Upload File dengan Go Fiber & MongoDB

Tujuan Pembelajaran

- **Memahami implementasi Clean Architecture di Go**
- **Mampu membuat REST API upload file dengan Go Fiber**
- **Mampu integrasi dengan MongoDB**

Struktur Folder

```
project-upload/
├── main.go
├── config/
│   └── database.go
├── models/
│   └── file.go
├── repositories/
│   └── file_repository.go
├── services/
│   └── file_service.go
├── routes/
│   └── file_route.go
└── uploads/
```

Config - Database Connection

```go
package config

import (
    "context"
    "fmt"
    "log"
    "time"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

var DB *mongo.Database

func ConnectDB() {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()
```

```
    // Ganti dengan connection string MongoDB Anda
    clientOptions := options.Client().ApplyURI("mongodb://localhost:27017")

    client, err := mongo.Connect(ctx, clientOptions)
    if err != nil {
        log.Fatal(err)
    }

    // Test connection
    err = client.Ping(ctx, nil)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Connected to MongoDB!")

    // Set database
    DB = client.Database("upload_db")
}
```

Model - Data Structure

```
package models

import (
    "time"

    "go.mongodb.org/mongo-driver/bson/primitive"
)

type File struct {
    ID           primitive.ObjectID `json:"id" bson:"_id,omitempty"`
    FileName     string             `json:"file_name" bson:"file_name"`
    OriginalName string             `json:"original_name"
bson:"original_name"`
    FilePath     string             `json:"file_path" bson:"file_path"`
    FileSize     int64              `json:"file_size" bson:"file_size"`
    FileType     string             `json:"file_type" bson:"file_type"`
    UploadedAt   time.Time          `json:"uploaded_at" bson:"uploaded_at"`
}

type FileResponse struct {
    ID           string    `json:"id"`
    FileName     string    `json:"file_name"`
    OriginalName string    `json:"original_name"`
    FilePath     string    `json:"file_path"`
    FileSize     int64     `json:"file_size"`
```

```
    FileType     string    `json:"file_type"`
    UploadedAt   time.Time `json:"uploaded_at"`
}
```

Repository - Database Query Layer

```
package repositories

import (
    "context"
    "project-upload/models"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
    "go.mongodb.org/mongo-driver/mongo"
)

type FileRepository interface {
    Create(file *models.File) error
    FindAll() ([]models.File, error)
    FindByID(id string) (*models.File, error)
    Delete(id string) error
}

type fileRepository struct {
    collection *mongo.Collection
}

func NewFileRepository(db *mongo.Database) FileRepository {
    return &fileRepository{
        collection: db.Collection("files"),
    }
}

func (r *fileRepository) Create(file *models.File) error {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    file.UploadedAt = time.Now()
    result, err := r.collection.InsertOne(ctx, file)
    if err != nil {
        return err
    }

    file.ID = result.InsertedID.(primitive.ObjectID)
    return nil
}
```

```go
func (r *fileRepository) FindAll() ([]models.File, error) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    var files []models.File
    cursor, err := r.collection.Find(ctx, bson.M{})
    if err != nil {
        return nil, err
    }
    defer cursor.Close(ctx)

    if err = cursor.All(ctx, &files); err != nil {
        return nil, err
    }

    return files, nil
}

func (r *fileRepository) FindByID(id string) (*models.File, error) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    objectID, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return nil, err
    }

    var file models.File
    err = r.collection.FindOne(ctx, bson.M{"_id": objectID}).Decode(&file)
    if err != nil {
        return nil, err
    }

    return &file, nil
}

func (r *fileRepository) Delete(id string) error {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    objectID, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return err
    }

    _, err = r.collection.DeleteOne(ctx, bson.M{"_id": objectID})
    return err
```

```
}
```

Service - Business Logic Layer

```
package services

import (
    "errors"
    "fmt"
    "mime/multipart"
    "os"
    "path/filepath"
    "project-upload/models"
    "project-upload/repositories"

    "github.com/gofiber/fiber/v2"
    "github.com/google/uuid"
)

type FileService interface {
    UploadFile(c *fiber.Ctx) error
    GetAllFiles(c *fiber.Ctx) error
    GetFileByID(c *fiber.Ctx) error
    DeleteFile(c *fiber.Ctx) error
}

type fileService struct {
    repo       repositories.FileRepository
    uploadPath string
}

func NewFileService(repo repositories.FileRepository, uploadPath string)
FileService {
    return &fileService{
        repo:       repo,
        uploadPath: uploadPath,
    }
}

func (s *fileService) UploadFile(c *fiber.Ctx) error {
    // Get file from form
    fileHeader, err := c.FormFile("file")
    if err != nil {
        return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
            "success": false,
            "message": "No file uploaded",
            "error":   err.Error(),
        })
```

```
    }

    // Validasi ukuran file (max 10MB)
    if fileHeader.Size > 10*1024*1024 {
        return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
            "success": false,
            "message": "File size exceeds 10MB",
        })
    }

    // Validasi tipe file
    allowedTypes := map[string]bool{
        "image/jpeg":      true,
        "image/png":       true,
        "image/jpg":       true,
        "application/pdf": true,
    }

    contentType := fileHeader.Header.Get("Content-Type")
    if !allowedTypes[contentType] {
        return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
            "success": false,
            "message": "File type not allowed",
        })
    }

    // Generate unique filename
    ext := filepath.Ext(fileHeader.Filename)
    newFileName := uuid.New().String() + ext
    filePath := filepath.Join(s.uploadPath, newFileName)

    // Buat folder jika belum ada
    if err := os.MkdirAll(s.uploadPath, os.ModePerm); err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to create upload directory",
            "error":   err.Error(),
        })
    }

    // Simpan file
    file, err := fileHeader.Open()
    if err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to open file",
            "error":   err.Error(),
        })
```

```
    }
    defer file.Close()

    out, err := os.Create(filePath)
    if err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to save file",
            "error":   err.Error(),
        })
    }
    defer out.Close()

    if _, err := out.ReadFrom(file); err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to write file",
            "error":   err.Error(),
        })
    }

    // Simpan metadata ke database
    fileModel := &models.File{
        FileName:     newFileName,
        OriginalName: fileHeader.Filename,
        FilePath:     filePath,
        FileSize:     fileHeader.Size,
        FileType:     contentType,
    }

    if err := s.repo.Create(fileModel); err != nil {
        // Hapus file jika gagal simpan ke database
        os.Remove(filePath)
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to save file metadata",
            "error":   err.Error(),
        })
    }

    return c.Status(fiber.StatusCreated).JSON(fiber.Map{
        "success": true,
        "message": "File uploaded successfully",
        "data":    s.toFileResponse(fileModel),
    })
}

func (s *fileService) GetAllFiles(c *fiber.Ctx) error {
```

```
    files, err := s.repo.FindAll()
    if err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to get files",
            "error":   err.Error(),
        })
    }

    var responses []models.FileResponse
    for _, file := range files {
        responses = append(responses, *s.toFileResponse(&file))
    }

    return c.JSON(fiber.Map{
        "success": true,
        "message": "Files retrieved successfully",
        "data":    responses,
    })
}

func (s *fileService) GetFileByID(c *fiber.Ctx) error {
    id := c.Params("id")

    file, err := s.repo.FindByID(id)
    if err != nil {
        return c.Status(fiber.StatusNotFound).JSON(fiber.Map{
            "success": false,
            "message": "File not found",
            "error":   err.Error(),
        })
    }

    return c.JSON(fiber.Map{
        "success": true,
        "message": "File retrieved successfully",
        "data":    s.toFileResponse(file),
    })
}

func (s *fileService) DeleteFile(c *fiber.Ctx) error {
    id := c.Params("id")

    file, err := s.repo.FindByID(id)
    if err != nil {
        return c.Status(fiber.StatusNotFound).JSON(fiber.Map{
            "success": false,
            "message": "File not found",
```

```
            "error":   err.Error(),
        })
    }

    // Hapus file dari storage
    if err := os.Remove(file.FilePath); err != nil {
        fmt.Println("Warning: Failed to delete file from storage:", err)
    }

    // Hapus dari database
    if err := s.repo.Delete(id); err != nil {
        return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
            "success": false,
            "message": "Failed to delete file",
            "error":   err.Error(),
        })
    }

    return c.JSON(fiber.Map{
        "success": true,
        "message": "File deleted successfully",
    })
}

func (s *fileService) toFileResponse(file *models.File) *models.FileResponse {
    return &models.FileResponse{
        ID:           file.ID.Hex(),
        FileName:     file.FileName,
        OriginalName: file.OriginalName,
        FilePath:     file.FilePath,
        FileSize:     file.FileSize,
        FileType:     file.FileType,
        UploadedAt:   file.UploadedAt,
    }
}
```

Routes - HTTP Routing

```
package routes

import (
    "project-upload/services"

    "github.com/gofiber/fiber/v2"
)

func SetupFileRoutes(app *fiber.App, service services.FileService) {
    api := app.Group("/api")
```

```
    files := api.Group("/files")

    files.Post("/upload", service.UploadFile)
    files.Get("/", service.GetAllFiles)
    files.Get("/:id", service.GetFileByID)
    files.Delete("/:id", service.DeleteFile)
}
```

Main App

```
package main

import (
    "log"
    "project-upload/config"
    "project-upload/repositories"
    "project-upload/routes"
    "project-upload/services"

    "github.com/gofiber/fiber/v2"
    "github.com/gofiber/fiber/v2/middleware/cors"
    "github.com/gofiber/fiber/v2/middleware/logger"
)

func main() {
    // Connect to database
    config.ConnectDB()

    // Initialize app
    app := fiber.New(fiber.Config{
        BodyLimit: 10 * 1024 * 1024, // 10MB
    })

    // Middleware
    app.Use(cors.New())
    app.Use(logger.New())

    // Serve static files (uploaded files)
    app.Static("/uploads", "./uploads")

    // Dependency Injection
    fileRepo := repositories.NewFileRepository(config.DB)
    fileService := services.NewFileService(fileRepo, "./uploads")

    // Setup routes
    routes.SetupFileRoutes(app, fileService)

    // Start server
```

```
    log.Fatal(app.Listen(":3000"))
}
```

Testing dengan Postman

- Upload File
  - Method: POST
  - URL: http://localhost:3000/api/files/upload
  - Body: form-data
  - Key: file (type: File)
  - Value: Select your file
- Get All Files
  - Method: GET
  - URL: http://localhost:3000/api/files/
- Get File by ID
  - Method: GET
  - URL: http://localhost:3000/api/files/{id}
- Delete File
  - Method: DELETE
  - URL: http://localhost:3000/api/files/{id}

Tugas

Buat Dua Endpoint untuk upload file foto dan sertifikat pada project yang sudah dikembangkan sebagai berikut :

1. upload foto dengan validasi format jpeg / jpg / png dengan batasan maksimal 1MB
2. upload sertifikat / ijazah dengan format pdf dengan batasan maksimal 2MB
3. implementasikan midlleware untuk user dan admin, dengan skema admin bisa menambahkan uploadan di semua user, sedangkan user hanya bisa upload untuk dirinya sendiri