

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM**  
**MAROSVÁSÁRHELYI KAR**

# **Szoftver Tesztelés**

## **Dokumentáció**

**Hallgató neve: Ráduly Botond**

**Szak: Automatika és alkalmazott informatika IV év**

**2024**

# 1. A kiválasztott unit test csomag leírása

## 1.1 Motiváció

A projekt kitűzött feladatainak megoldása során eltértem az ajánlott pytest használatától és ezért én a unittest-et használtam. Tudatában voltam annak hogy bár a unittest egy erős eszköz a teszteléshez, vannak olyan harmadik féltől származó tesztelő keretrendszerek is, mint például a pytest és a nose2, amelyek más filozófiát követnek, és bizonyos esetekben kényelmesebbek lehetnek, például rövidebb és olvashatóbb kóddal, de ennek ellenére én mégis a unittestet használtam mert a projekt során könnyebben találtam a használatára vonatkozó segédanyagokat. Továbbá már egy régebbi alkalommal használtam a unittest-et és azt gondoltam hogy így ez számomra előnyösebb lesz. Szerintem az volt bár ezt nem tudom pontosan megítélni mivel csak a unittestet ismerem átfogóbban. Természetesen a unittestnek vannak előnyei is, mint például:

- Beépített modul: A unittest beépített a Pythonban, így nincs szükség harmadik féltől származó könyvtárak telepítésére. Ez könnyebbé teszi a projektjeid függőségeinek kezelését.
- Széles körű támogatás és dokumentáció: A unittest széles körben használt és támogatott a Python közösségben. Ennek eredményeként gazdag dokumentáció és erőforrások állnak rendelkezésre annak megértéséhez és hatékony használatához
- Tesztosztályok és eszközök: A unittest segítségével könnyen létrehozhat sz tesztosztályokat, és különböző teszteszközöket használhatsz, például az assert metódusokat, a kivételkezelést, és egyéb funkciókat.
- Tesztek függetlensége: A unittest segít az egyes tesztek függetlenségének megőrzésében, ami azt jelenti, hogy a tesztek egymástól elszigeteltek és nem befolyásolják egymást.

Ezen fenti indokok összessége miatt használtam a projekt során a unittestet.

## 1.2 Telepítés

A unittest használatának nagy előnye a pytesttel szemben hogy amíg a pytestet telepíteni kell addig, lévén hogy a unittest egy beépített tesztelőmodul a pythonban. Ennek következtében a unittest használatához semminemű telepítés nem szükséges csak egyszerűen be importoljuk a „py” fileba és már használhatjuk is minden előnyével együtt.

## 2. A unit test repository leírása

A projekten belül a teszt állományok nincsenek külön mappákba szervezve, azonban minden tesztescsoport külön „.py” állományba van szervezve. Ez azt jelenti hogy a relation managerrel kapcsolatos tesztek a „Relation\_Manager\_Test.py” python állományba vannak szervezve míg az employee managerrel kapcsolatos tesztek az „Employee\_Manager\_Test.py” állományba vannak szervezve.

Ezekben az állományokban az egyes tesztek jól felismerhető tesztnev konvenció alapján vannak rendezve. A tesztnev konvenció második része utal a tesztet elváró feladat részleteire amelyek a feladat felhívószövegéből származnak. Az tesztnev első része pedig maga a „test” szó ami egyértelműen utal arra hogy ez egy teszt függvény. A jobb megértés kedvéért példaként ide idézem a relation managerrel kapcsolatos feladatscsoport első feladatára megírt unittest elnevezését a python állományból: „test\_team\_leader\_john\_doe”.

Továbbá a tesztelendő egységek rövid leírására megemlíthetjük azt, hogy a TestEmployeeManager osztályban egységtesztek vannak írva az EmployeeManager osztály metódusaira, a TestRelationsManager osztályban egységtesztek vannak írva a RelationsManager osztály metódusaira. Mindkét teszt esetben a setUp metódusban inicializálják a szükséges objektumokat, például RelationsManager és Employee példányokat.

Mindezen eddig felsoroltak után még meg kell említeni a unit test repository leírásának részeként azt is hogy hogyan lehet ezeket futtatni vagyis a futtatási utasításokat. Az általam írt tesztek esetében mindkét teszt esetnél a \_\_main\_\_ blokk alatt található a unittest.main(), ami a teszteseteket futtatja, ha a fájlt közvetlenül futtatják. A teszteset ilyen módú futtatása rendkívül egyszerű a unittest.main() meghívásával futtathatók a tesztesetek, például a „python your\_test\_file.py” parancs segítségével.

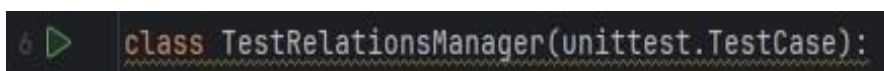
Végül még meg kell említenem a unit test repository leírásához egy elengedhetetlennek tűnő részletet ami nem más mint a függőségek és a környezet. Jelen projekt esetében mindkét teszt eset függ a unittest modultól, amely a Python beépített egységtesztelő keretrendszer. Emellett a TestEmployeeManager osztályban a @patch dekorátorral teszten belül "meghamisítják" a calculate\_salary\_and\_send\_email metódust, hogy ellenőrizzék a hívásokat. A felsoroltakon kívül a tesztesetek más modulokat is importálnak, mint például az employee, employee\_manager, relations\_manager.

A fent említett alponatok mindegyikét figyelembe kell vennünk ahhoz hogy pontos képet kaphassunk a unit test repository kinézetéről, struktúrájáról és leírásáról.

### 3. A fejlesztett unit test-ek részletes leírása

#### 3.1 „Relation\_Manager\_Test.py” leírása

A „Relation\_Manager\_Test.py” állományban szereplő tesztek leírását megelőzően szükségét érzem röviden összefoglalni hogy hogyan is van megírva az állomány teljes kódja. Ez a kód elsősorban egy unittest teszteseteket tartalmazó osztályt definiál (lásd 3.1.1 Ábra), amelyek a RelationsManager osztály működését tesztelik.



```
6 ▶ class TestRelationsManager(unittest.TestCase):
```

3.1.1 Ábra Osztály definiálás

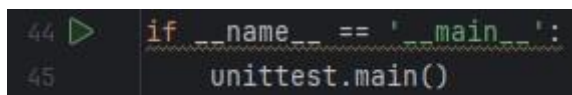
Ezen tesztek során (lásd 3.1.2 Ábra) az osztály különböző tesztekkel ellenőrzi, hogy a RelationsManager helyesen kezeli az alkalmazottak és a csapatvezetők közötti kapcsolatokat.



```
7 ▶ def setUp(self):...  
9  
10 ▶ def test_team_leader_john_doe(self):...  
13  
14 ▶ def test_team_members_of_john_doe(self):...  
19  
20 ▶ def test_tomas_andre_not_team_member_of_john_doe(self):...  
24  
25 ▶ def test_gretchen_walford_base_salary(self):...  
28  
29 ▶ def test_tomas_andre_not_team_leader(self):...  
32  
33 ▶ def test_retrieve_team_members_of_tomas_andre(self):...  
38  
39 ▶ def test_jude_overcash_not_in_database(self):...  
43
```

3.1.2 Ábra Egységtesztek listája

Ezután a kód a unittest.main() blokkal (lásd 3.1.3 Ábra) az összes tesztet futtatja, ha a fájlt közvetlenül futtatják.



```
44 ▶ if __name__ == '__main__':  
45     unittest.main()
```

3.1.3 Ábra unittest.main() blokk

Miután átfogó képet kaptunk arról hogy hogyan is néz ki a relation manager teszt állomány, most szükséges felsorolni azt hogy mit is csinálnak a unit tesztek egyesével. Vegyük szépen sorban a teszteseteket egyesével.

Az első kód egy setUp metódust definiál (lásd 3.1.4 Ábra) egy unittest teszteset osztályban. A setUp metódus a unittest.TestCase osztály része, és minden egyes teszt előtt lefut.

```

7  @
8  def setUp(self):
    self.manager = RelationsManager()

```

### 3.1.4 Ábra setUp

A setUp metódus célja az, hogy előkészítse a tesztek környezetét, például inicializálja a szükséges objektumokat, amelyekre a tesztek hivatkozni fognak. Ez azt jelenti, hogy minden egyes teszt előtt létrehoz egy új RelationsManager példányt és azt az osztályváltozóba (self.manager) menti. Ezáltal minden egyes tesztnek elérhető lesz egy friss RelationsManager példány, amellyel dolgozhatnak, és a tesztek nem befolyásolják egymást a futás során. A setUp metódus ezt a friss példányt biztosítja minden teszt számára, amely a teszteset osztály része.

A következő kód vagyis az első teszt (lásd 3.1.5 Ábra) az első feladat részeként azt vizsgálja, hogy a RelationsManager helyesen tárolja-e John Doe születési dátumát, és ha nem, akkor hibaüzenettel jelzi a problémát. A self.assertEqual azt várja, hogy a kiválasztott alkalmazott születési dátuma megegyezzen a megadott értékkel, különben hibaüzenetet dob.

```

10  def test_team_leader_john_doe(self):
11      john_doe = [employee for employee in self.manager.get_all_employees()
12                  if employee.first_name == "John" and employee.last_name == "Doe"][0]
13      self.assertEqual(john_doe.birth_date, date(year=1970, month=1, day=31))

```

### 3.1.5 Ábra

Az első tesztsor második tesztje, egyben a második feladat megoldása (lásd 3.1.6 Ábra) arra szolgál, hogy bizonyítsa, hogy a RelationsManager helyesen kezeli John Doe csapatát, vagyis hogy a csapat tagjai azok az alkalmazottak, akiket vártunk, vagyis az elvárt tagok listájával egyezik meg.

```

15  def test_team_members_of_john_doe(self):
16      john_doe = [employee for employee in self.manager.get_all_employees()
17                  if employee.first_name == "John" and employee.last_name == "Doe"][0]
18      team_members = self.manager.get_team_members(john_doe)
19      expected_members = [2, 3]
20      self.assertEqual(team_members, expected_members)

```

### 3.1.6 Ábra

A következő teszt (lásd 3.1.7 Ábra) azt ellenőrzi, hogy a RelationsManager osztály helyesen kezeli-e azt a helyzetet, amikor Tomas Andre-t nem sorolják be John Doe csapatának tagjaként. Ez a teszt arra szolgál, hogy meggyőződjön róla, hogy a RelationsManager helyesen kezeli azt az esetet, amikor egy adott alkalmazottat nem sorolnak be a megadott csapatba mint az adott csapat tagja, és a teszt sikeresnek tekinti, ha Tomas Andre nem szerepel a csapatban.

```

22  def test_tomas_andre_not_team_member_of_john_doe(self):
23      john_doe = [employee for employee in self.manager.get_all_employees()
24                  if employee.first_name == "John" and employee.last_name == "Doe"][0]
25      team_members = self.manager.get_team_members(john_doe)
26      self.assertNotIn(member=5, team_members) # Tomas Andre's ID is 5

```

### 3.1.7 Ábra

Az ezt követő unit test (lásd 3.1.8 Ábra) teszt azt ellenőrzi, hogy a RelationsManager osztály helyesen tárolja-e Gretchen Walford alkalmazott alapbérét. A teszt először kiválasztja a "Gretchen Watford" nevű alkalmazottat az összes alkalmazott közül, majd ellenőrzi, hogy az adott alkalmazott alapbére megegyezik-e az elvárt értékkel (4000). A teszt sikeresnek minősül, ha az elvárt és tényleges alapbér értéke megegyezik, különben hibaüzenetet generál.

```
28 > def test_gretchen_walford_base_salary(self):
29     gretchen_walford = [employee for employee in self.manager.get_all_employees()
30                          if employee.first_name == "Gretchen" and employee.last_name == "Watford"][0]
31     self.assertEqual(gretchen_walford.base_salary, second: 4000)
```

3.1.8 Ábra

A következő unit test (lásd 3.1.9 Ábra) azt ellenőrzi, hogy a RelationsManager osztály helyesen kezeli-e azt az esetet, amikor Tomas Andre nem szerepel csapatvezetőként. A teszt kiválasztja a "Tomas Andre" nevű alkalmazottat, majd ellenőrzi, hogy a RelationsManager osztály is\_leader metódusa hamis értékkel tér-e vissza az adott alkalmazottra vonatkozóan. A teszt sikeresnek tekinti magát, ha a vizsgált alkalmazott nem csapatvezető.

```
33 > def test_tomas_andre_not_team_leader(self):
34     tomas_andre = [employee for employee in self.manager.get_all_employees()
35                   if employee.first_name == "Tomas" and employee.last_name == "Andre"][0]
36     self.assertFalse(self.manager.is_leader(tomas_andre))
```

3.1.9 Ábra

A következő teszt (lásd 3.1.10 Ábra) azt ellenőrzi, hogy a RelationsManager osztály helyesen reagál-e arra, amikor megpróbálják lekérdezni Tomas Andre csapatának tagjait, de ő maga nem számít csapatvezetőnek. A teszt kiválasztja a "Tomas Andre" nevű alkalmazottat, majd ellenőrzi, hogy a RelationsManager osztály get\_team\_members metódusa egy KeyError kivételt vált-e ki. A teszt sikeresnek tekinti magát, ha a kivétel megfelelően keletkezik, jelezve, hogy Tomas Andre nem rendelkezik saját csapattagokkal.

```
38 > def test_retrieve_team_members_of_tomas_andre(self):
39     tomas_andre = [employee for employee in self.manager.get_all_employees() if
40                   employee.first_name == "Tomas" and employee.last_name == "Andre"][0]
41     with self.assertRaises(KeyError):
42         self.manager.get_team_members(tomas_andre)
```

3.1.10 Ábra

Az első tesztállomány utolsó tesztje (lásd 3.1.11 Ábra) ellenőrzi, hogy a RelationsManager osztály helyesen kezeli azt az esetet, amikor egy olyan alkalmazottat próbálnak lekérdezni, aki nincs az adatbázisban. A teszt létrehoz egy új alkalmazottat (Jude Overcash néven), majd ellenőrzi, hogy ez az alkalmazott ne szerepeljen az összes alkalmazottat visszaadó get\_all\_employees eredményében. A teszt sikeresnek tekinti magát, ha az adott alkalmazott nincs jelen az adatbázisban.

```

44 ▶ def test_jude_overcash_not_in_database(self):
45     jude_overcash = Employee(id=7, first_name="Jude", last_name="Overcash", base_salary=2000,
46                             birth_date=date(year=1985, month=1, day=1), hire_date=date(year=2010, month=1, day=1))
47     self.assertNotIn(jude_overcash, self.manager.get_all_employees())

```

3.1.11 Ábra

## 3.2 „Employee\_Manager\_Test.py” leírása

A „Employee\_Manager\_Test.py” állományban szereplő tesztek leírását megelőzően szükségét érzem röviden összefoglalni hogy hogyan is van megírva az állomány teljes kódja. Ez a kód elsősorban egy unittest teszteseteket tartalmazó osztályt definiál (lásd 3.2.1 Ábra), amelyek az EmployeeManager osztály működését tesztelik.

```

▶ class TestEmployeeManager(unittest.TestCase):

```

3.2.1 Ábra Osztály definiálás

Az ebben az osztályban létrehozott tesztek (lásd 3.2.2 Ábra) tehát ellenőrzik, hogy a EmployeeManager osztály helyesen kezeli a fizetés kiszámítását mind vezető, mind nem vezető alkalmazottak esetén, és megfelelően interakcióba lép-e a calculate\_salary\_and\_send\_email metódussal. A tesztek általánosan hozzájárulnak a kód stabilitásához és az elvárt funkcionalitás fenntartásához.

```

18 ▶ def setUp(self):...
32
    ▶ RBotond1
33 ▶ def test_calculate_salary_non_leader(self):...
36
    ▶ RBotond1
37 ▶ def test_calculate_salary_team_leader(self):...
40
    ▶ RBotond1
41 @patch('employee_manager.EmployeeManager.calculate_salary_and_send_email')
42 ▶ def test_email_notification(self, mock_calculate_salary_and_send_email):...

```

3.2.2 Ábra

Ezután a kód a unittest.main() blokkal (lásd 3.2.3 Ábra) az összes tesztet futtatja, ha a fájlt közvetlenül futtatják.

```

47 ▶ if __name__ == '__main__':
48     unittest.main()

```

3.2.3 Ábra

Miután átfogó képet kaptunk arról hogy hogyan is néz ki a relation manager teszt állomány, most szükséges felsorolni azt hogy mit is csinálnak a unit tesztek egyesével. Vegyük szépen sorban a teszteseteket egyesével.

Az első kódrészlet a setUp metódust tartalmazza (lásd 3.2.4 Ábra), amely egy unittest.TestCase alaposztályból származó osztályban található. A setUp metódus azt a célt szolgálja, hogy előkészítse azokat az objektumokat és állapotokat, amelyekre a tesztesetek futtatásakor szükség van.



```

10 def setUp(self):
11     self.rm = RelationsManager()
12
13     self.e1 = Employee(id=1, first_name="John", last_name="Doe", base_salary=3000,
14                        birth_date=date(year=1970, month=1, day=31), hire_date=date(year=1990, month=10, day=1))
15
16     self.e2 = Employee(id=2, first_name="Myrta", last_name="Tonkelson", base_salary=1000,
17                        birth_date=date(year=1980, month=1, day=1), hire_date=date(year=2000, month=1, day=1))
18
19     self.e3 = Employee(id=3, first_name="Jettie", last_name="Lynch", base_salary=1500,
20                        birth_date=date(year=1987, month=1, day=1), hire_date=date(year=2015, month=1, day=1))
21
22     self.e4 = Employee(id=4, first_name="Gretchen", last_name="Watford", base_salary=4000,
23                        birth_date=date(year=1960, month=1, day=1), hire_date=date(year=1990, month=1, day=1))
24
25     self.e5 = Employee(id=5, first_name="Tomas", last_name="Andre", base_salary=1600,
26                        birth_date=date(year=1995, month=1, day=1), hire_date=date(year=2015, month=1, day=1))
27
28     self.e6 = Employee(id=6, first_name="Scotty", last_name="Bomba", base_salary=1000,
29                        birth_date=date(year=1977, month=1, day=1), hire_date=date(year=1998, month=10, day=10))
30
31     self.em = EmployeeManager(self.rm)

```

3.2.4 Ábra

Ez a setUp metódus biztosítja, hogy minden teszt futása előtt előkészítésre kerüljenek azok az objektumok és állapotok, amelyekre a tesztek szükségleteihez szükség lehet. Ezzel javítja a tesztek tisztaságát, és megakadályozza az ismétlődő kódokat, mivel a szükséges inicializációk csak egyszer kerülnek definiálásra.

Ezek után az első teszt kód egy egységtesztet hajt végre (lásd 3.2.5 Ábra) a EmployeeManager osztály calculate\_salary metódusára a nem vezető alkalmazott (self.e6) esetén.

```

33 def test_calculate_salary_non_leader(self):
34     salary = self.em.calculate_salary(self.e6)
35     self.assertEqual(salary, second=3600)

```

3.2.5 Ábra

Ez a teszt azt ellenőrzi, hogy a calculate\_salary metódus helyesen számolja ki a fizetést a self.e6 alkalmazott esetén, és az elvárt eredmény 3600. Ha a teszt sikeresen lefut, az azt jelzi, hogy a nem vezető alkalmazott fizetési számításai megfelelnek az elvárásoknak.

A következő egységteszt kódrészlet (lásd 3.2.6 Ábra) egy másik egységtesztet hajt végre a EmployeeManager osztály calculate\_salary metódusára, ezúttal a vezető alkalmazott (self.e1) esetén.

```

37 def test_calculate_salary_team_leader(self):
38     salary = self.em.calculate_salary(self.e1)
39     self.assertEqual(salary, second=4200)

```

3.2.6 Ábra

Ez a teszt azt ellenőrzi, hogy a **calculate\_salary** metódus helyesen számolja ki a fizetést a vezető alkalmazott (self.e1) esetén, és az elvárt eredmény 4200. Ha a teszt sikeresen lefut, az azt jelzi, hogy a vezető alkalmazott fizetési számításai megfelelnek az elvárásoknak.



A következő és egyben az `Employee_Manager_Test` állomány utolsó unit test kódja (lásd 3.2.7 Ábra) egy egységtesztet hajt végre a **EmployeeManager** osztály **calculate\_salary\_and\_send\_email** metódusára, különös figyelemmel a levelezési értesítésre. A **@patch** dekorátor segítségével egy mock objektummal helyettesíti a valódi **calculate\_salary\_and\_send\_email** metódust, így elkerülve a valós e-mail küldést.

```
41 @patch('employee_manager.EmployeeManager.calculate_salary_and_send_email')
42 def test_email_notification(self, mock_calculate_salary_and_send_email):
43     self.em.calculate_salary_and_send_email(self.e1)
44     mock_calculate_salary_and_send_email.assert_called_once_with(self.e1)
```

3.2.7 Ábra

Ez a teszt azt ellenőrzi, hogy a `calculate_salary_and_send_email` metódus helyesen hívja-e meg a valódi metódust a megfelelő paraméterekkel (`self.e1`), anélkül hogy valódi e-mail értesítéseket küldene. A mock objektum segítségével ellenőrizhető, hogy a metódus hívása megfelelő volt-e a tervezett módon.

## 4. Összegzés

Összességében a unittest keretrendszer választása a projekt során számos előnyt kínált. A beépített modul, a széles körű támogatás és dokumentáció, valamint a tesztosztályok és eszközök használata révén hatékony és strukturált tesztelési folyamatot biztosított. A tesztek függetlenségének megőrzése és a könnyen hozzáférhető erőforrások miatt a unittest keretrendszer optimális választásnak bizonyult a projekt céljaira.