Zona Custom Utils Service - Low-Level Design (LLD)

Table of Contents

Technical Implementation Overview

Technology Stack Details

Core Technologies

```
// Go 1.17+ with essential libraries
module zona_services/zona_custom_utils

require (
    github.com/gorilla/mux v1.8.0           // HTTP routing
    go.mongodb.org/mongo-driver v1.14.0     // MongoDB driver
    github.com/go-redis/redis/v8 v8.11.5    // Redis client
    github.com/docker/docker v1.13.0        // Docker API client
    github.com/dgrijalva/jwt-go v3.2.0       // JWT token handling
    securaa_lib v0.0.0                      // Internal utilities
)
```

Service Architecture

- **Port**: 8109 (HTTP REST API)
- **Runtime**: Go 1.17+ on Ubuntu 20.04
- **Container**: Docker with multi-stage build
- **Dependencies**: MongoDB, Redis, Docker Engine

Application Structure

```
zona_custom_utils/
├── main.go                 # Entry point
├── app.go                  # Application setup
├── Dockerfile             # Container definition
├── Makefile               # Build automation
├── go.mod                 # Module dependencies
├── controllers/           # HTTP request handlers
├── services/              # Business logic
├── models/                # Data structures
├── utils/                 # Utility functions
├── builders/              # Object builders
├── handlers/              # Error handlers
└── constants/             # Application constants
```

Detailed Component Design

Application Layer Implementation

main.go - Service Entry Point

```
package main

import "securaa_lib/securaalog"

var logger = securaalog.New("zona_custom_utils")

func main() {
    // Initialize logging subsystem
    securaalog.Init("CORE_SERVICE_LOGS")

    // Create and initialize application
    a := App{}
    a.Initialize()

    // Start HTTP server on port 8109
    a.Run(":8109")
}
```

app.go - Application Configuration

```
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
}

func (a *App) Initialize() {
    a.AccessTokenHashMap = make(map[string]int64)
    a.InitConfig()                    // Load configuration
    a.InitMongoSession(a.ConfigObject) // Setup databases
    a.Router = mux.NewRouter()
    a.initializeRoutes()              // Setup API routes
    go cache.CacheHealthCheck("test") // Start cache monitoring
    a.ClearCache()                    // Clear startup cache
}
```

Controller Layer Implementation

CustomUtilsController Structure

```
type CustomUtilsController struct{}

func NewCustomUtilsController() *CustomUtilsController {
    return &CustomUtilsController{}
}

// Primary endpoints implementation
func (cu *CustomUtilsController) UpsertCustomUtils(w http.ResponseWriter, r *http.Request,
    dbSession map[string]common.SessionStruct, configObject config.ConfigStruct) {

    response := services.CreateOrUpdateUtils(r, dbSession, configObject)
    utils.RespondWithJSON(w, http.StatusOK, response)
}

func (cu *CustomUtilsController) GetCustomUtils(w http.ResponseWriter, r *http.Request,
    dbSession map[string]common.SessionStruct, configObject config.ConfigStruct) {

    // Extract path parameters
    vars := mux.Vars(r)
    tenantCode := vars["tenant-code"]
    utilsName := vars["name"]

    // Process request and return response
    // ... implementation details
}
```

Request Routing Configuration

```
func (a *App) initializeRoutes() {
    a.Router.Use(a.loggingMiddleware)

    cu := controllers.NewCustomUtilsController()
    vc := controllers.NewValidateController()
    tc := controllers.NewTasksController()
    cc := controllers.NewContentController()

    // Custom Utils Management
    a.Router.HandleFunc("/utils/v1/custom-utils/{tenant-code}/{userid}/{name}",
        func(w http.ResponseWriter, r *http.Request) {
            cu.GetCustomUtils(w, r, a.DBSession, a.ConfigObject)
        }).Methods("GET")

    a.Router.HandleFunc("/utils/v1/custom-utils/{tenant-code}/{task-id}",
        func(w http.ResponseWriter, r *http.Request) {
            cu.GetCustomUtilsOnId(w, r, a.DBSession, a.ConfigObject)
        }).Methods("GET")

    a.Router.HandleFunc("/utils/v1/custom-utils",
        func(w http.ResponseWriter, r *http.Request) {
            cu.UpsertCustomUtils(w, r, a.DBSession, a.ConfigObject)
        }).Methods("POST")

    // Additional routes...
}
```

Service Layer Implementation

UtilsService - Core Business Logic

```
func CreateOrUpdateUtils(r *http.Request, dbSession map[string]common.SessionStruct,
    configObject config.ConfigStruct) models.ResponseModel {

    var response models.ResponseModel

    // Read configuration
    config, err := config.ReadConfig(configObject)
    if err != nil {
        return handlers.HandleError(err, "Unable to read configuration")
    }

    // Extract form parameters
    title := r.FormValue("title")
    description := r.FormValue("description")
    userid := r.FormValue("userid")
    tenantcode := r.FormValue("tenantcode")
    oldhash := r.FormValue("filehash")

    // Validate input parameters
    if strings.TrimSpace(title) == "" {
        response.Success = false
        response.DisplayMessage = "Utility name is required"
        return response
    }

    // File hash verification for updates
    if taskIdInt > 0 {
        filelocation := filepath.Join(pythonUtilDirectory, title+".py")
        currentHash, err := localUtils.CalculateFileHash(filelocation)
        if err != nil {
            return handlers.HandleError(err, "Unable to verify util.")
        }

        if oldhash != currentHash {
            return handlers.HandleError(errors.New("Invalid util version"),
                "Util has been altered by other user.")
        }
    }

    // Process utility creation/update
    // ... detailed implementation

    return response
}
```

TasksService - Task Management

```
func CopyutilsToTenant(newtenantcode string, dbSession map[string]common.SessionStruct,
    configObject config.ConfigStruct) (models.ResponseModel, error) {

    var response models.ResponseModel

    // Get configuration
    config, err := config.ReadConfig(configObject)
    if err != nil {
        return response, err
    }

    // Get tenant count to determine if this is first tenant
    mongoDbClient := dbSession["localhost"].Client
    mongoDbClient.CollectionName = config["tenantsCollection"]
    tenantsCount, err := mongoDbClient.GetDocumentsCount(
        bson.M{"status": "active"}, nil)

    if tenantsCount <= 1 {
        response.Success = true
        response.DisplayMessage = "Ignoring copy utils for first tenant"
        return response, nil
    }

    // Get source tenant utilities
    // Copy utilities to new tenant
    // ... implementation details

    return response, nil
}
```

Utils Layer Implementation

Common Utilities

```
package utils

import (
    "crypto/sha256"
    "encoding/hex"
    "io"
    "os"
)

func CalculateFileHash(filePath string) (string, error) {
    file, err := os.Open(filePath)
    if err != nil {
        return "", err
    }
    defer func() {
        if err := file.Close(); err != nil {
            logger.Error("Error in closing file ", err.Error())
        }
    }()

    hash := sha256.New()
    if _, err := io.Copy(hash, file); err != nil {
        return "", err
    }

    return hex.EncodeToString(hash.Sum(nil)), nil
}
```

Container Management

```
func RemoveCustomUtils(servicetype string) error {
    command := `docker service rm python-` + servicetype
    _, err := exec.Command("sh", "-c", command).CombinedOutput()
    return err
}

func CheckIfTrailContainerIsUp(imgtype string) bool {
    var command string
    if imgtype == "trial" {
        command = `docker ps -f name=python-utils-trial --format "{{.ID}}"`
    } else {
        command = `docker ps -f name=python-utils --format "{{.ID}}"`
    }

    opt, _ := exec.Command("sh", "-c", command).CombinedOutput()
    return strings.TrimSpace(string(opt)) != ""
}

func ClearProcess(userId string) error {
    existingPids, _ := GetAllPIDofProcessInContainer(userId)

    if len(existingPids) > 0 {
        pids := strings.Join(existingPids, " ")
        if pids != "" {
            commandString := "kill -TERM " + pids
            output, err := ExecuteCommandInContainer(commandString)
            if err != nil {
                logger.Error("Error in stopping running service: "+output, err.Error())
                return err
            }
        }
    }
    return nil
}
```

Builder Layer Implementation

TaskObjectBuilder

```
func BuildTaskObject(outputs string, title string, description string, userId int,
    utilsdirectory string, tenantcode string, port string, alteredFunctionName bool,
    inputLabel string, filterLabel string, filterRequired string, category string,
    localIp string) models.CustomUtilsObject {

    now := time.Now()
    currentTimeNanos := now.UnixNano()
    currentTimemillis := currentTimeNanos / 1000000
    savingObject := models.CustomUtilsObject{}

    if filterRequired != "true" {
        filterRequired = "false"
    }

    if alteredFunctionName {
        // Update existing utility
        savingObject.Name = title
        savingObject.Description = description
        savingObject.Updateddate = currentTimemillis
        savingObject.RestEnd = "/" + title + "/"
        savingObject.RestURL = "http://" + constants.PYTHON_UTILS_SERVICE + ":" + port + "/" + title
        savingObject.FunctionName = title + "()"
        savingObject.FileName = title + ".py"
        savingObject.Category = category
    } else {
        // Create new utility
        inputParams := `{"params":[{"name":"data", "order":1,"type":"string"},` +
                    `{"name":"filter", "order":2,"type":"string"}]}`
        inputFields := `{"inputfields":[{"name":"data","label":"` + inputLabel +
                    `","type":"textarea","id":"data","required":true,"value":"",` +
                    `"placeholder":"Enter ` + inputLabel + `"},` +
                    `{"name":"filter","label":"` + filterLabel +
                    `","type":"textbox","id":"filter","required":` + filterRequired +
                    `,"value":"","placeholder":"Enter data for ` + filterLabel + `"}]}`

        savingObject.Name = title
        savingObject.Description = description
        savingObject.Status = "active"
        savingObject.TaskHandler = "Utils"
        savingObject.TasksTag = "custom_utils_" + title
        savingObject.Createddate = currentTimemillis
        savingObject.Updateddate = currentTimemillis
        savingObject.Method = "POST"
        savingObject.CustomUtils = true
        savingObject.Type = "customutils"
        savingObject.TenantCode = tenantcode
        savingObject.UserID = userId
        savingObject.InputParams = inputParams
        savingObject.Fields = inputFields
        savingObject.FolderName = utilsdirectory
        savingObject.IntegrationID = -1
        savingObject.UtilsUsed = false
        savingObject.RestEnd = "/" + title + "/"
        savingObject.RestURL = "http://" + constants.PYTHON_UTILS_SERVICE + ":" + port + "/" + title
        savingObject.FunctionName = title + "()"
        savingObject.FileName = title + ".py"
        savingObject.Category = category
        savingObject.OutputFields = outputs
    }

    return savingObject
}
```

Database Design

MongoDB Collections Schema

tasks Collection (Custom Utils)

```
{
  "_id": ObjectId("..."),
  "name": "data_transformer",           // String: Unique utility name
  "id": 1001,                           // Int32: Auto-generated ID
  "displayname": "Data Transformer",    // String: Human-readable name
  "description": "Transform data",      // String: Utility description
  "type": "customutils",                // String: Always "customutils"
  "fields": "{\"inputfields\":[...]}",  // String: JSON-encoded input fields
  "createddate": NumberLong("1640995200000"), // Int64: Creation timestamp
  "updateddate": NumberLong("1640995200000"), // Int64: Update timestamp
  "status": "active",                   // String: active/draft/inactive
  "tasks_tag": "custom_utils_data_transformer", // String: Execution tag
  "task_handler": "Utils",              // String: Handler type
  "rest_url": "http://python-utils:8110/data_transformer", // String: Service URL
  "method": "POST",                     // String: HTTP method
  "tenantcode": "tenant001",            // String: Tenant identifier
  "filename": "data_transformer.py",    // String: Python file name
  "packages": ["pandas", "numpy"],      // Array: Required packages
  "port": "8110",                       // String: Service port
  "foldername": "/opt/zona/custom_utils/python/", // String: Storage path
  "userid": NumberLong("1001"),         // Int64: Creator user ID
  "customutils": true,                  // Boolean: Custom utility flag
  "integration_id": -1,                 // Int32: Integration reference
  "utils_used": false,                  // Boolean: Usage flag
  "rest_end": "/data_transformer/",     // String: Endpoint suffix
  "input_params": "{\"params\":[...]}",// String: Input parameters JSON
  "parameters": "",                     // String: Runtime parameters
  "functionname": "data_transformer()",// String: Function name
  "outputfields": "{\"outputs\":[...]}",// String: Output definition JSON
  "playbook_name": [],                  // Array: Associated playbooks
  "category": "Data Processing",        // String: Utility category
  "filecontents": BinData(0, "...")     // Binary: Base64 encoded file content
}
```

task_execution Collection

```
{
  "_id": ObjectId("..."),
  "taskid": NumberLong("1001"),         // Int64: Task reference
  "userid": NumberLong("1001"),         // Int64: User ID
  "tenantcode": "tenant001",            // String: Tenant code
  "createddate": NumberLong("1640995200000"), // Int64: Execution timestamp
  "request": "{\"data\":\"...\"}",      // String: JSON request data
  "response": "{\"result\":\"...\"}",   // String: JSON response data
  "status": "completed",                // String: Execution status
  "execution_time": 1.234,              // Double: Duration in seconds
  "error_message": "",                  // String: Error details if any
  "logs": ["INFO: Starting...", "..."], // Array: Execution logs
  "username": "john_doe",               // String: Executor name
  "name": "data_transformer",           // String: Utility name
  "task_handler": "Utils"               // String: Handler type
}
```

Database Indexes

```
// Performance indexes for tasks collection
db.tasks.createIndex({
    "name": 1,
    "type": 1,
    "status": 1
}, {
    "name": "idx_name_type_status"
})

db.tasks.createIndex({
    "tenantcode": 1,
    "userid": 1
}, {
    "name": "idx_tenant_user"
})

db.tasks.createIndex({
    "type": 1,
    "status": 1,
    "updateddate": -1
}, {
    "name": "idx_type_status_updated"
})

// Indexes for task_execution collection
db.task_execution.createIndex({
    "taskid": 1,
    "userid": 1,
    "createddate": -1
}, {
    "name": "idx_task_user_created"
})

db.task_execution.createIndex({
    "tenantcode": 1,
    "createddate": -1
}, {
    "name": "idx_tenant_created"
})
```

Data Model Implementation

Core Models

```
type CustomUtilsObject struct {
    Name                string           `json:"name" bson:"name"`
    ID                  int              `json:"id" bson:"id"`
    Displayname         string           `json:"displayname" bson:"displayname"`
    Description         string           `json:"description" bson:"description"`
    Type                string           `json:"type" bson:"type"`
    Fields              string           `json:"fields" bson:"fields"`
    Createddate         int64            `json:"createddate" bson:"createddate"`
    Updateddate         int64            `json:"updateddate" bson:"updateddate"`
    Status              string           `json:"status" bson:"status"`
    TasksTag            string           `json:"tasks_tag" bson:"tasks_tag"`
    TaskHandler         string           `json:"task_handler" bson:"task_handler"`
    RestURL             string           `json:"rest_url" bson:"rest_url"`
    Method              string           `json:"method" bson:"method"`
    TenantCode          string           `json:"tenantcode" bson:"tenantcode"`
    FileName            string           `json:"filename" bson:"filename"`
    Packages            []string         `json:"packages" bson:"packages"`
    Port                string           `json:"port" bson:"port"`
    FolderName          string           `json:"foldername" bson:"foldername"`
    UserID              int              `json:"userid" bson:"userid"`
    CustomUtils         bool             `json:"customutils" bson:"customutils"`
    IntegrationID       int              `json:"integration_id" bson:"integration_id"`
    UtilsUsed           bool             `json:"utils_used" bson:"utils_used"`
    RestEnd             string           `json:"rest_end" bson:"rest_end"`
    InputParams         string           `json:"input_params" bson:"input_params"`
    Parameters          string           `json:"parameters" bson:"parameters"`
    FunctionName        string           `json:"functionname" bson:"functionname"`
    OutputFields        string           `json:"outputfields" bson:"outputfields"`
    PlaybookNames       []string         `json:"playbook_name" bson:"playbook_name"`
    Category            string           `json:"category" bson:"category"`
    FileContent         primitive.Binary `json:"filecontents" bson:"filecontents"`
    FileContent_InString string          `json:"filecontents_instring"`
}

type ResponseModel struct {
    Success        bool        `json:"success"`
    Data           interface{} `json:"data"`
    Error          string      `json:"error"`
    Time           int64       `json:"time"`
    DisplayMessage string      `json:"displaymessage" bson:"displaymessage"`
    Status         string      `json:"status"`
    ErrorPath      string      `json:"errorpath"`
    SessionExpired bool        `json:"sessionexpired"`
    Field          []string    `json:"field"`
    Count          int         `json:"count"`
}
```

API Implementation

REST Endpoint Implementation

Create/Update Custom Utility

```
POST /utils/v1/custom-utils
Content-Type: multipart/form-data

Request Parameters:
- title: string (required)
- description: string (required)
- userid: integer (required)
- tenantcode: string (required)
- input_label: string (required)
- filter_label: string (required)
- required_filter: boolean
- category: string (required)
- filehash: string (for updates)
- username: string (for audit)
- email: string (for audit)
- output: string (output definitions)
- taskid: integer (for updates)
- oldtitle: string (for renames)
- code: file (Python code file)
```

Implementation Example

```
func (cu *CustomUtilsController) UpsertCustomUtils(w http.ResponseWriter, r *http.Request,
    dbSession map[string]common.SessionStruct, configObject config.ConfigStruct) {

    // Parse multipart form
    err := r.ParseMultipartForm(32 << 20) // 32MB limit
    if err != nil {
        utils.RespondWithJSON(w, http.StatusBadRequest,
            handlers.HandleError(err, "Invalid form data"))
        return
    }

    // Validate required fields
    title := r.FormValue("title")
    if strings.TrimSpace(title) == "" {
        utils.RespondWithJSON(w, http.StatusBadRequest,
            handlers.HandleError(errors.New("missing title"), "Utility name is required"))
        return
    }

    // Process request
    response := services.CreateOrUpdateUtils(r, dbSession, configObject)
    utils.RespondWithJSON(w, http.StatusOK, response)
}
```

Get Custom Utilities

```
func (cu *CustomUtilsController) GetCustomUtils(w http.ResponseWriter, r *http.Request,
    dbSession map[string]common.SessionStruct, configObject config.ConfigStruct) {

    response := models.ResponseModel{}
    vars := mux.Vars(r)

    tenantCode := vars["tenant-code"]
    utilsName := vars["name"]

    // Get database client for tenant
    mongoDbClient := dbSession[tenantCode].Client
    if mongoDbClient.Client == nil {
        // Create new connection if not exists
        coreMongoDbClient := dbSession["localhost"].Client
        mongoDbClient, err = utils.GetTenantDBClient(tenantCode, coreMongoDbClient, configObject)
        if err != nil {
            utils.RespondWithJSON(w, http.StatusInternalServerError,
                handlers.HandleError(err, "Database connection failed"))
            return
        }

        sessionStruct := common.SessionStruct{}
        sessionStruct.Client = mongoDbClient
        dbSession[tenantCode] = sessionStruct
    }

    // Build query
    var filter bson.M
    if utilsName != "-" {
        filter = bson.M{
            "status": bson.M{"$in": []string{"active", "draft"}},
            "name": utilsName,
            "type": "customutils",
        }
    } else {
        filter = bson.M{
            "status": bson.M{"$in": []string{"active", "draft"}},
            "type": "customutils",
        }
    }

    // Execute query
    utilsData := []models.CustomUtilsResponse{}
    projection := bson.M{
        "name": 1, "id": 1, "displayname": 1, "description": 1, "type": 1,
        "createddate": 1, "updateddate": 1, "status": 1, "filename": 1,
        // ... other fields
    }

    options := options.Find().SetProjection(projection).SetSort(bson.D{{"updateddate", -1}})
    mongoDbClient.CollectionName = config["taskCollection"]
    err = mongoDbClient.FindMultipleDocuments(filter, options, &utilsData)

    if err != nil {
        utils.RespondWithJSON(w, http.StatusInternalServerError,
            handlers.HandleError(err, "Failed to retrieve utilities"))
        return
    }

    // Calculate file hashes
    for i := 0; i < len(utilsData); i++ {
        hash, err := localUtils.CalculateFileHash(
            config["customUtilsCodeDirectory"] + "python/" + utilsData[i].FileName)
        if err != nil {
            logger.Error("Unable to calculate file hash: ", err.Error())
            continue
        }
        utilsData[i].FileHash = hash
    }

    if len(utilsData) > 0 {
        response.Success = true
        response.Data = utilsData
        response.Count = len(utilsData)
    } else {
        response.Success = false
        response.DisplayMessage = "No utilities found"
    }
```

```
    utils.RespondWithJSON(w, http.StatusOK, response)
}
```

Middleware Implementation

Logging Middleware

```
func (a *App) loggingMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        middleWareData := common.MiddleWareData{}
        middleWareData.LogRequestResponse = a.RequestResponseLog
        middleWareData.DBSession = a.DBSession
        middleWareData.ConfigObject = a.ConfigObject
        middleWareData.AccessTokenHashMap = a.AccessTokenHashMap

        err := common.MiddleWareForMongoDriver(next, w, r,
            constants.SkipRoutes, constants.SkipURI, middleWareData)

        if err != nil {
            logger.Error("Error in Middleware: ", err.Error())
            utils.RespondWithJSON(w, http.StatusOK,
                common.HandleError(err, "Error in creating Tenant Session"))
            return
        }
    })
}
```

Security Implementation

Input Validation

```
func validateUtilityName(name string) error {
    if len(name) == 0 {
        return errors.New("utility name is required")
    }
    if len(name) > 50 {
        return errors.New("utility name too long (max 50 characters)")
    }

    // Check for valid characters (alphanumeric and underscore only)
    validName := regexp.MustCompile(`^[a-zA-Z0-9_]+$`)
    if !validName.MatchString(name) {
        return errors.New("utility name contains invalid characters")
    }

    // Check for reserved names
    reservedNames := []string{"main", "init", "test", "admin"}
    for _, reserved := range reservedNames {
        if strings.ToLower(name) == reserved {
            return errors.New("utility name is reserved")
        }
    }

    return nil
}
```

Code Validation Rules

```
type ValidationRule struct {
    Name     string
    Pattern  string
    Severity string
    Message  string
}

var PythonValidationRules = []ValidationRule{
    {
        Name:     "NoImportOS",
        Pattern:  `import\s+os|from\s+os\s+import`,
        Severity: "HIGH",
        Message:  "OS module imports are not allowed for security reasons",
    },
    {
        Name:     "NoSubprocess",
        Pattern:  `import\s+subprocess|from\s+subprocess\s+import`,
        Severity: "HIGH",
        Message:  "Subprocess module is not allowed",
    },
    {
        Name:     "NoFileSystem",
        Pattern:  `open\s*\(|file\s*\(`,
        Severity: "MEDIUM",
        Message:  "Direct file operations should be reviewed",
    },
    {
        Name:     "NoNetwork",
        Pattern:  `socket\.|urllib\.|requests\.`,
        Severity: "MEDIUM",
        Message:  "Network operations should be reviewed",
    },
    {
        Name:     "NoExec",
        Pattern:  `exec\s*\(|eval\s*\(`,
        Severity: "HIGH",
        Message:  "Dynamic code execution is not allowed",
    },
}

func ValidateCode(code string) []ValidationResult {
    var results []ValidationResult

    for _, rule := range PythonValidationRules {
        matched, _ := regexp.MatchString(rule.Pattern, code)
        if matched {
            results = append(results, ValidationResult{
                Rule:     rule.Name,
                Severity: rule.Severity,
                Message:  rule.Message,
                Line:     findLineNumber(code, rule.Pattern),
            })
        }
    }

    return results
}
```

Encryption Implementation

```
func EncryptFileContent(content string, key []byte) (string, error) {
    block, err := aes.NewCipher(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(content), nil)
    return base64.StdEncoding.EncodeToString(ciphertext), nil
}

func DecryptFileContent(encryptedContent string, key []byte) (string, error) {
    data, err := base64.StdEncoding.DecodeString(encryptedContent)
    if err != nil {
        return "", err
    }

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

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

    nonceSize := gcm.NonceSize()
    if len(data) < nonceSize {
        return "", errors.New("ciphertext too short")
    }

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

    return string(plaintext), nil
}
```

Performance Implementation

Caching Strategy

```
type CacheManager struct {
    redisClient *redis.Client
    localCache  map[string]interface{}
    mutex       sync.RWMutex
}

func (cm *CacheManager) Get(key string) (interface{}, bool) {
    // Try L1 cache first (local memory)
    cm.mutex.RLock()
    if value, exists := cm.localCache[key]; exists {
        cm.mutex.RUnlock()
        return value, true
    }
    cm.mutex.RUnlock()

    // Try L2 cache (Redis)
    value, err := cm.redisClient.Get(context.Background(), key).Result()
    if err == nil {
        // Store in L1 cache for faster access
        cm.mutex.Lock()
        cm.localCache[key] = value
        cm.mutex.Unlock()
        return value, true
    }

    return nil, false
}

func (cm *CacheManager) Set(key string, value interface{}, ttl time.Duration) error {
    // Store in both L1 and L2 cache
    cm.mutex.Lock()
    cm.localCache[key] = value
    cm.mutex.Unlock()

    // Store in Redis with TTL
    return cm.redisClient.Set(context.Background(), key, value, ttl).Err()
}
```

Database Connection Pooling

```
func InitMongoClient(configObject config.ConfigStruct) (*mongo.Client, error) {
    clientOptions := options.Client().ApplyURI(mongoURI)

    // Connection pool settings
    clientOptions.SetMaxPoolSize(50)           // Maximum connections
    clientOptions.SetMinPoolSize(5)            // Minimum connections
    clientOptions.SetMaxConnIdleTime(30 * time.Minute) // Idle timeout
    clientOptions.SetConnectTimeout(10 * time.Second)  // Connection timeout
    clientOptions.SetSocketTimeout(30 * time.Second)   // Socket timeout

    // Create client
    client, err := mongo.Connect(context.Background(), clientOptions)
    if err != nil {
        return nil, err
    }

    // Verify connection
    err = client.Ping(context.Background(), nil)
    if err != nil {
        return nil, err
    }

    return client, nil
}
```

Query Optimization

```
func GetUtilitiesOptimized(tenantCode string, userID int, limit int, offset int) ([]models.CustomUtilsResponse, error) {
    // Use aggregation pipeline for complex queries
    pipeline := []bson.M{
        // Match stage - filter documents
        {
            "$match": bson.M{
                "tenantcode": tenantCode,
                "userid":     userID,
                "type":       "customutils",
                "status":     bson.M{"$in": []string{"active", "draft"}},
            },
        },
        // Project stage - select only needed fields
        {
            "$project": bson.M{
                "name":        1,
                "displayname": 1,
                "description": 1,
                "status":      1,
                "createddate": 1,
                "updateddate": 1,
                "category":    1,
                "_id":         0,
            },
        },
        // Sort stage - order by update date
        {
            "$sort": bson.M{
                "updateddate": -1,
            },
        },
        // Skip stage - for pagination
        {
            "$skip": offset,
        },
        // Limit stage - limit results
        {
            "$limit": limit,
        },
    }

    cursor, err := collection.Aggregate(context.Background(), pipeline)
    if err != nil {
        return nil, err
    }
    defer cursor.Close(context.Background())

    var results []models.CustomUtilsResponse
    if err = cursor.All(context.Background(), &results); err != nil {
        return nil, err
    }

    return results, nil
}
```

Error Handling Implementation

Error Types and Handling

```
type ErrorType string

const (
    ValidationError ErrorType = "VALIDATION_ERROR"
    DatabaseError   ErrorType = "DATABASE_ERROR"
    FileSystemError ErrorType = "FILESYSTEM_ERROR"
    SecurityError   ErrorType = "SECURITY_ERROR"
    ContainerError  ErrorType = "CONTAINER_ERROR"
    NetworkError    ErrorType = "NETWORK_ERROR"
)

type CustomError struct {
    Type        ErrorType `json:"type"`
    Code        string    `json:"code"`
    Message     string    `json:"message"`
    Details     string    `json:"details"`
    Timestamp   time.Time `json:"timestamp"`
    RequestID   string    `json:"request_id"`
    UserMessage string    `json:"user_message"`
}

func (e *CustomError) Error() string {
    return fmt.Sprintf("[%s] %s: %s", e.Code, e.Message, e.Details)
}

func NewValidationError(code, message, details string) *CustomError {
    return &CustomError{
        Type:        ValidationError,
        Code:        code,
        Message:     message,
        Details:     details,
        Timestamp:   time.Now(),
        UserMessage: "Please check your input and try again",
    }
}
```

Error Handler Implementation

```
func HandleError(err error, userMessage string) models.ResponseModel {
    response := models.ResponseModel{
        Success:        false,
        Time:           time.Now().UnixNano() / 1000000,
        DisplayMessage: userMessage,
    }

    if err == nil {
        return response
    }

    // Log the error
    logger.Error("Error occurred: ", err.Error())

    // Determine error type and set appropriate response
    switch e := err.(type) {
    case *CustomError:
        response.Error = e.Code
        response.ErrorPath = e.Type
        response.DisplayMessage = e.UserMessage

    case *mongo.WriteError:
        if e.Code == 11000 { // Duplicate key error
            response.Error = "DUPLICATE_ENTRY"
            response.DisplayMessage = "A utility with this name already exists"
        } else {
            response.Error = "DATABASE_ERROR"
            response.DisplayMessage = "Database operation failed"
        }

    default:
        response.Error = "INTERNAL_ERROR"
        response.DisplayMessage = userMessage
    }

    return response
}
```

Panic Recovery Middleware

```
func RecoveryMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        defer func() {
            if err := recover(); err != nil {
                logger.Error("Panic recovered: ", err)

                // Log stack trace
                debug.PrintStack()

                // Return error response
                response := models.ResponseModel{
                    Success:        false,
                    Error:          "INTERNAL_ERROR",
                    DisplayMessage: "An unexpected error occurred",
                    Time:           time.Now().UnixNano() / 1000000,
                }

                w.Header().Set("Content-Type", "application/json")
                w.WriteHeader(http.StatusInternalServerError)
                json.NewEncoder(w).Encode(response)
            }
        }()

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

Monitoring and Logging Implementation

Structured Logging

```
type LogEntry struct {
    Timestamp   time.Time              `json:"timestamp"`
    Level       string                `json:"level"`
    Service     string                `json:"service"`
    Component   string                `json:"component"`
    Function    string                `json:"function"`
    Message     string                `json:"message"`
    TenantCode  string                `json:"tenant_code,omitempty"`
    UserID      string                `json:"user_id,omitempty"`
    RequestID   string                `json:"request_id,omitempty"`
    Duration    float64               `json:"duration_ms,omitempty"`
    Metadata    map[string]interface{} `json:"metadata,omitempty"`
}

func LogRequest(r *http.Request, duration time.Duration, statusCode int) {
    entry := LogEntry{
        Timestamp: time.Now(),
        Level:     "INFO",
        Service:   "zona-custom-utils",
        Component: "HTTP",
        Function:  "RequestHandler",
        Message:   "HTTP request processed",
        RequestID: getRequestID(r),
        Duration:  float64(duration.Nanoseconds()) / 1000000,
        Metadata: map[string]interface{}{
            "method":      r.Method,
            "path":        r.URL.Path,
            "status_code": statusCode,
            "user_agent":  r.UserAgent(),
            "remote_addr": r.RemoteAddr,
        },
    }

    logJSON, _ := json.Marshal(entry)
    logger.Info(string(logJSON))
}
```

Metrics Collection

```
var (
    requestDuration = prometheus.NewHistogramVec(
        prometheus.HistogramOpts{
            Name: "http_request_duration_seconds",
            Help: "HTTP request duration in seconds",
            Buckets: prometheus.DefBuckets,
        },
        []string{"method", "endpoint", "status"},
    )

    activeConnections = prometheus.NewGauge(
        prometheus.GaugeOpts{
            Name: "active_database_connections",
            Help: "Number of active database connections",
        },
    )

    utilityExecutions = prometheus.NewCounterVec(
        prometheus.CounterOpts{
            Name: "utility_executions_total",
            Help: "Total number of utility executions",
        },
        []string{"tenant", "utility", "status"},
    )
)

func init() {
    prometheus.MustRegister(requestDuration)
    prometheus.MustRegister(activeConnections)
    prometheus.MustRegister(utilityExecutions)
}

func MetricsMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        start := time.Now()

        wrapped := &statusWriter{ResponseWriter: w, statusCode: http.StatusOK}
        next.ServeHTTP(wrapped, r)

        duration := time.Since(start)
        requestDuration.WithLabelValues(
            r.Method,
            r.URL.Path,
            strconv.Itoa(wrapped.statusCode),
        ).Observe(duration.Seconds())
    })
}
```

Health Check Implementation

```
type HealthCheck struct {
    Status    string                 `json:"status"`
    Timestamp time.Time              `json:"timestamp"`
    Version   string                 `json:"version"`
    Uptime    string                 `json:"uptime"`
    Checks    map[string]CheckResult `json:"checks"`
}

type CheckResult struct {
    Status  string        `json:"status"`
    Message string        `json:"message,omitempty"`
    Latency time.Duration `json:"latency_ms"`
}

func HealthCheckHandler(w http.ResponseWriter, r *http.Request) {
    health := HealthCheck{
        Status:    "healthy",
        Timestamp: time.Now(),
        Version:   version,
        Uptime:    time.Since(startTime).String(),
        Checks:    make(map[string]CheckResult),
    }

    // Database health check
    dbStart := time.Now()
    if err := checkDatabaseHealth(); err != nil {
        health.Checks["database"] = CheckResult{
            Status:  "unhealthy",
            Message: err.Error(),
            Latency: time.Since(dbStart),
        }
        health.Status = "unhealthy"
    } else {
        health.Checks["database"] = CheckResult{
            Status:  "healthy",
            Latency: time.Since(dbStart),
        }
    }

    // Cache health check
    cacheStart := time.Now()
    if err := checkCacheHealth(); err != nil {
        health.Checks["cache"] = CheckResult{
            Status:  "unhealthy",
            Message: err.Error(),
            Latency: time.Since(cacheStart),
        }
        health.Status = "degraded"
    } else {
        health.Checks["cache"] = CheckResult{
            Status:  "healthy",
            Latency: time.Since(cacheStart),
        }
    }

    // Docker health check
    dockerStart := time.Now()
    if err := checkDockerHealth(); err != nil {
        health.Checks["docker"] = CheckResult{
            Status:  "unhealthy",
            Message: err.Error(),
            Latency: time.Since(dockerStart),
        }
        health.Status = "degraded"
    } else {
        health.Checks["docker"] = CheckResult{
            Status:  "healthy",
            Latency: time.Since(dockerStart),
        }
    }

    // Set response status
    statusCode := http.StatusOK
    if health.Status == "unhealthy" {
        statusCode = http.StatusServiceUnavailable
    }

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

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

Configuration Management

Configuration Structure

```
type Config struct {
    Server   ServerConfig   `json:"server"`
    Database DatabaseConfig `json:"database"`
    Cache    CacheConfig    `json:"cache"`
    Security SecurityConfig `json:"security"`
    Storage  StorageConfig  `json:"storage"`
    Docker   DockerConfig   `json:"docker"`
    Logging  LoggingConfig  `json:"logging"`
}

type ServerConfig struct {
    Port                int    `json:"port"`
    ReadTimeout         int    `json:"read_timeout"`
    WriteTimeout        int    `json:"write_timeout"`
    IdleTimeout         int    `json:"idle_timeout"`
    MaxHeaderBytes      int    `json:"max_header_bytes"`
    LogRequestResponse  bool   `json:"log_request_response"`
}

type DatabaseConfig struct {
    Host            string `json:"host"`
    Port            int    `json:"port"`
    Database        string `json:"database"`
    Username        string `json:"username"`
    Password        string `json:"password"`
    MaxPoolSize     int    `json:"max_pool_size"`
    MinPoolSize     int    `json:"min_pool_size"`
    ConnectTimeout  int    `json:"connect_timeout"`
    SocketTimeout   int    `json:"socket_timeout"`
}
```

Environment-based Configuration

```
func LoadConfig() (*Config, error) {
    config := &Config{}

    // Set defaults
    setDefaults(config)

    // Load from file if exists
    if configFile := os.Getenv("CONFIG_FILE"); configFile != "" {
        if err := loadFromFile(config, configFile); err != nil {
            return nil, err
        }
    }

    // Override with environment variables
    overrideWithEnv(config)

    // Validate configuration
    if err := validateConfig(config); err != nil {
        return nil, err
    }

    return config, nil
}

func overrideWithEnv(config *Config) {
    if port := os.Getenv("SERVER_PORT"); port != "" {
        if p, err := strconv.Atoi(port); err == nil {
            config.Server.Port = p
        }
    }

    if host := os.Getenv("DB_HOST"); host != "" {
        config.Database.Host = host
    }

    if password := os.Getenv("DB_PASSWORD"); password != "" {
        config.Database.Password = password
    }

    // ... additional environment variable overrides
}
```

Development Guidelines

Code Organization Standards

```
// Package structure guidelines
package controllers  // HTTP request handlers
package services     // Business logic implementation
package models       // Data structures and types
package utils        // Utility functions and helpers
package builders     // Object construction patterns
package handlers     // Error handling utilities
package constants    // Application constants

// Naming conventions
// Constants: UPPER_SNAKE_CASE
const MAX_FILE_SIZE = 10485760

// Functions: PascalCase for exported, camelCase for private
func CreateCustomUtils() {}
func validateInput() {}

// Structs: PascalCase
type CustomUtilsController struct{}

// Variables: camelCase
var logger = securaalog.New("zona_custom_utils")
```

Error Handling Patterns

```
// Always handle errors explicitly
result, err := someOperation()
if err != nil {
    logger.Error("Operation failed: ", err.Error())
    return handleError(err, "User-friendly message")
}

// Use structured error responses
func handleDatabaseError(err error) models.ResponseModel {
    return models.ResponseModel{
        Success:        false,
        Error:          "DATABASE_ERROR",
        DisplayMessage: "Database operation failed",
        ErrorPath:      "DatabaseLayer",
        Time:           time.Now().UnixNano() / 1000000,
    }
}
```

Security Best Practices

```
// Input validation example
func validateInput(input string) error {
    // Check for SQL injection patterns
    sqlPatterns := []string{"'", "--", "/*", "*/", "xp_", "sp_"}
    for _, pattern := range sqlPatterns {
        if strings.Contains(strings.ToLower(input), pattern) {
            return errors.New("potentially malicious input detected")
        }
    }

    // Length validation
    if len(input) > 1000 {
        return errors.New("input too long")
    }

    // Character validation
    if !regexp.MustCompile(`^[a-zA-Z0-9\s\-_\.]+$`).MatchString(input) {
        return errors.New("invalid characters in input")
    }

    return nil
}

// Secure file handling
func secureFileOperation(filename string) error {
    // Prevent path traversal
    if strings.Contains(filename, "..") {
        return errors.New("path traversal attempt detected")
    }

    // Validate file extension
    allowedExts := []string{".py", ".txt", ".json"}
    ext := filepath.Ext(filename)
    isAllowed := false
    for _, allowed := range allowedExts {
        if ext == allowed {
            isAllowed = true
            break
        }
    }

    if !isAllowed {
        return errors.New("file type not allowed")
    }

    return nil
}
```

Testing Strategy

Unit Test Implementation

```
func TestCreateCustomUtils(t *testing.T) {
    // Setup test environment
    mockDB := setupMockDatabase()
    mockConfig := setupMockConfig()
    service := services.NewUtilsService(mockDB, mockConfig)

    tests := []struct {
        name           string
        input          CreateUtilRequest
        expectedResult bool
        expectedError  string
    }{
        {
            name: "Valid utility creation",
            input: CreateUtilRequest{
                Title:       "test_utility",
                Description: "Test utility for unit testing",
                UserID:      1001,
                TenantCode:  "test_tenant",
                Code:        "def test_function():\n    return 'Hello World'",
            },
            expectedResult: true,
            expectedError:  "",
        },
        {
            name: "Invalid utility name",
            input: CreateUtilRequest{
                Title:       "",
                Description: "Test utility",
                UserID:      1001,
                TenantCode:  "test_tenant",
                Code:        "def test_function():\n    return 'Hello World'",
            },
            expectedResult: false,
            expectedError:  "utility name is required",
        },
        {
            name: "Malicious code detection",
            input: CreateUtilRequest{
                Title:       "malicious_utility",
                Description: "Malicious utility",
                UserID:      1001,
                TenantCode:  "test_tenant",
                Code:        "import os\nos.system('rm -rf /')",
            },
            expectedResult: false,
            expectedError:  "prohibited code patterns detected",
        },
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            result, err := service.CreateUtility(tt.input)

            assert.Equal(t, tt.expectedResult, result.Success)
            if tt.expectedError != "" {
                assert.Contains(t, err.Error(), tt.expectedError)
            }
        })
    }
}
```

Integration Test Example

```
func TestUtilityEndToEnd(t *testing.T) {
    // Setup test server
    testServer := httptest.NewServer(setupTestApp())
    defer testServer.Close()

    // Test utility creation
    createPayload := map[string]interface{}{
        "title":       "integration_test_utility",
        "description": "Integration test utility",
        "userid":      "1001",
        "tenantcode":  "test_tenant",
        "category":    "Testing",
    }

    createBody, _ := json.Marshal(createPayload)
    resp, err := http.Post(testServer.URL+"/utils/v1/custom-utils",
        "application/json", bytes.NewBuffer(createBody))

    assert.NoError(t, err)
    assert.Equal(t, http.StatusOK, resp.StatusCode)

    // Test utility retrieval
    getResp, err := http.Get(testServer.URL+"/utils/v1/custom-utils/test_tenant/1001/integration_test_utility")
    assert.NoError(t, err)
    assert.Equal(t, http.StatusOK, getResp.StatusCode)

    // Test utility deletion
    deletePayload := map[string]interface{}{
        "functionname": "integration_test_utility",
        "tenantcode":   "test_tenant",
        "userid":       "1001",
    }

    deleteBody, _ := json.Marshal(deletePayload)
    req, _ := http.NewRequest("DELETE", testServer.URL+"/utils/v1/custom-utils", bytes.NewBuffer(deleteBody))
    client := &http.Client{}
    deleteResp, err := client.Do(req)

    assert.NoError(t, err)
    assert.Equal(t, http.StatusOK, deleteResp.StatusCode)
}
```

Deployment Implementation

Docker Configuration

```
# Multi-stage build for optimal image size
FROM golang:1.17-alpine AS builder

# Install build dependencies
RUN apk --no-cache add make git bash ca-certificates curl

# Copy source code
COPY ./ /project
WORKDIR /project

# Set Go module mode
ENV GOFLAGS="-mod=vendor"

# Build stage
FROM builder AS build
ARG GIT_REF
ARG GIT_BRANCH
ARG BUILD_NUMBER
ARG BUILD_VERSION
ARG TARGET

# Production stage
FROM ubuntu:20.04

# Set build labels
LABEL gitBranch=$GIT_BRANCH \
      build=$BUILD_NUMBER \
      gitRef=$GIT_REF \
      version=$BUILD_VERSION

# Copy binary from build stage
COPY --from=build /project/build/$TARGET /$TARGET

# Create run script
RUN echo '#!/usr/bin/env sh' > run.sh \
  && echo 'exec' "/$TARGET" >> run.sh \
  && chmod +x run.sh

# Install runtime dependencies
RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates curl \
    gnupg python3-pip software-properties-common

# Install Docker CLI
RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" && \
    apt-get update && \
    apt-get install -y docker-ce-cli

# Create necessary directories
VOLUME ["/opt/zona/"]
RUN mkdir -p /opt/zona/logs

# Expose service port
EXPOSE 8109

# Set startup command
CMD ["/run.sh"]
```

CI/CD Pipeline Configuration

```
# .github/workflows/deploy.yml
name: Build and Deploy

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Set up Go
      uses: actions/setup-go@v2
      with:
        go-version: 1.17

    - name: Run tests
      run: |
        go test -v ./...
        go test -race -coverprofile=coverage.out ./...

    - name: Upload coverage
      uses: codecov/codecov-action@v1
      with:
        file: ./coverage.out

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Build Docker image
      run: |
        docker build -t zona/custom-utils:${{ github.sha }} .
        docker tag zona/custom-utils:${{ github.sha }} zona/custom-utils:latest

    - name: Push to registry
      run: |
        echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin
        docker push zona/custom-utils:${{ github.sha }}
        docker push zona/custom-utils:latest

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
    - name: Deploy to production
      run: |
        docker stack deploy -c docker-compose.yml zona-custom-utils
```

This Low-Level Design document provides comprehensive technical implementation details for the Zona Custom Utils Service, covering code structure, database implementation, API details, security measures, performance optimizations, error handling, monitoring, configuration management, development guidelines, testing strategies, and deployment procedures.