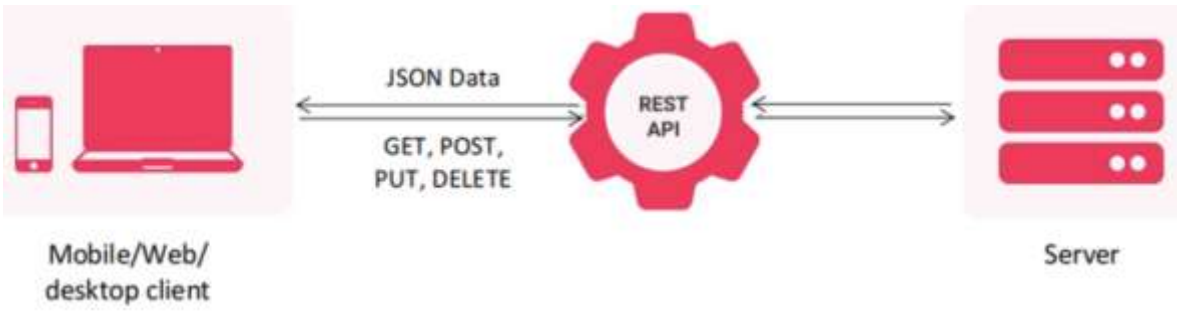
PRAKTIKUM MCS BAB 6

Card bridge

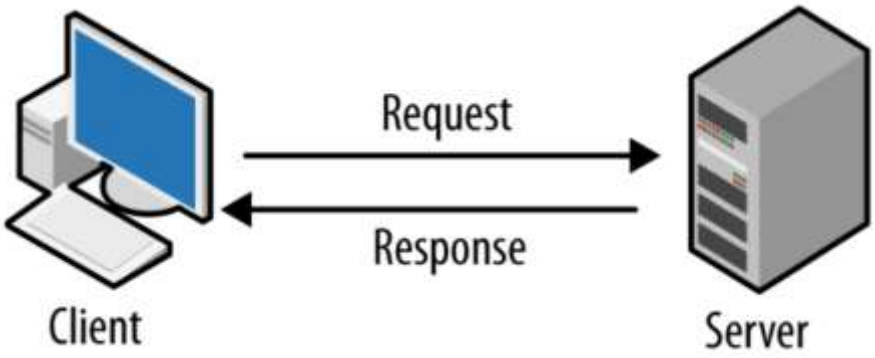
PENDAHULUAN

Pada praktikum MCS bab 6 akan belajar bagaimana caranya membangun RESTFUL API sebuah aplikasi back end sebuah layanan yang menjadi penghubung antara end user dengan Internet of Things (IOT). Server menjadi connector antara aplikasi android mobile dan IOT yang dibangun berbasis Uniform Resource Locator (URL). Aplikasi back end yang dibangun menggunakan bahasa pemrograman Go dengan framework yang bernama Gin. Adapun dari sisi IOT menggunakan micro controller esp32 dan sensor Radio Frequency Identification (RFID). Setiap kartu yang dibaca oleh RFID akan masuk ke server kemudian data kartu yang ada di server akan dibaca oleh aplikasi android. Untuk berkomunikasi dengan back end berbasis URL (RESTFUL API) diperlukan beberapa method, beberapa method yang digunakan adalah GET, POST, PUT, DELETE.



Connect server

User mengirim request dengan beberapa method yang digunakan untuk berkomunikasi dengan server kemudian server mengirim response.

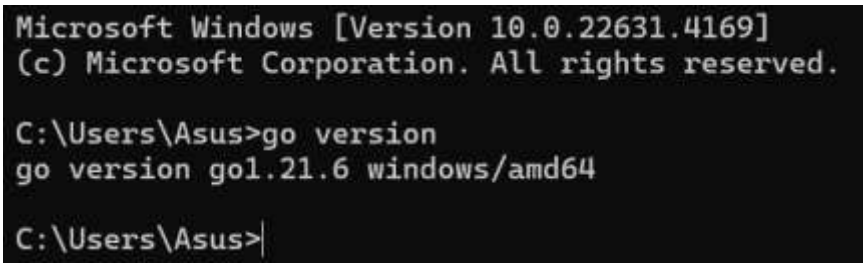


Request and response

Di praktikum ini hanya akan membangun dari sisi server saja dan praktikum akan berlanjut di bab 7 dan 8.

PRAKTIKUM BAB 6

Pastikan perangkat yang digunakan untuk membangun REST API sudah terdapat software Go language, Visual Studio Code dengan extensionnya golang dan code runner, Postman, web browser.



Cek versi golang

Buat folder project misalnya “mcs_bab_6” kemudian buka command prompt masuk ke path di dalam folder tersebut dan ketik command berikut :

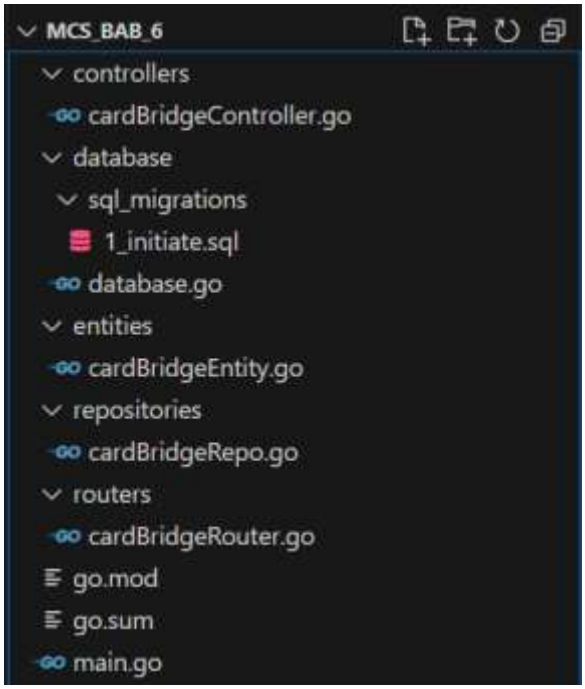
```
go mod init mcs_bab_6
```

Setelah muncul file go.mod masukkan perintah berikut :

```
go get -u "github.com/gin-gonic/gin"
go get -u "github.com/lib/pq"
go get -u "github.com/rubenv/sql-migrate"
go get -u "github.com/joho/godotenv"
```

Perintah `go get -u "github.com/gin-gonic/gin"` untuk package Gin framework, Gin framework ,e,udahkan pengembangan API. Package ini menyediakan berbagai fitur seperti routing, middleware dan handling JSON. Perintah `go get -u "github.com/lib/pq"` adalah instalasi driver untuk PostgreSQL, di mana aplikasi yang dibangun menggunakan Go dapat berkomunikasi dengan PostgreSQL dan mengirim query. Perintah `go get -u "github.com/rubenv/sql-migrate"` adalah mengunduh / instalasi migrasi sql, dengan migrasi sql pengembang dapat mengelola konfigurasi database. Perintah `go get -u "github.com/joho/godotenv"` adalah untuk mengunduh / instalasi godotenv yang digunakan untuk membaca file .env. file .env adalah file yang berisi konfigurasi. Semua perintah yang dimasukkan menggunakan `-u` untuk mendapatkan package terbaru jika ditemukan versi terbaru.

Buat root project dengan folder dan file sebagai berikut :



Root project

Buka file cardBridgeEntity.go dan isi dengan code berikut

```go
package entities

type Card struct {
    ID string `json:"id"`
}
```

Struct yang ada pada file ini digunakan sebagai model yang disesuaikan dengan table di database. Karena hanya menyimpan id dari kartu saja maka Card dengan tipe struct memiliki 1 property saja yaitu id.

Buka file 1_initiate.sql dan isi dengan coide berikut :

```sql
-- +migrate Up
-- +migrate StatementBegin

CREATE TABLE card(
    id varchar(20)
);

-- +migrate StatementEnd
```

Code ini adalah untuk membuat table database. Table bernama card dan di dalam table ini memiliki 1 field bernama id dengan tipe varchar yang bisa diisi 20 karakter. Untuk penulisan migrate up, statement begin dan statement end merupakan instruksi. Migrate up adalah instruksi utuk menerapkan semua query SQL ke yang lebih baru. Statement begin dan statemnet end merupakan perintah yang harus dieksekusi sebagai satu kesatuan yang tidak bisa dipisahkan, menjadikan hal tersebut atomisitas sebagai pernyataan yang kompleks.

Buka file database.go dan isi dengan code berikut :

```go
package database

import (
    "database/sql"
    "embed"
    "fmt"

    migrate "github.com/rubenv/sql-migrate"
)

//go:embed sql_migrations/*.sql
var dbMigrations embed.FS
var DbCoonnection *sql.DB

func DBMigrate(dbParam *sql.DB) {
    migrations := &migrate.EmbedFileSystemMigrationSource{
        FileSystem: dbMigrations,
        Root:       "sql_migrations",
    }

    n, errs := migrate.Exec(dbParam, "postgres", migrations, migrate.Up)

    if errs != nil {
        panic(errs)
    }

    DbCoonnection = dbParam

    fmt.Println("Migrations success applied", n, migrations)
}
```

Code pada file ini bertujuan untuk melakukan migrasi database. Melalui parameter yang ada pada funtion DBMigrate() menerima koneksi database dan menjalankan migrasi berdasarkan file yang ada di dalam sql_migrations. `//go:embed  sql_migrations/*.sql` digunakan

menyematkan semua file .sql yang ada di dalam folder sql_migrations ke dalam variabel dbMigrations.

Buka file cardBridgeRepo.dart dan isi dengan code berikut :

```
package repositories

import (
    "database/sql"
    "mcs_bab_6/entities"
)

func GetCards(db *sql.DB) (result []entities.Card, err error) {
    sql := "SELECT * FROM card"

    rows, err := db.Query(sql)

    if err != nil {
        return
    }

    defer rows.Close()

    for rows.Next() {
        var data entities.Card
        err = rows.Scan(&data.ID)
        if err != nil {
            return
        }
        result = append(result, data)
    }
    return
}

func InsertCard(db *sql.DB, card entities.Card) (err error) {
    sql := "INSERT INTO card(id) values($1)"

    // errs := db.QueryRow(sql, 1)
    // return errs.Err()

    _, err = db.Exec(sql, card.ID)
    return err
}

func DeleteCard(db *sql.DB, card entities.Card) (err error) {
    sql := "DELETE FROM card WHERE id = $1"

    // errs := db.QueryRow(sql, 1)
```

```
    // return errs.Err()

    _, err = db.Exec(sql, card.ID)
    return err
}
```

Code di bagian repositories bertujuan untuk melakukan interaksi dengan database. GetCards() digunakan untuk membaca data dari table bernama card. Data dibaca dengan menggunakan query select dan dilakukan perulangan untuk mengisi ke dalam variabel result. InsertCard() digunakan untuk memasukkan data ke dalam table card dan DeleteCard() merupakan sebaliknya yaitu digunakan untuk menghapus data dari table card.

Buka file cardBridgeController.go dan isi dengan code berikut :

```
package controllers

import (
    "mcs_bab_6/database"
    "mcs_bab_6/entities"
    "mcs_bab_6/repositories"
    "net/http"

    "github.com/gin-gonic/gin"
)

func GetCards(c *gin.Context) {
    var result gin.H

    card, err := repositories.GetCards(database.DbCoonnection)

    if err != nil {
        result = gin.H{
            "result": err.Error(),
        }
    } else {
        result = gin.H{
            "result": card,
        }
    }

    c.JSON(http.StatusOK, result)
}

func InsertCard(c *gin.Context) {
    var card entities.Card
    idCard := c.Param("id")
```

```
    card.ID = idCard

    err := repositories.InsertCard(database.DbCoonnection, card)

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"errpr": err.Error()})
        return
    }

    c.JSON(http.StatusOK, card)
}

func DeleteCard(c *gin.Context) {
    var card entities.Card
    idCard := c.Param("id")

    card.ID = idCard

    err := repositories.DeleteCard(database.DbCoonnection, card)

    if err != nil {
        // panic(err)
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
    }

    c.JSON(http.StatusOK, gin.H{"message": "data berhasil dihapus", "id":
idCard})
}
```

Controller pada dasarnya digunakan untuk mengontrol apa yang akan dilakukan oleh sistem. Pada code ini controller digunakan untuk mengatur response sekaligus memanggil repositori agar terdapat perubahan pada database. Fungsi GetCards digunakan untuk mengambil data list kartu yang ada di database, jika error maka response akan menampilkan error jika tidak error maka data akan ditampilkan. Fungsi InsertCard() untuk memasukkan data kartu. Data kartu dimasukkan menggunakan parameter di akhir url bernama id. Variabel idCard memiliki nilai yang diambil dari parameter id, lalu card.ID diinisialisasi dengan nilai dari idCard. Setelah itu memanggil InsertCard() dari package repositories agar data masuk ke database. Jika error maka response yang akan diberikan adalah error dengan StatusInternalServerError. Fungsi DeleteCard() untuk menghapus salah satu kartu yang ada di database dengan memanggil DeleteCard() yang ada di package repositories. Jika terjadi error maka response yang diberikan adalah StatusInternalServerError.

Buka file cardBridgeRouter.go dan isi dengan code berikut :

```go
package routers

import (
    "mcs_bab_6/controllers"

    "github.com/gin-gonic/gin"
)

func StartServer() *gin.Engine {
    router := gin.Default()

    router.GET("/cards", controllers.GetCards)
    router.POST("/card/input/:id", controllers.InsertCard)
    router.DELETE("/card/delete/:id", controllers.DeleteCard)

    return router
}
```

Router digunakan untuk mengatur arah yang akan dituju di setiap end point yang berbeda. Setiap end point memiliki url inti, kemudian setelah url inti terdapat beberapa jalur yang berbeda biasanya menggunakan tanda slash (/). Code di atas /cards digunakan untuk mendapatkan data list kartu yang tersedia dan memanggil fungsi GetCards() yang ada di package controllers. Sedangkan /card/input/:id digunakan untuk memasukkan kartu baru yang mana :id adalah parameter untuk mengisi data kartu dengan memanggil fungsi InsertCard() yang ada di package controllers. Lalu card/delete/:id digunakan untuk menghapus data list kartu dengan memasukkan data kartu ke parameternya dan memanggil DeleteCard() yang ada di package controllers.

Buka pgAdmin 4 dan buat database dengan nama praktikum_mcs_bab6

Database bab 6

Buka file main.go dan isi dengan code berikut :

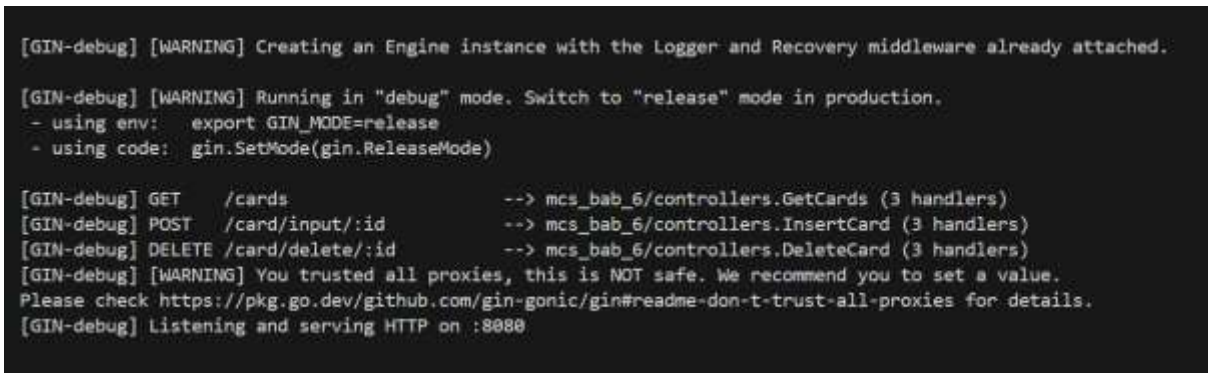
```go
package main

import (
    "database/sql"
    "fmt"
```

```
    "log"
    "mcs_bab_6/database"
    "mcs_bab_6/routers"

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = ""
    dbName   = "praktikum_mcs_bab6"
)

var (
    DB  *sql.DB
    err error
)

func main() {
    var PORT = ":8080"

    psqlInfo := fmt.Sprintf(
        `host=%s port=%d user=%s password=%s dbname=%s sslmode=disable`,
        host, port, user, password, dbName,
    )

    DB, err = sql.Open("postgres", psqlInfo)

    if err != nil {
        log.Fatalf("Error Open DB: %v\n", err)
    }

    database.DBMigrate(DB)

    defer DB.Close()

    routers.StartServer().Run(PORT)
    fmt.Printf("Success Connected")
}
```

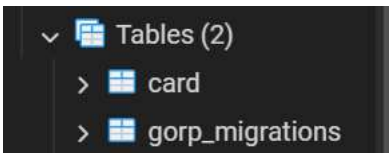
Di dalam main.go buat variabel konstanta bernama host dengan nama “localhost”, port 5432, user “postgres”, password disesuaikan dengan password yang ada di aplikasi PostfreSQL dan dbName berisi nama database yang sudah dibuat di pgAdmin 4 yaitu praktikum_mcs_bab6. Variabel konstanta ini nantinya digunakan untuk berkomunikasi dengan PostgreSQL. Dan buat

juga variabel global lainnya yaitu DB dengan tipe data *sql.DB dan err error. Di dalam fungsi main() menginisialisasi variabel PORT dengan isi :8080 dan sql dibuka dengan mengisi parameter Open() yaitu “postgres” dan psqlInfo. Kemudian panggil DBMigrate yang ada di package database untuk migrasi konfigurasi sql ke aplikasi PostgreSQL. Lalu koneksi database ditutup setelah fungsi main() selesai dieksekusi dengan menggunakan defer agar tidak terjadi kebocoran koneksi. Terakhir untuk memulainya StartServer() dari package routers dan port menggunakan variabel PORT.

Jalankan aplikasi back end yang sudah dibuat dan lihat table card di pgAdmin 4

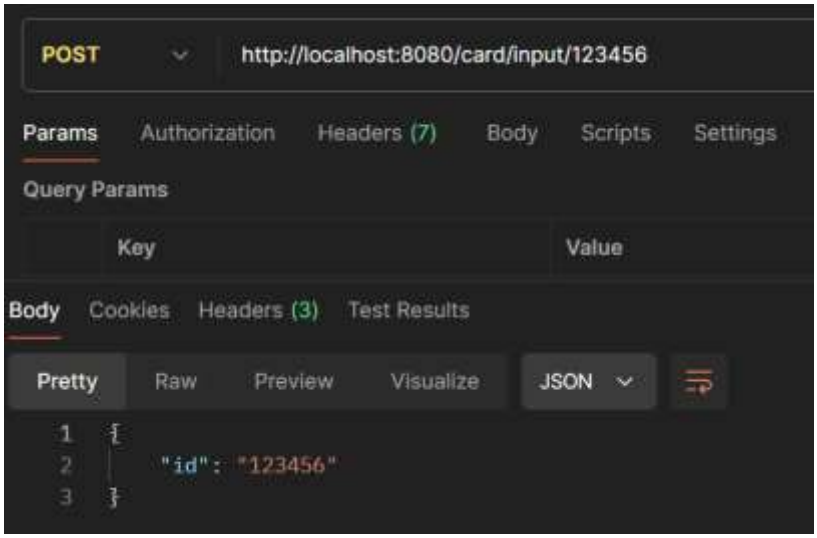


Run back end

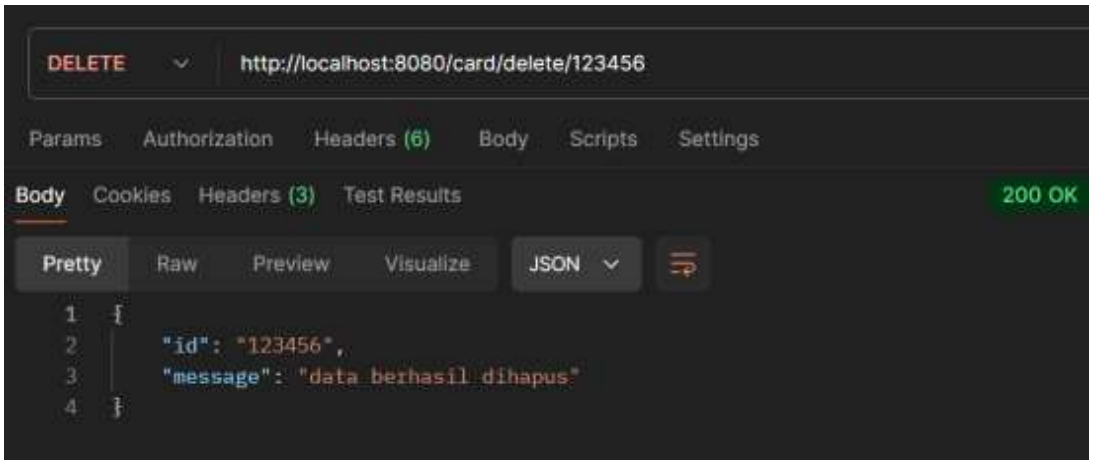


Result of migrate

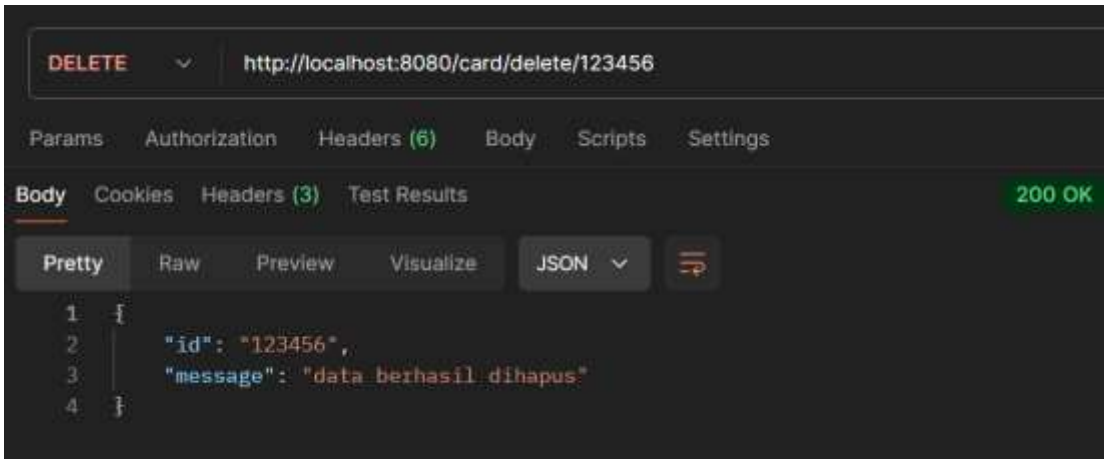
Buka aplikasi postman dan jalankan endpoint dengan rute yang sudah dibuat



Metode post



Metode get



Metode delete

Dari semua url yang dijalankan bisa dilihat status codenya ketika suatu metode (get, post, put, delete) dijalankan.

Status code

LAPORAN PENDAHULUAN (LP)

1. Berikan penjelasan apa itu back end!
2. Berikan penjelasan apa itu panic dan defer pada bahasa pemrograman Golang!
3. Berikan pengertian yang kamu ketahui tentang aplikasi PostgreSQL!
4. Berikan pengertian yang kamu ketahui tentang aplikasi Postman!

LAPORAN AKHIR (LA)

1. Berikan kesimpulan pada praktikum bab 6