

# APEX Rules Engine REST API - Quick Reference

**Version:** 1.0 **Date:** 2025-08-02 **Author:** Mark Andrew Ray-Smith Cityline Ltd

## Base URL

`http://localhost:8080/api`

## Content Type

All endpoints accept and return JSON:

Content-Type: application/json

## Endpoints Overview

### Transformation API ( /api/transformations )

Method	Endpoint	Description	Request Body
GET	/transformers	Get registered transformers	None
POST	/transformerName	Transform data with registered transformer	Object (data to transform)
POST	/dynamic	Transform with dynamic rules	DynamicTransformationRequest
POST	/transformerName/detailed	Transform with detailed result	Object (data to transform)

### Enrichment API ( /api/enrichment )

Method	Endpoint	Description	Request Body
GET	/configurations	Get predefined configurations	None
POST	/enrich	Enrich object with YAML config	EnrichmentRequest
POST	/batch	Batch enrichment	BatchEnrichmentRequest
POST	/predefined/configName	Enrich with predefined config	Object (target object)

### Template Processing API ( /api/templates )

Method	Endpoint	Description	Request Body
POST	/json	Process JSON template	TemplateProcessingRequest
POST	/xml	Process XML template	TemplateProcessingRequest
POST	/text	Process text template	TemplateProcessingRequest
POST	/batch	Process multiple templates	BatchTemplateProcessingRequest

## Data Source API ( /api/datasources )

Method	Endpoint	Description	Request Body
GET	/	Get all data sources	None
GET	/ {name}	Get specific data source	None
POST	/ {name} /test	Test data source	Map<String, Object> (test parameters)
POST	/ {name} /lookup	Perform lookup	Map<String, Object> (lookup parameters)

## Expression API ( /api/expressions )

Method	Endpoint	Description	Request Body
POST	/evaluate	Evaluate expression	ExpressionEvaluationRequest
POST	/evaluate/detailed	Evaluate with detailed result	ExpressionEvaluationRequest
POST	/batch	Batch expression evaluation	BatchExpressionRequest
POST	/validate	Validate expression syntax	ExpressionValidationRequest
GET	/functions	Get available functions	None

## Rules API ( /api/rules )

Method	Endpoint	Description	Request Body
POST	/check	Evaluate a single rule condition	RuleEvaluationRequest
POST	/validate	Validate data against multiple rules	ValidationRequest
POST	/define/ {name}	Define a named rule for reuse	Map<String, String> (condition, message)
POST	/test/ {name}	Test a previously defined rule	Map<String, Object> (test data)
GET	/defined	Get all defined rules	None
POST	/execute	Execute single rule	RuleExecutionRequest
POST	/batch	Execute batch rules	BatchRuleExecutionRequest

## System API ( /api/version )

Method	Endpoint	Description	Request Body
GET	/info	Get API version information	None
GET	/compatibility	Get compatibility information	None
GET	/deprecation	Get deprecation information	None

## Request/Response Models

### Core Request Models

DynamicTransformationRequest

```
{
  "data": { /* object to transform */ },
  "transformerRules": [
    {
      "name": "rule-name",           // @NotBlank
      "condition": "SpEL expression", // @NotBlank
      "transformation": "SpEL expression", // @NotBlank
      "targetField": "field-name"    // @NotBlank
    }
  ]
}
```

EnrichmentRequest

```
{
  "targetObject": { /* object to enrich */ },
  "yamlConfiguration": "YAML config string"
}
```

BatchEnrichmentRequest

```
{
  "yamlConfiguration": "YAML config string", // @NotNull
  "targetObjects": [ /* array of objects */ ] // @NotNull
}
```

TemplateProcessingRequest

```
{
  "template": "template string with #{expressions}", // @NotBlank
  "context": { /* context variables */ }              // @NotNull
}
```

BatchTemplateProcessingRequest

```

{
  "templates": [
    {
      "name": "template-name",
      "type": "JSON|XML|TEXT",
      "template": "template string"
    }
  ],
  "context": { /* shared context variables */ }
}

```

#### RuleExecutionRequest

```

{
  "rule": {
    "name": "rule-name",          // @NotBlank
    "condition": "SpEL expression", // @NotBlank
    "message": "optional message",
    "priority": "HIGH|MEDIUM|LOW"
  },
  "facts": { /* context data */ } // @NotNull
}

```

#### BatchRuleExecutionRequest

```

{
  "rules": [
    {
      "name": "rule-name",
      "condition": "SpEL expression",
      "message": "optional message"
    }
  ],
  "facts": { /* shared context data */ }
}

```

#### RuleEvaluationRequest

```

{
  "condition": "SpEL expression",    // @NotBlank
  "data": { /* evaluation context */ }, // @NotNull
  "ruleName": "optional-name",
  "message": "optional message",
  "includeMetrics": false
}

```

#### ExpressionEvaluationRequest

```

{
  "expression": "SpEL expression",    // @NotBlank
  "context": { /* variables */ },      // @NotNull
  "includeMetrics": false,
  "validateSyntax": true
}

```

## ValidationRequest

```
{
  "data": { /* data to validate */ }, // @NotNull
  "rules": [
    {
      "name": "validation-rule",
      "condition": "SpEL expression",
      "message": "error message",
      "severity": "ERROR|WARNING|INFO"
    }
  ]
}
```

# Common Request Examples

## Transform Data

```
POST /api/transformations/dynamic
{
  "data": { "firstName": "john", "email": "JOHN@EXAMPLE.COM" },
  "transformerRules": [
    {
      "name": "normalize-name",
      "condition": "#firstName != null",
      "transformation": "#firstName.substring(0,1).toUpperCase() + #firstName.substring(1).toLowerCase()",
      "targetField": "firstName"
    },
    {
      "name": "normalize-email",
      "condition": "#email != null",
      "transformation": "#email.toLowerCase()",
      "targetField": "email"
    }
  ]
}
```

## Enrich Object

```
POST /api/enrichment/enrich
{
  "targetObject": { "customerId": "CUST001", "amount": 1000.0 },
  "yamlConfiguration": "metadata:\n  name: \"Customer Enrichment\"\n  version: \"1.0.0\"\nenrichments:\n  - name: \"custo"
}
```

## Process JSON Template

```
POST /api/templates/json
{
  "template": "{\n  \"customerId\": \"#{@customerId}\",\n  \"customerName\": \"#{@customerName}\",\n  \"totalAmount\": #{
  \"context\": {
    \"customerId\": \"CUST001\",
    \"customerName\": \"John Doe\",

```

```
    "totalAmount": 1500.0,
    "currency": "USD",
    "amount": 1500.0
  }
}
```

## Process XML Template

```
POST /api/templates/xml
{
  "template": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<trade>\n  <tradeId>#{#tradeId}</tradeId>\n  <instrument>#{#in
  \"context\": {
    \"tradeId\": \"TRD001\",
    \"instrumentName\": \"AAPL\",
    \"quantity\": 1500,
    \"price\": 150.25,
    \"currency\": \"USD\"
  }
}
```

## Evaluate Expression

```
POST /api/expressions/evaluate
{
  \"expression\": \"#amount * #rate + #fee\",
  \"context\": { \"amount\": 1000, \"rate\": 0.05, \"fee\": 25 },
  \"includeMetrics\": true
}
```

## Execute Rule

```
POST /api/rules/execute
{
  \"rule\": {
    \"name\": \"high-value-trade\",
    \"condition\": \"#amount > 10000\",
    \"message\": \"High value trade detected\",
    \"priority\": \"HIGH\"
  },
  \"facts\": {
    \"amount\": 15000.0,
    \"currency\": \"USD\",
    \"customerId\": \"CUST001\"
  }
}
```

## Define Named Rule

```
POST /api/rules/define/adult-check
{
  \"condition\": \"#age >= 18\",
  \"message\": \"Customer is an adult\"
}
```

```
}
```

## Validate Data

```
POST /api/rules/validate
{
  "data": { "age": 25, "email": "john@example.com", "balance": 1500 },
  "rules": [
    {
      "name": "age-validation",
      "condition": "#data.age >= 18",
      "message": "Age must be at least 18",
      "severity": "ERROR"
    },
    {
      "name": "email-validation",
      "condition": "#data.email != null && #data.email.contains('@')",
      "message": "Valid email required",
      "severity": "ERROR"
    }
  ]
}
```

## Response Formats

### Standard Success Response

```
{
  "success": true,
  "data": { /* response data */ },
  "timestamp": "2025-08-10T10:30:00Z"
}
```

### Detailed Success Response (with metrics)

```
{
  "success": true,
  "data": { /* response data */ },
  "timestamp": "2025-08-10T10:30:00Z",
  "metrics": {
    "executionTimeMs": 45,
    "memoryUsedBytes": 1024,
    "rulesEvaluated": 3
  }
}
```

### Rule Execution Response

```
{
  "success": true,
  "rule": {
    "name": "rule-name",

```

```

    "condition": "SpEL expression",
    "message": "rule message",
    "priority": "HIGH"
  },
  "result": {
    "triggered": true,
    "ruleName": "rule-name",
    "message": "rule message",
    "resultType": "SUCCESS",
    "timestamp": "2025-08-10T10:30:00Z"
  },
  "timestamp": "2025-08-10T10:30:00Z"
}

```

## Batch Response

```

{
  "success": true,
  "results": [
    { /* individual result 1 */ },
    { /* individual result 2 */ }
  ],
  "summary": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "timestamp": "2025-08-10T10:30:00Z"
}

```

## Error Response

```

{
  "success": false,
  "error": "VALIDATION_ERROR",
  "message": "Detailed error description",
  "details": {
    "field": "condition",
    "rejectedValue": "invalid expression",
    "reason": "SpEL syntax error"
  },
  "correlationId": "abc123-def456",
  "timestamp": "2025-08-10T10:30:00Z"
}

```

## Validation Error Response

```

{
  "success": false,
  "error": "VALIDATION_ERROR",
  "message": "Request validation failed",
  "validationErrors": [
    {
      "field": "rule.condition",
      "message": "Condition cannot be blank",
      "rejectedValue": null
    }
  ]
}

```



```
],  
  "correlationId": "abc123-def456",  
  "timestamp": "2025-08-10T10:30:00Z"  
}
```

## HTTP Status Codes

### Success Codes

- 200 OK - Request successful
- 201 Created - Resource created (e.g., rule defined)

### Client Error Codes

- 400 Bad Request - Invalid request format or validation error
- 404 Not Found - Resource not found (e.g., rule name, transformer)
- 409 Conflict - Resource already exists (e.g., rule name conflict)
- 422 Unprocessable Entity - Valid request format but business logic error

### Server Error Codes

- 500 Internal Server Error - Unexpected server error
- 503 Service Unavailable - Service temporarily unavailable

### Error Categories

- VALIDATION\_ERROR - Request validation failed
- RULE\_EVALUATION\_ERROR - Error evaluating rule condition
- EXPRESSION\_ERROR - SpEL expression syntax or evaluation error
- TRANSFORMATION\_ERROR - Data transformation failed
- ENRICHMENT\_ERROR - Object enrichment failed
- TEMPLATE\_ERROR - Template processing failed
- DATA\_SOURCE\_ERROR - Data source lookup failed
- CONFIGURATION\_ERROR - Configuration parsing error

## SpEL Expression Syntax Reference

### Variables and Context Access

- Use # prefix: #amount , #customer.name , #data.field
- Nested properties: #customer.address.city
- Array/List access: #items[0] , #data['key']
- Map access: #context['customerId']

### Operators

#### Arithmetic

- + Addition: #amount + #fee

- - Subtraction: `#total - #discount`
- \* Multiplication: `#quantity * #price`
- / Division: `#total / #count`
- % Modulo: `#value % 10`
- ^ Power: `#base ^ #exponent`

## Comparison

- == Equal: `#status == 'ACTIVE'`
- != Not equal: `#type != 'INVALID'`
- < Less than: `#age < 18`
- > Greater than: `#amount > 1000`
- <= Less than or equal: `#score <= 100`
- >= Greater than or equal: `#balance >= 0`

## Logical

- && AND: `#age >= 18 && #verified == true`
- || OR: `#type == 'GOLD' || #amount > 10000`
- ! NOT: `!#expired`

## String Operations

- + Concatenation: `#firstName + ' ' + #lastName`
- matches Regex: `#email matches '.*@.*\..*'`

## Ternary Operator

- `condition ? value1 : value2`
- Example: `#amount > 1000 ? 'HIGH' : 'LOW'`

## Built-in Functions

### String Functions

```
#text.length()           // String length
#text.toLowerCase()      // Convert to lowercase
#text.toUpperCase()      // Convert to uppercase
#text.trim()             // Remove whitespace
#text.substring(0, 5)    // Extract substring
#text.contains('substring') // Check if contains
#text.startsWith('prefix') // Check if starts with
#text.endsWith('suffix') // Check if ends with
#text.replace('old', 'new') // Replace text
#text.split(',')         // Split into array
#text.isEmpty()         // Check if empty
```

### Mathematical Functions

```
T(java.lang.Math).abs(#value) // Absolute value
T(java.lang.Math).ceil(#value) // Ceiling
T(java.lang.Math).floor(#value) // Floor
T(java.lang.Math).round(#value) // Round
T(java.lang.Math).max(#a, #b) // Maximum
T(java.lang.Math).min(#a, #b) // Minimum
```

```
T(java.lang.Math).pow(#base, #exp) // Power
T(java.lang.Math).sqrt(#value)    // Square root
```

## Date/Time Functions

```
T(java.time.LocalDate).now()      // Current date
T(java.time.LocalDateTime).now()  // Current date/time
T(java.time.Instant).now()        // Current timestamp
T(java.time.LocalDate).parse('2025-01-15') // Parse date
#date.isAfter(#otherDate)        // Date comparison
#date.isBefore(#otherDate)       // Date comparison
#date.plusDays(7)                // Add days
#date.minusMonths(1)             // Subtract months
```

## Collection Operations

```
#items.size()                    // Collection size
#items.isEmpty()                 // Check if empty
#items.contains(#value)          // Check if contains
#items.[condition]               // Filter collection
#items.[expression]              // Project/transform collection
#items.^[condition]              // First matching element
#items.$[condition]              // Last matching element
#items.[price > 100].![name]     // Filter then project
```

## Type Conversion

```
T(java.lang.Integer).parseInt(#str) // String to int
T(java.lang.Double).parseDouble(#str) // String to double
T(java.lang.Boolean).parseBoolean(#str) // String to boolean
#value.toString()                  // Convert to string
```

## Advanced Examples

### Complex Conditions

```
// Multi-condition validation
#age >= 18 && #email != null && #email.contains('@') && #balance > 0

// Nested object access
#customer.address.country == 'US' && #customer.tier == 'GOLD'

// Collection filtering
#orders.[status == 'PENDING'].size() > 0

// Date range check
#orderDate.isAfter(T(java.time.LocalDate).now().minusDays(30))
```

### Transformation Expressions

```
// Name normalization
#firstName.substring(0,1).toUpperCase() + #firstName.substring(1).toLowerCase()
```

```
// Email normalization
#email.toLowerCase().trim()

// Price calculation
#quantity * #unitPrice * (1 - #discountRate)

// Status determination
#amount > 10000 ? 'HIGH_VALUE' : (#amount > 1000 ? 'MEDIUM_VALUE' : 'LOW_VALUE')
```

## Collection Processing

```
// Sum of order amounts
#orders.![amount].sum()

// High-value orders
#orders.?[amount > 1000]

// Customer names from orders
#orders.![customer.name]

// Average order value
#orders.![amount].sum() / #orders.size()
```

# Batch Operations

## Batch Rules Execution

```
POST /api/rules/batch
{
  "rules": [
    {
      "name": "high-value-check",
      "condition": "#amount > 1000",
      "message": "High value transaction"
    },
    {
      "name": "gold-tier-check",
      "condition": "#tier == 'GOLD'",
      "message": "Gold tier customer"
    },
    {
      "name": "risk-assessment",
      "condition": "#amount > 10000 && #country != 'US'",
      "message": "High risk international transaction"
    }
  ],
  "facts": {
    "amount": 1500,
    "tier": "GOLD",
    "country": "US",
    "customerId": "CUST001"
  }
}
```

## Batch Expression Evaluation

```

POST /api/expressions/batch
{
  "expressions": [
    {
      "name": "total-calculation",
      "expression": "#amount * #rate + #fee"
    },
    {
      "name": "risk-score",
      "expression": "#amount > 1000 ? 0.8 : 0.2"
    },
    {
      "name": "customer-tier",
      "expression": "#balance > 100000 ? 'PLATINUM' : (#balance > 10000 ? 'GOLD' : 'SILVER')"
    }
  ],
  "context": {
    "amount": 1500,
    "rate": 0.05,
    "fee": 25,
    "balance": 50000
  }
}

```

## Batch Template Processing

```

POST /api/templates/batch
{
  "templates": [
    {
      "name": "json-response",
      "type": "JSON",
      "template": "{\n  \"customerId\": \"#{@customerId}\",\n  \"status\": \"#{@status}\",\n  \"amount\": #{@amount}\n}"
    },
    {
      "name": "xml-notification",
      "type": "XML",
      "template": "<notification>\n  <customer>#{@customerId}</customer>\n  <message>#{@message}</message>\n</notification>"
    },
    {
      "name": "email-template",
      "type": "TEXT",
      "template": "Dear Customer #{@customerId},\n\nYour transaction of $#{@amount} has been #{@status}.\n\nThank you."
    }
  ],
  "context": {
    "customerId": "CUST001",
    "status": "APPROVED",
    "amount": 1500.0,
    "message": "Transaction approved successfully"
  }
}

```

## Batch Enrichment

```

POST /api/enrichment/batch
{
  "yamlConfiguration": "metadata:\n  name: \"Customer Enrichment\"\n  version: \"1.0.0\"\nenrichments:\n  - name: \"custo

```

```
"targetObjects": [  
  { "customerId": "CUST001", "amount": 1000.0 },  
  { "customerId": "CUST002", "amount": 2500.0 },  
  { "customerId": "CUST003", "amount": 750.0 }  
]  
}
```

## Batch Size Limits

- **Maximum batch size:** 100 items per request
- **Recommended batch size:** 10-50 items for optimal performance
- **Timeout:** 30 seconds for batch operations
- **Memory limit:** 50MB per batch request

## Common Workflows

### 1. Data Processing Pipeline

1. POST /api/transformations/dynamic → Normalize/transform data
2. POST /api/enrichment/enrich → Add external data
3. POST /api/rules/batch → Apply business rules
4. POST /api/templates/json → Generate output format

### 2. Risk Assessment Workflow

1. POST /api/expressions/evaluate → Calculate risk scores
2. POST /api/rules/execute → Apply risk rules
3. POST /api/templates/text → Generate alert messages
4. POST /api/datasources/{name}/lookup → Check blacklists

### 3. Customer Onboarding Workflow

1. POST /api/transformations/dynamic → Normalize customer data
2. POST /api/enrichment/predefined/customer-profile → Enrich profile
3. POST /api/rules/validate → Validate against business rules
4. POST /api/templates/batch → Create welcome messages

### 4. Transaction Processing Workflow

1. POST /api/expressions/batch → Calculate fees, totals
2. POST /api/rules/batch → Apply validation rules
3. POST /api/enrichment/enrich → Add customer/product data
4. POST /api/templates/json → Generate transaction record

## Authentication & Security

## Authentication Types

The API supports multiple authentication methods:

### No Authentication (Default)

```
curl -X POST http://localhost:8080/api/rules/execute \  
-H "Content-Type: application/json" \  
-d '{"rule": {...}, "facts": {...}}'
```

### API Key Authentication

```
curl -X POST http://localhost:8080/api/rules/execute \  
-H "Content-Type: application/json" \  
-H "X-API-Key: your-api-key" \  
-d '{"rule": {...}, "facts": {...}}'
```

### Bearer Token Authentication

```
curl -X POST http://localhost:8080/api/rules/execute \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer your-jwt-token" \  
-d '{"rule": {...}, "facts": {...}}'
```

### Basic Authentication

```
curl -X POST http://localhost:8080/api/rules/execute \  
-H "Content-Type: application/json" \  
-u "username:password" \  
-d '{"rule": {...}, "facts": {...}}'
```

## Security Headers

```
X-API-Key: your-api-key  
Authorization: Bearer jwt-token  
Authorization: Basic base64-credentials  
Content-Type: application/json  
Accept: application/json
```

## Performance & Optimization

### Performance Tips

1. **Use Batch Operations** - Process multiple items in single request
2. **Enable Caching** - Cache frequently accessed data sources
3. **Validate Expressions** - Use `/api/expressions/validate` before evaluation
4. **Optimize SpEL** - Avoid complex nested expressions
5. **Monitor Metrics** - Include `includeMetrics: true` in requests

6. **Limit Batch Sizes** - Keep batches under 100 items
7. **Use Async Processing** - For large datasets, consider background processing

## Caching Configuration

```
rules:
  cache:
    enabled: true
    ttl-seconds: 3600    # 1 hour cache
    max-size: 1000      # Maximum cached items
```

## Circuit Breaker Settings

```
datasources:
  circuit-breaker:
    enabled: true
    failure-threshold: 5
    timeout-seconds: 60
    success-threshold: 3
```

## Rate Limiting

- **Default limit:** 1000 requests/minute per client
- **Burst limit:** 100 requests/second
- **Headers:** X-RateLimit-Remaining , X-RateLimit-Reset

# Error Handling Best Practices

## 1. Response Validation

```
// Always check success field
if (response.success) {
  // Process data
  const result = response.data;
} else {
  // Handle error
  console.error(`Error: ${response.error} - ${response.message}`);
  if (response.correlationId) {
    console.error(`Correlation ID: ${response.correlationId}`);
  }
}
```

## 2. Retry Logic

```
async function executeRuleWithRetry(request, maxRetries = 3) {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      const response = await fetch('/api/rules/execute', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(request)
      });
    } catch (error) {
      // Handle retry logic
    }
  }
}
```



```

    });

    if (response.ok) {
        return await response.json();
    }

    if (response.status >= 400 && response.status < 500) {
        // Client error - don't retry
        throw new Error(`Client error: ${response.status}`);
    }

    // Server error - retry
    if (attempt === maxRetries) {
        throw new Error(`Max retries exceeded`);
    }

    await new Promise(resolve => setTimeout(resolve, 1000 * attempt));
} catch (error) {
    if (attempt === maxRetries) throw error;
}
}
}

```

### 3. Batch Error Handling

```

// Handle partial failures in batch operations
const batchResponse = await executeBatchRules(request);
if (batchResponse.success) {
    const { successful, failed } = batchResponse.summary;
    console.log(`Processed: ${successful} successful, ${failed} failed`);

    // Process individual results
    batchResponse.results.forEach((result, index) => {
        if (result.success) {
            // Handle successful result
        } else {
            console.error(`Item ${index} failed: ${result.message}`);
        }
    });
}
}

```

### 4. Input Validation

```

// Validate before sending
function validateRuleRequest(request) {
    if (!request.rule || !request.rule.name) {
        throw new Error('Rule name is required');
    }
    if (!request.rule.condition) {
        throw new Error('Rule condition is required');
    }
    if (!request.facts) {
        throw new Error('Facts are required');
    }
}
}

```

## Testing & Development

## Using curl

### Test Expression Evaluation

```
curl -X POST http://localhost:8080/api/expressions/evaluate \
-H "Content-Type: application/json" \
-d '{
  "expression": "#amount * #rate + #fee",
  "context": {"amount": 1000, "rate": 0.05, "fee": 25},
  "includeMetrics": true
}'
```

### Test Rule Execution

```
curl -X POST http://localhost:8080/api/rules/execute \
-H "Content-Type: application/json" \
-d '{
  "rule": {
    "name": "high-value-check",
    "condition": "#amount > 1000",
    "message": "High value transaction",
    "priority": "HIGH"
  },
  "facts": {"amount": 1500, "currency": "USD"}
}'
```

### Test Data Transformation

```
curl -X POST http://localhost:8080/api/transformations/dynamic \
-H "Content-Type: application/json" \
-d '{
  "data": {"firstName": "john", "email": "JOHN@EXAMPLE.COM"},
  "transformerRules": [
    {
      "name": "normalize-name",
      "condition": "#firstName != null",
      "transformation": "#firstName.substring(0,1).toUpperCase() + #firstName.substring(1).toLowerCase()",
      "targetField": "firstName"
    }
  ]
}'
```

### Test Template Processing

```
curl -X POST http://localhost:8080/api/templates/json \
-H "Content-Type: application/json" \
-d '{
  "template": "{\"id\": \"#{@customerId}\", \"name\": \"#{@customerName}\", \"total\": #{@amount}}",
  "context": {"customerId": "CUST001", "customerName": "John Doe", "amount": 1500}
}'
```

### Test Data Source Lookup

```
curl -X POST http://localhost:8080/api/datasources/customer-db/lookup \
-H "Content-Type: application/json" \
-d '{"customerId": "CUST001"}'
```

## Test Validation

```
curl -X POST http://localhost:8080/api/rules/validate \
-H "Content-Type: application/json" \
-d '{
  "data": {"age": 25, "email": "john@example.com"},
  "rules": [
    {
      "name": "age-check",
      "condition": "#data.age >= 18",
      "message": "Must be 18 or older",
      "severity": "ERROR"
    }
  ]
}'
```

## Using Swagger UI

- **URL:** <http://localhost:8080/swagger-ui.html>
- **Features:** Interactive API documentation, request/response examples, try-it-out functionality
- **Authentication:** Configure API keys or tokens in Swagger UI

## Using Postman

Import the OpenAPI specification from: <http://localhost:8080/v3/api-docs>

## Health Checks

```
# Application health
curl http://localhost:8080/actuator/health

# Detailed health (if authorized)
curl http://localhost:8080/actuator/health \
-H "Authorization: Bearer your-token"

# Application info
curl http://localhost:8080/actuator/info

# Metrics
curl http://localhost:8080/actuator/metrics
```

# Configuration

## Application Properties (application.yml)

```
# Server configuration
server:
```

```
port: 8080
servlet:
  context-path: /
compression:
  enabled: true

# Spring configuration
spring:
  application:
    name: apex-rest-api
  jackson:
    serialization:
      write-dates-as-timestamps: false
      indent-output: true
    deserialization:
      fail-on-unknown-properties: false
      default-property-inclusion: NON_NULL
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB

# Rules Engine configuration
rules:
  performance:
    monitoring:
      enabled: true
  error:
    recovery:
      enabled: true
  cache:
    enabled: true
    ttl-seconds: 3600
    max-size: 1000

# Management endpoints
management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
      base-path: /actuator
  endpoint:
    health:
      show-details: when-authorized
      show-components: always

# Logging configuration
logging:
  level:
    dev.mars.apex: INFO
    org.springframework.web: WARN
  pattern:
    console: "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"
    file: "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"

# OpenAPI documentation
springdoc:
  swagger-ui:
    path: /swagger-ui.html
    enabled: true
  api-docs:
    path: /v3/api-docs
```

## Environment-Specific Configuration

### Development Profile

```
spring:
  profiles:
    active: dev
logging:
  level:
    dev.mars.apex: DEBUG
management:
  endpoint:
    health:
      show-details: always
```

### Production Profile

```
spring:
  profiles:
    active: prod
logging:
  level:
    dev.mars.apex: INFO
    org.springframework.web: WARN
  file:
    name: /var/log/apex/apex-rest-api.log
management:
  endpoint:
    health:
      show-details: never
rules:
  cache:
    ttl-seconds: 7200 # 2 hours in production
```

## Data Source Configuration

```
# Example data source configuration
datasources:
  customer-db:
    type: "rest-api"
    baseUrl: "https://api.customer-service.com"
    authentication:
      type: "bearer-token"
      token: "${CUSTOMER_API_TOKEN}"
    cache:
      enabled: true
      ttl-seconds: 300
    circuit-breaker:
      enabled: true
      failure-threshold: 5
      timeout-seconds: 60
```

## Monitoring & Observability

## Key Metrics to Monitor

- **Response Times:** Average, P95, P99 per endpoint
- **Success/Failure Rates:** HTTP status code distribution
- **Expression Evaluation Performance:** SpEL execution times
- **Data Source Performance:** Lookup times and cache hit rates
- **Memory Usage:** Heap usage during batch operations
- **Rule Execution:** Rules evaluated per second
- **Cache Performance:** Hit/miss ratios
- **Circuit Breaker Status:** Open/closed state of data sources

## Available Metrics Endpoints

```
# All metrics
curl http://localhost:8080/actuator/metrics

# Specific metrics
curl http://localhost:8080/actuator/metrics/http.server.requests
curl http://localhost:8080/actuator/metrics/jvm.memory.used
curl http://localhost:8080/actuator/metrics/rules.execution.time
```

## Health Check Endpoints

```
# Basic health check
curl http://localhost:8080/actuator/health

# Detailed health (requires authorization)
curl http://localhost:8080/actuator/health \
  -H "Authorization: Bearer your-token"

# Component health
curl http://localhost:8080/actuator/health/diskSpace
curl http://localhost:8080/actuator/health/db
```

## Prometheus Metrics

```
# Prometheus format metrics
curl http://localhost:8080/actuator/prometheus
```

## Custom Metrics

The API exposes custom metrics for:

- `rules.execution.count` - Number of rules executed
- `rules.execution.time` - Rule execution duration
- `expressions.evaluation.count` - Number of expressions evaluated
- `expressions.evaluation.time` - Expression evaluation duration
- `transformations.count` - Number of transformations performed
- `enrichments.count` - Number of enrichments performed
- `templates.processed.count` - Number of templates processed

# Troubleshooting

## Common Issues

### 1. Expression Syntax Errors

```
{
  "success": false,
  "error": "EXPRESSION_ERROR",
  "message": "SpEL syntax error: Unexpected token 'invalid' at position 5"
}
```

**Solution:** Validate expressions using `/api/expressions/validate`

### 2. Rule Evaluation Failures

```
{
  "success": false,
  "error": "RULE_EVALUATION_ERROR",
  "message": "Cannot resolve property 'nonExistentField' on object"
}
```

**Solution:** Ensure all referenced fields exist in the context

### 3. Data Source Timeouts

```
{
  "success": false,
  "error": "DATA_SOURCE_ERROR",
  "message": "Connection timeout to data source 'customer-db'"
}
```

**Solution:** Check data source configuration and network connectivity

### 4. Batch Size Exceeded

```
{
  "success": false,
  "error": "VALIDATION_ERROR",
  "message": "Batch size 150 exceeds maximum allowed size of 100"
}
```

**Solution:** Reduce batch size or increase limit in configuration

## Debug Mode

Enable debug logging for detailed troubleshooting:

```
logging:
  level:
```

# Support & Resources

## Documentation

- **Swagger UI:** <http://localhost:8080/swagger-ui.html>
- **OpenAPI Spec:** <http://localhost:8080/v3/api-docs>
- **Technical Reference:** See `APEX_TECHNICAL_REFERENCE.md`
- **API Guide:** See `APEX_REST_API_GUIDE.md`

## Monitoring

- **Health Check:** <http://localhost:8080/actuator/health>
- **Metrics:** <http://localhost:8080/actuator/metrics>
- **Application Info:** <http://localhost:8080/actuator/info>

## Development

- **Source Code:** Check controller implementations for detailed behavior
- **Examples:** See `apex-core/src/main/resources/examples/`
- **Test Cases:** See `apex-rest-api/src/test/` for usage examples

## Performance Tuning

- **Batch Operations:** Use for multiple items to reduce overhead
- **Caching:** Enable for frequently accessed data sources
- **Expression Optimization:** Avoid complex nested expressions
- **Memory Management:** Monitor heap usage during large batch operations
- **Connection Pooling:** Configure for external data sources