Study Jam 1: API Sederhana dengan Go Fiber

1. Persiapan

Untuk mengikuti pembelajaran ini kamu perlu menyiapkan:

- Bahasa **Go Lang (**https://go.dev/**).**
- **Git** untuk clone starter projek (https://git-scm.com/).
- **VS Code** untuk text editor, atau editor kesukaanmu (https://code.visualstudio.com/). Pastikan menginstall Plugin Go Lang.
- **Postman** untuk test API yang kita buat nanti (https://www.postman.com/).

2. Clone Starter Projek

Untuk memulai, kita akan clone repository starter project dari Github. Ikuti langkah-langkah berikut:

1. Buka terminal atau command prompt di komputer kamu
2. Arahkan ke folder dimana kamu ingin menyimpan project
3. Jalankan perintah git clone berikut:

```
git clone https://github.com/anjarmath/belajar_api_gofiber.git
```

4. Masuk ke direktori project yang sudah di-clone:

```
cd belajar_api_gofiber
```

5. Install semua dependencies yang dibutuhkan:

```
go mod tidy
```

Setelah proses clone selesai, kamu akan memiliki salinan lengkap dari starter project di komputermu dan siap untuk memulai pengembangan.

3. Pengenalan Fiber

Fiber (https://gofiber.io/) adalah framework web yang sangat cepat untuk Go yang terinspirasi dari Express.js. Framework ini dibangun di atas Fasthttp, mesin HTTP Go yang paling cepat. Beberapa keunggulan Fiber meliputi:

- Zero memory allocation dan performa tinggi
- API yang mirip dengan Express.js, memudahkan developer yang familiar dengan Node.js
- Middleware support yang powerful untuk extend fungsionalitas
- Routing yang fleksibel dengan dukungan untuk parameter dan wildcard
- Static file serving, WebSocket support, dan rate limiter bawaan

Contoh kode dasar menggunakan Fiber:

```
package main

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

func main() {
    app := fiber.New()

    // Route dasar
    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("Hello, World!")
    })
```

```
    // Menjalankan server pada port 3000
    app.Listen(":3000")
}
```

Framework ini sangat cocok untuk membangun REST API dan web services karena kemudahannya dalam penggunaan dan performanya yang tinggi. Fiber juga memiliki dokumentasi yang lengkap dan komunitas yang aktif.

4. Route & Handler

Route dan handler adalah komponen penting dalam pembuatan API menggunakan Fiber. Route menentukan endpoint atau URL yang dapat diakses, sedangkan handler adalah fungsi yang menangani request pada route tersebut.

Mari kita lihat contoh dasar pembuatan route dan handler:

```
package main

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

func main() {
    app := fiber.New()

    // Route GET dengan handler sederhana
    app.Get("/hello", handleHello)

    // Route POST dengan handler sederhana
    app.Post("/user", handleCreateUser)

    app.Listen(":3000")
}

// Handler untuk route GET /hello
func handleHello(c *fiber.Ctx) error {
    return c.SendString("Hello World!")
}

// Handler untuk route POST /user
func handleCreateUser(c *fiber.Ctx) error {
    return c.JSON(fiber.Map{
        "message": "User created successfully",
    })
}
```

Penjelasan komponen-komponen penting:

- **Route Methods:** Fiber mendukung berbagai HTTP methods seperti:
- app.Get(): untuk HTTP `GET` request
- app.Post(): untuk HTTP `POST` request

> 💡 Route Method adalah metode yang wajib dikirim oleh client (web/aplikasi frontend) untuk spesifikasi permintaan. Route Method sendiri ada banyak jenis, yang sering dipakai adalah:
>
> - `GET` : untuk mengambil data
> - `POST` : untuk eksekusi/mengirim data
> - `DELETE` : untuk menghapus data
> - `PUT` : untuk edit data
> - `PATCH` : untuk edit data sebagian

Contoh route dengan parameter:

```
// Route dengan parameter
app.Get("/user/:id", func(c *fiber.Ctx) error {
    id := c.Params("id")
    return c.SendString("User ID: " + id)
```

```
})

// Route dengan query parameter
app.Get("/search", func(c *fiber.Ctx) error {
    query := c.Query("q")
    return c.SendString("Search query: " + query)
})
```

Contoh handler yang menerima dan mengirim JSON:

```
// Struktur data untuk user
type User struct {
    Name  string `json:"name"`
    Email string `json:"email"`
}

// Handler untuk menerima JSON
app.Post("/user", func(c *fiber.Ctx) error {
    user := new(User)

    if err := c.BodyParser(user); err != nil {
        return err
    }

    return c.JSON(fiber.Map{
        "message": "User received",
        "user": user,
    })
})
```

Tips penting dalam membuat route dan handler:

- Selalu gunakan penamaan yang jelas dan deskriptif untuk route
- Gunakan grouping route untuk endpoint yang berkaitan

Contoh penggunaan route group:

```
// Membuat group route
api := app.Group("/api")

// Route dalam group
api.Get("/users", handleGetUsers)
api.Post("/users", handleCreateUser)
api.Get("/users/:id", handleGetUser)

// Akan menghasilkan route:
// /api/users
// /api/users/:id
```

Dengan pemahaman dasar tentang route dan handler ini, kamu sudah bisa mulai membuat endpoint-endpoint API sederhana menggunakan Fiber.

5. Melengkapi Projek

Untuk tutorial kali ini kita akan mencoba membuat API sedehana untuk koleksi buku, akan terdapat 2 route yakni:

- `GET` `/book` → mengambil semua koleksi buku
- `POST` `/book` → menambahkan buku

Lagkah-langkahnya sebagai berikut:

1. Panggil fungsi untuk inisialisasi buku-buku, secara default koleksi bukunya akan kosong, tambahkan line berikut di `main.go` :

   ```
   model.InitBook()
   ```

2. Buat endpoint untuk get koleksi buku seperti berikut:

```
app.Get("/book", func(c *fiber.Ctx) error {
        return c.JSON(model.BooksCollection)
})
```

3. Buat endpoint untuk menambahkan koleksi buku seperti berikut:

```
app.Post("/book", func(c *fiber.Ctx) error {
        book := new(model.Book)

        if err := c.BodyParser(book); err != nil {
            return c.Status(fiber.StatusBadRequest).JSON(fiber.Error{
                Message: "Mohon isikan data dengan benar!",
            })
        }

        if book.Judul == "" || book.Deskripsi == "" || book.Harga == 0 {
            return c.Status(fiber.StatusBadRequest).JSON(fiber.Error{
                Message: "Mohon isikan data dengan benar!",
            })
        }

        model.BooksCollection = append(model.BooksCollection, *book)

        return c.JSON(fiber.Map{
            "message": "Buku Berhasil ditambahkan",
            "buku":    book,
        })
    })
```

4. Jalankan lalu test menggunakan postman.

6. Refactor Projek

Pada contoh sebelumnya, kita membuat API dengan menempatkan semua logika (route, handler, dan validasi) dalam satu file. Pendekatan ini tidak ideal untuk project yang sebenarnya karena:

- Kode menjadi sulit dibaca dan dipelihara
- Sulit untuk melakukan testing pada komponen tertentu
- Tidak mengikuti prinsip Single Responsibility

Untuk itu, kita perlu melakukan refactoring dengan memisahkan kode ke dalam beberapa package/folder:

- `routes/` → Berisi definisi routing API
- `controller/` → Berisi logic untuk menangani request
- `repository/` → Berisi operasi terkait data

Contoh struktur project setelah refactoring:

```
project/
  ├── main.go
  ├── routes/
  │   └── book.route.go
  ├── controller/
  │   └── book.controller.go
  └── repository/
      ├── book.repository.go
      └── book.repository.impl.go
```

Dengan struktur seperti ini, kode menjadi lebih terorganisir dan mudah dikembangkan untuk skala yang lebih besar. Kita akan menggunakan teknik **Dependency Injection** untuk menghubungkan repository dan controller.

1. **Buat Interface Repository**

   Untuk repository, kita perlu membuat 2 file yakni `book.repository.go` dan `book.repository.impl.go` . `book.repository.go` hanya akan berisi interface, yakni mendefinisikan fungsi-fungsi apa saja yang akan dipakai, tanpa menjelaskan implementasi dari fungsi-fungsi tersebut. berikut adalah isinya:

```
package repository

import "github.com/anjarmath/01_golang_api_sederhana/model"

// Book repository sebagai interface saja
type BookRepository interface {
    AddBook(book *model.Book)
}
```

2. **Buat Implementasi Repository**

   Karena sebelumnya kita telah membuat `book.repository.go` yang hanya berisi interface, kita perlu membuat `book.repository.impl.go` yang berisi implementasi sesungguhnya dari repository ini. Berikut adalah isinya:

```
package repository

import "github.com/anjarmath/01_golang_api_sederhana/model"

type bookRepositoryImpl struct{}

// AddBook implements BookRepository.
func (b bookRepositoryImpl) AddBook(book *model.Book) {
    model.BooksCollection = append(model.BooksCollection, *book)
}

// Untuk membuat repository untuk buku
func NewBookRepository() BookRepository {
    return bookRepositoryImpl{}
}
```

3. **Buat Controller**

```
package controller

import (
    "github.com/anjarmath/01_golang_api_sederhana/model"
    "github.com/anjarmath/01_golang_api_sederhana/repository"
    "github.com/gofiber/fiber/v2"
)

type bookController struct {
    bookRepository repository.BookRepository
}

// Untuk membuat controller untuk buku, tentu kita butuh repository untuk berkomunikasi denga
n data
// Sehingga kita perlu menambahkan repository di parameter fungsi ini
func NewBookController(repo repository.BookRepository) *bookController {
    return &bookController{
        bookRepository: repo,
    }
}

func (bc bookController) GetBook(c *fiber.Ctx) error {
    return c.JSON(model.BooksCollection)
}

func (bc bookController) AddBook(c *fiber.Ctx) error {
    book := new(model.Book)

    if err := c.BodyParser(book); err != nil {
        return c.Status(fiber.StatusBadRequest).JSON(fiber.Error{
            Message: "Mohon isikan data dengan benar!",
        })
    }
```

```
    valid := validateBook(book)
    if !valid {
        return c.Status(fiber.StatusBadRequest).JSON(fiber.Error{
            Message: "Mohon isikan data dengan benar!",
        })
    }

    bc.bookRepository.AddBook(book)

    return c.JSON(fiber.Map{
        "message": "Buku Berhasil ditambahkan",
        "buku":    book,
    })
}

// Buat Validasi untuk buku yang diinputkan user
func validateBook(book *model.Book) bool {
    return !(book.Judul == "" || book.Deskripsi == "" || book.Harga == 0)
}
```

4. **Setup Routes**

```
package routes

import (
    "github.com/anjarmath/01_golang_api_sederhana/controller"
    "github.com/anjarmath/01_golang_api_sederhana/repository"
    "github.com/gofiber/fiber/v2"
)

func SetupBookRoutes(r *fiber.App) {
    r.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("Belajar bareng API Sederhana!")
    })

    // Buat repository dengan fungsi yang telah kita buat sebelumnya.
    bookRepo := repository.NewBookRepository()

    // Buat controller juga, dan inject repository ke dalam controller yang dibuat.
    bookCtrl := controller.NewBookController(bookRepo)

    r.Get("/book", bookCtrl.GetBook)

    r.Post("/book", bookCtrl.AddBook)
}
```

> 💡 Salah satu kegunaan teknik memisah antara interface dan implementasi pada repository di atas adalah jika misal kita menggunakan beberapa database sekaligus di sistem kita, akan repot jika untuk banyak database tersebut kita perlu memanggil repository yang berbeda di controller nantinya, sehingga dengan teknik ini kita hanya perlu membuat implementasi untuk masing-masing koneksi database dan cukup memanggil fungsi-fungsi yang sama di controller.

5. **Panggil Routes di** `main.go`

```
func main() {
    app := fiber.New()

    model.InitBook()

    routes.SetupBookRoutes(app)

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

6. Jalankan lalu test menggunakan postman.

7. Apa Selanjutnya

- Belajar PostgreSql sebagai DBMS
- Integrasi Go Lang REST API dengan PostgreSql