Pemrograman Backend Lanjutan

Aditya Rizky Ramadhan

Table of contents

1 Unit Testing Basic

1.1 Testing in development

Testing atau pengujian software adalah metode untuk memeriksa program yang dibuat sudah sesuai dengan persyaratan yang diharapkan dan memastikan agar program kita tidak cacat pada saat digunakan oleh user.

Tujuan :

1. **Validasi keinginan pengguna**: Testing membantu memvalidasi apakah program yang dikembangkan sudah memenuhi kebutuhan dan keinginan pengguna.
2. **Meningkatkan kualitas program**: Melalui testing, kita dapat memastikan bahwa program yang kita tulis mengikuti best practice dalam penulisan kode, sehingga meningkatkan kualitas secara keseluruhan.
3. **Deteksi error dan edge cases**: Testing membantu dalam mendeteksi adanya error berdasarkan berbagai kemungkinan input yang diberikan, termasuk kasus-kasus ekstrem (edge cases), sehingga meminimalkan kemungkinan terjadinya bug dan kecacatan dalam program.
4. **Meningkatkan keamanan**: Testing juga dapat membantu dalam meningkatkan keamanan program dengan mengidentifikasi potensi celah keamanan atau vulnerabilitas yang dapat dieksploitasi.

Untuk apa testing :

Testing juga merupakan tanggung jawab developer / software engineer sebagai seorang profesional, karena tidak ada developer / software engineer yang menulis kode langsung jalan sesuai kebutuhan tanpa di test sama sekali.

1.2 Type of Testing

1.2.1 Manual Testing

Proses pengujian ini dilakukan oleh manusia dengan cara memasukkan input dan memeriksa output secara manual. Proses ini dilakukan secara berulang-ulang sampai mendapatkan hasil yang diinginkan sesuai dengan kebutuhan dari prasyarat.

Testing ini masih sering digunakan oleh developer dengan melakukan testing manual code program dalam skala kecil. testing ini memiliki kelemahan, yaitu tidak dapat dilakukan secara otomatis. Jika terdapat perubahan pada kode program, maka proses testing harus dilakukan ulang.

Kesimpulan: Manual testing cocok untuk testing dalam skala kecil, namun tidak cocok untuk skala besar.

1.2.2 Unit Testing

Unit testing adalah pengujian yang dilakukan secara otomatis dengan berfokus pada pengujian unit yang terkecil pada desain aplikasi. Karena dalam sebuah aplikasi banyak memiliki unit-unit kecil, maka untuk mengujinya biasanya dibuat program kecil atau main program untuk menguji unit-unit aplikasi.

Unit-unit kecil ini dapat berupa fitur atau fungsi dan pengujian unit biasanya dilakukan saat kode program dibuat.

Unit testing umumnya dilakukan oleh developer, dengan membuat code testing dalam bahasa pemrograman yang sama dengan bahasa pemrograman di aplikasi

Kesimpulan: Unit testing cocok untuk pengujian dalam skala kecil hingga menengah, namun tidak cocok untuk skala besar.

1.2.3 Integration Testing

Integration testing juga merupakan pengujian yang dilakukan secara otomatis dengan pengujian secara integrasi. Integrasi di sini adalah pengujian dari hasil penggabungan unit-unit sebagai suatu kombinasi, bukan lagi sebagai suatu unit yang individual.

Integration testing sebaiknya dilakukan secara bertahap untuk menghindari kesulitan penelusuran jika terjadi kesalahan error / bug.

Testing ini bisa dilakukan oleh developer atau tester dengan membuat code testing (pada tingkat modul / package) dari sekumpulan unit-unit yang saling terintegrasi (testing pada tingkat modul / package / library).

1.2.4 End to end testing

End to end testing, sering disingkat E2E, adalah pengujian yang dilakukan untuk memvalidasi proses kerja aplikasi dari sudut pandang pengguna / user.

Testing ini dibuat secara otomatis dengan membuat code testing yang dapat mensimulasikan tingkah laku atau tindakan-tindakan dari pengguna, hampir mirip seperti bot.

E2E testing bisa dilakukan olah Developer atau tester, tapi lebih sering dilakukan oleh tim testing khusus yang disebut software engineer in test (SEIT).

1.2.5 Acceptance Testing

Acceptance Testing adalah tahap final dari proses pengujian aplikasi / software sebelum akhirnya dirilis atau diperbarui secara komersil. Testing ini hampir sama dengan E2E testing, namun dilakukan oleh pengguna akhir.

Proses ini dapat menentukan apakah suatu fitur dapat diterima atau harus diperbaiki oleh developer / software engineer sebelum dirilis.

Jika terdapat fitur yang tidak lolos pada tahap acceptance testing, maka artinya aplikasi / software tersebut tidak memenuhi spesifikasi meskipun banyak beberapa fitur lain yang sudah memenuhi spesifikasi karena akan mengganggu proses aplikasi jika nanti dirilis.

1.2.6 Software testing pyramid

Testing pyramid adalah konsep untuk membangun aplikasi / software yang berkualitas dengan cara meminimalkan waktu yang dibutuhkan selama pengembangan dan meminimalkan kerusakan dengan membuat alur testing yang dibutuhkan.

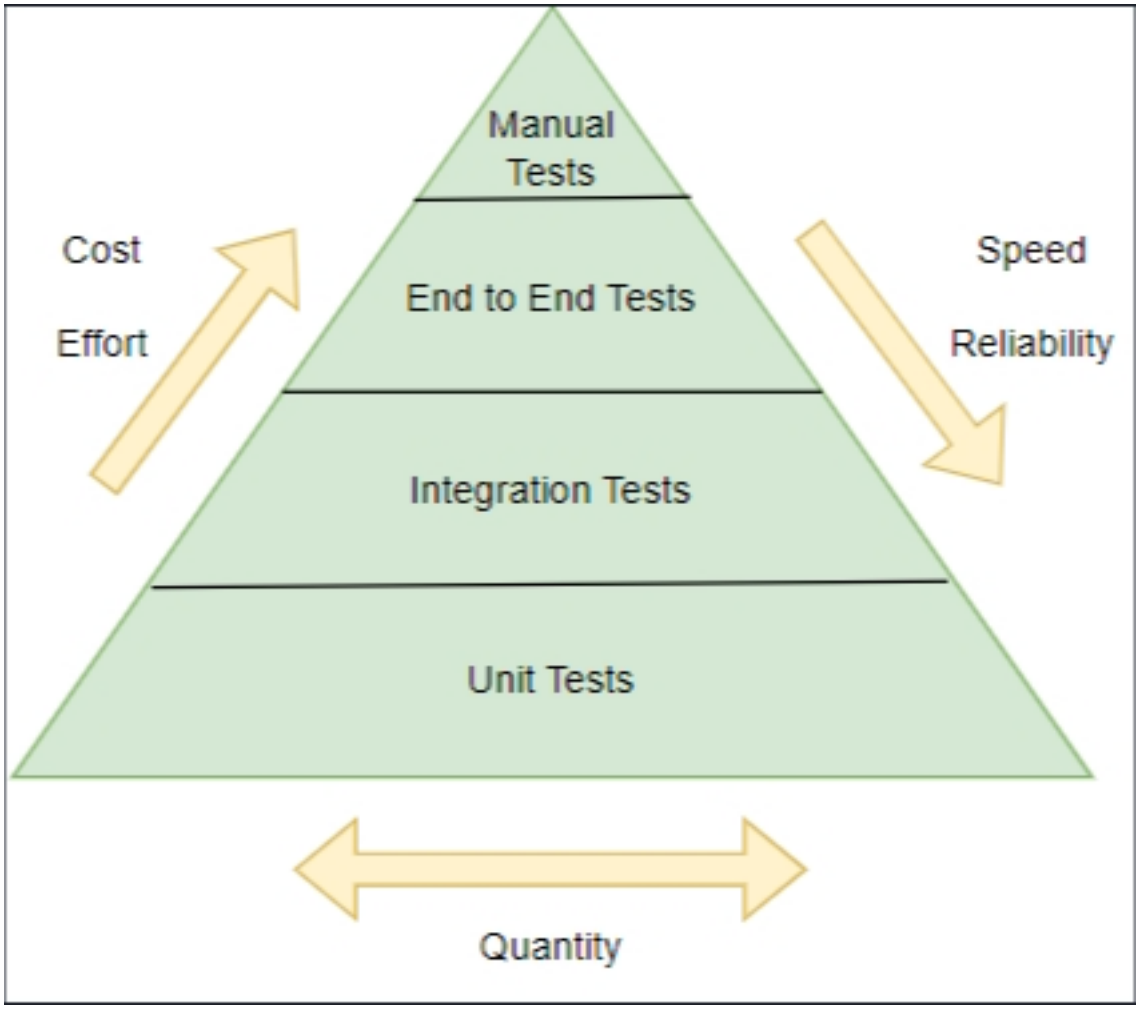

Figure 1: Hirarki Testing

Semakin ke bawah, maka semakin banyak jumlah test yang dibuat. Jumlah test yang dibuat pada tiap level testing harus seimbang, sehingga tidak terlalu banyak pada satu level dan tidak terlalu sedikit pada level lainnya.

1.3 Unit testing in Go

Contoh kode pada `main.go` sebagai berikut:

Struktur file sebagai berikut:

```
folder
   main.go
   main_test.go
```

```go
package main

import "math"

type Cube struct {
    Side float64
}

func (c Cube) Area() float64 {
    return 6 * math.Pow(c.Side, 2)
}

func (c Cube) Circumference() float64 {
    return 12 * c.Side
}

func (c Cube) Volume() float64 {
    return math.Pow(c.Side, 3)
}
```

Maka dapat dibuat unit testing pada `main_test.go` sebagai berikut:

```go
package main

import "testing"

func TestArea(t *testing.T) {
    var cube = Cube{Side: 2}
    var expected float64 = 24.0

    if cube.Area() != expected {
        t.Errorf("Expected %f, but got %f", expected, cube.Area())
    }
}

func TestCircumference(t *testing.T) {
    var cube = Cube{Side: 2}
    var expected float64 = 24.0

    if cube.Circumference() != expected {
```

```
        t.Errorf("Expected %f, but got %f", expected, cube.Circumference())
    }
}

func TestVolume(t *testing.T) {
    var cube = Cube{Side: 2}
    var expected float64 = 8.0

    if cube.Volume() != expected {
        t.Errorf("Expected %f, but got %f", expected, cube.Volume())
    }
}
```

Berikut merupakan contoh method yang disediakan oleh package `testing`:

METHOD	DESKRIPSI
Log()	Menampilkan log.
Logf()	Menampilkan log dengan format.
Fail()	Digunakan untuk menandakan bahwa proses testing gagal, namun proses testing akan tetap berlanjut.
FailNow()	Digunakan untuk menandakan bahwa proses testing gagal, dan proses testing langsung berhenti.
Failed()	Menampilkan laporan fail atau gagal.
SkipNow()	Melewati proses testing fungsi tertentu dan melanjutkan ke testing fungsi setelahnya.
Skipped()	Menampilkan laporan skip.
Error()	Memanggil Log() diikuti dengan Fail().
Errorf()	Memanggil Logf() diikuti dengan Fail().
Fatal()	Memanggil Log() diikuti dengan FailNow().
Fatalf()	Memanggil Logf() diikuti dengan FailNow().
Skip()	Memanggil Log() diikuti dengan SkipNow().
Skipf()	Memanggil Logf() diikuti dengan SkipNow().

1.4 Ginkgo

Ginkgo adalah framework (kerangka kerja) yang dibuat untuk membuat testing yang mudah dibaca dan ditulis di bahasa pemrograman Golang. Ginkgo dibuat berdasarkan konsep "Behaviour Driven Development" (BDD) dan dapat digunakan untuk segala jenis testing, seperti unit testing, integration testing, system testing, acceptance testing, dll.

1.4.1 Installation and Bootstrapping

Untuk menginstall Ginkgo, jalankan perintah berikut:

```
go install -mod=mod github.com/onsi/ginkgo/v2/ginkgo
go get -u github.com/onsi/ginkgo/v2
go get -u github.com/onsi/gomega/...
```

Setelah melakukan instalasi maka kita buat struktur project seperti dibawah ini

```
my-project/
   calculator
      calculator.go
   go.mod
   go.sum
```

Sekarang kita dapat mengisi file `calculator.go` dengan kode sebagai berikut:

```
package calculator

type Calculator struct{
    Base float64
}

func InitCalculator(base float64) *Calculator { // inisiasi kalkulator
    return &Calculator{
        Base: base,
    }
}

func (c *Calculator) Add(number float64) { // method untuk penjumlahan
    c.Base += number
}

func (c *Calculator) Subtract(number float64) { // method untuk pengurangan
    c.Base -= number
}

func (c *Calculator) Multiply(number float64) { // method untuk perkalian
    c.Base *= number
}

func (c *Calculator) Divide(number float64) error { // method untuk pembagian
    if number == 0 {
        return errors.New("cannot divide by zero")
    }

    c.Base /= number
    return nil
}
```

```
func (c *Calculator) Result() float64 { // method untuk menampilkan hasil
    return c.Base
}

func (c *Calculator) Reset() { // method untuk mereset hasil
    c.Base = 0
}
```

Setelah itu, jalankan perintah berikut untuk membuat file testing:

```
cd calculator
ginkgo bootstrap
```

setalah proses diatas maka muncul file baru pada golang sebagai berikut:

```
my-project/
   calculator
       calculator_suite_test.go
       calculator.go
   go.mod
   go.sum
```

Setelah itu dapat kita lanjutkan untuk melakukan generate unit test nya

```
ginkgo generate main
```

Maka susunan folder akan menjadi sebagai berikut

```
my-project
   calculator
       calculator_suite_test.go
       calculator_test.go
       calculator.go
   go.mod
   go.sum
```

Setelah itu file `calculator_test.go` dapat diisi sebagai berikut

```
type TestData struct {
    Spec string // deskripsi dari test case
    Base float64
    Input float64
    Expect float64
}

var _ = Describe("Calculator", func() {
    Describe("Add", func() {
```

```
        var tests = []TestData{
            {"Adding base positive number with positive number", 10, 10, 20},
            {"Adding base negative number with positive number", -5, 10, 5},
            {"Adding base positive number with negative number", 10, -20, -10},
            {"Adding base negative number with negative number", -10, -10, -20},
        }

        for _, test := range tests {
            Context(test.Spec, func() {
                It("Should return the correct result", func() {
                    calc := calculator.InitCalculator(test.Base)
                    calc.Add(test.Input)
                    Expect(calc.Result()).To(Equal(test.Expect))
                })
            })
        }
    })
})
```

Atau dapat diisi sebagai berikut:

```
var _ = Describe("Calculator", func() {
    // arrange
    var calc *calculator.Calculator

    BeforeEach(func() {
        calc = calculator.InitCalculator(5) // setiap kali pengujian, 'Base' di calculator akan d
    })

    Describe("Add", func() {
        It("Should add the base number with adder number", func() {
            calc.Add(4)
            Expect(calc.Result()).To(Equal(9.0))
        })
    })

    Describe("Subtract", func() {
        It("Should subtract the base number with subtractor number", func() {
            calc.Subtract(2)
            Expect(calc.Result()).To(Equal(3.0))
        })
    })

    Describe("Multiple", func() {
        It("Should multiply the base number with multiplier number", func() {
            calc.Multiply(10)
```

```
            Expect(calc.Result()).To(Equal(50.0))
        })
    })

    Describe("divide", func() {
        It("Should divide the base number with divider number", func() {
            calc.Divide(5)
            Expect(calc.Result()).To(Equal(1.0))
        })
    })

    AfterEach(func() {
        calc.Reset() // setiap kali selesai pengujian, 'Base' akan di reset dengan nilai 0
    })
})
```

2 Concurrency

2.1 Pengenalan

2.1.1 Apa itu concurrency?

Concurrency adalah teknik pemrograman yang digunakan untuk menyelesaikan beberapa tasks yang diselesaikan dalam waktu yang sama. Task dikerjakan bergantian dengan task lainnya, namun karena berlangsung dengan cepat sehingga terlihat seperti dilakukan bersamaan.

2.1.2 Perbedaan cocurrency dan parallelism

- **Concurrency**: Ini berkaitan dengan bagaimana cara dan struktur pengerjaan agar banyak tugas dapat dilakukan secara bersamaan, tetapi penting untuk dicatat bahwa "bersamaan" di sini tidak berarti dilakukan secara simultan. Dalam concurrency, tugas-tugas tersebut bisa dilakukan secara bergantian atau secara interleaved, yang mengarah pada kesan bahwa mereka berjalan bersamaan meskipun sebenarnya hanya satu tugas yang dieksekusi pada suatu waktu tertentu.
- **Parallelism**: Ini mengacu pada bagaimana cara melakukan banyak tugas secara benar-benar bersamaan atau serentak, biasanya menggunakan sumber daya komputasi yang terpisah. Dalam parallelism, beberapa tugas dapat dieksekusi secara bersamaan pada unit pemrosesan yang terpisah, memungkinkan untuk pemrosesan yang lebih cepat atau efisien secara keseluruhan.

2.2 Goroutine

Goroutine adalah lightweight thread yang dikelola oleh Go runtime. Cara membuat goroutine cukup dengan menambahkan keyword go sebelum fungsi yang ingin kita jalankan.

Berikut merupakan contoh kode yang menggunakan sequential dan concurrent:

```
package main

import (
    "fmt"
    "time"
)

var start time.Time

func init() {
    start = time.Now()
}
func main() {
    // Sequential: Program akan menunggu tiap pemanggilan APICall sehingga banyak waktu yang dipe
    APICallA()
    APICallB()

    // Concurrent: Tambahkan go saat melakukan APICall:
    // go APICallA()
    // go APICallB()
    time.Sleep(200 * time.Millisecond)
    fmt.Println("from main function at time", time.Since(start))
}

func APICallA() {
    time.Sleep(100 * time.Millisecond)
    fmt.Println("from APICallA at time", time.Since(start))
}

func APICallB() {
    time.Sleep(100 * time.Millisecond)
    fmt.Println("from APICallB at time", time.Since(start))
}
```

2.3 Channel dan Buffer Channel

Golang menyediakan channel untuk melakukan komunikasi antar goroutine. Channel adalah sebuah type data yang dapat digunakan untuk mengirimkan dan menerima data. Channel dapat

digunakan untuk mengirimkan data dari satu fungsi ke fungsi lainnya.

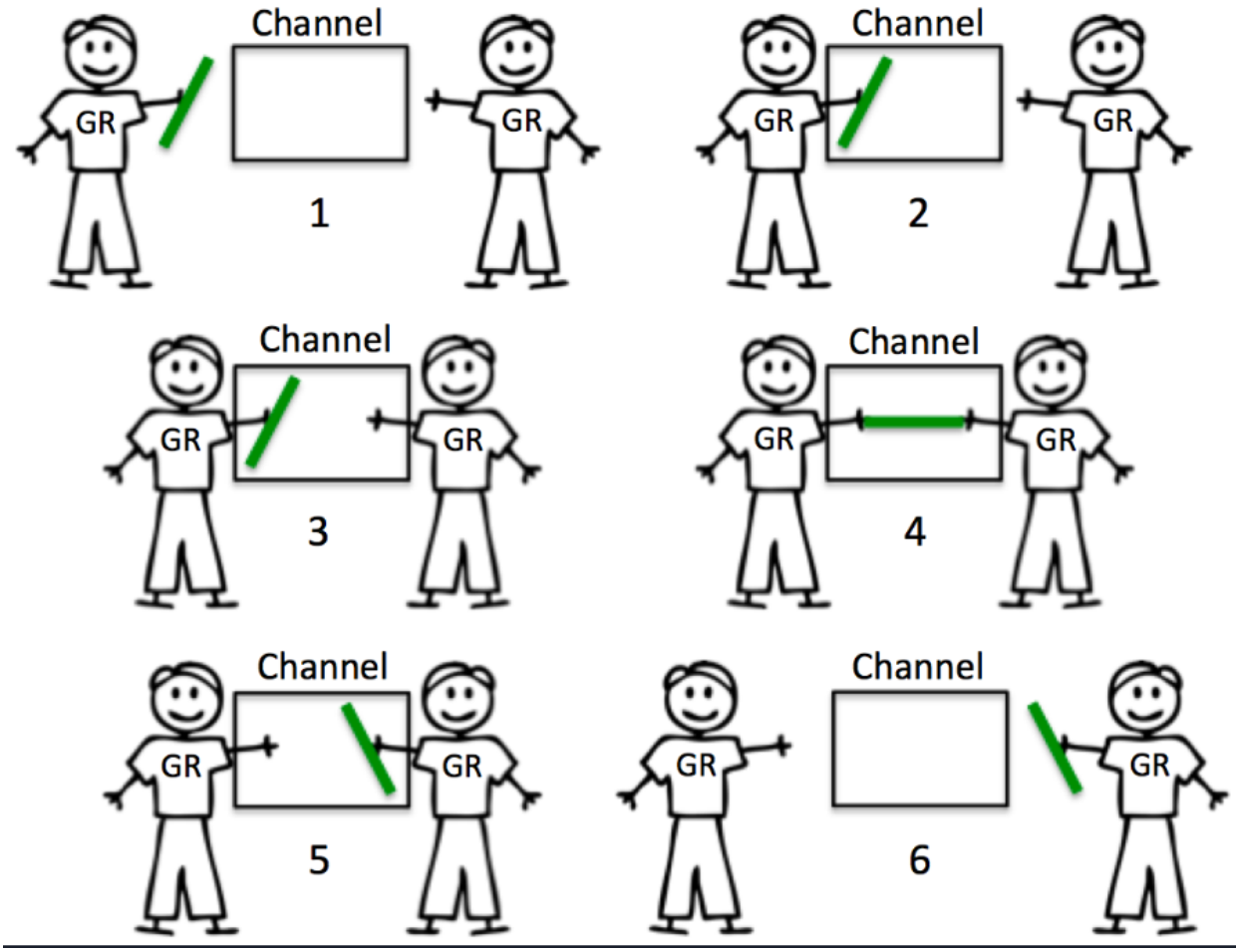

Figure 2: Ilustrasi channel

Channel adalah sebuah type data atau variable di Golang. Kita dapat mendeklarasikan channel dengan menggunakan keyword make diikuti chan diikuti dengan tipe data yang akan di kirimkan melalui channel. Seperti yang dijelaskan sebelumnya, tujuan dari channel adalah untuk mengirimkan dan menerima data antar goroutine.

```
func printMessage(msg chan string) {
    fmt.Println(<-msg)
}

func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)
```

```
    var names = []string{"Imam Permana", "Budi", "Anya Geraldine"}

    var sayHelloTo = func(who string) {
        // send data ke channel 'message'
        var data = fmt.Sprintf("hello %s", who)
        messages <- data
    }

    for _, each := range names {
        go sayHelloTo(each)
    }

    for i := 0; i < len(names); i++ {
        // fungsi ini sebagai receive data dari channel 'messages'
        printMessage(messages)
    }
}
```

Saat membuat channel, kita bisa menentukan sizenya dan itu berarti channel yang sizenya lebih dari 0 disebut dengan buffered channel.

Contoh kode untuk buffered channel:

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string, 2)

    var names = []string{"Imam Permana", "Budi", "Anya Geraldine"}

    var sayHelloTo = func(who string) {
        // send data ke channel 'message'
        var data = fmt.Sprintf("hello %s", who)
        messages <- data
    }

    for _, each := range names {
        go sayHelloTo(each)
    }

    for i := 0; i < len(names); i++ {
        // fungsi ini sebagai receive data dari channel 'messages'
        printMessage(messages)
    }
}
```

2.4 Select

Select adalah statement di Golang yang digunakan untuk memilih satu dari beberapa channel yang siap untuk digunakan. Select akan memilih channel yang siap untuk digunakan dan mengeksekusi blok kode yang sesuai.

Contoh kode select:

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)
    var quit = make(chan string)

    go func() {
        for {
            select {
            case message := <-messages:
                fmt.Println("receive message", message)
            case <-quit:
                fmt.Println("quit")
                return
            }
        }
    }()

    for i := 0; i < 5; i++ {
        messages <- fmt.Sprintf("message %d", i)
        time.Sleep(1 * time.Second)
    }

    quit <- "bye"
    time.Sleep(2 * time.Second)
}
```

2.5 Mutex

Mutex adalah singkatan dari Mutual Exclusion. Mutex digunakan untuk mengontrol akses ke suatu resource yang bersifat shared. Dengan menggunakan mutex, kita dapat memastikan bahwa hanya satu goroutine yang dapat mengakses resource tersebut pada satu waktu.

Contoh kode mutex:

```
package main

import (
```

```
    "fmt"
    "sync"
)

var x = 0
var wg sync.WaitGroup

func increment() {
    x = x + 1
    wg.Done()
}

func main() {
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go increment()
    }

    wg.Wait()
    fmt.Println("final value of x", x)
}
```

2.6 WaitGroup

WaitGroup adalah sebuah struct yang digunakan untuk menunggu sekelompok goroutine selesai. WaitGroup memiliki tiga method yaitu Add, Done, dan Wait.

- Add: digunakan untuk menambahkan jumlah goroutine yang akan dijalankan.
- Done: digunakan untuk mengurangi jumlah goroutine yang sedang berjalan.
- Wait: digunakan untuk menunggu semua goroutine selesai.

Contoh kode WaitGroup:

```
package main

import (
    "fmt"
    "sync"
)

var x = 0
var wg sync.WaitGroup

func increment() {
    x = x + 1
```

```
    wg.Done()
}

func main() {
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go increment()
    }

    wg.Wait()
    fmt.Println("final value of x", x)
}
```