

5.8 Propósito del Punto de Vista de Interfaces

Este punto de vista especifica cómo interactúan los componentes del sistema, tanto internamente como con sistemas externos, detallando los contratos de comunicación, APIs, entradas/salidas, validaciones y protocolos utilizados, cubriendo:

- APIs públicas y privadas
- Contratos de UI/UX
- Protocolos de comunicación
- Esquemas de validación

Objetivos clave:

1. Garantizar interoperabilidad entre módulos
2. Documentar expectativas de consumo
3. Establecer estándares de versionado

Taxonomía de Interfaces

Interfaces de Sistema

Tipo	Tecnología	Ejemplo	Responsable
REST API	Express.js (Node)	POST /api/calculate	Backend Team
WebSocket	Socket.IO	Canal calc-updates	Fullstack
GraphQL	Apollo Server	Query {history(userId: "123")}	Frontend

Interfaces de Usuario

Componente	Framework	Contrato

---

Keypad	React	Props: onPress(key: string) => void
--------	-------	-------------------------------------

---

History Panel	Vue	Event: @restore-operation
---------------	-----	---------------------------

---

## Especificación Detallada de APIs

Se definen los contratos para los principales servicios ofrecidos por el sistema:

API de cálculo:

Método: POST

Entrada: Tipo de operación, valores numéricos

Salida: Resultado numérico, metadatos

API de historial:

Método: GET

Entrada: Token de autenticación (si aplica)

Salida: Lista de operaciones realizadas

Los contratos están definidos en un formato estructurado y están sujetos a versionado para mantener compatibilidad.

API REST Principal

Endpoint: POST /api/v1/calculate

yaml

openapi: 3.0.0

paths:

/calculate:

post:

requestBody:

content:

application/json:

schema:

type: object

properties:

operation:

type: string

enum: [sum, subtract, sin, cos]

values:

type: array

items:

type: number

minItems: 1

responses:

'200':

description: Resultado del cálculo

content:

application/json:

schema:

\$ref: '#/components/schemas/CalculationResult'

Esquema de Respuesta:

json

```
{
  "success": true,
  "data": {
    "result": 4.0,
    "timestamp": "2024-06-18T10:30:00Z",
    "engine": "mathjs-v11"
  }
}
```

## API WebSocket

### Eventos Soportados:

Evento	Payload	Descripción
calculation	{expr: "2+2", id: "abc123"}	Nueva operación solicitada
result	{id: "abc123", value: 4}	Resultado disponible

### Ejemplo en JavaScript:

javascript

```
const socket = io('https://api.calculator.com');  
socket.emit('calculation', {expr: 'sqrt(16)'});  
socket.on('result', (data) => {  
  console.log(`Result: ${data.value}`);  
});
```

### Contratos de UI/UX

Para asegurar consistencia entre diseño y desarrollo, se establecen contratos claros en los componentes de interfaz:

Propiedades requeridas (ej. valores numéricos, callbacks de eventos).

Estados visuales esperados: normal, error, cargando.

Reglas de accesibilidad: navegación por teclado, etiquetas ARIA, contraste de colores.

El diseño se basa en principios de usabilidad, simplicidad e internacionalización (multilenguaje si aplica).

### Componente de Pantalla

PropTypes (React):

typescript

Copy

Download

```
interface DisplayProps {
```

```

value: string;


fontSize?: 'normal' | 'large';

theme?: 'light' | 'dark';

onError?: (message: string) => void;
}

```

Estados Obligatorios:

Estado	Visual	Disparadores
Normal	Texto negro sobre fondo blanco	-
Error	Texto rojo, icono 	Expresión inválida
Loading	Spinner + texto deshabilitado	Llamada a API en progreso

Guía de Teclado Virtual

Diagram

Keypad

7 | 8 | 9 | +

4 | 5 | 6 | -

1 | 2 | 3 | \*

0 | . | = | /

Reglas de Accesibilidad:

Focus visible en teclas (CSS :focus-visible)

Soporte para navegación por tab (tabindex="0")

ARIA labels para operaciones:

html

Run

```
<button aria-label="Sumar">+</button>
```

## Protocolos de Comunicación

REST (JSON): Usado entre frontend y backend por su simplicidad y soporte universal.

WebSocket (opcional): Para futuras mejoras como resultados en tiempo real.

gRPC (propuesto): Considerado para comunicación interna entre microservicios en futuras versiones.

Cada protocolo es elegido según criterios de latencia, compatibilidad y facilidad de integración.

### REST vs gRPC

Parámetro	REST (JSON)	gRPC (Protocol Buffers)
Latencia	100-300ms	50-150ms
Tamaño Payload	1KB (promedio)	300 bytes
Soporte Browser	Nativo	Requiere gRPC-Web

### Ejemplo gRPC:

```
proto
```

```
service Calculator {
```

```
  rpc Calculate (OperationRequest) returns (OperationResponse);
```

```
}
```

```
message OperationRequest {
```

```
  string operation = 1;
```

```
  repeated double values = 2;
```

```
}
```

## Estrategía de Caché

http

GET /api/history

Cache-Control: max-age=60, stale-while-revalidate=30

Flujo de Validación:

1. Devuelve caché si edad < 60s
2. Usa caché obsoleta mientras revalida (hasta 90s)
3. Actualiza en background

## Validación y Seguridad

Las interfaces incorporan validación tanto del lado del cliente como del servidor:

Validación de tipos, rangos y expresiones permitidas.

Sanitización de entradas para evitar inyecciones o errores inesperados.

Gestión de errores amigable para el usuario (por ejemplo: "División entre cero no permitida").

Adicionalmente, las APIs incluyen medidas de seguridad como:

Tokens de acceso (JWT): Para proteger el historial del usuario.

Cabeceras de seguridad: Para prevenir ataques comunes (XSS, CSRF).

Cifrado en tránsito (HTTPS).

Esquema de Validación (JSON Schema)

json

```
{  
  "$schema": "http://json-schema.org/draft-07/schema#",  
  "type": "object",  
  "properties": {  
    "operation": {  
      "type": "string",
```

```

    "pattern": "[a-zA-Z]+$"
  },
  "values": {
    "type": "array",
    "items": {
      "type": "number",
      "minimum": -1e6,
      "maximum": 1e6
    }
  }
}

```

## OWASP Top 10 Mitigaciones

Riesgo	Protección Implementada
Inyección	Sanitización con <code>mathjs.sanitize()</code>
XSS	CSP: <code>default-src 'self'</code>
CSRF	Tokens synchronizer en formularios

## Versionado y Evolución

### Estrategia de Versionado

Para mantener compatibilidad entre versiones, se adopta una estrategia de versionado:

URI Versioning: `/api/v1/calculate`

Header Versioning: `Accept: application/vnd.calculator.v1+json`

Política de Depreciación:



1. Versión marcada como obsoleta por 6 meses
2. Notificación a clientes vía Deprecation header
3. Remoción después de 12 meses

Documentación Interactiva. Se utiliza una herramienta de documentación interactiva (como Swagger UI o Redoc) que permite:

Visualizar los endpoints disponibles. Probar solicitudes directamente desde la interfaz.  
Descargar especificaciones para su uso en pruebas automatizadas.

Esto mejora la colaboración entre equipos y reduce errores de integración.

Swagger UI

yaml

```
# swagger-config.yml
```

```
urls:
```

```
- url: '/api/v1/swagger.json'
```

```
  name: 'Calculator API v1'
```

Features clave:

Try-it-out para endpoints

Ejemplos pre-cargados

Esquemas descargables

Pruebas de Interfaces

Las interfaces se someten a pruebas para validar su correcto funcionamiento:

Pruebas de contrato (API): Verifican que las respuestas cumplen con el esquema definido.

Pruebas de interfaz gráfica (UI): Validan el correcto renderizado, interacción y comportamiento esperado de los componentes.

Pruebas de accesibilidad: Evaluación de navegación, lectura por lectores de pantalla y cumplimiento de normas WCAG.

Estas pruebas son clave para asegurar calidad, accesibilidad y compatibilidad multiplataforma.

## Pruebas de Contrato (Pact)

javascript

// Consumer test

```
await provider.addInteraction({
  state: 'servicio operativo',
  uponReceiving: 'petición de suma',
  willRespondWith: {
    status: 200,
    body: {
      result: 3
    }
  }
});
```

## Pruebas de UI (Cypress)

javascript

```
describe('Calculator UI', () => {
  it('debe sumar 2+2 correctamente', () => {
    cy.get('[data-testid="btn-2"]').click();
    cy.get('[data-testid="btn-plus"]').click();
    cy.get('[data-testid="btn-2"]').click();
    cy.get('[data-testid="display"]').should('contain', '4');
  });
});
```

## Monitorización

Durante la operación del sistema, se monitorean métricas clave relacionadas con las interfaces:

Tasa de errores por endpoint (por ejemplo, errores 400 o 500).

Tiempo de respuesta medio.

Uso por tipo de operación o componente.

Esto permite detectar problemas de integración o rendimiento de forma proactiva.

#### Métricas Clave

Métrica	Umbral	Acción
Tasa error API	> 1% (5min)	Alertar equipo
Latencia p95	> 500ms	Escalar instancias

#### Dashboard Ejemplo

json

```
{
  "widgets": [
    {
      "type": "timeseries",
      "title": "Llamadas API",
      "metrics": ["api.request.count"],
      "region": "us-east-1"
    }
  ]
}
```