Sistem Lost and Found Kampus dengan Mekanisme Verifikasi Tertutup Berbasis Golang

Implementasi Clean Architecture, Concurrency, dan Database Transaction

1st Edward Wibisono Yulianto
Jurusan Informatika
Universitas Widya Mandala Kalijudan
Surabaya, Indonesia
edward-w.inf24@ukwms.ac.id

2nd Bambang Herlambang
Jurusan Informatika
Universitas Widya Mandala Kalijudan
Surabaya, Indonesia
bambang-h.inf24@ukwms.com

3rd Nathanael Melvin
Jurusan Informatika
Universitas Widya Mandala Kalijudan
Surabaya, Indonesia
nathanael-m.inf24@ukwms.com

***Abstract*—Paper ini menyajikan implementasi sistem Lost and Found digital berbasis Golang untuk lingkungan kampus. Sistem ini mengimplementasikan mekanisme "Verifikasi Tertutup" untuk mencegah klaim palsu dengan menyembunyikan detail sensitif dari publik. Arsitektur mengadopsi Clean Architecture dengan pemisahan Handler-Service-Repository, implementasi Goroutine untuk background worker, Database Transaction untuk integritas data, dan Context Timeout untuk reliability. Sistem mencakup 20+ REST API endpoints dengan fitur filtering, pagination, sorting, file upload, dan automated testing dengan coverage 40**

***Index Terms*—Lost and Found, Golang, Clean Architecture, Concurrency, Database Transaction, RBAC, REST API**

I. Pendahuluan

A. Latar Belakang

Pengelolaan barang hilang di lingkungan kampus memerlukan sistem terpusat yang dapat memverifikasi kepemilikan secara akurat. Sistem manual rentan terhadap fraud dan kehilangan jejak audit. Sistem yang diusulkan mengimplementasikan backend modern dengan Golang yang menekankan concurrency, transaction safety, dan security.

B. Tujuan

Membangun sistem Lost and Found yang memenuhi kriteria:

- 20+ REST API endpoints dengan operasi kompleks
- Database Transaction untuk ACID compliance
- Goroutine untuk background processing
- Context & Timeout untuk reliability
- Clean Architecture dengan Dependency Injection
- Testing dengan Table-Driven Tests dan Mocking

II. Arsitektur Sistem

A. Tech Stack

- **Backend:** Golang 1.21, Gin Framework
- **Database:** MySQL 8.0, GORM ORM
- **Frontend:** HTML5, Vanilla JavaScript, Tailwind CSS
- **Security:** JWT Authentication, Bcrypt Hashing, AES-256 Encryption
- **Testing:** Testify, Gomock

B. Clean Architecture

Sistem mengimplementasikan tiga layer terpisah:

1. Handler Layer (Presentation)

```
// controllers/item_controller.go
type ItemController struct {
    itemService *services.ItemService
}

func (c *ItemController) CreateItem(ctx *gin.
    Context) {
    // Parse request, call service, return
        response
}
```

2. Service Layer (Business Logic)

```
// services/item_service.go
type ItemService struct {
    itemRepo *repositories.ItemRepository
}

func (s *ItemService) CreateItem(...) (*models
    .Item, error) {
    // Validation, transaction logic
}
```

3. Repository Layer (Data Access)

```
// repositories/item_repo.go
func (r *ItemRepository) Create(item *models.
    Item) error {
    return r.db.Create(item).Error
}
```

C. Role-Based Access Control

Tiga role dengan permission granular:

III. API Endpoints

Sistem memiliki 25+ endpoints yang dikelompokkan berdasarkan fungsionalitas:

TABLE I
MATRIKS RBAC

Operasi	User	Manager	Admin
Buat Item			
Edit Item Sendiri			
Edit Item Lain			
Verifikasi Claim			
Kelola User			
Ekspor Data			

A. Authentication Endpoints

- `POST /api/register` - Registrasi user baru dengan validasi email & NRP
- `POST /api/login` - Login dengan JWT token generation
- `POST /api/refresh-token` - Refresh expired token
- `GET /api/me` - Get current user info

B. Item Management Endpoints

- `GET /api/items` - List items dengan filtering (status, category, search), pagination, dan sorting
- `GET /api/items/:id` - Get item detail (public view untuk user, full detail untuk manager)
- `POST /api/items` - Create item dengan secret_details tersembunyi
- `PUT /api/items/:id` - Update item dengan revision log
- `PATCH /api/items/:id/status` - Update status item
- `DELETE /api/items/:id` - Soft delete item
- `GET /api/items/:id/revisions` - Get revision history
- `GET /api/user/items` - Get items by reporter

C. Claim Management Endpoints

- `GET /api/claims` - List claims dengan filter status
- `GET /api/claims/:id` - Get claim detail dengan verification score
- `POST /api/claims` - Create claim dengan idempotency key
- `POST /api/claims/:id/verify` - Verify claim (manager only)
- `GET /api/claims/:id/verification` - Get similarity score
- `POST /api/claims/:id/close` - Close case dengan berita acara
- `POST /api/claims/:id/reopen` - Reopen closed case
- `POST /api/claims/:id/cancel-approval` - Cancel approval
- `DELETE /api/claims/:id` - Delete pending claim

D. Lost Item Endpoints

- `GET /api/lost-items` - List lost item reports
- `POST /api/lost-items` - Create lost item report
- `POST /api/lost-items/:id/find-similar` - Trigger auto-matching
- `GET /api/lost-items/:id/matches` - Get match results

E. Archive & Admin Endpoints

- `GET /api/archives` - List archived items
- `GET /api/admin/dashboard` - Dashboard stats dari database views
- `GET /api/admin/audit-logs` - Audit trail dengan pagination
- `POST /api/reports/export` - Export PDF/Excel reports

F. File Upload Endpoints

- `POST /api/upload/item-image` - Upload gambar dengan resize otomatis
- `POST /api/upload/claim-proof` - Upload bukti klaim
- `POST /api/upload/multiple` - Batch upload (max 5 files)
- `DELETE /api/upload/delete` - Delete uploaded file

IV. FITUR TEKNIS UTAMA

A. Database Transaction

Setiap operasi kompleks menggunakan transaction dengan proper rollback:

```
func (s *ClaimService) VerifyClaim(...) error
    {
    return s.db.Transaction(func(tx *gorm.DB)
        error {
        // 1. Lock claim
        var claim models.Claim
        if err := tx.Clauses(clause.Locking{
            Strength: "UPDATE"}).
            First(&claim, claimID).Error; err
                != nil {
            return err
        }

        // 2. Create verification record
        verification := &models.
            ClaimVerification{...}
        tx.Create(verification)

        // 3. Update claim status
        claim.Status = models.
            ClaimStatusApproved
        tx.Save(&claim)

        // 4. Update item status
        tx.Model(&models.Item{}).
            Where("id = ?", claim.ItemID).
            Update("status", models.
                ItemStatusVerified)

        // 5. Create notification
        notification := &models.Notification
            {...}
        tx.Create(notification)
```

```
        return nil // Commit if all success
    })
}
```

B. Goroutine & Background Workers

Sistem menggunakan 4 background worker:

1. Expire Worker

```
func (w *ExpireWorker) Start() {
    go func() {
        ticker := time.NewTicker(1 * time.Hour
            )
        for {
            select {
            case <-ticker.C:
                w.expireItems() // Archive
                    expired items
            case <-w.stopChan:
                return
            }
        }
    }()
}
```

2. Matching Worker - Auto-match lost items dengan found items setiap 30 menit

3. Notification Worker - Kirim notifikasi tertunda setiap 5 menit

4. Audit Worker - Cleanup old logs setiap 24 jam

C. Context & Timeout

Setiap database query menggunakan context dengan timeout:

```
func (s *ItemService) GetAllItems(...) ([]
    models.Item, error) {
    ctx, cancel := context.WithTimeout(
        context.Background(),
        3*time.Second
    )
    defer cancel()

    txRepo := repositories.NewItemRepository(
        s.db.WithContext(ctx)
    )
    items, total, err := txRepo.FindAll(...)

    if ctx.Err() == context.DeadlineExceeded {
        return nil, errors.New("request
            timeout")
    }

    return items, total, err
}
```

D. Idempotency

Mencegah double-submit pada endpoint kritis dengan idempotency key di header:

```
func IdempotencyMiddleware() gin.HandlerFunc {
    return func(ctx *gin.Context) {
        key := ctx.GetHeader("Idempotency-Key
            ")
        if key == "" {
            ctx.Next()
            return
        }

        // Check if already processed
        if result, exists := cache.Get(key);
            exists {
            ctx.JSON(200, gin.H{
                "idempotent": true,
                "data": result,
            })
            ctx.Abort()
            return
        }

        ctx.Next()
        // Cache result after processing
    }
}
```

E. File Upload dengan Validasi

- Validasi MIME type (image/jpeg, image/png)
- Max size 10MB
- Auto-resize gambar ke 1920x1080
- Generate unique filename dengan timestamp
- Path traversal prevention

V. Mekanisme Verifikasi Tertutup

A. Konsep Utama

Sistem menyembunyikan `secret_details` dari public view. Ketika user mengklaim item:

1) User hanya melihat nama, foto, lokasi, kategori
2) User mengisi deskripsi sendiri tentang ciri khas item
3) Manager membandingkan deskripsi user dengan `secret_details` tersimpan
4) Sistem menghitung similarity score menggunakan Levenshtein Distance
5) Jika score 70%, rekomendasi APPROVE
6) Jika 50-69%, rekomendasi REVIEW
7) Jika ¡ 50%, rekomendasi REJECT

B. Algoritma Similarity

```
func CalculateStringSimilarity(s1, s2 string)
    float64 {
    distance := levenshteinDistance(s1, s2)
    maxLen := max(len(s1), len(s2))
    similarity := 1.0 - (float64(distance) /
        maxLen)
    return max(0, similarity)
}
```

VI. Testing Strategy

A. Table-Driven Tests

```
func TestAuthService_Login(t *testing.T) {
    tests := []struct {
        name          string
        input         LoginRequest
```

```
        mockSetup     func(*MockUserRepository
            )
        expectedError bool
    }{
        {
            name: "Success: Valid Login",
            input: LoginRequest{
                Email: "test@example.com",
                Password: "password123",
            },
            mockSetup: func(repo *
                MockUserRepository) {
                repo.On("FindByEmail", "
                    test@example.com").
                    Return(&models.User{...},
                        nil)
            },
            expectedError: false,
        },
        {
            name: "Failed: Wrong Password",
            input: LoginRequest{
                Email: "test@example.com",
                Password: "wrongpass",
            },
            mockSetup: func(repo *
                MockUserRepository) {
                repo.On("FindByEmail", "
                    test@example.com").
                    Return(&models.User{...},
                        nil)
            },
            expectedError: true,
        },
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            mockRepo := new(MockUserRepository
                )
            tt.mockSetup(mockRepo)

            service := &AuthService{userRepo:
                mockRepo}
            _, err := service.Login(tt.input,
                "", "")

            if tt.expectedError {
                assert.Error(t, err)
            } else {
                assert.NoError(t, err)
            }
        })
    }
}
```

B. Mocking

Menggunakan testify/mock untuk unit test tanpa database:

```
type MockUserRepository struct {
    mock.Mock
}

func (m *MockUserRepository) FindByEmail(email
     string) (*models.User, error) {
    args := m.Called(email)
    if args.Get(0) == nil {
        return nil, args.Error(1)
    }
    return args.Get(0).(*models.User), args.
        Error(1)
}
```

VII. Configuration Management

Semua config diambil dari environment variables menggunakan `godotenv`:

```
# .env file
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASSWORD=secret
DB_NAME=lost_and_found

JWT_SECRET_KEY=your-secret-key
ENCRYPTION_KEY=32-byte-encryption-key

PORT=8080
ENVIRONMENT=production
```

Tidak ada hardcoded credentials di source code.

VIII. Error Handling & Logging

A. Structured Logging

Menggunakan Zap untuk JSON structured logs:

```
logger.Info("Server starting",
    zap.String("port", "8080"),
    zap.String("environment", "production"),
)

logger.Error("Database connection failed",
    zap.Error(err),
    zap.String("host", dbHost),
)
```

B. HTTP Status Code

Response menggunakan status code yang tepat:

- 200 OK - Success
- 201 Created - Resource created
- 400 Bad Request - Invalid input
- 401 Unauthorized - Authentication required
- 403 Forbidden - Insufficient permission
- 404 Not Found - Resource not found
- 422 Unprocessable Entity - Validation error
- 429 Too Many Requests - Rate limit exceeded
- 500 Internal Server Error - Server error

IX. Graceful Shutdown

Server mengimplementasikan graceful shutdown untuk menyelesaikan request yang sedang berjalan:

```
// 1. Stop accepting new requests
srv.Shutdown(ctx)

// 2. Stop background workers
expireWorker.Stop()
matchingWorker.Stop()
```

```
notificationWorker.Stop()
auditWorker.Stop()

// 3. Close database connections
db.Close()
```

X. Frontend Implementation

Frontend menggunakan vanilla JavaScript tanpa framework:

- **Item List:** Pagination, search, filter by category/status
- **Claim Form:** Dynamic form validation, file upload preview
- **Manager Dashboard:** Real-time stats dari database views
- **Verification UI:** Side-by-side comparison dengan similarity score
- **Notification Bell:** Real-time notification count

XI. Database Design

A. Core Tables

- `users` - User accounts dengan encrypted NRP & phone
- `roles` - RBAC roles
- `items` - Found items dengan secret_details
- `lost_items` - Lost item reports
- `claims` - Claim submissions
- `claim_verifications` - Similarity scores
- `match_results` - Auto-match results
- `archives` - Expired/closed items
- `audit_logs` - Audit trail
- `revision_logs` - Edit history
- `notifications` - User notifications

B. Database Views

Sistem menggunakan 5 views untuk optimasi query:

- `vw_dashboard_stats` - Aggregate statistics
- `vw_items_detail` - Items dengan join category & reporter
- `vw_claims_detail` - Claims dengan verification data
- `vw_match_results_detail` - Match results lengkap
- `vw_recent_activities` - Latest audit logs (limit 100)

XII. API Documentation

Dokumentasi lengkap API tersedia di Postman:

Postman Documentation URL:

https://documenter.getpostman.com/view/48307750/2sB3dTs84R

Dokumentasi mencakup:

- **Request Examples** - Sample request body untuk setiap endpoint
- **Response Examples** - Expected response format (success & error)
- **Authentication** - JWT Bearer token setup
- **Environment Variables** - Base URL configuration
- **Error Codes** - Comprehensive error handling documentation

A. Postman Collection Highlights

Collection terbagi dalam folder:

1) **Auth** - Register, Login, Refresh Token, Get Me
2) **Items** - CRUD operations dengan filtering & pagination
3) **Claims** - Full claim workflow dari create hingga close case
4) **Lost Items** - Report & matching features
5) **Upload** - File upload dengan multi-part form data
6) **Admin** - Dashboard stats, audit logs, user management
7) **Manager** - Verification & case management

XIII. Compliance Checklist

Sistem memenuhi seluruh kriteria penilaian UAS Backend:

A. A. Kompleksitas & Fungsionalitas (25 Poin)

TABLE II
Checklist Kompleksitas Fitur

Kriteria	Status
Minimal 15 endpoint (target: 20+)?	(25+)
Pencarian dengan filter & pagination?	
Create/Update menggunakan DB Transaction?	
File handling (upload/download dengan validasi)?	

Detail Implementation:

- **25 Endpoints** tersebar di 7 resource groups
- **Advanced Filtering:** `GET /api/items?status=unclaimed&category=electronic`
- **Transactional Operations:** VerifyClaim, CloseClaim, CreateItem dengan audit log
- **File Upload:** Validasi MIME type (image/jpeg, image/png), max 10MB, auto-resize

B. B. Golang Engineering (15 Poin)

TABLE III
Checklist Golang Engineering

Kriteria	Status
Background process dengan Goroutine?	
Aplikasi lulus tes `go run -race`?	
Context & Timeout pada database query?	
Idempotency untuk endpoint kritis?	

Detail Implementation:

- **4 Background Workers:** ExpireWorker (archiving), MatchingWorker (auto-match), NotificationWorker (alerts), AuditWorker (cleanup)
- **Race-free:** Menggunakan `sync.WaitGroup`, `sync.Mutex`, dan proper channel closing
- **Context Timeout:** Setiap database operation dibatasi 3-5 detik
- **Idempotency:** Header `Idempotency-Key` pada POST /api/claims

TABLE IV
CHECKLIST ARCHITECTURE

Kriteria	Status
Clean Architecture (Handler-Service-Repository)?	
Config dari .env (tidak hardcoded)?	
Error handling & structured logging?	
API mengembalikan HTTP status code yang benar?	
Graceful shutdown?	

C. C. Architecture & Quality (30 Poin)

Detail Implementation:

- **Dependency Injection:** Service layer menerima repository interface
- **Environment Variables:** DB credentials, JWT secret, encryption key dari .env
- **Zap Logger:** JSON structured logs dengan level (Info, Warn, Error)
- **Status Codes:** 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 422 Validation Error, 429 Rate Limit, 500 Internal Error
- **Graceful Shutdown:** Stop HTTP server → Stop workers → Close DB connections

D. D. Testing & Reliability (30 Poin)

TABLE V
CHECKLIST TESTING

Kriteria	Status
Unit Test menggunakan Table-Driven pattern?	
Unit Test menggunakan Mock (offline)?	
Test coverage minimal 40%?	
Test untuk negative cases?	

Detail Implementation:

- **Table-Driven Tests:** `TestAuthService_Login` dengan 4+ test cases
- **Mocking:** `MockUserRepository`, `MockRoleRepository` menggunakan testify/mock
- **Coverage:** 45% (target 40%)
- **Negative Tests:** Wrong password, expired token, unauthorized access, validation errors

XIV. KESIMPULAN

Sistem Lost and Found yang diimplementasikan memenuhi 100% kriteria penilaian UAS Backend:

- **25+ Endpoints** dengan operasi kompleks (filtering, pagination, sorting)
- **Database Transaction** untuk ACID compliance pada 5+ operasi kritis
- **4 Background Workers** dengan Goroutine, WaitGroup, dan proper synchronization
- **Context Timeout** pada setiap database query untuk reliability
- **Clean Architecture** dengan Dependency Injection untuk testability
- **Environment Variables** untuk semua konfigurasi (zero hardcoded secrets)
- **Structured Logging** dengan Zap (JSON format)
- **Proper HTTP Status Codes** pada semua response
- **Graceful Shutdown** dengan proper cleanup sequence
- **Table-Driven Tests** dengan 45% coverage
- **Mocking** untuk offline unit testing
- **Race-free** (lulus `go run -race`)
- **Idempotency** untuk mencegah double-submit
- **File Upload** dengan validasi MIME type dan size

Sistem ini menyediakan solusi yang aman, scalable, dan maintainable untuk pengelolaan barang hilang dan ditemukan di lingkungan kampus dengan mekanisme verifikasi tertutup yang mencegah klaim palsu.

REFERENCES

[1] The Go Programming Language Specification, "https://go.dev/ref/spec," 2024.
[2] Gin Web Framework, "https://gin-gonic.com/docs/," 2024.
[3] GORM Documentation, "https://gorm.io/docs/," 2024.
[4] Clean Architecture, Robert C. Martin, Prentice Hall, 2017.