Low-Level Design (Low Level Design) - Securaa Custom Services

Document Information

- **Service Name**: Securaa Custom Services
- **Version**: 1.0
- **Date**: September 2025
- **Author**: Development Team
- **Related Documents**: High Level Design

Table of Contents

1. Implementation Overview

1.1 Technology Stack

- **Language**: Go 1.17
- **Web Framework**: Gorilla Mux
- **Database**: MongoDB with official Go driver
- **Cache**: Redis
- **Authentication**: JWT tokens with SAML integration
- **Encryption**: AES encryption for sensitive data
- **Containerization**: Docker
- **Build System**: Make with custom Makefile

1.2 Project Structure Analysis

```
zona_custom/
├── main.go                    # Application entry point
├── app.go                     # Application initialization and routing
├── go.mod                     # Go module dependencies
├── Dockerfile                 # Container configuration
├── Makefile                   # Build automation
├── constants/
│   └── constants.go           # Application constants
├── controllers/               # HTTP request handlers
│   ├── customAppController.go
│   ├── genericAppController.go
│   ├── exportController.go
│   ├── integrationController.go
│   └── eventsController.go
├── handlers/                  # Error handling
│   └── errorHandler.go
├── models/                    # Data structures
│   ├── Response.go
│   ├── export.go
│   ├── process.go
│   └── customtaskhandler.go
└── services/                  # Business logic
    ├── exportservice.go
    ├── importservice.go
    ├── integrationservice.go
    └── eventsService.go
```

Copy

2. Detailed Component Design

2.1 Application Bootstrap (main.go & app.go)

2.1.1 Main Function Flow

```
func main() {
    securaalog.Init("CORE_SERVICE_LOGS")  // Initialize logging
    app := App{}                          // Create app instance
    app.Initialize()                      // Setup dependencies
    app.Run(":8063")                      // Start HTTP server
}
```

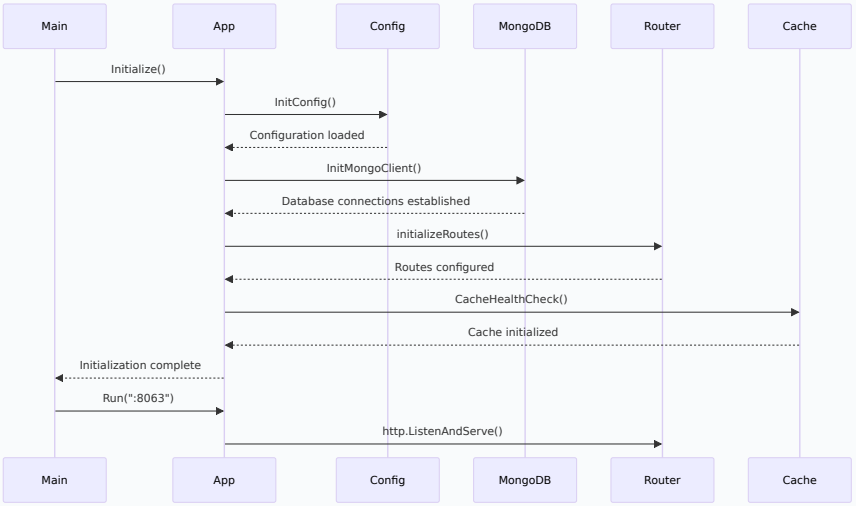
Copy

2.1.2 App Structure Design

```
type App struct {
    Router                *mux.Router                     // HTTP router
    AccessTokenHashMap    map[string]int64                // Token cache
    DBSession             map[string]common.SessionStruct // DB connections
    ConfigObject          config.ConfigStruct             // Configuration
    BuildType             string                          // Deployment type
    RequestResponseLog    bool                            // Logging flag
}
```

Copy

2.1.3 Initialization Sequence



2.2 Database Connection Management

2.2.1 Connection Pool Configuration

Copy

```
type SessionStruct struct {
    MongoDatabase string
    MongoClient   *mongo.Client
    RedisClient   *redis.Client
}

func InitMongoClient(app *App) {
    for _, tenant := range app.ConfigObject.TenantDatabase {
        client, err := mongo.Connect(ctx, options.Client().
            ApplyURI(tenant.ConnectionString).
            SetMaxPoolSize(100).
            SetMinPoolSize(10))

        app.DBSession[tenant.TenantId] = SessionStruct{
            MongoDatabase: tenant.Database,
            MongoClient:   client,
            RedisClient:   redis.NewClient(&redis.Options{
                Addr: tenant.RedisAddr,
            }),
        }
    }
}
```

2.3 Controller Layer Implementation

2.3.1 Custom Application Controller

Copy

```
type CustomAppController struct {
    DBSession map[string]common.SessionStruct
    Config    config.ConfigStruct
}

func (c *CustomAppController) CreateCustomApp(w http.ResponseWriter, r
*http.Request) {
    tenantId := r.Header.Get("X-Tenant-ID")

    // Parse multipart form for file uploads
    err := r.ParseMultipartForm(32 << 20) // 32MB limit
    if err != nil {
        http.Error(w, "Unable to parse form", http.StatusBadRequest)
        return
    }

    // Extract application data
    appData := extractCustomAppData(r)

    // Handle file uploads (logos, configurations)
    files := handleFileUploads(r, tenantId)
    appData.Files = files

    // Validate application data
    if err := validateCustomApp(appData); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    // Save to database
    result, err := c.saveCustomApp(appData, tenantId)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // Return success response
    response := models.Response{
        Status:  "success",
        Message: "Custom application created successfully",
        Data:    result,
```

```
    }

    json.NewEncoder(w).Encode(response)
}
```

2.3.2 Generic Task Controller

Copy

```
func (c *GenericAppController) CreateGenericTask(w http.ResponseWrit
*http.Request) {
    tenantId := r.Header.Get("X-Tenant-ID")

    var taskData models.GenericTask
    if err := json.NewDecoder(r.Body).Decode(&taskData); err != nil {
        http.Error(w, "Invalid JSON data", http.StatusBadRequest)
        return
    }

    // Validate task parameters
    if err := c.validateTaskParameters(taskData); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    // Check dependencies
    if err := c.validateTaskDependencies(taskData, tenantId); err != nil
{
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    // Save task definition
    taskId, err := c.saveGenericTask(taskData, tenantId)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    response := models.Response{
        Status:  "success",
        Message: "Generic task created successfully",
        Data:    map[string]interface{}{"taskId": taskId},
    }

    json.NewEncoder(w).Encode(response)
}
```

2.4 Service Layer Implementation

2.4.1 Export Service

Copy

```
type ExportService struct {
    DBSession map[string]common.SessionStruct
    Config    config.ConfigStruct
}

func (es *ExportService) ExportApplicationData(tenantId string, appIds
[]string) (*models.ExportData, error) {
    session := es.DBSession[tenantId]
    collection :=
session.MongoClient.Database(session.MongoDatabase).Collection("custom_ap
ps")

    ctx, cancel := context.WithTimeout(context.Background(),
30*time.Second)
    defer cancel()

    // Build export filter
    filter := bson.M{}
    if len(appIds) > 0 {
        objectIds := make([]primitive.ObjectID, len(appIds))
        for i, id := range appIds {
            objectId, _ := primitive.ObjectIDFromHex(id)
            objectIds[i] = objectId
        }
        filter["_id"] = bson.M{"$in": objectIds}
    }

    // Query applications
    cursor, err := collection.Find(ctx, filter)
    if err != nil {
        return nil, err
    }
    defer cursor.Close(ctx)

    var applications []models.CustomApplication
    if err := cursor.All(ctx, &applications); err != nil {
        return nil, err
    }

    // Build export data structure
    exportData := &models.ExportData{
```

```
        Version:     "1.0",
        ExportDate:  time.Now(),
        TenantId:    tenantId,
        Applications: applications,
        Dependencies: es.extractDependencies(applications, tenantId),
        Metadata:    es.buildExportMetadata(applications),
    }

    return exportData, nil
}
```

2.4.2 Integration Service

Copy

```
type IntegrationService struct {
    DBSession     map[string]common.SessionStruct
    Config        config.ConfigStruct
    httpClient    *http.Client
    rateLimiters  map[string]*rate.Limiter
}

func (is *IntegrationService) TestIntegrationConnection(tenantId string,
integrationId string) (*models.TestResult, error) {
    // Get integration configuration
    integration, err := is.getIntegration(tenantId, integrationId)
    if err != nil {
        return nil, err
    }

    // Apply rate limiting
    limiter := is.getRateLimiter(integrationId)
    if !limiter.Allow() {
        return &models.TestResult{
            Success: false,
            Message: "Rate limit exceeded",
        }, nil
    }

    // Decrypt credentials
    credentials, err :=
is.decryptCredentials(integration.EncryptedCredentials)
    if err != nil {
        return nil, err
    }

    // Build test request
    req, err := http.NewRequest("GET", integration.HealthCheckEndpoint,
nil)
    if err != nil {
        return nil, err
    }

    // Add authentication headers
    is.addAuthHeaders(req, credentials, integration.AuthType)

    // Execute test request
    ctx, cancel := context.WithTimeout(context.Background(),
```

```
30*time.Second)
    defer cancel()
    req = req.WithContext(ctx)

    resp, err := is.httpClient.Do(req)
    if err != nil {
        return &models.TestResult{
            Success: false,
            Message: fmt.Sprintf("Connection failed: %v", err),
        }, nil
    }
    defer resp.Body.Close()

    // Evaluate response
    success := resp.StatusCode >= 200 && resp.StatusCode < 300
    message := fmt.Sprintf("HTTP %d", resp.StatusCode)

    if !success {
        body, _ := ioutil.ReadAll(resp.Body)
        message = fmt.Sprintf("HTTP %d: %s", resp.StatusCode,
string(body))
    }

    return &models.TestResult{
        Success:      success,
        Message:      message,
        ResponseTime: time.Since(time.Now()).Milliseconds(),
    }, nil
}
```

3. Database Design & Data Models

3.1 Custom Applications Collection

Copy

```
type CustomApplication struct {
    ID              primitive.ObjectID `bson:"_id,omitempty" json:"id"`
    TenantId        string             `bson:"tenant_id" json:"tenantId"`
    Name            string             `bson:"name" json:"name"`
    Description     string             `bson:"description"
json:"description"`
    Version         string             `bson:"version" json:"version"`
    Logo            string             `bson:"logo" json:"logo"`
    Parameters      []Parameter        `bson:"parameters"
json:"parameters"`
    Tags            []string           `bson:"tags" json:"tags"`
    Category        string             `bson:"category" json:"category"`
    IsPublic        bool               `bson:"is_public" json:"isPublic"`
    CreatedBy       string             `bson:"created_by"
json:"createdBy"`
    CreatedAt       time.Time          `bson:"created_at"
json:"createdAt"`
    UpdatedAt       time.Time          `bson:"updated_at"
json:"updatedAt"`
    Configurations  []Configuration    `bson:"configurations"
json:"configurations"`
    Files           []FileAttachment   `bson:"files" json:"files"`
    Dependencies    []string           `bson:"dependencies"
json:"dependencies"`
    Status          string             `bson:"status" json:"status"`
}

type Parameter struct {
    Name         string      `bson:"name" json:"name"`
    DisplayName  string      `bson:"display_name" json:"displayName"`
    Type         string      `bson:"type" json:"type"`
    Required     bool        `bson:"required" json:"required"`
    DefaultValue interface{} `bson:"default_value" json:"defaultValue"`
    Validation   Validation  `bson:"validation" json:"validation"`
    Encrypted    bool        `bson:"encrypted" json:"encrypted"`
    Sensitive    bool        `bson:"sensitive" json:"sensitive"`
}

type Validation struct {
    MinLength int    `bson:"min_length" json:"minLength"`
```

```
    MaxLength int    `bson:"max_length" json:"maxLength"`
    Pattern   string `bson:"pattern" json:"pattern"`
    Options   []string `bson:"options" json:"options"`
}
```

3.2 Generic Tasks Collection

Copy

```
type GenericTask struct {
    ID           primitive.ObjectID `bson:"_id,omitempty" json:"id"`
    TenantId     string             `bson:"tenant_id" json:"tenantId"`
    Name         string             `bson:"name" json:"name"`
    Description  string             `bson:"description"
json:"description"`
    TaskType     string             `bson:"task_type" json:"taskType"`
    InputFields  []TaskField        `bson:"input_fields"
json:"inputFields"`
    OutputFields []TaskField        `bson:"output_fields"
json:"outputFields"`
    Dependencies []TaskDependency   `bson:"dependencies"
json:"dependencies"`
    Configuration TaskConfiguration `bson:"configuration"
json:"configuration"`
    CreatedBy    string             `bson:"created_by" json:"createdBy"`
    CreatedAt    time.Time          `bson:"created_at" json:"createdAt"`
    UpdatedAt    time.Time          `bson:"updated_at" json:"updatedAt"`
    IsActive     bool               `bson:"is_active" json:"isActive"`
}

type TaskField struct {
    Name         string      `bson:"name" json:"name"`
    DisplayName  string      `bson:"display_name" json:"displayName"`
    Type         string      `bson:"type" json:"type"`
    Required     bool        `bson:"required" json:"required"`
    DefaultValue interface{} `bson:"default_value" json:"defaultValue"`
    Mapping      string      `bson:"mapping" json:"mapping"`
}

type TaskDependency struct {
    TaskId       string            `bson:"task_id" json:"taskId"`
    OutputField  string            `bson:"output_field"
json:"outputField"`
    InputField   string            `bson:"input_field" json:"inputField"`
    Transformation string          `bson:"transformation"
json:"transformation"`
}
```

3.3 Database Indexes

Copy

```
// Custom Applications Indexes
db.custom_apps.createIndex({"tenant_id": 1})
db.custom_apps.createIndex({"tenant_id": 1, "name": 1}, {"unique": true})
db.custom_apps.createIndex({"tenant_id": 1, "category": 1})
db.custom_apps.createIndex({"tenant_id": 1, "tags": 1})
db.custom_apps.createIndex({"tenant_id": 1, "created_at": -1})
db.custom_apps.createIndex({"tenant_id": 1, "status": 1})

// Generic Tasks Indexes
db.generic_tasks.createIndex({"tenant_id": 1})
db.generic_tasks.createIndex({"tenant_id": 1, "name": 1}, {"unique":
true})
db.generic_tasks.createIndex({"tenant_id": 1, "task_type": 1})
db.generic_tasks.createIndex({"tenant_id": 1, "is_active": 1})
db.generic_tasks.createIndex({"tenant_id": 1, "created_at": -1})

// Integrations Indexes
db.integrations.createIndex({"tenant_id": 1})
db.integrations.createIndex({"tenant_id": 1, "name": 1}, {"unique":
true})
db.integrations.createIndex({"tenant_id": 1, "status": 1})
```

4. API Specifications

4.1 Custom Application APIs

ENDPOINT	METHOD	DESCRIPTION	AUTH REQUIRED
/api/v1/custom-apps	GET	List custom applications	Yes
/api/v1/custom-apps	POST	Create custom application	Yes
/api/v1/custom-apps/{id}	GET	Get application details	Yes
/api/v1/custom-apps/{id}	PUT	Update application	Yes
/api/v1/custom-apps/{id}	DELETE	Delete application	Yes

4.2 Generic Task APIs

ENDPOINT	METHOD	DESCRIPTION	AUTH REQUIRED
/api/v1/generic-tasks	GET	List generic tasks	Yes
/api/v1/generic-tasks	POST	Create generic task	Yes
/api/v1/generic-tasks/{id}/execute	POST	Execute task	Yes
/api/v1/generic-tasks/{id}/status	GET	Get execution status	Yes

5. Security Implementation Details

5.1 Authentication Implementation

Copy

```
type AuthMiddleware struct {
    TokenCache map[string]int64
    Config     config.ConfigStruct
}

func (am *AuthMiddleware) ValidateToken(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request)
{
        token := r.Header.Get("Authorization")
        if token == "" {
            http.Error(w, "Missing authorization header",
http.StatusUnauthorized)
            return
        }

        // Remove "Bearer " prefix
        token = strings.TrimPrefix(token, "Bearer ")

        // Check token cache first
        if expiry, exists := am.TokenCache[token]; exists {
            if time.Now().Unix() < expiry {
                next.ServeHTTP(w, r)
                return
            }
            delete(am.TokenCache, token)
        }

        // Validate JWT token
        claims, err := am.validateJWT(token)
        if err != nil {
            http.Error(w, "Invalid token", http.StatusUnauthorized)
            return
        }

        // Cache valid token
        am.TokenCache[token] = claims.ExpiresAt

        // Add user context
        ctx := context.WithValue(r.Context(), "user", claims.Subject)
        ctx = context.WithValue(ctx, "tenant", claims.TenantId)
```

```
        next.ServeHTTP(w, r.WithContext(ctx))
    })
}
```

5.2 Data Encryption Implementation

Copy

```
type EncryptionService struct {
    Key []byte
}

func (es *EncryptionService) EncryptField(data string) (string, error) {
    block, err := aes.NewCipher(es.Key)
    if err != nil {
        return "", err
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return "", err
    }

    nonce := make([]byte, gcm.NonceSize())
    if _, err = io.ReadFull(rand.Reader, nonce); err != nil {
        return "", err
    }

    ciphertext := gcm.Seal(nonce, nonce, []byte(data), nil)
    return base64.URLEncoding.EncodeToString(ciphertext), nil
}

func (es *EncryptionService) DecryptField(encrypted string) (string,
error) {
    ciphertext, err := base64.URLEncoding.DecodeString(encrypted)
    if err != nil {
        return "", err
    }

    block, err := aes.NewCipher(es.Key)
    if err != nil {
        return "", err
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return "", err
    }

    nonceSize := gcm.NonceSize()
    if len(ciphertext) < nonceSize {
```

```
        return "", errors.New("ciphertext too short")
    }

    nonce, ciphertext := ciphertext[:nonceSize], ciphertext[nonceSize:]
    plaintext, err := gcm.Open(nil, nonce, ciphertext, nil)
    if err != nil {
        return "", err
    }

    return string(plaintext), nil
}
```

6. Concurrency & Threading

6.1 Request Processing Concurrency

Copy

```
type TaskExecutor struct {
    WorkerPool chan chan models.TaskExecution
    Workers    []Worker
    Quit       chan bool
}

type Worker struct {
    ID         int
    WorkerPool chan chan models.TaskExecution
    JobChannel chan models.TaskExecution
    Quit       chan bool
}

func NewTaskExecutor(maxWorkers int) *TaskExecutor {
    workerPool := make(chan chan models.TaskExecution, maxWorkers)
    workers := make([]Worker, maxWorkers)

    for i := 0; i < maxWorkers; i++ {
        worker := Worker{
            ID:         i,
            WorkerPool: workerPool,
            JobChannel: make(chan models.TaskExecution),
            Quit:       make(chan bool),
        }
        workers[i] = worker
    }

    return &TaskExecutor{
        WorkerPool: workerPool,
        Workers:    workers,
        Quit:       make(chan bool),
    }
}
```

7. Error Handling Implementation

7.1 Centralized Error Handler

Copy

```
type ErrorHandler struct {
    Logger *log.Logger
}

type APIError struct {
    Code      int    `json:"code"`
    Message   string `json:"message"`
    Details   string `json:"details,omitempty"`
    RequestId string `json:"requestId"`
}

func (eh *ErrorHandler) HandleError(w http.ResponseWriter, r
*http.Request, err error, statusCode int) {
    requestId := r.Header.Get("X-Request-ID")
    if requestId == "" {
        requestId = generateRequestID()
    }

    apiError := APIError{
        Code:      statusCode,
        Message:   err.Error(),
        RequestId: requestId,
    }

    // Log error with context
    eh.Logger.Printf("Error [%s]: %v - Request: %s %s",
        requestId, err, r.Method, r.URL.Path)

    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(statusCode)
    json.NewEncoder(w).Encode(apiError)
}
```

8. Testing Strategy

8.1 Unit Testing

```
func TestCustomAppController_CreateCustomApp(t *testing.T) {
    // Setup test database
    testDB := setupTestDatabase()
    defer teardownTestDatabase(testDB)

    controller := &CustomAppController{
        DBSession: map[string]common.SessionStruct{
            "test-tenant": {
                MongoClient:   testDB.Client,
                MongoDatabase: testDB.Name,
            },
        },
    }

    // Create test request
    appData := models.CustomApplication{
        Name:        "Test App",
        Description: "Test Description",
        TenantId:    "test-tenant",
    }

    jsonData, _ := json.Marshal(appData)
    req, _ := http.NewRequest("POST", "/api/v1/custom-apps",
        bytes.NewBuffer(jsonData))
    req.Header.Set("Content-Type", "application/json")
    req.Header.Set("X-Tenant-ID", "test-tenant")

    // Execute request
    recorder := httptest.NewRecorder()
    controller.CreateCustomApp(recorder, req)

    // Assert response
    assert.Equal(t, http.StatusOK, recorder.Code)

    var response models.Response
    err := json.Unmarshal(recorder.Body.Bytes(), &response)
    assert.NoError(t, err)
    assert.Equal(t, "success", response.Status)
}
```

Copy

8.2 Integration Testing

```
func TestApplicationWorkflow(t *testing.T) {
    // Setup test environment
    testEnv := setupIntegrationTestEnvironment()
    defer teardownIntegrationTestEnvironment(testEnv)

    // Test complete workflow
    t.Run("Create Application", func(t *testing.T) {
        // Test application creation
    })

    t.Run("Create Generic Task", func(t *testing.T) {
        // Test task creation
    })

    t.Run("Execute Task", func(t *testing.T) {
        // Test task execution
    })

    t.Run("Export Data", func(t *testing.T) {
        // Test data export
    })
}
```

Copy

9. Deployment & Infrastructure

9.1 Docker Configuration

Copy

```
FROM golang:1.17-alpine AS builder

WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o securaa-custom-services
./main.go

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/

COPY --from=builder /app/securaa-custom-services .
COPY --from=builder /app/config.yaml .

EXPOSE 8063

CMD ["./securaa-custom-services"]
```

10. Monitoring Implementation

10.1 Health Check Implementation

Copy

```
type HealthChecker struct {
    DBSessions map[string]common.SessionStruct
    RedisClient *redis.Client
}

func (hc *HealthChecker) HealthCheck(w http.ResponseWriter, r
*http.Request) {
    status := models.HealthStatus{
        Service:   "securaa-custom-services",
        Status:    "healthy",
        Timestamp: time.Now(),
        Checks:    make(map[string]models.CheckResult),
    }

    // Check database connections
    for tenantId, session := range hc.DBSessions {
        ctx, cancel := context.WithTimeout(context.Background(),
5*time.Second)
        err := session.MongoClient.Ping(ctx, nil)
        cancel()

        checkName := fmt.Sprintf("mongodb-%s", tenantId)
        if err != nil {
            status.Checks[checkName] = models.CheckResult{
                Status:  "unhealthy",
                Message: err.Error(),
            }
            status.Status = "unhealthy"
        } else {
            status.Checks[checkName] = models.CheckResult{
                Status:  "healthy",
                Message: "Connection OK",
            }
        }
    }

    // Check Redis connection
    _, err := hc.RedisClient.Ping().Result()
    if err != nil {
        status.Checks["redis"] = models.CheckResult{
```

```
            Status:  "unhealthy",
            Message: err.Error(),
        }
        status.Status = "unhealthy"
    } else {
        status.Checks["redis"] = models.CheckResult{
            Status:  "healthy",
            Message: "Connection OK",
        }
    }

    w.Header().Set("Content-Type", "application/json")
    if status.Status != "healthy" {
        w.WriteHeader(http.StatusServiceUnavailable)
    }

    json.NewEncoder(w).Encode(status)
}
```

10.2 Metrics Collection

Copy

```
type MetricsCollector struct {
    RequestCount    *prometheus.CounterVec
    RequestDuration *prometheus.HistogramVec
    DatabaseOps     *prometheus.CounterVec
    ErrorCount      *prometheus.CounterVec
}

func NewMetricsCollector() *MetricsCollector {
    return &MetricsCollector{
        RequestCount: prometheus.NewCounterVec(
            prometheus.CounterOpts{
                Name: "zona_custom_services_requests_total",
                Help: "Total number of HTTP requests",
            },
            []string{"method", "endpoint", "status"},
        ),
        RequestDuration: prometheus.NewHistogramVec(
            prometheus.HistogramOpts{
                Name:    "zona_custom_services_request_duration_seconds",
                Help:    "HTTP request duration in seconds",
                Buckets: prometheus.DefBuckets,
            },
            []string{"method", "endpoint"},
        ),
        DatabaseOps: prometheus.NewCounterVec(
            prometheus.CounterOpts{
                Name: "zona_custom_services_database_operations_total",
                Help: "Total number of database operations",
            },
            []string{"operation", "collection", "status"},
        ),
        ErrorCount: prometheus.NewCounterVec(
            prometheus.CounterOpts{
                Name: "zona_custom_services_errors_total",
                Help: "Total number of errors",
            },
            []string{"type", "code"},
        ),
    }
}
```