Remote Integrated Services (RIS) System - Low-Level Design (LLD)

Table of Contents

Introduction

This Low-Level Design document provides comprehensive technical specifications for the Remote Integrated Services (RIS) System implementation. It includes detailed class structures, method signatures, database schemas, API endpoints, protocol specifications, and implementation specifics for both server and client components.

Document Scope

This LLD covers: - **Implementation Details**: Specific code structures, methods, and algorithms - **Technical Specifications**: Database schemas, API endpoints, protocol definitions - **Integration Patterns**: How components interact and communicate - **Quality Attributes**: Performance, security, reliability, and maintainability considerations - **Operational Aspects**: Deployment, monitoring, and maintenance procedures

Design Principles

The implementation follows these core principles: - **Simplicity**: Clear, readable, and maintainable code - **Performance**: Optimized for high throughput and low latency - **Security**: Security-by-design with comprehensive protection mechanisms - **Scalability**: Horizontal and vertical scaling capabilities - **Reliability**: Fault tolerance and automatic recovery mechanisms

Detailed Component Design

1. RIS Server Component Design

1.1 Main Application Structure

```
// App represents the main application structure
type App struct {
    Router                 *mux.Router                    // HTTP router for REST endpoints
    AccessTokenHashMap     map[string]int64               // JWT token expiry tracking
    Socket                 *websocket.Conn                // WebSocket connection handler
    SocketClients          []*websocket.Conn              // Multiple WebSocket connections
    DBSession              map[string]common.SessionStruct // Database sessions per tenant
    ConfigObject           config.ConfigStruct            // Application configuration
    SecondaryMongoDBHosts  []string                       // Secondary DB hosts for HA
    BuildType              string                         // Deployment type (mssp/standalone)
    DisconnectedRISClients map[string]int                 // Disconnected client tracking
    RISClientsResponse     map[string]int                 // Client response tracking
    MapMutex               sync.RWMutex                   // Thread-safe map access
    RequestResponseLog     bool                           // Logging configuration
}

// Core application methods
func (a *App) Initialize()
func (a *App) Run(port string, certFile string, keyFile string)
func (a *App) initializeRoutes()
func (a *App) loggingMiddleware(next http.Handler) http.Handler
func (a *App) liveWebSockets(socketConnection *websocket.Conn,
                           dbSession map[string]common.SessionStruct,
                           configObject config.ConfigStruct)
```

1.2 RIS Controller Design

```
type RISController struct{}

// Core RIS management methods
func (rc RISController) GetAllRIS(w http.ResponseWriter, r *http.Request,
                                dbSession map[string]common.SessionStruct,
                                configObject config.ConfigStruct)

func (rc RISController) AddNewRIS(w http.ResponseWriter, r *http.Request,
                                dbSession map[string]common.SessionStruct,
                                configObject config.ConfigStruct)

func (rc RISController) UpdateRIS(w http.ResponseWriter, r *http.Request,
                                dbSession map[string]common.SessionStruct,
                                configObject config.ConfigStruct)

func (rc RISController) DeleteRIS(w http.ResponseWriter, r *http.Request,
                                dbSession map[string]common.SessionStruct,
                                configObject config.ConfigStruct)

// Client management methods
func (rc RISController) GetSingleRISClient(w http.ResponseWriter, r *http.Request,
                                         dbSession map[string]common.SessionStruct,
                                         configObject config.ConfigStruct)

func (rc RISController) GetRISClientConfigData(w http.ResponseWriter, r *http.Request,
                                             dbSession map[string]common.SessionStruct,
                                             configObject config.ConfigStruct,
                                             secondaryMongoDBHosts []string)

// Status management methods
func (rc RISController) SetClientStatus(dbSession map[string]common.SessionStruct,
                                      configObject config.ConfigStruct) error

func (rc RISController) SetTenantStatus(dbSession map[string]common.SessionStruct,
                                      configObject config.ConfigStruct) error
```

1.3 RIS Model Design

```
type RIS struct {
    ID               int    `json:"id" bson:"id"`
    Name             string `json:"name" bson:"name"`
    Description      string `json:"description" bson:"description"`
    Host             string `json:"host" bson:"host"`
    Status           string `json:"status" bson:"status"`
    ConnectionStatus string `json:"connection_status" bson:"connection_status"`
    CreatedDate      int64  `json:"createddate" bson:"createddate"`
    UpdatedDate      int64  `json:"updateddate" bson:"updateddate"`
    Version          string `json:"version" bson:"version"`
    UniqueClientID   string `json:"unique_client_id" bson:"unique_client_id"`
    TenantCode       string `json:"tenantcode" bson:"tenantcode"`
}

// Model methods
func (ris *RIS) GetRISList(mongoDBClient mongo_driver.MongoClientWrapper,
                         configObject config.ConfigStruct) ([]map[string]string, error)

func (ris *RIS) GetRISData(mongoDBClient mongo_driver.MongoClientWrapper,
                         risID int, configObject config.ConfigStruct) (RIS, error)

func (ris *RIS) AddRIS(mongoDBClient mongo_driver.MongoClientWrapper,
                     configObject config.ConfigStruct) error

func (ris *RIS) UpdateRIS(mongoDBClient mongo_driver.MongoClientWrapper,
                        configObject config.ConfigStruct) error

func (ris *RIS) DeleteRIS(mongoDBClient mongo_driver.MongoClientWrapper,
                        configObject config.ConfigStruct) error
```

2. RIS Client Component Design

2.1 Main Application Structure

```
type App struct {
    Router       *mux.Router            // HTTP router for client endpoints
    ConfigObject config.ConfigStruct    // Configuration object
    CLI          *client.Client         // Docker client for service management
}

// Core application methods
func (a *App) Initialize()
func (a *App) Run(port string)
func (a *App) initializeRoutes()
func (a *App) loggingMiddleware(next http.Handler) http.Handler
func (a *App) risClient() // WebSocket client connection management
```

2.2 Task Handler Controller Design

```
type TaskHandlerController struct{}

// Task execution methods
func (tc TaskHandlerController) RunTaskByHandler(w http.ResponseWriter, r *http.Request)
func (tc TaskHandlerController) RunTask(w http.ResponseWriter, r *http.Request)
func (tc TaskHandlerController) ADTestConnectivity(w http.ResponseWriter, r *http.Request)

// Utility methods
func IsJson(str string) bool
```

2.3 RIS Client Controller Design

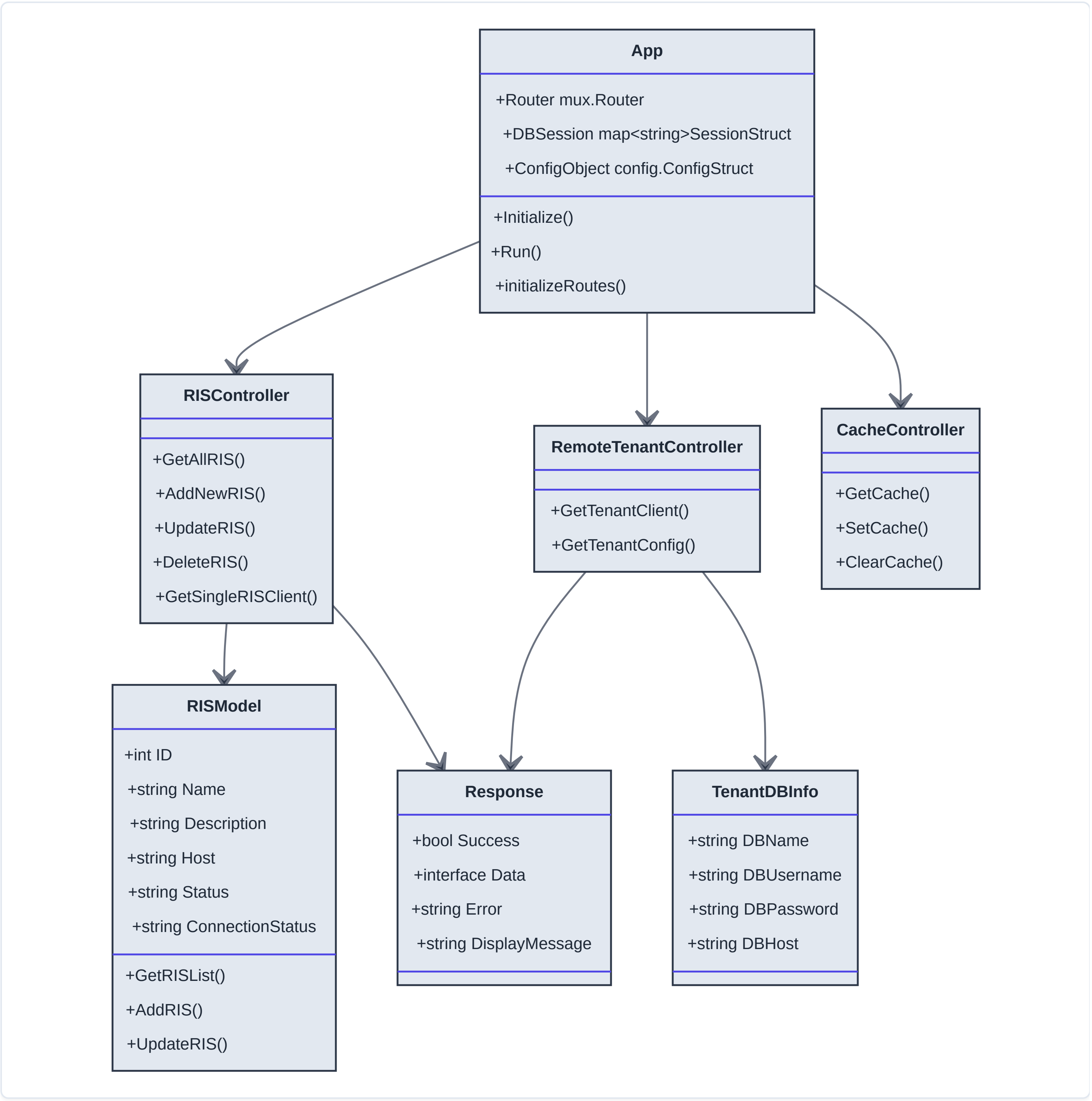
```
type RISClientController struct{}

// Service management methods
func (rc RISClientController) DeployServices(configObject config.ConfigStruct) error
func (rc RISClientController) CheckClientStatus(configObject config.ConfigStruct) (string, error)
func (rc RISClientController) IsServiceRunning(cli *client.Client, serviceName string) (bool, error)

// Network management methods
func (rc RISClientController) GetExternalIP() string
func (rc RISClientController) GetInternalIP() (string, error)

// File management methods
func readHostFile() string
func updateHostConfig(host string) error
```

2.4 Task Models Design

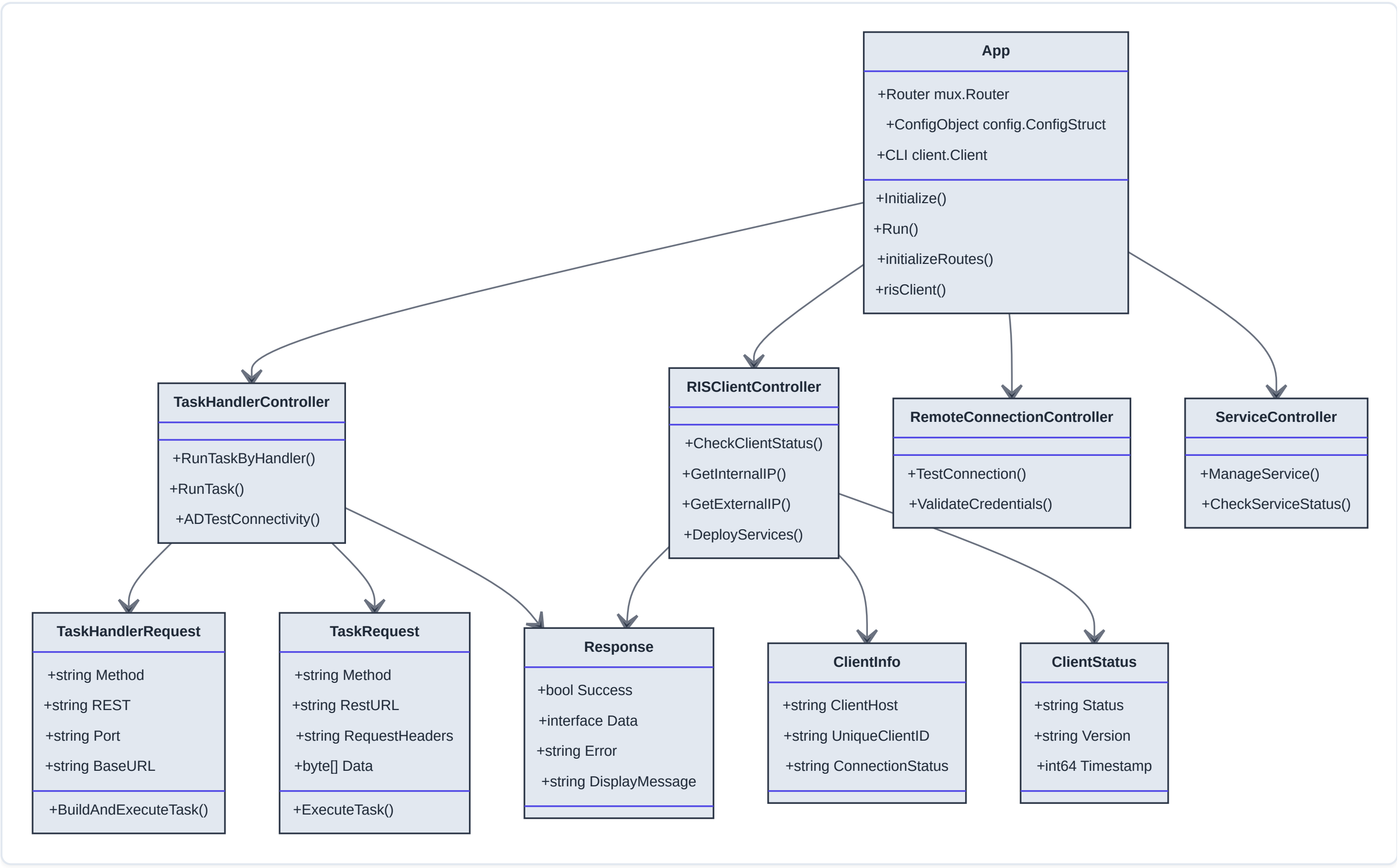
```
type TaskHandlerRequest struct {
    Method      string `json:"method" bson:"method"`
    REST        string `json:"rest" bson:"rest"`
    Port        string `json:"port" bson:"port"`
    BaseURL     string `json:"baseurl" bson:"baseurl"`
    Params      string `json:"params" bson:"params"`
    Data        []byte `json:"data" bson:"data"`
    AccessToken string `json:"access_token" bson:"access_token"`
    JwtToken    string `json:"jwt_token" bson:"jwt_token"`
}

type TaskRequest struct {
    Method         string               `json:"method" bson:"method"`
    RestURL        string               `json:"resturl" bson:"resturl"`
    RequestHeaders string               `json:"headers" bson:"headers"`
    Data           []byte               `json:"data" bson:"data"`
    AuthType       string               `json:"authtype" bson:"authtype"`
    Username       string               `json:"username" bson:"username"`
    Password       string               `json:"password" bson:"password"`
    BearerToken    string               `json:"bearertoken" bson:"bearertoken"`
    Cookies        map[int]*http.Cookie `json:"cookies" bson:"cookies"`
}

// Task execution methods
func (taskRequest *TaskHandlerRequest) BuildAndExecuteTask() ([]byte, error)
func (taskRequest *TaskRequest) ExecuteTask() ([]byte, error, int, string,
                                             map[int]*http.Cookie, http.Header)
```

Class Diagrams

1. Server-Side Class Hierarchy



2. Client-Side Class Hierarchy



Database Design

1. MongoDB Collections Schema

RIS Collection (`risCollection`)

```
{
    "_id": ObjectId,
    "id": Number,                    // Auto-increment ID
    "name": String,                  // RIS client name
    "description": String,           // Description
    "host": String,                  // Client host URL
    "status": String,                // active/inactive/deleted
    "connection_status": String,     // connected/disconnected/created
    "createddate": NumberLong,       // Creation timestamp
    "updateddate": NumberLong,       // Last update timestamp
    "version": String,               // RIS client version
    "unique_client_id": String,      // UUID for client identification
    "tenantcode": String             // Associated tenant code
}
```

Tenants Collection (`tenantsCollection`)

```
{
    "_id": ObjectId,
    "id": Number,                   // Tenant ID
    "db_name": String,              // Database name
    "db_username": String,          // Database username
    "db_password": String,          // Database password
    "db_host": String,              // Database host
    "tenantcode": String,           // Tenant code
    "unique_client_id": String,     // UUID
    "connection_status": String,    // Connection status
    "status": String,               // Tenant status
    "host_type": String,            // local/remote
    "createddate": NumberLong,      // Creation timestamp
    "updateddate": NumberLong       // Update timestamp
}
```

Active Instances Collection (`mongoActiveInstanceCollection`)

```
{
    "_id": ObjectId,
    "instance_id": NumberLong,      // Instance identifier
    "integration_id": NumberLong,   // Integration type ID
    "title": String,                // Instance title
    "status": String,               // active/inactive
    "base_url": String,             // Service base URL
    "uuid": String,                 // Associated RIS client UUID
    "credentials_id": Number,       // Credentials reference
    "customapp": Boolean,           // Custom application flag
    "instance_status": String,      // Runtime status
    "tenantcode": String            // Tenant association
}
```

2. Index Strategy

```
// RIS Collection Indexes
db.ris.createIndex({"unique_client_id": 1})
db.ris.createIndex({"status": 1, "connection_status": 1})
db.ris.createIndex({"tenantcode": 1})

// Tenants Collection Indexes
db.tenants.createIndex({"unique_client_id": 1})
db.tenants.createIndex({"tenantcode": 1})
db.tenants.createIndex({"status": 1, "connection_status": 1})

// Active Instances Collection Indexes
db.active_instances.createIndex({"uuid": 1})
db.active_instances.createIndex({"instance_id": 1})
db.active_instances.createIndex({"tenantcode": 1, "status": 1})
```

API Specifications

1. Server REST API Endpoints

RIS Management APIs

```
GET    /platform/v1/ris                     // Get all RIS clients
POST   /platform/v1/ris                     // Add new RIS client
PUT    /platform/v1/ris                     // Update RIS client
DELETE /platform/v1/ris                     // Delete RIS client

GET    /platform/v1/risclient/{ris-id}/{host-name}  // Get RIS config data
GET    /internal/v1/risclient/{ris-uuid}             // Get single RIS client
GET    /internal/v1/dockertag                        // Get Docker tag info
```

Tenant Management APIs

```
GET    /internal/v1/remotetenantclient/{tenant-uuid}          // Get tenant client
GET    /platform/v1/remotetenantconfigdata/{tenant-id}/{hostname}  // Get tenant config
```

WebSocket Endpoints

```
WS     /pingpong                           // RIS client connection
WS     /pingpongdbdetails                  // Tenant DB connection
```

2. Client REST API Endpoints

Task Execution APIs

```
POST   /runtaskbyhandler/                  // Execute task via handler
POST   /runtask/                           // Execute third-party task
POST   /adtestconnectivity/                // Test AD connectivity
```

Service Management APIs

```
POST   /remoteconnection/                  // Remote connection test
POST   /manageservices/                    // Docker service management
GET    /test/                              // Health check endpoint
```

3. Request/Response Specifications

RIS Client Registration Request

```
{
    "clienthost": "192.168.1.100",
    "unique_client_id": "uuid-string",
    "connection_status": "connected",
    "isfirstping": "true",
    "serverhost": "server.example.com",
    "host_type": "private",
    "isnetworkchanged": "false",
    "tenantcode": "tenant1",
    "ris_nat_ip": "10.0.0.100"
}
```

Task Execution Request

```
{
    "method": "POST",
    "resturl": "https://api.example.com/endpoint",
    "headers": "{\"Content-Type\": \"application/json\"}",
    "data": "base64-encoded-data",
    "authtype": "bearer",
    "bearertoken": "jwt-token",
    "cookies": {}
}
```

Standard Response Format

```
{
    "success": true,
    "data": {},
    "error": "",
    "displaymessage": "",
    "errorpath": "",
    "sessionexpired": false,
    "cookies": {},
    "headers": {}
}
```

WebSocket Protocol Design

1. Connection Lifecycle

```
// Connection establishment
func (a *App) liveWebSockets(socketConnection *websocket.Conn,
                            dbSession map[string]common.SessionStruct,
                            configObject config.ConfigStruct) {

    clientInfo := ClientInfo{}
    messageMap := make(map[string]string)
    hostSavedMap := make(map[string]string)

    defer func() {
        socketConnection.Close()
        // Handle disconnection cleanup
    }()

    // Message processing loop
    for {
        _, bytes, err := socketConnection.ReadMessage()
        if err != nil {
            break
        }

        // Process client information
        json.Unmarshal(bytes, &clientInfo)

        // Handle different message types
        switch {
        case clientInfo.IsNetworkChanged == "true":
            // Handle network change
        default:
            // Regular processing
        }

        // Send response
        socketConnection.WriteMessage(websocket.TextMessage, responseBytes)
    }
}
```

2. Message Types and Formats

Client Registration Message

```
{
    "clienthost": "client-ip",
    "unique_client_id": "client-uuid",
    "connection_status": "connected",
    "isfirstping": "true",
    "host_type": "private|public|ris_nat_ip",
    "isnetworkchanged": "false"
}
```

Server Response Messages

- `"pong"` - Normal acknowledgment
- `"private ip not reachable"` - Network connectivity issue
- `"client not reachable"` - Client unreachable
- `"ris nat ip not reachable"` - NAT IP connectivity issue

3. Connection Management

```
// Connection tracking
type ConnectionManager struct {
    DisconnectedRISClients map[string]int
    RISClientsResponse     map[string]int
    MapMutex               sync.RWMutex
}

func (cm *ConnectionManager) SetDisconnectedRISClientsCount(uuid string, count int) {
    cm.MapMutex.Lock()
    defer cm.MapMutex.Unlock()
    cm.DisconnectedRISClients[uuid] = count
}

func (cm *ConnectionManager) GetDisconnectedRISClientsCount(uuid string) int {
    cm.MapMutex.Lock()
    defer cm.MapMutex.Unlock()
    return cm.DisconnectedRISClients[uuid]
}
```

Task Execution Framework

1. Task Types and Processing

Task Handler Request Processing

```
func (tc TaskHandlerController) RunTaskByHandler(w http.ResponseWriter, r *http.Request) {
    var taskrequest models.TaskHandlerRequest
    var response models.Response

    // Decode request
    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&taskrequest)

    // Execute task
    responseData, err := taskrequest.BuildAndExecuteTask()

    // Process response
    json.Unmarshal(responseData, &response)
    utils.RespondWithJSON(w, http.StatusOK, response)
}
```

Task Execution Implementation

```
func (taskRequest *TaskHandlerRequest) BuildAndExecuteTask() ([]byte, error) {
    var response []byte
    var err error

    switch taskRequest.Method {
    case "GET":
        restURL := taskRequest.BaseURL + ":" + taskRequest.Port +
                  "/" + taskRequest.REST + "/" + taskRequest.Params
        response, err = utils.SecuraaHTTPRequest(restURL, nil,
                       taskRequest.Method, taskRequest.AccessToken,
                       taskRequest.JwtToken)

    case "POST", "PUT", "DELETE", "PATCH":
        restURL := taskRequest.BaseURL + ":" + taskRequest.Port +
                  "/" + taskRequest.REST + "/"
        response, err = utils.SecuraaHTTPRequest(restURL, taskRequest.Data,
                       taskRequest.Method, taskRequest.AccessToken,
                       taskRequest.JwtToken)
    }

    return response, err
}
```

2. Service Management

Docker Service Management

```
func (rc RISClientController) IsServiceRunning(cli *client.Client,
                                             serviceName string) (bool, error) {
    serviceList, err := cli.ServiceList(context.Background(),
                                       types.ServiceListOptions{})
    if err != nil {
        return false, err
    }

    for _, service := range serviceList {
        if service.Spec.Name == serviceName {
            return true, nil
        }
    }

    return false, nil
}
```

Error Handling and Logging

1. Error Handling Strategy

Standardized Error Response

```
func HandleError(err error, message string) models.Response {
    response := models.Response{}
    response.Success = false
    response.Error = err.Error()
    response.DisplayMessage = message
    response.ErrorPath = getCallerInfo()
    return response
}
```

Error Categories

1. **Network Errors**: Connection timeouts, unreachable hosts
2. **Authentication Errors**: Invalid tokens, expired sessions
3. **Authorization Errors**: Insufficient permissions
4. **Validation Errors**: Invalid input data
5. **System Errors**: Database connection issues, service unavailable

2. Logging Implementation

Structured Logging

```
var logger = securaalog.New("component_name")

// Log levels and usage
logger.Debug("Debug information")
logger.Info("Informational message")
logger.Error("Error message with context", err.Error())
```

Log Rotation and Management

- File-based logging with automatic rotation
- Configurable log levels per component
- Centralized log aggregation support
- Security-aware logging (password masking)

Configuration Management

1. Configuration Structure

```
type ConfigStruct struct {
    ServerHost         string
    MongoDBHost        string
    ElasticSearchHost  string
    LoggingPath        string
    CertFile           string
    KeyFile            string
    // ... additional configuration fields
}
```

2. Dynamic Configuration Updates

Configuration Hot-reload

```
func (a *App) reloadConfiguration() error {
    newConfig := utils.InitConfig()
    if validateConfig(newConfig) {
        a.ConfigObject = newConfig
        return nil
    }
    return errors.New("invalid configuration")
}
```

Security Implementation Framework

1. Comprehensive TLS Configuration

```go
// TLS Configuration with Advanced Security Parameters
func configureTLS() *tls.Config {
    cert, err := tls.LoadX509KeyPair("ssl/certs/securaa_server.crt", "ssl/private/securaa_server.key")
    if err != nil {
        log.Fatal("Failed to load TLS certificates:", err)
    }

    // Load CA certificate for client verification
    caCert, err := ioutil.ReadFile("ssl/certs/securaa-ca.pem")
    if err != nil {
        log.Fatal("Failed to load CA certificate:", err)
    }
    caCertPool := x509.NewCertPool()
    caCertPool.AppendCertsFromPEM(caCert)

    return &tls.Config{
        Certificates:             []tls.Certificate{cert},
        ClientAuth:               tls.RequireAndVerifyClientCert,
        ClientCAs:                caCertPool,
        MinVersion:               tls.VersionTLS12,
        MaxVersion:               tls.VersionTLS13,
        PreferServerCipherSuites: true,
        CipherSuites: []uint16{
            tls.TLS_AES_256_GCM_SHA384,           // TLS 1.3
            tls.TLS_CHACHA20_POLY1305_SHA256,      // TLS 1.3
            tls.TLS_AES_128_GCM_SHA256,           // TLS 1.3
            tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,    // TLS 1.2
            tls.TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,     // TLS 1.2
            tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,    // TLS 1.2
        },
        CurvePreferences: []tls.CurveID{
            tls.X25519,
            tls.CurveP256,
            tls.CurveP384,
        },
        SessionTicketsDisabled: false,
        InsecureSkipVerify:     false,
        ServerName:             "securaa-ris-server",
    }
}

// WebSocket TLS Upgrader with Security Headers
func createSecureWebSocketUpgrader() websocket.Upgrader {
    return websocket.Upgrader{
        ReadBufferSize:    4096,
        WriteBufferSize:   4096,
        EnableCompression: true,
        CheckOrigin: func(r *http.Request) bool {
            origin := r.Header.Get("Origin")
            return isAllowedOrigin(origin)
        },
        Subprotocols: []string{"securaa-ris-protocol"},
        Error: func(w http.ResponseWriter, r *http.Request, status int, reason error) {
            log.Printf("WebSocket error: %v", reason)
            w.Header().Set("Sec-WebSocket-Version", "13")
            http.Error(w, http.StatusText(status), status)
        },
    }
}
```

2. Advanced Authentication and Authorization Framework

Multi-Factor JWT Token Validation

```
type JWTClaims struct {
    UserID      string    `json:"user_id"`
    TenantID    string    `json:"tenant_id"`
    Role        string    `json:"role"`
    Permissions []string  `json:"permissions"`
    IssuedAt    time.Time `json:"iat"`
    ExpiresAt   time.Time `json:"exp"`
    Issuer      string    `json:"iss"`
    Subject     string    `json:"sub"`
    SessionID   string    `json:"session_id"`
    DeviceID    string    `json:"device_id"`
    IPAddress   string    `json:"ip_address"`
    jwt.StandardClaims
}

func validateJWTToken(tokenString string, configObject config.ConfigStruct) (*JWTClaims, error) {
    // Parse JWT token with validation
    token, err := jwt.ParseWithClaims(tokenString, &JWTClaims{}, func(token *jwt.Token) (interface{}, error) {
        // Validate signing method
        if _, ok := token.Method.(*jwt.SigningMethodRSA); !ok {
            return nil, fmt.Errorf("unexpected signing method: %v", token.Header["alg"])
        }

        // Load public key for verification
        publicKey, err := loadRSAPublicKey(configObject.JWTPublicKeyPath)
        if err != nil {
            return nil, fmt.Errorf("failed to load public key: %v", err)
        }

        return publicKey, nil
    })

    if err != nil {
        return nil, fmt.Errorf("token parsing failed: %v", err)
    }

    claims, ok := token.Claims.(*JWTClaims)
    if !ok || !token.Valid {
        return nil, fmt.Errorf("invalid token claims")
    }

    // Additional validation checks
    if err := validateTokenClaims(claims); err != nil {
        return nil, fmt.Errorf("token validation failed: %v", err)
    }

    // Check token blacklist
    if isTokenBlacklisted(claims.SessionID) {
        return nil, fmt.Errorf("token has been revoked")
    }

    // Verify IP address binding (if enabled)
    if configObject.EnableIPBinding && claims.IPAddress != getClientIP(r) {
        return nil, fmt.Errorf("token IP address mismatch")
    }

    return claims, nil
}

// Comprehensive Role-Based Access Control
type Permission struct {
    Resource string `json:"resource"`
    Action   string `json:"action"`
    Scope    string `json:"scope"`
}

type Role struct {
    Name        string       `json:"name"`
    Permissions []Permission `json:"permissions"`
    Inherits    []string     `json:"inherits"`
}

func checkPermissions(userClaims *JWTClaims, resource string, action string, context map[string]interface{}) bool {
    // Load user role and permissions
    userRole := getUserRole(userClaims.Role)
    if userRole == nil {
        return false
    }

    // Check direct permissions
    if hasDirectPermission(userRole, resource, action, context) {
        return true
    }

    // Check inherited permissions
    for _, inheritedRoleName := range userRole.Inherits {
        inheritedRole := getUserRole(inheritedRoleName)
        if inheritedRole != nil && hasDirectPermission(inheritedRole, resource, action, context) {
            return true
        }
    }

    // Check tenant-specific permissions
    if userClaims.TenantID != "" {
        return hasTenantPermission(userClaims.TenantID, userClaims.UserID, resource, action, context)
    }

    return false
}

// Multi-Tenant Security Context
func createSecurityContext(claims *JWTClaims, request *http.Request) *SecurityContext {
    return &SecurityContext{
        UserID:      claims.UserID,
        TenantID:    claims.TenantID,
        Role:        claims.Role,
        Permissions: claims.Permissions,
        SessionID:   claims.SessionID,
        IPAddress:   getClientIP(request),
        UserAgent:   request.Header.Get("User-Agent"),
        Timestamp:   time.Now(),
        RequestID:   generateRequestID(),
    }
}
```

3. Encryption and Data Protection

End-to-End Data Encryption

```
type EncryptionService struct {
    masterKey    []byte
    keyDerivation string
    cipher       string
}

func NewEncryptionService(masterKey []byte) *EncryptionService {
    return &EncryptionService{
        masterKey:     masterKey,
        keyDerivation: "PBKDF2",
        cipher:        "AES-256-GCM",
    }
}

func (es *EncryptionService) EncryptData(data []byte, tenantID string) ([]byte, error) {
    // Derive tenant-specific key
    tenantKey := es.deriveTenantKey(tenantID)

    // Generate random nonce
    nonce := make([]byte, 12)
    if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
        return nil, err
    }

    // Create AES-GCM cipher
    block, err := aes.NewCipher(tenantKey)
    if err != nil {
        return nil, err
    }

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

    // Encrypt data
    ciphertext := gcm.Seal(nil, nonce, data, nil)

    // Combine nonce and ciphertext
    encrypted := append(nonce, ciphertext...)

    return encrypted, nil
}

func (es *EncryptionService) DecryptData(encryptedData []byte, tenantID string) ([]byte, error) {
    if len(encryptedData) < 12 {
        return nil, errors.New("invalid encrypted data")
    }

    // Extract nonce and ciphertext
    nonce := encryptedData[:12]
    ciphertext := encryptedData[12:]

    // Derive tenant-specific key
    tenantKey := es.deriveTenantKey(tenantID)

    // Create AES-GCM cipher
    block, err := aes.NewCipher(tenantKey)
    if err != nil {
        return nil, err
    }

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

    // Decrypt data
    plaintext, err := gcm.Open(nil, nonce, ciphertext, nil)
    if err != nil {
        return nil, err
    }

    return plaintext, nil
}

// Field-Level Database Encryption
```

```go
func (es *EncryptionService) EncryptField(field string, value interface{}, tenantID string) (string, error) {
    // Serialize value
    data, err := json.Marshal(value)
    if err != nil {
        return "", err
    }

    // Encrypt data
    encrypted, err := es.EncryptData(data, tenantID)
    if err != nil {
        return "", err
    }

    // Encode to base64
    return base64.StdEncoding.EncodeToString(encrypted), nil
}
```

4. Security Monitoring and Audit System

Comprehensive Audit Logging

```go
type AuditEvent struct {
    EventID     string                 `json:"event_id"`
    Timestamp   time.Time             `json:"timestamp"`
    UserID      string                `json:"user_id"`
    TenantID    string                `json:"tenant_id"`
    Action      string                `json:"action"`
    Resource    string                `json:"resource"`
    IPAddress   string                `json:"ip_address"`
    UserAgent   string                `json:"user_agent"`
    SessionID   string                `json:"session_id"`
    Result      string                `json:"result"`
    ErrorCode   string                `json:"error_code,omitempty"`
    Details     map[string]interface{} `json:"details"`
    Severity    string                `json:"severity"`
    Category    string                `json:"category"`
    Compliance  []string              `json:"compliance_tags"`
}

type AuditLogger struct {
    database     *mongo.Database
    encryption   *EncryptionService
    retention    time.Duration
    complianceMode bool
}

func (al *AuditLogger) LogSecurityEvent(event *AuditEvent) error {
    // Add automatic fields
    event.EventID = generateEventID()
    event.Timestamp = time.Now().UTC()

    // Encrypt sensitive data
    if al.encryption != nil {
        if err := al.encryptSensitiveFields(event); err != nil {
            return fmt.Errorf("failed to encrypt audit data: %v", err)
        }
    }

    // Add compliance tags
    event.Compliance = al.getComplianceTags(event)

    // Store in database
    collection := al.database.Collection("audit_logs")
    _, err := collection.InsertOne(context.Background(), event)
    if err != nil {
        return fmt.Errorf("failed to store audit event: %v", err)
    }

    // Real-time security monitoring
    if al.isHighRiskEvent(event) {
        go al.triggerSecurityAlert(event)
    }

    return nil
}

// Real-time Security Monitoring
type SecurityMonitor struct {
    alertThresholds map[string]int
    timeWindows     map[string]time.Duration
    alerter         *AlertManager
}

func (sm *SecurityMonitor) MonitorSecurityEvents(events <-chan *AuditEvent) {
    eventCounts := make(map[string]int)

    for event := range events {
        // Track event patterns
        key := fmt.Sprintf("%s:%s:%s", event.UserID, event.Action, event.IPAddress)
        eventCounts[key]++

        // Check for suspicious patterns
        if sm.detectSuspiciousActivity(event, eventCounts) {
            alert := &SecurityAlert{
                Type:        "SUSPICIOUS_ACTIVITY",
                Severity:    "HIGH",
                Description: "Multiple failed authentication attempts detected",
                UserID:      event.UserID,
                IPAddress:   event.IPAddress,
                Timestamp:   time.Now(),
                Events:      []string{event.EventID},
            }

            sm.alerter.SendAlert(alert)
        }

        // Check for compliance violations
        if sm.detectComplianceViolation(event) {
            alert := &SecurityAlert{
                Type:        "COMPLIANCE_VIOLATION",
                Severity:    "CRITICAL",
                Description: "Data access violation detected",
                UserID:      event.UserID,
                TenantID:    event.TenantID,
                Timestamp:   time.Now(),
                Events:      []string{event.EventID},
            }

            sm.alerter.SendAlert(alert)
        }
    }
}
```

5. Network Security and Rate Limiting

Advanced Rate Limiting with Redis

```
type RateLimiter struct {
    redis       *redis.Client
    limits      map[string]RateLimit
    defaultRate RateLimit
}

type RateLimit struct {
    Requests int           `json:"requests"`
    Window   time.Duration `json:"window"`
    Burst    int           `json:"burst"`
}

func (rl *RateLimiter) CheckRateLimit(identifier string, action string) (bool, error) {
    key := fmt.Sprintf("rate_limit:%s:%s", identifier, action)

    // Get rate limit configuration
    limit := rl.getLimitForAction(action)

    // Use sliding window log algorithm
    now := time.Now().Unix()
    windowStart := now - int64(limit.Window.Seconds())

    pipe := rl.redis.Pipeline()

    // Remove expired entries
    pipe.ZRemRangeByScore(context.Background(), key, "0", fmt.Sprintf("%d", windowStart))

    // Count current requests
    pipe.ZCard(context.Background(), key)

    // Add current request
    pipe.ZAdd(context.Background(), key, &redis.Z{Score: float64(now), Member: fmt.Sprintf("%d", now)})

    // Set expiration
    pipe.Expire(context.Background(), key, limit.Window)

    results, err := pipe.Exec(context.Background())
    if err != nil {
        return false, err
    }

    currentCount := results[1].(*redis.IntCmd).Val()

    // Check if rate limit exceeded
    if int(currentCount) >= limit.Requests {
        // Check burst allowance
        burstKey := fmt.Sprintf("burst:%s:%s", identifier, action)
        burstCount, err := rl.redis.Get(context.Background(), burstKey).Int()
        if err == redis.Nil {
            burstCount = 0
        } else if err != nil {
            return false, err
        }

        if burstCount >= limit.Burst {
            return false, nil // Rate limit exceeded
        }

        // Allow burst request
        rl.redis.Incr(context.Background(), burstKey)
        rl.redis.Expire(context.Background(), burstKey, time.Minute)
    }

    return true, nil
}

// DDoS Protection Middleware
func (rl *RateLimiter) DDoSProtectionMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        clientIP := getClientIP(r)
        userAgent := r.Header.Get("User-Agent")

        // Check IP-based rate limiting
        allowed, err := rl.CheckRateLimit(clientIP, "global")
        if err != nil {
            http.Error(w, "Internal Server Error", http.StatusInternalServerError)
            return
        }

        if !allowed {
            // Log potential DDoS attack
            log.Printf("Rate limit exceeded for IP: %s, User-Agent: %s", clientIP, userAgent)

            // Add security headers
            w.Header().Set("Retry-After", "60")
            w.Header().Set("X-RateLimit-Limit", "100")
            w.Header().Set("X-RateLimit-Remaining", "0")

            http.Error(w, "Too Many Requests", http.StatusTooManyRequests)
            return
        }

        // Add rate limit headers
        remaining, _ := rl.getRemainingRequests(clientIP, "global")
        w.Header().Set("X-RateLimit-Limit", "100")
        w.Header().Set("X-RateLimit-Remaining", fmt.Sprintf("%d", remaining))

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

6. Secure Configuration Management

Configuration Security Framework

```go
type SecureConfig struct {
    encrypted    map[string]string
    plaintext    map[string]interface{}
    keyService   *KeyManagementService
    validator    *ConfigValidator
}

func LoadSecureConfiguration(configPath string, keyService *KeyManagementService) (*SecureConfig, error) {
    configData, err := ioutil.ReadFile(configPath)
    if err != nil {
        return nil, err
    }

    var rawConfig map[string]interface{}
    if err := json.Unmarshal(configData, &rawConfig); err != nil {
        return nil, err
    }

    config := &SecureConfig{
        encrypted:  make(map[string]string),
        plaintext:  make(map[string]interface{}),
        keyService: keyService,
        validator:  NewConfigValidator(),
    }

    // Process configuration fields
    for key, value := range rawConfig {
        if isSecretField(key) {
            // Decrypt secret fields
            decrypted, err := keyService.Decrypt(value.(string))
            if err != nil {
                return nil, fmt.Errorf("failed to decrypt %s: %v", key, err)
            }
            config.encrypted[key] = decrypted
        } else {
            config.plaintext[key] = value
        }
    }

    // Validate configuration
    if err := config.validator.Validate(config); err != nil {
        return nil, fmt.Errorf("configuration validation failed: %v", err)
    }

    return config, nil
}

// Environment-Based Security Profiles
type SecurityProfile struct {
    Environment       string              `json:"environment"`
    TLSMinVersion     uint16             `json:"tls_min_version"`
    RequireClientCert bool               `json:"require_client_cert"`
    SessionTimeout    time.Duration      `json:"session_timeout"`
    PasswordPolicy    PasswordPolicy     `json:"password_policy"`
    AuditLevel        string             `json:"audit_level"`
    ComplianceMode    bool               `json:"compliance_mode"`
    EncryptionStrength string            `json:"encryption_strength"`
}

func GetSecurityProfile(environment string) *SecurityProfile {
    profiles := map[string]*SecurityProfile{
        "production": {
            Environment:       "production",
            TLSMinVersion:     tls.VersionTLS13,
            RequireClientCert: true,
            SessionTimeout:    time.Hour * 8,
            PasswordPolicy:    StrictPasswordPolicy,
            AuditLevel:        "FULL",
            ComplianceMode:    true,
            EncryptionStrength: "AES-256",
        },
        "development": {
            Environment:       "development",
            TLSMinVersion:     tls.VersionTLS12,
            RequireClientCert: false,
            SessionTimeout:    time.Hour * 24,
            PasswordPolicy:    BasicPasswordPolicy,
            AuditLevel:        "BASIC",
            ComplianceMode:    false,
            EncryptionStrength: "AES-128",
        },
    }

    if profile, exists := profiles[environment]; exists {
        return profile
    }

    // Default to production profile for unknown environments
    return profiles["production"]
}
```

Document Version: 2.0
Last Updated: October 6, 2025
Author: Development Team
Classification: Confidential - Internal Use Only