

Securaa Custom Utils - Low Level Design

Document Information

- **Service Name:** Securaa Custom Utils Service
- **Version:** 1.0
- **Date:** September 2025
- **Author:** Development Team
- **Related Documents:** [High Level Design](#)

Table of Contents

1. [Technical Implementation](#)
2. [Database Design](#)
3. [API Specifications](#)
4. [Class Design](#)
5. [Security Implementation](#)
6. [Performance Optimization](#)
7. [Error Handling](#)
8. [Deployment Configuration](#)
9. [Monitoring & Logging](#)

Technical Implementation

Technology Stack

- **Programming Language:** Python 3.9+
- **Web Framework:** FastAPI
- **Database:** MongoDB with Motor (async driver)
- **Cache:** Redis with aioredis
- **Container Runtime:** Docker
- **Authentication:** JWT with custom middleware
- **Validation:** Pydantic models
- **Testing:** pytest with async support

Project Structure

Copy

```
securaa_custom_utils/
├── app/
│   ├── __init__.py
│   ├── main.py          # FastAPI application entry point
│   ├── config.py        # Configuration management
│   └── dependencies.py # Dependency injection setup
|
├── api/
│   ├── __init__.py
│   ├── v1/
│   │   ├── __init__.py
│   │   ├── endpoints/
│   │   │   ├── __init__.py
│   │   │   ├── utils.py    # Utils management endpoints
│   │   │   └── execution.py # Code execution endpoints
│   │   └── health.py     # Health check endpoints
│   ├── api.py           # API router
│   └── deps.py          # API dependencies
|
├── core/
│   ├── __init__.py
│   ├── config.py        # Core configuration
│   ├── security.py      # Security utilities
│   ├── logging.py        # Logging configuration
│   └── exceptions.py   # Custom exceptions
|
├── models/
│   ├── __init__.py
│   ├── base.py          # Base Pydantic models
│   ├── utils.py          # Utils domain models
│   ├── execution.py     # Execution models
│   └── user.py          # User models
|
├── schemas/
│   ├── __init__.py
│   ├── utils.py          # Utils API schemas
│   ├── execution.py      # Execution API schemas
│   └── common.py         # Common response schemas
|
├── services/
│   ├── __init__.py
│   ├── utils_service.py  # Utils business logic
│   ├── execution_service.py # Execution business logic
│   ├── validation_service.py # Code validation service
│   ├── container_service.py # Container management
│   └── audit_service.py   # Audit logging service
|
└── repositories/
    ├── __init__.py
    └── base.py            # Base repository pattern
```

```
utils_repository.py # Utils data access
execution_repository.py # Execution data access
cache_repository.py # Cache operations

utils/
    __init__.py
    security.py      # Security utilities
    validation.py    # Code validation utilities
    container.py     # Container utilities
    file_manager.py  # File management utilities

middleware/
    __init__.py
    auth.py          # Authentication middleware
    tenant.py        # Multi-tenant middleware
    logging.py       # Request logging middleware

tests/
deployment/
docs/
requirements.txt
pyproject.toml
README.md
```

Database Design

MongoDB Collections

Custom Utils Collection

Copy

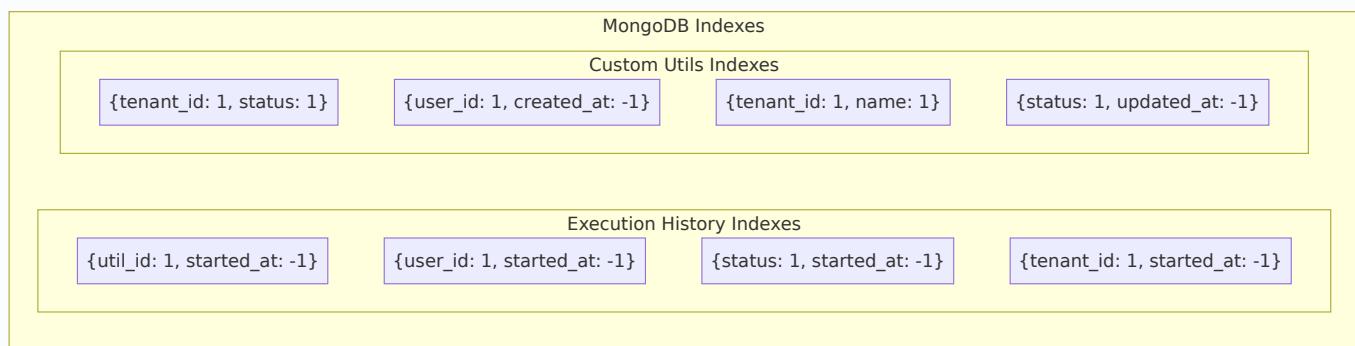
```
{
  "_id": "ObjectId",
  "util_id": "string (UUID)",
  "tenant_id": "string",
  "user_id": "string",
  "name": "string",
  "description": "string",
  "code": "string",
  "language": "python",
  "parameters": {
    "input_schema": {},
    "output_schema": {},
    "dependencies": []
  },
  "metadata": {
    "version": "string",
    "tags": []
  }
}
```

```

    "category": "string"
},
"validation": {
    "is_valid": "boolean",
    "validation_errors": [],
    "security_score": "number"
},
"execution_config": {
    "timeout_seconds": "number",
    "memory_limit_mb": "number",
    "cpu_limit": "number"
},
"status": "active|inactive|deleted",
"created_at": "datetime",
"updated_at": "datetime",
"created_by": "string",
"updated_by": "string"
}
}

```

Database Indexes



API Specifications

REST API Endpoints

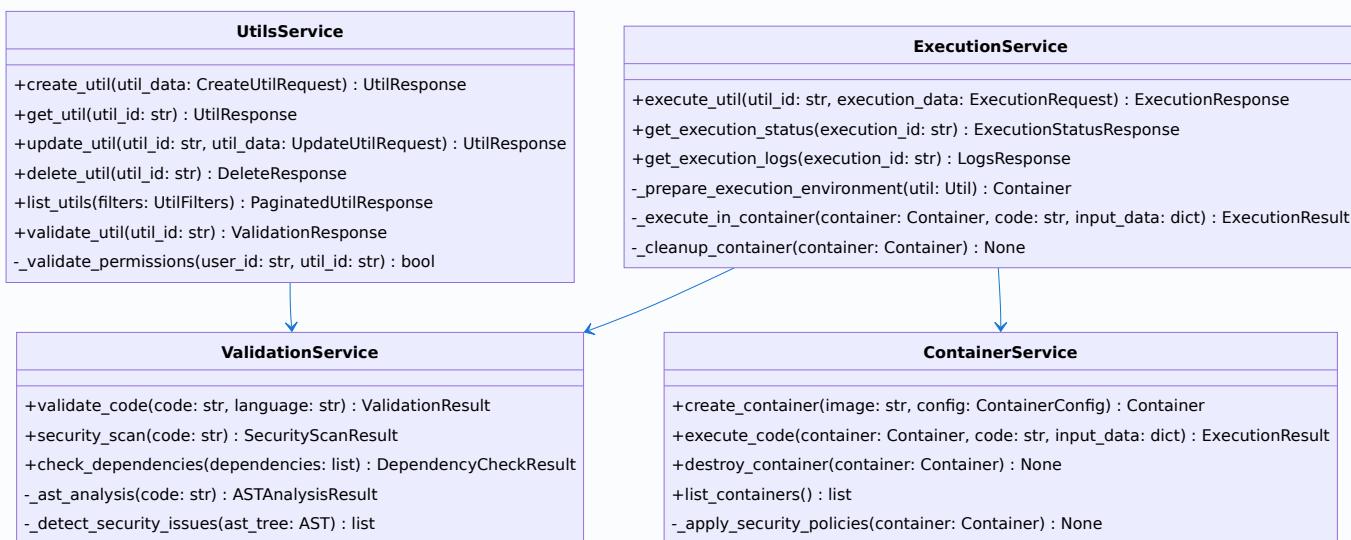
Utils Management APIs

Endpoint	Method	Description	Request Body	Response
/api/v1/utils	POST	Create custom utility	CreateUtilRequest	UtilResponse
/api/v1/utils	GET	List utilities (paginated)	Query parameters	PaginatedUtilResponse

Endpoint	Method	Description	Request Body	Response
/api/v1/utils/{util_id}	GET	Get utility details	None	UtilResponse
/api/v1/utils/{util_id}	PUT	Update utility	UpdateUtilRequest	UtilResponse
/api/v1/utils/{util_id}	DELETE	Delete utility	None	DeleteResponse
/api/v1/utils/{util_id}/execute	POST	Execute utility	ExecutionRequest	ExecutionResponse

Class Design

Core Service Classes



Security Implementation

Authentication & Authorization

JWT Authentication Middleware

```

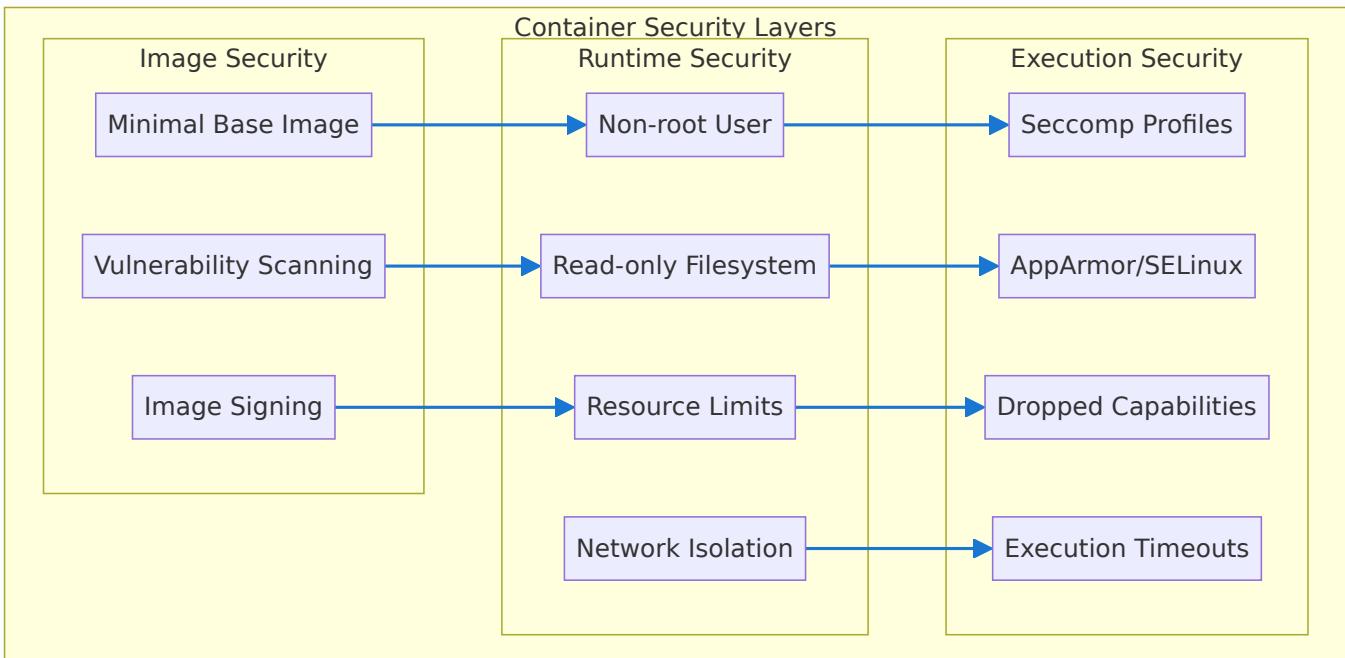
class JWTAuthMiddleware:
    def __init__(self, secret_key: str, algorithm: str = "HS256"):
        self.secret_key = secret_key
        self.algorithm = algorithm
        self.jwt_decoder = JWTDecoder(secret_key, algorithm)

    async def __call__(self, request: Request, call_next):
        # Skip authentication for health check endpoints
  
```

Copy

```
if request.url.path in SKIP_AUTH_PATHS:  
    return await call_next(request)  
  
# Extract JWT token from Authorization header  
auth_header = request.headers.get("Authorization")  
if not auth_header or not auth_header.startswith("Bearer "):  
    raise HTTPException(  
        status_code=401,  
        detail="Missing or invalid authorization header"  
    )  
  
token = auth_header.split(" ")[1]  
  
try:  
    # Decode and validate JWT token  
    payload = self.jwt_decoder.decode(token)  
  
    # Extract user and tenant information  
    user_id = payload.get("user_id")  
    tenant_id = payload.get("tenant_id")  
    permissions = payload.get("permissions", [])  
  
    # Add user context to request state  
    request.state.user_id = user_id  
    request.state.tenant_id = tenant_id  
    request.state.permissions = permissions  
  
except JWTError as e:  
    raise HTTPException(  
        status_code=401,  
        detail=f"Invalid token: {str(e)}"  
    )  
  
return await call_next(request)
```

Container Security



Performance Optimization

Caching Strategy Implementation

Multi-Level Cache

```

class CacheManager:
    def __init__(self, redis_client: Redis):
        self.redis = redis_client
        self.local_cache = {}
        self.cache_stats = CacheStats()

    @async def get(self, key: str, fetch_func: callable = None) -> any:
        """Multi-level cache get with fallback"""

        # L1: Check local cache first
        if key in self.local_cache:
            self.cache_stats.l1_hits += 1
            return self.local_cache[key]

        # L2: Check Redis cache
        redis_value = await self.redis.get(key)
        if redis_value:
            self.cache_stats.l2_hits += 1
            # Store in local cache for future requests
            self.local_cache[key] = json.loads(redis_value)
            return self.local_cache[key]

```

[Copy](#)

```

# L3: Fetch from source if fetch function provided
if fetch_func:
    self.cache_stats.cache_misses += 1
    value = await fetch_func()

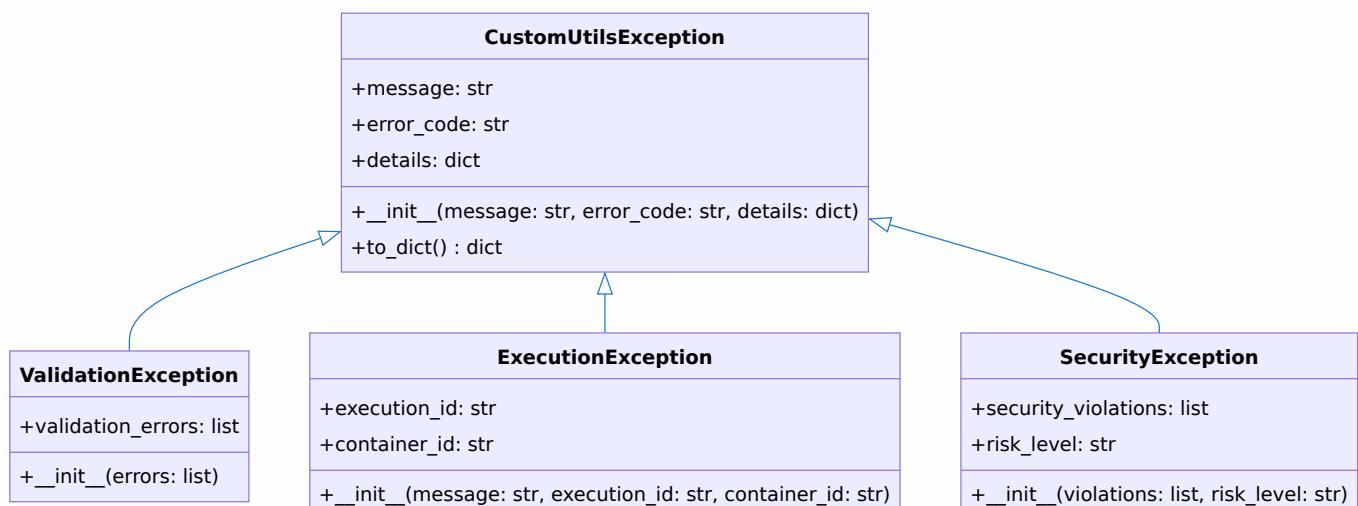
    # Store in both caches
    await self.set(key, value, ttl=300) # 5 minutes TTL
return value

return None

```

Error Handling

Exception Hierarchy



Deployment Configuration

Docker Configuration

Application Dockerfile

```

FROM python:3.9-slim as builder

# Install build dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*

```

```
# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Production stage
FROM python:3.9-slim

# Create non-root user
RUN groupadd -r appuser && useradd -r -g appuser appuser

# Install runtime dependencies
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Copy Python packages from builder stage
COPY --from=builder /usr/local/lib/python3.9/site-packages /usr/local/lib/python3.9/site-packages
COPY --from=builder /usr/local/bin /usr/local/bin

# Create app directory
WORKDIR /app

# Copy application code
COPY app/ ./app/
COPY deployment/scripts/ ./scripts/

# Set ownership and permissions
RUN chown -R appuser:appuser /app
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Expose port
EXPOSE 8000

# Start application
CMD ["python", "-m", "app.main"]
```

Monitoring & Logging

Metrics Collection

Application Metrics

Copy

```
from prometheus_client import Counter, Histogram, Gauge
```

```
# Define metrics
utils_created_total = Counter(
    "utils_created_total",
    "Total number of utilities created",
    ['tenant_id', 'user_id']
)

utils_executed_total = Counter(
    "utils_executed_total",
    "Total number of utility executions",
    ['tenant_id', 'util_id', 'status']
)

execution_duration_seconds = Histogram(
    "execution_duration_seconds",
    "Time spent executing utilities",
    ['tenant_id', 'util_id']
)

active_containers = Gauge(
    "active_containers",
    "Number of active execution containers"
)
```

Structured Logging

Log Format

Copy

```
{
  "timestamp": "2025-09-30T10:30:00.123Z",
  "level": "INFO",
  "service": "securaa-custom-utils",
  "version": "1.0.0",
  "logger": "app.services.execution_service",
  "message": "Utility execution completed successfully",
  "context": {
    "tenant_id": "tenant_123",
    "user_id": "user_456",
    "util_id": "util_789",
    "execution_id": "exec_101112",
    "execution_time_ms": 1250,
    "memory_used_mb": 45,
    "container_id": "container_abcd123"
  },
  "request_id": "req_987654321",
  "trace_id": "trace_555666777"
```

Conclusion

This low-level design provides a comprehensive technical implementation guide for the Securaa Custom Utils Service. The design emphasizes security, performance, and maintainability through:

- **Secure Architecture:** Multi-layered security with container isolation and code validation
- **Scalable Design:** Microservice architecture with horizontal scaling capabilities
- **Performance Optimization:** Multi-level caching and optimized database queries
- **Comprehensive Monitoring:** Detailed metrics collection and structured logging
- **Maintainable Codebase:** Clear separation of concerns and well-defined interfaces

The implementation follows industry best practices for microservice development, security, and DevOps, ensuring a production-ready solution that can scale with business requirements.