TESZTAUTOMATIZÁLÁS

7. forduló

<epam>

A kategória támogatója: EPAM

RENDELKEZÉSRE ÁLLÓ IDŐ:

45:00

Ismertető a feladathoz

Felhasznált idő: 01:33/45:00 **Elért pontszám:** 0/46

1. feladat 0/12 pont

Tegyük fel, hogy van egy test.spec.ts file-unk, ami TypeScript unit teszteket tartalmaz, és Mocha-t használ tesztek futtatásához. Ha lefuttatjuk a test.spec.ts-ben lévő tesztet ts-mocha segítségével és az alap (spec) reporterrel, milyen végeredményt fogunk kapni a konzolban. Test.spec.ts file tartalma:

```
let assert = require('assert');

describe('Array', function() {
 before(() => {
  console.log(`Executing test: ${this.title}`)
 })

 describe('#indexOf()', function() {
   before(() => {
    console.log(`Executing test: ${this.title}`)
   })

   it('should probably return the index of the value', function() {
    console.log(`Executing test: ${this.test?.fullTitle()}`);
    assert.equal([1, 2, 3].indexOf(4), -1);
  });
```

```
  });
});
```

Válasz

```
$ ts-mocha *.spec.ts


 Array
Executing test: Array
  #indexOf()
Executing test: #indexOf()
Executing test: Array #indexOf() should probably return the index of the value
   ✓ should probably return the index of the value


 1 passing
```



```
$ ts-mocha *.spec.ts


 Array
Executing test: Array
  #indexOf()
Executing test: #indexOf()
Executing test: Array #indexOf() should probably return the index of the value
   1) should probably return the index of the value


 0 passing (12ms)
 1 failing

 1) Array
    #indexOf()
     should probably return the index of the value:

   AssertionError [ERR_ASSERTION]: 2 == -1
   + expected - actual

   -2
   +-1
```



```
$ ts-mocha *.spec.ts


 Array
Executing test: Array
  #indexOf()
Executing test: #indexOf()
```

```
    Executing test: undefined
      ✓ should probably return the index of the value


     1 passing
```



```
    $ ts-mocha *.spec.ts

     1 passing
```

Magyarázat

Az első before hookban kiíratjuk a jelenlegi suite nevét, ami 'Array', innen jön az "Executing test: Array" rész.

Ugyanez igaz a második describe-ban lévő before hookra, csak ott a jelenlegi title "#indexOf()".

Az it-ben kiiratjuk a teljes címet, az azt jelenti, hogy az összes eddigi describe szövegét hozzáadjuk a jelenlegi it-hez, igy kiírjuk, hogy 'Executing test: Array #indexOf() should probably return the index of the value'.

Maga a mocha is kiírja a jelenleg futó describe/it-eket. Innen jönnek azok a stringek, amelyek nem "Executing"-al kezdődnek.

Maga a teszt pedig tökéletesen lefut, mivel az indexOf függvény -1-et ad vissza ha nem találja a megadott argumentum értékét a tömbben.

Így áll össze a helyes megoldás.

2. feladat 0/12 pont

Adott az alábbi Python kód:

```python
import pytest


class Element:
  def __init__(self, name):
    self.name = name

@pytest.fixture
def my_element():
  return Element("div")


@pytest.fixture
def Elements(my_element):
  return [Element("li"), my_element]
```

```
def test_my_element_in_elements():
  #?
```

Hogyan kell kinéznie a 'test_my_element_in_elements' függvénynek, hogy azt ellenőrizze, hogy a 'my_element' szerepel-e az 'Elements'-ben?

Válasz

```
def test_my_element_in_elements(Elements):
  assert my_element in Elements
```

```
def test_my_element_in_elements(my_element, Elements):
  assert my_element in Elements
```

```
def test_my_element_in_elements(my_element, Elements):
  assert my_element present in Elements
```

```
def test_my_element_in_elements():
  assert my_element in Elements
```

Magyarázat

@pytest.fixture használata során a két, @pytest.fixture tag-gel ellátott függvény nevét át kell adnunk a 'test_my_element_in_elements' paraméterlistájának, így 'my_element()' fog először lefutni, ami visszatér egy 'Element' objektummal, majd az 'Elements' függvény létrehoz egy listát egy 'li' objektummal és az előzőleg létrehozott 'div' objektummal. Ezután az 'assert my_element in Element'-tel verifikálhatjuk, hogy a 'my_element' szerepel az 'Elements' listában.

3. feladat 0/12 pont

Tegyük fel, hogy van egy Java metódusunk, amely 2 egész számot ad össze és visszaadja az eredményt. Tekintsük az alábbi kódrészletet.

```
public class MyClass {
  public int addNumbers(int number1, int number2){
    return number1 + number2;
  }
}
```

Tekintsük ehhez a metódushoz az alábbi kódrészletben TestNG és Hamcrest matcher használatával írt **HIBÁS** tesztet (a tesztesetek nem teljesek, csak példaértékűek a kódhoz):

```
public class MyTestClass extends AbstractTestNGSpringContextTests {
  @TestNGTest(testDataProvider())
  public void addNumbersShouldBeSuccesfulOnValidInput(String input1, String input2) {
     // Given
     // When
     int result = addNumbers(input1, input2);
     // Then
     assertThat(result, equalTo(expectedResult));
  }

  @TestDataProvider
  private Object[][] testDataProvider() {
    return new Object[][]{
      { 1, 1.5, 2.5},
      { 1, 1, "2" }
    };
  }
}
```

Válaszd ki azt a kódot amely kijavítja az összes szintaktikai és logikai hibát a tesztben!

Válasz

```
@Test(testDataProvider())
 public void addNumbersShouldBeSuccesfulOnValidInput(int input1, int input2) {
 // Given
 // When
   int result = addNumbers(input1, input2);
 // Then
   assertThat(result, equalTo(expectedResult));
}

@DataProvider
 private Object[][] testDataProvider() {
    return new Object[][]{
      1, 1.5, 2.5,
      1, 1, "2"
    };
  }
```

```
@TestTNGTest(dataProvider = testDataProvider())
 public void addNumbersShouldBeSuccesfulOnValidInput(int input1, int input2, int expectedResult) {
 // Given
 // When
   int result = addNumbers(input1, input2);
 // Then
   assertThat(result, equalTo(expectedResult));
}
```

```
@TestDataProvider
 private Object[][] testDataProvider() {
    return new Object[][]{
      1, 2, 3,
      1, 1, 2
    };
  }
```

```
@Test(dataProvider = "testDataProvider")
 public void addNumbersShouldBeSuccesfulOnValidInput(int input1, int input2, int expectedResult)
 // Given
 // When
   int result = addNumbers(input1, input2);
 // Then
   assertThat(result, equalTo(expectedResult));
}

@DataProvider
 private Object[][] testDataProvider() {
    return new Object[][]{
       { 1, 2, 3 },
       { 1, 1, 2 }
    };
  }
```

```
@Test(dataProvider = "testDataProvider")
 public void addNumbersShouldBeSuccesfulOnValidInput(int input1, int input2, int expectedResult)
 // Given
 // When
   int result = addNumbers(input1, input2);
 // Then
   assertThat(result, equalTo(expectedResult));
}

@TestDataProvider
 private Object[][] testDataProvider() {
    return new Object[][]{
       { 1, 2, 3 },
       { 1, 1, 2 }
    };
  }
```

Magyarázat

TestNG teszteknél a @Test a megfelelő annotáció

A TestNG dataprovider helyes megadása: @Test(dataProvider = "a data provider metódus neve zárójelek nélkül")

A teszt metódusa 3 db input értékkel dolgozik, amelyet a dataproviderből kap meg. Ki kell egészítenünk a teszt metódus paramétereid a 3. expectedResult-tal, amely máshol nincs is deklarálva a metóduson belül.

A tesztelt kódrészlet integer inputokat vár, így a teszt metódus inputjait cseréljük le String-ről int-re

TestNG dataproviderek helyes annotációja: @DataProvider

A dataprovider metódusban {}-ek között kell megadnunk az 1 tesztesethez tartozó inputkombinációkat

Mivel a teszt metódus integer inputokat vár, a dataproviderben nem adhatunk meg törteket és stringeket

4. feladat 0/10 pont

Mik a folyamatos integráció/folyamatos szállítás (Continous Integration/Continous Delivery) előnyei az alábbiak közül?

Válaszok

- [x] Csökkenti a hibás szoftver éles környezetbe jutásának esélyét az automatizált teszteknek köszönhetően
- [ ] Automatikus canary telepítést tartalmaz
- [x] A javításhoz szükséges átlagosan eltelt idő (Mean Time To Repair [MTTR]) lerövidül a kisebb kódváltozások és gyorsabb probléma izolálció miatt
- [ ] Szignifikánsan csökkenti a projekt költségét, mivel csökkenti a szükséges környezetek (environments) számát

Magyarázat

A második válasz a folyamatos telepítés (continous deployment) része.

Az utolsó válasz esetében pedig sok szemponttól függ, hogy a CI/CD implementálása csökkenti-e a projekt költségeket, azonban a használt környezetek számát nem csökkenti.