Testování softwaru (KI/TSW)

Pavel Beránek

25 Března, 2025

6. Testování API

6.1. Aplikační programové rozhraní

API (Application Programming Interface) je rozhraní, které umožňuje komunikaci mezi dvěma softwarovými systémy. Jedná se o formu dohody/závazku ohledně formy komunikace.

Příklady API: - Aplikace v mobilu si pomocí API stahuje počasí ze serveru. - Webová stránka volá API YouTube, aby zobrazila videa. - Aplikace e-shopu volá API, aby vytvořila objednávku. - Třída kalkulačka poskytuje metody pro základní aritmetické operace.

Pojďme se nejprve zaměřit na poslední zmíněný příklad, který souvisí s obecným programováním.

Příkladem této situace může být následující pseudokód, ve kterém jsou vybrané metody k veřejnémá používání (jinými třídami) tzv. veřejné (public) a vybrané metody pro interní používání třídou jsou soukromé (private).

```
class Kalkulačka:

  public konstruktor(API_klíč, URL_na_AI):
    ChytraAI = připojit_se(URL_na_AI, API_klíč)

  public num sečti(a, b):
    součet = zeptejse_AI_protoze_neumis_pocitat(a, b, "sečti")
    return součet

  public num odečti(a, b):
    rozdíl = zeptejse_AI_protoze_neumis_pocitat(a, b, "rozdíl")
    return rozdíl

  public num vynásob(a, b):
    součin = zeptejse_AI_protoze_neumis_pocitat(a, b, "násobení")
    return součin

  public num vyděl(a, b):
    podíl = zeptejse_AI_protoze_neumis_pocitat(a, b, "dělení")
    return podíl

  private num zeptejse_AI_protoze_neumis_pocitat(a, b, operace):
    return ChytraAI.request("Proved operaci {operace} nad cisli {a}, {b}")
```

Tyto veřejné metody vytváří tzv. API pro třídu Kalkulačka. Korektní postup by byl dále takový, že existuje veřejné rozhraní jako programová struktura, která pro ostatní třídy poskytuje informace o závazku svého chování. To se děla ve většině jazycích programovou strukturou Interface nebo Abstraktní třídou. Python má pro takové závazky relativně nový mechanismus Protokolů.

```
public interface IKalkulačka:
  public num sečti(a, b)
  public num odečti(a, b)
  public num vynásob(a, b)
  public num vyděl(a, b)
```

Pokud nějaká jiná třída využívá metod kalkulačky, tak by se měl dobrovolně omezit na její veřejné rozhraní (přímo kódem).

```
# rozhraní             # třída
IKalkulačka kalk = new Kalkulačka()
```

Tím se zajistí, že levá strana přiřazování v právě napsaném výrazu může kdykoliv nahradit instanci kalkulačky na pravé straně za instanci libovolné jiné třídy splňující rozhraní ve výrazu vlevo. Nic by se v kódu kromě jedné strany přiřazovacího výrazu nemělo měnit.

Této programátorské technice se říká "programování vůči rozhraní (programming to an interface)", které je technickou realizací klíčového principu dobré/čisté architektury (clean architecture) s názvem "Dependency Inversion Principle (DIP)", což je jedno z pěti klíčových pravidel zvaných SOLID principy.

Vzhledem k tomu, že nám rozhraní (Interface, Abstraktní třídy, Protokoly) vytváří API, tak lze jednoduše psát integrační testy s využitím testovacích dvojníků, které testují propojení komponent přes jejich abstraktní závislosti.

6.2. REST API

Každý datový sklad (Relační databáze, Tabulka v Excelu, .txt soubor) potřebuje pro plné využívání ve formě datového skladu 4 operace - Create, Read, Update, Delete (CRUD operace). Create vytváří do skladu nové záznamy, Read čte existující záznamy, Update upravuje existující záznamy, Delete maže existující záznamy. Webové portály také potřebují pracovat v převážné míře jako datové sklady.

Standardní možností bylo přihlásit se do webového portálu přes přihlašovací formulář a ukládat údaj o přihlášení do tzv. session. Pokud byl uživatel v session aktivní, pak mu byly tyto operace umožněné a to přes operace GET a POST z HTTP protokolu. Služba tedy musela nést informace o stavu přihlášení uživatele - přihlášen/nepřihlášen. Takový způsob komunikace s uživatelem se nazývá stavový. Některé webové portály však vyžadují nestavový způsob operování s datovým skladem.

REST API (Representational State Transfer) je konkrétní způsob, jak navrhnout API webových služeb bezestavově. Volené CRUD operace (a další) se programují na základě typu metody HTTP protokolu.

HTTP metody:

- POST: vytvoř nová data (Create operace
- GET: získej data (Read operace)
- PUT: nahraď existující data (Update operace)
- PATCH: uprav data (také Update operace, méně agresivní, ale skoro nikdo takto REST nerealizuje)
- DELETE: smaž data (Delete operace)

Pomocí těchto metod pak pracujete s webovým portálem jako s datovým skladem, kde pomocí URL blíže specifikujete operativní místo (tzv. zdroj). Zdroje se nám typicky vrací ve formátu JSON (JavaScript Object Notation). Architektura URL zdrojů by měla být řádně promyšlená. Příklad koncových bodů (endpointů) pro zdroje:

- /users – seznam uživatelů
- /users/1 – konkrétní uživatel s ID 1

Pro silnější metody dotazování existují i varianty REST API jako je GraphQL.

```
import requests

def test_graphql_user_query():
    url = "https://api.example.com/graphql"
    query = """
    query {
      user(id: 1) {
        id
        name
      }
    }
    """
    response = requests.post(url, json={"query": query})

    assert response.status_code == 200

    data = response.json()

    # Ověření základní struktury
    assert "data" in data
    assert "user" in data["data"]
    assert isinstance(data["data"]["user"]["id"], int)
    assert isinstance(data["data"]["user"]["name"], str)
```

Možná si teď říkáte, že webové portály řešeny bezestavově by nebyly příliš k užitku z pohledu businessu, jelikož často si chcete za služby poskytované Vaším serverem zaplatit a bezestavovým způsobem by nemusel platit nikdo (Vám to jen bude brát energetické zdroje). Řešením je bezestavová komunikace s identifikací uživatele. Identifikátor je tajný unikátní řetězec, který se s požadavkem zasílá a říká se mu token.

6.3. Testování REST API

Když teď máme ponětí o tom, co je to REST API, tak ho pojďme otestovat. Jedná se opravdu o velkou část celého testování.

Pokud máte malinký projekt a chcete mít všechny testy napsané v pythonu (třeba provozované pytestem), pak lze REST API testovat díky knihovně requests.

```
pip install requests
```

Následující kód ukazuje jen pomocí základního příkazu assert, jak by takový test vypadal:

```
import requests

def test_product_exists():
    # neexistující webový portál a endpoint, jen pro ukázku
    response = requests.get("http://api.mujshop.cz/products/1")
    assert response.status_code == 200
    assert response.json()["name"] == "Chytrý telefon"
```

V navracené odpovědi z webového serveru se typicky testují následující prvky:

- stavové kódy: mají odpovědi ze serveru správný kód? (200, 404)
- chybové stavy: jsou chybové stavy správné? (400 vs 500)
- validita dat: jsou data ve správném formátu a se správným obsahem?
- autentizace: je přístup řádně chráněn před přístupem bez přihlašovacího tokenu?
- zabezpečení: neunikají tajná data? Není náchylné zmást API pomocí penetračních útoků?
- výkon: odpovídá server dostatečně rychle?

Test stavového kódu:

```
# Chceme ověřit, že API vrací očekávaný stavový kód (např. 200 OK).
def test_status_code():
    #jsonplaceholder.typicode.com je webový portál pro testování kódu pracující s REST API
    response = requests.get("https://jsonplaceholder.typicode.com/posts/1")
    assert response.status_code == 200
```

Test chybových stavů:

```
# Tento test na portálu jsonplaceholder neprojde, je to hodnej server a nevhodný POST přijme :)
def test_bad_request():
    bad_data = {"title": 1234}  # špatný datový typ
    response = requests.post("https://jsonplaceholder.typicode.com/posts", json=bad_data)
    assert response.status_code in [400, 422]  # Záleží na implementaci kódu na serveru
```

Test validity dat:

```
#pip install jsonschema
from jsonschema import validate

# Ověřujeme, že struktura odpovědi odpovídá očekávanému schématu.
post_schema = {
    "type": "object",
    "properties": {
        "userId": {"type": "integer"},
        "id": {"type": "integer"},
        "title": {"type": "string"},
        "body": {"type": "string"}
    },
    "required": ["userId", "id", "title", "body"]
}

def test_json_schema():
    response = requests.get("https://jsonplaceholder.typicode.com/posts/1")
    data = response.json()
    validate(instance=data, schema=post_schema)
```

Test autentizace:

```
# Ověřujeme, že API je chráněno a bez tokenu odmítne přístup.
def test_auth_required():
    response = requests.get("https://api.openweathermap.org/data/2.5/weather?lat=0&lon=0")
    assert response.status_code == 401  # Unauthorized

def test_auth_success():
    token = "SkákalBeránekPřesOves"  # svůj tam rozhodně nenapíšu :)
    response = requests.get(f"https://api.openweathermap.org/data/2.5/weather?lat=0&lon=0&appid={token}")
    assert response.status_code == 200
```

Test zabezpečení (nejzákladnější možný):

```
# Ověříme, že API nevrací interní chyby nebo citlivé informace.
def test_no_stack_trace_on_error():
    response = requests.get("https://jsonplaceholder.typicode.com/invalid-endpoint")
    assert response.status_code in [404, 400]
    assert "Exception" not in response.text
    assert "Traceback" not in response.text
```

Test výkonu (rychlosti odezvy):

```
# Změříme, jak dlouho trvá odpověď (response time).
import time

def test_response_time():
    start = time.time()
    response = requests.get("https://jsonplaceholder.typicode.com/posts")
    end = time.time()
    duration = end - start
    assert duration < 0.5  # musí odpovědět do půl sekundy
```

6.4. Testování REST API pomocí POSTMAN

Nejznámější nástroj na testování REST API se nazývá Postman, který je základním znalostním předpokladem každého člověka, který se uchází o pozici v softwarovém testování. POSTMAN je jádrem tohoto cvičení a chtěl bych, ať se s ním opravdu podrobně doma naučíte. V této lekci si ho jen nakousnete a doma si prosím dodělejte vše (třeba i včetně certifikace).

Důležité partie Postmana:

- První kroky s Postman ZDE
- Zasílání požadavků (primárně zkusit) ZDE
- Psaní skriptů v Postman pro princip "Early testing" ZDE
- Efektivní práce s kolekcemi testů (test suite) ZDE
- Postman reportování ZDE
- Nově: vizuální editor Postman Flows ZDE
- Nově: tvorba testovacích AI agentů ZDE
- Postman akademie ZDE

Udělejte jednu z následujících variant cvičení:

1. Otestujte REST API rozhraní libovolného webového portálu.
2. Otestujte REST API Vašeho vlastního portálu nad kterým máte kontrolu (např. z mého cvičení NSQL: ZDE.
3. Projděte si tutoriál na POSTMANa a testujte weby z jejich portálu (trošku zdlouhavé).

Domácí cvičení

1. Dokončete celý tutoriál na Postmana.
2. Vytvořte si kolekci testů v Postmanovi testující CRUD operace na nějakém lokálním nebo existujícícm webovém serveru.