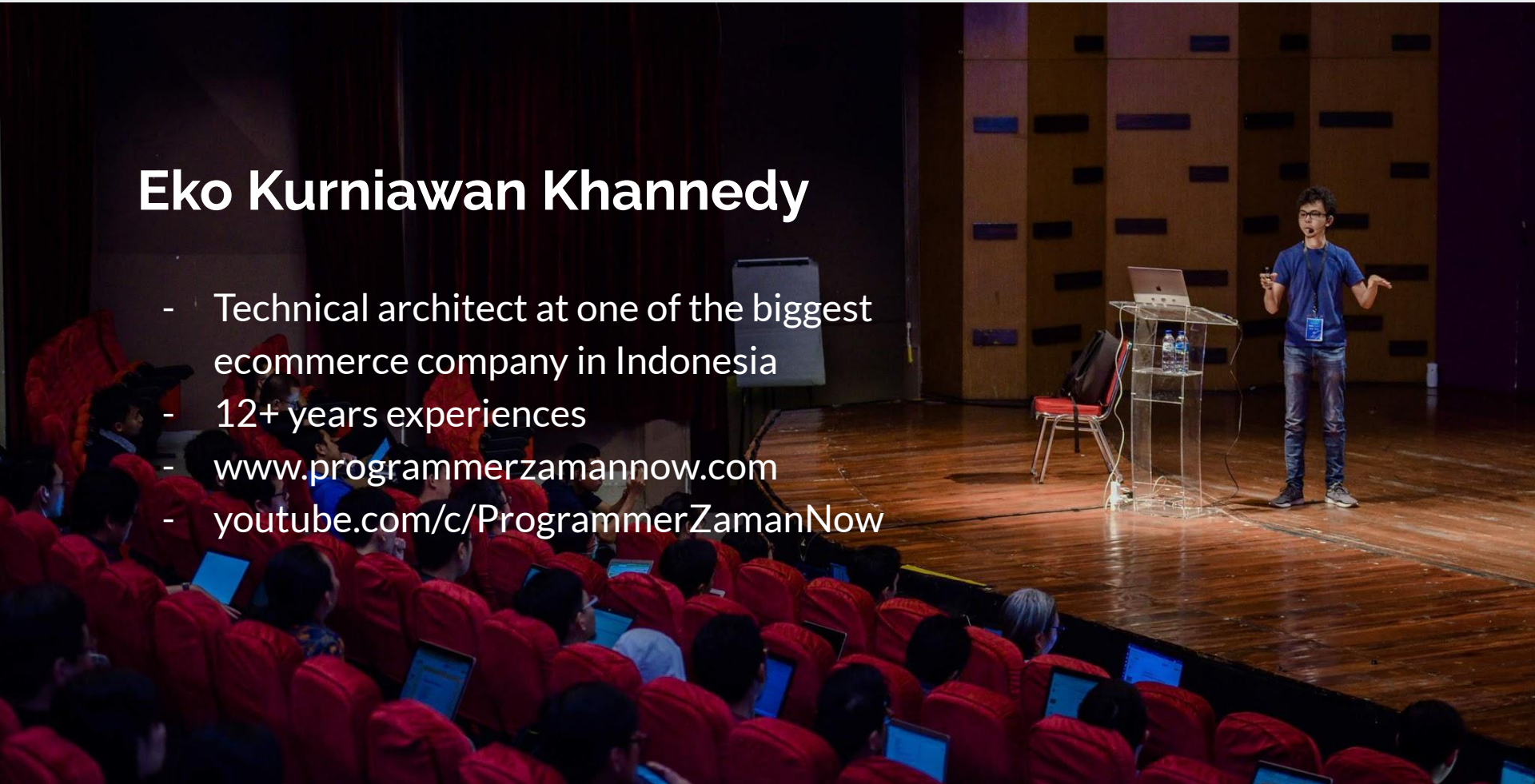
Go-Lang Fiber

Eko Kurniawan Khannedy



Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 12+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Linkedin : https://www.linkedin.com/company/programmer-zaman-now/
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Tiktok : https://tiktok.com/@programmerzamannow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- Golang Dasar
- Golang Modules
- Golang Unit Test
- Golang Web

Pengenalan Golang Fiber

Web Framework

- Saat kita belajar kelas Golang Web, kita sudah belajar bagaimana cara membuat aplikasi web menggunakan Golang
- Dengan tambahan HTTPRouter, sebenarnya sudah cukup untuk membuat Web menggunakan Golang
- Namun, terkadang masih banyak beberapa hal yang harus dilakukan secara manual menggunakan bawaan dari Golang HTTP, sehingga kita membutuhkan Web Framework untuk mempermudah saat membuat aplikasi web menggunakan Golang
- Terutama Golang sangat populer untuk membuat RESTful API

Fiber

- Sebenarnya ada banyak sekali Web Framework yang bisa kita gunakan untuk memudahkan kita membuat aplikasi Web dengan Golang
- Pada kelas ini, kita akan membahas salah satu Web Framework yang populer di Golang, yaitu Fiber
- Fiber adalah Web Framework untuk Golang yang terinspirasi dari ExpressJS, oleh karena itu cara penggunaannya sangat mudah dan sederhana
- Hal ini otomatis bisa mempermudah kita ketika membuat aplikasi Web / RESTful API menggunakan Golang
- https://gofiber.io/

Membuat Project

Membuat Project

- Buat folder belajar-golang-fiber
- go mod init belajar-golang-fiber
- Tambahkan dependency Testify
- go get github.com/stretchr/testify

Dependency Fiber

- Tambahkan dependency Fiber
- go get github.com/gofiber/fiber/v2

Fiber App

Fiber App

- Saat kita menggunakan Fiber, hal pertama yang perlu kita buat adalah fiber.App
- Untuk membuatnya kita bisa menggunakan fiber.New(fiber.Config) yang menghasilkan pointer *fiber.App
- fiber.App adalah representasi dari aplikasi Web Fiber
- Setelah membuat fiber.App, selanjutnya untuk menjalankan aplikasi web nya, kita bisa menggunakan method Listen(address)

Kode : Fiber App

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

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

    err := app.Listen("localhost:3000")
    if err != nil {
        panic(err)
    }
}
```

Configuration

Configuration

- Saat kita membuat fiber.App menggunakan fiber.New() terdapat parameter fiber.Config yang bisa kita gunakan
- Ada banyak sekali konfigurasi yang bisa kita ubah, dan kita akan bahas secara bertahap
- Contoh yang bisa kita gunakan adalah mengubah konfigurasi timeout

Kode : Configuration

```go
func main() {
    app := fiber.New(fiber.Config{
        IdleTimeout:  time.Second * 5,
        ReadTimeout:  time.Second * 5,
        WriteTimeout: time.Second * 5,
    })

    err := app.Listen("localhost:3000")
    if err != nil {
        panic(err)
    }
}
```

Routing

Routing

- Saat kita menggunakan Web Framework, pertama yang kita tanyakan adalah, bagaimana cara membuat endpoint nya?
- Di Fiber, untuk membuat Routing, sudah disediakan semua Method di fiber.App yang sesuai dengan HTTP Method
- Parameternya membutuhkan 2, yaitu path nya dan juga fiber.Handler

Kode : Routing Method

```
// HTTP methods
func (app *App) Get(path string, handlers ...Handler) Router
func (app *App) Head(path string, handlers ...Handler) Router
func (app *App) Post(path string, handlers ...Handler) Router
func (app *App) Put(path string, handlers ...Handler) Router
func (app *App) Delete(path string, handlers ...Handler) Router
func (app *App) Connect(path string, handlers ...Handler) Router
func (app *App) Options(path string, handlers ...Handler) Router
func (app *App) Trace(path string, handlers ...Handler) Router
func (app *App) Patch(path string, handlers ...Handler) Router
```

Kode : Hello World

```go
func TestRoutingHelloWorld(t *testing.T) {
    app := fiber.New()
    app.Get("/", func(ctx *fiber.Ctx) error {
        return ctx.SendString("Hello World!")
    })

    request := httptest.NewRequest("GET", "/", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    assert.Equal(t, 200, response.StatusCode)

    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Hello World!", string(bytes))
}
```

Ctx

Ctx

- Saat kita membuat Handler di Fiber Routing, kita hanya cukup menggunakan parameter fiber.Ctx
- Ctx ini merupakan representasi dari Request dan Response di Fiber
- Oleh karena itu, kita bisa mendapatkan informasi HTTP Request, dan juga bisa membuat HTTP Response menggunakan fiber.Ctx
- Untuk detail HTTP Request dan HTTP Response, akan kita bahas lebih detail di materi-materi selanjutnya

Kode : Ctx

```go
var app = fiber.New()  4 usages

func TestCtx(t *testing.T) {
    app.Get("/hello", func(ctx *fiber.Ctx) error {
        name := ctx.Query("name", "Guest")
        return ctx.SendString("Hello " + name)
    })

    request := httptest.NewRequest("GET", "/hello?name=Eko", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Hello Eko", string(bytes))
}
```

HTTP Request

HTTP Request

- Representasi dari HTTP Request di Fiber adalah Ctx
- Untuk mengambil informasi dari HTTP Request, kita bisa menggunakan banyak sekali Method yang terdapat di Ctx
- Kita bisa baca seluruh method yang tersedia di Ctx di halaman Doc nya
- https://pkg.go.dev/github.com/gofiber/fiber/v2#Ctx

Kode : HTTP Request

```go
func TestHttpRequest(t *testing.T) {
    app.Get("/request", func(ctx *fiber.Ctx) error {
        first := ctx.Get("firstname")   // header
        last := ctx.Cookies("lastname") // cookie
        return ctx.SendString("Hello " + first + " " + last)
    })

    request := httptest.NewRequest("GET", "/request", nil)
    request.Header.Set("firstname", "Eko")
    request.AddCookie(&http.Cookie{Name: "lastname", Value: "Khannedy"})
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Hello Eko Khannedy", string(bytes))
}
```

Route Parameter

Route Parameter

- Fiber mendukung Route Parameter, dimana kita bisa menambahkan parameter di path URL
- Ini sangat cocok ketika kita membuat RESTful API, yang butuh data dikirim via path URL
- Saat membuat Route Parameter, kita perlu memberi nama, dan di Ctx, kita bisa mengambil seluruh data menggunakan method AllParams(), atau menggunakan menthod Params(nama)

Kode : Route Parameter

```go
func TestRouteParameter(t *testing.T) {
    app.Get("/users/:userId/orders/:orderId", func(ctx *fiber.Ctx) error {
        userId := ctx.Params("userId")
        orderId := ctx.Params("orderId")
        return ctx.SendString("Get Order " + orderId + " From User " + userId)
    })

    request := httptest.NewRequest("GET", "/users/1/orders/2", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    assert.Equal(t, 200, response.StatusCode)
    bytes, err := io.ReadAll(response.Body)
    assert.Equal(t, "Get Order 2 From User 1", string(bytes))
}
```

Request Form

Request Form

- Ketika kita mengirim data menggunakan HTTP Form, kita bisa menggunakan method FormValue(name) pada Ctx untuk mendapatkan data yang dikirim nya

Kode : Request Form

```go
func TestFormRequest(t *testing.T) {
    app.Post("/hello", func(ctx *fiber.Ctx) error {
        name := ctx.FormValue("name")
        return ctx.SendString("Hello " + name)
    })

    body := strings.NewReader("name=Eko")
    request := httptest.NewRequest("POST", "/hello", body)
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Hello Eko", string(bytes))
}
```

Multipart Form

Multipart Form

- Untuk mengambil data File yang yang terdapat di Multipart Form, kita bisa menggunakan method FormFile di Ctx, namun method ini bisa menghasilkan error
- Selain itu Ctx juga memiliki method SaveFile yang bisa digunakan untuk menyimpan file

Persiapan

- Buatlah sebuah folder source dan target
- Buat file contoh.txt di folder source
- Kita akan coba upload file contoh.txt, lalu kita akan coba simpan di folder target

Kode : Multipart Form

```go
//go:embed source/contoh.txt
var contohFile []byte  1 usage

func TestFormUpload(t *testing.T) {
    app.Post("/upload", func(ctx *fiber.Ctx) error {
        file, err := ctx.FormFile("file")
        if err != nil {
            return err
        }

        err = ctx.SaveFile(file, "./target/"+file.Filename)
        if err != nil {
            return err
        }

        return ctx.SendString("Upload Success")
    })
```

```go
    body := new(bytes.Buffer)
    writer := multipart.NewWriter(body)
    file, _ := writer.CreateFormFile("file", "contoh.txt")
    file.Write(contohFile)
    writer.Close()

    request := httptest.NewRequest("POST", "/upload", body)
    request.Header.Set("Content-Type", writer.FormDataContentType())
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Upload Success", string(bytes))
}
```

Request Body

Request Body

- Saat kita membuat RESTful API, kadang kita ingin mengambil informasi request body yang dikirim oleh client, misal JSON, XML, atau yang lainnya
- Kita bisa mengambil informasi request body menggunakan method Body()

Kode : Request Body

```go
type LoginRequest struct {  1 usage
    Username string `json:"username"`
    Password string `json:"password"`
}

func TestRequestBody(t *testing.T) {
    app.Post("/login", func(ctx *fiber.Ctx) error {
        body := ctx.Body()
        request := new(LoginRequest)
        err := json.Unmarshal(body, request)
        if err != nil {
            return err
        }
        return ctx.SendString("Login Success " + request.Username)
    })
```

```go
body := strings.NewReader(`{"username":"Eko","password":"Khannedy"}`)
request := httptest.NewRequest("POST", "/login", body)
request.Header.Set("Content-Type", "application/json")
response, err := app.Test(request)
assert.Nil(t, err)
bytes, err := io.ReadAll(response.Body)
assert.Nil(t, err)
assert.Equal(t, "Login Success Eko", string(bytes))
```

Body Parser

Body Parser

- Melakukan konversi request body dari []byte menjadi struct sangat menyulitkan jika harus dilakukan manual terus menerus
- Untungnya, Fiber bisa melakukan parsing otomatis sesuai tipe data yang dikirim, dan otomatis dikonversi ke Struct
- Ada beberapa Content-Type yang didukung oleh Fiber, caranya dengan menambahkan tag pada Struct nya

Body Parser Content Type

Content Type	Struct Tag
application/x-www-form-urlencoded	form
multipart/form-data	form
application/json	json
application/xml	xml
text/xml	xml

Kode : Register Request

```
type RegisterRequest struct {  no usages
    Username string `json:"username" xml:"username" form:"username"`
    Password string `json:"password" xml:"password" form:"password"`
    Name     string `json:"name" xml:"name" form:"name"`
}
```

Kode : Body Parser

```go
func TestBodyParser(t *testing.T) {
    app.Post("/register", func(ctx *fiber.Ctx) error {
        request := new(RegisterRequest)
        err := ctx.BodyParser(request)
        if err != nil {
            return err
        }

        return ctx.SendString("Register Success " + request.Username)
    })
}
```

Kode : Body Parser JSON

```go
func TestBodyParserJSON(t *testing.T) {
    TestBodyParser(t)

    body := strings.NewReader(`{"username":"Eko","password":"Khannedy","name":"Eko Khannedy"}`)
    request := httptest.NewRequest("POST", "/register", body)
    request.Header.Set("Content-Type", "application/json")
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Register Success Eko", string(bytes))
}
```

Kode : Body Parser Form

```go
func TestBodyParserForm(t *testing.T) {
    TestBodyParser(t)

    body := strings.NewReader(`username=Eko&password=Khannedy&name=Eko+Khannedy`)
    request := httptest.NewRequest("POST", "/register", body)
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Register Success Eko", string(bytes))
}
```

Kode : Body Parser XML

```
func TestBodyParserXML(t *testing.T) {
    TestBodyParser(t)

    body := strings.NewReader(
        `<RegisterRequest>
            <username>Eko</username>
            <password>Khannedy</password>
            <name>Eko Khannedy</name>
        </RegisterRequest>`)
    request := httptest.NewRequest("POST", "/register", body)
    request.Header.Set("Content-Type", "application/xml")
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Register Success Eko", string(bytes))
```

HTTP Response

HTTP Response

- Selain sebagai representasi HTTP Request, Ctx juga digunakan sebagai representasi HTTP Response
- Sebelumnya kita sudah menggunakan SendString() untuk mengembalikan Response Body dalam bentuk String
- Ada banyak method yang tersedia yang bisa kita gunakan untuk mengubah HTTP Response

HTTP Response Method

Response Method	Keterangan
Set(key, value)	Mengubah header ke response
Status(status)	Mengubah response status
SendString(body)	Mengubah response body menjadi string
XML(body)	Mengubah response body menjadi XML
JSON(body)	Mengubah response body menjadi JSON
Redirect(path)	Mengubah response menjadi redirect ke path
Cookie(cookie)	Menambah cookie ke response

Kode : Response JSON

```go
func TestResponseJSON(t *testing.T) {
    app.Get("/user", func(ctx *fiber.Ctx) error {
        return ctx.JSON(fiber.Map{
            "username": "khannedy",
            "name":     "Eko Khannedy",
        })
    })

    request := httptest.NewRequest("GET", "/user", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, `{"name":"Eko Khannedy","username":"khannedy"}`, string(bytes))
}
```

Download File

Download File

- Sebelumnya kita sudah bahas tentang Upload File, Fiber juga bisa digunakan untuk mengembalikan response File atau byte[]
- Ada beberapa Method yang bisa kita gunakan

Method Download File

Method Download File	Keterangan
Download(file, filename)	Mengubah response menjadi isi file
Send([]byte)	Mengubah response menjadi data []byte
SendFile(file, compress)	Mengubah response menjadi isi file

Kode : Download File

```go
func TestDownloadFile(t *testing.T) {
    app.Get("/download", func(ctx *fiber.Ctx) error {
        return ctx.Download("./source/contoh.txt", "contoh.txt")
    })

    request := httptest.NewRequest("GET", "/download", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    assert.Equal(t, 200, response.StatusCode)
    assert.Equal(t, "attachment; filename=\"contoh.txt\"", response.Header.Get("Content-Disposition"))
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "This is sample text", string(bytes))
}
```

Routing Group

Routing Group

- Saat kita membuat aplikasi Web dengan banyak endpoint, kadang-kadang kita perlu melakukan Grouping beberapa routing
- Hal ini agar lebih rapi dan mudah dikembangkan ketika membuat banyak routing
- Kita bisa membuat grup menggunakan Group() di fiber.App
- Dan kita bisa menambahkan Routing ke Group() yang sudah kita buat

Kode : Routing Group

```go
func TestRoutingGroup(t *testing.T) {
    helloWorld := func(ctx *fiber.Ctx) error {
        return ctx.SendString("Hello World!")
    }

    api := app.Group("/api")
    api.Get("/hello", helloWorld)
    api.Get("/world", helloWorld)

    web := app.Group("/web")
    web.Get("/hello", helloWorld)
    web.Get("/world", helloWorld)
```

```go
request := httptest.NewRequest("GET", "/api/hello", nil)
response, err := app.Test(request)
assert.Nil(t, err)
assert.Equal(t, 200, response.StatusCode)
bytes, err := io.ReadAll(response.Body)
assert.Nil(t, err)
assert.Equal(t, "Hello World!", string(bytes))
```

Static

Static

- Saat kita membuat static file seperti HTML, CSS, JavaScript, maka akan menyulitkan jika harus membuat routing untuk tiap file
- Untungnya Fiber memiliki fitur untuk menyediakan file static menggunakan method Static()

Kode : Static

```go
func TestStatic(t *testing.T) {
    app.Static("/public", "./source")

    request := httptest.NewRequest("GET", "/public/contoh.txt", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    assert.Equal(t, 200, response.StatusCode)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "This is sample text", string(bytes))
}
```

Pre Fork

Standalone

- Secara default, saat aplikasi Fiber berjalan, dia akan berjalan secara standalone dalam satu proses
- Kadang-kadang, ketika kita jalankan dalam sebuah server yang jumlah CPU nya banyak, mungkin ada baiknya kita menjalankan beberapa proses agar semua CPU terpakai dengan optimal
- Namun yang jadi masalah, jika kita jalankan beberapa aplikasi Fiber secara bersamaan, maka kita harus menggunakan port yang berbeda-beda

Pre Fork

- Fiber memiliki konfigurasi bernama PreFork, yang default nya adalah false, dan bisa kita ubah menjadi true
- PreFork adalah fitur menjalankan beberapa proses Fiber namun menggunakan port yang sama. Fitur ini memanfaatkan fitur di sistem operasi bernama SO_REUSEPORT : https://lwn.net/Articles/542629/
- Teknik ini juga biasa digunakan di NGINX https://www.nginx.com/blog/socket-sharding-nginx-release-1-9-1/

Kode : PreFork

```go
func main() {
    app := fiber.New(fiber.Config{
        IdleTimeout:  time.Second * 5,
        ReadTimeout:  time.Second * 5,
        WriteTimeout: time.Second * 5,
        Prefork:      true,
    })

    if fiber.IsChild() {
        fmt.Println("I'm child process")
    } else {
        fmt.Println("I'm parent process")
    }

    err := app.Listen("localhost:3000")
    if err != nil {
        panic(err)
    }
```

Error Handling

Error Handling

- Kita bisa perhatikan pada Fiber Handler, return value nya berupa error, yang artinya jika terjadi error, kita bisa return langsung data errornya
- Dan ketika terjadi error, secara otomatis akan ditangkap oleh ErrorHandler pada fiber.Config
- Kita bisa mengubah implementasi dari ErrorHandler nya jika kita mau

Kode : Error Handler

```
var app = fiber.New(fiber.Config{  29 usages
    ErrorHandler: func(ctx *fiber.Ctx, err error) error {
        ctx.Status(fiber.StatusInternalServerError)
        return ctx.SendString("Error : " + err.Error())
    },
})
```

Kode : Test Error Handler

```go
func TestErrorHandling(t *testing.T) {
    app.Get("/error", func(ctx *fiber.Ctx) error {
        return errors.New("ups")
    })

    request := httptest.NewRequest("GET", "/error", nil)
    response, err := app.Test(request)
    assert.Nil(t, err)
    assert.Equal(t, 500, response.StatusCode)
    bytes, err := io.ReadAll(response.Body)
    assert.Nil(t, err)
    assert.Equal(t, "Error : ups", string(bytes))
}
```

Template

Template

- Fiber secara default tidak memiliki template engine seperti Web Framework kebanyakan
- Namun Fiber mendukung banyak integrasi dengan template engine yang sudah populer
- Kita bisa lihat semua template engine yang didukung oleh Fiber di halaman dokumentasinya :
- https://docs.gofiber.io/template/

Mustache

- Untuk menggunakan template engine, kita harus menginstall dulu dependency nya, lalu meregistrasikan View ke Fiber nya
- Misal kita akan coba menggunakan Mustache Template Engine
- Kita bisa menginstall dependency-nya terlebih dahulu
- go get github.com/gofiber/template/mustache/v2

Kode : Template

index.mustache

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{{title}}</title>
</head>
<body>
<h1>{{header}}</h1>
<p>{{content}}</p>
</body>
</html>
```

Kode : View

```
var engine = mustache.New("./template", ".mustache")  1 usage

var app = fiber.New(fiber.Config{  31 usages
    Views: engine,
    ErrorHandler: func(ctx *fiber.Ctx, err error) error {
        ctx.Status(fiber.StatusInternalServerError)
        return ctx.SendString("Error : " + err.Error())
    },
})

func TestView(t *testing.T) {
    app.Get("/view", func(ctx *fiber.Ctx) error {
        return ctx.Render("index", fiber.Map{
            "title":   "Hello Title",
            "header":  "Hello Header",
            "content": "Hello Content",
        })
    })
```

```
request := httptest.NewRequest("GET", "/view", nil)
response, err := app.Test(request)
assert.Nil(t, err)
assert.Equal(t, 200, response.StatusCode)
bytes, err := io.ReadAll(response.Body)
fmt.Println(string(bytes))
assert.Nil(t, err)
assert.Contains(t, string(bytes), "Hello Title")
assert.Contains(t, string(bytes), "Hello Header")
assert.Contains(t, string(bytes), "Hello Content")
```

Middleware

Middleware

- Fiber juga mendukung Middleware
- Dengan Middleware, kita bisa membuat Handler yang bisa melakukan sebelum dan setelah request itu dikerjakan oleh Handler yang memproses request nya
- Fiber sendiri menyediakan banyak sekali Middleware-Middleware yang umum digunakan
- Di materi ini kita tidak akan membahas Middleware yang sudah disediakan, melainkan kita akan membahas cara membuat Middleware

Membuat Middleware

- Membuat Middleware itu sederhana, cukup membuat Handler seperti pada Routing
- Namun dalam Handlernya, jika kita ingin meneruskan Request ke Handler selanjutnya, kita perlu memanggil Next() pada Ctx
- Dan untuk menggunakan Middleware, kita bisa menggunakan Use() pada fiber.App

Kode : Log Middleware

```
app := fiber.New(fiber.Config{
    IdleTimeout:  time.Second * 5,
    ReadTimeout:  time.Second * 5,
    WriteTimeout: time.Second * 5,
    Prefork:      true,
})

app.Use(func(ctx *fiber.Ctx) error {
    fmt.Println("I'm middleware before processing request")
    err := ctx.Next()
    fmt.Println("I'm middleware after processing request")
    return err
})

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

Prefix

- Middleware juga bisa ditempatkan pada prefix tertentu, misal kita ingin menggunakan middleware pada route dengan prefix /api
- Maka saat menggunakan Use(), kisa bisa tambahkan /api di parameter pertama

Kode : Prefix Middleware

```
app := fiber.New(fiber.Config{
    IdleTimeout:  time.Second * 5,
    ReadTimeout:  time.Second * 5,
    WriteTimeout: time.Second * 5,
    Prefork:      true,
})

app.Use("/api", func(ctx *fiber.Ctx) error {
    fmt.Println("I'm middleware before processing request")
    err := ctx.Next()
    fmt.Println("I'm middleware after processing request")
    return err
})

app.Get("/api/hello", func(ctx *fiber.Ctx) error {
    return ctx.SendString("Hello World")
})
```

Middleware Lainnya

Middleware Lainnya

- Fiber sudah menyediakan banyak sekali Middleware yang bisa kita gunakan secara langsung, seperti RequestId, BasicAuth, FileSystem, ETag, dan masih banyak yang lainnya
- Di kelas ini, kita tidak akan membahas Middleware tersebut, karena pasti akan selalu bertambah dan berubah, oleh karena itu sangat disarankan untuk membaca dokumentasi penggunakan Middleware nya, dan gunakan ketika memang kita butuhkan
- https://docs.gofiber.io/category/-middleware

HTTP Client

HTTP Client

- Saat kita membuat aplikasi, kadang kita juga sering menggunakan HTTP Client untuk mengambil data atau mengirim data ke API / RESTful lainnya
- Fiber menyediakan fitur untuk HTTP Client, sehingga kita bisa menggunakan Fiber sebagai HTTP Client
- Kita bisa menggunakan fiber.Client

Membuat Client

- Fiber menyediakan Client Pool, agar ketika kita menggunakan Client dan selesai menggunakannya, kita bisa menggunakan ulang Client
- Pool nya di manage oleh Fiber, jadi kita cukup ambil ketika membutuhkan menggunakan fiber.AcquireClient()
- Dan ketika selesai, kembalikan ke Pool menggunakan fiber.ReleaseClient(client)

Kode : Membuat Client

```go
func TestClient(t *testing.T) {
    client := fiber.AcquireClient()

    fiber.ReleaseClient(client)
}
```

HTTP Method

- Saat menggunakan HTTP Client, kita bisa menggunakan method yang sesuai dengan HTTP Method yang mau kita gunakan
- Misal Get, Post, Delete, Put, dan lain-lain
- Cara penggunaannya sangat mudah, hampir mirip seperti Ctx di Fiber
- https://docs.gofiber.io/api/client

Kode : HTTP Client

```go
func TestClient(t *testing.T) {
    client := fiber.AcquireClient()

    agent := client.Get("https://example.com/")
    status, response, errors := agent.String()
    assert.Nil(t, errors)
    assert.Equal(t, 200, status)
    assert.Contains(t, response, "Example Domain")

    fiber.ReleaseClient(client)
}
```

Penutup