

**LAPORAN TUGAS BESAR CLO 2**  
**CUI AUTO VENDING Using C# with Visual Studio 2022**



Dibuat Oleh:

EVI FITRIYA

1201222005

**SOFTWARE ENGINEERING**

**TELKOM UNIVERSITY**

**2025**

# DAFTAR ISI

DAFTAR ISI .....	2
DESKRIPSI SINGKAT .....	3
DAFTAR ANGGOTA KELOMPOK.....	3
GITHUB .....	4
IMLEMENTASI DESIGN BY KONTRAK .....	5
HASIL UNIT TESTING .....	7
HASIL PERFORMANCE TESTING .....	11

## DESKRIPSI SINGKAT

Auto Vending adalah aplikasi smart vending machine berbasis self-checkout yang memungkinkan pengguna melakukan pemesanan dan pembayaran produk secara mandiri tanpa bantuan operator. Aplikasi ini dikembangkan menggunakan C# dalam Visual Studio 2022 dan dirancang untuk diintegrasikan langsung dengan perangkat vending fisik. Dengan antarmuka yang intuitif dan pengalaman transaksi yang cepat, Auto Vending bertujuan untuk meningkatkan efisiensi layanan penjualan otomatis, serta memberikan fleksibilitas tinggi dalam pengelolaan produk dan konfigurasi sistem.

Fitur-fitur yang tersedia:

- Perubahan Status
- Pemilihan Produk
- Ganti Bahasa
- Manajemen Produk
- Jam operasional
- Convert Mata Uang
- Checkout
- Transaksi

## DAFTAR ANGGOTA KELOMPOK

- |                                  |            |
|----------------------------------|------------|
| 1. Zidan Irfan Zaky              | 1201220003 |
| 2. Farhan Nugraha Sasongko Putra | 1201220449 |
| 3. Radinka Putra Rahadian        | 1201220020 |
| 4. Giovan Deo Pratama            | 1201220450 |
| 5. Evi Fitriya                   | 1201222005 |

Dengan Pembagian Tugas sesuai arahan dimana 1 teknik konstruksi dipegang oleh maksimal 2 orang:

Nama Anggota	Zidan Irfan Zaky	Farhan Nugraha Sasongko	Radnka Putra Rahadian	Giovan Deo Pratama	Evi Fitriya	Total Teknik Dipegang
Nama Teknik konstruksi						
Automata		✓			✓	2
Table Driven		✓	✓			2
Runtime	✓			✓		2
Code Reuse			✓		✓	2
Generics	✓			✓		2
Total Peserta Memegang	2	2	2	2	2	

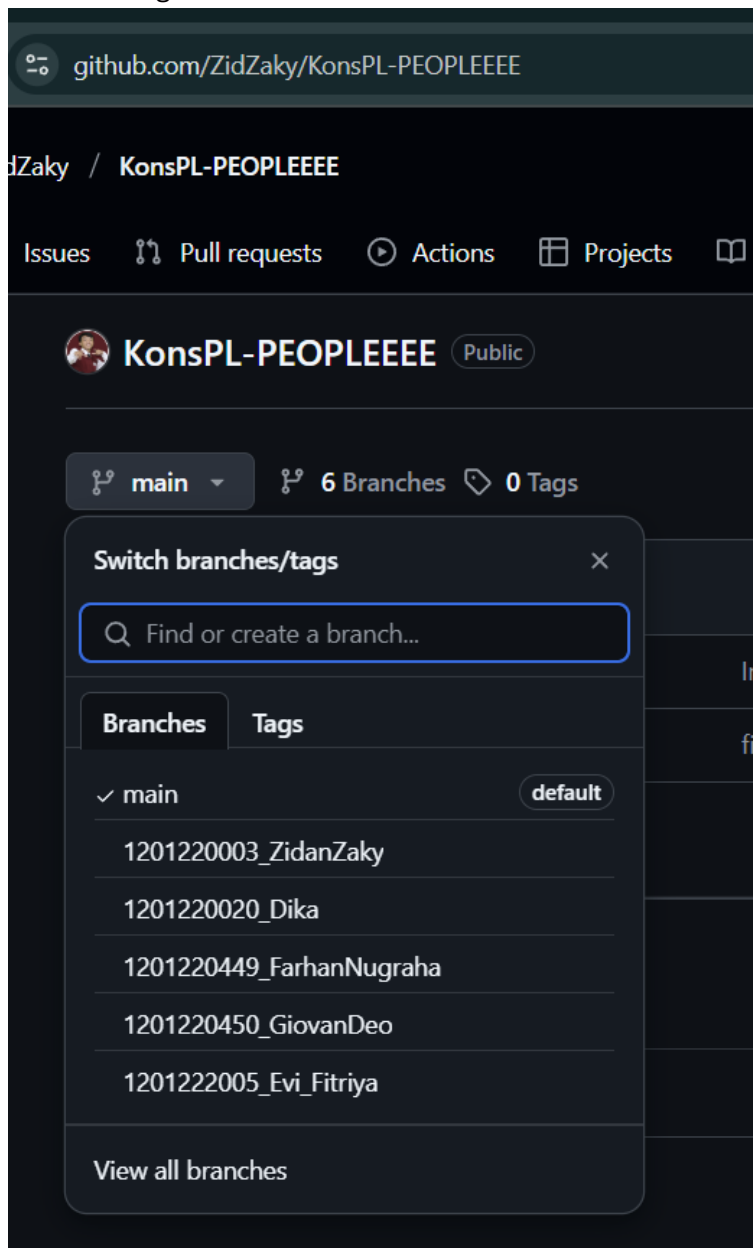
# GITHUB

Repositori ini menggunakan sistem version control GitHub untuk mendokumentasikan proses konstruksi proyek secara kolaboratif. Terdapat *branch* pribadi yang digunakan untuk pengembangan fitur atau modul secara terpisah, sebelum digabungkan ke *branch* utama (*main*) melalui proses *commit* dan *merge*. Pendekatan ini memastikan riwayat perubahan tercatat dengan jelas serta mempermudah kolaborasi dan pengelolaan versi.

Berikut Link Github Kami:

<https://github.com/ZidZaky/KonsPL-PEOPLEEEE.git>

Dengan bukti:



# IMPLEMENTASI DESIGN BY KONTRAK

```

namespace UTS_Evi
{
    public class CheckTransaction
    {
        public String Idle()...
        public Object[][] Product()...

        public List<int> Order(String inputan)...

        public String invalidInput()...

        public int ShowProductBuy(List<int> inputan)...

        public void Payment(List<int> Products)...
    }
}

namespace StatusState
{
    public class statusVending
    {
        public enum Status { Idle, Order, Payment, Done };
        public Status status = Status.Idle;

        public String Get_Current_Status()...
        public String set_Status_Idle()...

        public String set_Status_Order()...

        public String set_Status_Payment()...

        public String set_Status_Done()...
    }
}

```

```

namespace UTS_Evi
{
    public class Program()
    {
        public static void Main()...
    }
}

```

Dari semua fungsi yang saya miliki, hanya fungsi ini yg menerima parameter, meskipun ada fungsi ShowProductBuy() dengan parameter inputan tapi variable ini berasal dari Order(), jadi cukup di cek di fungsi Order() saja.

```

public int ShowProductBuy(List<int> inputan)
{
    Console.WriteLine("Product yang dibeli:");
}

```

Ada juga fungsi Payment() yang menerima parameter products tapi ini juga sama-sama berasal dari Order jadi saya rasa ini hanya perlu di validasi sekali di Order() saja.

```

public void Payment(List<int> Products)
{
    ShowProductBuy(Products);
    Console.WriteLine("=====");
    Console.WriteLine("|                               ORIS PAYME

```

Berikut implementasi kodenya:

```

5 references | 0/1 passing
public List<int> Order(String inputan)
{
    Debug.Assert(!string.IsNullOrEmpty(inputan), "Inputan tidak boleh kosong");
    Debug.Assert((inputan!="0"), "Inputan tidak boleh 0");
}

```

Dimana fungsi `string.IsNullOrEmpty()` jika inputan dari berisi null/""/"\t"/"\n" dll akan bernilai true, dan agar debugnya bekerja harus dalam mode false maka dari itu kita beri ! sebagai tanda not, kemudian ketika nilainya false akan memunculkan "Inputan tidak boleh kosong"

Lalu `inputan!="0"` mengecek apakah isinya "0" jika ya maka akan muncul Inputan tidak boleh 0 di debug.

berikut buktinya:

```

UTS_Evi.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.N
'UTS_Evi.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.N
'UTS_Evi.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.N
---- DEBUG ASSERTION FAILED ----
---- Assert Short Message ----
Inputan tidak boleh kosong
---- Assert Long Message ----

at UTS_Evi.CheckTransaction.Order(String inputan) in C:\APRIBADI\EVI FITRIYA\ITTE
at UTS_Evi.CheckTransaction.UI() in C:\APRIBADI\EVI FITRIYA\ITTELKOM\SEMESTER 6\K
at UTS_Evi.Program.Main() in C:\APRIBADI\EVI FITRIYA\ITTELKOM\SEMESTER 6\KONSTRUK
The program '[34708] UTS_Evi.exe' has exited with code 3221225786 (0xc000013a).

```

Dan akan menghentikan program lalu memberitahu kita kode mana yang terhenti:

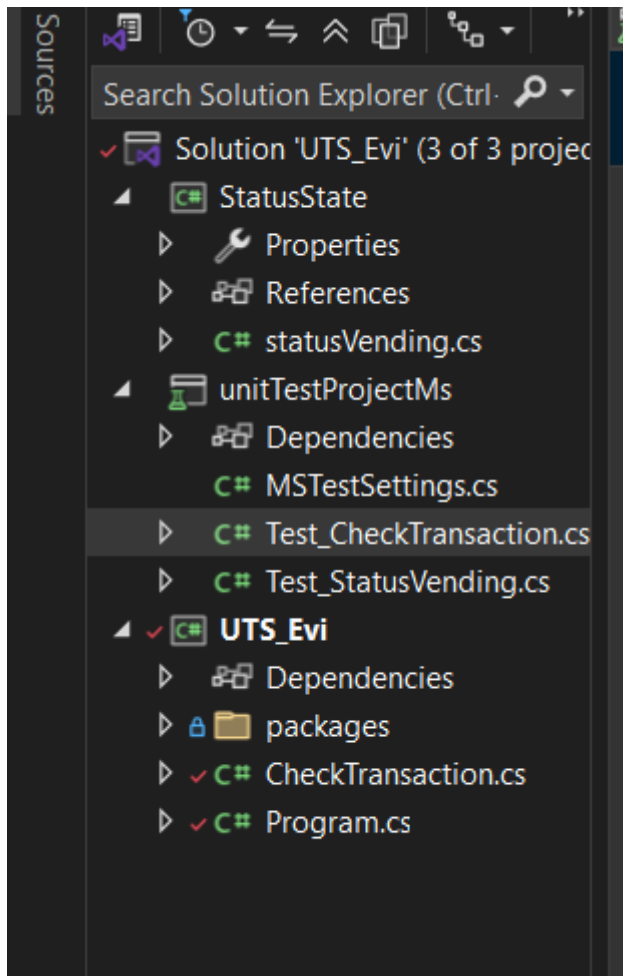
```

5 references | 0/1 passing
public List<int> Order(String inputan)
{
    Debug.Assert(!string.IsNullOrEmpty(inputan), "Inputan tidak boleh kosong");
    Debug.Assert((inputan!="0"), "Inputan tidak boleh 0");

    List<int> Product = new List<int>();
}

```

## HASIL UNIT TESTING



Terdapat 2 Class yang saya test yakni statusVending dan CheckTransaction, adapun untuk CheckTransaction sendiri sebagai berikut:

```
0 references
public sealed class Test_CheckTransaction
{
    UTS_Evi.CheckTransaction CT = new UTS_Evi.CheckTransaction();
}
```

Yang paling pertama kita persiapan variable yang mewakili class CheckTransaction(),

```
[TestMethod]
0 references
public void ShowProductBuyTesting()
{
    Assert.AreEqual(35000, CT.ShowProductBuy([1, 2]));
    Assert.AreEqual(0, CT.ShowProductBuy([50]));
    Assert.AreEqual(15000, CT.ShowProductBuy([1, 50]));
}
```

showProductTesting, adalah mengecek apakah hitungan total transaksinya sesuai dengan yang diminta, kita gunakan Assert.AreEqual untuk mengecek ekspektasi dan realitanya sama atau belum.

Selanjutnya kita testing function Product apakah hasil returnnya sama dengan data produk yg sudah di siapkan.

```
[TestMethod]
0 references
public void ReturnProductTesting()
{
    Object[][] cek = new object[][]
    {
        new object[]
        {
            "Lays", "Pringles", "Cheetos", "Taro",
            "Roma", "Khong Guan", "Oops! Cookies",
            "Oreo", "BelVita", "Marie Regal",
            "Garuda", "Mamasuka", "ABC",
            "Cap Matahari", "Tanggo", "Lays",
            "Richeese", "Richeese Roll", "Chiki",
            "Coca-Cola", "Pepsi", "Sprite", "Fanta",
            "Teh Botol Sosro", "Lipton", "Sariwangi",
            "Minute Maid", "Tropicana", "Mizone",
            "Red Bull", "Monster Energy", "Pocari Sweat",
            "Indomilk", "Frisian Flag", "Ovaltine",
            "Nescafe", "Starbucks", "Kopi Kapal Api"
        },
        new object[]
        {
            15000, 20000, 18000, 12000,
            10000, 5000, 25000, 30000,
            25000, 22000, 12000, 15000,
            8000, 7000, 15000, 18000,
            12000, 10000, 13000, 25000,
            15000, 12000, 10000, 13000,
            16000, 18000, 25000, 30000,
            22000, 25000, 30000, 25000,
            22000, 15000, 35000, 25000, 5000, 54000
        }
    };

    Object[][] tes = CT.Product();

    for(int i = 0; i < cek.Length; i++)
    {
        CollectionAssert.AreEqual(tes[i], cek[i]);
    }
}
```

Dengan sama sama menggunakan Assert.AreEqual tp harus diloooping terlebih dahulu agar terbaca sama antara yang diharapkan dengan kenyataannya.

Lalu di classTest status Vending



```
StatusState.statusVending Sv = new StatusState.statusVending();

[TestMethod]
public void Test_Set_Idle()
{
    Assert.AreEqual("Idle", Sv.set_Status_Idle());
    Assert.AreEqual("Idle", Sv.GetCurrent_Status());
}
```

Kita cek apakah berhasil menset status ke idle menggunakan sama sama AreEqual

```
[TestMethod]
public void Test_Get_Current_Status()
{
    Assert.AreEqual("Idle", Sv.GetCurrent_Status());
}

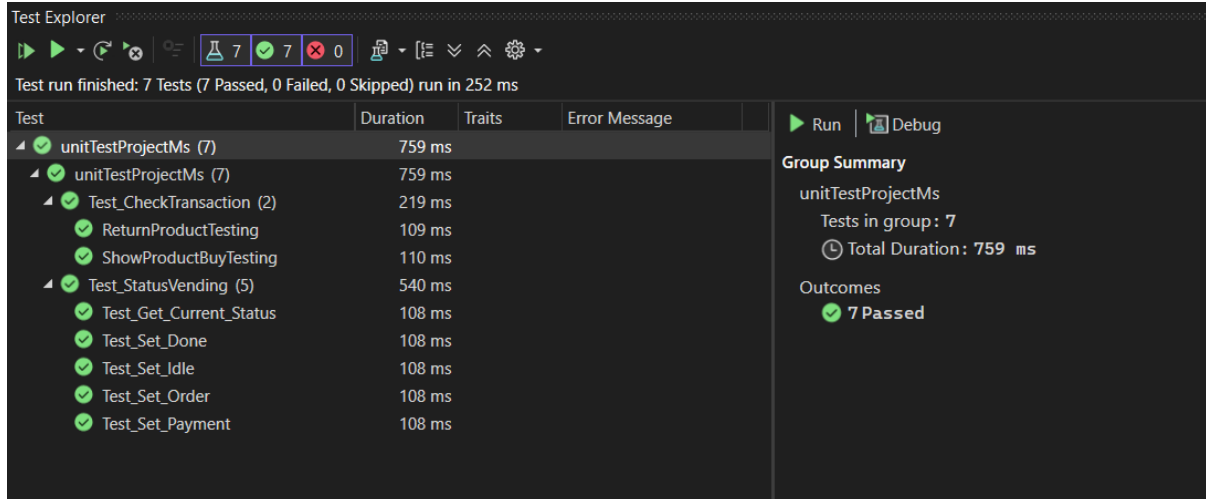
[TestMethod]
public void Test_Set_Order()
{
    Assert.AreEqual("Order", Sv.set_Status_Order());
    Assert.AreEqual("Order", Sv.GetCurrent_Status());
}

[TestMethod]
public void Test_Set_Payment()
{
    Assert.AreEqual("Payment", Sv.set_Status_Payment());
    Assert.AreEqual("Payment", Sv.GetCurrent_Status());
}

[TestMethod]
public void Test_Set_Done()
{
    Assert.AreEqual("Done", Sv.set_Status_Done());
    Assert.AreEqual("Done", Sv.GetCurrent_Status());
}
```

Dan semuanya juga sama.

Berikut hasil unit testnya:



The screenshot shows the Test Explorer window with the following details:

- Test run finished:** 7 Tests (7 Passed, 0 Failed, 0 Skipped) run in 252 ms
- Test Results Table:**

Test	Duration	Traits	Error Message
unitTestProjectMs (7)	759 ms		
unitTestProjectMs (7)	759 ms		
Test_CheckTransaction (2)	219 ms		
ReturnProductTesting	109 ms		
ShowProductBuyTesting	110 ms		
Test_StatusVending (5)	540 ms		
Test_Get_Current_Status	108 ms		
Test_Set_Done	108 ms		
Test_Set_Idle	108 ms		
Test_Set_Order	108 ms		
Test_Set_Payment	108 ms		

**Group Summary**

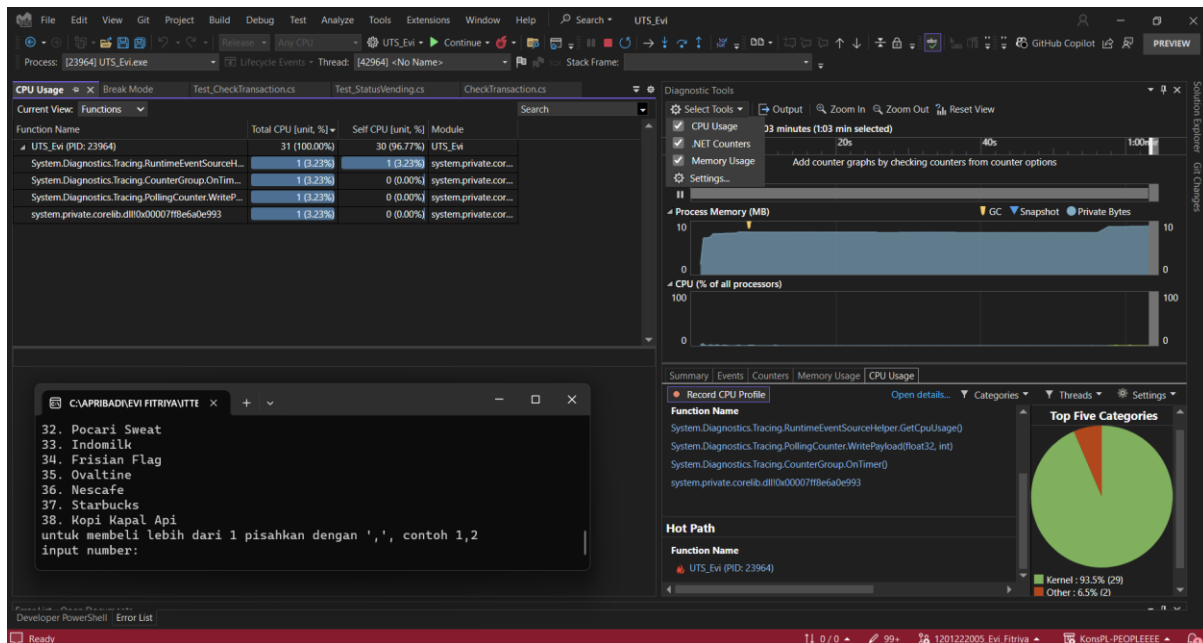
- unitTestProjectMs
- Tests in group: 7
- Total Duration: 759 ms

**Outcomes**

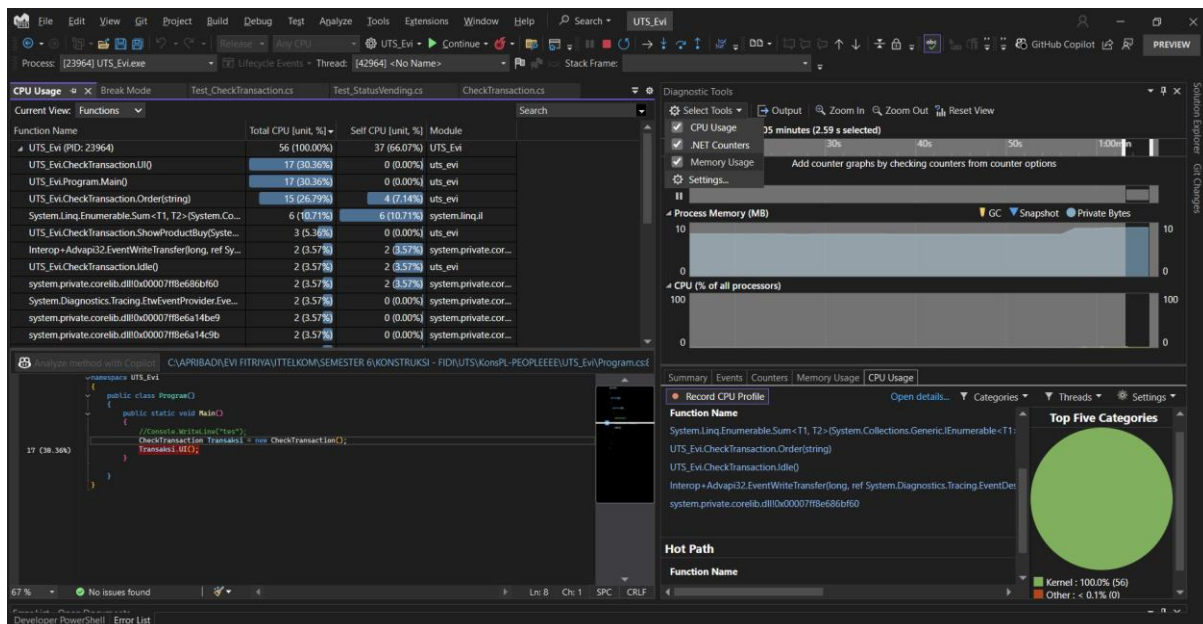
- 7 Passed

Menunjukkan bahwa semua test berhasil atau passed.

# HASIL PERFORMANCE TESTING

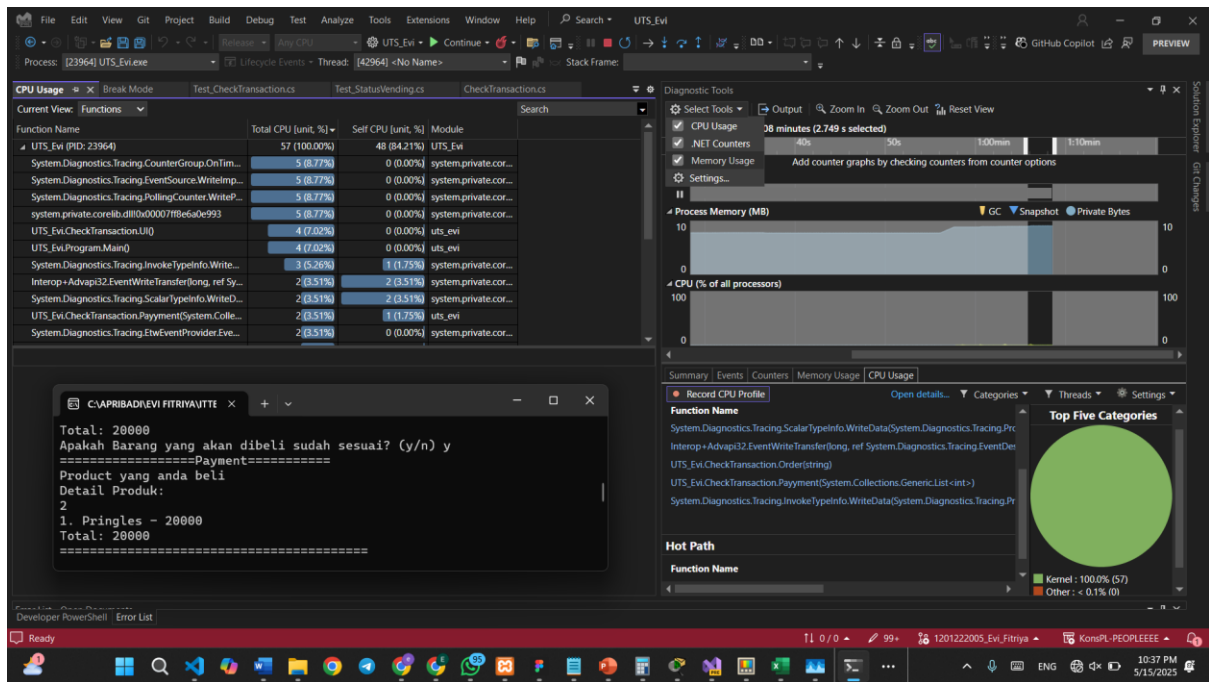


Berdasarkan data diatas ketika baru pertama menjalankan aplikasi, akan menggunakan kernel sebesar 93.5% dan other nya 6.5%, lalu fungsi dengan konsumsi CPU terbanyak berada pada UTS\_Evi

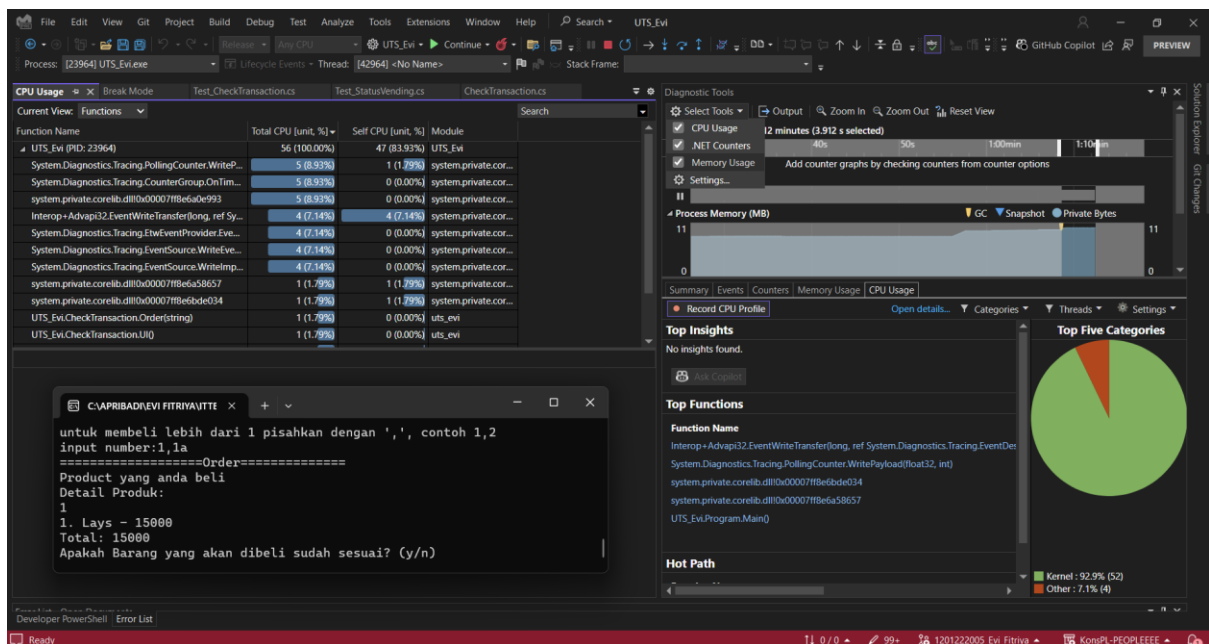


Kemudian disini fungsi UI() menggunakan 30.36% dari total CPU lalu Main() milik class program juga di 30% lebih, kemudian di Order() di 26.79%.

Lalu Ketika run aplikasi di inputan selanjutnya ini menggunakan kernel sebanyak 100%



Pada inputan ini atau perjalanan selanjutnya, tidak ada yang benar-benar menggunakan CPU banyak, tetpi rata namun masih 57, dan kernel digunakan sebanyak 100%



Lalu ini kembali lagi ke pertama aplikasi di mulai atau kondisi idle, jadi kernel digunakan sekitar 90% lebih seperti sebelumnya.