Error Handling & Logging

Error Types and Handling

Standard Error Types

```
enum ErrorType {
  VALIDATION_ERROR = 'VALIDATION_ERROR',
  AUTHENTICATION_ERROR = 'AUTHENTICATION_ERROR',
  AUTHORIZATION_ERROR = 'AUTHORIZATION_ERROR',
  BLOCKCHAIN_ERROR = 'BLOCKCHAIN_ERROR',
  BUSINESS_LOGIC_ERROR = 'BUSINESS_LOGIC_ERROR',
  EXTERNAL_SERVICE_ERROR = 'EXTERNAL_SERVICE_ERROR',
  SYSTEM_ERROR = 'SYSTEM_ERROR'
}

interface BaseError extends Error {
  type: ErrorType;
  code: string;
  details?: Record<string, any>;
  timestamp: string;
  correlationId: string;
}

class VaultError extends BaseError {
  constructor(type: ErrorType, code: string, message: string, details?: Record<string, any>) {
    super(message);
    this.type = type;
    this.code = code;
    this.details = details;
    this.timestamp = new Date().toISOString();
    this.correlationId = getCurrentCorrelationId();
  }
}
```

Error Implementation

```
// Error throwing
async function createVault(params: VaultParams) {
  try {
```

```
    await validateVaultParams(params);
  } catch (error) {
    throw new VaultError(
      ErrorType.VALIDATION_ERROR,
      'INVALID_VAULT_PARAMS',
      'Invalid vault parameters provided',
      { params, validationErrors: error.details }
    );
  }
}

// Error handling middleware
app.use((error: Error, req: Request, res: Response, next: NextFunction) => {
  if (error instanceof VaultError) {
    logger.error('Vault operation failed', {
      type: error.type,
      code: error.code,
      message: error.message,
      details: error.details,
      correlationId: error.correlationId
    });

    return res.status(getHttpStatus(error.type)).json({
      status: 'error',
      error: {
        code: error.code,
        message: error.message,
        correlationId: error.correlationId
      }
    });
  }
  next(error);
});
```

Logging Implementation

Logger Configuration

```
import winston from 'winston';
import { ElasticsearchTransport } from 'winston-elasticsearch';
```

```
const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  defaultMeta: { service: 'vault-service' },
  transports: [
    new winston.transports.Console(),
    new ElasticsearchTransport({
      level: 'info',
      clientOpts: {
        node: process.env.ELASTICSEARCH_URL,
        auth: {
          username: process.env.ELASTICSEARCH_USERNAME,
          password: process.env.ELASTICSEARCH_PASSWORD
        }
      },
      index: 'vault-logs-${process.env.NODE_ENV}-%{YYYY.MM.DD}',
      mappingTemplate: {
        index_patterns: ['vault-logs-*'],
        settings: {
          number_of_shards: 1,
          number_of_replicas: 1
        },
        mappings: {
          properties: {
            '@timestamp': { type: 'date' },
            level: { type: 'keyword' },
            message: { type: 'text' },
            correlationId: { type: 'keyword' },
            type: { type: 'keyword' },
            code: { type: 'keyword' },
            details: { type: 'object' }
          }
        }
      }
    })
  ]
```

```
});
```

Structured Logging

```
// Request logging middleware
app.use((req: Request, res: Response, next: NextFunction) => {
  const correlationId = generateCorrelationId();
  setCurrentCorrelationId(correlationId);

  logger.info('Incoming request', {
    method: req.method,
    path: req.path,
    correlationId,
    ip: req.ip,
    userAgent: req.get('user-agent')
  });

  next();
});

// Business operation logging
async function executeProposal(proposalId: string) {
  logger.info('Starting proposal execution', {
    proposalId,
    correlationId: getCurrentCorrelationId()
  });

  try {
    const result = await performExecution(proposalId);

    logger.info('Proposal execution completed', {
      proposalId,
      result,
      correlationId: getCurrentCorrelationId()
    });

    return result;
  } catch (error) {
    logger.error('Proposal execution failed', {
      proposalId,
```

```
      error: error.message,
      stack: error.stack,
      correlationId: getCurrentCorrelationId()
    });
    throw error;
  }
}
```

ELK Stack Configuration

Logstash Pipeline

```
input {
  beats {
    port => 5044
  }
}

filter {
  json {
    source => "message"
  }

  date {
    match => [ "@timestamp", "ISO8601" ]
  }

  if [type] == "BLOCKCHAIN_ERROR" {
    grok {
      match => { "message" => "%{GREEDYDATA:transaction_hash}" }
    }
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "vault-logs-%{+YYYY.MM.dd}"
    document_type => "_doc"
  }
```

```
}
```

ElasticSearch Index Template

```
{
  "index_patterns": ["vault-logs-*"],
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1,
    "index.refresh_interval": "5s"
  },
  "mappings": {
    "properties": {
      "@timestamp": { "type": "date" },
      "service": { "type": "keyword" },
      "level": { "type": "keyword" },
      "correlationId": { "type": "keyword" },
      "type": { "type": "keyword" },
      "code": { "type": "keyword" },
      "message": { "type": "text" },
      "details": {
        "type": "object",
        "dynamic": true
      },
      "trace": {
        "properties": {
          "stack": { "type": "text" },
          "method": { "type": "keyword" },
          "line": { "type": "integer" }
        }
      },
      "request": {
        "properties": {
          "method": { "type": "keyword" },
          "path": { "type": "keyword" },
          "ip": { "type": "ip" },
          "userAgent": { "type": "text" }
        }
      }
    }
```

```
  }
}
```

Kibana Error Monitoring Dashboard

```
{
  "title": "Vault Error Monitoring",
  "panels": [
    {
      "type": "visualization",
      "title": "Errors by Type",
      "visualization": {
        "type": "pie",
        "aggs": [
          { "type": "count", "schema": "metric" },
          { "type": "terms", "field": "type", "schema": "segment" }
        ]
      }
    },
    {
      "type": "visualization",
      "title": "Error Timeline",
      "visualization": {
        "type": "line",
        "aggs": [
          { "type": "count", "schema": "metric" },
          { "type": "date_histogram", "field": "@timestamp", "schema": "segment" }
        ]
      }
    }
  ]
}
```

Implementation Guidelines

Best Practices

1. Error Handling
- Use custom error classes
- Include correlation IDs
- Add contextual details
- Handle errors at boundaries

- Never expose internal errors

2. Logging
- Use structured logging
- Include correlation IDs
- Log at appropriate levels
- Avoid sensitive data
- Add business context

3. Monitoring
- Set up alerting
- Monitor error rates
- Track response times
- Watch resource usage
- Set up dashboards

4. Security
- Sanitize logged data
- Encrypt sensitive logs
- Implement log retention
- Control log access
- Monitor suspicious patterns

Getting Started

1. Install Dependencies

```
npm install winston winston-elasticsearch @elastic/elasticsearch
```

2. Configure Environment

```
# .env file
ELASTICSEARCH_URL=http://localhost:9200
ELASTICSEARCH_USERNAME=elastic
ELASTICSEARCH_PASSWORD=changeme
LOG_LEVEL=info
```

3. Set Up ELK Stack

```
docker-compose up -d elasticsearch logstash kibana
```

4. Initialize Templates

```
curl -X PUT "localhost:9200/_template/vault-logs" -H "Content-Type: application/json" -d @template.json
```

5. Verify Setup

```
# Test logging
logger.info('Test log entry');
```

```
# Check Elasticsearch
curl -X GET "localhost:9200/vault-logs-*/_search"
```

Revision #1
Created 25 November 2024 17:15:21 by Aric Fedida
Updated 26 November 2024 01:25:20 by Aric Fedida