

A large, solid yellow circle is positioned on the left side of the slide, partially cut off by the edge.

Migrating From MVC to Clean Code

A large, light yellow circle is positioned on the right side of the slide, partially cut off by the edge.

Before Migrate


Disini kita akan memakai contoh project mvc yang ada para repository berikut :

- mvc

The goal ketika kita menggunakan clean code adalah kode kita lebih modular, scallabel, dan maintainable.

- Modular dalam artian kita bisa dengan mudah mengganti dipendensi satu ke dipendensi lain
- Scallabel dalam artian kita dapat dengan mudah menambahkan feature baru dan lain sebagainya
- Maintainable dalam artian kita dapat dengan mudah memperbaiki issue bilamana terdapat issue pada kode kita

Controller



```
userController.go

func (u *userController) Create(c echo.Context) error {
    var payloads []CreateOrderRequest
    if err := c.Bind(&payloads); err != nil {
        return err
    }
    var (
        orderData model.Order
        orderItems []model.OrderItem
    )
    for _, payload := range payloads {
        var product model.Product
        err := u.db.Model(&product).
            Where("id = ?", payload.ProductID).
            First(&product).Error;
        if err != nil {
            return err
        }
    }
    ...
}
```

Kalau kita lihat kode disamping, sangat tidak mencerminkan goal yang akan kita tuju.

- Sangat terikat dengan dependensi contohnya gorm.

Controller

```
userController.go

...
price := product.Price * float64(payload.Qty)
// add discount 10% if qty > 5
if payload.Qty > 5 {
    price = price - (float64(10) / float64(100) *
price)
}
orderData.TotalPrice += price

orderItems = append(orderItems, model.OrderItem{
    ProductID: payload.ProductID,
    Qty:       payload.Qty,
    Price:     price,
})
}
orderData.OrderItems = orderItems
if err := u.db.Create(&orderData).Error; err != nil {
    return err
}
return c.String(http.StatusOK, "Success")
}
```

- Bisnis logic atau usecase berada dalam fungsi yang sama sehingga akan susah ketika nantinya kode program kita akan di scaling atau di maintenance.

Options Migrating

Ada 3 opsi bila kita mau melakukan migrasi arsitektur kode dari mvc ke clean code :

1. Pertahankan desain sekarang dengan memisahkan dependensi.
2. Pertahankan desainnya tetapi kita pindahkan kodenya kedalam suatu layer.
3. Ubah desainnya dan pisahkan dependensi.

Pada contoh kali ini kita akan memakai cara kedua, dimana kita akan memisahkan kode kita ke dalam layer layer yaitu usecase dan repository.

Structure Code

```

✓ clean-code
  > controller
  > dto
  > model
✓ repository
  -GO order_repository.go
  -GO product_repository.go
✓ usecase
  -GO order_usecase.go
  -GO main.go
  -GO route.go
  go.mod
  go.sum

```

Berikut struktur kode yang akan kita pakai nantinya,

- **Controller** : berisi kode yang berhubungan langsung ke user (interface layer)
- **Repository** : berisi kode yang berhubungan langsung ke database (interface layer)
- **Usecase** : berisi bisnis logik yang dipakai

Di dalam folder usecase berisi file **order_usecase.go**

```
order_usecase.go

type OrderUsecase interface {
    Create(payloads []dto.CreateOrderRequest) error
}

type orderUsecase struct {
    orderRepository repository.OrderRepository
    productRepository repository.ProductRepository
}

func NewOrderUsecase(
    orderRepo repository.OrderRepository,
    productRepo repository.ProductRepository,
) *orderUsecase {
    return &orderUsecase{
        orderRepository: orderRepo,
        productRepository: productRepo,
    }
}
```

- Blueprint dari order usecase
- Memakai interface dari repository bukan struct dari repository

Repository

Order repository :

```
order_repository.go

type OrderRepository interface {
    Create(data model.Order) error
}

type orderRepository struct {
    db *gorm.DB
}

func NewOrderRepository(db
*gorm.DB) *orderRepository {
    return &orderRepository{db}
}
```

- **OrderRepository** adalah blueprint dari repository order
- Perlu memakai sharing-by-pointer untuk variabel yang menyimpan koneksi dari pada memakai sharing-by-value

Repository

Product repository :

```
product_repository.go

type ProductRepository interface {
    Find(productID uint) (model.Product,
    error)
}

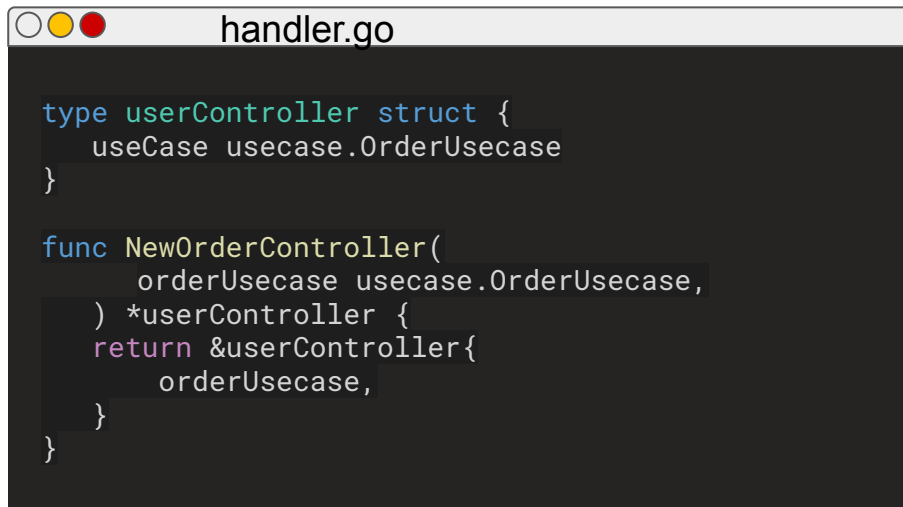
type productRepository struct {
    db *gorm.DB
}

func NewProductRepository(db *gorm.DB)
*productRepository {
    return &productRepository{db}
}
```

- **ProductRepository** adalah blueprint dari repository product
- Perlu memakai sharing-by-pointer untuk variabel yang menyimpan koneksi dari pada memakai sharing-by-value

Controller

Selanjutnya kita sedikit ubah pada controllernya



```
type userController struct {  
    useCase usecase.OrderUsecase  
}  
  
func NewOrderController(  
    orderUsecase usecase.OrderUsecase,  
) *userController {  
    return &userController{  
        orderUsecase,  
    }  
}
```

- Kita ganti field dari struct `userController` dari `gorm.DB` ke interface `orderUsecase`.
- **NewOrderController** adalah fungsi yang digunakan untuk membuat instance dari struct **orderController**

Controller

```
handler.go

func (u *UserController) Create(c echo.Context)
error {

    var payloads []dto.CreateOrderRequest

    if err := c.Bind(&payloads); err != nil {
        return err
    }

    err := u.useCase.Create(payloads)
    if err != nil {
        return err
    }

    return c.String(http.StatusOK, "Success")
}
```

Lalu kita bisa memakai method dari usecase

Result

Bisa kita lihat, kode program kita sekarang terlihat jauh lebih rapi dan mudah dimaintenan dari pada sebelumnya. Hasil nya bisa dilihat di repo ini

- Clean Code

