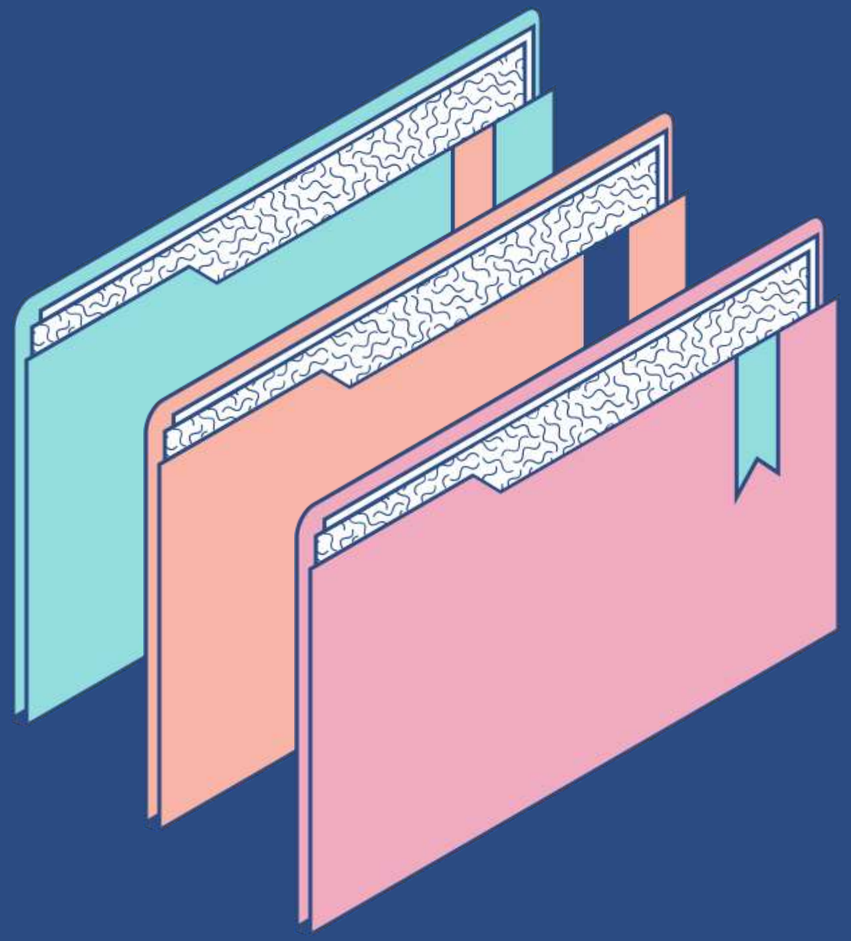
TESTING & QA PERANGKAT LUNAK

White Box & Unit Test

Ujian Tengah Semester

Nama : Anisa Aprilia Pasyah
Nim : 201011402007



Pengertian White Box

White Box, adalah suatu metode pengujian aplikasi yang menggunakan penjelasan struktur kontrol sebagai bagian dari component-level design untuk membuat test cases. White Box sendiri mempunyai beberapa teknik di dalam pengujiannya, seperti : Data Flow Testing, Control Flow Testing, Basic Path / Path Testing, dan Loop Testing. Dalam Pengujian White Box para penguji perlu mengetahui secara dalam source code yang akan diuji. Pengujian White Box dapat mengungkap kesalahan implementasi dari sebuah aplikasi. Pengujian ini dapat diterapkan pada tingkatan integrasi, unit dan system.

Implementasi White Box

Dalam pembuatan aplikasi form login ini, aplikasi melakukan validasi dengan memeriksa file yang berada di dalam folder yang sama di mana aplikasi python dijalankan. Nama penggunanya diambil dari nama file yang dibuat sedangkan kata sandinya diambil dari isi file yang ada di dalam file tersebut. Dalam pengujian aplikasi ini telah dibuat sebuah file dengan nama “farhan” yang dijadikan nama pengguna dan didalamnya berisi kata sandi “197006024”.

Source Code Validasi Login

```
def login_verify():
 username1 = username_verify.get()
 password1 = password_verify.get()

 list_of_files = os.listdir()
  if not username1:
   if not password1:
    form_not_filled()
   else:
    user_not_filled()
  elif username1 not in list_of_files:
   user_not_found()
  elif username1 in list_of_files:
   file1 = open(username1, "r")
   verify = file1.read().splitlines()
    if password1 in verify:
            login_success()
    elif not password1:
          password_not_filled()
  else:
   password_not_recognised()
```

Pengertian Unit Test

Unit testing adalah sebuah langkah pengujian terhadap perangkat lunak atau komponen dari sebuah perangkat lunak. Biasanya, unit testing dilakukan disaat masa development atau pengembangan dari sebuah aplikasi yang dilakukan oleh developer. Pengujian unit testing ini meliputi dari function, method, procedure, module, serta object.

Implementasi Unit Test

Dalam implementasi ini kita akan membahas tentang penggunaan dasar modul unittest Python dan menulis beberapa kasus pengujian unit python untuk menguji fungsi kelas.

Untuk membuat kode untuk menguji unitnya. Kita akan membuat Kelas Python. Tujuan utama kelas python adalah untuk menyimpan dan mengambil nama seseorang. Jadi, kita menulis set_name() fungsi untuk meniympan data dan get_name() fungsi untuk mengambil nama dari kelas.

```
class Person:
    name = []

    def set_name(self, user_name):
        self.name.append(user_name)
        return len(self.name) - 1

    def get_name(self, user_id):
        if user_id >= len(self.name):
            return 'There is no such user'
        else:
            return self.name[user_id]


if __name__ == '__main__':
    person = Person()
    print('User Abbas has been added with id ', person.set_name('Abbas'))
    print('User associated with id 0 is ', person.get_name(0))
```

Selanjutnya kode untuk pengujian unit. Sebuah testcase individual dibuat dengan membuat subkelas unittest.TestCase. Dengan mengganti atau menambahkan fungsi yang sesuai, kita dapat menambahkan logika untuk diuji. Kode berikut akan berhasil jika a sama dengan b.

Source Code Unit Test (kelas python)

```
import unittest


class Testing(unittest.TestCase):
    def test_string(self):
        a = 'some'
        b = 'some'
        self.assertEqual(a, b)

    def test_boolean(self):
        a = True
        b = True
        self.assertEqual(a, b)

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

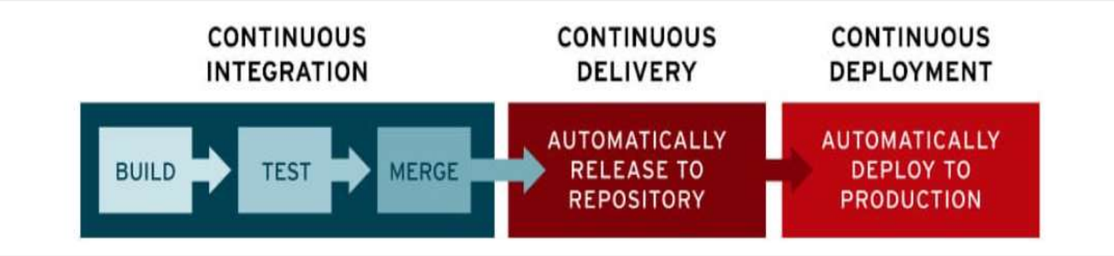
Implementasi Unit Test

Setelah itu saatnya untuk pengujian unit untuk kelas sumber kita Person. Di kelas ini kita telah mengimplementasikan dua fungsi get_name() dan set_name().
Sekarang, kita akan menguji fungsi tersebut menggunakan unittest. Jadi kita telah merancang dua kasus uji untuk kedua fungsi tersebut. Perhatikan kode berikut,

```
import unittest

# This is the class we want to test. So, we need to import it
import Person as PersonClass


class Test(unittest.TestCase):
    """
    The basic class that inherits unittest.TestCase
    """
    person = PersonClass.Person()  # instantiate the Person Class
    user_id = []  # variable that stores obtained user_id
    user_name = []  # variable that stores person name

    # test case function to check the Person.set_name function
    def test_0_set_name(self):
        print("Start set_name test\n")
        """
        Any method which starts with ``test_`` will considered as a test case.
        """
        for i in range(4):
            # initialize a name
            name = 'name' + str(i)
            # store the name into the list variable
            self.user_name.append(name)
            # get the user id obtained from the function
            user_id = self.person.set_name(name)
            # check if the obtained user id is null or not
            self.assertIsNotNone(user_id)  # null user id will fail the test
            # store the user id to the list
            self.user_id.append(user_id)
        print("user_id length = ", len(self.user_id))
        print(self.user_id)
        print("user_name length = ", len(self.user_name))
        print(self.user_name)
        print("\nFinish set_name test\n")

    # test case function to check the Person.get_name function
    def test_1_get_name(self):
        print("\nStart get_name test\n")
        """
        Any method that starts with ``test_`` will be considered as a test case.
        """
        length = len(self.user_id)  # total number of stored user information
        print("user_id length = ", length)
        print("user_name length = ", len(self.user_name))
        for i in range(6):
            # if i not exceed total length then verify the returned name
            if i < length:
                # if the two name not matches it will fail the test case
                self.assertEqual(self.user_name[i], self.person.get_name(self.user_id[i]))
            else:
                print("Testing for get_name no user test")
                # if length exceeds then check the 'no such user' type message
                self.assertEqual('There is no such user', self.person.get_name(i))
        print("\nFinish get_name test\n")


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

Perhatikan bahwa modul unittest menjalankan fungsi pengujian sesuai urutan namanya, bukan sesuai urutan definisinya.
Dan karena kita ingin pengujian set_name kita dijalankan terlebih dahulu, kita menamai fungsi kasus pengujian kita sebagai test_0_set_name dan test_1_get_name.

Pengertian CI/CD

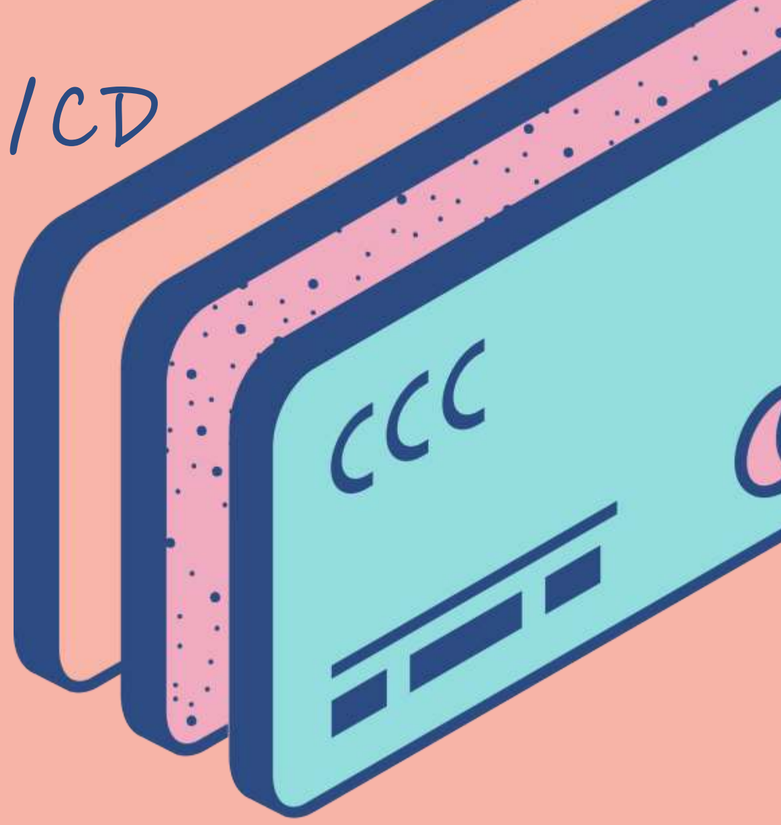


CI/CD adalah metode pengembangan perangkat lunak dengan mengotomatisasi setiap proses yang dilakukan. Tujuan CI/CD adalah website atau aplikasi yang dihasilkan punya performa yang andal dan minim bug.

Manfaat CI/CD yang bisa Anda rasakan, yaitu:

1. Mampu mendeteksi program error sejak dini. Sebab, setiap kode yang disubmit oleh developer akan diuji terlebih dahulu. Jika lolos testing, maka pengembangan software bisa berlanjut ke tahap selanjutnya.
2. Dapat meningkatkan produktivitas seluruh personil DevOps. Alasan yang pertama, karena proses testing akan dilakukan secara otomatis, alih-alih manual seperti yang biasa dilakukan.
3. Sanggup mempercepat waktu perilisan software. Hal Ini dinikmati karena error bisa terdeteksi lebih awal, produktivitas tim yang tinggi, serta kolaborasi yang dilakukan terus menerus.

IMPLEMENTASI & KONFIGURASI CI/CD



1. Konfigurasi Repositori

1. Buat repositori Git Untuk proyek di layanan seperti GitHub atau GitLab
2. Buat berkas '.gitignore' untuk mengabaikan berkas0berkas yang tidak perlu dimasukkan ke dalam repositori

2. Konfigurasi Jenkins

1. Instasl Jenkins di server dan pastikan itu berjalan
2. Buka Jenkins di browser dan konfigurasi Jenkinsfile
3. Pasang plugin "Docker" pada Jenkins
4. Konfigurasi Jenkins untuk mengintegrasikan dengan repositori Git

3. Konfigurasi Docker

1. Buat berkas Dockerfile untuk proyek.
2. Buat berkas 'docker-compose.yml' untuk konfigurasi layanan Docker

4. Konfigurasi Docker

1. Setiap kali ada perubahan pada repositori Git, Jenkins akan secara otomatis menjalankan pipeline CI/CD.
2. Jenkins akan mengambil kode dari repositori Git, membangun Docker image, menjalankan pengujian, dan akhirnya melakukan penyebaran dengan menggunakan Docker Compose.
3. Hasil dari pipeline dapat dilihat di Jenkins Dashboard, dan Anda juga dapat mengatur notifikasi jika ada kesalahan dalam proses ini.

Referensi

https://jurnal.unsil.ac.id/index.php/jssainstek/article/view/4086/1929

https://www.codepolitan.com/blog/apa-itu-unit-testing-yuk-kenalan/

https://www.digitalocean.com/community/tutorials/python-unittest-unit-test-example

https://www.niagahoster.co.id/blog/ci-cd-adalah/