**Especificación de requisitos de software**

**Proyecto: ALKOSTO-DUPLICACIÓN DE PÁGINA WEB**

Revisión 2.0

Juan Manuel Galvis

Michael Hurtado

Alexánder Mesa Gómez

Harold Ordoñez

Octubre 2025

**TABLA DE CONTENIDO**

[**1. Introducción 4**](#_u2hfw3q5p2jd)

[**2. Marco Teórico 4**](#_ewt6yi2s0zl)

[**2.1. Definición de Pruebas de Software 4**](#_h7rqb1mw3w7z)

[**2.2. Tipos de Pruebas 5**](#_xyq0fl0zl2b)

[**3. Clasificación de Pruebas en el Proyecto 6**](#_69yd182zx4qt)

[**3.1. Pruebas Unitarias 7**](#_t7ovtx2ijzgg)

[**3.2. Pruebas de Integración 7**](#_b1bwwuwdkmp4)

[**3.3. Pruebas del Sistema 7**](#_eg5mvzl5xwzv)

[**3.4. Pruebas de Aceptación 7**](#_okpksw3bbf59)

[**3.5. Pruebas de Rendimiento 7**](#_harocr8ynrei)

[**3.6. Pruebas de Seguridad 7**](#_uk5bffotg8)

[**3.7. Pruebas de Usabilidad 7**](#_jixu1k7twnvr)

[**3.8. Pruebas de Compatibilidad 7**](#_lc18fwfu4ryb)

[**4. Diagramas UML Aplicados a Requerimientos 8**](#_6wvs8qfzuc8)

[**4.1. Diagramas de Clases: 8**](#_akhm5yzij1ch)

[**4.2. Diagramas de Paquetes: Organizan los módulos del sistema y sus dependencias. 19**](#_6n7oel9dzb2b)

[4.2.1. RF01 - Registrar Usuario 20](#_82ltm31pt61y)

[4.2.2. RF02 - Iniciar sesión 21](#_82ltm31pt61y)

[4.2.3. RF03 - Recuperar contraseña 22](#_82ltm31pt61y)

[4.2.4. RF04 Verificar correo y teléfono 23](#_82ltm31pt61y)

[4.2.5. RF06 - Buscar Producto 24](#_82ltm31pt61y)

[4.2.6. RF07 - Filtrar categorías 25](#_82ltm31pt61y)

[4.2.7. RF10 - Añadir a favoritos 26](#_82ltm31pt61y)

[4.2.8. RF12 - Ver favoritos 27](#_82ltm31pt61y)

[4.2.9. RF14 - Añadir al carrito 28](#_82ltm31pt61y)

[4.2.10. RF17 - Ver el carrito 29](#_1svodiiryjra)

[**4.3. Diagramas de Secuencia: 30**](#_7oajy319qy8q)

[4.3.1. RF01 - Registrar Usuario 31](#_fxj3xmam8g0e)

[4.3.2. RF02 - Iniciar sesión 32](#_fxj3xmam8g0e)

[4.3.3. RF03 - Recuperar contraseña 33](#_fxj3xmam8g0e)

[4.3.4. RF04 - Verificar correo y teléfono 34](#_fxj3xmam8g0e)

[4.3.5. RF06 - Buscar Producto 35](#_fxj3xmam8g0e)

[4.3.6. RF07 - Filtrar categorías 35](#_fxj3xmam8g0e)

[4.3.7. RF10 - Añadir a favoritos 36](#_fxj3xmam8g0e)

[4.3.8. RF12 - Ver favoritos 37](#_fxj3xmam8g0e)

[4.3.9. RF14 - Añadir al carrito 38](#_fxj3xmam8g0e)

[4.3.10. RF17 - Ver el carrito 39](#_e5550jujko4a)

[**4.4. Diagramas de Actividades: 40**](#_lplsm7e6vxen)

[**5. Aplicación Práctica de Pruebas 52**](#_68pr9e56ant6)

[5.1. Selección de Requerimientos a Probar: Se priorizan los RF relacionados con login, registro, carrito y compra. 52](#_iuhhv0pvcnbg)

[5.2. Diseño de Casos de Prueba: Se definen entradas, salidas esperadas y criterios de aceptación por caso. 52](#_dob4i6wtokn0)

[5.3. Ejecución de Casos de Prueba: Se realizan las pruebas en entorno controlado, documentando evidencias. 52](#_p2wsmjx1esje)

[5.4. Resultados y Evidencias: Se anexan capturas de pantalla y resultados de cada ejecución. 52](#_moozl46i9wkt)

[**6. Conclusiones 52**](#_ei8mmqpogm4q)

## 

# Introducción

El presente documento tiene como propósito principal describir, planificar y aplicar de manera sistemática diferentes tipos de pruebas de software al proyecto Alkosto React, una réplica funcional del portal de compras en línea de Alkosto. Este proyecto se desarrolla bajo un enfoque ágil (Scrum), utilizando tecnologías modernas como React.js para el frontend y Django (Python) para el backend, junto con una base de datos MySQL, lo cual garantiza una arquitectura escalable, mantenible y segura.

La importancia de esta documentación radica en asegurar la calidad del producto, verificando que cada uno de los módulos desarrollados cumpla con los requerimientos funcionales y no funcionales definidos durante la fase de análisis y diseño. Para ello, se aplican diversas técnicas de prueba, entre ellas pruebas unitarias, de integración, de sistema, de aceptación, de rendimiento, de seguridad, de usabilidad y de compatibilidad, todas ellas alineadas con las buenas prácticas de Ingeniería de Software.

Asimismo, este documento incluye diagramas UML que ilustran la estructura del sistema y el comportamiento de los componentes, lo cual permite una mejor comprensión del flujo de ejecución y facilita la detección temprana de errores. La correcta aplicación de las pruebas y su documentación proporcionan una visión clara del estado del software, garantizando un producto confiable, funcional y alineado con las expectativas del cliente y los usuarios finales.

# Marco Teórico

## Definición de Pruebas de Software

Las pruebas de software son un proceso fundamental en el ciclo de desarrollo, orientado a verificar y validar que un sistema cumple con los requerimientos definidos. Su objetivo principal es detectar errores o desviaciones respecto al comportamiento esperado antes de que el producto llegue al usuario final. Estas pruebas no solo garantizan la calidad técnica del sistema, sino también su confiabilidad, escalabilidad, seguridad y facilidad de uso, atributos esenciales para el éxito de un producto en producción.

## 

## 2.2. Tipos de Pruebas

1. **Pruebas Unitarias:** Verifican el funcionamiento de componentes individuales o “unidades” de código.
2. **Pruebas Funcionales:** Comprueban que cada función especificada en los requerimientos opera correctamente.
3. **Pruebas de Integración:** Validan la interacción entre módulos o subsistemas (por ejemplo, UI ⇄ Backend).
4. **Pruebas del Sistema:** Evalúan el sistema en su conjunto frente a los requisitos funcionales y no funcionales.
5. **Pruebas de Aceptación:** Realizadas por el cliente o usuario final para validar que el sistema cumple con sus expectativas y criterios de negocio.
6. **Pruebas de Rendimiento:** Miden velocidad, capacidad de respuesta y uso de recursos bajo carga (estrés, carga, escalabilidad).
7. **Pruebas de Seguridad:** Identifican vulnerabilidades, asegurando confidencialidad, integridad y disponibilidad.
8. **Pruebas de Usabilidad:** Evalúan la facilidad de uso, accesibilidad y experiencia de usuario.
9. **Pruebas de Compatibilidad:** Verifican funcionamiento en distintas plataformas, navegadores, dispositivos y sistemas operativos.

| **Tipo** | **Descripción** | **Objetivo** |
| --- | --- | --- |
| **Unitarias** | Se prueban funciones o componentes individuales. | Verificar que cada unidad funcione correctamente. |
| **Integración** | Se prueban los módulos combinados. | Validar comunicación entre componentes. |
| **Sistema** | Prueba todo el sistema completo. | Validar cumplimiento de requerimientos. |
| **Aceptación** | Prueba final con el cliente. | Confirmar que cumple con lo solicitado. |
| **Rendimiento** | Evalúa tiempos de respuesta y carga. | Garantizar eficiencia. |
| **Seguridad** | Evalúa vulnerabilidades y accesos. | Garantizar confidencialidad y protección de datos. |
| **Usabilidad** | Evalúa facilidad de uso. | Validar experiencia del usuario. |
| **Compatibilidad** | Verifica funcionamiento en distintos dispositivos y/o navegadores. | Asegurar accesibilidad total. |

# Clasificación de Pruebas en el Proyecto

Para el aplicativo Alkosto, se seleccionaron los siguientes tipos de pruebas como los más relevantes según el alcance actual del proyecto:

• **Pruebas unitarias:** validar funciones críticas como añadir al carrito o iniciar sesión.

• **Pruebas de integración:** verificar la conexión entre frontend (React) y backend (Django).

• **Pruebas de sistema:** validar el flujo completo de compra desde el carrito hasta la confirmación.

• **Pruebas de aceptación:** confirmar cumplimiento con los requerimientos del cliente.

• **Pruebas de rendimiento:** evaluar la carga de catálogos y búsqueda de productos.

• **Pruebas de seguridad:** validar autenticación, cifrado y manejo de sesiones.

• **Pruebas de usabilidad:** analizar la facilidad de uso e interacción.

• **Pruebas de compatibilidad:** comprobar funcionamiento en navegadores y dispositivos móviles.

## Pruebas Unitarias

## Pruebas de Integración

## Pruebas del Sistema

## Pruebas de Aceptación

## Pruebas de Rendimiento

## Pruebas de Seguridad

## Pruebas de Usabilidad

## Pruebas de Compatibilidad

# Diagramas UML Aplicados a Requerimientos

## Diagramas de Clases:

Los diagramas de clases son una herramienta de UML que permiten representar de manera visual la estructura estática de un sistema, mostrando las clases que lo conforman, sus atributos, métodos y las relaciones que existen entre ellas. Son útiles para comprender cómo se organiza la información y la lógica del sistema, así como para establecer las bases de la implementación en código orientado a objetos.

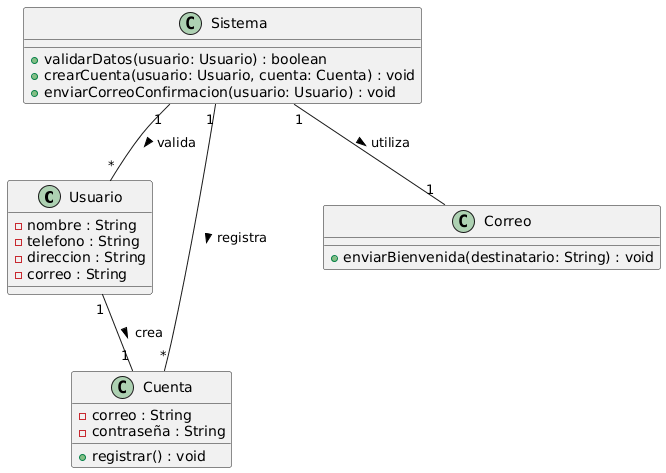
En nuestro proyecto, los diagramas de clases permiten modelar elementos como usuarios, cuentas, productos y pedidos, junto con sus interacciones con el sistema. De esta forma, ayudan a visualizar cómo se gestionan los datos, cómo se conectan las distintas entidades y qué operaciones se pueden realizar, facilitando la transición entre el diseño conceptual y la implementación.

**¿Cómo se realizaron los diagramas?** Para crear los diagramas de clases, seguimos estos pasos:

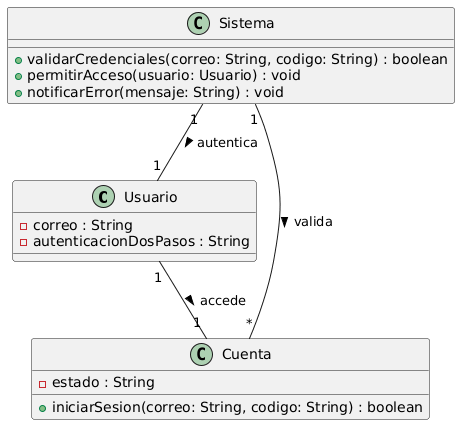
* **Identificación de entidades principales:** Se seleccionaron los elementos clave del sistema a partir de los casos de uso (usuarios, cuentas, productos, pedidos, etc.).
* **Definición de atributos y métodos:** Se especificaron los datos que cada clase debía contener y las operaciones que podía realizar.
* **Establecimiento de relaciones:** Se modelaron las asociaciones entre clases (uno a uno, uno a muchos, dependencias o composiciones) según los flujos descritos en los casos de uso.
* **Revisión de consistencia:** Se verificó que cada clase y relación respondiera a un requerimiento del sistema y no existieran duplicaciones o contradicciones.

Para la generación de los diagramas, utilizamos **PlantUML**, un lenguaje de modelado que permite definir clases y relaciones mediante texto, generando diagramas claros y consistentes. Esto facilitó mantener un diseño organizado y comprensible de la arquitectura del sistema.

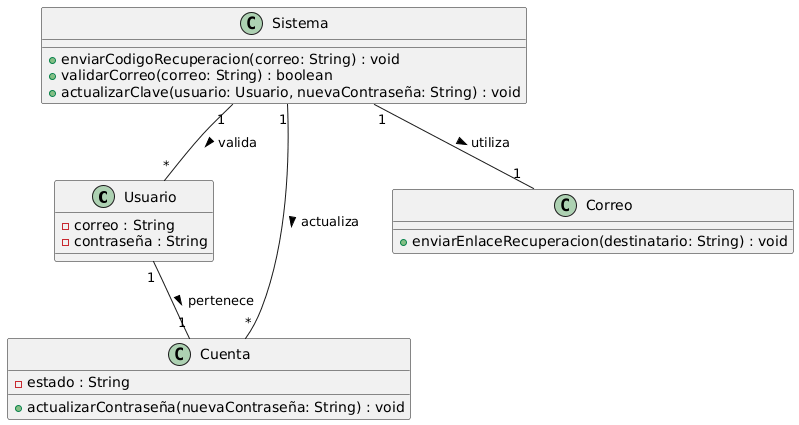
* + 1. RF01: Registrar Usuario



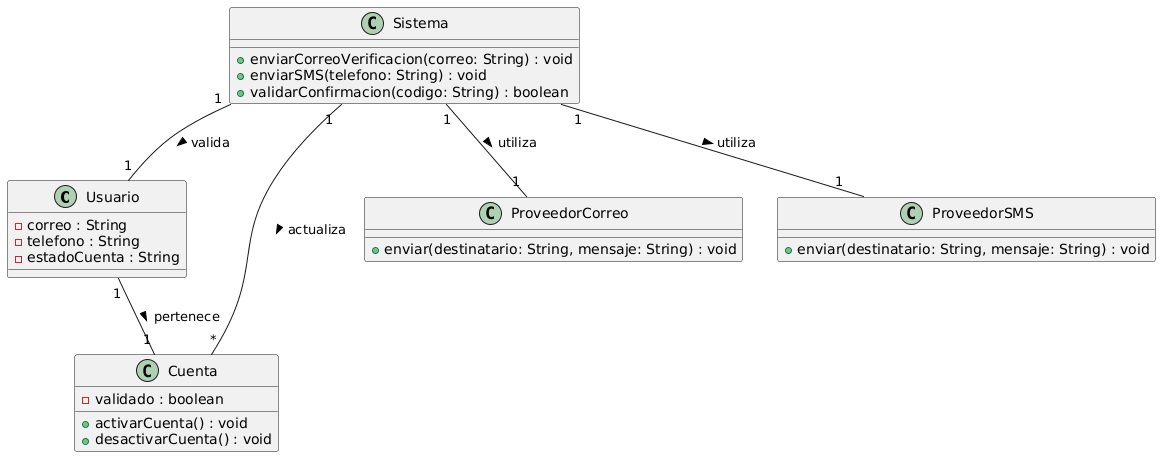
* + 1. RF02: Iniciar Sesión



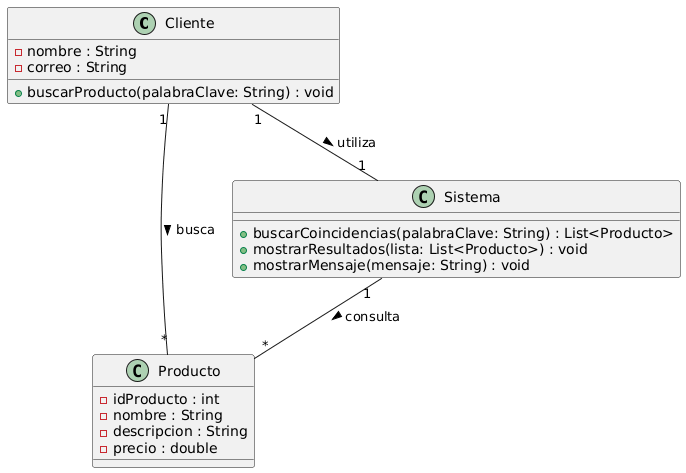
* + 1. RF03 - Recuperar contraseña

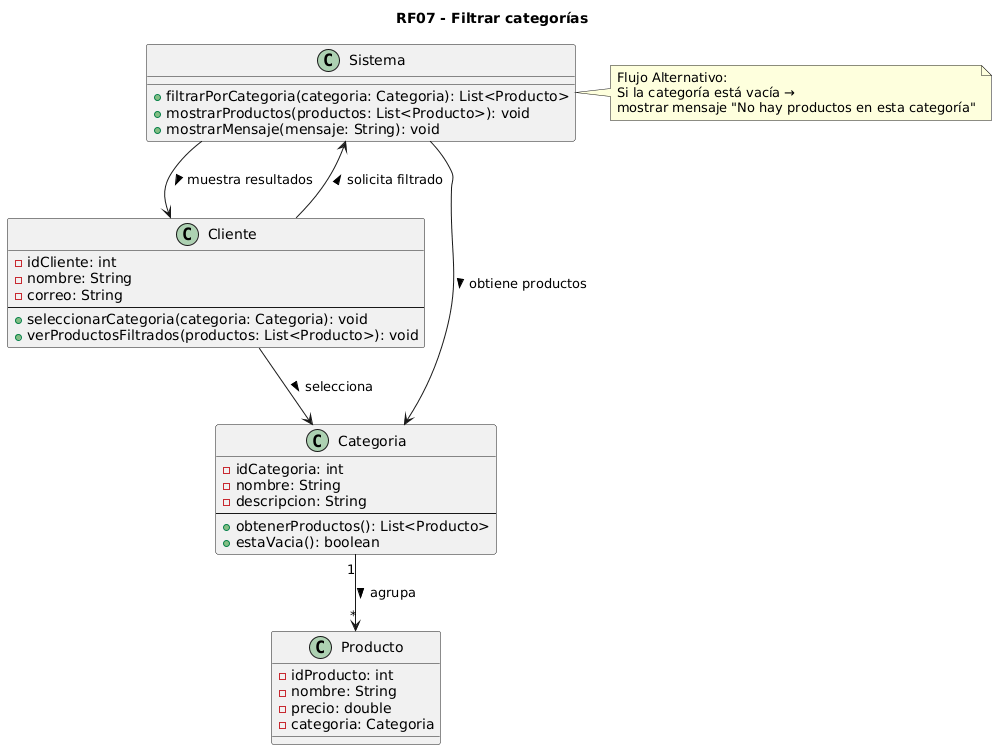


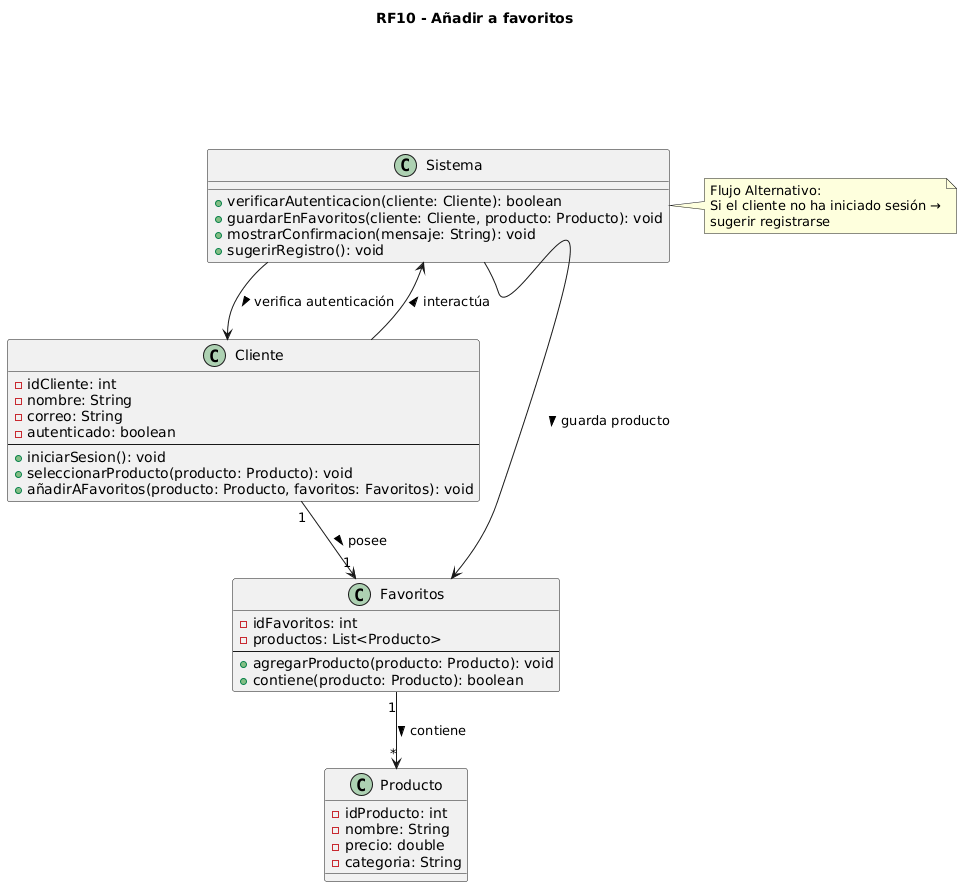
* + 1. RF04 - Verificar correo y teléfono

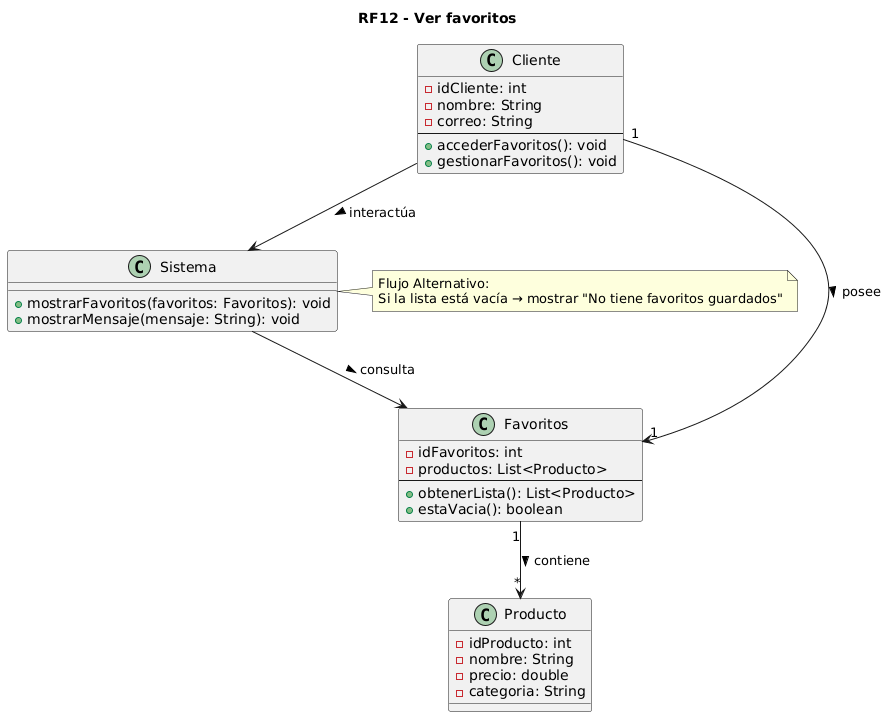
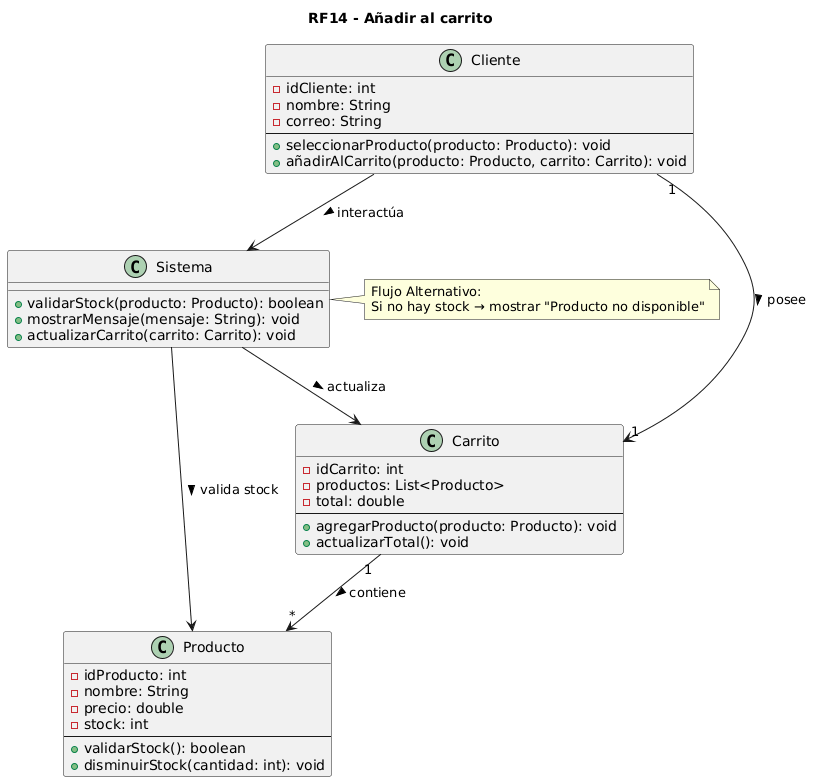


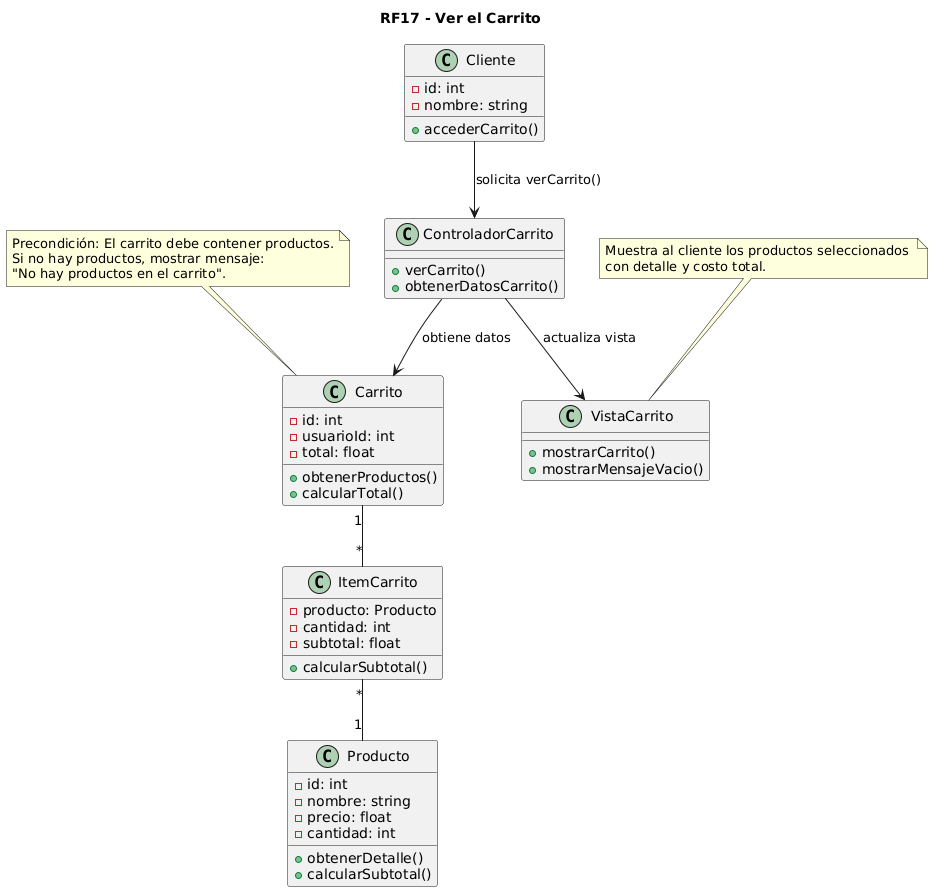
* + 1. RF06 - Buscar Producto



* + 1. RF07 - Filtrar categorías
    2. RF10 - Añadir a favoritos



* + 1. RF12 - Ver favoritos
    2. RF14 - Añadir al carrito
    3. RF17 - Ver el carrito



## Diagramas de Paquetes: Organizan los módulos del sistema y sus dependencias.

Los diagramas de paquetes en UML son una herramienta que permite organizar y estructurar los elementos del sistema en módulos o agrupaciones lógicas, facilitando la comprensión de su arquitectura. Estos diagramas muestran cómo se dividen las diferentes partes del software en paquetes y cómo interactúan entre sí, lo que resulta especialmente útil en sistemas complejos donde se requiere modularidad y separación de responsabilidades.

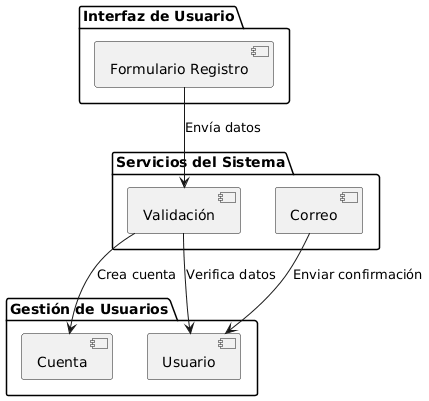
En nuestro proyecto, los diagramas de paquetes permiten visualizar cómo se distribuyen las funcionalidades principales, como la gestión de usuarios, productos y servicios, así como los módulos de interfaz y los servicios internos del sistema. Esto ayuda a tener una visión clara de la organización del software antes de su implementación.

**¿Cómo se realizaron los diagramas de paquetes?**

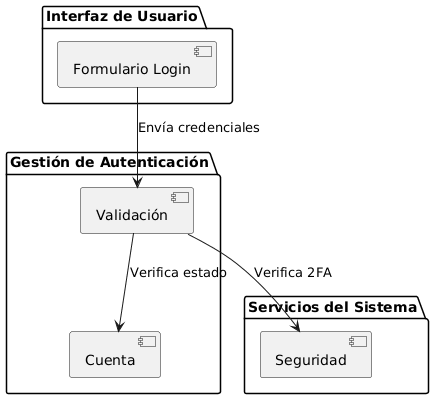
1. **Identificación de módulos principales:** Se agruparon las funcionalidades del sistema en paquetes como interfaz de usuario, gestión de productos, servicios del sistema y seguridad.
2. **Definición de responsabilidades:** Se asignaron los elementos y clases relevantes dentro de cada paquete, según su rol en el sistema.
3. **Establecimiento de dependencias:** Se representaron las relaciones entre los paquetes para mostrar cómo interactúan y se comunican.
4. **Verificación de coherencia:** Se revisó que la organización reflejara de forma clara y ordenada los requerimientos del sistema.

Para la generación de estos diagramas utilizamos PlantUML, una herramienta que permite crear representaciones visuales del modelo de manera clara, estructurada y fácilmente integrable en entornos de desarrollo colaborativos.

## RF01 - Registrar Usuario

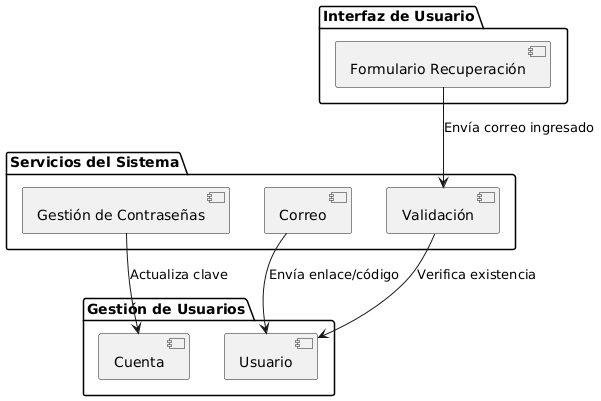


## RF02 - Iniciar sesión

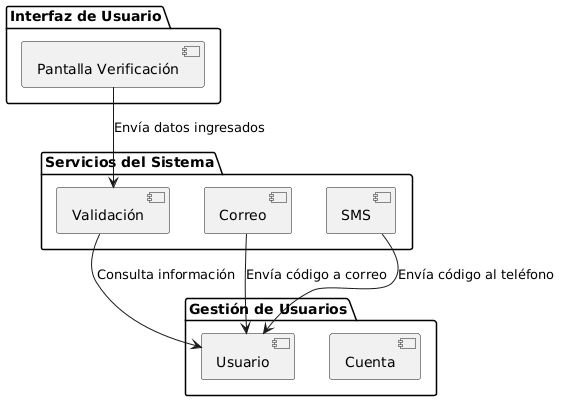


## RF03 - Recuperar contraseña

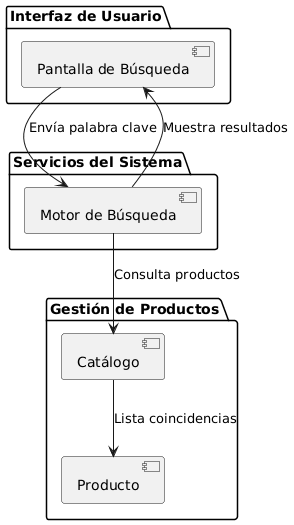
## 



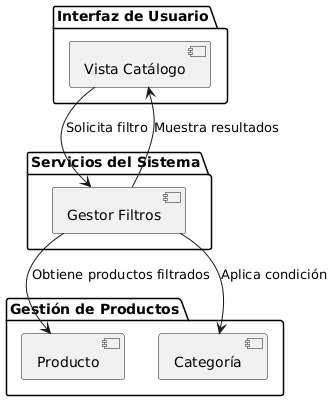
## RF04 Verificar correo y teléfono



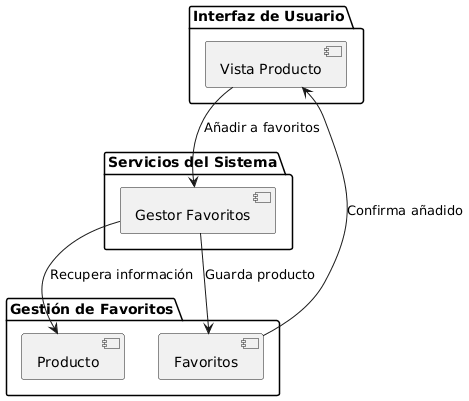
## RF06 - Buscar Producto



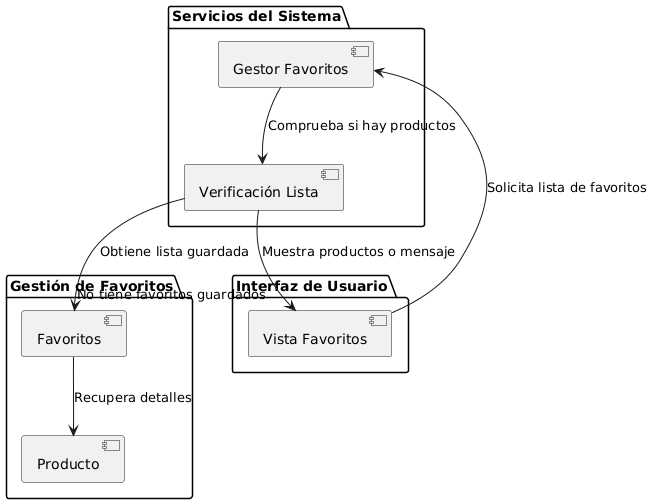
## RF07 - Filtrar categorías



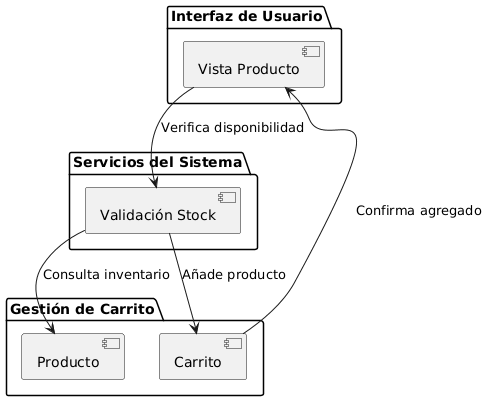
## RF10 - Añadir a favoritos



## RF12 - Ver favoritos



## RF14 - Añadir al carrito



## RF17 - Ver el carrito

## 

## 

## Diagramas de Secuencia:

Los diagramas de secuencia son una herramienta de UML que permiten representar de forma visual cómo interactúan los actores y los componentes del sistema a lo largo del tiempo. Estos diagramas muestran el intercambio de mensajes entre objetos o entidades, organizados en un eje temporal, lo que facilita entender la dinámica de un proceso. Son útiles para detallar el comportamiento de un caso de uso, identificando el orden de las acciones, los mensajes enviados y las posibles alternativas en la ejecución.

En nuestro proyecto, los diagramas de secuencia permiten visualizar de manera clara el flujo de interacción en procesos clave como registrar un usuario, iniciar sesión, recuperar la contraseña, verificar datos de contacto y buscar productos. Esto contribuye a comprender no solo qué acciones se realizan, sino también en qué orden y entre qué participantes, lo que facilita la implementación y validación de los requerimientos del sistema.

**¿Cómo se realizaron los diagramas?**  
 Para elaborar los diagramas de secuencia seguimos estos pasos:

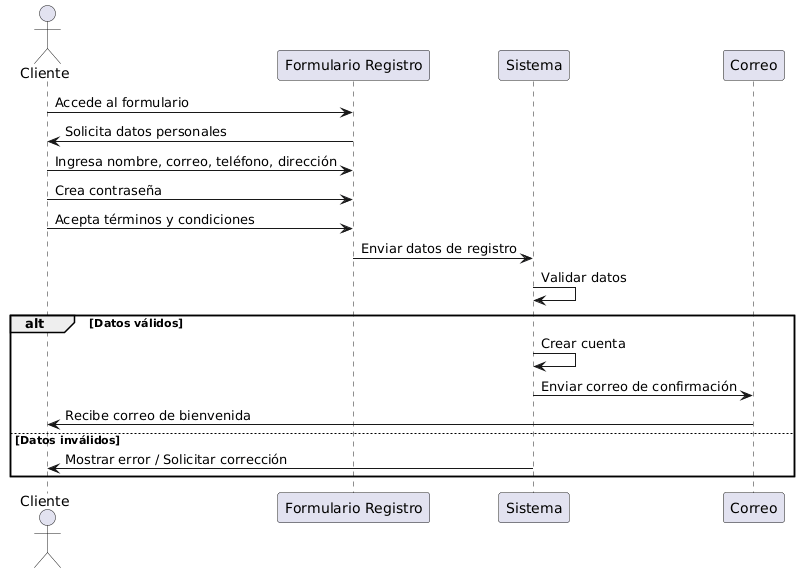
* **Identificación de participantes:** Se determinaron los actores (cliente) y componentes del sistema (formularios, módulos, base de datos, servicios externos) involucrados en cada caso de uso.
* **Definición de mensajes:** Se establecieron las interacciones que ocurren entre participantes, especificando el tipo de acción (envío de datos, validación, confirmación, error).
* **Secuenciación temporal:** Se organizó el orden de los mensajes de arriba hacia abajo, representando el flujo cronológico de la ejecución.
* **Incorporación de alternativas:** Se utilizaron bloques condicionales (alt) para mostrar escenarios alternativos, como errores, datos inválidos o confirmaciones exitosas.

Para la generación de los diagramas, utilizamos PlantUML, un lenguaje de modelado que permite describir secuencias con texto y convertirlas en diagramas visuales claros. Esto nos permitió representar de forma precisa las interacciones entre los usuarios y el sistema en cada caso de uso.

## 

## 

## RF01 - Registrar Usuario



## RF02 - Iniciar sesión

## 

## 

## 

## 

## RF03 - Recuperar contraseña

## 

## 

## 

## 

## 

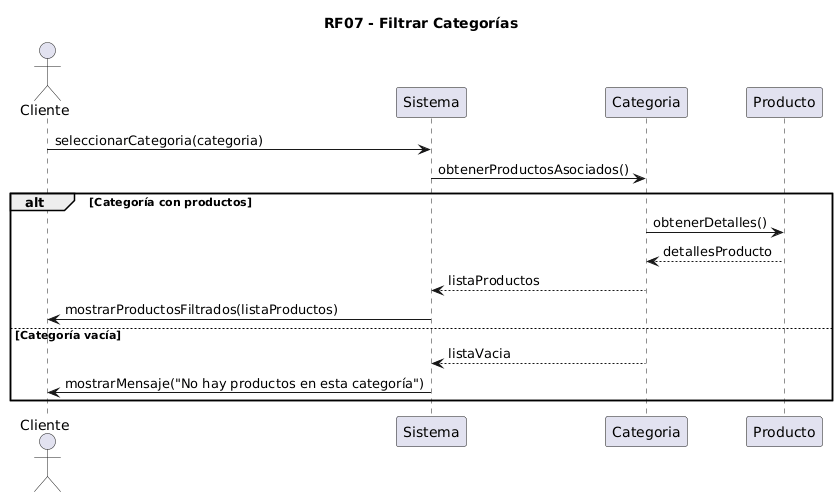
## RF04 - Verificar correo y teléfono

## 

## 

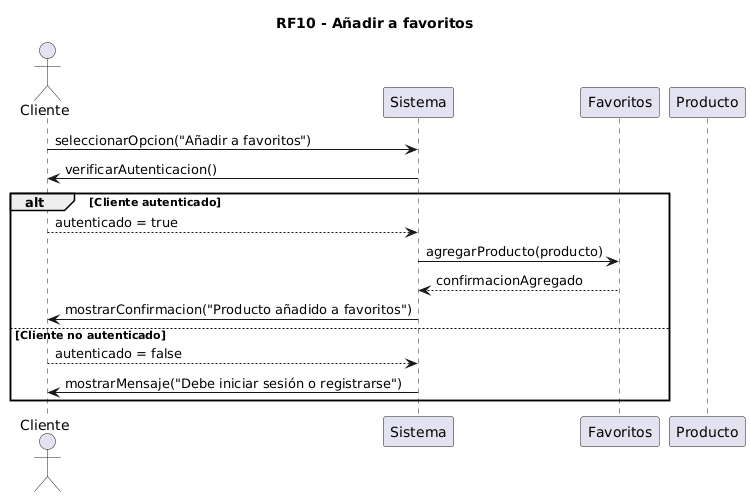
## RF06 - Buscar Producto

## RF07 - Filtrar categorías

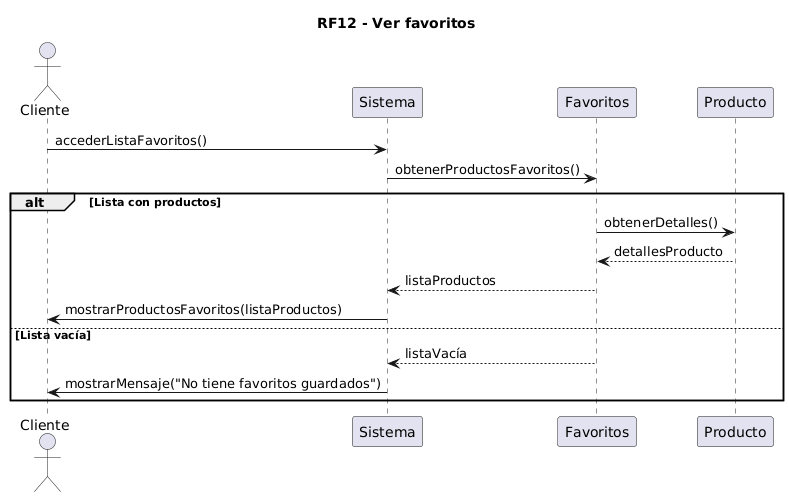


## RF10 - Añadir a favoritos

## 



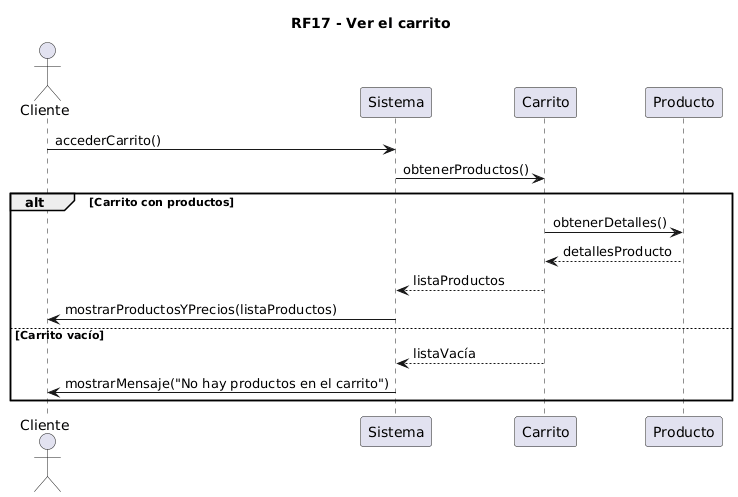
## RF12 - Ver favoritos



## RF14 - Añadir al carrito

## 

## RF17 - Ver el carrito



## 

## Diagramas de Actividades:

Los diagramas de actividad son una herramienta de UML que permiten representar de manera visual el flujo de trabajo o los procesos de un sistema, mostrando cómo se llevan a cabo las acciones paso a paso. Son útiles para comprender la interacción entre los usuarios y el sistema, así como los diferentes caminos que puede tomar un proceso, incluyendo decisiones y excepciones.

En nuestro proyecto, los diagramas de actividad permiten visualizar procesos como registrar un usuario, iniciar sesión, buscar productos, gestionar favoritos y añadir productos al carrito, facilitando la comprensión de las operaciones principales del sistema antes de su implementación.

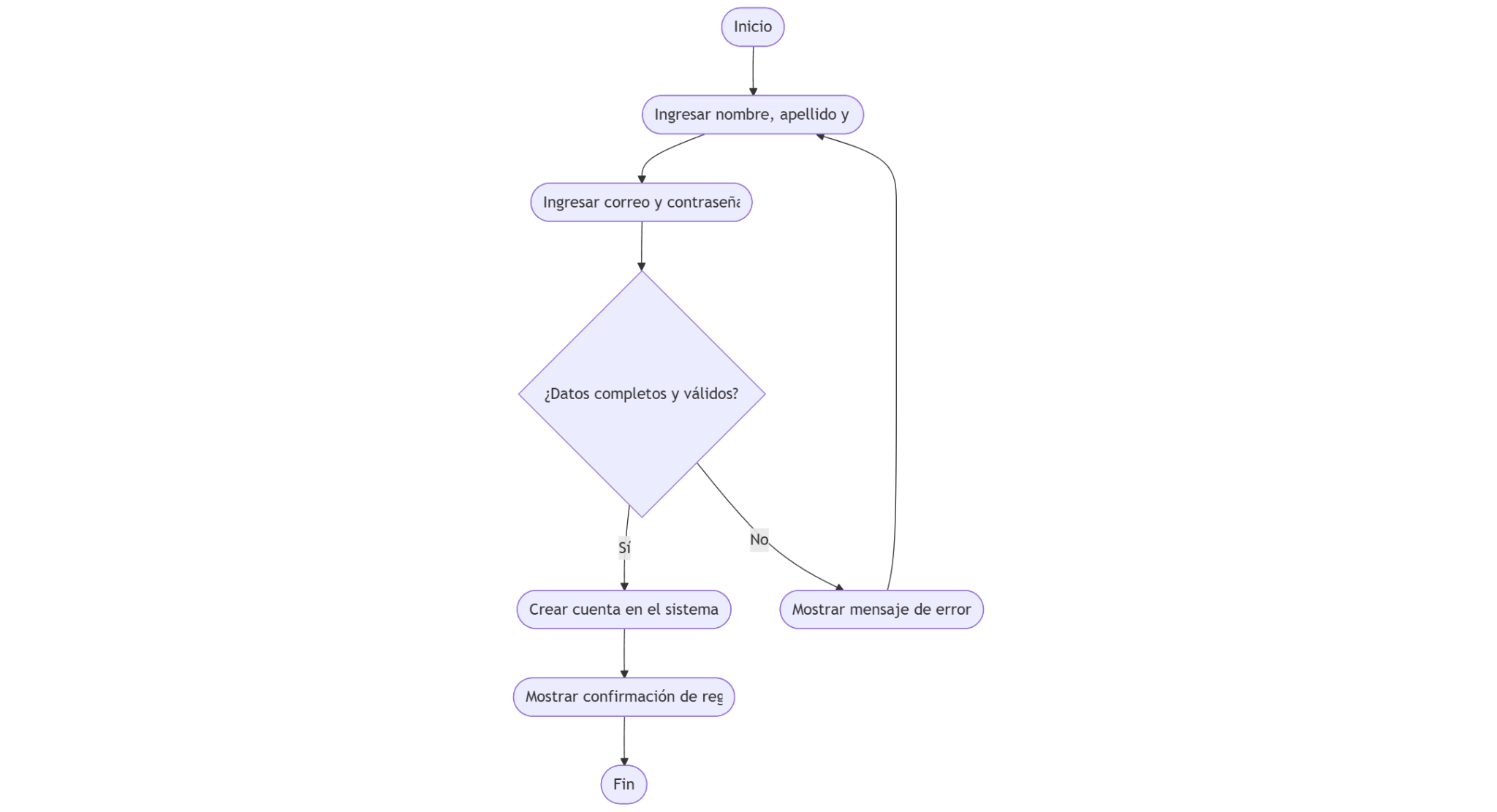
**¿Cómo se realizaron los diagramas?**

Para crear los diagramas de actividad, seguimos estos pasos:

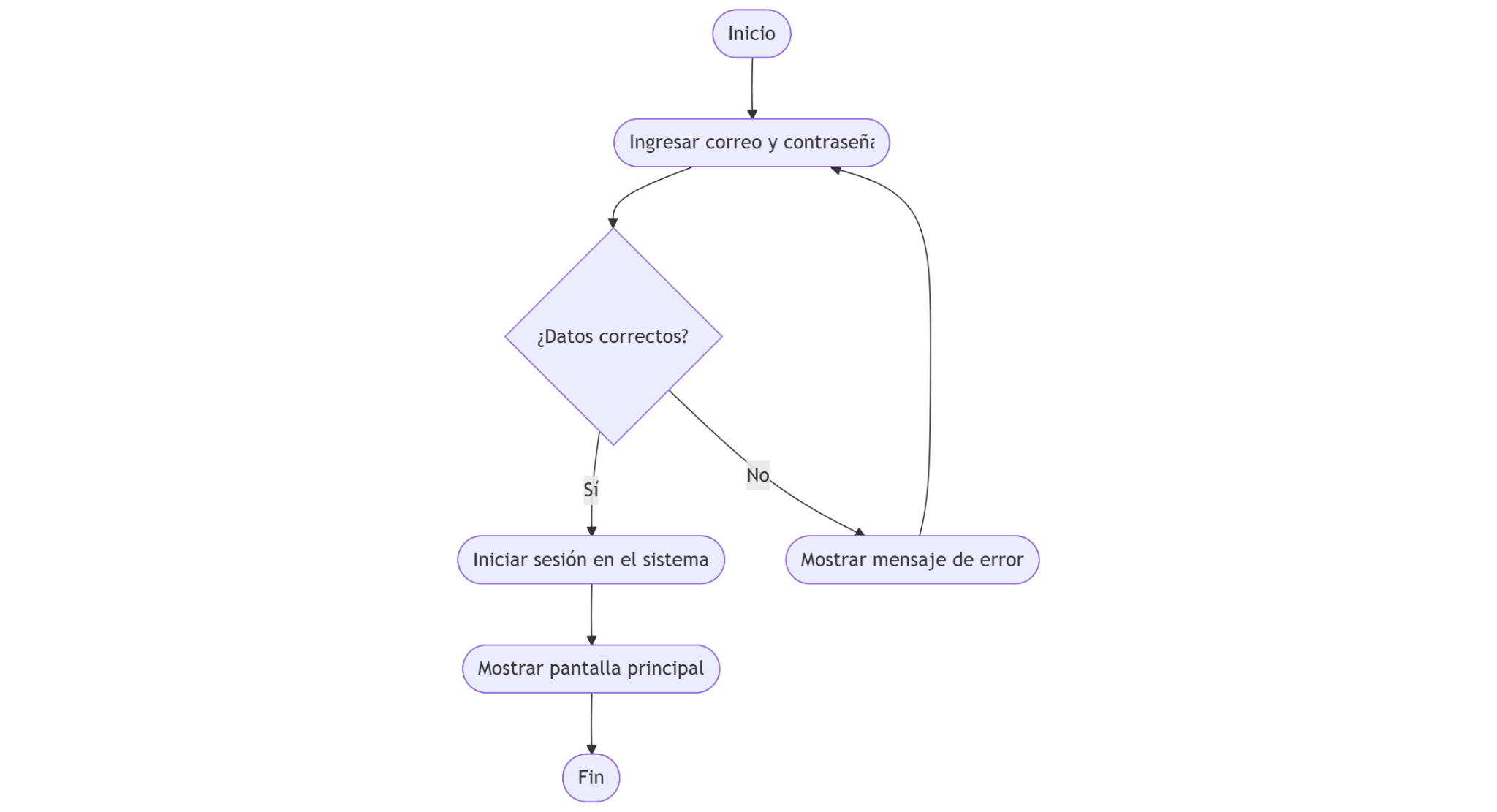
1. **Identificación de procesos clave:** Se seleccionaron los requerimientos más importantes y estructurales del sistema.
2. **Definición de actividades y decisiones**: Se desglosaron las acciones que realiza el usuario y el sistema, incluyendo los puntos de decisión donde el flujo puede dividirse según condiciones específicas.
3. **Secuenciación del flujo:** Se conectaron las actividades mediante flechas, mostrando el orden lógico de ejecución.
4. **Verificación de consistencia:** Se revisó que todas las rutas posibles (éxito, error o excepción) estuvieran correctamente representadas.

Para la generación de los diagramas, utilizamos Mermaid, un lenguaje de marcado que permite crear diagramas visuales de manera sencilla y clara, compatible con entornos en línea. Esto nos permitió producir diagramas limpios y funcionales que reflejan fielmente el flujo de los procesos del sistema.

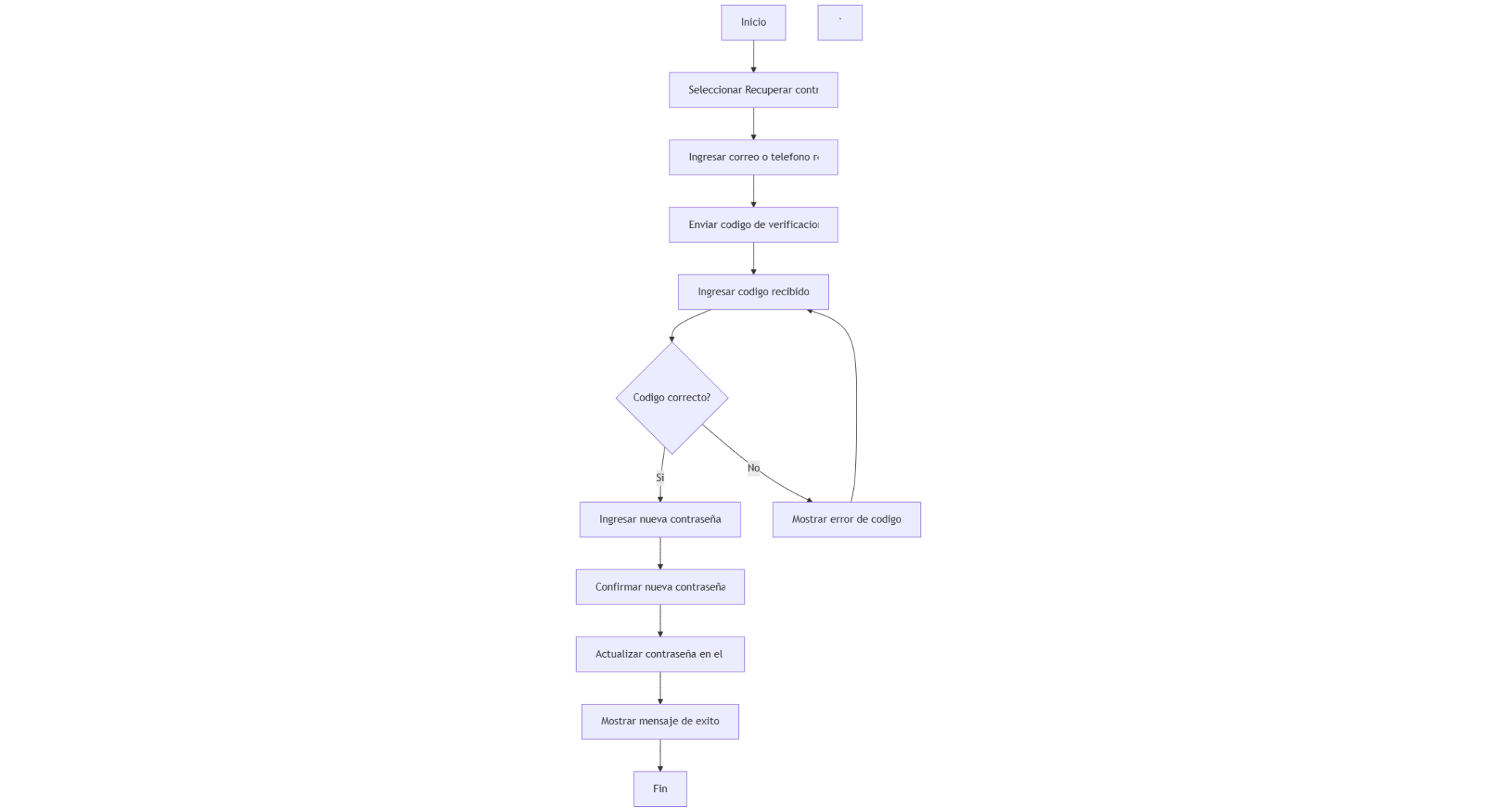
* + 1. **RF01 - Registrar Usuario**



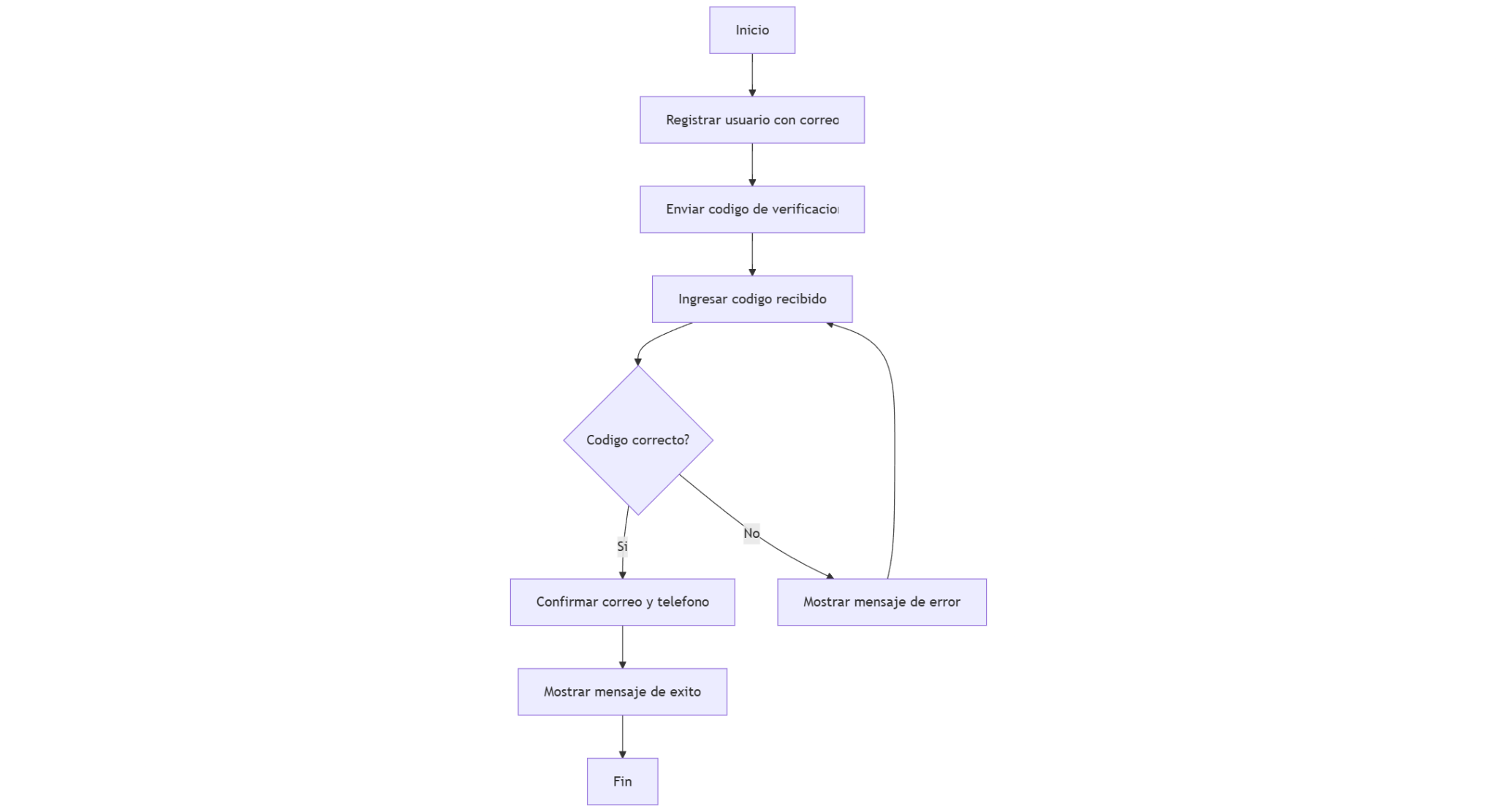
* + 1. **RF02 - Iniciar Sesión**



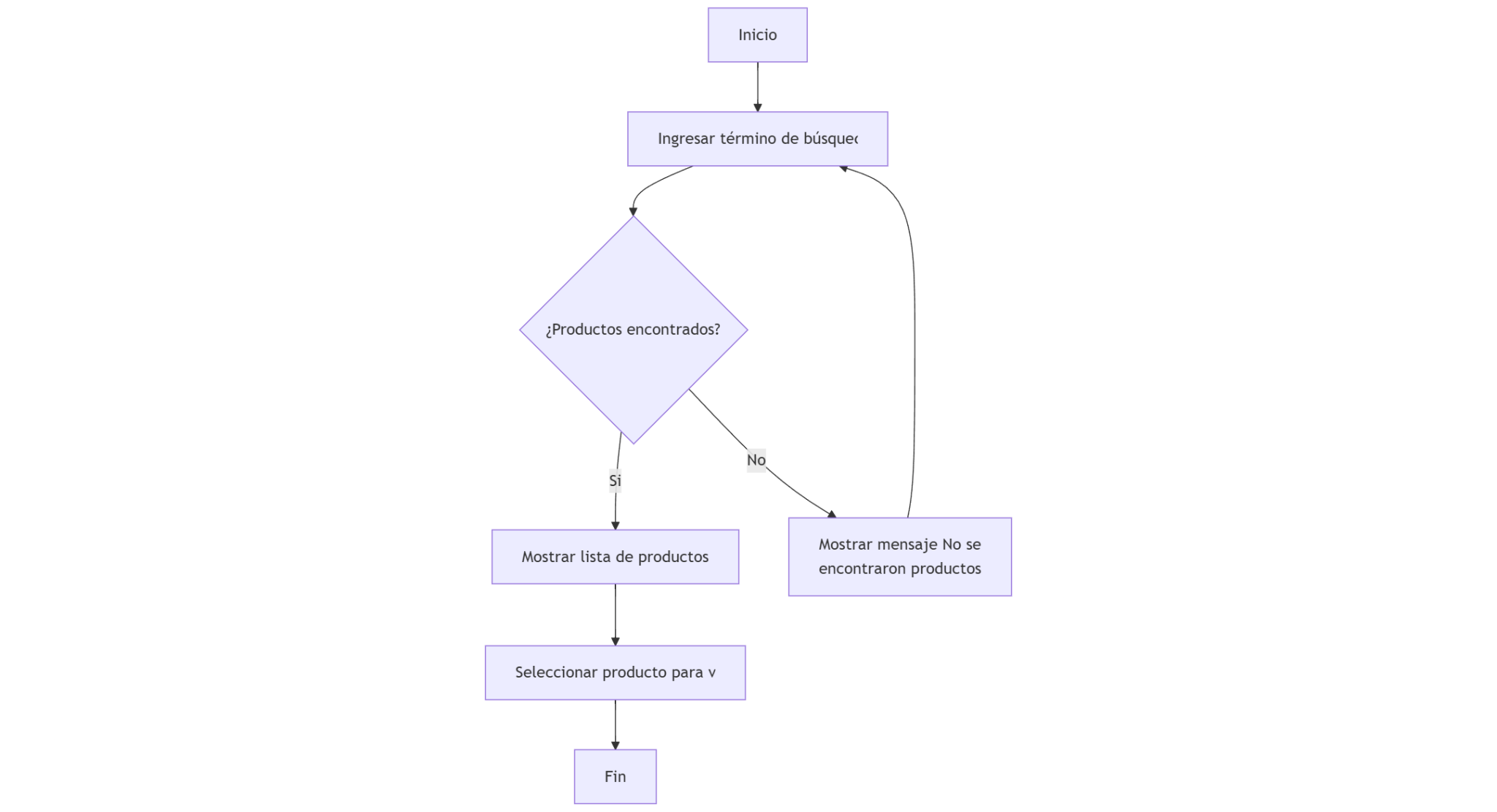
* + 1. **RF03 - Recuperar Contraseña**



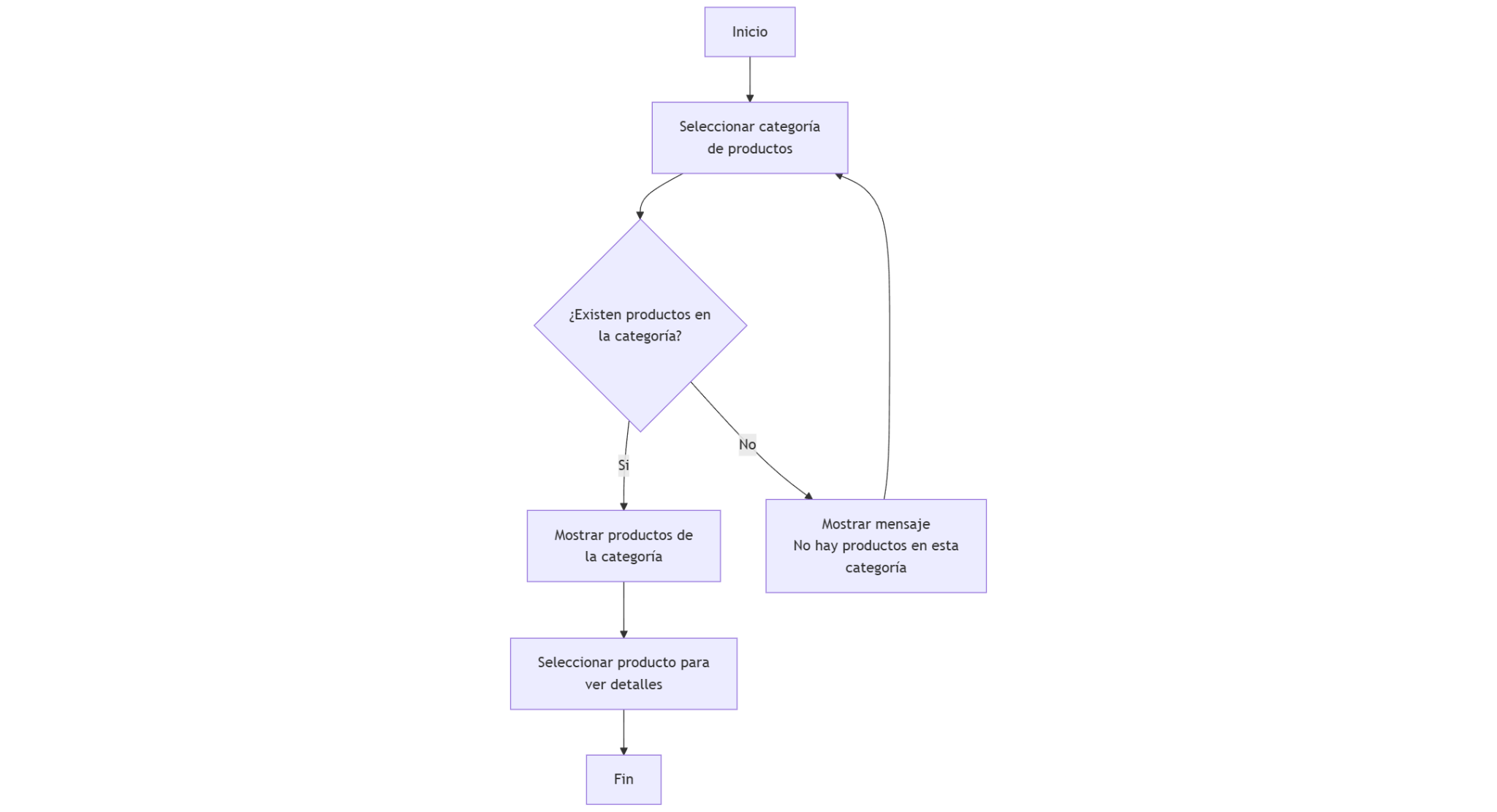
* + 1. **RF04 - Verificar correo y teléfono**



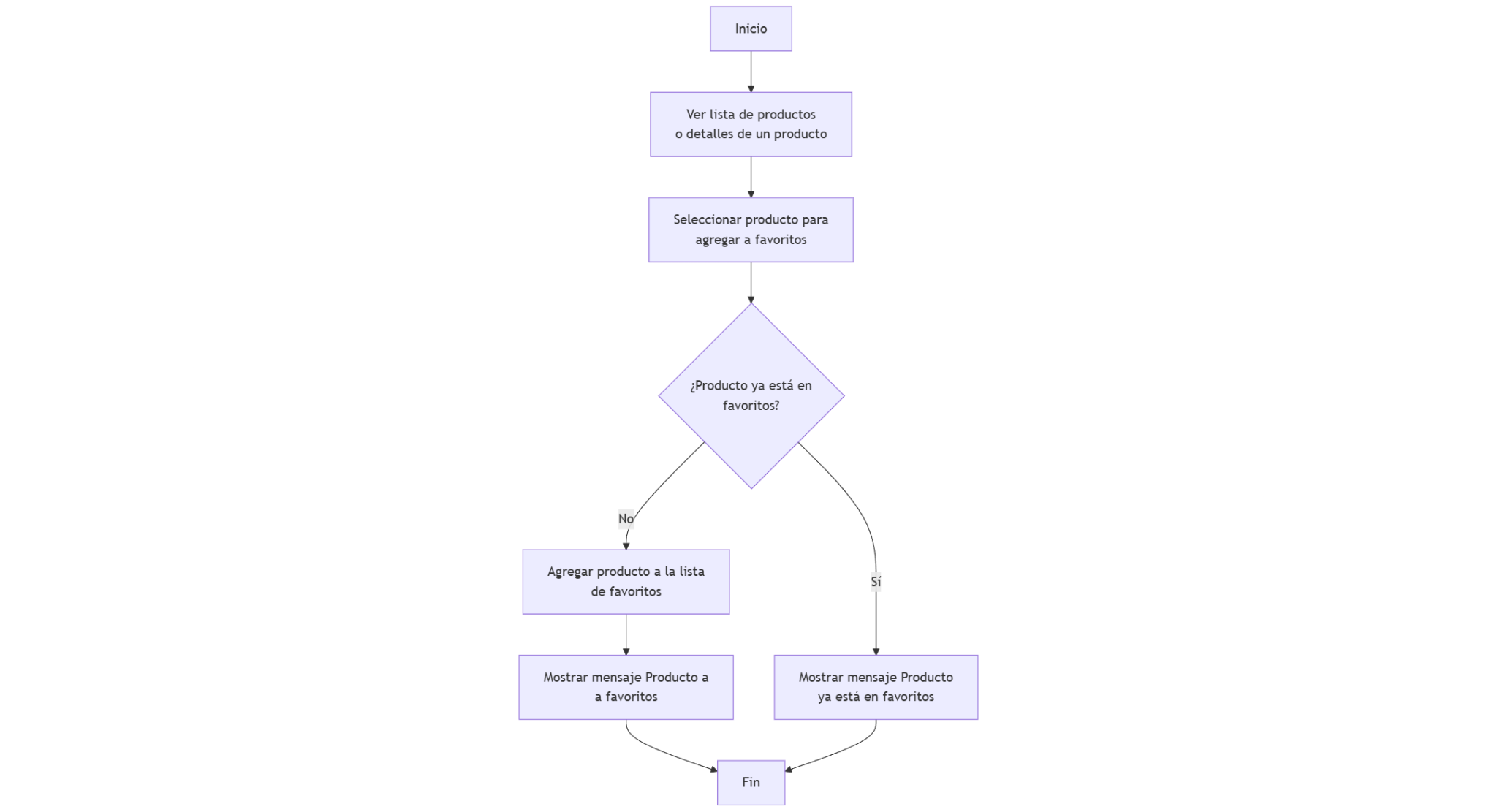
* + 1. **RF06 - Buscar Producto**.



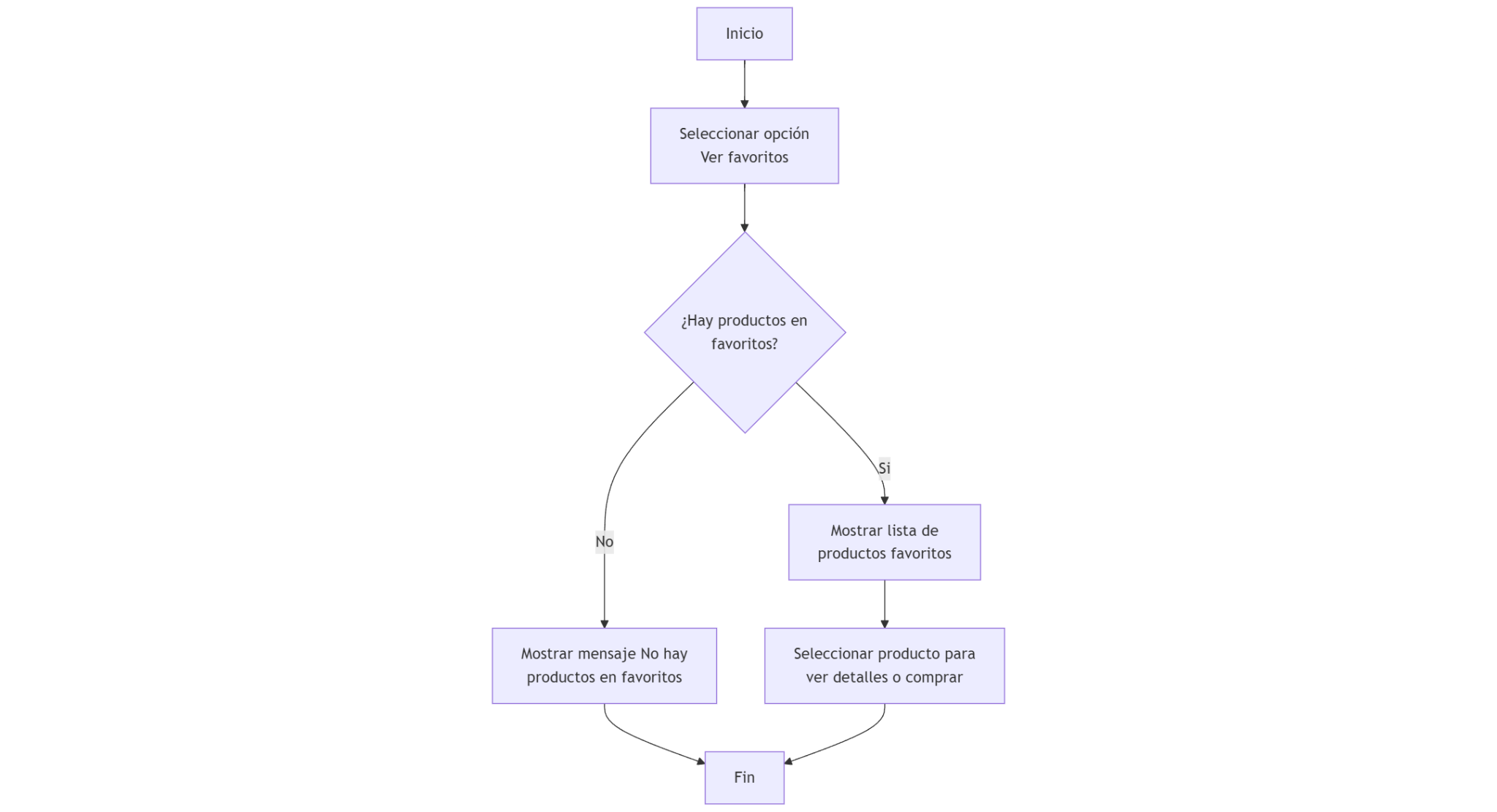
* + 1. **RF07: Filtrar categorías**



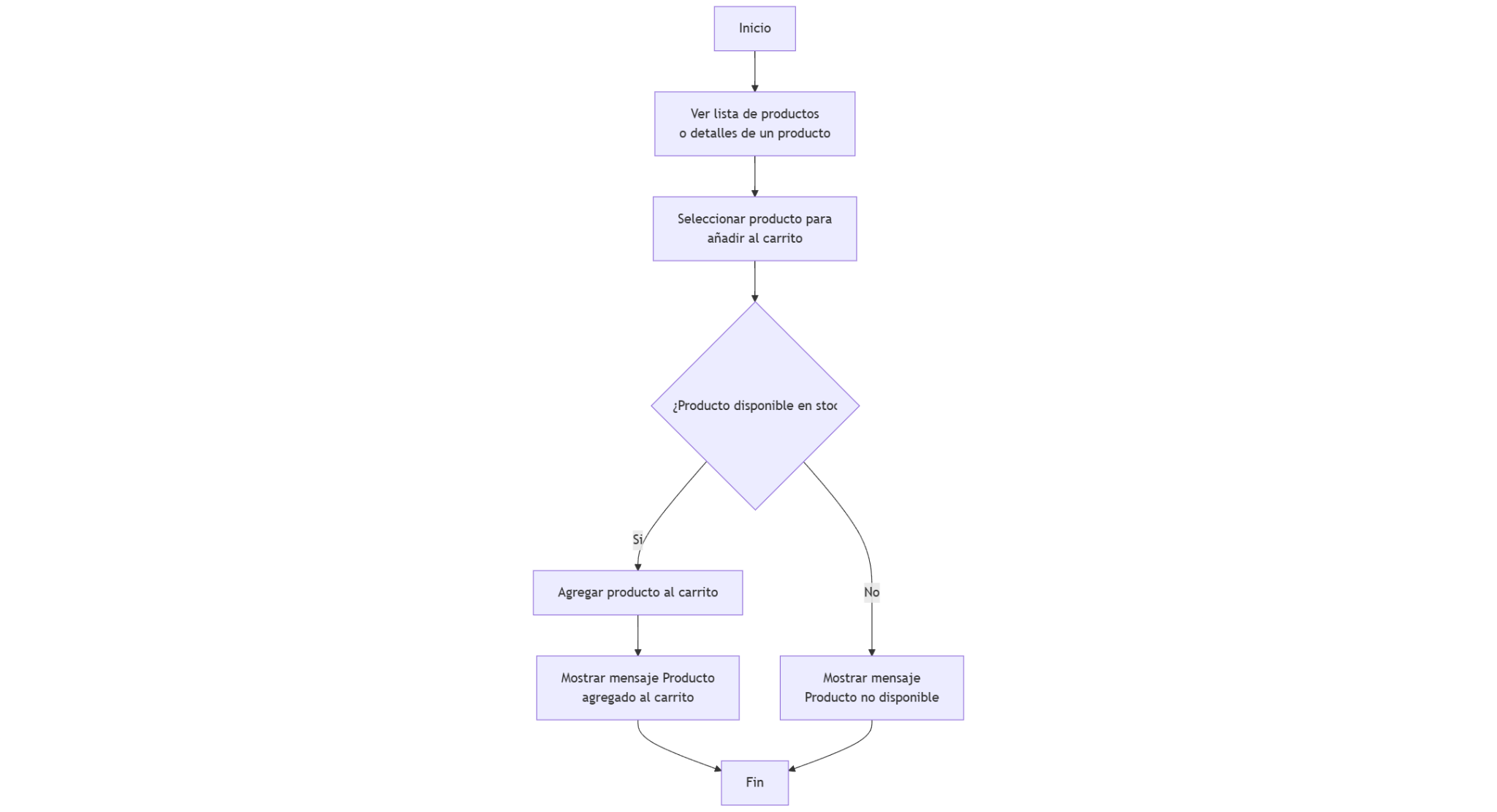
* + 1. **RF10 - Añadir a favoritos**

****

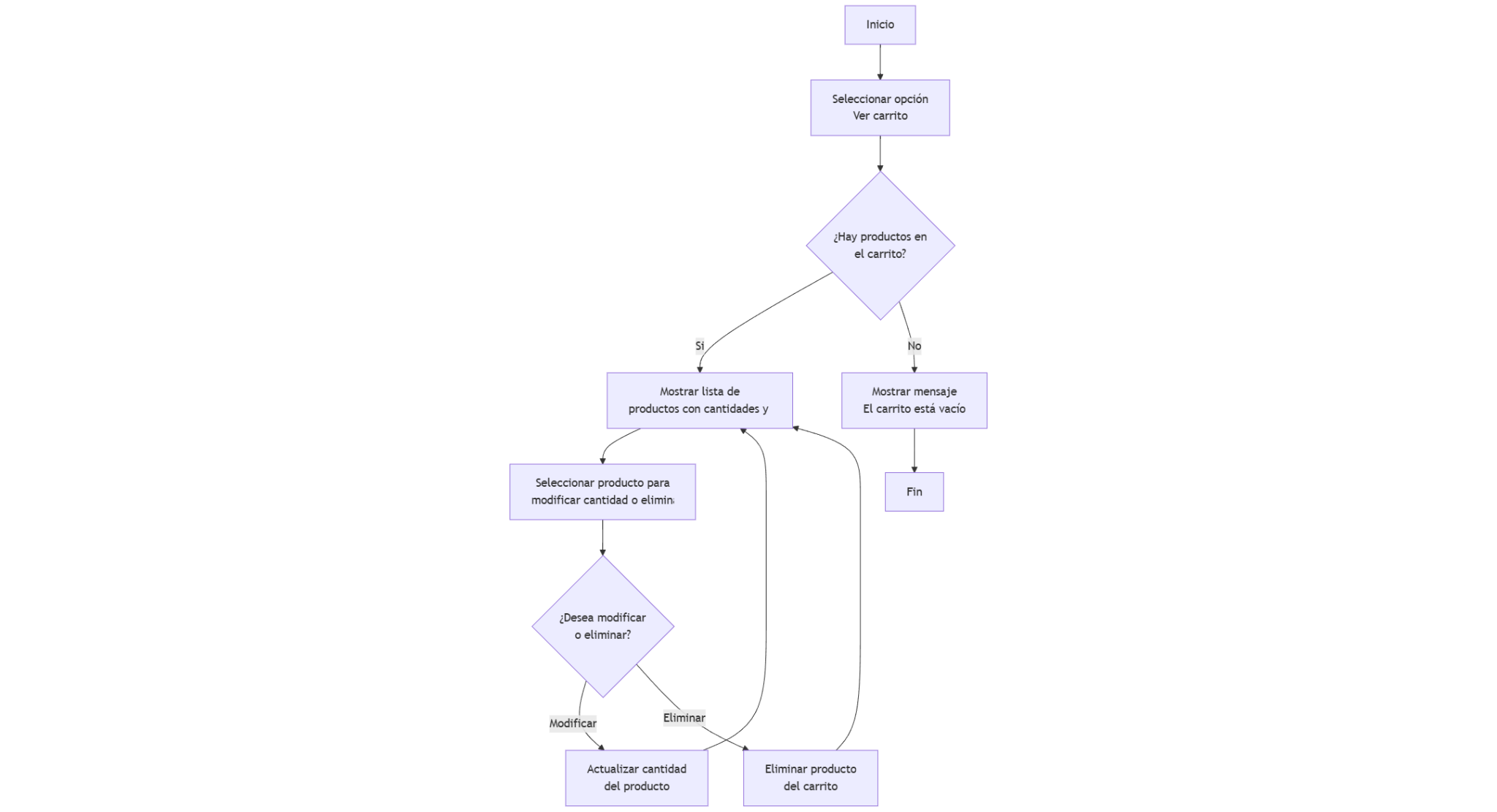
* + 1. **RF12 - Ver favoritos**

****

* + 1. **RF14 - Añadir al carrito**

****

* + 1. **RF17 - Ver el carrito**



# Aplicación Práctica de Pruebas

## Selección de Requerimientos a Probar: Se priorizan los RF relacionados con login, registro, carrito y compra.

## Diseño de Casos de Prueba: Se definen entradas, salidas esperadas y criterios de aceptación por caso.

## Ejecución de Casos de Prueba: Se realizan las pruebas en entorno controlado, documentando evidencias.

## Resultados y Evidencias: Se anexan capturas de pantalla y resultados de cada ejecución.

# Conclusiones