, integrar la definición y ejecución de procesos de negocio con la generación automática de documentación HTML a partir de una misma estructura transaccional.

La evaluación funcional mostró que los 34 escenarios de aceptación, integrados por 236 pasos, se ejecutaron sin fallos. Asimismo, en las sagas con múltiples compensaciones se verificó que las operaciones de reverso se ejecutaron en orden inverso respecto de la secuencia original. En cuanto a la documentación generada, se comprobó la correspondencia de las 12 transacciones, los 23 usos de rutinas y las 12 secuencias evaluadas; también coincidieron los 17 usos de rutinas con operaciones de reverso y la rutina reutilizada validate-funds, sin identificarse omisiones, elementos adicionales ni discrepancias estructurales. Estos resultados permiten establecer la viabilidad del enfoque dentro de las condiciones del caso estudiado.

Para futuros estudios, se recomienda evaluar el *framework* en otros dominios, arquitecturas y niveles de complejidad, así como en entornos más próximos a

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

producción; medir de forma específica su efecto sobre el tiempo, esfuerzo y consistencia del mantenimiento documental; incorporar mecanismos para validar o enriquecer los metadatos descriptivos; ampliar la interfaz hacia funciones interactivas; y comparar experimentalmente el enfoque con otras estrategias de documentación automática.

Referencias

Aghajani, E., Nagy, C., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., & Shepherd, D. C. (2020). Software documentation: The practitioners' perspective. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 590–601). *Association for Computing Machinery.* https://doi.org/10.1145/3377811.3380405

Banias, O., Florea, D., Gyalai, R., & Curiac, D.-I. (2021). Automated specification-based testing of REST APIs. *Sensors,* 21(16), 5375. https://doi.org/10.3390/s21165375

Ciurumelea, A., Alexandru, C. V., Gall, H. C., & Proksch, S. (2023). Completing function documentation comments using structural information. *Empirical Software Engineering,* 28(4), Article 86. https://doi.org/10.1007/s10664-022-10284-6

Daraghmi, E., Zhang, C.-P., & Yuan, S.-M. (2022). Enhancing saga pattern for distributed transactions within a microservices architecture. *Applied Sciences*, 12(12), 6242. https://doi.org/10.3390/app12126242

Dvivedi, S. S., Vijay, V., Pujari, S. L. R., Lodh, S., & Kumar, D. (2023). A comparative analysis of large language models for code documentation generation. *arXiv.* https://doi.org/10.48550/arXiv.2312.10349

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023). Large language models for software engineering: Survey and open problems. *arXiv.* https://doi.org/10.48550/arXiv.2310.03533

Fantino, G., Vetro', A., Torchiano, M., & Cappelluti, F. (2026). Beyond syntax: Enhancing automated documentation with data differences. *Automated Software Engineering,* 33, Article 77. https://doi.org/10.1007/s10515-026-00623-y

Farooq, M. S., Omer, U., Ramzan, A., Rasheed, M. A., & Atal, Z. (2023). Behavior driven development: A systematic literature review. *IEEE Access,* 11, 88008–88024. https://doi.org/10.1109/ACCESS.2023.3302356

Farshidi, S., Beer Kwantes, I., & Jansen, S. (2024). Business process modeling language selection for research modelers. *Software and Systems Modeling,* 23, 137–162. https://doi.org/10.1007/s10270-023-01110-8

Gao, C., Hu, X., Gao, S., Xia, X., & Jin, Z. (2025). The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology,* 34(5), Article 127, 1–30. https://doi.org/10.1145/3712005

Ghale, D. P., & Dabbagh, M. (2025). Automated code comments generation using large language models: Empirical evaluation of T5 and BART. *IEEE Access,* 13, 141420–141433. https://doi.org/10.1109/ACCESS.2025.3597601

Haapakoski, J. (2023). *Implementing asynchronous sagas in a microservice architecture* [Master's thesis, Tampere University].

Hassan, H. B., Sarhan, Q. I., & Beszédes, Á. (2024). Evaluating Python static code analysis tools using FAIR principles. *IEEE Access,* 12, 173647–173659. https://doi.org/10.1109/ACCESS.2024.3503493

Hovhannisyan, A., Fan, Y., Rodríguez-Pérez, G., & Kula, R. G. (2024). Linking code and documentation churn: Preliminary analysis. In 2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 107–108). *IEEE.* https://doi.org/10.1109/ISSREW63542.2024.00058

Karlsson, S., Čaušević, A., & Sundmark, D. (2020). QuickREST: Property-based test generation of OpenAPI-described RESTful APIs. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 131–141). *IEEE.* https://doi.org/10.1109/ICST46399.2020.00023

Khan, J. Y., & Uddin, G. (2022). Automatic code documentation generation using GPT-3. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (pp. 1–6). *Association for Computing Machinery.* https://doi.org/10.1145/3551349.3559548

Kiecker, T., Sparka, J. A., Reuter, M., Ziegler, A., & Grunske, L. (2026). CASCADE: Detecting inconsistencies between code and documentation with automatic test generation. *Proceedings of the ACM on Software Engineering,* 3(FSE), Article FSE168, 3816–3838. https://doi.org/10.1145/3808175

Sciencevolution
ISSN: 2810-8728 (En línea)
5.3
JULIO - SEPTIEMBRE
2026
Artículo Original
33 - 48

Gabriel Sánchez Bautista
ORCID:0000-0002-9955-8711
Erick Medel Galindo
ORCID: 0009-0009-8008-6310

https://doi.org/10.61325/pj992973

Lee, W.-T., Song, P.-Y., & Tsai, M.-K. (2025). A data-consistent microservices architecture library using saga design pattern and backup mechanism. Journal of Information *Science and Engineering*, 41(1), 121–136. https://doi.org/10.6688/JISE.202501_41(1).0007

Padoan, F., Santos, R. de S., & Medeiros, R. P. (2024). Charting a path to efficient onboarding: The role of software visualization. In Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE '24) (pp. 133–143). *Association for Computing Machinery.* https://doi.org/10.1145/3641822.3641873

Qiao, M., Li, L., Hou, B., Shang, Z., & Luo, Z. (2025). Research on automated software documentation generation based on modular design. In Proceedings of the 2025 6th International Conference on Computer Science and Management Technology (pp. 1197–1202). *Association for Computing Machinery.* https://doi.org/10.1145/3795154.3795348

Rai, S., Belwal, R. C., & Gupta, A. (2022). A review on source code documentation. *ACM Transactions on Intelligent Systems and Technology,* 13(5), 1–44. https://doi.org/10.1145/3519312

Santos, S., Pimentel, T., Rocha, F. G., & Soares, M. S. (2024). Using behavior-driven development (BDD) for non-functional requirements. *Software*, 3(3), 271–283. https://doi.org/10.3390/software3030014

Tan, W. S., Wagner, M., & Treude, C. (2024). Detecting outdated code element references in software repository documentation. *Empirical Software Engineering*, 29(1), Article 5. https://doi.org/10.1007/s10664-023-10397-6

Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access,* 11, 88339–88358. https://doi.org/10.1109/ACCESS.2023.3305687

Wu, X. (2024). *Diseño e implementación de una plataforma de gestión financiera personal* [Trabajo de Fin de Grado, Universidad Politécnica de Madrid]. https://oa.upm.es/82475/

Yang, D., Simoulin, A., Qian, X., Liu, X., Cao, Y., Teng, Z., & Yang, G. (2025). DocAgent: A multi-agent system for automated code documentation generation. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations) (pp. 460–471). *Association for Computational Linguistics.* https://doi.org/10.18653/v1/2025.acl-demo.44

Yıldız, A., & Demirörs, O. (2025). MicroArc: Event driven analysis and design method for microservices. *Procedia Computer Science,* 263, 583–590. https://doi.org/10.1016/j.procs.2025.07.070

Conflicto de intereses
Los autores declaran que no tienen ningún conflicto de intereses.

Financiación
Todos los costos asociados a la realización de este estudio fueron cubiertos por el autor. No se recibió financiación, subvención ni patrocinio externo.

Nota del editor
La revista Sciencevolution mantiene una posición neutral respecto a posibles reclamaciones jurisdiccionales derivadas de mapas publicados o afiliaciones institucionales.

© El/Los autor(es) 2026. Este artículo se distribuye bajo los términos de la Licencia Internacional Creative Commons Atribución 4.0 (CC BY 4.0), que permite el uso, la distribución, la adaptación y la reproducción en cualquier medio o formato, siempre que se otorgue el crédito adecuado al/los autor(es) y a la fuente, se incluya un enlace a la licencia y se indique si se han realizado cambios.

Las imágenes u otro material de terceros incluidos en este artículo están cubiertos por la licencia Creative Commons del artículo, salvo que se indique lo contrario en una línea de crédito. Si el material no está incluido en la licencia Creative Commons y el uso previsto no está permitido por la legislación vigente o excede el uso permitido, será necesario obtener autorización directamente del titular de los derechos.

Puede consultarse una copia de la licencia en: http://creativecommons.org/licenses/by/4.0/

La renuncia a la dedicación al dominio público de Creative Commons (CC0 1.0; http://creativecommons.org/publicdomain/zero/1.0/)
Se aplica a los datos disponibles en este artículo, salvo indicación contraria en la línea de crédito correspondiente.