WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Programowanie

Ignacy Tokarz
Nr albumu studenta w67657

Program "Sklep" - Interfejs dla pracownika klubu gamingowego połączonego ze sklepem

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiekotwe C#

Rzeszów 2024

Spis treści

Wstęp

W dzisiejszym dynamicznym środowisku biznesowym, gdzie czas jest cennym zasobem, skomplikowane systemy zarządzania kasą mogą stwarzać znaczne utrudnienia w codziennej pracy. Zrozumienie potrzeb przedsiębiorców oraz pracowników branży usługowej, zwłaszcza tych związanych z ofertą przekąsek, napojów i usług w nowoczesnych kafejkach internetowych, stało się kluczowym punktem wyjścia do stworzenia efektywnego narzędzia. Wychodząc naprzeciw tym potrzebom, powstał program, który nie tylko eliminuje zawiłości związane z obsługą kasy, lecz także dostarcza prostą i intuicyjną aplikację dla sprzedawców.

Aplikacja została starannie zaprojektowana z myślą o instytucjach oferujących szeroką gamę produktów oraz usług, szczególnie skupiając się na nowoczesnych kafejkach internetowych. Zdecydowanie aplikacja skupia się na dostarczeniu narzędzia, które nie tylko ułatwia proces sprzedaży, ale także efektywnie wspomaga zarządzanie magazynem i obsługą kasy fiskalnej. Wierze, że nasz program nie tylko usprawni codzienną pracę, ale także przyczyni się do wzrostu efektywności działania przedsiębiorstw z branży.

Aplikacja zawiera kompleksowy zestaw funkcji, które umożliwiają nie tylko sprawną realizację transakcji, ale również skuteczne monitorowanie stanu magazynowego. Aplikacja nie jest jedynie narzędziem do obsługi kasy; to kompleksowe rozwiązanie, które integruje funkcje sprzedaży, zarządzania magazynem i generowania raportów. W tym dokumencie szczegółowo omówię kluczowe aspekty mojego programu, prezentując korzyści, jakie niesie dla nowoczesnych kafejek internetowych oraz innych podobnych instytucji. Aplikacja dąży do nie tylko funkcjonalności, ale również łatwej obsługi.

Rozdział 1

Opis założeń projektu

1.1 Cele projetu

Głównym celem programu jest prosta i łatwa w obsłudze aplikacja, która dodatkowo będzie posiadać wszystkie najważniejsze i najpotrzebniejsze funkcje tak, aby pracownicy bez problemowo mogli zająć się klientem. Z doświadczenia wiemy jak źle zrobiony program potrafi utrudnić codzienne obowiązki w firmie, przy pracy z klientem program musi być szybki, oraz niezawodny, aby to osiągnąć moja aplikacja została stworzona w sposób minimalistyczny. Nie zawiera żadnych zbędnych funkcji. Postawiłem na prosty interfejs konsolowy z krótkim oraz prostym menu, które zawiera wszystkie najpotrzebniejsze funkcje potrzebne do pracy.
Program zawiera 4 główne zakładki:

- Stan Kasy
- Magazyn
- Sprzedaż
- Koniec Dnia

Każda z nich zawiera dostęp do innych funkcjonalności projektu

1.2 Wymagania funkcjonale i niefunkcjonalne

Definicja:
Wymagania funkcjonalne

- Program posiada 3 bazy danych, kasę, magazyn oraz koniec dnia.
- Koniec Dnia
  - Każda transakcja, która przechodzi przez system jest zapisywana w bazie danych "koniec dnia"gdzie pokazane są szczegóły każdej wykonanej transakcji do momentu zakończenia dnia. w tej kategorii są dwie funkcje:
    * Wyświetl utarg : program wyświetli wszystkie wykonane transakcje w danym dniu (od momentu rozpoczęcia do momentu zakończenia)
    * Zakończ dzień : Program wysyła informacje do kasy fiskalnej, o końcu dnia (co, ile), oraz resetuje bazę danych przygotowując się na kolejny dzien. Istotną rzeczą jest fakt, że mimo zamknięcia programu w bez zakończenia dnia, dzień się nie zakończy i zachowa swoje dane na następną sesje.
- Stan Kasy

  - Kasa zapisuje ilość pobranej gotówki lub kwoty transakcji kartą płatniczą, oraz posiada pole gdzie sumuje obie metody płatności. Dzięki temu wiemy ile mamy pieniędzy w systemie

- Magazyn
  - Magazyn posiada 4 zasadnicze funkcjonalności, wyświetlanie, dodawanie stanu do istniejącego produktu, dodawanie nowego oraz usunięcie produktu z magazynu
  - Wyświetl stan: Program pokaże wszystkie stany oraz cenę netto produktów, aktualnie są dwie kategorie więc wyświetli napoje, oraz akcesoria
  - Dodaj stan: Po wybraniu tej opcji program zapyta się na jakim typie produktu chcemy działać, następnie wyświetli ponumerowaną listę aktualnie dodanych produktów razem z ich stanem należy wybrać pozycję oraz ilość którą chcemy dodać.
  - Nowy Produkt: Podobnie program zapyta do jakiej listy chcemy dodać produkt po wybraniu rodzaju program zapyta o nazwę, ilość, oraz cenę netto
  - Usuń Produkt: Standardowo program pyta o kategorie, następnie wyświetla listę wybieramy który produkt ma zostać usunięty

- Sprzedaż
  - Kategoria sprzedaż w aktualnej wersji posiada trzy podkategorie, w których można dokonać sprzedaży, usługi gier, sprzedaży napojów oraz akcesoriów
  - Wejścia Gry: dodaje możliwość sprzedaży usługi wejść na sprzęt
  - Napoje: umożliwia sprzedaż napojów
  - Akcesoria: Dodaje sprzedaż akcesoriów

- Wymagania sprzętowe mojego programu są minimalne, każdy komputer z aktualnie dostępnym procesorem poradzi sobie z obsługą programu

Wymagania niefunkcjonalne

- Jedna z najważniejszych rzeczy podczas korzystania z programu jest dopilnowanie, aby nie usunąć plików z bazami danych, program jest zabezpieczony na taki wyjątek i stworzy wszystkie brakujące bazy natomiast wszystkie dane przepadną, które mogą być kluczowe do funkcjonowania firmy.

- Program w pełni jest konsolowy dzięki czemu jego wymagania do uruchomienia są minimalne oraz jest niezawodny a interfejs prosty i przyjazny dla użytkownika

- Aktualna wersja programu jest przygotowana pod potencjalną rozbudowę o kolejne produkty, kolejne rodzaje usług, bez problemu również można do niego dodać kolejne funkcjonalności

- Bezpieczeństwo aplikacji jest na wysokim poziomie, ponieważ wszystko odbywa się ze strony klienta i nie jest w żaden sposób połączone z Internetem czy innymi komputerami, jedyną rzeczą jaką pracownicy muszą zadbać to przypilnować pliki baz danych, aby nikt ich nie edytował ręcznie

- Program jest również zabezpieczony na wypadek nieoczekiwanego zamknięcia komputera lub programu, wszystkie zmiany na bieżąco się zapisują po nagłym wyłączeniu dane nie zostaną utracone

Rozdział 2

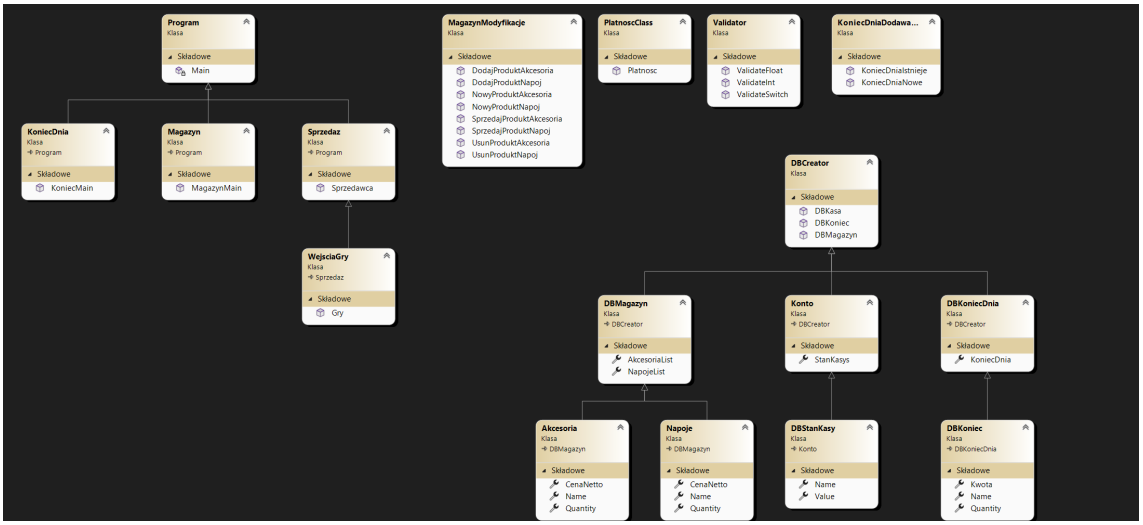
Opis struktury projektu

2.1 Struktura programu

Program posiada dwie główne struktury oraz cztery pomocnicze klasy. Pierwszą strukturą jest pięć klas połączonych, które tworzą funkcjonalną oraz wizualną część programu, zawiera się w nim Klasa Program, jest to główna pętla programu oraz główne menu wyboru działań dalej z menu możemy wybrać KoniecDnia, Magazyn, Sprzedaż. Koniec Dnia zapisuje wszystkie obsłużone sprzedaże, w magazynie mamy możliwość ręcznie edycji stanów magazynowych, takich jak dodawanie, stanu dodawanie nowych produktów czy ich usuwanie. Sprzedaż zaś jest kluczowym narzędziem w pracy programu, w tej klasie jest zawarte sprzedawanie produktów z magazynu jak i usług, wszystkie działania, które przechodzą przez klasę Sprzedaż w jakiś sposób edytują każdą z baz danych.

Drugą kluczową strukturą programu jest grupa "Baz danych"bazy danych są stworzone w pliku JSON na potrzeby programu zostały przygotowane trzy bazy danych, Magazyn, Kasa(Konto), KoniecDnia służą one do zapisania stanów magazynowych, kasa do przechowywania informacji ile pieniędzy wpłynęło do kasy przez gotówkę lub terminal, koniecDnia zapisuje wszystkie transakcje danego dnia w szczegółowy sposób. Na rysunku są przedstawione wszystkie ich właściwości.

Powstały również cztery klasy "Pomocnicze"MagazynModyfikacje, PlatnoscClass, Validator, KoniecDniaDodawanie, każda z tych klas pomaga w działaniu programu w innym jego aspekcie. MagazynModywikacje obsługuję wszystkie działania, które wpływają na bazę danych Magazyn operacje dodawania, usuwania oraz modyfikacji danych. PlatnoscClass Klasa uniwersalna przypasana do każdej operacji sprzedaży, zawiera wygląd graficzny jak i przekazuje dane do baz danych konca dnia oraz stanu kasy. Validator jak sama nazwa wskazuje pomaga w sprawdzaniu(Validacji) przekazywanych danych przez użytkownika, kiedy potrzeba wybrać numer w menu, lub wpisać ilość pieniędzy lub produktu.



Rysunek 2.1: Diagram klas Programu

2.2 Wymagania sprzętowe i programowe

Minimalne wymagania systemowe do poprawnego działania programu:

- 4 MB wolnego miejsca na dysku
- Procesor o częstotliwości taktowania 1Ghz lub więcej
- 1 GB pamięci RAM

Obsługiwane systemy operacyjne:

- Windows 10
- System 64-bit

2.3 Zarządzanie danymi i bazy danych

W programie są stworzone trzy bazy danych Magazyn, Kasa, KoniecDnia. Każda baza jest stworzona w formacie JSON oraz ma swoją klasę "DBMagazyn", "DBKoniecDnia", "DBStanKasys", rys. 2.2. Główne ich właściwości to Nazwa, Ilość oraz Cena netto z której później program korzysta do obliczenia kosztu danych produktów. Polityka firmy stawia na pełne kwoty więc w końcowej formie mimo że cena netto najczęściej jest z wprowadzana "z groszami"(float) w końcowej formie przedstawiona jest dla klienta jako równa suma (int). Stan kasy różni się pod względem wyglądu oraz zawartości bo jako jedyna do niej nie są dodawane żadne rekordy tylko zawsze posiada trzy pola które są edytowane, przez dodawanie sum pieniędzy.

```
namespace Aplikacja_sklep
{
    Odwołania: 2
    internal class Konto : DBCreator
    {
        1 odwołanie
        public List<DBStanKasy> StanKasys { get; set; }
    }

    Odwołania: 5
    internal class DBStanKasy :Konto
    {
        Odwołania: 3
        public string Name { get; set; }
        Odwołania: 3
        public float Value { get; set; }
    }

}
```

Stan kasys

```
namespace Aplikacja_sklep
{
    Odwołania: 4
    internal class DBKoniecDnia : DBCreator
    {
        Odwołania: 2
        public List<DBKoniec> KoniecDnia {  get; set; }
    }

    Odwołania: 4
    internal class DBKoniec : DBKoniecDnia
    {
        Odwołania: 2
        public string Name { get; set; }
        Odwołania: 2
        public int Quantity { get; set; }
        Odwołania: 2
        public float Kwota { get; set; }
    }
}
```

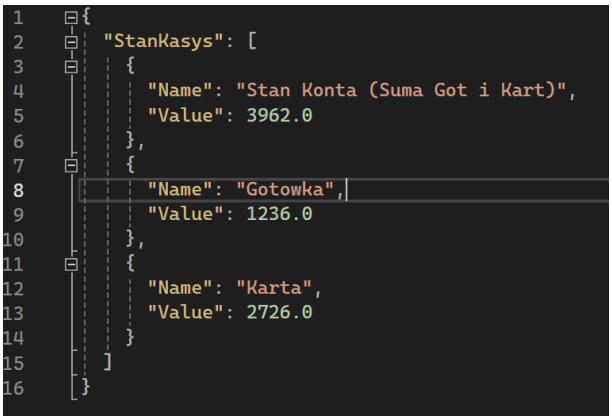
KoniecDnia

```
namespace Aplikacja_sklep
{
    Odwołania: 11
    internal class DBMagazyn : DBCreator
    {
        Odwołania: 3
        public List<Napoje> NapojeList {  get; set; }
        Odwołania: 3
        public List<Akcesoria> AkcesoriaList { get; set; }
    }

    Odwołania: 4
    internal class Napoje : DBMagazyn
    {
        Odwołania: 2
        public string Name { get; set; }
        Odwołania: 2
        public int Quantity { get; set; }
        Odwołania: 2
        public float CenaNetto { get; set; }
    }

    Odwołania: 4
    internal class Akcesoria : DBMagazyn
    {
        Odwołania: 2
        public string Name { get; set; }
        Odwołania: 2
        public int Quantity { get; set; }
        Odwołania: 2
        public float CenaNetto { get; set; }
    }
}
```

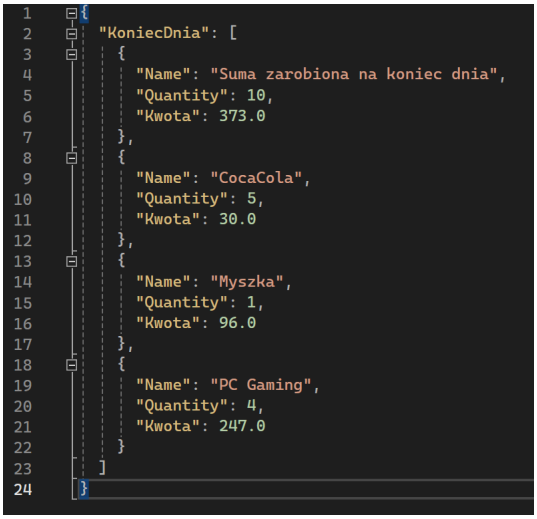
Magazyn

Rysunek 2.2: Klasy z właściwościami baz danych

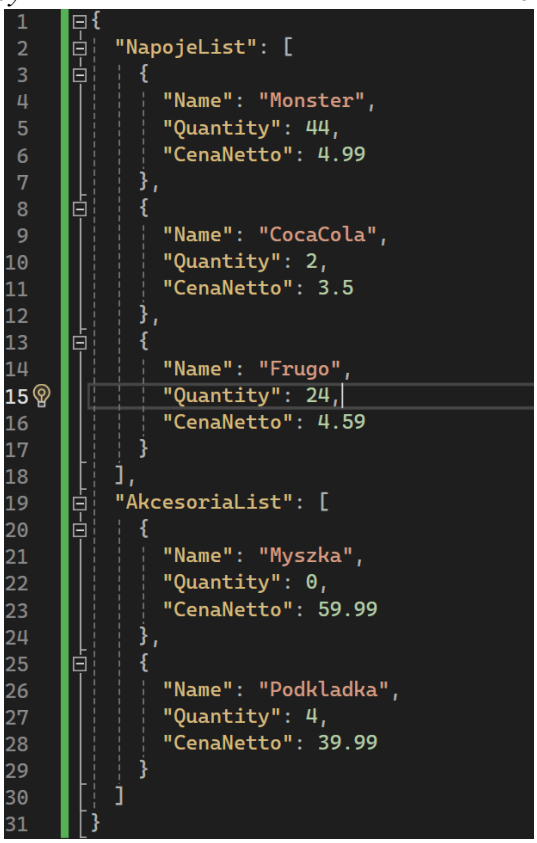
Baza danych Magazyn rys.2.3 jest przygotowana pod przyszły rozwój, bez problemowo można dodawać kolejne kategorię. Baza o nazwie StanKasy rys.2.3 nie powinna być edytowana ponieważ jest przygotowana tylko do wyświetlania 3 danych sumy zarobionej, sumy gotówki oraz sumy płatności przez terminal. Ostatnia baza czyli KoniecDnia rys.2.3 najbardziej dynamiczna Baza danych, ponieważ każda transakcja zostaje w niej zapisana aż do zakończenia dnia, wtedy cała baza przekazuje dane do drukarki a baza sama w sobie się czyści oraz przygotowuje się do następnego dnia.



StanKasy



KoniecDnia



Magazyn

Rysunek 2.3: Wnętrze baz danych

Program jest zabezpieczony na wypadek uszkodzenia, usunięcia lub utraty plików baz danych. Przy każdym starcie program sprawdza czy pliki baz danych są stworzone, jeżeli ich nie znajdzie stworzy utraconą bazę od nowa z podstawowymi danymi w środku, dzięki czemu program będzie dalej mógł pracować. Do zrobienia tych operacji stworzyłem klasę "DBCreator"w której znajduje się trzy metody każda odpowiedzialna za poszczególną bazę danych rys. 2.4.

```
public static void DBMagazyn(string path)
{
    if (!File.Exists(path))
    {
        StreamWriter sw;
        sw = File.CreateText(path);
        sw.Close();

        var magazyn = new DBMagazyn()
        {
            NapojeList = new List<Napoje>()
            {
                new Napoje()
                {
                    Name = "Monster",
                    Quantity = 1,
                    CenaNetto = 4.99f
                }
            },

            AkcesoriaList = new List<Akcesoria>()
            {
                new Akcesoria()
                {
                    Name = "Myszka",
                    Quantity = 1,
                    CenaNetto = 59.99f
                }
            }
        };

        string magazynSerialized = JsonConvert.SerializeObject(magazyn);

        File.WriteAllText(path, magazynSerialized);
    }
}
```

Magazyn

```
Odwołania: 2
public static void DBKoniec(string path)
{
    if (!File.Exists(path))
    {
        StreamWriter sw;
        sw = File.CreateText(path);
        sw.Close();

        var koniec = new DBKoniecDnia()
        {
            KoniecDnia = new List<DBKoniec>()
            {
                new DBKoniec()
                {
                    Name = "Suma zarobiona na koniec dnia",
                    Quantity = 0,
                    Kwota = 0f
                }
            }

        };

        string koniecSerialized = JsonConvert.SerializeObject(koniec);

        File.WriteAllText(path, koniecSerialized);

    }
}
}
```

KoniecDnia

```
Odwołania: 7
internal class DBCreator
{
    1 odwołanie
    public static void DBKasa(string path)
    {
        if (!File.Exists(path))
        {
            StreamWriter sw;
            sw = File.CreateText(path);
            sw.Close();

            var konto = new Konto()
            {
                StanKasys = new List<DBStanKasy>()
                {
                    new DBStanKasy()
                    {
                        Name = "Stan Konta (Suma Got i Kart)",
                        Value = 0
                    },
                    new DBStanKasy()
                    {
                        Name = "Gotowka",
                        Value = 0
                    },
                    new DBStanKasy()
                    {
                        Name = "Karta",
                        Value = 0
                    }
                }
            };
            string kontoSerialized = JsonConvert.SerializeObject(konto);
            File.WriteAllText(path, kontoSerialized);
        }
```

StanKasy

Rysunek 2.4: Klasa Creator oraz jej metody

Rozdział 3

Harmonogram realizacji projektu

3.1 Diagram Grantta

Diagram przedstawia tok w jakim powstawała aplikacja rys.3.1. Pierwszymi mi krokami było stworzenie głównego menu oraz wstępne rozplanowanie co gdzie będzie się znajdywać, po etapie planowania przystąpiłem do tworzenia kolejnych funkcjonalności programu, zaraz po pierwszych linijkach kodu zauważyłem, że niezbędną będzie stworzenie klasy "Validator"do sprawdzania danych wejściowych z konsoli, ponieważ mały błąd użytkownika taki jak podanie "null"lub stringa podczas wyboru operacji, powodował błąd krytyczny dlatego powstała klasa "Validator".

Kolejnymi krokami jakie podjąłem była implementacja funkcji sprzedaży oraz testowanie programu, funkcja sprzedaż na tym etapie była tylko gotowa w 30%, ponieważ do sprzedawania produktów potrzebna była mi baza danych "Magazyn", co było moim kolejnym etapem. Tworzenie baz danych - W tym etapie stworzenie baz danych ,powstały klasy tworzące bazy danych oraz na potrzebę zapobiegania błędów klasa "DBCreator". Po stworzeniu baz danych mogłem w końcu przystąpić do tworzenia kolejnej funkcjonalności programu jaką była "Magazyn", gdzie dodałem możliwość edycji bazy danych.

Na końcu musiałem wszystko połączyć w całość, oraz przetestować działanie i upewnić się, że program zapobiega pomyłkom użytkowników. Dokończyłem funkcje "Sprzedaż"dodałem możliwość kupowania produktów, które są wpisane na magazyn, następnie wybór metody płatności i zapisanie transakcji do baz "koniec dnia"i "Kasa". Ostaniem etapem było finalne przetestowanie działania programu oraz debugowanie oraz napisanie dokumentacji.

Rysunek 3.1: Diagram Grantta

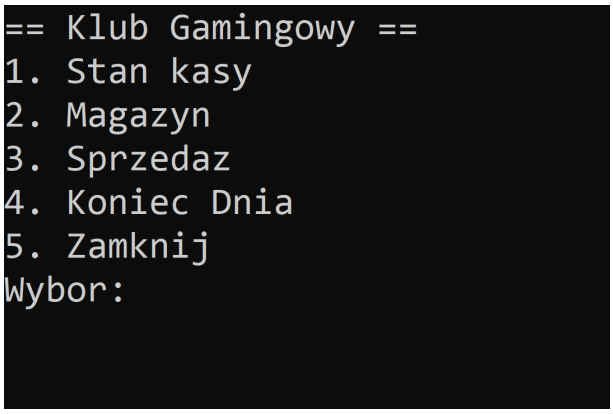
3.2 Repozytorium

Repozytorium z wszystkimi plikami źródłowymi znajduje się na moim profilu Github "B0b0lin"pod linkiem : Strona github z plikami, kliknij tutaj!

Rozdział 4

Prezentacja warstwy użytkowej projektu

Ten rozdział pokazuje kod źródłowy każdej z funkcji programu oraz wygląd w programie. Główną pętlą na jakiej opiera się program to pętla "Switch-Case", był to najlepszy wybór jeżeli chodzi o menu rys.4.1.

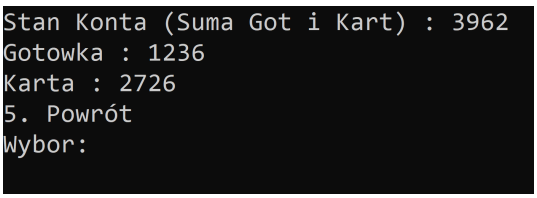


Rysunek 4.1: Menu główne programu

4.1 Stan kasy

Na początku wykorzystuje bibliotekę "Newtonsoft Json"deserializacja bazy danych oraz przypisanie do zmiennej, następnie iteracja po bazie i wypisanie po kolei wszystkich elementów w tym wypadku trzy elementy, następnie program czeka na wprowadzenie liczby "5", aby powrócić do lobby rys.4.2.

```
// %  STAN KASY  %  ZROBIONE
case 1:
    while (dziala)
    {
        string kontoSerialized = File.ReadAllText("kasa.json");
        dynamic konto = Newtonsoft.Json.JsonConvert.DeserializeObject(kontoSerialized);

        for (int i = 0; i < 3; i++)
        {
            Console.Write(konto["StanKasys"][i]["Name"] + " : ");
            Console.Write(konto["StanKasys"][i]["Value"]);
            Console.WriteLine();
        }

        Console.WriteLine("5. Powrót ");
        Console.Write("Wybor: ");
        if (Validator.ValidateSwitch(Convert.ToString(Console.ReadLine())) == 5) { Console.Clear(); dziala = false; }
        else
        {
            Console.Clear();
            Console.WriteLine("=============================================");
            Console.WriteLine("Nieprawidłowy wybor, prosze podać numer operacji");
            Console.WriteLine("=============================================");
        }
    }
    break;
```



Kod stanu kasy *Wygląd w programie*

Rysunek 4.2: Kod do wyświetlenia stanu kasy oraz wygląd w programie

4.2 Magazyn

Standardowe czyszczenie konsoli oraz wypisanie wszystkich możliwości jakie, może podjąć użytkownik. Następnie odczytanie inputa oraz jego validacja i jeżeli jest on inny niż "5"przekazywany wybór jest dalej do metody "MagazynMain"rys.4.3.

```
// %  MAGAZYN  %  ZROBIONE
case 2:
    Console.Clear();
    while (dziala)
    {
        Console.WriteLine("===  Magazyn  ===");
        Console.WriteLine("1. Wyswietl stan");
        Console.WriteLine("2. Dodaj stan");
        Console.WriteLine("3. Nowy produkt");
        Console.WriteLine("4. Usun produkt");
        Console.WriteLine("5. Powrot");
        Console.Write("Wybor: ");
        string temp = Convert.ToString(Console.ReadLine());
        if (temp == "5") { dziala = false; }
        Magazyn.MagazynMain(temp);
    }
    break;
```

Kod Menu magazynu

```
===  Magazyn  ===
1. Wyswietl stan
2. Dodaj stan
3. Nowy produkt
4. Usun produkt
5. Powrot
Wybor:
```

Wygląd Menu magazynu

Rysunek 4.3: Kod do wyświetlenia magazynu oraz wygląd w programie

4.2.1 Wyświetl stan

Kod deserializuje i przypisuje do zmiennej bazę danych magazynu, oraz wypisuje w tym wypadku dwa rodzaje produktów. Program wypisze nazwę, ilość, oraz cenę netto, następnie program oczekuje na input od użytkownika aż zakończy przeglądanie przekazując liczbę "5"co spowoduje powrót do poprzedniego menu rys.4.4.

```
public static void MagazynMain(string num)
{
    bool dziala = true;

    if (Validator.ValidateSwitch(num) < 6)
    {
        switch (Validator.ValidateSwitch(num))
        {
            //wyswietl stan magazynu ZROBIONE
            case 1:
                while (dziala)
                {
                    string magazynSerialized = File.ReadAllText("magazyn.json");
                    dynamic magazyn = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynSerialized);

                    Console.WriteLine("======= Napoje =======");
                    //loop NAPOJE
                    for (int i = 0; i < magazyn["NapojeList"].Count; i++)
                    {
                        Console.Write(magazyn["NapojeList"][i]["Name"] + " : ");
                        Console.Write(magazyn["NapojeList"][i]["Quantity"] + ", Cena netto: ");
                        Console.Write(magazyn["NapojeList"][i]["CenaNetto"]);
                        Console.WriteLine();
                    }

                    Console.WriteLine();
                    Console.WriteLine("======= Akcesoria =======");
                    //loop AKCESORIA
                    for (int i = 0; i < magazyn["AkcesoriaList"].Count; i++)
                    {
                        Console.Write(magazyn["AkcesoriaList"][i]["Name"] + " : ");
                        Console.Write(magazyn["AkcesoriaList"][i]["Quantity"] + ", Cena netto: ");
                        Console.Write(magazyn["AkcesoriaList"][i]["CenaNetto"]);
                        Console.WriteLine();
                    }

                    Console.WriteLine();
                    Console.WriteLine("5. Zamknij");
                    Console.Write("Wybor: ");
                    int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
                    if (temp == 5) { dziala = false; }

                }
                break;
```

Kod wyświetlenia stanu magazynowego

```
======= Napoje =======
Monster : 44, Cena netto: 4,99
CocaCola : 2, Cena netto: 3,5
Frugo : 24, Cena netto: 4,59

======= Akcesoria =======
Myszka : 0, Cena netto: 59,99
Podkladka : 4, Cena netto: 39,99

5. Zamknij
Wybor:
```

Wygląd stanu magazynowego

Rysunek 4.4: Kod do wyświetlenia stanu magazynu oraz wygląd w programie

4.2.2 Dodaj stan

Po wybraniu opcji dodania stanu staniemy przed wyborem jaki typ produktu chcemy dostać, w aktualnej wersji są to napoje i akcesoria, po wybraniu zostanie użytkownikowi wyświetlona lista produktów z ilością stanu. Program będzie czekać na wybór produktu, a następnie na podanie ilości jaką chcemy dodać do aktualnego stanu produktu rys.4.5.

```
==== Dodaj Stan Istniejacego Produktu ====
1. Dodaj stan napojow
2. Dodaj stan akcesoriow
5. Powrot
Wybor:
```

Menu dodawania stanu magazynu

```
======= Napoje =======
1. Monster, Stan: 44
2. CocaCola, Stan: 2
3. Frugo, Stan: 24

0. Anuluj
== Podaj numer produktu, do ktorego chcesz dodac stan ==
Wybor:
```

Wygląd dodawania stanu magazynu

```
// dodaj stan magazynu ZROBIONE
case 2:
    while (dziala)
    {
        Console.WriteLine("==== Dodaj Stan Istniejacego Produktu ====");
        Console.WriteLine("1. Dodaj stan napojow");
        Console.WriteLine("2. Dodaj stan akcesoriow");
        Console.WriteLine("5. Powrot");
        Console.Write("Wybor: ");
        int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
        if (temp == 5) { dziala = false; }
        else
        {
            switch (temp)
            {
                //Dodawanie stanu istniejacego napoju
                case 1:
                    //wyswietlenie numerowanej listy produktów
                    string magazynSerialized = File.ReadAllText("magazyn.json");
                    dynamic magazyn = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynSerialized);

                    Console.WriteLine("======= Napoje =======");
                    for (int i = 0; i < magazyn["NapojeList"].Count; i++)
                    {
                        Console.WriteLine($"{i + 1}. " + magazyn["NapojeList"][i]["Name"] + ", Stan: "+ magazyn["NapojeList"][i]["Quantity"]);
                    }
                    Console.WriteLine();

                    //wybor pozycji do dodania stanu
                    Console.WriteLine("0. Anuluj");
                    Console.WriteLine("== Podaj numer produktu, do ktorego chcesz dodac stan ==");
                    Console.Write("Wybor: ");
                    int wybor = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    if (wybor > magazyn["NapojeList"].Count) { Console.WriteLine("Nie prawidlowy numer..."); }
                    if (wybor > magazyn["NapojeList"].Count || wybor == 0)
                    {
                        Console.WriteLine("Anulacja...");
                        Thread.Sleep(2000);
                        Console.Clear();
                    }
                    else
                    {
                        Console.Clear();
                        Console.WriteLine("Podaj ilosc jaka chcesz dodac do wybranego produktu");
                        Console.Write("Ilosc: ");
                        int iloscN = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                        if (iloscN < 1) {Console.Clear(); Console.WriteLine("Wartosc musi byc wieksza od 0!"); }
                        else
                        {
                            Console.Clear();
                            MagazynModyfikacje.DodajProduktNapoj(wybor - 1, iloscN);
                        }
```

Kod dodawania stanu magazynu

Rysunek 4.5: Dodawanie stanów produktów do magazynu

4.2.3 Nowy produkt

Wybranie opcji dodania nowego produktu analogicznie program zada użytkownikowi pytanie do jakie kategorii chcemy przypisać produkt następnie poprosi o wpisanie jego nazwa, ilości oraz ceny netto, po zatwierdzeniu wszystkich danych zostanie on dodany do listy magazynowej rys.4.6.

```
==== Dodawanie nowego produktu ====
1. Dodaj napoj
2. Dodaj akcesoria
5. Powrot
Wybor:
```

Menu dodawanie nowego produktu

```
Podaj nazwe nowego produktu: test
Podaj ilosc: 1
Podaj cene netto: 1
Czy dodac produkt?
1. Tak
2. Nie
Wybor:
```

Wygląd dodawania nowego produktu

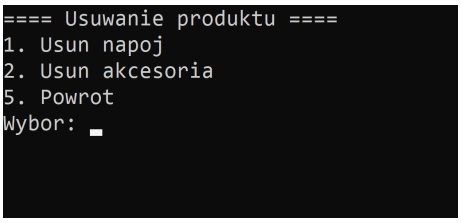
```
// Nowy produkt magazynu ZROBIONE
case 3:
    while (dziala)
    {
        Console.WriteLine("==== Dodawanie nowego produktu ====");
        Console.WriteLine("1. Dodaj napoj");
        Console.WriteLine("2. Dodaj akcesoria");
        Console.WriteLine("5. Powrot");
        Console.Write("Wybor: ");
        int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
        if (temp == 5) { dziala = false; }
        else
        {
            switch (temp)
            {
                //dodawanie napoju
                case 1:
                    Console.Clear();
                    Console.Write("Podaj nazwe nowego produktu: ");
                    string nazwaN = Convert.ToString(Console.ReadLine());
                    Console.Write("Podaj ilosc: ");
                    int iloscN = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    Console.Write("Podaj cene netto: ");
                    float nettoN = Validator.ValidateFloat(Console.ReadLine());
                    Console.WriteLine("Czy dodac produkt?");
                    Console.WriteLine("1. Tak");
                    Console.WriteLine("2. Nie");
                    Console.Write("Wybor: ");
                    int tempWybor = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    if(tempWybor == 1) { MagazynModyfikacje.NowyProduktNapoj(nazwaN, iloscN, nettoN); }
                    if(tempWybor == 2) { Console.Clear(); dziala = false; }
                break;

                //dodawanie akcesoriow
                case 2:
                    Console.Clear();
                    Console.Write("Podaj nazwe nowego produktu: ");
                    string nazwaA = Convert.ToString(Console.ReadLine());
                    Console.Write("Podaj ilosc: ");
                    int iloscA = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    Console.Write("Podaj cene netto: ");
                    float nettoA = Validator.ValidateFloat(Console.ReadLine());
                    Console.WriteLine("Czy dodac produkt?");
                    Console.WriteLine("1. Tak");
                    Console.WriteLine("2. Nie");
                    Console.Write("Wybor: ");
                    int tempWybor1 = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    if (tempWybor1 == 1) { MagazynModyfikacje.NowyProduktAkcesoria(nazwaA, iloscA, nettoA); }
                    if (tempWybor1 == 2) {Console.Clear(); dziala = false; }

                break;
```

Kod dodawania nowego produktu

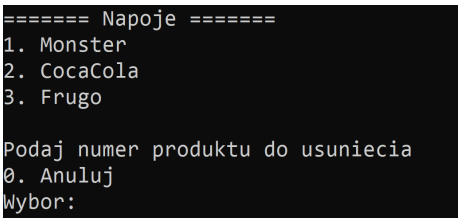
Rysunek 4.6: Dodawanie nowych produktów do magazynu

4.2.4 Usun produkt

Wybranie opcji dodania nowego produktu analogicznie program zada użytkownikowi pytanie do jakiej kategorii chcemy przypisać produkt następnie poprosi o wpisanie jego nazwa ,ilości oraz ceny netto, po zatwierdzeniu wszystkich danych zostanie on dodany do listy magazynowej rys.4.7.



Menu usuwania produktów



Wygląd usuwania produktów

```
//Usuwanie produktów ZROBIONE
case 4:
    while (dziala)
    {
        Console.WriteLine("==== Usuwanie produktu ====");
        Console.WriteLine("1. Usun napoj");
        Console.WriteLine("2. Usun akcesoria");
        Console.WriteLine("5. Powrot");
        Console.Write("Wybor: ");
        int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
        if (temp == 5) { dziala = false; }
        else
        {
            switch (temp)
            {
                //usuwanie napoju
                case 1:
                    //wyswietlenie numerowanej listy produktów
                    string magazynSerialized = File.ReadAllText("magazyn.json");
                    dynamic magazyn = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynSerialized);

                    Console.WriteLine("======= Napoje =======");
                    for (int i = 0; i < magazyn["NapojeList"].Count; i++)
                    {
                        Console.WriteLine($"{i+1}. "+magazyn["NapojeList"][i]["Name"]);
                    }
                    Console.WriteLine();

                    //wybor pozycji do usuniecia
                    Console.WriteLine("Podaj numer produktu do usuniecia");
                    Console.WriteLine("0. Anuluj");
                    Console.Write("Wybor: ");
                    int wybor = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
                    if (wybor > magazyn["NapojeList"].Count) { Console.WriteLine("Nie prawidlowy numer..."); }
                    if (wybor > magazyn["NapojeList"].Count || wybor == 0)
                    {
                        Console.WriteLine("Anulacja...");
                        Thread.Sleep(2000);
                        Console.Clear();
                    }
                    else
                    {
                        Console.Clear();
                        MagazynModyfikacje.UsunProduktNapoj(wybor - 1);
                    }
                break;
```

Kod usuwania produktów napoje

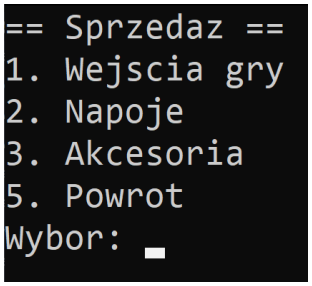
```
//usuwanie akcesoriow
case 2:
    //wyswietlenie numerowanej listy produktów
    string magazynSerialized1 = File.ReadAllText("magazyn.json");
    dynamic magazyn1 = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynSerialized1);

    Console.WriteLine("======= Akcesoria =======");
    for (int i = 0; i < magazyn1["AkcesoriaList"].Count; i++)
    {
        Console.WriteLine($"{i + 1}. " + magazyn1["AkcesoriaList"][i]["Name"]);
    }
    Console.WriteLine();

    //wybor pozycji do usuniecia
    Console.WriteLine("Podaj numer produktu do usuniecia");
    Console.WriteLine("0. Anuluj");
    Console.Write("Wybor: ");
    int wybor1 = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
    if (wybor1 > magazyn1["AkcesoriaList"].Count) { Console.WriteLine("Nie prawidlowy numer..."); }
    if (wybor1 > magazyn1["AkcesoriaList"].Count || wybor1 == 0)
    {
        Console.WriteLine("Anulacja...");
        Thread.Sleep(2000);
        Console.Clear();
    }
    else
    {
        Console.Clear();
        MagazynModyfikacje.UsunProduktAkcesoria(wybor1 - 1);
    }
    break;
```

Kod usuwania produktów akcesoria

Rysunek 4.7: Usuwanie produktów z magazynu

4.3 Sprzedaż

Trzecia opcja to sprzedaż, po wybraniu tej opcji pojawi się przed nami wybór rodzaju sprzedaży, wejścia gry oraz produkty - napoje, akcesoria. Wszystkie operacje przechodzące przez tą opcje będą miały wpływ na bazy danych, dodawanie zarobionych pieniędzy, usuwanie sprzedanych produktów z magazynu czy dodawanie informacji do bazy końca dnia rys.4.8.



Menu sprzedaży

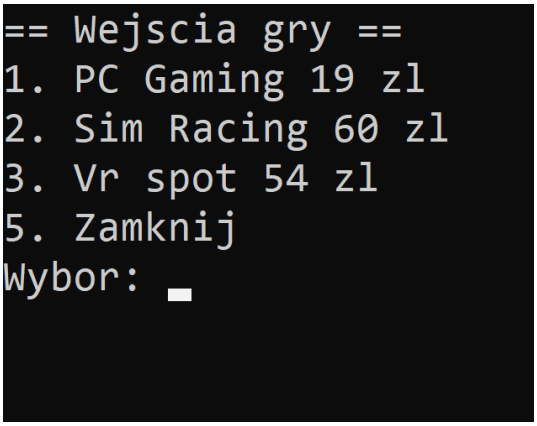
```
// % SPRZEDARZ  %  ZROBIONE
case 3:

    Console.Clear();
    while (dziala)
    {
        Console.WriteLine("== Sprzedaz ==");
        Console.WriteLine("1. Wejscia gry");
        Console.WriteLine("2. Napoje");
        Console.WriteLine("3. Akcesoria");
        Console.WriteLine("5. Powrot");
        Console.Write("Wybor: ");
        string temp = Convert.ToString(Console.ReadLine());
        if (temp == "5") { dziala = false; }
        Sprzedaz.Sprzedawca(temp);
    }
    break;
```

Kod menu sprzedaży

Rysunek 4.8: Menu sprzedaży

4.3.1 Wejscia gry

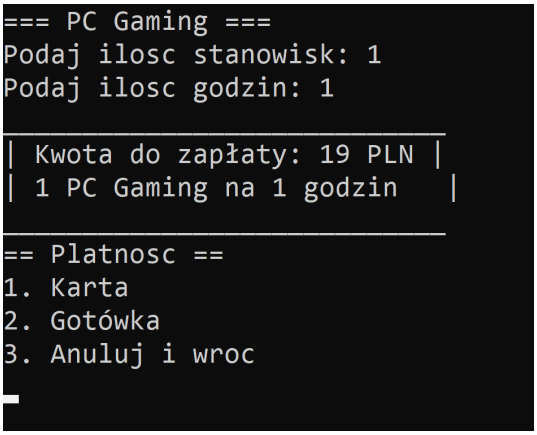
Po wyborze tej funkcji użytkownik musi wybrać jedną z trzech dostępnych atrakcji w lokalu, po wyborze rodzaju użytkownik musi podać ilość stanowisk oraz ilość godzin, następnie program podliczy oraz wyświetli co zostaje sprzedane. Ostatnim krokiem jest wybór sposobu płatności lub anulacja transakcji. Dalsza część płatności jest w klasie Płatność rys.4.9.



Menu Wejscia gry

```
//Sprzedarz uslugi gier   ZROBIONE
case 1:
    while (dziala)
    {
        Console.WriteLine("== Wejscia gry ==");
        Console.WriteLine("1. PC Gaming 19 zl");
        Console.WriteLine("2. Sim Racing 60 zl");
        Console.WriteLine("3. Vr spot 54 zl");
        Console.WriteLine("5. Zamknij");
        Console.Write("Wybor: ");
        int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
        if (temp == 5) { dziala = false; }
        WejsciaGry.Gry(temp);
        dziala = false;
    }
    break;
```

Kod menu wejscia gry



Menu sprzedaży usługi

```
while (dziala)
{
    Console.WriteLine("=== PC Gaming ===");
    int ilosc;
    int godziny;
    do
    {
        Console.Write("Podaj ilosc stanowisk: ");
        ilosc = Validator.ValidateInt(Console.ReadLine());
    } while (ilosc < 0);

    do
    {
        Console.Write("Podaj ilosc godzin: ");
        godziny = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
    } while (godziny < 0);

    Console.WriteLine( "_____________________________");
    Console.WriteLine($"| Kwota do zapłaty: {ilosc * godziny * 19} PLN |");
    Console.WriteLine($"| {ilosc} PC Gaming na {godziny} godzin   |");
    Console.WriteLine( "_____________________________");
    Console.WriteLine($"== Platnosc ==");
    Console.WriteLine( "1. Karta");
    Console.WriteLine( "2. Gotówka");
    Console.WriteLine( "3. Anuluj i wroc");
    string temp = Console.ReadLine();
    if (temp == "3") {dziala = false; }
    PlatnoscClass.Platnosc(temp, ilosc * godziny * 19);

    // dodawanie do bazy końca dnia
    string requestDeserialized = File.ReadAllText("koniec.json");
    dynamic request = Newtonsoft.Json.JsonConvert.DeserializeObject(requestDeserialized);
    bool istnieje = false;

    string name = "PC Gaming";
    for (int i = 0; i < request["KoniecDnia"].Count; i++)
    {
        if (request["KoniecDnia"][i]["Name"] == name) { istnieje = true; }
    }

    if (istnieje)
    { KoniecDniaDodawanie.KoniecDniaIstnieje(ilosc, name, ilosc * godziny * 19);}
    else
    { KoniecDniaDodawanie.KoniecDniaNowe(ilosc, name, ilosc * godziny * 19);}
    dziala = false;
}
break;
```

Kod sprzedaży usługi

Rysunek 4.9: Menu Wejscia gry i sprzedaż usługi

4.3.2 Produkty

Po wybraniu sprzedaży produktu (napoju lub akcesoria) program wyświetli użytkownikowi listę wybranej kategorii, znajduję się w niej nazwa, ilość stanu, oraz kwota brutto, która jest wyliczana z kwoty netto z magazynu, Cena zawsze będzie liczbą rzeczywistą przez politykę firmy. Następnie po wyborze produktu program zapyta o ilość, po wpisaniu ilości wyświetli podsumowanie i zapyta o metodę płatności rys.4.10.

```
1. Monster, Stan: 44
   Cena Brutto: 8 zl

2. CocaCola, Stan: 2
   Cena Brutto: 6 zl

3. Frugo, Stan: 24
   Cena Brutto: 7 zl


0. Anuluj
== Wybierz produkt do sprzedania ==
Wybor:
```

Menu Wejscia gry

```
_____________________________
| Kwota do zapłaty: 16 PLN |
| 2 X Monster |
_____________________________
== Platnosc ==
1. Karta
2. Gotówka
3. Anuluj i wroc
```

Menu wyboru zapłaty i podsumowanie

```
// sprzedarz napojow    ZROBIONE
case 2:
    //wyswietlenie numerowanej listy produktów z cenami
    string magazynSerialized = File.ReadAllText("magazyn.json");
    dynamic magazyn = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynSerialized);

    Console.WriteLine("======= Napoje Na Sprzedarz =======");
    for (int i = 0; i < magazyn["NapojeList"].Count; i++)
    {
        Console.WriteLine($"{i + 1}. " + magazyn["NapojeList"][i]["Name"] +", Stan: " + magazyn["NapojeList"][i]["Quantity"]);
        Console.WriteLine($"   Cena Brutto: " + Convert.ToInt32(magazyn["NapojeList"][i]["CenaNetto"] +
            (magazyn["NapojeList"][i]["CenaNetto"] * 60) / 100) + " zl");
        Console.WriteLine();
    }
    Console.WriteLine();

    Console.WriteLine("0. Anuluj");
    Console.WriteLine("== Wybierz produkt do sprzedania ==");
    Console.Write("Wybor: ");
    int wybor = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
    if (wybor > magazyn["NapojeList"].Count) { Console.WriteLine("Nie prawidlowy numer..."); }
    if (wybor > magazyn["NapojeList"].Count || wybor == 0)
    {
        Console.WriteLine("Anulacja...");
        Thread.Sleep(2000);
        Console.Clear();
    }
    else
    {
        int cenaNapoj = Convert.ToInt32(magazyn["NapojeList"][wybor-1]["CenaNetto"] + (magazyn["NapojeList"][wybor-1]["CenaNetto"] * 60) / 100);

        Console.Clear();
        Console.WriteLine("Ile produktów chce kupic klient?");
        Console.Write("Ilosc: ");
        int iloscN = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
        if (iloscN < 1) { Console.Clear(); Console.WriteLine("Wartosc musi byc wieksza od 0!"); }
        if (iloscN > Convert.ToInt32(magazyn["NapojeList"][wybor - 1]["Quantity"])) { Console.WriteLine("Brak towaru"); }
```

Kod menu sprzedaży napoje #1

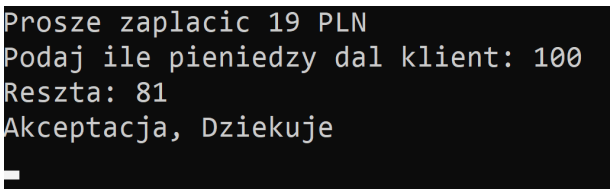
```
else
{
    Console.Clear();
    Console.WriteLine("_____________________________");
    Console.WriteLine($"| Kwota do zapłaty: {cenaNapoj * iloscN} PLN |");
    Console.WriteLine($"| {iloscN} X {magazyn["NapojeList"][wybor-1]["Name"]} |");
    Console.WriteLine("_____________________________");
    Console.WriteLine($"== Platnosc ==");
    Console.WriteLine("1. Karta");
    Console.WriteLine("2. Gotówka");
    Console.WriteLine("3. Anuluj i wroc");
    string temp = Console.ReadLine();
    if (temp == "1" || temp == "2")
    {
        PlatnoscClass.Platnosc(temp, cenaNapoj * iloscN);
        MagazynModyfikacje.SprzedajProduktNapoj(wybor - 1, iloscN);

        string requestDeserialized = File.ReadAllText("koniec.json");
        dynamic request = Newtonsoft.Json.JsonConvert.DeserializeObject(requestDeserialized);
        bool istnieje = false;

        string name = Convert.ToString(magazyn["NapojeList"][wybor - 1]["Name"]);
        for (int i = 0; i < request["KoniecDnia"].Count; i++)
        {
            if (request["KoniecDnia"][i]["Name"] == name) { istnieje = true; }
        }

        if (istnieje)
        { KoniecDniaDodawanie.KoniecDniaIstnieje(iloscN, Convert.ToString(magazyn["NapojeList"][wybor - 1]["Name"]), cenaNapoj * iloscN);break; }
        else
        { KoniecDniaDodawanie.KoniecDniaNowe(iloscN, Convert.ToString(magazyn["NapojeList"][wybor - 1]["Name"]), cenaNapoj * iloscN); break; }
    }
    else
    {
        Console.WriteLine("Anulacja...");
        Thread.Sleep(2000);
        Console.Clear();
    }
```

Kod menu sprzedaży napoje #2

Rysunek 4.10: Menu Produktów i kod

4.4 Płatność

Klasa płatność jest klasą pomocniczą używaną podczas każdej transakcji w funkcji sprzedaży, zawiera ona dodawanie sumy transakcji do bazy "kasa"do odpowiedniego pola, w zależności od wybranej metody płatności rys.4.11.



Wygląd płatności

```
string path = "kasa.json";
switch (Validator.ValidateSwitch(num))
{
    //platnosc karta
    case 1:
        //komunikacja z terminalem validacja karty kredytowej

        //zapis do pliku JSON (kasa.json)
        string kontoSerialized = File.ReadAllText(path);
        dynamic konto = Newtonsoft.Json.JsonConvert.DeserializeObject(kontoSerialized);
        konto["StanKasys"][2]["Value"] += kwota ;
        konto["StanKasys"][0]["Value"] += kwota ;
        string output = Newtonsoft.Json.JsonConvert.SerializeObject(konto, Newtonsoft.Json.Formatting.Indented);
        File.WriteAllText(path, output);

        Console.WriteLine("%Komunikacja z terminalem%");
        Thread.Sleep(1000);
        Console.WriteLine("Akceptacja, Dziekuje");
        Thread.Sleep(2000);
        Console.Clear();
        break;

    //platnosc gotowka
    case 2:
        int temp;
        do
        {
            Console.Clear();
            Console.WriteLine($"Prosze zaplacic {kwota} PLN");
            Console.Write("Podaj ile pieniedzy dal klient: ");
            temp = Validator.ValidateInt(Convert.ToString(Console.ReadLine()));
        } while (temp < kwota);

        //zapis do pliku JSON (kasa.json)
        string kontoSerialized1 = File.ReadAllText(path);
        dynamic konto1 = Newtonsoft.Json.JsonConvert.DeserializeObject(kontoSerialized1);
        konto1["StanKasys"][1]["Value"] += kwota;
        konto1["StanKasys"][0]["Value"] += kwota;
        string output1 = Newtonsoft.Json.JsonConvert.SerializeObject(konto1, Newtonsoft.Json.Formatting.Indented);
        File.WriteAllText(path, output1);


        Console.WriteLine($"Reszta: {temp - kwota}");
        Console.WriteLine("Akceptacja, Dziekuje");
        Thread.Sleep(3000);
        Console.Clear();
        break;
```

Kod płatności

Rysunek 4.11: kod z klasy PlatnosciClass

4.5 Magazyn modyfikacje

Następna klasa wspomagająca edycje produktów z bazy "Magazyn". Wykorzystywana w funkcjonalności "Sprzedaż"oraz "Magazyn", w sprzedaży wykorzystywana jest tylko metoda sprzedaj, pozostałe są do edycji magazynu poprzez "Magazyn". Dodatkowo każda metoda jest podwójna w zależności na jakim typie produktu użytkownik chce dokonać zmian.

4.5.1 Dodaj stan produktu napoj/akcesoria

Metoda używana w funkcji dodawania stanu do magazynu, deserializacja zapisanie i ponowna serializacja do pliku. Istnieje identyczna metoda natomiast ze zmienionymi danymi dla produktu akcesoria rys.4.12

```
public static void DodajProduktNapoj(int num, int ilosc)
{
    string path = "magazyn.json";

    string magazynDerialized = File.ReadAllText(path);
    dynamic konto = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynDerialized);
    konto["NapojeList"][num]["Quantity"] += ilosc;
    string output = Newtonsoft.Json.JsonConvert.SerializeObject(konto, Newtonsoft.Json.Formatting.Indented);
    File.WriteAllText(path, output);
    Console.WriteLine("Produkt zostal pomyslnie dodany!");
    Thread.Sleep(2000);
    Console.Clear();
}
```

Rysunek 4.12: Kod dodawania stanu produktu do magazynu

4.5.2 Nowy produkt napoj/akcesoria

Metoda wykorzystana w funkcji dodawania nowych produktów do magazynu rys.4.13.

```
Odwołania: 8
internal class MagazynModyfikacje
{
    1 odwołanie
    public static void NowyProduktNapoj(string name,int quantity,float cenaNetto)
    {
        //deserializacja
        string dbmagazynDeserialized = File.ReadAllText("magazyn.json");
        DBMagazyn dBMagazyn = JsonConvert.DeserializeObject<DBMagazyn>(dbmagazynDeserialized);

        //dodanie nowej pozycji do listy z napojami
        dBMagazyn.NapojeList.Add(new Napoje() { Name = name, Quantity = quantity, CenaNetto = cenaNetto });

        //serializacja danych oraz zapis do pliku
        string magazynSerialized = JsonConvert.SerializeObject(dBMagazyn);
        File.WriteAllText("magazyn.json", magazynSerialized);

        Console.WriteLine($"Produkt {name}, zostal pomyslnie dodany do magazynu");
        Thread.Sleep(2000);
        Console.Clear();
    }
```

Rysunek 4.13: Kod dodawania produktu do stanu magazynu

4.5.3 Sprzedaj produkt napoj/akcesoria

Metoda wykorzystana w funkcji sprzedawania produktów z magazynu rys.4.14.

```
public static void SprzedajProduktNapoj(int num, int ilosc)
{
    string path = "magazyn.json";

    string magazynDerialized = File.ReadAllText(path);
    dynamic konto = Newtonsoft.Json.JsonConvert.DeserializeObject(magazynDerialized);
    konto["NapojeList"][num]["Quantity"] -= ilosc;
    string output = Newtonsoft.Json.JsonConvert.SerializeObject(konto, Newtonsoft.Json.Formatting.Indented);
    File.WriteAllText(path, output);
}
```

Rysunek 4.14: Kod sprzedawania produktu z magazynu

4.5.4 Usun produkt napoj/akcesoria

Metoda wykorzystana w funkcji usuwania produktów z magazynu rys.4.15.

```
public static void UsunProduktNapoj(int num)
{
    //deserializacja
    string dbmagazynDeserialized = File.ReadAllText("magazyn.json");
    DBMagazyn dBMagazyn = JsonConvert.DeserializeObject<DBMagazyn>(dbmagazynDeserialized);

    //dodanie nowej pozycji do listy z napojami
    dBMagazyn.NapojeList.RemoveAt(num);

    //serializacja danych oraz zapis do pliku
    string magazynSerialized = JsonConvert.SerializeObject(dBMagazyn);
    File.WriteAllText("magazyn.json", magazynSerialized);

    Console.WriteLine($"Produkt, zostal pomyslnie usunięty z magazynu");
    Thread.Sleep(2000);
    Console.Clear();
}
```

Rysunek 4.15: Kod usuwania produktu z magazynu

4.6 Koniec Dnia

Menu Koniec dnia, daje nam możliwość sprawdzenia utargu jaki był w danym dniu oraz zakończenie dnia co spowoduje przesłanie informacji do kasy fiskalnej oraz wyczyszczenie i przygotowanie bazy danych "koniec dnia"na następny dzień rys.4.16.

```
===  Koniec Dnia  ===
1. Wyswietl utarg
2. Zakoncz dzien
5. Powrot
Wybor:
```

Rysunek 4.16: Menu końca dnia

4.6.1 Wyświetl utarg

Po wyświetleniu utargu Użytkownik może tylko powrócić do menu, natomiast program wyświetlki kluczowe dane ze wszystkich transakcji z aktualnego dnia pracy rys.4.17.

```
======= Utarg =======
Suma zarobiona na koniec dnia
Ilosc transakcji: 14, Suma zarobiona: 427

CocaCola
Ilosc transakcji: 5, Suma zarobiona: 30

Myszka
Ilosc transakcji: 1, Suma zarobiona: 96

PC Gaming
Ilosc transakcji: 6, Suma zarobiona: 285

Monster
Ilosc transakcji: 2, Suma zarobiona: 16


5. Zamknij
Wybor:
```

Wygląd wyświetlania utargu

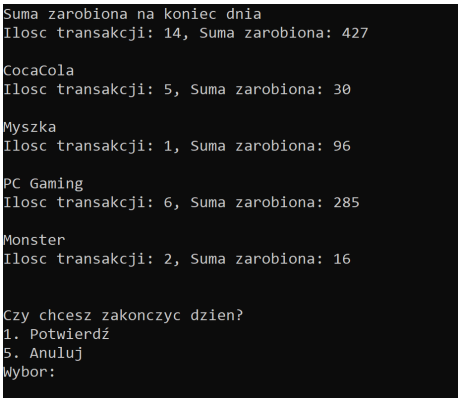
```
//zakoncz dzien
case 2:
    Console.WriteLine("=======  Koniec Dnia =======");
    while (dziala)
    {
        string koniecDniaSerialized = File.ReadAllText("koniec.json");
        dynamic koniecDnia = Newtonsoft.Json.JsonConvert.DeserializeObject(koniecDniaSerialized);

        Console.WriteLine("Utarg Z Dnia");
        Console.WriteLine();
        for (int i = 0; i < koniecDnia["KoniecDnia"].Count; i++)
        {
            Console.Write(koniecDnia["KoniecDnia"][i]["Name"] + " \nIlosc transakcji: ");
            Console.Write(koniecDnia["KoniecDnia"][i]["Quantity"] + ", Suma zarobiona: ");
            Console.Write(koniecDnia["KoniecDnia"][i]["Kwota"]);
            Console.WriteLine(); Console.WriteLine();
        }

        Console.WriteLine();
        Console.WriteLine("Czy chcesz zakonczyc dzien?");
        Console.WriteLine("1. Potwierdź");
        Console.WriteLine("5. Anuluj");
        Console.Write("Wybor: ");
        int temp = Validator.ValidateSwitch(Convert.ToString(Console.ReadLine()));
        if (temp == 1 && File.Exists("koniec.json"))
        {
            Console.WriteLine("Drukowanie...");
            Thread.Sleep(1000);
            Console.Clear();
            Console.WriteLine("Dzień zakończony");
            Thread.Sleep(1000);
            File.Delete("koniec.json");
            DBCreator.DBKoniec("koniec.json");
        }
```

Kod wyświetl utarg

Rysunek 4.17: kod z wyświetlania utargu

4.6.2 Zakończ dzień

Wybranie opcji zakończ dzień rozpocznie sekwencje kończenia dnia, najpierw program wyświetli wszystkie transakcje z danego dnia tak samo jak w "wyświetl utarg"natomiast tutaj użytkownik dostanie pytanie czy chce zakończyć dzień, po potwierdzeniu rozpocznie się przesyłanie danych do kasy fiskalnej oraz czyszczenie i przygotowanie bazy "Koniec dnia"na następny dzień roboczy rys.4.18.



Wygląd zakończ dzień

Kod zakończ dzień

Rysunek 4.18: kod z zakończ dzien

4.7 Koniec Dnia Dodawanie

Klasa pomocnicza w dodawaniu produktów do bazy "koniec dnia". Program najpierw sprawdza czy dany produkt jest już w bazie a następnie decyduje której metody użyje.

4.7.1 Produkt istnieje

Jeżeli produkt widnieje już w bazie danych zostaje wykorzystana ta metoda, przeszukuje bazę, żeby znaleźć pasujący produkt i dodać do niego wartości rys.4.19.

```
public static void KoniecDniaIstnieje(int ilosc,string typ, int kwota)
{
    string path = "koniec.json";

    string koniecDerialized = File.ReadAllText(path);
    dynamic koniec = Newtonsoft.Json.JsonConvert.DeserializeObject(koniecDerialized);

    koniec["KoniecDnia"][0]["Quantity"] += ilosc;
    koniec["KoniecDnia"][0]["Kwota"] += kwota;

    for (int i = 0; i < koniec["KoniecDnia"].Count; i++)
    {
        if(koniec["KoniecDnia"][i]["Name"] == typ)
        {
            koniec["KoniecDnia"][i]["Quantity"] += ilosc;
            koniec["KoniecDnia"][i]["Kwota"] += kwota;
        }
    }

    string output = Newtonsoft.Json.JsonConvert.SerializeObject(koniec, Newtonsoft.Json.Formatting.Indented);
    File.WriteAllText(path, output);
}
```

Rysunek 4.19: Kod jeżeli produkt istnieje

4.7.2 Produkt nieistnieje

Jeżeli program nie znajdzie danego produktu na liście w bazie "koniec dnia"to wykorzysta tą metodę do utworzenia nowego rekordu ze wszystkimi najważniejszymi danymi rys.4.20.

```
public static void KoniecDniaNowe(int ilosc, string name, int kwota)
{
    string path = "koniec.json";
    //deserializacja
    string koniecDeserialized = File.ReadAllText(path);
    dynamic koniec = Newtonsoft.Json.JsonConvert.DeserializeObject(koniecDeserialized);

    koniec["KoniecDnia"][0]["Quantity"] += ilosc;
    koniec["KoniecDnia"][0]["Kwota"] += kwota;
    string koniecSerialized = JsonConvert.SerializeObject(koniec);
    File.WriteAllText(path, koniecSerialized);

    string koniecDeserialized1 = File.ReadAllText(path);
    DBKoniecDnia dBKoniec = JsonConvert.DeserializeObject<DBKoniecDnia>(koniecDeserialized1);

    //dodanie nowej pozycji do listy z napojami
    dBKoniec.KoniecDnia.Add(new DBKoniec() { Name =  name, Quantity= ilosc, Kwota=kwota});

    //serializacja danych oraz zapis do pliku
    string dbKoniecSerialized = JsonConvert.SerializeObject(dBKoniec);
    File.WriteAllText(path, dbKoniecSerialized);

    Console.Clear();
}
```

Rysunek 4.20: Kod jeżeli produkt nieistnieje

4.8 Walidator

Klasa pomocnicza stworzona do walidacji inputów przekazywanych przez użytkownika, wykorzystywana głownie do walidacji wyborów w menu, ale również do sprawdzania przekazywanych wartości int oraz float.

4.8.1 Walidator Float

Sprawdza przekazany input czy jest wartością float, jeżeli nie prosi o wpisanie odpowiedniej liczby rys.4.21.

```
Odwołania: 2
public static float ValidateFloat(string num)
{
    float wybor;

    try
    {
        wybor = float.Parse(num, CultureInfo.InvariantCulture.NumberFormat);
        return wybor;
    }
    catch (System.FormatException ex)
    {

        Console.WriteLine("=======================================");
        Console.WriteLine("Nieprawidłowy wybor, prosze podać liczbe");
        Console.WriteLine("=======================================");
        return 0;
    }
}
```

Rysunek 4.21: Kod validator Float

4.8.2 Walidator Int

Sprawdza przekazany input czy jest wartością int, jeżeli nie prosi o wpisanie odpowiedniej liczby rys.4.22.

```
Odwołania: 21
public static int ValidateInt(string num)
{
    int wybor;

    try
    {
        wybor = Convert.ToInt32(num);
        return wybor;
    }
    catch (System.FormatException ex)
    {

        Console.WriteLine("=====================================");
        Console.WriteLine("Nieprawidłowy wybor, prosze podać liczbe");
        Console.WriteLine("=====================================");
        return 0;
    }
}
```

Rysunek 4.22: Kod validator int

4.8.3 Walidator Switch

Sprawdza przekazany input czy jest wartością int natomiast różni się wiadomością error od validator Int rys.4.23.

```
Odwołania: 18
public static int ValidateSwitch(string num)
{
    int wybor;

    try
    {
        Console.Clear();
        wybor = Convert.ToInt32(num);
        return wybor;
    }
    catch (System.FormatException ex)
    {
        Console.Clear();
        Console.WriteLine("=============================================");
        Console.WriteLine("Nieprawidłowy wybor, prosze podać numer operacji");
        Console.WriteLine("=============================================");
        return 0;
    }
}
```

Rysunek 4.23: Kod validator switch

Rozdział 5

Podsumowanie

Prezentuje najnowszą platformę do obsługi kasy i zarządzania magazynem, która obecnie znajduje się w fazie 1.0. Jednakże, to jedynie pierwszy krok w fascynującej podróży, ponieważ projekt będzie stale rozwijany, kładąc nacisk na poszerzanie funkcjonalności oraz podnoszenie jakości oprawy graficznej.

W krótkim okresie planowane jest przeprowadzić istotną zmianę, przenosząc system bazodanowy z formatu JSON na potężniejsze i bardziej elastyczne bazy danych SQL. Ten krok nie tylko zwiększy wydajność, ale także umożliwi bardziej zaawansowane i złożone operacje na danych, co przyczyni się do jeszcze lepszej efektywności aplikacji.

Innym ważnym krokiem na drodze rozwoju jest wprowadzenie możliwości manualnej edycji stanu magazynowego. To oznacza, że użytkownicy będą mogli dostosować ilość dostępnych produktów, zmieniać ceny netto czy modyfikować nazwy produktów, co znacznie zwiększy elastyczność i personalizację korzystania z systemu.

Jednak plany na rozwój nie ograniczają się jedynie do technicznych ulepszeń. W dążeniu do stworzenia kompleksowego i zintegrowanego narzędzia, w planach jest również dodanie funkcji raportowania. Użytkownicy muszą mieć dostęp do szczegółowych raportów dotyczących sprzedaży, stanu magazynowego czy preferencji klientów, co pomoże im podejmować bardziej świadome decyzje biznesowe.

Finalnie, aplikacja będzie nie tylko funkcjonalna, ale także estetyczna dzięki oprawie graficznej.Interfejs użytkownika jest projektowany z myślą o prostocie obsługi, eliminując zbędne elementy i skupiając się na ergonomicznym designie. Celem jest stworzenie oprogramowania nie tylko funkcjonalnego, ale i przyjemnego w codziennym użytkowaniu.

Podsumowując, kafejki internetowe nieustannie rozwija się i ewoluują, aby sprostać oczekiwaniom dynamicznego rynku, dlatego w duchu motta aplikacji , program zostanie zaprojektowany tak, aby był łatwy, niezawodny oraz przyjemny w użytkowaniu.

Bibliografia

[1] https://www.newtonsoft.com/json/help/html/Introduction.htm , Dokumentacja JSON.NET z dnia 20.01.2024

[2] Jacek Matulewski, *C#: lekcje programowania: praktyczna nauka programowania dla platform .NET i .NET Core*, Helion, Gliwice 2021

[3] Joseph Albahari, Eric Johannsen, *C# 8.0 w pigułce*, Helion, Gliwice, 2021

[4] R. S. Miles, *C#: zacznij programować!*, Helion, Gliwice, 2020

Spis rysunków

Spis tabel