INTRODUCCIÓN A LA CLASE

STORAGE IMPLEMENTATION

Objetivos de esta clase

- Conocer distintas alternativas para testear nuestro repository sql
- Aprender a usar el paquete go-sqlmock
- Aprender a utilizar el paquete go-txdb

TESTEANDO UN REPOSITORY

STORAGE IMPLEMENTATION

Como testeamos nuestro repository?

Lo que es necesario testear no es la interfaz, sino la implementación del mismo. Ya sea un repositorio de una base de datos, en memoria o incluso el file system.
Es por esto que dependiendo la tecnología que se utilice para implementar un repository se seleccionarán las herramientas que se utilizaran para el testeo del mismo.

ALTERNATIVAS TESTING

STORAGE IMPLEMENTATION

// Alternativas de Testing

Existen packages que proporcionan funcionalidades para mockear o simular la interacción con la Base de Datos

// Alternativas de Testing

Y también para interactuar con la base de datos mediante transacciones para implementar los test

Package go-sqlmock

IT BOARDING

BOOTCAMP

// Package DATA-DOG/go-sqlmock

Es una librería que implementa el paquete sql/driver con el propósito de simular un motor de base de datos en los tests sin la necesidad de una real.

Características

- Hace uso del type **sql.DB (del paquete database/sql)** para gestionar la conexión y la ejecución.
- Permite simular una base de datos sin la presencia de una realmente.
- Brinda la posibilidad de simular errores o probar casos borde.

Instalación

Para utilizarlo es necesario instalar el paquete de la siguiente manera:

```
$ go get github.com/DATA-DOG/go-sqlmock
```

Una vez instalado, para hacer uso de dicho paquete solo es necesario agregar lo siguiente:

```
{}
db, mock, err := sqlmock.New()
assert.NoError(t, err)
defer db.Close()
```

Implementación

```
db, mock, err := sqlmock.New()
assert.NoError(t, err)
defer db.Close()
```

El código que vemos es todo lo necesario para generar mocks ante interacciones con el driver de la base de datos.

El primer parámetro recibido (db) es un puntero de sql.DB, por ende lo podemos reemplazar en donde hagamos uso del mismo.

El siguiente parámetro “mock” es quien nos permite agregar mediante regex respuestas tanto para Queries como para Exec.

Los mismos utilizan expresiones regulares para realizar el matching. Por último se recibe “err” en el caso de que haya ocurrido algún problema con la inicialización de la mock db.

Implementación

Como se puede observar en el siguiente ejemplo, luego de inicializar sqlmock. Se utiliza la instancia de mock para establecer que realizar al cumplir con cada consulta sql.

Por un lado se indica que el "*INSERT INTO users*" devuelva un resultado donde el ultimo id insertado va a ser el 1 y además que hubo un row afectado.

{}

```
func Test_sqlRepository_Store_Mock(t *testing.T) {
  db, mock, err := sqlmock.New()
  assert.NoError(t, err)
  defer db.Close()

  mock.ExpectExec("INSERT INTO
users").WillReturnResult(sqlmock.NewResult(1, 1))
  ...
}
```

Implementación

Luego, observamos cómo se mockea la query sobre la tabla users.

El string dentro de ExpectedQuery es un regex que incluye cualquier cadena entre SELECT y FROM.

Para simular el resultado de una query es necesario especificar cuáles son las columnas que devuelve y con AddRow se pueden agregar registros a la respuesta.

{}

```
func Test_sqlRepository_Store_Mock(t *testing.T) {
  ...
  columns := []string{"uuid", "firstname", "lastname",
  "username", "password", "email", "ip", "macAddress",
  "website", "image"}
  rows := sqlmock.NewRows(columns)
  userId := uuid.New()
  rows.AddRow(userId, "", "", "", "", "", "", "", "", "")
  mock.ExpectQuery("SELECT .* FROM
users").WithArgs(userId).WillReturnRows(rows)
  ...
}
```

Implementación

Por último, se inicializa el repository con la instancia de la db mockeada y se pasa a hacer uso del repository con el comportamiento esperado con una base de datos real. En este caso:

Se valida que al obtener algo que no existe no devuelve resultado

Se inserta y luego se verifica que exista.

Se espera que los métodos mockeados sean llamados

{}

```
func Test_sqlRepository_Store_Mock(t *testing.T) {
  ...
  repository := NewSqlRepository(db)
  ctx := context.TODO()
  user := models.User{
     UUID: userId,
  }
  getResult, err := repository.GetOne(ctx, userId)
  assert.NoError(t, err)
  assert.Nil(t, getResult)
  err = repository.Store(ctx, &user)
  assert.NoError(t, err)
  getResult, err = repository.GetOne(ctx, userId)
  assert.NoError(t, err)
  assert.NotNil(t, getResult)
  assert.Equal(t, user.UUID, getResult.UUID)
  assert.NoError(t, mock.ExpectationsWereMet())
}
```

Implementación

Al unir las secciones analizadas anteriormente vemos como es el código completo del test, respetando las secciones de:

Arrange (inicializar lo que vamos a utilizar)

Act (hacer uso del repository)

Assert (verificar que ocurra lo esperado)

{}

```
func Test_sqlRepository_Store_Mock(t *testing.T) {
  db, mock, err := sqlmock.New()
  assert.NoError(t, err)
  defer db.Close()
  mock.ExpectPrepare("INSERT INTO users")
  mock.ExpectExec("INSERT INTO users").WillReturnResult(sqlmock.NewResult(1, 1))
  columns := []string{"uuid", "firstname", "lastname", "username", "password",
"email", "ip", "macAddress", "website", "image"}
  rows := sqlmock.NewRows(columns)
  userId := uuid.New()
  rows.AddRow(userId, "", "", "", "", "", "", "", "", "")
  mock.ExpectQuery("SELECT .* FROM users").WithArgs(userId).WillReturnRows(rows)
  repository := NewSqlRepository(db)
  ctx := context.TODO()
  user := models.User{
     UUID: userId,
  }
  getResult, err := repository.GetOne(ctx, userId)
  assert.NoError(t, err)
  assert.Nil(t, getResult)
  err = repository.Store(ctx, &user)
  assert.NoError(t, err)
  getResult, err = repository.GetOne(ctx, userId)
  assert.NoError(t, err)
  assert.NotNil(t, getResult)
  assert.Equal(t, user.UUID, getResult.UUID)
  assert.NoError(t, mock.ExpectationsWereMet())
}
```

Package go-txdb

IT BOARDING

BOOTCAMP

// Package DATA-DOG/go-txdb

Es un paquete que proporciona un driver de sql que al generar una conexión inicia una transacción y genera un rollback al realizar un cierre.

IT BOARDING

BOOTCAMP

Características

- Soporta cualquier driver del paquete database/sql.
- Permite realizar tests más robustos ya que utilizara las restricciones reales del motor de base de datos que se utilice (unique, tipos de dato, claves foráneas).
- Depende de una conexión real, solo aísla el código dentro de una transacción.

Instalación

Para utilizarlo es necesario instalar el paquete de la siguiente manera:

```
$ go get github.com/DATA-DOG/go-txdb
```

Una vez instalado, para hacer uso de dicho paquete solo es necesario agregar lo siguiente:

```{}
txdb.Register("txdb", "mysql", "root:mariadb@/mydb")
db, err := sql.Open("txdb", uuid.New().String())
```

Implementación

Luego de inicializar txdb con el driver de mysql, es posible abrir una conexión de txdb.

Esa conexión a su vez es un drive que devuelve un puntero de sql.DB el cual puede ser utilizado por nuestro repository.

{}

```go
package util

import (
    "database/sql"
    "github.com/DATA-DOG/go-txdb"
    _ "github.com/go-sql-driver/mysql"
    "github.com/google/uuid"
)

func init() {
    txdb.Register("txdb", "mysql", "root:mariadb@/mydb")
}

func InitDb() (*sql.DB, error) {
    db, err := sql.Open("txdb", uuid.New().String())

    if err == nil {
        return db, db.Ping()
    }

    return db, err
}
```

Implementación

Luego de inicializar la base de datos.

Se utiliza la instancia de db para inicializar el repo al igual que lo haríamos con un driver real como el de mysql.

Luego se procede a hacer uso de dicho repo tal como lo utilizaremos en código productivo.

{}

```
func Test_sqlRepository_Store(t *testing.T) {
   db, err := util.InitDb()
   assert.NoError(t, err)
   repository := NewSqlRepository(db)
   ctx := context.TODO()
   userId := uuid.New()
   user := models.User{
       UUID: userId,
   }
   err = repository.Store(ctx, &user)
   assert.NoError(t, err)
   getResult, err := repository.GetOne(ctx, uuid.New())
   assert.NoError(t, err)
   assert.Nil(t, getResult)
   getResult, err = repository.GetOne(ctx, userId)
   assert.NoError(t, err)
   assert.NotNil(t, getResult)
   assert.Equal(t, user.UUID, getResult.UUID)
}
```

Gracias.

IT BOARDING

BOOTCAMP

Autor: Alejandro Herrera
Email: alejandro.herrera@digitalhouse.com
Última fecha de actualización: 25-07-21

IT BOARDING
BOOTCAMP