LAPORAN TUGAS BESAR CLO 2

CUI AUTO VENDING Using C# with Visual Studio 2022

Dibuat Oleh:

Farhan Nugraha Sasongko Putra 1201220449

SOFTWARE ENGINEERING

TELKOM UNIVERSITY

2025

DAFTAR ISI

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 maximal 2 orang:

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

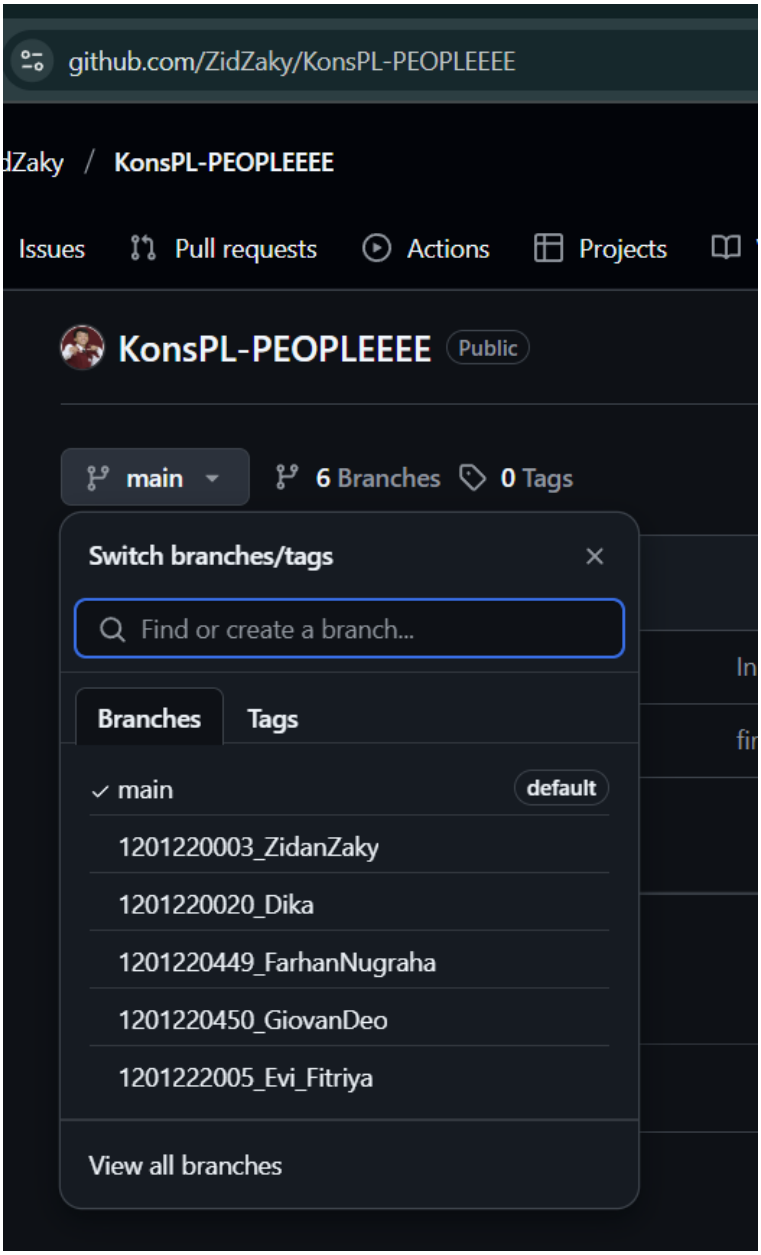
GITHUB

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:



IMLEMENTASI DESIGN BY CONTRACT

Design by Contract (DbC) adalah pendekatan untuk merancang perangkat lunak. DbC menekankan pada pendefinisian kontrak antara penyedia (sebuah kelas atau metode) dan klien (kode lain yang menggunakan kelas atau metode tersebut). Kontrak ini terdiri dari:

1. Prekondisi : Kondisi yang harus benar sebelum sebuah metode dipanggil.
2. Postkondisi : Kondisi yang harus benar setelah sebuah metode dieksekusi.
3. Invariant Kelas : Kondisi yang harus selalu benar untuk semua instance dari sebuah kelas.

Pada kode C# projek yang ada, terdapat beberapa penerapan konsep DbC meskipun C# tidak memiliki sintaksis *built-in* untuk menegakkan kontrak secara eksplisit. Diantaranya:

1. Prekondisi
   Pada kontrak ini, diantaranya diterapkan pada bagian kode berikut:

```
    // Metode untuk memilih produk
    public void SelectProduct(ProductCode code)
    {
        try
        {
            if (currentState != VendingState.Order)
            {
                throw new InvalidOperationException("Tidak dapat memilih
produk saat ini.");
            }

            Product selectedProduct = GetProductByCode(code);

            if (selectedProduct.Stock < 1)
            {
                throw new InvalidOperationException($"Stok produk
{selectedProduct.Name} tidak mencukupi.");
            }
            {…}
    }
```

   Pada metode SelectProduct(),dilakukan dua pendefinisian kondisi sebelum program pada metode tersebut dijalankan. Jika status vending tidak pada status ‘order’, maka tidak dapat melakukan pemesanan produk vending; jika stok produk di bawah angka satu, maka stok dianggap habis dan tidak bisa dibeli.

2. Postkondisi

Pada kontrak ini, diantaranya diterapkan pada bagian kode berikut:

```
    // Metode untuk memilih produk
    public void SelectProduct(ProductCode code)
    {
        try
        {
            {…}
            totalAmount = selectedProduct.Price;
            selectedProductCode = code;
            quantity = 1;

            // Update State
            Console.WriteLine($"1 {selectedProduct.Name} berhasil ditambahkan
ke pesanan.");
            Console.WriteLine($"Total Harga: {totalAmount}");
        }
        {…}
    }
```

Masih pada metode SelectProduct(), setelah pemesanan produk berhasil dan stok produk dikurangi satu (karna sekali transaksi hanya bisa beli satu item), total harga yang perlu dibayar langsung diatur sesuai dengan harga per 1pcs produk tersebut.

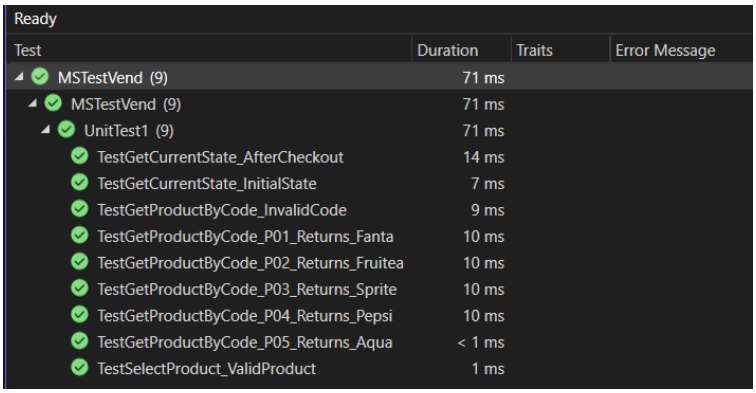
3. Invariant

Pada kontrak ini, diantaranya diterapkan pada bagian kode berikut:

```
    // Metode transisi state
    public void TransitionState(string eventName)
    {
        if (stateTable.ContainsKey(currentState) &&
stateTable[currentState].ContainsKey(eventName))
        {
            currentState = stateTable[currentState][eventName];
        }
        else
        {
            Console.WriteLine($"Invalid transition: {currentState} -
{eventName}");
            currentState = VendingState.Error;
        }
    }
```

Pada metode TransitionState(), diberlakukan kondisi yang harus benar sebelum dan sesudah program metode tersebut dijalankan. Pada konteks ini, currentState serta eventName pada parameter harus selalu sama dan sesuai dengan vendingState yang sudah didefinisikan pada awal program. Jika berbeda, maka currentState otomatis dianggap error.

HASIL UNIT TESTING



Unit Testing dilakukan menggunakan MSTest sebagai tools pengujiannya. Pengecekan dilakukan pada tiap produk yang ada. Dikarenakan pada vending terdapat 5 dummies product, maka tiap produk diujikan kesesuaian nama produk, harga, serta stoknya. Contoh penerapan pada kodenya kurang-lebih sebagai berikut:

```
public void TestGetProductByCode_P01_Returns_Fanta()
{
    // Arrange
    ProductCode code = ProductCode.P01;

    // Act
    Product result = vendingMachine.GetProductByCode(code);

    // Assert
    Assert.AreEqual(ProductName.Fanta, result.Name);
    Assert.AreEqual(10000, result.Price);
    Assert.AreEqual(10, result.Stock);
}
```

Setelah itu, dilakukan pengecekan terhadap kode produk yang valid (P01 – P05) serta yang invalid (selain yang ada pada program) dengan kode sebagai berikut:

```
public void TestSelectProduct_ValidProduct()
{
    // Arrange
    ProductCode code = ProductCode.P01;
    vendingMachine.SelectProduct(code);

    // Act

    // Assert
    Assert.AreEqual(code, vendingMachine.selectedProductCode);
}
```

```
[ExpectedException(typeof(ArgumentException))]
public void TestGetProductByCode_InvalidCode()
```

```
    {
        // Arrange
        ProductCode invalidCode = (ProductCode)99;

        // Act & Assert
        vendingMachine.GetProductByCode(invalidCode);
    }
```

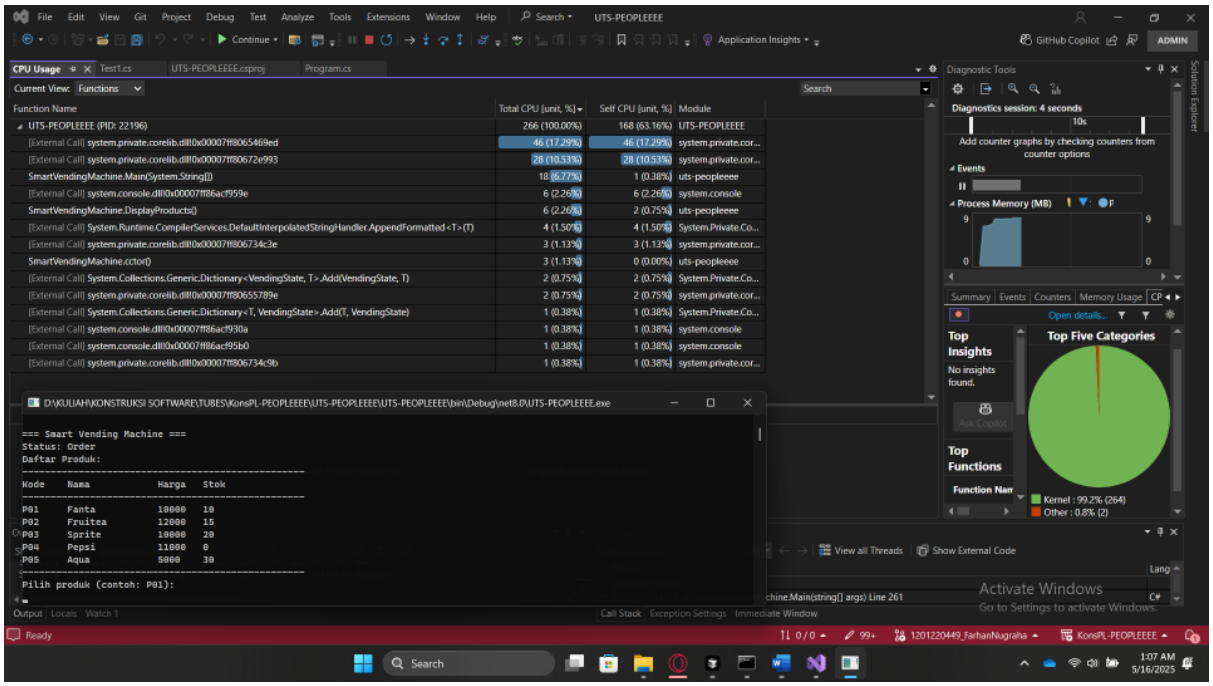
Lalu juga dicek pada vendingState program, dimana pada saat program dijalankan haruslah berada pada state ‘order’ dan bukan state lainnya, dengan kode sebagai berikut:

```
    public void TestGetCurrentState_InitialState()
    {
        // Arrange
        SmartVendingMachine vendingMachine = new SmartVendingMachine();

        // Act
        VendingState currentState = vendingMachine.GetCurrentState();

        // Assert
        Assert.AreEqual(VendingState.Order, currentState);
    }
```

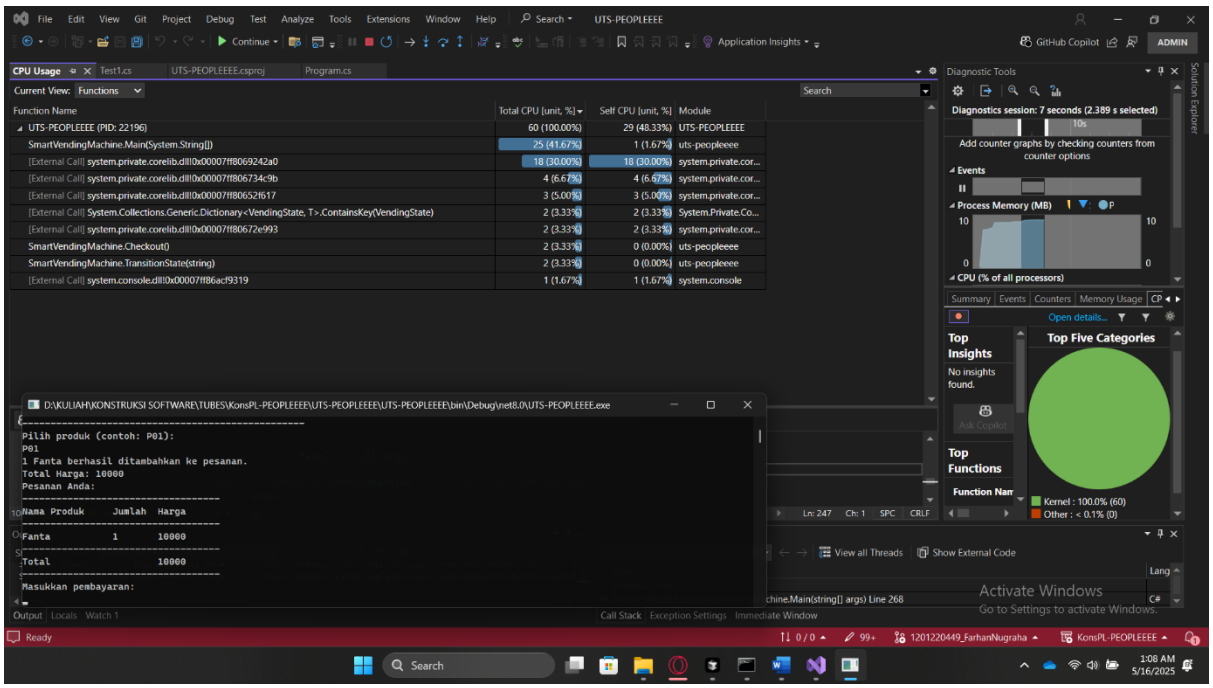
Sebagai perwakilan pengecekan terhadap perubahan state, dilakukan juga pengecekan state saat setelah pelakukan checkout produk, dimana hal ini haruslah berpindah ke state ‘payment’.

```
    public void TestGetCurrentState_AfterCheckout()
    {
        // Arrange
        SmartVendingMachine vendingMachine = new SmartVendingMachine();
        vendingMachine.SelectProduct(ProductCode.P01);
        vendingMachine.Checkout();

        // Act
        VendingState currentState = vendingMachine.GetCurrentState();

        // Assert
        Assert.AreEqual(VendingState.Payment, currentState);
    }
```

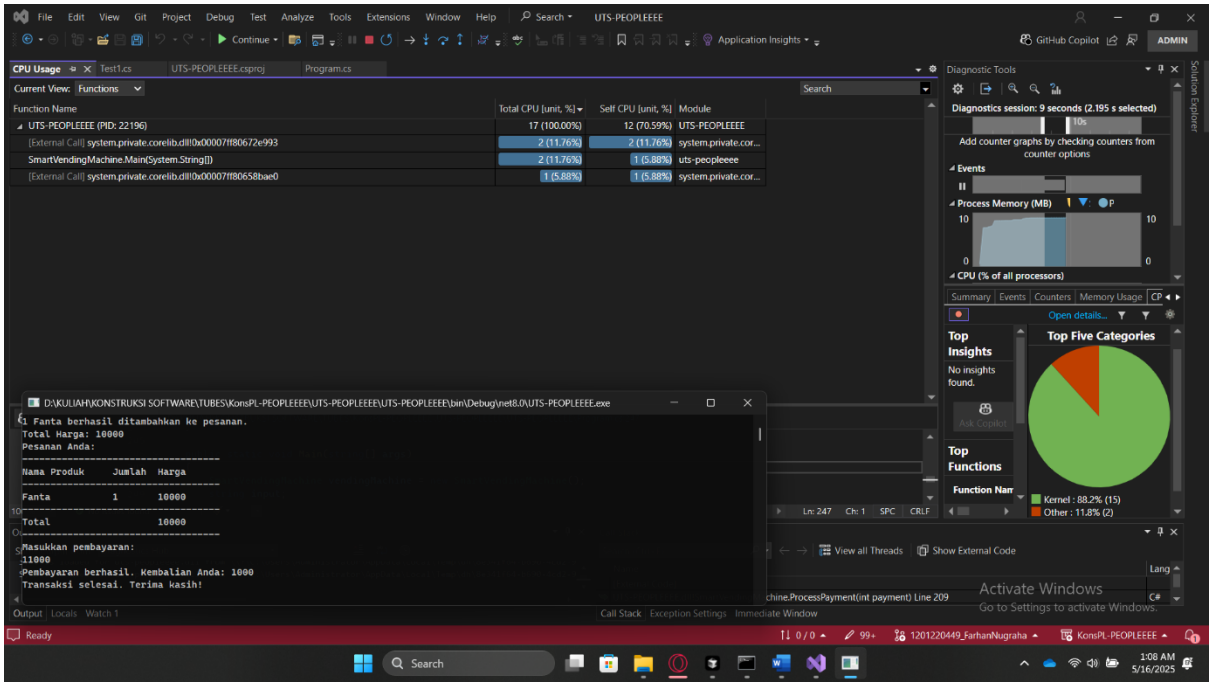
HASIL PERFOMANCE TESTING



Pada penjalanan program pertama kali, masih menampilkan list produk yang dijual pada vending. Hal ini masih cenderung ringan, ditandai dengan penggunaan kernel yang mencapai 99.2%.



Kemudian setelah dipilihnya kode produk yang ingin dibeli, dilakukan pembayaran dimana kuantitas produk otomatis hanya satu (sekali transaksi vending machine hanya bisa checkout satu produk). Disini juga hanya menampilkan produk yang terpilih beserta total harganya, dimana hal ini cenderung ringan ditandai dengan penggunaan kernel yang mencapai 100%.



Kemudian, dilakukan proses perhitungan update stok jika pembayaran berhasil, serta perhitungan kembalian yang sesuai dengan harga produk per unitnya. Hal ini menggunakan beberapa algoritma perhitungan yang sedikit lebih rumit, ditandai dengan penggunaan kernel yang cenderung berkurang drastis menjadi 88.2%.