Tugas 1

Software Testing and Quality Assurance

Disusun Oleh:

Alif Rezky Maulana

H071221014

PROGRAM STUDI SISTEM INFORMASI

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS HASANUDDIN

2024

1. Penjelasan, Langkah/proses dan contoh kasus dari:

A. White-Box Testing

Buku *The Art of Software Testing* mendalami konsep *white-box testing* namun dengan pendekatan yang lebih mendasar dan teknis. Myers mengarahkan pembaca untuk memahami bahwa tujuan utama *white-box testing* adalah mengidentifikasi dan menangkap kesalahan pada tingkat yang sangat rendah, termasuk kesalahan logika, cabang, dan jalur eksekusi.

Buku ini membahas beberapa teknik utama yang mirip dengan yang dibahas Pressman, tetapi menambahkan detail lebih lanjut tentang:

- **Statement Coverage**: Setiap baris kode harus dijalankan setidaknya satu kali untuk memverifikasi bahwa tidak ada baris kode yang tidak tercakup.
- **Decision Coverage**: Setiap cabang logika diuji untuk memastikan semua hasil keputusan tercakup.
- **Multiple Condition Coverage**: Setiap kondisi logika diuji dalam semua kombinasi yang mungkin.

Secara umum, Myers juga menekankan bahwa *white-box testing* cocok untuk menemukan kesalahan dalam logika program yang mungkin sulit diidentifikasi melalui *black-box testing*.

Langkah/Proses White-box Testing:

Berikut adalah langkah-langkah umum dalam melakukan *white-box testing* yang disarankan dalam buku *Software Engineering: A Practitioner's Approach* oleh Roger S. Pressman dan *The Art of Software Testing* oleh Glenford J. Myers.

1. Memahami Kode yang Akan Diuji

- **Analisis Struktur Kode**: Sebelum pengujian dimulai, pahami struktur internal, logika, dan alur data dari kode yang akan diuji. Pelajari diagram alur, *flowchart*, atau *pseudocode* jika ada.
- **Identifikasi Jalur Eksekusi Utama**: Tentukan jalur eksekusi utama dalam kode untuk memastikan semua kemungkinan jalur diidentifikasi.

2. Membuat Kasus Uji Berdasarkan Struktur Kode

- **Mengidentifikasi Statement dan Branch**: Setiap pernyataan atau *statement* dalam kode harus diuji setidaknya sekali. Jika ada kondisi atau cabang (seperti pernyataan if-else atau switch), buat kasus uji untuk mengeksekusi setiap cabang tersebut.
- **Pembuatan Kasus Uji untuk Kondisi Logika**: Buat kasus uji untuk setiap kondisi logika agar semua kemungkinan nilai kondisi diuji, termasuk nilai true dan false.
- **Loop Testing**: Jika ada loop dalam kode, buat kasus uji untuk berbagai skenario, seperti:
    - Tidak memasuki loop sama sekali (iterasi 0).
    - Melakukan iterasi satu kali (iterasi minimal).
    - Melakukan iterasi maksimum.
    - Melakukan iterasi lebih dari jumlah batas, jika memungkinkan, untuk menguji kondisi ekstrem.

3. Melakukan Eksekusi Kasus Uji

- Jalankan setiap kasus uji yang telah dibuat pada kode. Lacak jalur yang diambil oleh setiap kasus uji untuk memastikan bahwa semua jalur dan kondisi telah diuji.
- Gunakan alat bantu atau debugger untuk mengamati alur eksekusi, variabel, dan keluaran program pada setiap langkah.

4. Analisis Hasil

- Setelah setiap kasus uji dijalankan, bandingkan hasil aktual dengan hasil yang diharapkan. Jika ada perbedaan, catat dan telusuri bagian kode yang menyebabkan kesalahan tersebut.
- Pastikan bahwa setiap pernyataan, cabang, kondisi, dan jalur telah diuji.

5. Perbaikan dan Pengulangan

- Jika ada bug yang ditemukan, perbaiki kode dan ulangi proses pengujian untuk memverifikasi bahwa perbaikan tidak menimbulkan masalah lain atau kesalahan baru.
- Jika diperlukan, tambahkan kasus uji baru untuk memastikan semua jalur eksekusi dalam kode tercakup.

6. Melakukan Laporan Pengujian

- Dokumentasikan hasil pengujian secara menyeluruh. Laporan ini harus mencakup semua jalur, kondisi, dan loop yang telah diuji, serta hasil dari setiap pengujian. Catat juga kasus uji yang menyebabkan kesalahan dan langkah perbaikannya.

Contoh Kasus:

Misalkan kita memiliki fungsi yang menentukan apakah sebuah bilangan genap:

```
def is_even(n):
    if n % 2 == 0:
        return True
    else:
        return False
```

Fungsi ini mengembalikan True jika n adalah bilangan genap, dan False jika sebaliknya.

Langkah-langkah White-Box Testing

Langkah 1: Memahami Kode

- Fungsi is_even memeriksa apakah n habis dibagi 2. Jika True, maka n adalah bilangan genap.
- Jika tidak, fungsi mengembalikan False.

Langkah 2: Membuat Kasus Uji Berdasarkan Struktur Kode

1. **Branch Testing**:
    - Uji kondisi ketika n % 2 == 0 bernilai True (contoh: n = 4).
    - Uji kondisi ketika n % 2 == 0 bernilai False (contoh: n = 3).

Langkah 3: Eksekusi Kasus Uji

Dari analisis di atas, kita bisa membuat dua kasus uji:

Kasus Uji	Input	Ekspektasi Output	Keterangan
1	n = 4	True	Menguji kondisi n adalah genap.
2	n = 3	False	Menguji kondisi n adalah ganjil.

Langkah 4: Analisis Hasil

- Jalankan kasus uji 1: is_even(4) seharusnya mengembalikan True.
- Jalankan kasus uji 2: is_even(3) seharusnya mengembalikan False.

Jika kedua kasus memberikan hasil yang sesuai, maka fungsi telah melewati pengujian *white-box*.

Langkah 5 dan 6: Dokumentasi

Setiap kasus uji dicatat, termasuk hasilnya:

- **Kasus uji 1**: n = 4 → Output: True, sesuai ekspektasi.
- **Kasus uji 2**: n = 3 → Output: False, sesuai ekspektasi.

Pengujian singkat ini memastikan bahwa fungsi bekerja dengan benar untuk kedua kondisi utama: bilangan genap dan ganjil.

B. Black-box Testing

Berdasarkan buku *Software Engineering: A Practitioner's Approach* oleh Roger S. Pressman dan *The Art of Software Testing* oleh Glenford J. Myers, Tom Badgett, dan Corey Sandler, inti dari black-box testing adalah menguji perangkat lunak berdasarkan spesifikasi tanpa melihat kode internalnya. Penguji fokus pada fungsi, antarmuka, dan respons perangkat lunak, serta memastikan hasil akhir sesuai dengan yang diharapkan pengguna.

Langkah/Proses Black-box Testing:

1. Pahami Spesifikasi Perangkat Lunak

- Pelajari fungsi yang harus dijalankan perangkat lunak berdasarkan spesifikasi, seperti apa input yang diterima dan output yang diharapkan.

2. Buat Kasus Uji Berdasarkan Teknik *Black-Box*

- **Equivalence Partitioning**: Buat kasus uji dengan membagi input ke dalam kelompok atau kelas yang menghasilkan perilaku serupa.
- **Boundary Value Analysis**: Uji nilai di sekitar batas (misalnya, jika input valid 1-100, uji nilai 0, 1, 100, 101).
- **Error Guessing**: Coba input ekstrem atau tidak biasa untuk memancing kesalahan.

3. Jalankan Kasus Uji

- Masukkan input dari setiap kasus uji ke perangkat lunak dan catat hasilnya.

4. Analisis dan Bandingkan Hasil

- Bandingkan hasil aktual dengan yang diharapkan. Jika ada ketidaksesuaian, identifikasi dan dokumentasikan sebagai masalah.

5. Perbaikan dan Uji Ulang

- Jika ada kesalahan, perbaiki perangkat lunak dan jalankan kembali pengujian untuk memastikan perbaikan berhasil.

6. Dokumentasi

- Simpan hasil pengujian, termasuk kasus uji, hasil yang diharapkan, hasil aktual, dan catatan kesalahan jika ada.

Contoh Kasus:

```
def validate_age(age):
    if 18 <= age <= 60:
        return "Valid age"
    else:
        return "Invalid age"
```

Fungsi ini mengembalikan "Valid age" jika umur berada dalam rentang 18-60, dan "Invalid age" jika di luar rentang tersebut.

Langkah-langkah Black-Box Testing Berdasarkan Fungsi Ini

Langkah 1: Pahami Spesifikasi Perangkat Lunak

- Fungsi ini menerima input age dan memberikan output:
    - "Valid age" untuk umur 18 hingga 60.
    - "Invalid age" untuk umur di luar rentang tersebut.

Langkah 2: Buat Kasus Uji Berdasarkan Teknik Black-Box

1. **Equivalence Partitioning**:
    - Bagi input menjadi kelas atau kelompok yang menghasilkan output serupa:
        - Umur valid: 18–60.
        - Umur tidak valid: kurang dari 18 dan lebih dari 60.
2. **Boundary Value Analysis**:
    - Uji nilai di batas rentang (18 dan 60), serta tepat di luar batas (17 dan 61).
3. **Error Guessing**:
    - Coba masukkan input yang tidak biasa atau ekstrem, misalnya age = -1 (umur negatif) dan age = 150.

Langkah 3: Jalankan Kasus Uji

Langkah 4: Analisis dan Bandingkan Hasil

- Jalankan setiap kasus uji pada fungsi validate_age dan catat hasilnya.
- Jika hasil aktual sesuai dengan hasil yang diharapkan, maka fungsi bekerja dengan baik untuk skenario tersebut.

Langkah 5: Perbaikan dan Uji Ulang

- Jika ditemukan ketidaksesuaian antara hasil aktual dan hasil yang diharapkan, perbaiki kode dan ulangi pengujian.

Langkah 6: Dokumentasi

- Simpan dokumentasi hasil pengujian dengan mencatat setiap kasus uji, hasil aktual, hasil yang diharapkan, dan catatan kesalahan jika ada.

C. Unit Testing

Berdasarkan buku "The Art of Software Testing" (edisi ketiga) oleh Glenford Myers, unit testing didefinisikan sebagai proses pengujian yang berfokus pada pengujian unit

terkecil dari perangkat lunak, seperti fungsi, metode, atau kelas, secara terpisah untuk memastikan bahwa mereka berfungsi sesuai dengan spesifikasi yang telah ditetapkan.

Langkah/Proses Unit Testing:

Berikut adalah langkah-langkah singkat dalam melakukan unit testing berdasarkan buku "The Art of Software Testing" (edisi ketiga):

1. **Persiapan**: Siapkan alat dan framework testing yang diperlukan.
2. **Identifikasi Unit**: Tentukan unit atau komponen yang akan diuji (fungsi, metode, atau kelas).
3. **Menentukan Test Cases**: Buat test cases untuk berbagai skenario, termasuk kondisi normal dan edge cases.
4. **Menulis Unit Test**: Tulis kode unit test menggunakan pendekatan Arrange-Act-Assert (AAA).
5. **Eksekusi Unit Test**: Jalankan unit test dan amati hasilnya; pastikan semua test cases berhasil.
6. **Analisis dan Debugging**: Jika ada kegagalan, analisis penyebabnya dan lakukan perbaikan.
7. **Memperbarui dan Mengeksekusi Ulang**: Perbarui unit test jika perlu dan jalankan kembali untuk memastikan perbaikan berhasil.
8. **Automasi Pengujian**: Integrasikan unit testing ke dalam alur kerja pengembangan secara otomatis.
9. **Mengukur Cakupan Pengujian**: Gunakan alat untuk mengukur cakupan pengujian dan pastikan banyak kode telah diuji.

Contoh Kasus:

```
def tambah(a, b):
    return a + b
```

1. Persiapan Lingkungan:
    - Pastikan Python dan pustaka `unittest` terinstal di lingkungan pengembangan.
2. Identifikasi Unit:
    - Unit yang akan diuji adalah fungsi `tambah`.
3. Menentukan Test Cases:
    - Test cases yang akan dibuat:Menjumlahkan dua angka positif.

4. Menulis Unit Test:
    - Tulis kode unit test menggunakan `unittest`:

```
import unittest

class TestTambah(unittest.TestCase):

    def test_tambah_positif(self):
        self.assertEqual(tambah(1, 2), 3)

    def test_tambah_negatif(self):
        self.assertEqual(tambah(-1, -2), -3)

    def test_tambah_nol(self):
        self.assertEqual(tambah(0, 0), 0)

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

5. Eksekusi Unit Test:
    - Jalankan unit test dengan perintah python -m unittest nama_file.py dan periksa hasilnya.
6. Analisis dan Debugging:
    - Jika ada test case yang gagal, analisis hasilnya dan perbaiki fungsi tambah jika diperlukan.
7. Memperbarui dan Mengeksekusi Ulang:
    - Setelah memperbaiki kesalahan, jalankan kembali unit test untuk memastikan semua test case lulus.
8. Automasi Pengujian:
    - Integrasikan unit testing ke dalam alur kerja pengembangan dengan menggunakan CI/CD tools untuk menjalankan tes secara otomatis setiap kali ada perubahan kode.
9. Mengukur Cakupan Pengujian:
    - Gunakan alat seperti coverage.py untuk mengukur seberapa banyak kode yang telah diuji oleh unit tests.

D. Integration Testing

Dalam *Software Engineering: A Practitioner's Approach* oleh Roger S. Pressmani, *integration testing* didefinisikan sebagai proses menguji interaksi antar modul atau komponen yang telah diuji secara terpisah (unit testing) untuk memastikan bahwa mereka bekerja bersama dengan baik.

Langkah/Proses Integration Testing:

1. **Identifikasi Komponen**: Tentukan modul yang akan diintegrasikan.
2. **Pilih Strategi Integrasi**: Tentukan apakah akan menggunakan pendekatan *big bang* atau *incremental*.
3. **Persiapkan Kasus Uji**: Buat skenario pengujian untuk interaksi antar modul.
4. **Lakukan Integrasi**: Gabungkan modul sesuai strategi yang dipilih.
5. **Eksekusi Pengujian**: Jalankan kasus uji dan amati hasilnya.
6. **Analisis Hasil**: Tinjau hasil untuk menemukan masalah dalam interaksi.
7. **Perbaiki Kode jika Diperlukan**: Lakukan perbaikan pada modul dan ulangi pengujian.
8. **Dokumentasi**: Catat hasil pengujian dan masalah yang ditemukan.

Contoh Kasus:

Contoh kasus *integration testing* berdasarkan modul dalam aplikasi e-commerce yang mengelola proses pemesanan dan pembayaran.

Modul yang Terlibat:

1. **Modul Pemesanan**: Mengelola data pemesanan, termasuk produk yang dipesan dan informasi pengguna.
2. **Modul Pembayaran**: Mengelola proses pembayaran, termasuk validasi metode pembayaran dan transaksi.

Langkah-langkah dan Kasus Uji dari Contoh Kasus

1. **Identifikasi Komponen yang Akan Diintegrasikan**:
    - Modul Pemesanan dan Modul Pembayaran.
2. **Pilih Strategi Integrasi**:

    - **Incremental**: Mengintegrasikan modul satu per satu untuk pengujian.
3. **Persiapkan Kasus Uji**:
    - **Kasus Uji 1**: Pemesanan Berhasil dengan Pembayaran Valid
        - **Input**: Pengguna melakukan pemesanan produk dengan detail valid dan memilih metode pembayaran yang valid (mis. kartu kredit).
        - **Ekspektasi**: Transaksi berhasil dan pemesanan tercatat di sistem.
    - **Kasus Uji 2**: Pemesanan Gagal karena Pembayaran Tidak Valid
        - **Input**: Pengguna melakukan pemesanan tetapi memasukkan detail pembayaran yang salah (mis. nomor kartu kredit tidak valid).
        - **Ekspektasi**: Transaksi gagal dan pengguna menerima pesan kesalahan.
    - **Kasus Uji 3**: Pemesanan dengan Saldo Tidak Cukup
        - **Input**: Pengguna mencoba memesan produk dengan saldo akun yang tidak mencukupi untuk menyelesaikan pembayaran.
        - **Ekspektasi**: Transaksi tidak dapat diproses dan pengguna menerima informasi bahwa saldo tidak mencukupi.
    - **Kasus Uji 4**: Pemesanan dengan Diskon
        - **Input**: Pengguna melakukan pemesanan produk yang memenuhi syarat diskon.
        - **Ekspektasi**: Diskon diterapkan dengan benar pada total pembayaran dan transaksi berhasil.
4. **Lakukan Integrasi**:
    - Gabungkan Modul Pemesanan dengan Modul Pembayaran.
5. **Eksekusi Pengujian**:
    - Jalankan semua kasus uji yang telah disiapkan dan amati hasilnya.
6. **Analisis Hasil**:
    - Tinjau apakah interaksi antara Modul Pemesanan dan Modul Pembayaran berfungsi sesuai dengan ekspektasi.
7. **Perbaiki Kode jika Diperlukan**:
    - Jika ada kasus yang gagal, lakukan perbaikan pada kode di modul terkait.
8. **Dokumentasi**:
    - Catat hasil pengujian, termasuk kasus uji yang berhasil dan gagal, serta masalah yang ditemukan.

E. System Testing

System testing adalah tahap pengujian dalam siklus pengembangan perangkat lunak yang bertujuan untuk mengevaluasi sistem secara keseluruhan. Ini dilakukan setelah tahap integration testing dan sebelum acceptance testing. Tujuan utama dari system testing adalah untuk memastikan bahwa semua komponen perangkat lunak berfungsi dengan baik secara bersama-sama dan memenuhi spesifikasi yang ditetapkan, baik dari segi fungsional maupun non-fungsional.

Langkah/Proses System Testing:

1. Perencanaan Pengujian:
    - Tentukan ruang lingkup, tujuan, sumber daya, dan pendekatan pengujian. Misalnya, rencanakan untuk menguji aplikasi e-commerce dengan fokus pada fungsionalitas, performa, dan keamanan.
2. Desain Pengujian:
    - Buat test cases berdasarkan spesifikasi dan persyaratan sistem. Contohnya, untuk fitur login pengguna, desain test cases untuk skenario kredensial yang valid dan tidak valid.
3. Eksekusi Pengujian:
    - Jalankan test cases yang telah dibuat dan catat hasilnya. Pastikan untuk menguji interaksi antar komponen serta respons sistem terhadap berbagai skenario.
4. Analisis Hasil:
    - Tinjau hasil pengujian untuk mengidentifikasi cacat atau masalah dalam sistem. Jika ada kegagalan, analisis penyebabnya.
5. Perbaikan dan Uji Ulang:
    - Lakukan perbaikan berdasarkan temuan dari analisis hasil dan jalankan kembali pengujian untuk memastikan bahwa masalah telah diperbaiki.
6. Dokumentasi:
    - Catat semua hasil pengujian, termasuk test cases yang berhasil dan gagal, serta tindakan perbaikan yang dilakukan.

Contoh Kasus:

Kita ambil contoh aplikasi e-commerce yang memiliki fitur login pengguna.

1. Perencanaan Pengujian:
    - Rencanakan pengujian untuk memastikan aplikasi dapat menangani login pengguna dengan benar.
2. Desain Pengujian:
    - Buat test cases:Test Case 1: Login dengan kredensial yang valid.
3. Eksekusi Pengujian:
    - Jalankan setiap test case di atas dan catat hasilnya.
4. Analisis Hasil:
    - Jika Test Case 2 gagal (misalnya, sistem masih membolehkan akses), analisis logika di balik proses login.
5. Perbaikan dan Uji Ulang:
    - Perbaiki logika otentikasi dan jalankan kembali semua test case untuk memastikan tidak ada masalah baru muncul.
6. Dokumentasi:
    - Buat laporan pengujian yang mencakup semua test case yang dijalankan beserta hasilnya, serta langkah-langkah perbaikan yang diambil.\

F. Regression Testing

Pressman menjelaskan bahwa *regression testing* bertujuan untuk memastikan bahwa fitur yang telah diuji sebelumnya tetap berfungsi dengan baik setelah kode diubah. Pengujian ini biasanya dilakukan setelah pengembangan atau perbaikan untuk memverifikasi bahwa tidak ada efek samping dari perubahan tersebut. Pressman juga menekankan pentingnya memilih kasus uji yang tepat untuk memastikan cakupan pengujian yang memadai.

Langkah/Proses Regression Testing:

1. **Identifikasi Perubahan**: Tentukan bagian kode yang telah diubah atau diperbaiki.
2. **Pilih Kasus Uji**: Pilih kasus uji yang relevan dari pengujian sebelumnya yang mungkin terpengaruh oleh perubahan.
3. **Persiapkan Lingkungan Pengujian**: Siapkan lingkungan yang sama seperti sebelumnya untuk menjalankan pengujian.

4. **Eksekusi Kasus Uji**: Jalankan semua kasus uji yang dipilih dan amati hasilnya.
5. **Analisis Hasil**: Tinjau hasil untuk mengidentifikasi masalah baru yang mungkin timbul akibat perubahan.
6. **Dokumentasi**: Catat hasil pengujian, termasuk kasus yang gagal dan masalah yang ditemukan.
7. **Perbaiki Jika Diperlukan**: Jika ada cacat baru yang ditemukan, lakukan perbaikan dan ulangi pengujian regresi.

Contoh Kasus:

Skenario: Setelah memperbaiki bug pada proses pemesanan, pengujian regresi dilakukan untuk memastikan bahwa fungsionalitas lainnya tetap utuh.

1. **Identifikasi Perubahan**:
    - Bug diperbaiki di modul pemesanan yang menyebabkan kesalahan saat menambahkan produk ke keranjang.
2. **Pilih Kasus Uji**:
    - Kasus Uji 1: Proses Pemesanan (memastikan produk dapat ditambahkan ke keranjang).
    - Kasus Uji 2: Proses Pembayaran (memastikan transaksi berjalan setelah pemesanan).
    - Kasus Uji 3: Lihat riwayat pesanan (memastikan pengguna dapat melihat pesanan mereka sebelumnya).
3. **Persiapkan Lingkungan Pengujian**:
    - Siapkan server pengujian dengan versi terbaru dari aplikasi.
4. **Eksekusi Kasus Uji**:
    - Jalankan semua kasus uji yang telah dipilih.
5. **Analisis Hasil**:
    - Tinjau hasil dan catat jika ada yang gagal.
6. **Dokumentasi**:
    - Buat laporan mengenai hasil pengujian, termasuk bug baru yang mungkin muncul.
7. **Perbaiki Jika Diperlukan**:
    - Jika ada masalah baru, perbaiki dan lakukan pengujian ulang.

2. Lakukan Evaluasi Hasil Testing dan Identifikasi Penyebab Masalah dari suatu kasus!

Berikut adalah contoh Evaluasi hasil testing dari suatu kasus spesifik. Dalam kasus ini, kita akan mengevaluasi hasil pengujian untuk fitur "Login" pada aplikasi web.

Kasus: Pengujian Fitur Login pada Aplikasi Web

1. Pengumpulan Hasil Testing

Setelah dilakukan pengujian terhadap fitur login, sebagian besar test case menunjukkan hasil yang sesuai dengan ekspektasi. Test case yang mencakup login dengan username dan password yang valid, username yang tidak terdaftar, serta password yang salah semuanya berhasil. Validasi untuk kasus username kosong dan penggunaan karakter khusus dalam username juga berfungsi dengan baik. Namun, ditemukan satu masalah pada test case ketika pengguna memasukkan username yang panjangnya lebih dari 50 karakter. Pada test case tersebut, aplikasi mengalami **Internal Server Error**, yang seharusnya menampilkan pesan error bahwa "Username terlalu panjang."

2. Analisis Hasil Testing

Dari hasil pengujian di atas, kita melihat bahwa sebagian besar test case berhasil (status **Pass**), namun ada satu test case yang gagal (status **Fail**) pada test case nomor 5, yaitu saat pengguna memasukkan username yang melebihi 50 karakter.

Detail Hasil:

Test case yang berhasil: Test case 1-4 dan 6 memberikan hasil yang sesuai dengan ekspektasi. Fitur validasi seperti username yang tidak terdaftar, password yang salah, serta pengecekan karakter khusus berfungsi dengan baik.

Test case yang gagal: Test case 5 memberikan hasil yang tidak sesuai. Aplikasi mengalami **Internal Server Error** ketika pengguna memasukkan username yang lebih dari 50 karakter, sementara output yang diharapkan adalah **pesan error** "Username terlalu panjang."

3. Reproduksi Masalah

Test Case 5: Login dengan username lebih dari 50 karakter.

Langkah yang diambil:

1. Buka halaman login.

2. Masukkan username dengan panjang 51 karakter atau lebih.
3. Masukkan password yang valid.
4. Klik tombol login.

Hasil: Aplikasi menghasilkan "Internal Server Error" setiap kali test case ini dijalankan. Masalah ini terjadi secara konsisten.

4. Identifikasi Penyebab Masalah (Root Cause Analysis)

Internal Server Error terjadi saat aplikasi menerima input username yang panjang, yang berarti ada kesalahan dalam menangani input yang melampaui batas panjang yang diizinkan.

Setelah memeriksa kode, ditemukan bahwa:

1. Database hanya mengizinkan kolom username dengan maksimal 50 karakter.
2. Namun, validasi di sisi frontend (antarmuka pengguna) tidak membatasi panjang input username.
3. Ketika username yang panjang dikirimkan ke server, server tidak dapat memprosesnya dan menghasilkan kesalahan.

5. Laporan Masalah

Masalah: Aplikasi gagal menangani input username dengan panjang lebih dari 50 karakter, menyebabkan "Internal Server Error" alih-alih menampilkan pesan validasi yang sesuai.

Penyebab Utama: Kurangnya validasi panjang username di sisi frontend, serta kegagalan server dalam memproses input yang melampaui batasan yang diterapkan di database.

Dampak: Pengguna akan mengalami crash ketika mencoba memasukkan username panjang, yang menyebabkan penurunan pengalaman pengguna dan risiko keamanan (jika ada eksploitasi terhadap kesalahan ini).

6. Saran Perbaikan

Frontend: Tambahkan validasi input di sisi frontend untuk membatasi panjang username menjadi maksimal 50 karakter sebelum mengirimkannya ke server.

Backend: Pastikan ada penanganan error yang lebih baik di backend, sehingga jika ada input yang melampaui batas yang diizinkan, aplikasi dapat memberikan pesan error yang jelas kepada pengguna tanpa menghasilkan "Internal Server Error."

Pengujian Tambahan: Setelah perbaikan dilakukan, lakukan pengujian ulang untuk memastikan bahwa masalah telah teratasi. Tambahkan test case baru untuk memverifikasi bahwa aplikasi menangani berbagai panjang username dengan benar.

Berikut ini adalah analisis **5 Whys** untuk mengidentifikasi penyebab masalah pada pengujian fitur login:

1. Mengapa aplikasi mengalami "Internal Server Error" ketika username lebih dari 50 karakter dimasukkan?

: Karena server tidak dapat menangani input username yang melebihi batas panjang yang diizinkan oleh database.

2. Mengapa server tidak dapat memproses input username yang lebih panjang dari 50 karakter?

: Karena tidak ada mekanisme validasi di backend untuk menangani panjang username sebelum mengirimkannya ke database.

3. Mengapa backend tidak memvalidasi panjang username?

: Karena proses validasi panjang username tidak dilakukan di sisi frontend, sehingga input yang melebihi batas panjang dikirim langsung ke backend.

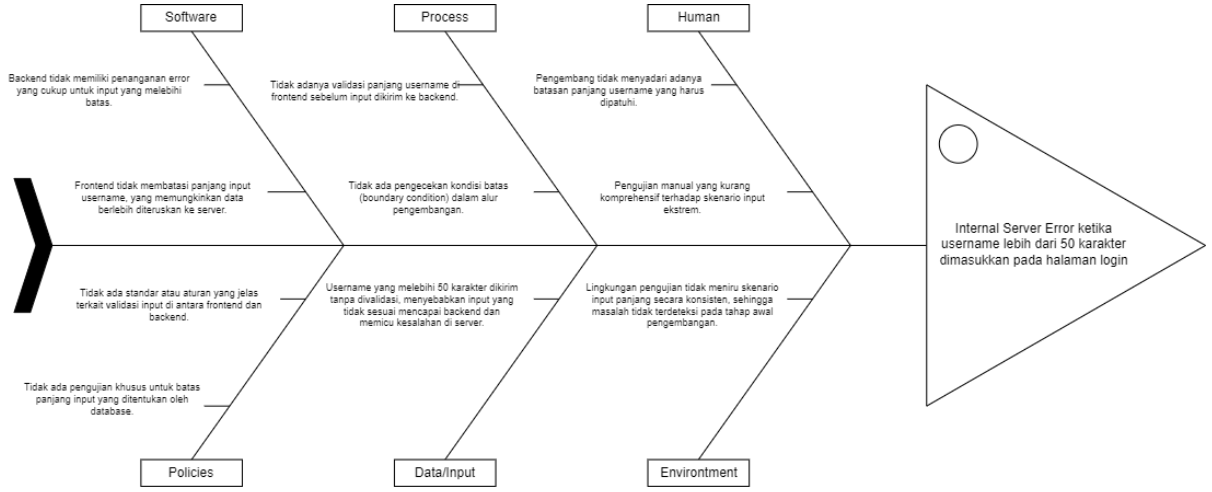
4. Mengapa frontend tidak memvalidasi panjang username?

: Karena pengembang belum menambahkan batasan panjang input username di frontend sesuai dengan batas yang diterapkan di database.

5. Mengapa pengembang tidak menambahkan batasan panjang input di frontend?

: Karena saat perancangan validasi, pengembang mungkin tidak menyadari atau lupa bahwa ada batasan panjang di database yang harus dijaga oleh frontend juga.

Berikut adalah analisis Fishbone (Diagram Tulang Ikan) untuk mengidentifikasi penyebab utama dari masalah **"Internal Server Error"** saat pengguna memasukkan username lebih dari 50 karakter pada pengujian fitur login aplikasi web.



Referensi

Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). John Wiley &Sons

Pressman, R. S., & Maxim, B. R. (2015). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Education.