Testing

Testing

O teste é o processo de avaliar e verificar se um aplicativo de software faz o que se espera que faça.
Os benefícios dos testes incluem a prevenção de erros, a redução dos custos de desenvolvimento e a melhoria do desempenho.

Manual vs Automated testing

- Às testes manuais são realizados por pessoas, que navegam e interagem com o software (usando ferramentas apropriadas para cada caso).

- Por outro lado, os testes automatizados são realizados por máquinas, que executam um "test script" que já foi escrito anteriormente. A qualidade desses testes automatizados depende de quão bem escritos estão os "test scripts".

Tipos de test automatizados

- Unit tests
- Integration tests
- Functional tests
- End-to-end tests
- Performance testing

Testing em GO

Testing em GO

Uma função de teste em Go inclui esta assinatura:

func Testxxxx(t *testing.T).

- Todas as funções de teste devem começar com a palavra **Test**.
- Elas recebem apenas um parâmetro que é um ponteiro do tipo **testing.T**. Esse tipo contém métodos úteis que serão necessários para gerar resultados e registrar erros na saída, como o método **t.Errorf()** ou **t.Fail()**.
- Os testes são salvos em arquivos utilizando a seguinte convenção de nomenclatura: **filename_test.go**. Por exemplo **wallet_test.go**.

Executar testes

Comandos:

- Executar testes do pacote: **go test**
- Executar todos os testes: **go test ./...**
- Executar todos os testes com mais informações de saída: **go test ./... -v**
- Executar testes e verificar a cobertura: **go test -cover ./...**

Table-Driven Tests en Go

Table-Driven Tests en Go

En muitas situações, nos deparamos com o fato de que um único teste não abrange todos os casos possíveis. Nestas circunstâncias, temos duas alternativas:

- Escrever um teste para cada caso (Subtest).
- Escrever um único teste que receba como parâmetro todos os possíveis casos.

Testify

https://github.com/stretchr/testify

Ele fornece muitas ferramentas para garantir que seu código se comporte conforme pretendido.

Mocks

Un mock es un objeto simulado que se utiliza para **simular el comportamiento** de un objeto real en un entorno de prueba.

Los mocks se utilizan para aislar el código que se está probando de las dependencias externas, como bases de datos, servicios web u otros sistemas que puedan ser difíciles o costosos de configurar o interactuar con ellos en un entorno de prueba

https://github.com/golang/mock

Ginkgo

Behavior-driven Development (BDD)

https://github.com/onsi/ginkgo

Que es BDD

El desarrollo basado en el comportamiento (BDD) es una metodología ágil de desarrollo de software en la que una aplicación se documenta y diseña en torno al comportamiento que un usuario espera experimentar al interactuar con ella.

Fomenta la colaboración entre desarrolladores, expertos en control de calidad y representantes de clientes en un proyecto de software.

Alienta a los equipos a usar conversaciones y ejemplos concretos para formalizar una comprensión compartida de cómo debe comportarse la aplicación.

DSL

BDD se facilita en gran medida mediante el uso de un lenguaje específico de dominio simple (DSL) que utiliza construcciones de lenguaje natural que pueden expresar el comportamiento y los resultados esperados.

Title: Returns and exchanges go to inventory.

As a store owner,
I want to add items back to inventory when they are returned or exchanged,
so that I can track inventory.

Scenario 1: Items returned for refund should be added to inventory.
Given that a customer previously bought a black sweater from me
and I have three black sweaters in inventory,
when they return the black sweater for a refund,
then I should have four black sweaters in inventory.

Scenario 2: Exchanged items should be returned to inventory.
Given that a customer previously bought a blue garment from me
and I have two blue garments in inventory
and three black garments in inventory,
when they exchange the blue garment for a black garment,
then I should have three blue garments in inventory
and two black garments in inventory.

Ginkgo

Ginkgo es un framework de testing para Go diseñado para escribir pruebas expresivas. Se combina con Gomega, que es una librería de aserciones. Cuando se combinan, Ginkgo y Gomega proporcionan un DSL (lenguaje específico del dominio) rico y expresivo para las pruebas de escritura.

DSL en Ginkgo

Describe: bloque que define lo que estás probando.

Context: bloque que define el momento de la prueba (por ej, "cuando se recibe una solicitud no válida", "cuando el servicio no está disponible").

It: bloque que ejecuta el código para probar e indicar lo que espera que suceda.

BeforeEach: se ejecutan antes de cada prueba unitaria (It bloques).

AfterEach: se ejecutan después de cada prueba unitaria.

Terminología

- Spec Subjects: ***It / Specify***
- Setup: ***BeforeEach***
- Container nodes: ***Describe / Context / When***
- Setup: ***BeforeEach***

Ejemplo

- Generar suite de tests: ***ginkgo bootstrap***
- Generar archivo de tests: **ginkgo generate <package-name>**