**Golang RESTful API Tutorial**

Kita akan membuat API sederhana yang mencakup method GET, POST, PUT dan DELETE menggunakan bahasa pemrograman Golang dan framework fiber. Golang ini direkomendasikan untuk digunakan sebagai pembuatan backend (server) atau API karena pemrosesan compiling atau pemrosesan data nya itu memiliki performa yang tinggi dan atau cepat.

1. Struktur Folder Project

belajar-golang

* config
  + config.go
* database
  + connect.go
  + database.go
* internal
  + handler
    - note
      * noteHandler.go
  + model
    - model.go
  + routes
    - noteRoutes.go
* router
  + router.go
* Main.go
* Go.mod
* Go.sum

1. Dasar – Dasar Package

Go code didistribusikan dalam bentuk package. Package Go digunakan untuk distribusi kode dan logika berdasarkan penggunaannya. Ini dapat diamati pada struktur direktori diatas.

1. Pembuatan API

Kita akan mulai dengan 1 file, titik awal dari kode kita yaitu main.go. Buat file ini di direktori root project kita.

Pertama, kita inisialisasikan terlebih dahulu go module nya dengan perintah :

|  |
| --- |
| **go mod init <nama-folder\_project>** |

Atau bisa juga dengan menginisialisasikan dengan github profile seperti berikut :

|  |
| --- |
| **go mod init github.com/valenrio66/belajar-golang** |

Setelah itu, install library fiber nya dengan perintah berikut :

|  |
| --- |
| **go get github.com/gofiber/fiber/v2** |

Lalu, kita coba code dibawah ini, kemudian run aplikasi nya dengan perintah :

|  |
| --- |
| **go run main.go** atau **go run .** |

*package* main

*import* (

    "github.com/gofiber/fiber/v2"

)

**func** main() {

*// Start a new fiber app*

    app **:=** fiber.New()

*// Send a string back for GET calls to the endpoint "/"*

    app.Get("/", **func**(c **\***fiber.Ctx) *error* {

        err **:=** c.SendString("Hello world! The API is UP!")

**return** err

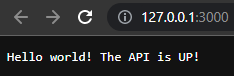
    })

*// Listen on PORT 3000*

    app.Listen(":3000")

}

Dan ini adalah hasilnya :



1. Konfigurasi Database

Sebelum melanjutkan ke konfigurasi database, pertama kita harus menginstall GORM terlebih dahulu. GORM adalah ORM yang dikembangkan untuk bahasa GO yang memungkinkan developer untuk melakukan operasi database dengan mudah dan efisien. ORM adalah teknik pemrograman yang memungkinkan developer untuk memetakan objek dalam kode ke baris dalam tabel database, sehingga memudahkan pengembangan aplikasi dengan mengurangi kompleksitas koneksi ke database.

Berikut ini adalah perintah untuk menginstall beberapa library yang dibutuhkan :

|  |
| --- |
| **go get gorm.io/gorm**  **go get gorm.io/driver/mysql**  **go get -u github.com/go-sql-driver/mysql**  **go get github.com/joho/godotenv** |

Selanjutnya, nyalakan Apache dan MySQL pada xampp dan buka localhost nya untuk membuat database nya terlebih dahulu.

Setelah membuat database, hubungkan database pada aplikasi kita. Buat file .env dan isi port, host, user, password dan nama database yang sesuai. Contoh :

**DB\_USER** = root

**DB\_PASSWORD** =

**DB\_HOST** = localhost

**DB\_PORT** = 3306

**DB\_NAME** = dbgolang

Kemudian, buat folder config dan file config.go di dalam folder tersebut yang berfungsi untuk load file .env yang sudah dibuat. Lalu, masukkan code seperti berikut :

*package* config

*import* (

    "fmt"

    "os"

    "github.com/joho/godotenv"

)

*// Config func to get env value*

**func** Config(key *string*) *string* {

*// load .env file*

    err **:=** godotenv.Load(".env")

**if** err **!=** nil {

        fmt.Print("Error loading .env file")

    }

*// Return the value of the variable*

**return** os.Getenv(key)

}

Setelah itu, buat folder database dan buat file connect.go di dalamnya untuk mengkonfigurasikan atau menghubungkan aplikasi kita dengan database yang sudah dibuat. Lalu, masukkan code berikut :

*package* database

*import* (

    "fmt"

    "log"

    "strconv"

    "belajar-golang/config"

    "gorm.io/driver/mysql"

    "gorm.io/gorm"

    "gorm.io/gorm/schema"

)

*// Mendeklarasikan variabel untuk database*

var DB **\***gorm.DB

*// Fungsi ConnectDB() untuk menghubungkan ke database*

**func** ConnectDB() {

    var err *error*

    p **:=** config.Config("DB\_PORT")

    port, err **:=** strconv.ParseUint(p, 10, 32)

**if** err **!=** nil {

        log.Fatal("Error parsing DB\_PORT")

    }

*// URL koneksi untuk menghubungkan ke database MySQL*

    dsn **:=** fmt.Sprintf("*%s*:*%s*@tcp(*%s*:*%d*)/*%s*?charset=utf8mb4&parseTime=True&loc=Local", config.Config("DB\_USER"), config.Config("DB\_PASSWORD"), config.Config("DB\_HOST"), port, config.Config("DB\_NAME"))

*// Menghubungkan ke DB dan menginisialisasikan variabel DB*

    DB, err **=** gorm.Open(mysql.Open(dsn), **&**gorm.Config{

*// Mematikan pluralisasi nama tabel secara global*

*// jika disetel ke true, `User` akan dipetakan ke tabel `users`*

*// alih-alih nama default, yaitu `user`*

        NamingStrategy: schema.NamingStrategy{

            SingularTable: true,

        },

    })

**if** err **!=** nil {

        log.Fatalf("Failed to connect to database: *%v*", err)

    }

    fmt.Println("Connection Opened to Database")

}

Setelah itu, coba simpan dan running aplikasi untuk memeriksa apakah aplikasi sudah terhubung ke database atau belum.

1. Model

Pembuatan model digunakan untuk menerjemahkan logika aplikasi, seperti validasi data, pemrosesan data, transformasi data, dan operasi CRUD (create, read, update, delete). Model ini juga bertanggung jawab untuk menghubungkan aplikasi dengan data yang tersimpan di dalam database atau sumber data lainnya. Misalkan, disini kita buat folder internal/model/model.go dan masukkan code seperti contoh berikut :

*package* model

**type** User struct {

    IdUser   *int*    `gorm:"primaryKey;column:id\_user;autoIncrement" json:"-"`

    Nama     *string* `gorm:"column:nama" json:"nama"`

    Npm      *string* `gorm:"column:npm" json:"npm"`

    Kelas    *string* `gorm:"column:kelas" json:"kelas"`

    AsalKota *string* `gorm:"column:asal\_kota" json:"asal\_kota"`

}

1. Handler

Handler dalam konteks pengembangan perangkat lunak adalah fungsi atau metode yang menangani permintaan (request) yang masuk dari pengguna atau klien. Handler bertanggung jawab untuk menerima permintaan dari klien, mengambil data yang diperlukan dari database atau sumber data lainnya, memproses permintaan tersebut, dan mengembalikan respons yang sesuai kepada klien.

Di dalam folder internal, buat folder handlers/belajar/belajar.go dan masukkan function code seperti berikut :

* GET All

*package* belajarHandler

*import* (

    "belajar-golang/database"

    "belajar-golang/internal/model"

    "github.com/gofiber/fiber/v2"

)

**func** GetUsers(c **\***fiber.Ctx) *error* {

    var users []model.User

*// Find all users in database*

    result **:=** database.DB.Find(**&**users)

*// Check for errors during query execution*

**if** result.Error **!=** nil {

**return** c.Status(fiber.StatusBadRequest).JSON(fiber.Map{

            "message": result.Error.Error(),

        })

    }

*// Return list of users*

**return** c.Status(fiber.StatusOK).JSON(fiber.Map{

        "message": "Data User Berhasil Ditampilkan!",

        "data":    users,

    })

}

* POST

**func** CreateUser(c **\***fiber.Ctx) *error* {

*// Parse request body*

    var user model.User

**if** err **:=** c.BodyParser(**&**user); err **!=** nil {

**return** err

    }

*// Insert new user into database*

    result **:=** database.DB.Create(**&**user)

*// Check for errors during insertion*

**if** result.Error **!=** nil {

**return** c.Status(fiber.StatusBadRequest).JSON(fiber.Map{

            "message": result.Error.Error(),

        })

    }

*// Return success message*

**return** c.Status(fiber.StatusCreated).JSON(fiber.Map{

        "message": "User Berhasil Ditambahkan!",

        "data":    user,

    })

}

* GET by id

**func** GetUser(c **\***fiber.Ctx) *error* {

*// Get id\_user parameter from request url*

    id **:=** c.Params("id\_user")

*// Find user by id\_user in database*

    var user model.User

    result **:=** database.DB.First(**&**user, id)

*// Check if user exists*

**if** result.RowsAffected **==** 0 {

**return** c.Status(fiber.StatusNotFound).JSON(fiber.Map{

            "message": "User Tidak Ditermukan!",

        })

    }

*// Check for errors during query*

**if** result.Error **!=** nil {

**return** c.Status(fiber.StatusBadRequest).JSON(fiber.Map{

            "message": result.Error.Error(),

        })

    }

*// Return user*

**return** c.Status(fiber.StatusOK).JSON(fiber.Map{

        "message": "Success",

        "data":    user,

    })

}

* PUT

**func** UpdateUser(c **\***fiber.Ctx) *error* {

*// Get id\_user parameter from request url*

    id **:=** c.Params("id\_user")

*// Find user by id\_user in database*

    var user model.User

    result **:=** database.DB.First(**&**user, id)

*// Check if user exists*

**if** result.RowsAffected **==** 0 {

**return** c.Status(fiber.StatusNotFound).JSON(fiber.Map{

            "message": "User Tidak Ditemukan",

        })

    }

*// Parse request body*

    var newUser model.User

**if** err **:=** c.BodyParser(**&**newUser); err **!=** nil {

**return** err

    }

*// Update user in database*

    result **=** database.DB.Model(**&**user).Updates(newUser)

*// Check for errors during update*

**if** result.Error **!=** nil {

**return** c.Status(fiber.StatusBadRequest).JSON(fiber.Map{

            "message": result.Error.Error(),

        })

    }

*// Return success message*

**return** c.Status(fiber.StatusOK).JSON(fiber.Map{

        "message": "User Berhasil Diperbarui!",

        "data":    user,

    })

}

* DELETE

**func** DeleteUser(c **\***fiber.Ctx) *error* {

*// Get id\_user parameter from request url*

    id **:=** c.Params("id\_user")

*// Find user by id\_user in database*

    var user model.User

    result **:=** database.DB.First(**&**user, id)

*// Check if user exists*

**if** result.RowsAffected **==** 0 {

**return** c.Status(fiber.StatusNotFound).JSON(fiber.Map{

            "message": "User Tidak Ditemukan",

        })

    }

*// Delete user from database*

    result **=** database.DB.Delete(**&**user)

*// Check for errors during deletion*

**if** result.Error **!=** nil {

**return** c.Status(fiber.StatusBadRequest).JSON(fiber.Map{

            "message": result.Error.Error(),

        })

    }

*// Return success message*

**return** c.Status(fiber.StatusOK).JSON(fiber.Map{

        "message": "User Berhasil Dihapus!",

        "data":    user,

    })

}

1. Routes

Routes digunakan untuk menentukan bagaimana aplikasi akan menangani permintaan (request) dari klien. Routes mendefinisikan jalur atau endpoint yang akan diakses oleh klien, dan menentukan bagaimana aplikasi akan merespons permintaan tersebut.

Dalam folder internal, buat folder routes/belajar/belajar.go dan masukkan code seperti contoh berikut :

*package* belajarRoutes

*import* (

*belajarHandler* "belajar-golang/internal/handlers/belajar"

    "github.com/gofiber/fiber/v2"

)

**func** SetupBelajarRoutes(router fiber.Router) {

    user **:=** router.Group("/belajar")

*// Create a user*

    user.Post("/", belajarHandler.CreateUser)

*// Read all users*

    user.Get("/", belajarHandler.GetUsers)

*// // Read one user*

    user.Get("/:id\_user", belajarHandler.GetUser)

*// // Update one user*

    user.Put("/:id\_user", belajarHandler.UpdateUser)

*// // Delete one user*

    user.Delete("/:id\_user", belajarHandler.DeleteUser)

}

1. Router

Router adalah suatu komponen pada framework web yang digunakan untuk menentukan bagaimana suatu permintaan HTTP harus diproses dan ditangani oleh aplikasi web. Router ini bertugas untuk memetakan URI (Uniform Resource Identifier) yang dikirim oleh klien ke fungsi atau handler yang tepat dalam aplikasi web.

Pada contoh kode dibawah ini, fungsi **SetupRoutes** merupakan router yang digunakan untuk menentukan rute-rute atau URI yang dapat diakses oleh klien. Di dalamnya terdapat grup **api** yang memiliki rute-rute yang ditentukan oleh **belajarRoutes.SetupBelajarRoutes(api)**.

Dalam kasus ini, **SetupBelajarRoutes** adalah fungsi yang berisi rute-rute yang ditentukan untuk grup **api**. Setiap kali klien mengakses URI yang telah ditentukan, router akan memanggil fungsi atau handler yang sesuai untuk menangani permintaan tersebut.

Pada root directory, buat folder router dan file router.go kemudian masukkan code seperti contoh berikut :

*package* router

*import* (

*belajarRoutes* "belajar-golang/internal/routes/belajar"

    "github.com/gofiber/fiber/v2"

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

)

**func** SetupRoutes(app **\***fiber.App) {

    api **:=** app.Group("/api", logger.New())

*// Setup the Node Routes*

    belajarRoutes.SetupBelajarRoutes(api)

}

1. main.go

Terakhir, konfigurasikan file main.go dengan menambahkan router agar semua endpoint dapat diakses seperti berikut :

*package* main

*import* (

    "belajar-golang/database"

    "belajar-golang/router"

    "github.com/gofiber/fiber/v2"

)

**func** main() {

*// Start a new fiber app*

    app **:=** fiber.New()

*// Connect to the Database*

    database.ConnectDB()

*// Setup the router*

    router.SetupRoutes(app)

*// Listen on PORT 3000*

    app.Listen(":3000")

}

1. Test

Jika semuanya sudah, kita bisa mencobanya dengan run aplikasi nya. Kemudian, kita test menggunakan tools seperti Postman. Atau, bisa juga menggunakan extension VSCode yang bernama Thunder Client. Aplikasi akan berjalan pada http://127.0.0.1:3000/.