5.6 Propósito del Punto de Vista de Información

Este punto de vista se centra en el diseño, organización y gestión de la información que fluye a través del sistema, así como en los datos que se almacenan o procesan, abarcando:

Modelado de estructuras de datos (debe ser coherente y eficiente)

Flujos de información críticos (el acceso a la información será rápido y seguro)

Estrategias de persistencia

Políticas de integridad y seguridad

Objetivos clave:

1. Garantizar consistencia en el manejo de datos
2. Optimizar acceso a información frecuente
3. Cumplir regulaciones (GDPR, HIPAA si aplica)

Modelo de Datos Principal

El sistema maneja distintos tipos de información estructurada. Entre las entidades clave se encuentran:

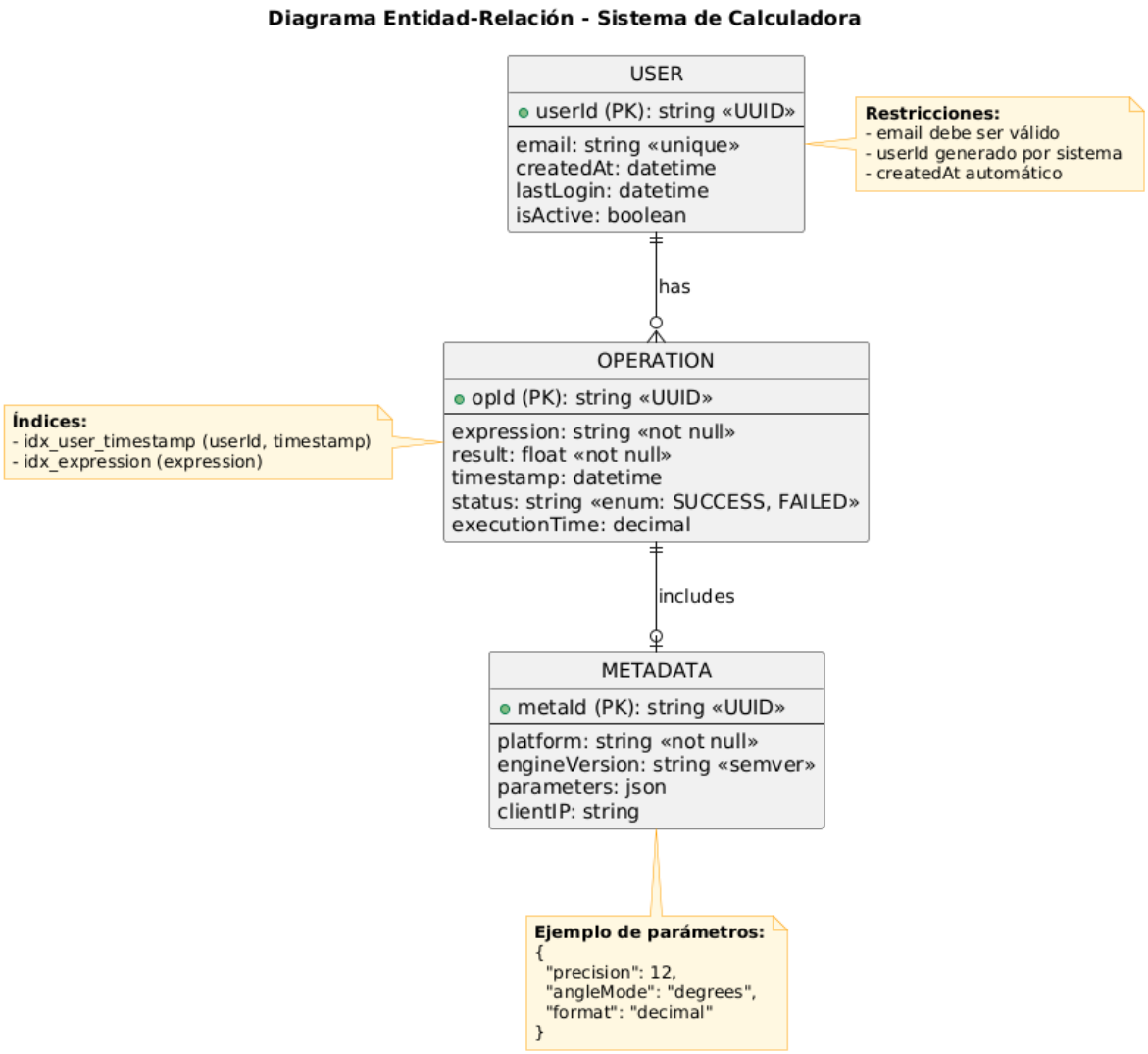
Usuario (opcional): Identificador anónimo o token local, útil para sesiones prolongadas o historial personalizado.

Operación: Representa cada cálculo realizado, incluyendo los operandos, el tipo de operación y el resultado.

Historial: Colección ordenada de operaciones asociadas a una sesión o usuario.

Metadatos: Información complementaria como fecha, navegador utilizado o versión de librería matemática.

Cada entidad se diseña con atributos bien definidos y restricciones de validación para evitar inconsistencias.



Diccionario de Datos

Entidad	Atributo	Tipo	Descripción	Restricciones
USER	userId	UUID	Identificador único	PRIMARY KEY
OPERATION	expression	String(255)	Expresión matemática	No nulo

METADATA	engineVersion	SemVer	Versión Math.js usada	Formato X.Y.Z

Estrategias de Persistencia

Se contempla una estrategia híbrida de almacenamiento:

Modo Offline: El historial y los datos recientes se guardan localmente en el navegador mediante mecanismos como localStorage o IndexedDB.

Modo Online: Los datos se sincronizan con una base de datos externa (como Firebase) para respaldo, sincronización entre dispositivos y análisis histórico.

Esta dualidad permite continuidad de uso sin conexión, y robustez cuando hay red disponible.

Almacenamiento Primario

Capa	Tecnología	Esquema	Volumen Estimado
Hot Data	Firebase Firestore	Documentos JSON	10K ops/día
Cold Data	AWS S3 + Glacier	Parquet	1TB/año

Modelo Híbrido Online/Offline

typescript

```
// Ejemplo estrategia caché
class DataManager {
  async getOperation(id: string) {
    const cached = localStorage.getItem(`op_${id}`);
    if (cached) return JSON.parse(cached);

    const remote = await firestore.doc(`ops/${id}`).get();
    localStorage.setItem(`op_${id}`, JSON.stringify(remote));
    return remote;
```

}

}

Flujos de Información Críticos

Los principales flujos de datos incluyen:

Ingreso de datos: Captura de expresión matemática ingresada por el usuario

Procesamiento: Evaluación del cálculo en frontend o backend.

Almacenamiento del resultado: Registro en historial si se permite o solicita.

Consulta del historial: Visualización y filtrado de cálculos previos.

Estos flujos están diseñados para minimizar latencia y asegurar consistencia entre componentes.

Procesamiento de Cálculos

plantuml

@startuml

participant "Frontend" as FE

participant "Backend" as BE

database "Math.js" as MATH

database "Firestore" as DB

FE -> BE : POST {"expr":"2+2"}

BE -> MATH : GET /?expr=2+2

MATH --> BE : 4

BE -> DB : INSERT {result:4}

DB --> BE : OK

BE --> FE : 200 OK {result:4}

@enduml

Sincronización Offline

Estado	Trigger	Acción	Conflict Resolution
Online	API Response	Cachear en IndexedDB	Timestamp más reciente
Offline	User Action	Queue en PouchDB	Merge manual

Esquemas de Bases de Datos

Se contemplan dos enfoques según el entorno:

Relacional (SQL): Adecuado para estructuras con relaciones claras, como usuarios y operaciones, útil para auditoría o administración.

NoSQL (JSON): Más flexible para datos de sesión en entornos web, especialmente con Firebase.

Ambos modelos incluyen medidas para asegurar integridad (ej. claves primarias, validaciones de tipo) y escalabilidad.

SQL (Historial)

sql

```
CREATE TABLE users (
  user_id VARCHAR(36) PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE operations (
  op_id VARCHAR(36) PRIMARY KEY,
  user_id VARCHAR(36) REFERENCES users(user_id),
  expression TEXT NOT NULL,
  result DOUBLE PRECISION,
```

```
  timestamp TIMESTAMP
);
```

NoSQL (Sesiones)

```
json
{
 "sessionId": "abc123",
 "calculations": [
  {
   "input": "sqrt(16)",
   "output": 4,
   "device": "mobile"
  }
 ],
 "ttl": 86400
}
```

Políticas de Datos

Para proteger y optimizar el manejo de información se aplican políticas como:

Retención: El historial se conserva por un periodo definido (ej. 30 días en local, 6 meses en la nube).

Encriptación: Los datos sensibles (si se almacenan usuarios) deben cifrarse tanto en tránsito como en reposo.

Minimización: Solo se guarda la información estrictamente necesaria para el funcionamiento del sistema.

Estas políticas pueden ajustarse en función de requisitos legales, técnicos o del usuario.

Retención

Tipo Dato	Periodo	Método Eliminación
Historial	2 años	Batch job mensual
Logs	30 días	Rotación automática

Encriptación

Campo	Método	Clave
email	AES-256	KMS AWS
userId	Hash	SHA-512

Modelado de Consultas Frecuentes

El sistema puede implementar funciones analíticas básicas como:

Mostrar operaciones más frecuentes realizadas por el usuario.

Filtrar el historial por tipo de operación o fecha.

Agrupar cálculos similares para detección de patrones de uso.

Estas funcionalidades mejoran la experiencia de usuario y permiten evolución futura del sistema con base en datos reales.

Top Operaciones

sql

```
-- PostgreSQL
SELECT expression, COUNT(*) as count
FROM operations
```

WHERE timestamp > NOW() - INTERVAL '7 days'

GROUP BY expression

ORDER BY count DESC

LIMIT 10;

Patrones de Uso

javascript

// Firebase Analytics

firebase.analytics().logEvent('common_operations', {

  operations: ['sin', 'cos', 'log']

});

Migraciones de Datos

A medida que el sistema crece, se prevé:

Versionamiento de esquemas: Para permitir nuevas estructuras de datos sin perder compatibilidad.

Scripts de migración: Para actualizar información antigua a formatos más recientes.

Auditoría de cambios: Registro de quién modificó qué dato, cuándo y desde dónde (si aplica autenticación).

Estas acciones aseguran la continuidad del servicio sin pérdida de datos ni ruptura de funcionalidades.

Estrategia Blue-Green

1. Nueva versión con schema v2
2. Dual-write a v1 y v2
3. Migración progresiva

Script de Migración

python

Ejemplo Pandas

def migrate_operations(source, target):

  df = pd.read_sql("SELECT * FROM operations", source)

  df['engine_version'] = '1.0.0'

```
df.to_sql('operations_v2', target, index=False)
```

Auditoría y Trazabilidad

Si se implementan usuarios o perfiles, debe garantizarse:

Registro estructurado de todas las operaciones relevantes.

Posibilidad de rastrear errores o accesos indebidos.

Control de acceso diferenciado por rol (en caso de una versión multiusuario).

Esto aumenta la transparencia del sistema y facilita el mantenimiento y cumplimiento regulatorio.

Log Estructurado

json

```
{
  "timestamp": "2024-06-15T10:00:00Z",
  "operation": "data_access",
  "user": "user123",
  "entity": "operations",
  "changes": {
    "old": null,
    "new": {"expr": "2+2"}
  }
}
```

Matriz de Acceso

Rol	Tabla	Permisos
User	operations	CRUD propios
Admin	users	Read-only

Rendimiento y Optimización

Se optimiza el uso de datos mediante:

Indexación en campos críticos (como fecha de operación).

Caché local para cálculos recientes o frecuentes.

Particionamiento de datos en bases grandes para consultas rápidas.

Estas técnicas reducen la carga en el backend y mejoran la respuesta para el usuario final.

Indexación Clave

sql

CREATE INDEX idx_operations_user ON operations(user_id, timestamp)

Particionamiento

sql

-- PostgreSQL

PARTITION BY RANGE (timestamp);