**Soal Teori**

1. Dalam pembuatan REST API, kita sering menggunakan method GET , POST , PUT, dan DELETE. Jelaskan perbedaan keempat method tersebut!

**Jawab:**

* **GET**: Digunakan untuk membaca atau mengambil data dari server. Data yang diambil biasanya berupa resource seperti JSON atau file, tanpa mengubah data di server. GET bersifat idempotent, artinya jika dipanggil berkali-kali dengan parameter yang sama, hasilnya tetap sama.
* **POST**: Digunakan untuk mengirim data baru ke server. Biasanya digunakan untuk operasi pembuatan (create), seperti menambahkan entri baru ke database. Tidak idempotent karena memanggil POST berulang kali dapat menghasilkan data yang berbeda atau duplikat.
* **PUT**: Digunakan untuk memperbarui data yang sudah ada di server. PUT bersifat idempotent, artinya jika operasi dilakukan berulang kali dengan data yang sama, hasilnya tetap konsisten.
* **DELETE**: Digunakan untuk menghapus resource dari server. Sama seperti PUT, DELETE juga bersifat idempotent, sehingga memanggil DELETE berulang kali untuk resource yang sama tidak akan mengakibatkan kesalahan.

1. Apa keunggulan menggunakan Web Component (seperti Polymer, LitElement, dan sebagainya) dibandingkan menggunakan framework besar seperti ReactJS, Vue, dan sebagainya)?

**Jawab:**

* **Standar Web Native**: Web Component menggunakan standar teknologi web (HTML, CSS, dan JavaScript), sehingga kompatibel dengan semua browser modern tanpa memerlukan library tambahan.
* **Reusable Components**: Komponen yang dibuat dapat digunakan kembali di proyek lain atau digabungkan dengan framework lain seperti React atau Angular tanpa konflik.
* **Ringan**: Tidak memerlukan bundling framework besar, sehingga lebih hemat dalam ukuran dan lebih cepat di-load.
* **Encapsulation**: Web Component memiliki Shadow DOM yang memberikan isolasi antara komponen dan halaman utama, mencegah konflik styling dan menjaga konsistensi.
* **Maintenance**: Tidak bergantung pada framework tertentu, sehingga lebih mudah dirawat dalam jangka panjang, meskipun framework besar yang ada saat ini menjadi usang.

1. Jelaskan pentingnya sebuah program terintegrasi dengan database!

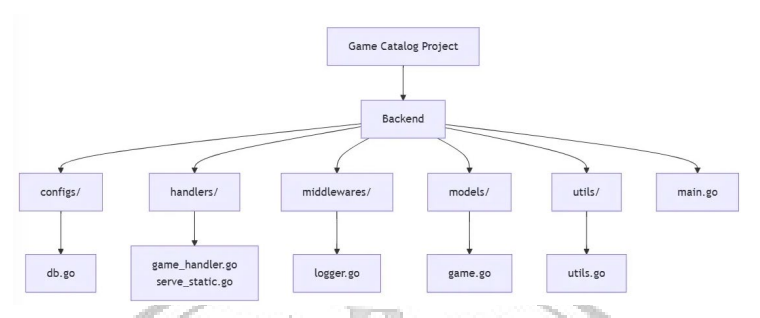
**Jawab:**

* **Penyimpanan Data**: Database memungkinkan program menyimpan data secara terstruktur dan terorganisasi dengan baik.
* **Pengambilan Data**: Memungkinkan program mengambil data dengan cepat dan efisien untuk ditampilkan atau digunakan dalam operasi lainnya.
* **Persistensi**: Data tetap tersedia meskipun program dihentikan atau server restart, memungkinkan data dapat diakses kembali kapan saja.
* **Manajemen Data Skala Besar**: Database dirancang untuk menangani data dalam jumlah besar dengan performa yang optimal.
* **Keamanan**: Database menyediakan fitur untuk mengontrol akses dan memastikan keamanan data melalui mekanisme seperti enkripsi dan otentikasi.
* **Integrasi dengan Aplikasi Lain**: Program yang terintegrasi dengan database dapat dengan mudah berbagi data dengan aplikasi atau sistem lain untuk keperluan analitik, pelaporan, atau layanan lainnya.

**Soal Activity**

Sebuah perusahaan game store digital, GameVault ID, sedang mengembangkan platform katalog game berbasis web untuk mengelola inventaris game mereka. Sebagai developer backend, Anda ditugaskan untuk membangun REST API yang akan mengintegrasikan database MySQL dengan frontend yang telah disediakan oleh tim UI/UX. Platform ini harus mampu menampilkan daftar game, menambahkan game baru ke katalog, memperbarui informasi game yang sudah ada, dan menghapus game dari katalog. Sistem juga harus dilengkapi dengan logging untuk memantau permintaan yang masuk ke server. Tim frontend telah menyiapkan antarmuka pengguna menggunakan Lit components, dan Anda perlu memastikan backend dapat melayani file statis serta menangani endpoint API dengan tepat. Ikutilah langkah dibawah ini !

1. Buatlah struktur direktori projek seperti dibawah ini!



**Jawab:**

A screenshot of a computer

Description automatically generated

1. Pada file **main.go**, import library yang diperlukan dan buatlah function main() yang akan menginisialisasi dan menjalankan server pada port xxxx (empat digit terakhir npm). Fungsi ini harus dapat melakukan koneksi ke database, menangani kemungkinan kegagalan koneksi ke database, menyiapkan file server untuk konten statis dari folder 'catalog', serta mendaftarkan route yang diperlukan untuk API games. Pastikan juga untuk mengimplementasikan middleware logging untuk mencatat semua request yang masuk ke server. Perhatikan bahwa koneksi database harus ditutup dengan benar ketika server berhenti beroperasi!

**Jawab:**

package main

import (

    "fmt"

    "log"

    "net/http"

    "Game\_Catalog\_Project/configs"

    "Game\_Catalog\_Project/handlers"

    "Game\_Catalog\_Project/middlewares"

    \_ "github.com/go-sql-driver/mysql"

)

func main() {

    PORT := 2650

    configs.ConnectDB()

    if configs.DB == nil {

        log.Fatal("Database connection failed")

    }

    defer func() {

        if err := configs.DB.Close(); err != nil {

            log.Printf("Error closing database connection: %v", err)

        }

    }()

mux := http.NewServeMux()

    fileServer := http.FileServer(http.Dir("catalog"))

    mux.Handle("/", func(w http.ResponseWriter, r \*http.Request) {

        handlers.ServeStaticFile(w, r, "catalog", fileServer)

    })

    mux.Handlefunc("/api/games/", handlers.HandleGames)

    mux.Handlefunc("/api/games", handlers.HandleGames)

    loggedMux := middlewares.LogRequestHandler(mux)

    fmt.Printf("Server berjalan di http://localhost:%d\n", PORT)

    log.Fatal(http.ListenAndServe(fmt.Sprintf(":%d", PORT), loggedMux))

}

1. Pada file **configs/db.go**, import library yang diperlukan dan buat fungsi ConnectDB() yang akan menginisialisasi koneksi database

**Jawab:**

package configs

import (

    "database/sql"

    "log"

    "time"

    \_ "github.com/go-sql-driver/mysql"

)

var DB \*sql.DB

func ConnectDB() {

    var err error

    DB, err = sql.Open("mysql", "root@tcp(127.0.0.1:3306)/steam\_catalog")

    if err != nil {

        log.Fatal("Failed to open database connection: ", err)

    }

    DB.SetMaxOpenConns(10)

    DB.SetMaxIdleConns(5)

    DB.SetConnMaxLifetime(time.Minute \* 5)

    err = DB.Ping()

    if err != nil {

        log.Fatal("Failed to ping database connection: ", err)

    }

    \_, err = DB.Query("SELECT 1 FROM games LIMIT 1")

    if err != nil {

        log.Fatal("Failed to query games table: ", err)

    }

    log.Println("Successfully connected to database")

}

1. Pada file **models/game.go**, buatlah struct Game dengan lima field: ID, Nama, Developer, Genre, dan Harga

**Jawab:**

package models

type Game struct {

**ID**        int     `json:"id\_game"`

**Nama**      string  `json:"nama\_game"`

**Developer** string  `json:"developer"`

**Genre**     string  `json:"genre"`

**Harga**     float64 `json:"harga"`

}

1. Pada file **utils/utils.go**, buatlah fungsi RespondJSON untuk mengirim response dalam format JSON. Fungsi ini menerima http.ResponseWriter dan data interface{} sebagai parameter

**Jawab:**

package utils

import (

    "encoding/json"

    "net/http"

)

func RespondJSON(w http.ResponseWriter, data interface{}) {

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

    json.NewEncoder(w).Encode(data)

}

1. Pada file **handlers/serve\_static.go**, buatlah fungsi ServeStaticFile untuk melayani file statis dari folder catalog. Fungsi ini harus dapat menangani path URL, melakukan pengecekan file, dan mengarahkan kembali ke halaman utama jika file tidak ditemukan. Pastikan request ke path '/api/' tidak dilayani oleh file server

**Jawab:**

package handlers

import (

    "net/http"

    "os"

    "path/filepath"

    "strings"

)

func ServeStaticFile(w http.ResponseWriter, r \*http.Request, baseDir string, fileServer http.Handler) {

    path := strings.TrimPrefix(r.URL.Path, "/")

    fullpath := filepath.Join(baseDir, path)

    if strings.HasPrefix(r.URL.Path, "/api/") {

        http.NotFound(w, r)

        return

    }

    \_, err := os.Stat(fullpath)

    if err != nil {

        http.Redirect(w, r, "/", http.StatusTemporaryRedirect)

        return

    }

    fileServer.ServeHTTP(w, r)

}

1. Pada file **handlers/game\_handler.go**, buatlah API endpoint untuk operasi CRUD pada data game. Buatlah fungsi utama HandleGames yang akan mengarahkan request ke fungsi yang sesuai berdasarkan method HTTP. Implementasikan juga fungsi handleGetGames untuk mengambil data game (baik semua game maupun game spesifik), handlePostGame untuk menambah game baru, handleUpdateGame untuk memperbarui data game, dan handleDeleteGame untuk menghapus game. Pastikan setiap fungsi menangani error dengan tepat dan memberikan response yang sesuai

**Jawab:**

package handlers

import (

    "Game\_Catalog\_Project/configs"

    "Game\_Catalog\_Project/models"

    "Game\_Catalog\_Project/utils"

    "database/sql"

    "encoding/json"

    "net/http"

    "strconv"

)

func HandleGames(w http.ResponseWriter, r \*http.Request) {

    path := r.URL.Path

    var id int

    var err error

    if len(path) > len("/api/games/") {

        id, err = strconv.Atoi(path[len("/api/games/"):])

        if err != nil {

            http.Error(w, "ID tidak valid", http.StatusBadRequest)

            return

        }

    } else {

        id = 0

    }

    switch r.Method {

    case http.MethodGet:

        handleGetGames(w, id)

    case http.MethodPost:

        handlePostGame(w, r)

    case http.MethodPut:

        handleUpdateGame(w, r, id)

    case http.MethodDelete:

        handleDeleteGame(w, id)

    }

}

func handleGetGames(w http.ResponseWriter, id int) {

    if id == 0 {

        rows, err := configs.DB.Query("SELECT id\_game, nama\_game, developer, genre, harga FROM games")

        if err != nil {

            http.Error(w, "Gagal mengambil data game", http.StatusInternalServerError)

            return

        }

        defer rows.Close()

        var games []models.Game

        for rows.Next() {

            var game models.Game

            err := rows.Scan(&game.ID, &game.Nama, &game.Developer, &game.Genre, &game.Harga)

            if err != nil {

                http.Error(w, "Gagal memproses data game", http.StatusInternalServerError)

                return

            }

            games = append(games, game)

        }

        utils.RespondJSON(w, games)

    } else {

        var game models.Game

        err := configs.DB.QueryRow(

            "SELECT id\_game, nama\_game, developer, genre, harga FROM games WHERE id\_game = ?",

            id,

        ).Scan(&game.ID, &game.Nama, &game.Developer, &game.Genre, &game.Harga)

        if err != nil {

            if err == sql.ErrNoRows {

                http.Error(w, "Game tidak ditemukan", http.StatusNotFound)

            } else {

                http.Error(w, "Gagal mengambil data game", http.StatusInternalServerError)

            }

            return

        }

        utils.RespondJSON(w, game)

    }

}

func handlePostGame(w http.ResponseWriter, r \*http.Request) {

    var game models.Game

    if err := json.NewDecoder(r.Body).Decode(&game); err != nil {

        utils.RespondJSON(w, map[string]string{"error": "Data tidak valid"})

        return

    }

    \_, err := configs.DB.Exec(

        "INSERT INTO games (nama\_game, developer, genre, harga) VALUES (?, ?, ?, ?)",

        game.Nama, game.Developer, game.Genre, game.Harga,

    )

    if err != nil {

        utils.RespondJSON(w, map[string]string{"error": err.Error()})

        return

    }

    utils.RespondJSON(w, map[string]string{"message": "Game berhasil ditambahkan!"})

}

func handleUpdateGame(w http.ResponseWriter, r \*http.Request, id int) {

    var game models.Game

    if err := json.NewDecoder(r.Body).Decode(&game); err != nil {

        http.Error(w, "Data tidak valid", http.StatusBadRequest)

        return

    }

    result, err := configs.DB.Exec(

        "UPDATE games SET nama\_game=?, developer=?, genre=?, harga=? WHERE id\_game=?",

        game.Nama, game.Developer, game.Genre, game.Harga, id,

    )

    if err != nil {

        http.Error(w, "Gagal memperbarui game", http.StatusInternalServerError)

        return

    }

    rowsAffected, \_ := result.RowsAffected()

    if rowsAffected == 0 {

        http.Error(w, "Game dengan ID tersebut tidak ditemukan", http.StatusNotFound)

        return

    }

    utils.RespondJSON(w, map[string]string{"message": "Game berhasil diperbarui!"})

}

func handleDeleteGame(w http.ResponseWriter, id int) {

    result, err := configs.DB.Exec("DELETE FROM games WHERE id\_game=?", id)

    if err != nil {

        http.Error(w, "Gagal menghapus game", http.StatusInternalServerError)

        return

    }

    rowsAffected, \_ := result.RowsAffected()

    if rowsAffected == 0 {

        http.Error(w, "Game dengan ID tersebut tidak ditemukan", http.StatusNotFound)

        return

    }

    utils.RespondJSON(w, map[string]string{"message": "Game berhasil dihapus!"})

}

1. Pada file **middlewares/logger.go**, buatlah fungsi LogRequestHandler untuk mencatat setiap request yang masuk ke server. Fungsi ini harus menerima dan meneruskan request ke handler berikutnya sambil melakukan logging method dan path URL request

**Jawab:**

package middlewares

import (

    "log"

    "net/http"

)

func LogRequestHandler(next http.Handler) http.Handler {

    return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

        log.Printf("%s %s", r.Method, r.URL.Path)

        next.ServeHTTP(w, r)

    })

}

1. Tampilan MariaDB

**Jawab:**

