Applying The Clean Architecture to Go applications (Manuel Kiessling)

Visualization is done by Eduard Sesigin

DOMAIN

$GOPATH/src/domain/domain.go
package domain

```
import (
    "errors"
)


type Customer struct {
    Id   int
    Name string
}

type Item struct {
    Id        int
    Name      string
    Value     float64
    Available bool
}

type Order struct {
    Id       int
    Customer Customer
    Items    []Item
}

func (order *Order) Add(item Item) error {
    if !item.Available {
        return errors.New("Cannot add unavailable items to order")
    }
    if order.value()+item.Value > 250.00 {
        return errors.New(`An order may not exceed
            a total value of $250.00`)
    }
    order.Items = append(order.Items, item)
    return nil
}

func (order *Order) value() float64 {
    sum := 0.0
    for i := range order.Items {
        sum = sum + order.Items[i].Value
    }
    return sum
}
```

$GOPATH/src/domain/domain.go
package domain

```
import (
    "errors"
)


type ItemRepository interface {
    Store(item Item)
    FindById(id int) Item
}

type OrderRepository interface {
    Store(order Order)
    FindById(id int) Order
}

type CustomerRepository interface {
    Store(customer Customer)
    FindById(id int) Customer
}
```

USE CASES

$GOPATH/src/usecases/usecases.go
package usecases

```
type OrderInteractor struct {
    UserRepository  UserRepository
    OrderRepository domain.OrderRepository
    ItemRepository  domain.ItemRepository
    Logger          Logger
}

func (interactor *OrderInteractor) Items(userId, orderId int) ([]Item,
error) {
    var items []Item
    user := interactor.UserRepository.FindById(userId)
    order := interactor.OrderRepository.FindById(orderId)
    if user.Customer.Id != order.Customer.Id {
        ...
        interactor.Logger.Log(err.Error())
        ...
        return items, err
    }
    items = make([]Item, len(order.Items))
    for i, item := range order.Items {
        items[i] = Item{item.Id, item.Name, item.Value}
    }
    return items, nil
}

func (interactor *OrderInteractor) Add(userId, orderId, itemId int) error {
    var message string
    user := interactor.UserRepository.FindById(userId)
    order := interactor.OrderRepository.FindById(orderId)
    if user.Customer.Id != order.Customer.Id {
        ...
        interactor.Logger.Log(err.Error())
        return err
    }
    item := interactor.ItemRepository.FindById(itemId)
    if domainErr := order.Add(item); domainErr != nil {
        ...
        interactor.Logger.Log(err.Error())
        return err
    }
    interactor.OrderRepository.Store(order)
    interactor.Logger.Log(fmt.Sprintf(
        "User added item '%s' (#%i) to order #%i",
        item.Name, item.Id, order.Id))
    return nil
}
```

$GOPATH/src/usecases/usecases.go
package usecases

```
import (
    "domain"
    "fmt"
)

type UserRepository interface {
    Store(user User)
    FindById(id int) User
}

type User struct {
    Id       int
    IsAdmin  bool
    Customer domain.Customer
}

type Item struct {
    Id    int
    Name  string
    Value float64
}

type Logger interface {
    Log(message string) error
}
```

$GOPATH/src/usecases/usecases.go
package usecases

```
type AdminOrderInteractor struct {
    OrderInteractor
}

func (interactor *AdminOrderInteractor) Add(userId, orderId, itemId int)
error {
    var message string
    user := interactor.UserRepository.FindById(userId)
    order := interactor.OrderRepository.FindById(orderId)
    if !user.IsAdmin {
        ...
        interactor.Logger.Log(err.Error())
        return err
    }
    item := interactor.ItemRepository.FindById(itemId)
    if domainErr := order.Add(item); domainErr != nil {
        ...
        interactor.Logger.Log(err.Error())
        return err
    }
    interactor.OrderRepository.Store(order)
    interactor.Logger.Log(fmt.Sprintf(
        "Admin added item '%s' (#%i) to order #%i",
        item.Name, item.Id, order.Id))
    return nil
}
```

INTERFACES

$GOPATH/src/interfaces/repositories.go
package interfaces

```
type DbUserRepo DbRepo

func NewDbUserRepo(dbHandlers map[string]DbHandler)
*DbUserRepo {
    dbUserRepo := new(DbUserRepo)
    dbUserRepo.dbHandlers = dbHandlers
    dbUserRepo.dbHandler = dbHandlers["DbUserRepo"]
    return dbUserRepo
}

func (repo *DbUserRepo) Store(user usecases.User) {
    isAdmin := "no"
    if user.IsAdmin {
        isAdmin = "yes"
    }
    repo.dbHandler.Execute(fmt.Sprintf(`INSERT INTO users (id,
customer_id, is_admin)
                        VALUES ('%d', '%d', '%v')`,
                        user.Id, user.Customer.Id, isAdmin))
    customerRepo := NewDbCustomerRepo(repo.dbHandlers)
    customerRepo.Store(user.Customer)
}

func (repo *DbUserRepo) FindById(id int) usecases.User {
    row := repo.dbHandler.Query(fmt.Sprintf(`SELECT is_admin,
customer_id
                        FROM users WHERE id = '%d' LIMIT 1`,
                        id))
    var isAdmin string
    var customerId int
    row.Next()
    row.Scan(&isAdmin, &customerId)
    customerRepo := NewDbCustomerRepo(repo.dbHandlers)
    u := usecases.User{Id: id, Customer:
customerRepo.FindById(customerId)}
    u.IsAdmin = false
    if isAdmin == "yes" {
        u.IsAdmin = true
    }
    return u
}
```

$GOPATH/src/interfaces/repositories.go
package interfaces

```
type DbItemRepo DbRepo

func NewDbItemRepo(dbHandlers map[string]DbHandler)
*DbItemRepo {
    dbItemRepo := new(DbItemRepo)
    dbItemRepo.dbHandlers = dbHandlers
    dbItemRepo.dbHandler = dbHandlers["DbItemRepo"]
    return dbItemRepo
}

func (repo *DbItemRepo) Store(item domain.Item) {
    available := "no"
    if item.Available {
        available = "yes"
    }
    repo.dbHandler.Execute(fmt.Sprintf(`INSERT INTO items (id, name,
value, available)
                        VALUES ('%d', '%v', '%f', '%v')`,
                        item.Id, item.Name, item.Value, available))
}

func (repo *DbItemRepo) FindById(id int) domain.Item {
    row := repo.dbHandler.Query(fmt.Sprintf(`SELECT name, value,
available
                        FROM items WHERE id = '%d' LIMIT 1`,
                        id))
    var name string
    var value float64
    var available string
    row.Next()
    row.Scan(&name, &value, &available)
    item := domain.Item{Id: id, Name: name, Value: value}
    item.Available = false
    if available == "yes" {
        item.Available = true
    }
    return item
}
```

$GOPATH/src/interfaces/repositories.go
package interfaces

```
type DbOrderRepo DbRepo

func NewDbOrderRepo(dbHandlers map[string]DbHandler)
*DbOrderRepo {
    dbOrderRepo := new(DbOrderRepo)
    dbOrderRepo.dbHandlers = dbHandlers
    dbOrderRepo.dbHandler = dbHandlers["DbOrderRepo"]
    return dbOrderRepo
}
func (repo *DbOrderRepo) Store(order domain.Order) {
    repo.dbHandler.Execute(fmt.Sprintf(`INSERT INTO orders (id,
customer_id)
                        VALUES ('%d', '%v')`,
                        order.Id, order.Customer.Id))
    for _, item := range order.Items {
        repo.dbHandler.Execute(fmt.Sprintf(`INSERT INTO items2orders
(item_id, order_id)
                        VALUES ('%d', '%d')`,
                        item.Id, order.Id))
    }
}
func (repo *DbOrderRepo) FindById(id int) domain.Order {
    row := repo.dbHandler.Query(fmt.Sprintf(`SELECT customer_id
FROM orders
                        WHERE id = '%d' LIMIT 1`,
                        id))
    var customerId int
    row.Next()
    row.Scan(&customerId)
    customerRepo := NewDbCustomerRepo(repo.dbHandlers)
    order := domain.Order{Id: id, Customer:
customerRepo.FindById(customerId)}
    var itemId int
    itemRepo := NewDbItemRepo(repo.dbHandlers)
    row = repo.dbHandler.Query(fmt.Sprintf(`SELECT item_id FROM
items2orders
                        WHERE order_id = '%d'`,
                        order.Id))
    for row.Next() {
        row.Scan(&itemId)
        order.Add(itemRepo.FindById(itemId))
    }
    return order
}
```

$GOPATH/src/interfaces/repositories.go
package interfaces

```
type DbCustomerRepo DbRepo

func NewDbCustomerRepo(dbHandlers map[string]DbHandler)
*DbCustomerRepo {
    dbCustomerRepo := new(DbCustomerRepo)
    dbCustomerRepo.dbHandlers = dbHandlers
    dbCustomerRepo.dbHandler = dbHandlers["DbCustomerRepo"]
    return dbCustomerRepo
}

func (repo *DbCustomerRepo) Store(customer domain.Customer) {
    repo.dbHandler.Execute(fmt.Sprintf(`INSERT INTO customers (id,
name)
                        VALUES ('%d', '%v')`,
                        customer.Id, customer.Name))
}

func (repo *DbCustomerRepo) FindById(id int) domain.Customer {
    row := repo.dbHandler.Query(fmt.Sprintf(`SELECT name FROM
customers
                        WHERE id = '%d' LIMIT 1`,
                        id))
    var name string
    row.Next()
    row.Scan(&name)
    return domain.Customer{Id: id, Name: name}
}
```

$GOPATH/src/interfaces/webservice.go
package interfaces

```
import (
    "fmt"
    "io"
    "net/http"
    "strconv"
    "usecases"
)

type OrderInteractor interface {
    Items(userId, orderId int) ([]usecases.Item, error)
    Add(userId, orderId, itemId int) error
}

type WebserviceHandler struct {
    OrderInteractor OrderInteractor
}

func (handler WebserviceHandler) ShowOrder(res
http.ResponseWriter, req *http.Request) {
    userId, _ := strconv.Atoi(req.FormValue("userId"))
    orderId, _ := strconv.Atoi(req.FormValue("orderId"))
    items, _ := handler.OrderInteractor.Items(userId, orderId)
    for _, item := range items {
        io.WriteString(res, fmt.Sprintf("item id: %d\n", item.Id))
        io.WriteString(res, fmt.Sprintf("item name: %v\n", item.Name))
        io.WriteString(res, fmt.Sprintf("item value: %f\n", item.Value))
    }
}
```

INFRASTUCTURE

$GOPATH/main.go
package main

```
import (
    "usecases"
    "interfaces"
    "infrastructure"
    "net/http"
)

func main() {
    dbHandler :=
infrastructure.NewSqliteHandler("/var/tmp/production.sqlite")

    handlers := make(map[string] interfaces.DbHandler)
    handlers["DbUserRepo"] = dbHandler
    handlers["DbCustomerRepo"] = dbHandler
    handlers["DbItemRepo"] = dbHandler
    handlers["DbOrderRepo"] = dbHandler

    orderInteractor := new(usecases.OrderInteractor)
    orderInteractor.UserRepository =
interfaces.NewDbUserRepo(handlers)
    orderInteractor.ItemRepository =
interfaces.NewDbItemRepo(handlers)
    orderInteractor.OrderRepository =
interfaces.NewDbOrderRepo(handlers)

    webserviceHandler := interfaces.WebserviceHandler{}
    webserviceHandler.OrderInteractor = orderInteractor

    http.HandleFunc("/orders", func(res http.ResponseWriter, req
*http.Request) {
        webserviceHandler.ShowOrder(res, req)
    })
    http.ListenAndServe(":8080", nil)
}
```

$GOPATH/src/infrastructure/sqlitehandler.go
package infrastructure

```
import (
    "database/sql"
    "fmt"
    _ "github.com/mattn/go-sqlite3"
    "interfaces"
)

type SqliteHandler struct {
    Conn *sql.DB
}

func (handler *SqliteHandler) Execute(statement string) {
    handler.Conn.Exec(statement)
}

func (handler *SqliteHandler) Query(statement string) interfaces.Row {
    rows, err := handler.Conn.Query(statement)
    if err != nil {
        fmt.Println(err)
        return new(SqliteRow)
    }
    row := new(SqliteRow)
    row.Rows = rows
    return row
}

type SqliteRow struct {
    Rows *sql.Rows
}

func (r SqliteRow) Scan(dest ...interface{}) {
    r.Rows.Scan(dest...)
}

func (r SqliteRow) Next() bool {
    return r.Rows.Next()
}

func NewSqliteHandler(dbfileName string) *SqliteHandler {
    conn, _ := sql.Open("sqlite3", dbfileName)
    sqliteHandler := new(SqliteHandler)
    sqliteHandler.Conn = conn
    return sqliteHandler
}
```

$GOPATH/src/interfaces/repositories.go
package interfaces

```
import (
    "domain"
    "fmt"
    "usecases"
)

type DbHandler interface {
    Execute(statement string)
    Query(statement string) Row
}

type Row interface {
    Scan(dest ...interface{})
    Next() bool
}

type DbRepo struct {
    dbHandlers map[string]DbHandler
    dbHandler  DbHandler
}
```