Szoftvertesztelés Projekt Dokumentáció

Készítette:

Magyari Csaba
Számítástechnika IV.A

1. Bevezetés

Ez a dokumentum részletes leírást nyújt az alkalmazotti menedzsment szoftverrendszer egységtesztelési implementációjáról. A projekt célja a szoftvertesztelési koncepciókkal való megismerkedés volt, elsősorban a kis kódrészletek (egységek) tesztelésére összpontosítva, hogy biztosítsuk a megfelelő működésüket.

2. Választott Tesztelési Keretrendszer: pytest

2.1 Motivációk

A pytest tesztelési keretrendszert az alábbi okok miatt választottuk:

- **Széles körű elterjedtség**: A pytest a Python 3.5+ egyik legszélesebb körben használt tesztelési keretrendszere
- **Sokoldalúság**: Támogatja az egységtesztelést, a funkcionális tesztelést és az API tesztelést is
- **Egyszerűség**: Lehetővé teszi kompakt és egyszerű tesztek készítését minimális kóddal
- **Bővíthetőség**: Nagymértékben bővíthető különböző pluginek segítségével, mint például pytest-randomly, pytest-cov, pytest-django
- **Erős közösségi támogatás**: Nagy aktív közösség, amely erőforrásokat és támogatást nyújt

2.2 Telepítés

A telepítési folyamat egyszerű volt:

Virtuális környezet létrehozása és aktiválása:

```
$ mkdir pytest_demo
$ cd pytest_demo
$ python3 -m venv pytest-env
$ source pytest-env/bin/activate
```

Pytest telepítése:

```
$ pip3 install pytest
```

Tesztek futtatása:

```
$ pytest -v
```

3. Teszt Repository Struktúra

3.1 Repository Szervezése

A repository az alábbiak szerint strukturálódik:

```
projekt-gyökér/
├── employee.py              # Employee adatosztály
├── relations_manager.py     # Alkalmazotti kapcsolatokat kezel
├── employee_manager.py      # Fizetés számításokat kezel
├── test_relations_manager.py # RelationsManager tesztek
└── test_employee_manager.py # EmployeeManager tesztek
```

3.2 Adatstruktúra

A kódbázis három fő osztályból áll:

1. **Employee**: Adatosztály, amely alkalmazotti információkat tárol
2. **RelationsManager**: Alkalmazotti kapcsolatokat és csapatstruktúrákat kezel
3. **EmployeeManager**: Alkalmazotti fizetéseket számol különböző tényezők alapján

Az egységtesztek két külön fájlban vannak szervezve, mindegyik egy-egy manager osztályt tesztel.

4. Egységteszt Leírások

4.1 RelationsManager Tesztek

4.1.1 test_john_doe_is_team_leader

Leírás: Ellenőrzi, hogy John Doe létezik-e a rendszerben csapatvezetőként, és hogy helyes-e a születési dátuma.

Bemenet: A RelationsManager példány előre definiált alkalmazotti adatokkal.

Elvárt kimenet:

- John Doe csapatvezetőként azonosítva (is_leader True értéket ad vissza)
- John Doe születési dátuma 1970. január 31.

Implementáció:

```python
def test_john_doe_is_team_leader(relations_manager):
    """Check if John Doe is a team leader and has the correct birthdate"""
    john_doe = next(e for e in relations_manager.get_all_employees() if e.first_name == "John" and e.last_name == "Doe")
    assert relations_manager.is_leader(john_doe)
    assert john_doe.birth_date == datetime.date(1970, 1, 31)
```

4.1.2 test_john_doe_team_members

Leírás: Ellenőrzi, hogy John Doe csapata pontosan Myrta Torkelsonból és Jettie Lynchből áll-e.

Bemenet: A RelationsManager példány előre definiált csapatstruktúrával.

Elvárt kimenet: John Doe csapattagjai Myrta Torkelson és Jettie Lynch.

Implementáció:

```
def test_john_doe_team_members(relations_manager):
    """Check if John Doe's team members are Myrta Torkelson and Jettie Lynch"""
    john_doe = next(e for e in relations_manager.get_all_employees() if e.id == 1)
    team_members = relations_manager.get_team_members(john_doe)
    member_names = {(e.first_name, e.last_name) for e in team_members}
    expected_members = {("Myrta", "Torkelson"), ("Jettie", "Lynch")}
    assert member_names == expected_members
```

4.1.3 test_tomas_andre_not_in_john_doe_team

Leírás: Biztosítja, hogy Tomas Andre nem tagja John Doe csapatának.

Bemenet: A RelationsManager példány John Doe csapatstruktúrájával.

Elvárt kimenet: Tomas Andre nem szerepel John Doe csapattagjai között.

Implementáció:

```
def test_tomas_andre_not_in_john_doe_team(relations_manager):
    """Make sure that Tomas Andre is not John Doe's team member"""
    john_doe = next(e for e in relations_manager.get_all_employees() if e.id == 1)
    team_members = relations_manager.get_team_members(john_doe)
    assert not any(e.first_name == "Tomas" and e.last_name == "Andre" for e in team_members)
```

4.1.4 test_gretchen_watford_salary

Leírás: Ellenőrzi, hogy Gretchen Watford alapfizetése helyesen 4000$-ra van beállítva.

Bemenet: A RelationsManager példány Gretchen Watford alkalmazotti adataival.

Elvárt kimenet: Gretchen Watford base_salary értéke 4000.

Implementáció:

```python
def test_gretchen_watford_salary(relations_manager):
    """Check if Gretchen Watford's base salary equals 4000$"""
    gretchen = next(e for e in relations_manager.get_all_employees() if e.first_name == "Gretchen" and e.last_name == "Watfor
    assert gretchen.base_salary == 4000
```

a

4.1.5 test_tomas_andre_not_a_leader

Leírás: Teszteli, hogy Tomas Andre nem csapatvezető, és ellenőrzi a viselkedést, amikor megpróbáljuk lekérni a csapattagjait.

Bemenet: A RelationsManager példány Tomas Andre alkalmazotti adataival.

Elvárt kimenet:

- Tomas Andre nem azonosítható csapatvezetőként
- Csapattagjainak lekérése None értéket ad vissza

Implementáció:

```python
def test_tomas_andre_not_a_leader(relations_manager):
    """Make sure Tomas Andre is not a team leader and check behavior when retrieving his team members"""
    tomas = next(e for e in relations_manager.get_all_employees() if e.first_name == "Tomas" and e.last_name == "Andre")
    assert not relations_manager.is_leader(tomas)
    assert relations_manager.get_team_members(tomas) is None
```

4.1.6 test_jude_overcash_not_in_db

Leírás: Biztosítja, hogy Jude Overcash nem létezik az alkalmazotti adatbázisban.

Bemenet: A RelationsManager példány minden alkalmazotti adattal.

Elvárt kimenet: Nem található Jude Overcash nevű alkalmazott.

Implementáció:

```python
def test_jude_overcash_not_in_db(relations_manager):
    """Make sure that Jude Overcash is not stored in the database"""
    employees = relations_manager.get_all_employees()
    assert not any(e.first_name == "Jude" and e.last_name == "Overcash" for e in employees)
```

4.2 EmployeeManager Tesztek

4.2.1 test_salary_non_leader

Leírás: Teszteli a fizetés kiszámítását egy olyan alkalmazott esetében, aki nem csapatvezető, 1998. október 10-én vették fel, és az alapfizetése 1000$.

Bemenet:

- Egy EmployeeManager példány
- Egy teszt alkalmazott specifikus attribútumokkal (felvételi dátum: 1998-10-10, alapfizetés: 1000$)

Elvárt kimenet:

- A kiszámított fizetés 3700$ (1000$ alap + 100$ * 27 év a cégnél)

Implementáció:

```python
def test_salary_non_leader(employee_manager):
    """Check an employee's salary who is not a team leader, hired on 10.10.1998, with base salary 1000$."""
    employee = Employee(id=99, first_name="Test", last_name="Employee", base_salary=1000,
                        birth_date=datetime.date(1980, 1, 1), hire_date=datetime.date(1998, 10, 10))
    expected_salary = 1000 + (2025 - 1998) * 100  # 3700
    assert employee_manager.calculate_salary(employee) == expected_salary
```

4.2.2 test_salary_team_leader

Leírás: Teszteli a fizetés kiszámítását egy 3 csapattaggal rendelkező csapatvezető esetében, akit 2008. október 10-én vettek fel, és az alapfizetése 2000$.

Bemenet:

- EmployeeManager és RelationsManager példányok
- Egy teszt csapatvezető specifikus attribútumokkal (felvételi dátum: 2008-10-10, alapfizetés: 2000$)
- Három, a vezetőhöz rendelt csapattag

Elvárt kimenet:

- A kiszámított fizetés 4300$ (2000$ alap + 100$ * 17 év + 200$ * 3 csapattag)

Implementáció:

```python
def test_salary_team_leader(employee_manager, relations_manager):
    """Check an employee's salary who is a team leader with 3 team members, hired on 10.10.2008, base salary 2000$."""
    leader = Employee(id=100, first_name="Leader", last_name="Example", base_salary=2000,
                      birth_date=datetime.date(1975, 6, 15), hire_date=datetime.date(2008, 10, 10))

    team_members = [
        Employee(id=101, first_name="Member1", last_name="Example", base_salary=1500,
                 birth_date=datetime.date(1990, 1, 1), hire_date=datetime.date(2015, 1, 1)),
        Employee(id=102, first_name="Member2", last_name="Example", base_salary=1600,
                 birth_date=datetime.date(1992, 2, 2), hire_date=datetime.date(2016, 2, 2)),
        Employee(id=103, first_name="Member3", last_name="Example", base_salary=1700,
                 birth_date=datetime.date(1993, 3, 3), hire_date=datetime.date(2017, 3, 3))
    ]

    relations_manager.teams[100] = [101, 102, 103]  # Hozzárendeljük a team tagokat
    relations_manager.employee_list.append(leader)  # Hozzáadjuk a vezetőt az employee listához
    relations_manager.employee_list.extend(team_members)  # Hozzáadjuk a csapattagokat is

    expected_salary = 2000 + (2025 - 2008) * 100 + 3 * 200  # 4300$
    assert employee_manager.calculate_salary(leader) == expected_salary
```

4.2.3 test_salary_email_notification

Leírás: Teszteli, hogy az e-mail értesítési funkció helyesen működik-e az alkalmazott fizetésének kiszámításakor.

Bemenet:

- Egy EmployeeManager példány
- Egy alkalmazott specifikus attribútumokkal

Elvárt kimenet:

- A funkciónak ki kell nyomtatnia egy értesítési üzenetet az alkalmazott nevével és a kiszámított fizetéssel

Implementáció:

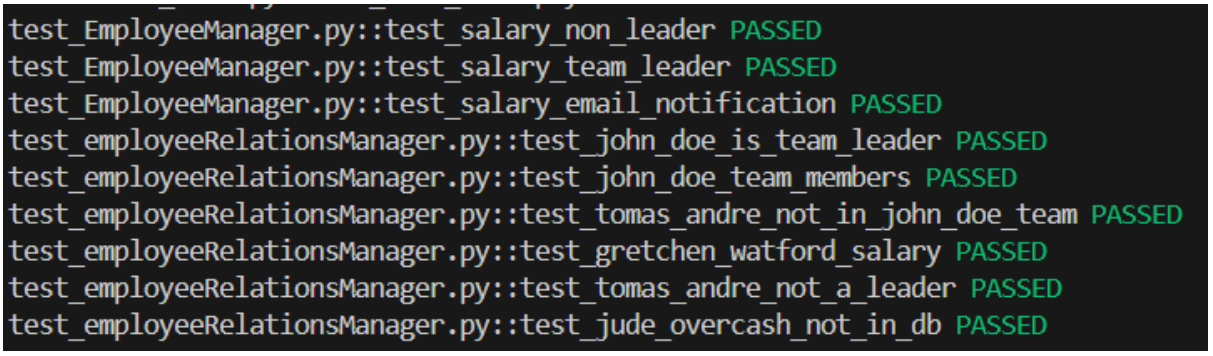
```python
def test_salary_email_notification(employee_manager):
    """Ensure salary calculation sends correct notification."""
    employee = Employee(id=101, first_name="Notify", last_name="Test", base_salary=2500,
                        birth_date=datetime.date(1985, 5, 20), hire_date=datetime.date(2010, 6, 15))

    expected_salary = 2500 + (2025 - 2010) * 100  # 4000
    with patch("builtins.print") as mock_print:
        employee_manager.calculate_salary_and_send_email(employee)
        mock_print.assert_called_with("Notify Test your salary: 4000 has been transferred to you.")
```

5. Teszt lefedettség és eredmények

Minden teszt sikeresen implementálva lett és átmegy a pytest végrehajtásakor. A tesztek lefedik a projekt leírásában meghatározott összes követelményt:

- Csapatvezetők és csapatösszetételek ellenőrzése
- Alkalmazotti adatok és fizetések validálása
- Bizonyos alkalmazottak nemlétezésének ellenőrzése
- Fizetésszámítások tesztelése különböző alkalmazotti típusokra és forgatókönyvekre
- Az e-mail értesítési rendszer ellenőrzése



6. Következtetés

Az alkalmazotti menedzsment rendszerhez készített egységtesztek megvalósítása értékes betekintést nyújtott a szoftvertesztelés folyamatába. A pytest használatával átfogó tesztcsomagot hoztunk létre, amely ellenőrzi a rendszer funkcionalitását. A tesztek biztosítják, hogy:

- Az alkalmazotti kapcsolati struktúra helyesen van fenntartva
- A fizetés számításai pontosak a különböző tényezők alapján
- A rendszer helyesen azonosítja a csapatvezetőket és csapattagokat
- A fizetés kiszámításakor a megfelelő értesítések elküldésre kerülnek

.