```go
package server

import (
	"ProtectMyBike/clients"
	"time"
)

type InterfaceClient interface {
	Create(name string, idObject clients.InterfaceID, server clients.InterfaceClient) *clients.Client
	Deactivate(id uint64, idObject clients.InterfaceID, server clients.InterfaceClient) *clients.Client
	GetAll(id uint64, idObject clients.InterfaceID, server clients.InterfaceClient)  *map[uint64]*clients.Client
}


func CreateClient(action bool, id uint64) *clients.Client {
	idObject, server := clients.InterfaceConnection()
	var cli *clients.Client
	if action{
		cli = Create("raynard", idObject, server)
	}
	//fmt.Println(server.GetAClient(cli.ID))
	return cli
}


func Create(name string, idObject clients.InterfaceID, server clients.InterfaceClient) *clients.Client{
	//idObject, server := clients.InterfaceConnection()
	cli := server.PairClient(idObject)
	//fmt.Println(cli)
	cli.Name = name
	cli.Secure = true
	cli.Duration = time.Now().UTC()
	clie := server.GetAClient(cli.ID)
	//fmt.Println(clie)
	return clie
}



//Deactivate
// takes in the interfaces
// deactivate an id
func Deactivate(id uint64, idObject clients.InterfaceID, server clients.InterfaceClient) *clients.Client {
	clie := server.GetAClient(id)
	clie.Secure = false
	clie2 := server.GetAClient(id)
	return clie2
}


func GetAll(id uint64, idObject clients.InterfaceID, server clients.InterfaceClient)  *map[uint64]*clients.Client {
	clie := server.GetAllClients()

	return clie
}

```

```go
package server

import (
    "fmt"
)

func Gyroscope(x,y,z int) bool {
    var threshold = 15  //degree
    if x > threshold || y > threshold || z >
threshold{
        return true
    }
    return false
}

func Accelerometer(x, y, z int) bool {
    var threshold = 4   //meter/sec
    if x > threshold || y > threshold || z >
threshold{
        return true
    }
    return false
}
//
func CheckSecurity(input [][]int) error {
    //A := make(chan int, 3)
    //B := make(chan int, 3)

    //loop:
    for {

        t := true
        fmt.Println("*************")
        //A <-rand.Intn(16)
        //A <-rand.Intn(16)
        //A <-rand.Intn(16)
        //B <-rand.Intn(5)
        //B <-rand.Intn(5)
        //B <-rand.Intn(5)
        //wg.Add(1)
        //go Notify(threshold, A, B, wg)
        fmt.Println("Reading Sensor...")
        //(input[1][0], input[1][1], input[1][2])
        if !Accelerometer(input[1][0], input[1][1
], input[1][2]) {
            //threshold <- true
            t = false
            ActivateStolen("Displacement")
        }
        //s1 := [][]int{{1,3,1}}

        if !Gyroscope(input[0][0], input[0][1],
input[0][2])  {
            t = false
            ActivateStolen("Angular Velocity")
        }
        if !t {
            break
        }
        fmt.Println("system secure")
    }
    return nil
}
func ActivateStolen(input string)  {
    SendMessage(input)
}

func SendMessage(input string)  {
    //Notify Owner
    fmt.Printf("Notify  owner of theft, System
detected change in %v\n", input)
}
```

```go
package server

import (
    "ProtectMyBike/clients"
    "fmt"
    "github.com/stretchr/testify/assert"
    "testing"
)

func TestCreateClient(t *testing.T) {
        cli := CreateClient(true, 0)
        assert.NotNil(t, cli)
        fmt.Println(cli)
}


func TestDeactivate(t *testing.T) {
    idObject, server := clients.InterfaceConnection
()
    var cli *clients.Client

    cli = Create("raynard", idObject, server)

    cl := Deactivate( cli.ID , idObject, server)

    fmt.Println(cl)
    assert.NotNil(t, cli)

}

```

```go
package server

import (
    "fmt"
    "github.com/stretchr/testify/assert"
    "sync"
    "testing"
)

func TestCheckSecurity(t *testing.T) {
    var wg sync.WaitGroup
    wg.Add(1)
    var err error
    //threshold := make(chan bool, 1)
    input := [][]int{
        {1,3,3},
        {2,3,1},
    }
    go func() {
        err = CheckSecurity(input)
        wg.Done()
    }()
    wg.Wait()
    assert.NoError(t, err)
    //<- threshold
}

func TestAccelerometer(t *testing.T) {
    s := [][]int{{1,5,1}, {5,1,1}, {1,1,5}}
    for _, j := range s{

            b := Accelerometer(j[0], j[1], j[2])
            fmt.Println(
                b)
            assert.Equal(t, true, b, fmt.Sprintf("%
v %v %v",j[0], j[1], j[2]))
    }
    s1 := [][]int{{1,3,1}}
    for _, j := range s1{
        //for h,j := range v{
        //  fmt.Println(h, j)
        b := Accelerometer(j[0], j[1], j[2])
        fmt.Println(
            b)
        assert.Equal(t, false, b, fmt.Sprintf("%v %v %v",j[0], j[1], j[2]))
    }

}

func TestGyroscope(t *testing.T) {
    s := [][]int{{1,16,1}, {16,1,1}, {1,1,16}}
    for _, j := range s{

        b := Gyroscope(j[0], j[1], j[2])
        assert.Equal(t, true, b, fmt.Sprintf("%v %v %v",j[0], j[1], j[2]))
    }
    s1 := [][]int{{1,3,1}}
    for _, j := range s1{
        //for h,j := range v{
        //  fmt.Println(h, j)
        b := Gyroscope(j[0], j[1], j[2])
        assert.Equal(t, false, b, fmt.Sprintf("%v %v %v",j[0], j[1], j[2]))
    }

}
```