5.3 Propósito del Punto de Vista de Composición

Este punto de vista descompone el sistema en módulos, componentes y subsistemas, definiendo sus responsabilidades y relaciones. Su objetivo es:

Facilitar y permitir el desarrollo paralelo (frontend/backend).

Identificar oportunidades de reutilización (ej: librerías compartidas).

Establecer contratos de interfaz entre componentes.

Facilitar el mantenimiento y comprensión del sistema

Audiencia principal:

Arquitectos de software: Para evaluar cohesión y acoplamiento.Esto ayudará para futuras mejoras en el producto ya que tendrían en esencia el comportamiento de todo el producto. (manteneabilidad, testeabilidad y escabilidad).

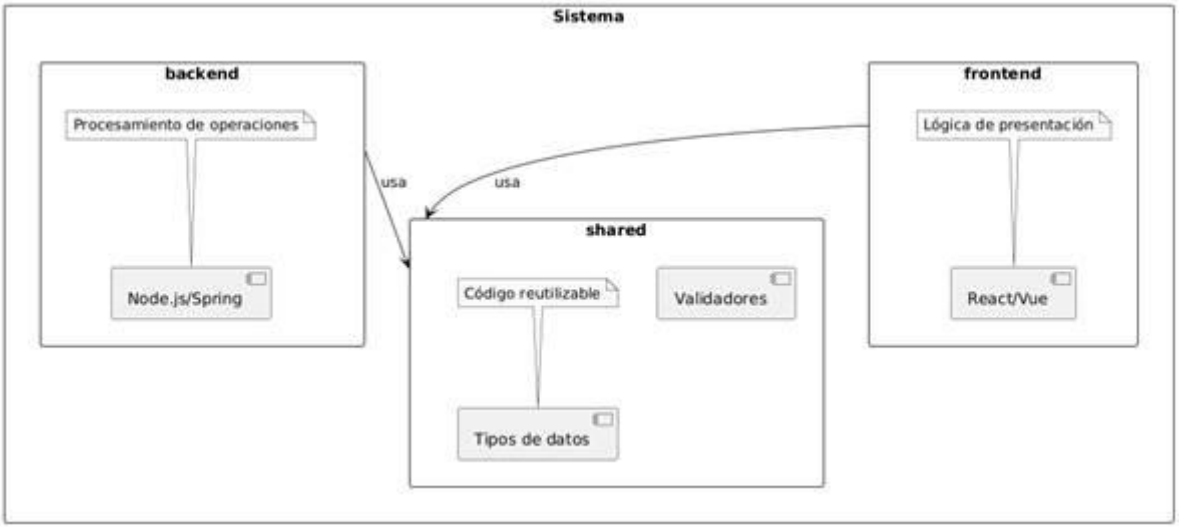
Líderes de equipo: Para asignar módulos a desarrolladores: para distribuir tareas y responsabilidades. Así poner un cronograma y partes correspondientes a cada equipo de ser necesario por el alcance del proyecto de ser necesario contratar un tercero para partes necesarias del producto.

Desarrolladores: Esto para que ellos tengan presentes el alcance de cada módulo y así mejorando el entendimiento y límites de cada equipo aliviando cargas de trabajo correspondiente.

Equipo de testers: Poner a prueba los módulos correspondientes para así diseñar las pruebas correspondientes de cada módulo ya sea por sección o todo en conjunto

Estructura Modular del Sistema

Diagrama de Paquetes (UML)



Descripción:

frontend: Contiene lógica de presentación. La principal encargada de mostrar errores ante el usuario y la estructura interna del producto también mostrará las entradas correspondientes al usuario, el uso de tecnologías responsables serán HTML y JavaScript

backend: Procesamiento de operaciones. Estas serán las operaciones básicas y la conexión con el servidor para las operaciones avanzadas es la interacción directa con el servidor.

share: Código reutilizable (ej: validadores, tipos de datos) este utilizara y tendrá contacto tanto con el front como con el back.

Componentes Principales

Frontend

Interfaz de Usuario: Encargada de representar visualmente la calculadora y sus botones. Contiene toda la interacción entre el usuario y la computadora. Se encargará de mostrar guías o mensajes sobre los botones desconocidos para el usuario ya sea en cálculos avanzados o básicos.

Gestor de Eventos: Captura las interacciones del usuario ya sea registro de números con el click o el teclado y lanza las acciones correspondientes. Muestra tanto errores como opciones para que el usuario pueda comprender más fácil el entorno.

Almacenamiento Local: Guarda historial de operaciones cuando no hay conexión con un límite de guardado del historial. Esto será con un aprox de 25 operaciones realizadas, ya con una conexión a internet se tratará de ajustar el límite unas 50 operaciones aprox.

Servicio de comunicación API: Encapsular la lógica para realizar llamadas HTTP/S al backend (ej: enviar operaciones para cálculo, solicitar historial). Manejar la serialización/deserialización de datos.

Backend

API REST: Expone endpoints (cualquier dispositivo que se conecta a una red para acceder a información o recursos) para enviar y recibir datos de operaciones esto conlleva la validación de datos ya sean de datos o escrito.

Servicio de Cálculo: Procesa operaciones avanzadas utilizando librerías matemáticas. Ya sea para cálculos avanzados o básicos. Esto conlleva a utilizar y aplicar la razón y lógica de las operaciones con esto hace el llamado,adonde a las librerías necesarias para cumplir con lo solicitado.

Middleware de Seguridad: Asegura la validación de solicitudes, como autenticación para historial o las solicitudes de computadora servidor evitando así ataques comunes hacia el producto o datos del servidor.

Capa de persistencia: interactúa con la base de datos para el historial ya sea de manera local o con el servidor esto también guarda sugerencias de operaciones anteriores

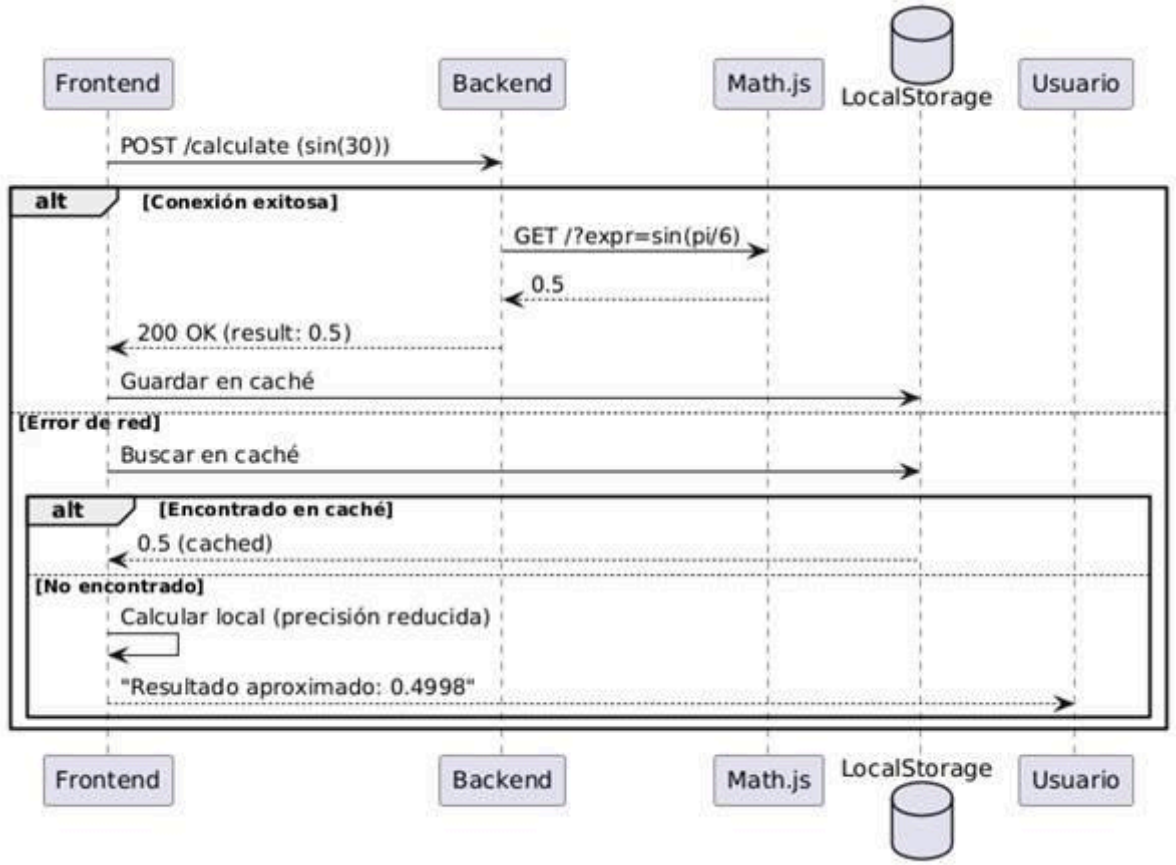
Frontend

Componente	Responsabilidad	Tecnología	Dependencias
Calculadora UI	Renderizar botones/pantalla.	React	Gestor Eventos
Gestor Eventos	Manejar clics/teclas → disparar cálculos.	JavaScript	Servicio API (backend)
LocalStorage	Cachear resultados offline.	Index	-

Backend

Componente	Responsabilidad	Protocolo
API REST	Exponer endpoints (POST /calculate).	Express.js
ServicioCalculo	Delegar operaciones a Math.js.	HTTP/HTTPS
AuthMiddleware	Validar tokens JWT para historial.	-

Diagrama



Reutilización de Componentes

El reuso de componentes para una mejor optimización de procesos y una mejora para el usuario, referente a que ayuda a la mantenibilidad, ayudando también al desarrollador en futuras entregas o correcto de errores:

Librerías matemáticas (Math.js): Utilizada tanto en el frontend (para validaciones rápidas esto en cuestión de operaciones básicas)como en el backend (esto para operaciones avanzadas esto como unico metodo de lógica verdadera para los resultados) para mantener la consistencia en los resultados. Al ya estar previamente cargados su utilidad es recursiva ayuda a la optimización de procesos.

Funciones de validación: Aplicadas para asegurar que los inputs del usuario sean válidos antes de ejecutar operaciones esto delimitará los errores o los reduce en un gran porcentaje. El uso de estas es inmediato en el front y en el back somo una segunda capa de seguridad y validación antes de su procesamiento. Esta reutilización mejora la mantenibilidad y reduce la duplicación de lógica.

Modelos de Datos/Tipos (del paquete share):

Ejemplos: OperationPayload { operand1: number, operand2: number, operator: string }, HistoryEntry { expression: string, result: number, timestamp: Date }.

Usados por front y back para asegurar consistencia en la estructura de datos intercambiada.

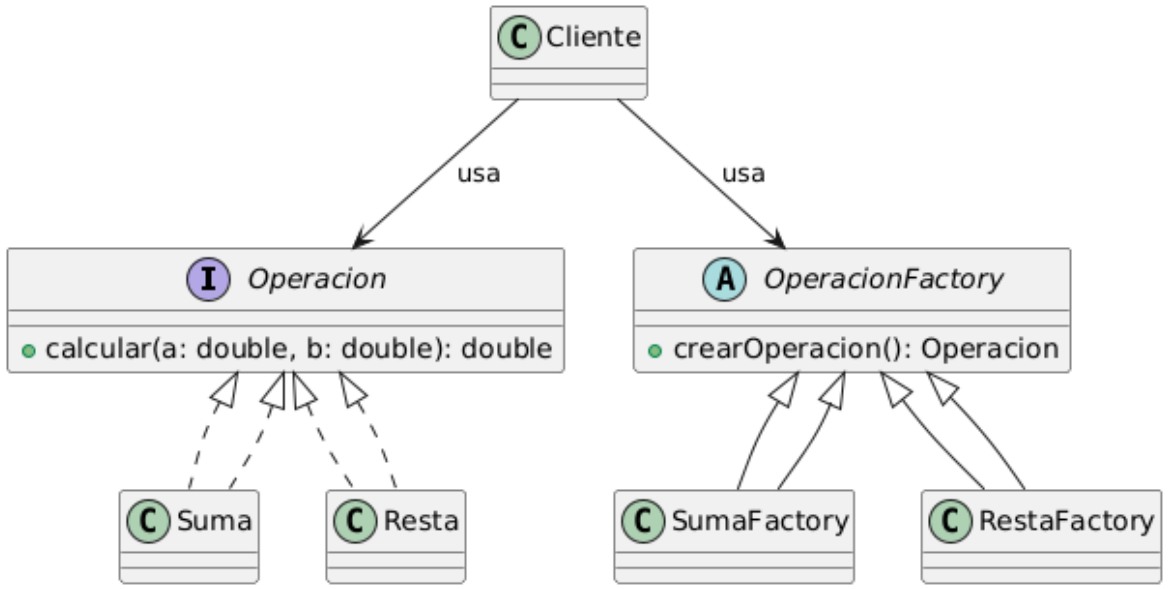
Librerías Compartidas

- Math.js: Usada en frontend (operaciones básicas y validaciones) y backend (avanzadas y como aseveración lógica). Configuración y utilidad a través de javascript y HTML
- De ser necesario añadir para futuras entregas más librerías encargadas de diferentes herramientas un ejemplo si el producto a lo largo de su realización necesita la hora o como futura referencia de problemas relacionados con el horario se añadirá la librería necesaria.

Patrones de Diseño Aplicados

Factory Method: Para instanciar operaciones (patrón de diseño creacional que define una interfaz para crear objetos, pero permite a las subclases decidir qué clase concretas se crearán.)

java

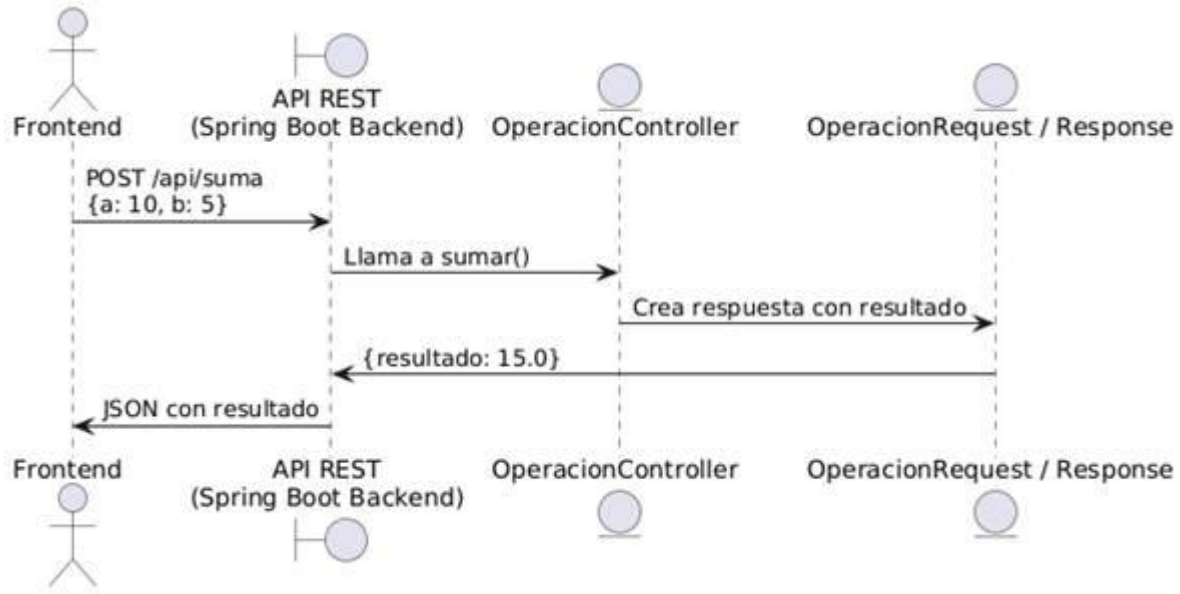


Interfaces Críticas

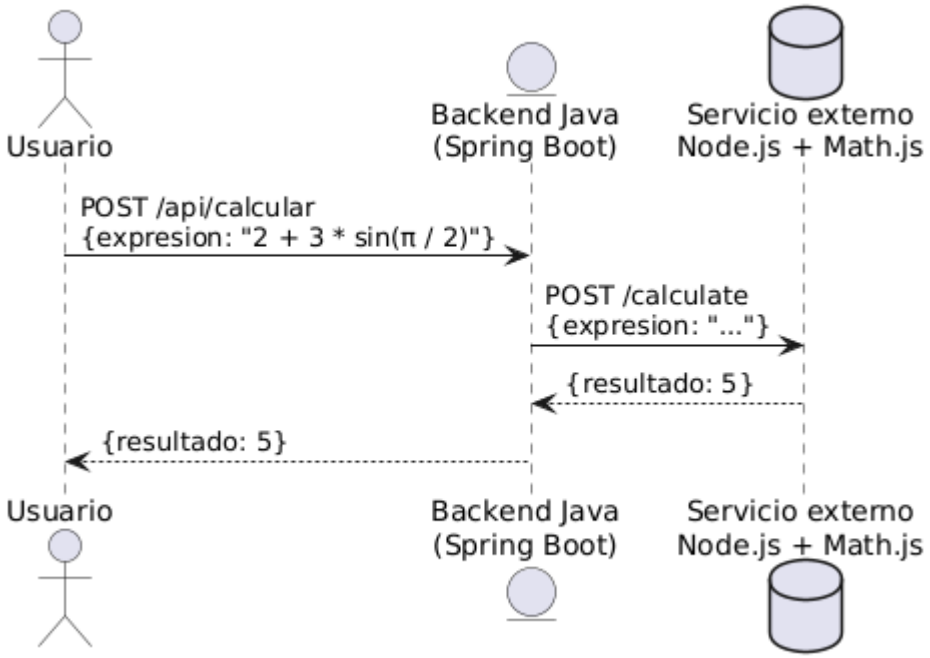
Las interfaces clave que conectan los componentes son:

Frontend ↔ Backend: A través de una API REST (Es una forma común de que los sistemas informáticos se comuniquen entre sí a través de internet, especialmente en arquitecturas cliente-servidor y de microservicios.), usando JSON. Se define una conexión clara para

cada operación matemática.



Backend ↔ Librería Math.js o servicios externos: Comunicación directa para ejecutar cálculos complejos con precisión. Reutilizada junto con otras librerías en el servidor.



Frontend ↔ Backend

Contrato API REST:

Método	Endpoint	Cuerpo de solicitud	Resultado escrito
POST	/api/calcutate	{"op": "sqrt", "val": 16}	{"result": 4}
GET	/api/history	Headers: Authorization: Bearer <JWT>	["sqrt(16)", "sin(30)"]

Manejo de error de acuerdo al error y a los códigos de HTML se proporcionarán diferentes códigos para saber mejor el error y su correspondientes factores y forma de solución tanto para el usuario como para el desarrollador.

Despliegue Físico

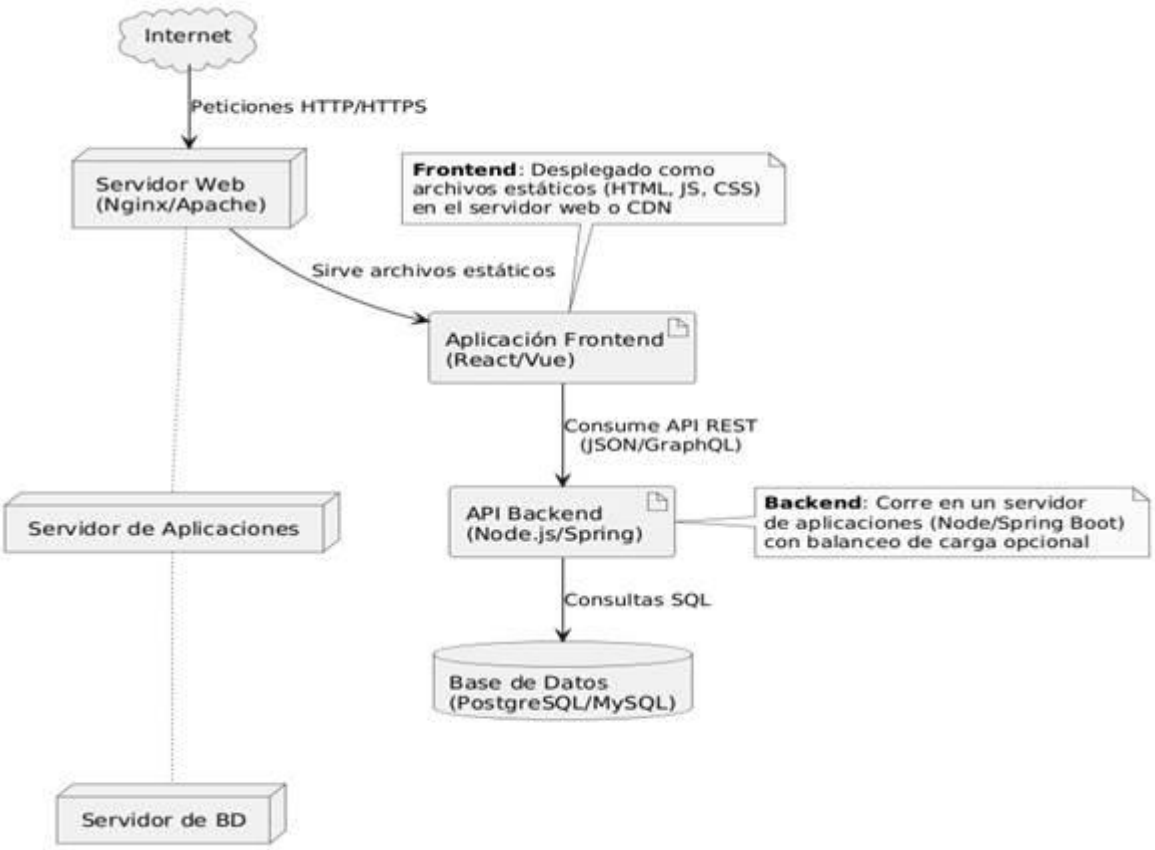
Requisitos por componente:

Frontend: Ligero, debe correr en cualquier navegador moderno. Al igual que en cualquier versión disponible que sea compatible ya sea en dispositivo móvil o de escritorio y que soporte la infraestructura del producto ya sea servicios de JavaScript o HTML

Backend: Debe poder escalar horizontalmente si aumenta el número de solicitudes. No tener errores de lógica o de conexión con el servidor evitando falsas respuestas y desconexiones de respuesta del servidor, interacción fluida con el servidor para evitar saturación de múltiples solicitudes de los usuarios.

Persistencia: Aplica almacenamiento local para modo offline y base de datos externa para modo conectado. En cuestión de almacenamiento local sin línea de red solo se almacenarán 25 operaciones.

Diagrama de Despliegue



Requisitos de Infraestructura

Componente	CPU	Memoria	Almacenamiento
Frontend	1 vCPU	512 MB	1 GB (CDN)
Backend	2 vCPUs	4 GB	10 GB (Docker)

Si el usuario va a ser una variedad pesada de operaciones se recomienda tener una red de banda ancha para la velocidad de procesamiento sin sufrir una desconexión del servidor por respuesta tardía.

Estrategia de Pruebas por Componente

Cada componente será probado de forma independiente:

Frontend: Pruebas visuales y de interacción (UI). Al igual sus dimensiones para que todo sea visible para el usuario sin necesidad de modificar la vista desde el zoom del navegador así proporcionando una vista general de todas las funciones principales.

De ser necesario realizar las pruebas necesarias para la lógica y las bibliotecas, retroceso y acciones del usuario.

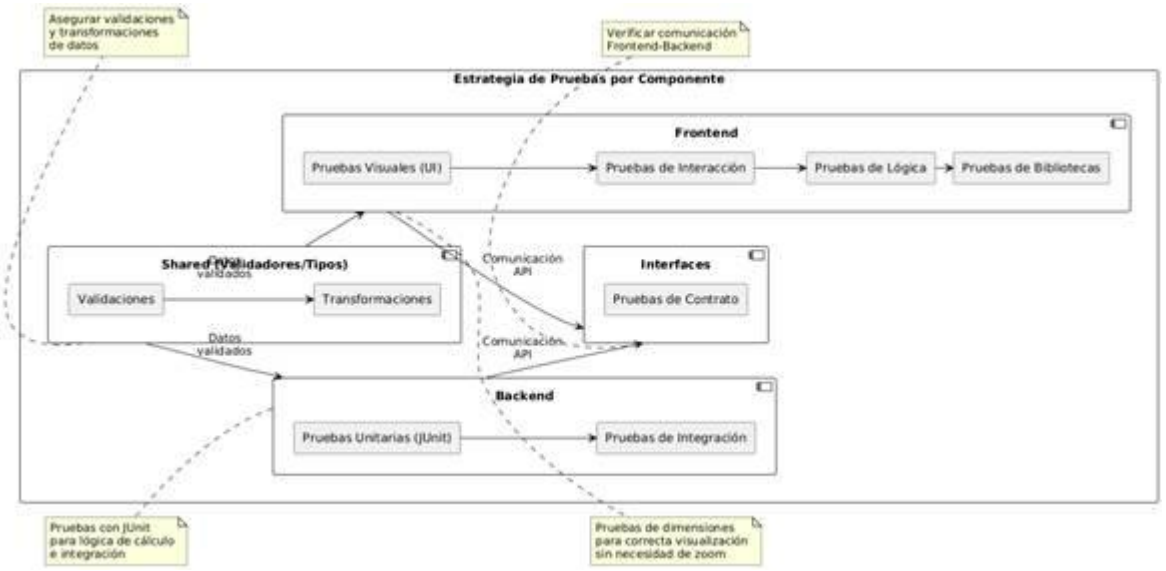
Backend: Pruebas unitarias e integración para lógica de cálculo. Las pruebas se realizarán con la integración de otros métodos y controladores pruebas realizadas con JUnit.

Share Validadores/Tipos: Pruebas unitarias para asegurar que las validaciones y transformaciones de datos funcionan como se espera.

Interfaces: Pruebas de contrato para asegurar que el frontend y backend se comuniquen correctamente.

Componente	Tipo de Prueba	Herramienta	Criterio de Éxito
CalculadoraUI	Pruebas de snapshot	JUnit	Renderizado coherente con diseño.
ServicioCalculo	Pruebas de integración	Mocha/Chai/JUnit	100% cobertura de operaciones.

Diagrama Pruebas Por Componentes



Evolución Arquitectónica

Las versiones que contienen después del primer lanzamiento del producto se darán para dar paso a un mejoramiento, esto se tomará de la siguiente forma.

Futuro posible:

WebAssembly: Para cálculos intensivos en frontend. Esto ayudará a reducir la carga del back en el procesamiento de los datos posiblemente se agregaran una parte de las funciones avanzadas sin necesidad del servidor.

GraphQL: Unificar endpoints de historial/operaciones. Mejora la navegación en el historial del usuario así dándole datos más específicos de ser necesarios modificando el procedimiento o mirarlo directamente para consultarlo.

Mini procesos: Si la tarea del cálculo es muy pesada podrían dividirse los cálculos o hacerlos en diferentes momentos a petición del usuario para que el proceso sea el deseado.

Diagrama Evaluacion de Arquitectura