

# Practical guide to testing Go services

Conf42 Golang

4 Apr 2025

Nikolay Kuznetsov

@nikolayk812

# About me

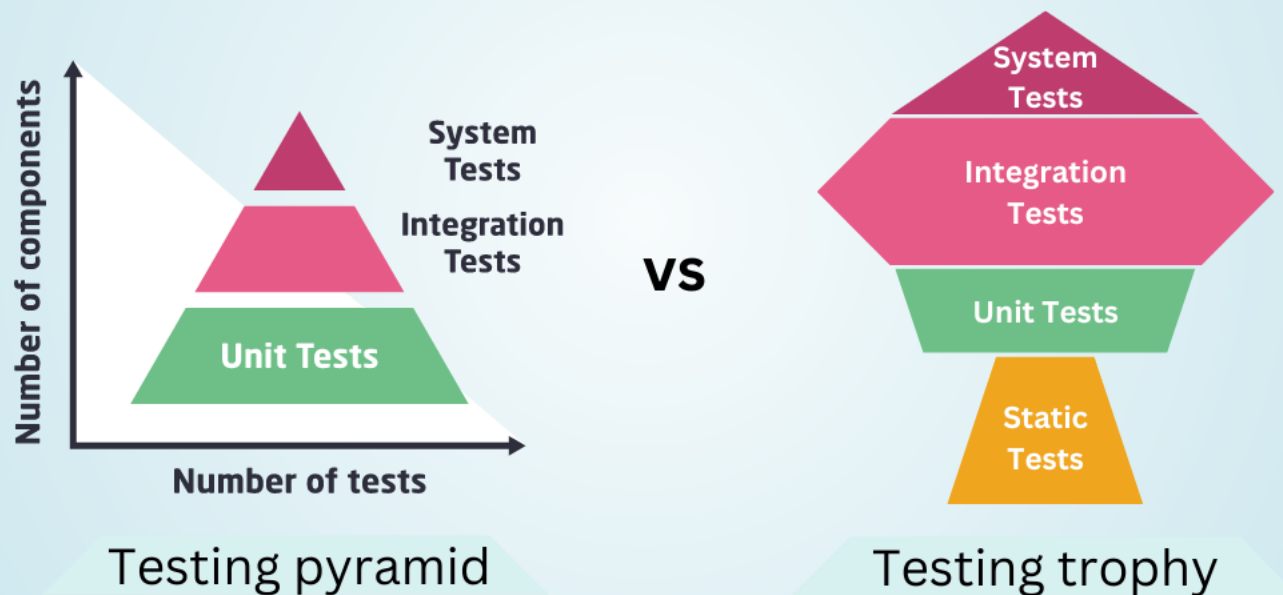
Senior software engineer

Pre-owned project at Zalando Helsinki

C → Java → Kotlin → Go

Author of [pgx-outbox](#) library

# Pyramid vs trophy



# Principles

Layer-by-layer testing

Table tests everywhere

Comprehensive comparison

Randomized test data

# Service under test

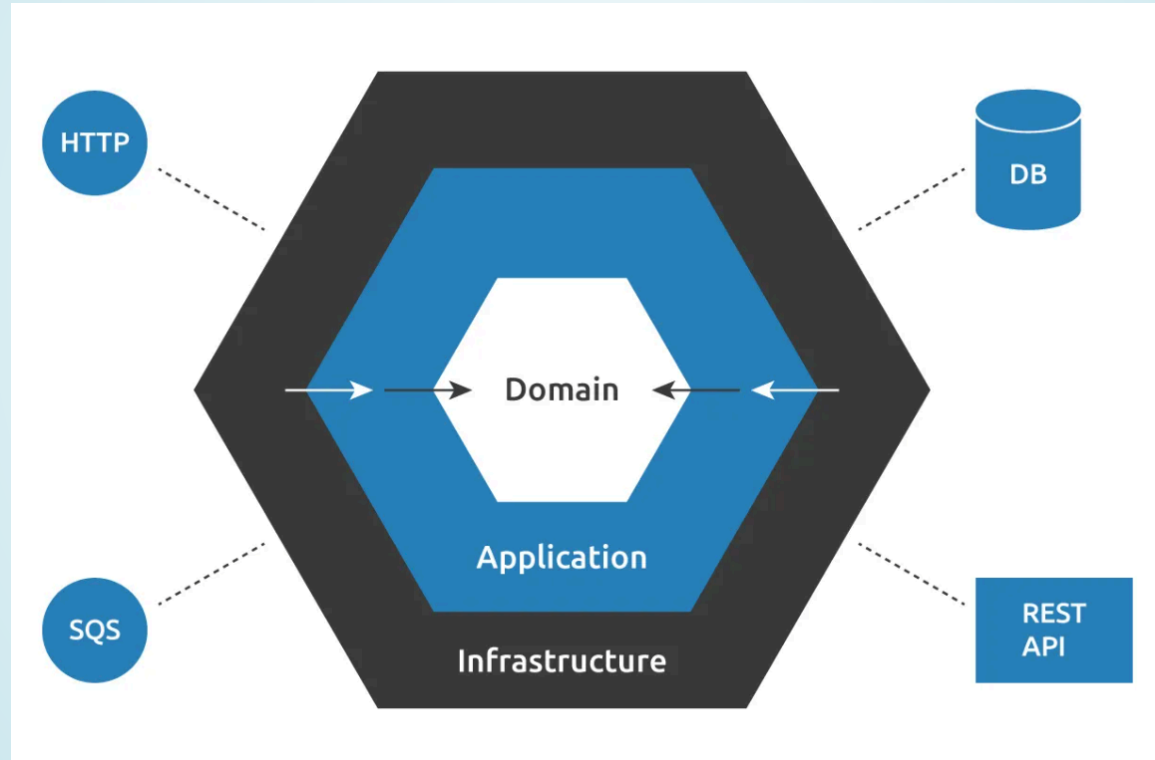
cart management of unique items

*pgx* driver for Postgres

*gin* framework for HTTP

hexagonal-like architecture

# Hexagonal architecture



# Repository layer

test against a real database

production engine version

spin up containers from Go code:

***testcontainers-go, ory/dockertest***

# Testcontainers modules

Postgres

Localstack

Kafka

Ollama

30+ more





# Testcontainers

```
import "github.com/testcontainers/testcontainers-go/modules/postgres"

pgContainer, err := postgres.Run(ctx, "postgres:17.4-alpine",
    postgres.BasicWaitStrategies(),
    postgres.WithInitScripts("migrations/01_cart_items.up.sql"),
)

//postgres://postgres:postgres@localhost:32847/postgres?sslmode=disable
connStr, err := pgContainer.ConnectionString(ctx, "sslmode=disable")
```

# Containers lifecycle

Prefer ***stretchr/testify/suite*** over

- *TestMain*
- *sync.Once*
- per each test

# Introduce suite

```
import "github.com/stretchr/testify/suite"

type cartRepoSuite struct {
    suite.Suite

    container testcontainers.Container
}

// entry point to run all tests in the suite
func TestCartRepoSuite(t *testing.T) {
    suite.Run(t, new(cartRepoSuite))
}
```

# Setup suite

```
import "github.com/jackc/pgx/v5/pgxpool"

func (s *cartRepoSuite) SetupSuite() {
    s.container, connStr, err = startPostgres(ctx)
    suite.NoError(err)

    pool, err := pgxpool.New(ctx, connStr)
    repo := repository.New(pool)
}

func (s *cartRepoSuite) TearDownSuite() {
    suite.NoError(s.container.Terminate(ctx)) // helping Ryuk
}
```

# Attach tests

```
func (s *cartRepoSuite) TestAddItem() {  
    // configure test cases  
    for _, tt := range tests {  
        suite.Run(tc.name, func() {  
            t := suite.T()  
  
            err := s.repo.AddItem(ctx, tt.ownerID, tt.item)  
            require.NoError(t, err)  
  
            // get actual cart  
            assertCart(t, tt.expectedCart, actualCart) // using go-cmp  
        })  
    }  
}
```

# go-cmp

semantic equality

more powerful than *reflect.DeepEqual*

customizable using options

not official Google product

# Use *go-cmp*

```
func assertCart(t *testing.T, expected, actual Cart) {  
    // custom comparer for golang.org/x/text/currency  
    comparer := cmp.Comparer(func(x, y currency.Unit) bool {  
        return x.String() == y.String()  
    })  
  
    opts := cmp.Options{  
        cmpopts.IgnoreFields(CartItem{}, "CreatedAt"),  
        cmpopts.EquateEmpty(), // treat empty and nil slices equally  
    }  
  
    diff := cmp.Diff(expected, actual, comparer, opts)  
    assert.Empty(t, diff)  
}
```

# Without comparer

```
=== RUN    TestCartRepositorySuite/TestAddItem/single_item:_ok
options.go:256: test panicked: cannot handle unexported field at
{Cart}.Items[0].Price.Currency.index:
"google.org/x/text/currency".Unit
consider using cmpopts.EquateComparable
to compare comparable Go types
```



# Service layer

pure unit tests (if possible)

mock dependencies:

repository and other ports

# Mocks in Go

***vektra/mockery***

*uber-go/mock*

*matryer/moq*

*golang/mock\**

manual mocks

# Mockery

mock generation from interfaces

*testify/mock* conventions, simple API

the most popular

# Generate mocks

```
//go:generate mockery --name=CartRepo --output=. --outpkg=repo
//      --structname=MockCartRepo --filename=cart_repo_mock.go
type CartRepo interface {
    GetCart(ctx, ownerID string) (Cart, error)
    AddItem(ctx, ownerID string, item CartItem) error
}
```

```
> go install github.com/vektra/mockery/v2@latest
> go generate ./...
```

# Configure mocks

```
tests := []struct {  
    name      string  
    mockSetup func(repo *MockCartRepo) // new field  
}{  
    {  
        name: "happy case",  
        mockSetup: func(repo *MockCartRepo) {  
            repo.On("AddItem",  
                mock.Anything, mock.AnythingOfType("string"), cartItem).  
                Return(nil)  
        },  
    },  
}
```

# Use mocks

```
t.Run(tt.name, func(t *testing.T) {
    mockRepo := new(MockCartRepo)

    if tt.mockSetup != nil {
        tt.mockSetup(mockRepo)
    }

    cartService, err := service.NewCart(mockRepo)
    require.NoError(t, err)

    // test cartService

    mockRepo.AssertExpectations(t)
})
```

# Data for tests

***brianvoe/gofakeit***

*jaswdr/faker*

*go-testfixtures*

hardcoded

# gofakeit

```
import (  
    "github.com/brianvoe/gofakeit"  
    "github.com/google/uuid"  
    "github.com/shopspring/decimal"  
)  
  
func fakeCartItem() CartItem {  
    return CartItem{  
        ProductID: uuid.MustParse(gofakeit.UUID()),  
        Price: Money{  
            Amount:    decimal.NewFromFloat(gofakeit.Price(1, 100)),  
            Currency:  currency.MustParseISO(gofakeit.CurrencyShort()),  
        },  
    }  
}
```



# HTTP layer

mock service layer

handler, router tests with *httptest*

also *gavv/httpexpect*, *steinfletcher/apitest*

# Gin handler

```
import "github.com/gin-gonic/gin"

type CartHandler struct {
    service service.CartService
}

func (h *CartHandler) GetCart(c *gin.Context) {
    c.JSON(http.StatusOK, cartDT0)
}

t.Run(tt.name, func(t *testing.T) {
    handler, err := rest.NewCart(mockService)
    require.NoError(t, err)
```

# Invoke handler

```
recorder := httptest.NewRecorder()

ginContext, _ := gin.CreateTestContext(recorder)
ginContext.Params = gin.Params{gin.Param{
    Key: "owner_id", Value: tt.ownerID}}

req, err := http.NewRequest(http.MethodGet, "any_path", nil)
ginContext.Request = req

// func (h *cartHandler) GetCart(c *gin.Context)
handler.GetCart(c)
```

# Assert from recorder

```
handler.GetCart(c)

assert.Equal(t, tt.wantStatus, recorder.Code)

if tt.wantStatus == http.StatusOK {
    var actualCart dto.Cart
    err := json.Unmarshal(recorder.Body.Bytes(), &actualCart)
    require.NoError(t, err)

    assertEqualsCart(t, tt.wantCart, actualCart)
}
```

# Router vs handler tests

| Characteristic | Handler              | Router                  |
|----------------|----------------------|-------------------------|
| Primary Focus  | Handler logic        | Routing mechanics       |
| Key Validation | Input processing     | Route matching          |
|                | Service interactions | Parameter extraction    |
|                | Error handling       | Middleware application  |
|                | Edge cases           | Happy path              |
| Test Depth     | Deep, comprehensive  | Shallow, broad coverage |

# Router test case

```
{
  name: "GetCart",
  method: http.MethodGet,
  url: "/carts/" + owner1,
  mockFunc: func() {
    mockService.On("GetCart", mock.Anything, owner1).
      Return(cart1, nil)
  },
  statusCode: http.StatusOK,
},
```

# Router test

```
t.Run(tt.name, func(t *testing.T) {  
    w := httptest.NewRecorder()  
  
    req, err := http.NewRequest(tt.method, tt.url, tt.bodyBytes)  
    req.Header.Set("Content-Type", "application/json")  
  
    router.ServeHTTP(w, req)  
  
    assert.Equal(t, tt.statusCode, w.Code)  
}
```

# Coverage report

```
> go test ./... -v -coverprofile=coverage.out -race  
> go tool cover -html=coverage.out -o coverage.html  
> open coverage.html
```

```
- name: Upload coverage to Coveralls  
  uses: coverallsapp/github-action@v2  
  with:  
    github-token: ${ secrets.COVERALLS_TOKEN }  
    path-to-lcov: coverage.out
```



# Misc

*pact-go*: contract testing

*testifylint*: linter for testify

smoke+ testing in test clusters

# Takeaways

| Layer      | Tools  |
|------------|--|
| All        | <i>testify/assert+require, go-cmp, brianvoe/gofakeit</i> |
| Repository | <i>testcontainers-go, testify/suite, uber-go/goleak</i>  |
| Service    | <i>vektra/mockery, testify/mock</i>                      |
| HTTP       | <i>httptest+, vektra/mockery, testify/mock</i>           |

Thank you and happy testing!

[github.com/nikolayk812/go-tests](https://github.com/nikolayk812/go-tests)