Modul 11
Python Pemrograman Jaringan

Topik Bahasan

- **Membuat Aplikasi Jaringan**
- **Melakukan Debugging**
- **Menguji Aplikasi Jaringan**

Untuk mendapatkan pemahaman yang mendalam tentang debugging dan pengujian aplikasi jaringan dalam Python, mari kita bahas contoh implementasi lengkap termasuk hasil running dan pembahasannya. Kita akan mengcover beberapa aspek penting:

- **Membuat Aplikasi Jaringan**
- **Melakukan Debugging**
- **Menguji Aplikasi Jaringan**

1. Membuat Aplikasi Jaringan

Untuk contoh ini, kita akan membuat aplikasi web sederhana menggunakan Flask, yang akan melayani permintaan HTTP.

a. Membuat Server HTTP dengan Flask

```
python
Copy code
# server_http.py
from flask import Flask, jsonify, request

app = Flask(__name__)

# Data contoh
data = {
    'users': [
        {'id': 1, 'name': 'Alice'},
        {'id': 2, 'name': 'Bob'}
    ]
}

@app.route('/')
def home():
    return "Hello, World!"

@app.route('/api/users', methods=['GET'])
def get_users():
    return jsonify(data['users'])
```

```
@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
    user = next((u for u in data['users'] if u['id'] == user_id), None)
    if user:
        return jsonify(user)
    else:
        return jsonify({'error': 'User not found'}), 404

@app.route('/api/users', methods=['POST'])
def add_user():
    new_user = request.get_json()
    data['users'].append(new_user)
    return jsonify(new_user), 201

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

Penjelasan:

- **Endpoint `/`**: Mengembalikan pesan "Hello, World!".
- **Endpoint `/api/users`**: Mengembalikan daftar pengguna.
- **Endpoint `/api/users/<int:user_id>`**: Mengembalikan detail pengguna berdasarkan ID.
- **Endpoint POST `/api/users`**: Menambahkan pengguna baru ke data.

2. Debugging Aplikasi Jaringan

a. Menggunakan `print` Statements

Tambahkan `print` statements untuk melihat data yang diterima dan dikirim:

```
python
Copy code
@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
    print(f"Request for user ID: {user_id}")  # Debug statement
    user = next((u for u in data['users'] if u['id'] == user_id), None)
    if user:
        print(f"User found: {user}")  # Debug statement
        return jsonify(user)
    else:
        print("User not found")  # Debug statement
        return jsonify({'error': 'User not found'}), 404
```

b. Menggunakan `pdb` (Python Debugger)

Tambahkan breakpoint menggunakan `pdb`:

```
python
Copy code
import pdb
```

```
@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
    pdb.set_trace()  # Set a breakpoint
    user = next((u for u in data['users'] if u['id'] == user_id), None)
    if user:
        return jsonify(user)
    else:
        return jsonify({'error': 'User not found'}), 404
```

Cara Menggunakan `pdb`:

- **`n`**: Langkah ke baris berikutnya.
- **`c`**: Melanjutkan eksekusi.
- **`p <variable>`**: Menampilkan nilai variabel.

3. Mengujian Aplikasi Jaringan

a. Pengujian Unit dengan `unittest`

Tulis tes unit menggunakan `unittest` untuk Flask server:

```
python
Copy code
# test_server.py
import unittest
from server_http import app

class FlaskTestCase(unittest.TestCase):

    def setUp(self):
        self.client = app.test_client()
        self.client.testing = True

    def test_home_page(self):
        response = self.client.get('/')
        self.assertEqual(response.data, b'Hello, World!')

    def test_get_users(self):
        response = self.client.get('/api/users')
        self.assertEqual(response.json, [
            {'id': 1, 'name': 'Alice'},
            {'id': 2, 'name': 'Bob'}
        ])

    def test_get_user(self):
        response = self.client.get('/api/users/1')
        self.assertEqual(response.json, {'id': 1, 'name': 'Alice'})

    def test_get_user_not_found(self):
        response = self.client.get('/api/users/999')
        self.assertEqual(response.status_code, 404)
        self.assertEqual(response.json, {'error': 'User not found'})
```

```
    def test_add_user(self):
        new_user = {'id': 3, 'name': 'Charlie'}
        response = self.client.post('/api/users', json=new_user)
        self.assertEqual(response.status_code, 201)
        self.assertEqual(response.json, new_user)
        # Verify that the user was actually added
        response = self.client.get('/api/users/3')
        self.assertEqual(response.json, new_user)

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

b. Pengujian Integrasi dengan `pytest`

Gunakan `pytest` untuk pengujian yang lebih fleksibel:

```
python
Copy code
# test_server.py
import pytest
from server_http import app

@pytest.fixture
def client():
    app.testing = True
    return app.test_client()

def test_home_page(client):
    response = client.get('/')
    assert response.data == b'Hello, World!'

def test_get_users(client):
    response = client.get('/api/users')
    assert response.json == [
        {'id': 1, 'name': 'Alice'},
        {'id': 2, 'name': 'Bob'}
    ]

def test_get_user(client):
    response = client.get('/api/users/1')
    assert response.json == {'id': 1, 'name': 'Alice'}

def test_get_user_not_found(client):
    response = client.get('/api/users/999')
    assert response.status_code == 404
    assert response.json == {'error': 'User not found'}

def test_add_user(client):
    new_user = {'id': 3, 'name': 'Charlie'}
    response = client.post('/api/users', json=new_user)
    assert response.status_code == 201
    assert response.json == new_user
    # Verify that the user was actually added
    response = client.get('/api/users/3')
```

```
    assert response.json == new_user
```

4. Hasil Running dan Pembahasan

a. Menjalankan Server

Jalankan server Flask dengan:

```
bash
Copy code
python server_http.py
```

Output Server:

```
csharp
Copy code
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Server aktif di `http://127.0.0.1:5000`.

b. Menjalankan Pengujian

Jalankan pengujian menggunakan `unittest` atau `pytest`:

```
bash
Copy code
python -m unittest test_server.py
```

atau

```
bash
Copy code
pytest
```

Output Pengujian:

- Untuk `unittest`, output mungkin terlihat seperti ini:

```
markdown
Copy code
....
----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK
```

- Untuk `pytest`, output mungkin terlihat seperti ini:

```
diff
Copy code
```

```
============================= test session starts
==============================
platform darwin -- Python 3.9.6, pytest-6.2.5, pluggy-0.13.1
collected 5 items

test_server.py .....
[100%]

============================== 5 passed in 0.02s
==============================
```

Penjelasan Output Pengujian:

- Semua tes berhasil menunjukkan bahwa aplikasi berjalan sesuai dengan harapan.
- Tes `test_home_page` memastikan bahwa endpoint root berfungsi.
- Tes `test_get_users` dan `test_get_user` memverifikasi bahwa data pengguna dikembalikan dengan benar.
- Tes `test_add_user` memastikan bahwa pengguna baru ditambahkan dan dapat diambil dari API.

Kesimpulan

- **Debugging**: Gunakan `print` statements, `pdb`, dan `logging` untuk menemukan dan memperbaiki masalah dalam aplikasi jaringan.
- **Pengujian**: Gunakan pustaka seperti `unittest` dan `pytest` untuk menulis dan menjalankan tes unit dan integrasi pada aplikasi jaringan.