**QUALITY ASSURANCE TECHNICAL TEST**

**Studi Kasus**

Terdapat sebuah booking dengan data tersimpan di table database sebagai berikut:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | Booking\_id | venue\_id | User\_id | date | Start\_time | end\_time | price |
| 1001 | BK/000001 | 15 | 12 | 2022-12-10 | 09:00:00 | 11:00:00 | 1200000 |
| 1005 | BK/000005 | 15 | 12 | 2022-12-10 | 09:00:00 | 11:00:00 | 1000000 |

Ternyata setelah diperiksa, harga yang tersimpan di table booking tidak sesuai dengan harga yang seharusnya untuk tanggal dan jam tersebut serta terjadi double book (dua booking dengan tanggal dan waktu yang sama). Berikut data jadwal dan harga yang tersimpan:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | venue\_id | date | start\_time | end\_time | price |
| 11 | 15 | 2022-12-10 | 07:00:00 | 09:00:00 | 800000 |
| 12 | 15 | 2022-12-10 | 09:00:00 | 11:00:00 | 1000000 |
| 13 | 15 | 2022-12-10 | 11:00:00 | 13:00:00 | 1200000 |

Sebagai seorang QA tester, tugas kamu adalah buatlah sebuah script automation test serta dokumen test scenario sederhana untuk memastikan agar permasalahan diatas dapat terdeteksi lebih awal.

**Jawaban**

Untuk Automation Script, saya menggunakan Bahasa Pemrograman Python. Pertama saya memasukkan import unittest, datetime dan collections. Setelah dibuat data importnya, masukkan simulasi database berupa list nerdasarkan soal case diatas. Setelah dibuat simulasi database, buat koding automation script tersebut. Untuk automation test, saya melakukan automation test berupa test scenario harga yang tidak sesuai dan test scenario booking yang bertumpukan / bersamaan atau juga disebut double booking. Untuk kode automation script file python saya masukkan ke Github. Berikut ini adalah hasil koding python yang saya kerjakan:

import unittest

from datetime import datetime

from collections import defaultdict

# Simulasi Database

# Data Booking

database\_data = [

    {"id": 1001, "Booking\_id": "BK/000001", "venue\_id": 15, "User\_id": 12, "date": "2022-12-10", "Start\_time": "09:00:00", "end\_time": "11:00:00", "price": 1200000},

    {"id": 1005, "Booking\_id": "BK/000005", "venue\_id": 15, "User\_id": 12, "date": "2022-12-10", "Start\_time": "09:00:00", "end\_time": "11:00:00", "price": 1000000}

]

# Database jadwal dan harga yang disimpan

schedule\_data = [

    {"id": 11, "venue\_id": 15, "date": "2022-12-10", "start\_time": "07:00:00", "end\_time": "09:00:00", "price": 800000},

    {"id": 12, "venue\_id": 15, "date": "2022-12-10", "start\_time": "09:00:00", "end\_time": "11:00:00", "price": 1000000},

    {"id": 13, "venue\_id": 15, "date": "2022-12-10", "start\_time": "11:00:00", "end\_time": "13:00:00", "price": 1200000}

]

class TestBookingPriceAndOverlap(unittest.TestCase):

    # Test konsisten Harga

    def test\_price\_consistency(self):

        price\_mismatch = []

        for booking in database\_data:

            booking\_date = booking["date"]

            start\_time = booking["Start\_time"]

            for schedule in schedule\_data:

                if schedule["date"] == booking\_date and schedule["start\_time"] == start\_time:

                    if schedule["price"] != booking["price"]:

                        price\_mismatch.append(booking)

                    break

        self.assertEqual(len(price\_mismatch), 0, f"Price mismatch detected: {price\_mismatch}")

    # Test Booking Bertumpukan atau Bersamaan

    def test\_overlap\_detection(self):

        overlap\_detected = defaultdict(list)

        for i, booking1 in enumerate(database\_data):

            for j, booking2 in enumerate(database\_data):

                if i != j and booking1["date"] == booking2["date"]:

                    start\_time1 = datetime.strptime(booking1["Start\_time"], "%H:%M:%S")

                    end\_time1 = datetime.strptime(booking1["end\_time"], "%H:%M:%S")

                    start\_time2 = datetime.strptime(booking2["Start\_time"], "%H:%M:%S")

                    end\_time2 = datetime.strptime(booking2["end\_time"], "%H:%M:%S")

                    if start\_time1 < end\_time2 and end\_time1 > start\_time2:

                        overlap\_detected[booking1["date"]].append((booking1["Booking\_id"], booking2["Booking\_id"]))

        overlap\_messages = []

        for date, overlaps in overlap\_detected.items():

            overlap\_messages.append(f"Date: {date}, Overlaps: {overlaps}")

        self.assertEqual(len(overlap\_messages), 0, f"Overlaps detected: {overlap\_messages}")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Selanjutnya adalah Dokumen Test Scenario Sederhana berdasarkan Hasil Automation Testing

**Test Scenario 1**: Uji Konsistensi Harga

Test Objective: Untuk memverifikasi bahwa harga yang disimpan di tabel pemesanan cocok dengan harga yang sesuai di tabel jadwal untuk tanggal dan waktu yang sama.

Test Steps:

1. Mengambil data pemesanan dari database.
2. Untuk setiap data pemesanan yang masuk, ekstrak tanggal dan waktu mulai.
3. Cari tabel jadwal untuk data yang cocok berdasarkan tanggal dan waktu mulai.
4. Bandingkan harga di data pemesanan dengan harga di database jadwal.
5. Jika harga tidak cocok, catat data pemesanan yang tidak cocok.

Expected Result: Data pemesanan tidak boleh memiliki harga yang tidak konsisten dibandingkan dengan data jadwal.

**Test Scenario 2**: Uji Deteksi Tumpang Tindih

Test Objective: Untuk mengidentifikasi jika ada kasus pemesanan ganda (pemesanan yang tumpang tindih) untuk tanggal dan waktu yang sama.

Test Step:

1. Mengambil data pemesanan dari database.
2. Untuk setiap pasang data pemesanan, bandingkan tanggalnya.
3. Jika tanggalnya cocok, uraikan waktu mulai dan berakhirnya kedua pemesanan.
4. Periksa apakah rentang waktu kedua Pemesanan dibooking secara bersamaan.
5. Jika terdeteksi booking diwaktu yang sama atau double booking, catat ID pemesanan dari pemesanan yang ke double booking.

Expected Result: Seharusnya tidak ada kasus pemesanan yang dibooking secara bersamaan atau double booking yang terekam.