

Projekt zaliczeniowy



Programowanie obiektowe

Rok akademicki 2024/2025

Autorki:

Wiktoria Papiz

Aleksandra Rodzinka

Katarzyna Wacławik

Temat projektu: Sklep z elektroniką

Spis treści:

Podział ról	3
Instrukcja obsługi	3
Opis funkcjonalności	7
Logika aplikacji	7
Produkt.cs	7
Smartfon.cs – klasa dziedzicząca po klasie Produkt	8
Tablet.cs – klasa dziedzicząca po klasie Produkt	9
Laptop.cs – klasa dziedzicząca po klasie Produkt	9
Magazyn.cs	10
Koszyk.cs	11
Klient.cs	13
Zamowienie.cs	14
Platnosc.cs	15
InvalidPaymentException.cs	16
ProduktNotFoundException.cs	16
Gui	18
MainWindow.xaml	18
OknoLogowania.xaml	21
Kategoria_Produktu.xaml	24
ListaProduktow.xaml	25
Specyfikacja.xaml	29
Koszyk_produkciów.xaml	34
Admin_ListaProduktów.xaml	35
DaneProduktu.xaml	38
Testy jednostkowe	39

W ramach projektu przygotowaliśmy aplikację do obsługi sklepu elektronicznego "Elektrosklep", która umożliwia użytkownikowi zalogowanie się na swoje konto lub rozpoczęcie zakupów jako gość, a następnie po wyborze odpowiedniej kategorii produktu (laptopy, tablety lub smartfony) dodanie wybranych przez siebie produktów do koszyka. Zakupy mogą być dalej kontynuowane. Aplikacja umożliwia również obsługę sklepu z konta administratora. Jako admin można dodać lub usunąć wybrane produkty z magazynu oraz zaktualizować dane produktów.

Podział ról

Katarzyna Wacławik: cała logika aplikacji, czyli następujące klasy: Produkt, Laptop, Smartfon, Tablet, Koszyk, Magazyn, Zamowienie, Platnosc, Klient, Program; własne wyjątki: InvalidPaymentException, ProduktNotFoundException. Zapis do bazy danych. Diagram UML.

Aleksandra Rodzinka: okna wraz z obsługą: MainWindow, OknoLogowania, Specyfikacja, ListaProduktow, funkcjonalność zapisu i odczytu z pliku, testy jednostkowe.

Wiktoria Papiz: okna wraz z obsługą: Admin_ListaProduktów, DaneProduktu, Koszyk_Produktów, KategoriaProduktu; dodanie dźwięku i gifu, zaokrąglone rogi i gradient na przyciskach, przygotowanie instrukcji obsługi.

Instrukcja obsługi

Przed uruchomieniem logiki aplikacji należy utworzyć lokalnie bazę danych o nazwie Elektrosklep.

Po uruchomieniu aplikacji należy zdecydować, czy wykorzystana ma zostać funkcjonalność przeznaczona dla użytkownika (klienta), czy administratora sklepu. W pierwszym przypadku mamy do wyboru przyciski: kontynuowanie zakupów jako gość, co nie wymaga logowania, informację o rabacie (wyświetlaną w MessageBoxie) i logowanie do systemu. Ostatnia opcja przenosi nas do okna logowania, z którego można wrócić z użyciem przycisku Powrót lub w odpowiednich okienkach (TextBoxach) wpisać email i hasło (dostępne opcje to: email1 i password1; email2 i password2; email3 i password3; email4 i password4, są one zapisane w słowniku). Podanie błędnych danych powoduje wyświetlenie MessageBoxa z adekwatną informacją. Po zalogowaniu lub kontynuacji jako gość jesteśmy przenoszani do okna z wyborem kategorii produktu (Laptopy, Tablety lub Smartfony). Ewentualnie można skorzystać z przycisku na górze ekranu i przejść od razu do koszyka. Wybór kategorii definiuje nam jednoznacznie wygląd następnego okna z listą dostępnych produktów danej grupy. Klient może przejrzeć tę listę, posortować produkty alfabetycznie i cenowo lub wrócić do poprzedniego okna. Kliknięcie któregoś z produktów pozwala nam zobaczyć jego pełną specyfikację (produkty różnych kategorii mają różne dane specyfikacyjne, co zostało odpowiednio dostosowane). Po zapoznaniu się z danymi klient może zrezygnować z tego zakupu poprzez powrót do poprzedniego okna, dodać produkt do koszyka (wywołuje to wyświetlenie się odpowiedniego MessageBoxa, animacji GIF z koszykiem i odtworzeniu melodii), a następnie przejść do koszyka za pomocą trzeciego z przycisków. Wtedy wyświetlane jest okno z zawartością koszyka – na liście pokazują się produkty, które klient dodawał do koszyka podczas całych zakupów (nawet gdy dodawał więcej produktów jednej kategorii lub też zmieniał kategorie produktów). W koszyku widoczne jest także podsumowanie płatności, a więc suma cen wybranych produktów, ewentualna wartość przyznanego rabatu i cena ostateczna stanowiąca różnicę pomiędzy kwotą

pierwotną a rabatem. Klikając na konkretny produkt znajdujący się na liście, klient może zwiększyć jego liczbę (sklonować go) z użyciem przygotowanego do tego celu przycisku lub usunąć wybrany produkt (można usunąć więcej zaznaczonych produktów). Oba działania powodują odświeżenie listy produktów w koszyku. Jeśli któryś z tych dwóch przycisków zostanie kliknięty, a żaden produkt nie jest zaznaczony, pojawia się MessageBox z odpowiednią informacją. Klient ma tutaj także możliwość kontynuacji zakupów, dzięki czemu jest przenoszony z powrotem do wyboru kategorii produktu. Ostatecznie użytkownik kończy zakupy przy otwartym oknie koszyka, używając przycisku Zakończ zakupy, który zamyka wszystkie otwarte okna i kończy działanie aplikacji.

Drugą możliwością jest użycie aplikacji w roli administratora. W oknie głównym należy wybrać zatem, znajdujący się w lewym górnym rogu, DockPanel z rozwijaną listą opcji wyboru. Menuitem "Wyjdź" zamyka okno główne, natomiast "Zaloguj" przenosi nas znowu do okna logowania, lecz zawiera ono pewne zmiany m.in. w etykietach znajdujących się nad TextBoxami. Aby poprawnie zalogować się jako administrator należy wpisać następujące dane: login - "admin1" i hasło - "admin123". Jeśli nie pojawiły się błędy, jesteśmy przenoszni do okna wyboru kategorii (jest ono takie samo jak poprzednio, lecz przycisk przeniesienia się do koszyka został schowany, ponieważ w tym momencie jego użycie nie miałoby sensu - administrator nie kupuje produktów). Po wybraniu kategorii pojawia się okno z listą produktów - podobne do tego, które widział użytkownik, jednak zostało stworzone osobno dla administratora, gdyż zawiera inne funkcjonalności. Lista produktów znów pobierana jest z pliku .xml, przechowującego magazyn dostępnych urządzeń. Oprócz powrotu do poprzedniego okna oraz identycznych sortowań, z czego mógł skorzystać użytkownik, administrator ma tutaj prawo do kontroli stanu produktów znajdujących się w magazynie. Klikając przycisk "Usuń z magazynu" zaznaczony produkt jest trwale

usuwany z danych w pliku .xml i dzięki temu usuwany na liście dostępnego asortymentu. Po wybraniu konkretnego urządzenia można także zaktualizować jego dane za pomocą odpowiedniego przycisku, który przenosi nas do okna ze specyfikacją tego produktu. Jest to inne okno niż w przypadku klienta, ponieważ dane wyświetlane w polach tekstowych są edytowalne, a po zatwierdzeniu zmian rzeczywiście jest to zapisywane w pliku .xml i aktualizowane na liście. Istnieje jeszcze opcja dodania nowego produktu do magazynu, co odbywa się w oknie takim jak poprzednio, jednakże w tej sytuacji wszystkie pola są początkowo puste i należy je uzupełnić. Po zatwierdzeniu nowy produkt zostaje także dodany do pliku .xml i dołożony do listy wyświetlanych urządzeń. Opisywane okno obsługujące wyżej wymienione działania dla każdej kategorii produktów zawiera właściwe dla niej pola, które należy uzupełnić lub skorygować. Gdy administrator będzie chciał zakończyć pracę, jest przygotowany do tego celu przycisk na dole okna – kończy on działanie aplikacji i zamyka wszystkie okna.

Opis funkcjonalności

Logika aplikacji

Produkt.cs

Klasa abstrakcyjna (bazowa dla klas Smartfon, Tablet, Laptop).

Klasa zawierająca właściwości: **Nazwa**, **Cena**, **Opis**, **Id**, **Rabat**.

Id - generowany automatycznie w oparciu o statyczną zmienną, atrybut do serializacji XML, klucz główny w bazie danych.

ostatnieId - zmienna statyczna do śledzenia ostatniego użytego unikalnego identyfikatora produktu..

Klasa implementuje trzy interfejsy:

- **Comparable** - umożliwia porównywanie produktów według ceny (**int CompareTo(Produkt other)**).
- **IComparable** - pozwala na porównywanie dwóch produktów według ich unikalnego ID (**bool Equals(Produkt other)**).
- **ICloneable** - umożliwia klonowanie produktów (**object Clone()**).

Konstruktory:

- **Produkt(string nazwa, double cena, string opis)** - przypisuje wartości przekazane w parametrach i generuje Id.
- **Produkt()** - domyślny bez parametrów.

Metody:

- `double CenaPoRabacie()` - oblicza cenę produktu po uwzględnieniu rabatu.
- `void ZapiszDoPliku(string sciezka)` - zapisuje obiekty do pliku XML, wykorzystuje klasę `XMLSerializer`, obsługuje wyjątki.
- `override string ToString()` - reprezentuje produkt poprzez wypisanie nazwy, ceny i opisu.
- `abstract void WyświetlSzczegóły()` - abstrakcyjna metoda, która została zaimplementowana w klasach dziedziczących, służy do wyświetlania specyficznych danych dla danego typu urządzenia.

Smartfon.cs - klasa dziedzicząca po klasie Produkt

Klasa pochodna, dziedzicząca po klasie `Produkt`, służy do prezentowania właściwości specyficznych dla smartfonów. Nadpisuje metodę `WyświetlSzczegóły()`.

Zawiera właściwości: `PrzekatnaEkranu`, `Aparat`, `PojemnoscBaterii`, `Procesor`.

Konstruktory:

- `Smartfon(string nazwa, double cena, string opis, double przekatnaEkranu, string aparat, int pojemnoscBaterii, string procesor) : base(nazwa, cena, opis)` - wywołuje konstruktor klasy bazowej i ustawia podstawowe właściwości.
- `Smartfon()` - domyślny bez parametrów.

Tablet.cs – klasa dziedzicząca po klasie Produkt

Klasa pochodna, dziedzicząca po klasie **Produkt**, służy do prezentowania właściwości specyficznych dla tabletów. Nadpisuje metodę **WyswietlSzczegoly()**.

Zawiera właściwości: **Wyswietlacz**, **SystemOperacyjny**, **CzyRysik**.

Konstruktory:

- **Tablet(string nazwa, double cena, string opis, double wyswietlacz, string systemOperacyjny, bool czyRysik):**
base(nazwa, cena, opis) – wywołuje konstruktor klasy bazowej i ustawia podstawowe właściwości
- **Tablet()** – domyślny bez parametrów.

Laptop.cs – klasa dziedzicząca po klasie Produkt

Klasa pochodna, dziedzicząca po klasie **Produkt**, służy do prezentowania właściwości specyficznych dla laptopów. Nadpisuje metodę **WyswietlSzczegoly()**.

Zawiera właściwości: **Procesor**, **RAM**, **Dysk**, **KartaGraficzna**.

Konstruktory:

- **Laptop(string nazwa, double cena, string opis, string procesor, int ram, int dysk, string kartaGraficzna) :**
base(nazwa, cena, opis) – wywołuje konstruktor klasy bazowej i ustawia podstawowe właściwości
- **Laptop()** – domyślny bez parametrów

Magazyn.cs

Klasa ta jest częścią aplikacji zarządzającej magazynem produktów. Posiada metody umożliwiające dodawanie, usuwanie, aktualizowanie i sprawdzanie produktów w magazynie.

Zawiera właściwość: **produkty**, która przechowuje produkty w magazynie.

Konstruktor:

- **Magazyn ()** - inicjalizuje pustą listę produktów.

Metody:

- **DodajProdukt(Produkt produkt, int ilosc)** - dodaje nowy produkt do magazynu lub aktualizuje ilość istniejącego produktu, jeśli już znajduje się w magazynie. Jeśli ilość jest mniejsza niż 1, wyświetla komunikat o błędzie.
- **UsunProdukt(int id)** - usuwa produkt z magazynu na podstawie identyfikatora produktu. Jeśli produkt nie istnieje, wyświetla komunikat o błędzie.
- **ZaktualizujIloscProduktu(int id, int nowaIlosc)** - zmienia ilość produktu na określoną wartość. Jeśli produkt nie istnieje lub ilość jest nieprawidłowa, wyświetla komunikat o błędzie.
- **SprawdzDostepnosc(int id)** - sprawdza, czy produkt o danym ID jest dostępny w magazynie. Jeśli produkt nie istnieje, rzuca wyjątek **ProduktNotFoundException**.
- **WyswietlProdukty()** - wyświetla listę wszystkich produktów w magazynie, pokazując ich nazwę i cenę. Jeśli magazyn jest pusty, wyświetla odpowiedni komunikat.

- **PobierzProduktyZKategorii(string kategoria)** - zwraca listę produktów należących do danej kategorii, filtrując je na podstawie ich typu.
- **ZapiszDoXml(string sciezka)** - zapisuje zawartość magazynu do pliku XML, serializując produkty przy użyciu **XmlSerializer**. Obsługuje wyjątki związane z błędami zapisu.
- **OdczytXml(string sciezka)** - wczytuje magazyn z pliku XML, deserializując go do obiektu typu **Magazyn**. Obsługuje wyjątki związane z błędami odczytu.
- **ZapiszDoBazy()** - zapisuje produkty w bazie danych przy użyciu kontekstu **MagazynContext** (EF Core). Używa łańcucha połączenia z konfiguracji (plik **App.config**).

Klasa MagazynContext - Jest to klasa kontekstu bazy danych, używająca Entity Framework. Zawiera zbiór **Produkty** reprezentujący tabelę produktów w bazie danych.

Koszyk.cs

Zawiera właściwości:

- **produkty** - jest to kolekcja produktów w koszyku, reprezentowana przez **ObservableCollection<Produkt>**. Ta kolekcja zapewnia, że wszystkie zmiany w jej zawartości (np. dodawanie lub usuwanie produktów) będą automatycznie aktualizowane w interfejsie użytkownika, jeśli ten jest związany z tą kolekcją.
- **rabatZastosowany** - to prywatna zmienna logiczna, która przechowuje informację, czy rabat został już zastosowany do produktów w koszyku.

Konstruktor:

- `Koszyk()`: Inicjalizuje kolekcję produktów jako pustą oraz ustawia `rabatZastosowany` na `false` (rabat nie został jeszcze zastosowany).

Metody:

- `DodajProdukt(Produkt produkt)` - dodaje produkt do kolekcji `produkty`.
- `UsunProdukt(Produkt produkt)` - usuwa wskazany produkt z kolekcji `produkty`.
- `ZastosujRabat()` - sprawdza, czy w koszyku znajdują się produkty z każdej z trzech kategorii: laptop, tablet i smartfon. Jeśli tak, na wszystkie produkty w koszyku zostanie zastosowany rabat w wysokości 10%. Po zastosowaniu rabatu, zmienna `rabatZastosowany` jest ustawiona na `true`, a użytkownik otrzymuje informację o zastosowanym rabacie.
- `CalkowitaCenaPrzedRabatem()` - oblicza całkowitą cenę produktów w koszyku przed zastosowaniem rabatu. Suma cen wszystkich produktów w koszyku.
- `ObliczCalkowitaCene()` - oblicza całkowitą cenę produktów w koszyku po uwzględnieniu rabatu. Metoda ta sumuje ceny produktów po rabacie.
- `ObliczWartoscRabatu()` - oblicza łączną wartość rabatu, czyli różnicę między ceną przed rabatem a ceną po rabacie dla każdego produktu w koszyku.
- `WyswietlKoszyk()` - wyświetla wszystkie produkty w koszyku za pomocą metody `ToString()` dla każdego produktu. Dodatkowo, jeżeli rabat został zastosowany, wyświetlana jest całkowita cena po rabacie oraz wartość

rabatu. Jeśli rabat nie został zastosowany, wyświetlana jest tylko całkowita cena produktów.

Klient.cs

Klasa ta reprezentuje klienta, przechowując dane osobowe klienta, jego adresy, oraz historię zamówień.

Zawiera właściwości: `Id`, `Imie`, `Nazwisko`, `Email`, `NumerTelefonu`, `Adresy`, `HistoriaZmian`.

Konstruktor:

- `Klient(int id, string imie, string nazwisko, string email, string numerTelefonu)` - inicjalizuje obiekt klienta z unikalnym identyfikatorem (`id`), imieniem, nazwiskiem, e-mailem oraz numerem telefonu. Domyślnie lista adresów (`Adresy`) i lista historii zamówień (`HistoriaZamowien`) są puste.

Metody:

- `DodajAdres(string adres)` - dodaje nowy adres do listy adresów klienta. Po dodaniu wyświetlany jest komunikat potwierdzający dodanie adresu.
- `DodajZamowienie(Zamowienie zamowienie)` - dodaje nowe zamówienie do historii zamówień klienta. Po dodaniu wyświetlany jest komunikat o dodaniu zamówienia do historii.
- `WyswietlHistorieZamowien()` - wyświetla historię zamówień klienta. Jeśli klient nie ma żadnych zamówień, wyświetlany jest komunikat o braku historii zamówień. Jeśli historia istnieje, każdemu zamówieniu przypisane są: ID zamówienia, Data złożenia zamówienia, Status zamówienia.

Zamowienie.cs

Klasa ta reprezentuje zamówienie złożone przez klienta w systemie. Zawiera informacje o zamówionych produktach, statusie zamówienia, oraz umożliwia obliczenie całkowitej ceny zamówienia.

Zawiera właściwości: **Id**, **Klient**, **Produkty**, **Status**, **DataZamowienia**.

Konstruktor:

- **Zamowienie(int id, Klient klient)** - inicjalizuje zamówienie z unikalnym identyfikatorem (**id**) oraz klientem (**klient**). Domyślnie przypisuje status "Nowe" oraz datę złożenia zamówienia jako bieżącą datę. Lista produktów jest początkowo pusta.

Metody:

- **DodajProdukt(Produkt produkt)** - dodaje produkt do listy produktów w zamówieniu. Wyświetla komunikat, że produkt został dodany.
- **ObliczCalkowitaCene()** - oblicza całkowitą cenę zamówienia, uwzględniając ceny produktów po rabacie. Sumuje ceny wszystkich produktów po rabacie i zwraca łączną kwotę.
- **ZaktualizujStatus(string nowyStatus)** - umożliwia zmianę statusu zamówienia na nowy status. Wyświetla komunikat o zaktualizowaniu statusu.
- **WyswietlSzczegoly()** - wyświetla szczegóły zamówienia.

Platnosc.cs

Klasa ta reprezentuje proces płatności za zamówienie w systemie. Zawiera informacje o płatności, metodzie płatności, statusie płatności oraz umożliwia dokonanie płatności, anulowanie jej, oraz wyświetlanie szczegółów płatności.

Zawiera właściwości: `Id`, `Zamowienie`, `Kwota`, `MetodaPlatnosci`, `Status`, `DataPlatnosci`.

Konstruktor:

- `Platnosc(Zamowienie zamowienie, string metodaPlatnosci)` - inicjalizuje płatność, ustawiając związane zamówienie oraz metodę płatności. Kwota do zapłaty jest obliczana na podstawie całkowitej ceny zamówienia (`Zamowienie.ObliczCalkowitaCene()`). Domyślnie status płatności jest ustawiony na "Oczekuje", a data płatności na `DateTime.MinValue` (czyli brak daty przed dokonaniem płatności).

Metody:

- `DokonajPlatnosci()` - symuluje dokonanie płatności. Jeśli status płatności jest "Oczekuje", płatność jest uznawana za zrealizowaną, status zmienia się na "Zrealizowana", a data płatności jest ustawiana na bieżącą datę. Jeśli płatność jest już zrealizowana lub ma inny status, wyświetlany jest komunikat o błędzie.
- `AnulujPlatnosc()` - umożliwia anulowanie płatności, jeżeli status jest "Oczekuje". W takim przypadku płatność jest anulowana, a status zmienia się na "Nieudana", a data płatności zostaje ustawiona na `DateTime.MinValue`.

Jeśli płatność została już zrealizowana lub anulowana, wyświetlany jest komunikat o niemożliwości anulowania.

- `WyświetlSzczegółyPłatności()` - wyświetla szczegóły płatności.

InvalidPaymentException.cs

Klasa ta jest specjalnym wyjątkiem, który rozszerza klasę `Exception` i jest używana do sygnalizowania błędów związanych z płatnościami, gdy wystąpią problemy w procesie płatności.

Konstruktory:

- `InvalidPaymentException()` - jest to konstruktor domyślny, który nie przyjmuje żadnych argumentów. Ustawia domyślny komunikat o błędzie: `"Płatność nie została zrealizowana z powodu błędu."`.
- `InvalidPaymentException(string message)` - konstruktor, który przyjmuje własny komunikat o błędzie jako argument (`message`). Komunikat ten jest przekazywany do klasy bazowej `Exception` w celu jego ustawienia.
- `InvalidPaymentException(string message, Exception inner)` - konstruktor, który przyjmuje dwa argumenty: komunikat o błędzie (`message`) oraz wyjątek wewnętrzny (`inner`). Ten konstruktor pozwala na przekazywanie dodatkowych informacji o błędzie, a także umożliwia zagnieżdżanie wyjątków, czyli przechwytywanie pierwotnego wyjątku, który może być powodem bieżącego problemu.

ProduktNotFoundException.cs

Klasa ta jest niestandardowym wyjątkiem, który dziedziczy po klasie `Exception` i jest używana w sytuacjach, gdy nie znaleziono produktu w systemie (np. w

magazynie). Dzięki tej klasie można lepiej obsługiwać sytuacje, w których wymagany produkt nie istnieje w danym kontekście.

Konstruktory:

- `ProduktNotFoundException()` - konstruktor domyślny, który nie przyjmuje żadnych argumentów. Ustawia domyślny komunikat o błędzie: `"Produkt nie został znaleziony w magazynie."`. Jest to komunikat, który jest przekazywany do klasy bazowej `Exception`.
- `ProduktNotFoundException(string message)` - konstruktor, który przyjmuje własny komunikat o błędzie jako argument (`message`). Dzięki temu, możesz dostarczyć bardziej szczegółowy komunikat o błędzie, dostosowany do kontekstu.
- `ProduktNotFoundException(string message, Exception inner)` - konstruktor, który przyjmuje dwa argumenty: komunikat o błędzie (`message`) oraz wyjątek wewnętrzny (`inner`). Ten konstruktor jest szczególnie przydatny, gdy chcesz zagnieździć wyjątek, który może być przyczyną obecnego problemu. Przekazanie wyjątku wewnętrznego pozwala na lepszą diagnostykę i śledzenie przyczyn błędu.

Gui

MainWindow.xaml



Okno główne jest punktem wejścia do aplikacji. Użytkownicy mogą:

- Przenieść się do okna logowania jako administrator lub użytkownik.
- Przenieść się do okna z ofertą sklepu jako gość.
- Uzyskać informacje o rabatach.

Elementy interfejsu:

- Etykiety (Label):
 - **LblNazwaSklepu**: Etykieta wyświetlająca powitanie w aplikacji: „Witamy w Elektrosklepie!”.
- Przyciski (Button):

- **BtnZaloguj**: Przycisk, który po kliknięciu przenosi użytkownika do okna logowania. Ma stylizowane tło gradientowe w kolorach różu, jak wszystkie inne przyciski w aplikacji.
- **BtnRabat**: Przycisk, który wyświetla informację o dostępnych rabatach.
- **BtnGosc**: Przycisk, który umożliwia kontynuowanie jako gość i przenosi użytkownika do widoku kategorii produktów.
- Obrazy:
 - Obrazek 1: Ikona laptopa wyświetlana w prawym górnym rogu.
 - Obrazek 2: Ikona smartfona wyświetlana na środku okna, zaraz pod **LblNazwaSklepu**.
 - Obrazek 3: Ikona tabletu na dole po lewej stronie ekranu.
 - Obrazek 4: Ikona sklepu wyświetlana nad przyciskiem **BtnZaloguj**.
- MenuItem Header: Przycisk "Admin" z dwoma opcjami:
 - Zaloguj: Po kliknięciu otwiera okno logowania administracyjnego.
 - Wyjdź: Po kliknięciu zamyka aplikację.

Logika okna:

Konstruktor klasy **MainWindow**:

1. Ładuje interfejs użytkownika.
2. Tworzy pusty koszyk, który będzie używany przez użytkownika do dodawania produktów.
3. Wczytuje dane o produktach z pliku XML i zapisuje je w magazynie.

Metoda **btnZalogujClick**:

Otwiera okno logowania dla użytkownika i ukrywa bieżące okno.

1. Tworzy nowe okno logowania (**OknoLogowania**) z parametrami: przekazuje koszyk oraz informację o typie użytkownika ("user").
2. Otwiera to nowe okno logowania za pomocą metody **Show()**.
3. Ukrywa bieżące okno (główne okno) za pomocą **this.Hide()**.

Metoda **btnGoscClick**:

Otwiera menu kategorii produktów dla gościa i ukrywa bieżące okno.

1. Tworzy nowe okno, które przedstawia kategorie produktów (**Kategoria_produktu**), z parametrami: przekazuje koszyk oraz informację o typie użytkownika ("user").
2. Otwiera to okno z kategoriami produktów za pomocą metody **Show()**.
3. Ukrywa bieżące okno (główne okno) za pomocą **this.Hide()**.

Metoda **btnRabatClick**:

Wyświetla komunikat o rabacie przy zakupie produktów z trzech kategorii.

1. Wyświetla okno komunikatu (**MessageBox**), które informuje o rabacie na zamówienie.
2. Komunikat mówi, że przy zakupie produktów z każdej z trzech kategorii użytkownik otrzymuje rabat w wysokości 10% na całe zamówienie.
3. Okno komunikatu jest typu **Information**, a po kliknięciu przycisku "OK", użytkownik wraca do aplikacji.

Metoda **MenuItem_Zaloguj_Click**:

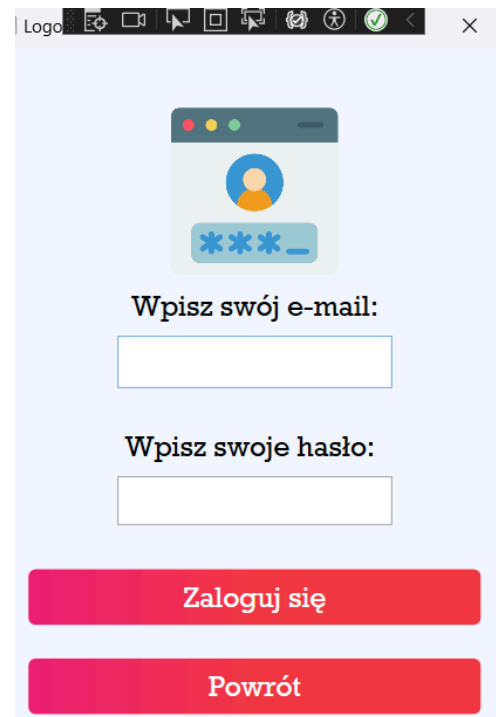
Otwiera okno logowania dla administratora i ukrywa bieżące okno.

1. Tworzy nowe okno logowania (**OknoLogowania**) z parametrami: przekazuje koszyk oraz informację o trybie administracyjnym ("admin").
2. Otwiera to nowe okno logowania za pomocą metody **Show()**.
3. Ukrywa bieżące okno (główne okno) za pomocą **this.Hide()**.

Metoda **MenuItem_Wyjdz_Click**: Zamyka aplikację.

OknoLogowania.xaml

OknoLogowania jest interfejsem odpowiedzialnym za obsługę procesu logowania zarówno dla użytkowników, jak i administratora. Posiada dynamicznie zmieniające się elementy interfejsu w zależności od rodzaju osoby logującej się (użytkownik lub administrator). Jego wygląd i funkcjonalność dostosowane są do wymagań obu grup. Okno to umożliwia wprowadzenie danych logowania oraz ich weryfikację, a następnie przekierowanie do odpowiedniego okna w zależności od wyniku weryfikacji.



Elementy interfejsu:

- Etykiety (Label)
 - **LblLogin** – Wyświetla dynamiczny tekst: „Wpisz swój e-mail” (dla użytkownika) lub „Wpisz login administratora” (dla administratora).
 - **LblHasło** – Wyświetla tekst z prośbą o wprowadzenie hasła.
- Pola wejściowe (TextBox)

- **txtEmail** – Pole tekstowe do wpisania e-maila (lub loginu administratora).
- **txtHaslo** – Pole typu PasswordBox do wprowadzenia hasła, ukrywające wpisywane znaki.
- Przyciski (Button)
 - **BtnZaloguj** – Przycisk inicjujący proces logowania po wprowadzeniu danych.
 - **BtnPowrot** – Przycisk umożliwiający powrót do głównego okna aplikacji.
- Obrazek
 - Obrazek symbolizujący proces logowania, wyświetlany w górnej części okna.

Logika okna:

Konstruktor klasy **OknoLogowania**:

1. Ładuje interfejs użytkownika.
2. Ustawia wartości w etykietach **LblLogin** i **LblHaslo**.
3. Inicjalizuje słownik **daneLogowania** przechowujący dane logowania dla użytkowników.
4. Przypisuje wartości pól **_koszyk** i **osoba**.
 - a. **Koszyk** (obiekt klasy **Koszyk**) – umożliwia przeniesienie danych koszyka pomiędzy oknami.
 - b. **kto** (string) – określa, czy logujący się jest użytkownikiem ("**user**") czy administratorem ("**admin**"). Na tej podstawie dynamicznie ustawiane są teksty etykiet w interfejsie.

Metoda `btnZaloguj_Click`:

Służy do weryfikacji poprawności danych logowania.

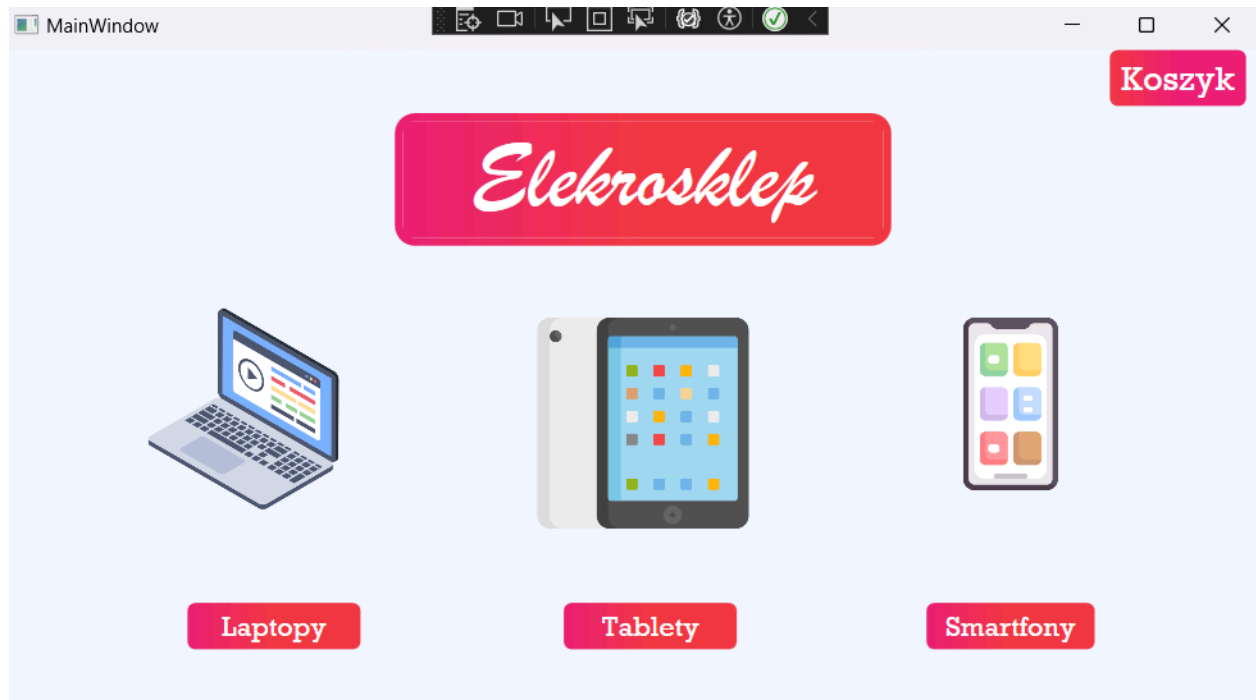
1. Pobiera dane z pól `txtEmail` i `txtHaslo`.
2. Sprawdza, czy użytkownik to zwykły użytkownik (`osoba == "user"`) czy administrator.
3. Dla użytkowników: Weryfikuje, czy podany e-mail istnieje w słowniku `daneLogowania` i czy hasło pasuje do tego e-maila.
 - a. W przypadku poprawnych danych otwiera okno `Kategoria_produktu` w trybie `"user"`.
 - b. W przypadku błędnych danych wyświetla komunikat o błędzie logowania.
4. Dla administratorów: Sprawdza, czy login i hasło są równe `"admin1"` i `"admin123"`.
 - a. W przypadku poprawnych danych otwiera okno `Kategoria_produktu` w trybie `"admin"`.
 - b. W przypadku błędnych danych wyświetla komunikat o błędzie logowania.

Metoda `btnPowrot_Click`:

Służy do obsługi powrotu do okna głównego.

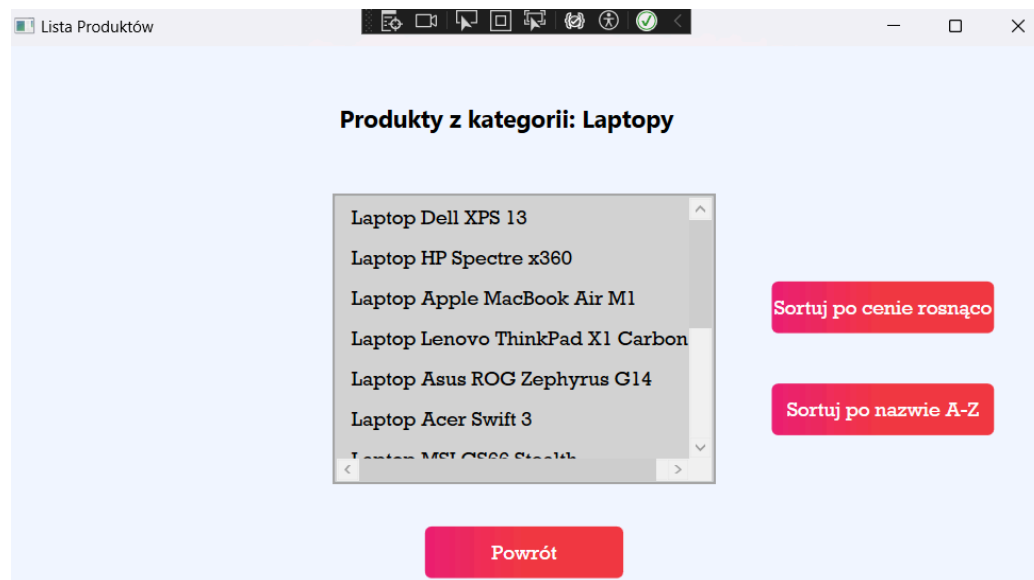
1. Tworzy instancję klasy `MainWindow`.
2. Otwiera okno główne.
3. Zamyka bieżące okno logowania (`this.Close()`).

Kategoria_Produktu.xaml



Okno to przechowuje istniejący koszyk klienta, tak aby dane w nim były zachowywane w ciągu zakupów, a także w konstruktorze wymaga informacji, kto korzysta w tym momencie z aplikacji. Dla administratora chowa przycisk z przejściem do koszyka, który dla usera przeniesie go do okna **Koszyk_produktów**, a obecne okno zamknie. Obecne okno zawiera obrazki z dostępnymi trzema kategoriami produktów, a pod nimi odpowiednie przyciski, które przekierowują użytkownika do następnego okna (dla admina jest to **Admin_ListaProduktów**, a dla usera - **ListProduktów**).

ListaProduktow.xaml



Okno służy do wyświetlania listy produktów (odczytanych z pliku xml) należących do określonej kategorii (wybranej w poprzednim oknie) oraz umożliwia interakcję z użytkownikiem w celu sortowania listy lub wyboru produktu do wyświetlenia szczegółowej specyfikacji.

Elementy interfejsu:

- Etykieta (**Label**):
 - Wyświetla nazwę kategorii produktów.
 - Dynamicznie zmienia swoją zawartość w zależności od wybranej kategorii.
- Lista (**ListBox**):
 - Wyświetla produkty należące do wybranej kategorii.
 - Elementy listy zmieniają kolor podczas najeżdżania na nie myszką (animacja **ColorAnimation**).
- Przyciski (**Button**):
 - **Powrót**: Powraca do okna kategorii produktów.

- **Sortuj po nazwie A-Z**: Sortuje listę produktów alfabetycznie.
- **Sortuj po cenie rosnąco**: Sortuje listę produktów według ceny.

Logika okna:

Konstruktor klasy **ListaProduktow**:

1. Pobiera nazwę kategorii i obiekt klasy **Koszyk**.
2. Ustawia nagłówek etykiety z nazwą kategorii.
3. Pobiera listę produktów dla danej kategorii za pomocą metody **PobierzProdukty** i ustawia ją jako źródło danych dla **ListBox**.

Metoda **PobierzProdukty**:

Metoda do pobierania produktów z pliku XML na podstawie wybranej kategorii (Laptopy, Tablety, Smartfony).

1. Wczytuje dane z pliku XML **magazyn.xml**.
2. W zależności od kategorii (np. **Laptopy**, **Tablety**, **Smartfony**) filtruje produkty i zwraca ich nazwy w formie listy.

Obsługa Wyboru Produktu (**listBoxProdukty_SelectionChanged**):

Obsługuje zmianę wyboru produktu w **listBoxProdukty**, ładuje szczegóły produktu i otwiera okno specyfikacji.

1. Po wyborze produktu użytkownikowi otwierane jest nowe okno **Specyfikacja**, w którym wyświetlana jest szczegółowa charakterystyka wybranego produktu.

2. Dane o produkcie (np. nazwa, cena, opis, dodatkowePola - słownik z dodatkowymi cechami w zależności od tego, jaka jest kategoria wybranego produktu) są pobierane z pliku XML `magazyn.xml`.

Metoda (`btnPowrot_Click`): Powraca do okna kategorii produktów (`Kategoria_produktu`).

Metoda `SortujPoNazwie`:

Sortuje listę produktów alfabetycznie po nazwie.

1. Lista produktów (`lista`) jest sortowana za pomocą metody `Sort` z wykorzystaniem porównania `x.CompareTo(y)`.
2. Porównanie odbywa się między elementami listy, co zapewnia alfabetyczną kolejność (A-Z).

Metoda `btnSortujNazwaClick`:

Obsługuje kliknięcie przycisku "Sortuj po nazwie" i wywołuje metodę sortującą po nazwie.

1. Wywołuje metodę `SortujPoNazwie` na liście produktów `p` (lokalna lista przechowująca nazwy produktów) - Lista produktów zostaje posortowana w porządku alfabetycznym (A-Z).
2. Odświeża źródło danych kontrolki `ListBox`:
 - a. Ustawa `listBoxProdukty.ItemsSource` na `null`, co powoduje wyczyszczenie widoku kontrolki.
 - b. Następnie `listBoxProdukty.ItemsSource` ustawia ponownie na posortowaną listę `p`, aby zaktualizować wyświetlaną zawartość.

Metoda **SortujPoCenie**:

Sortuje produkty w liście po cenie, przy czym najpierw konwertuje je na obiekty **Produkt**, a potem sortuje je.

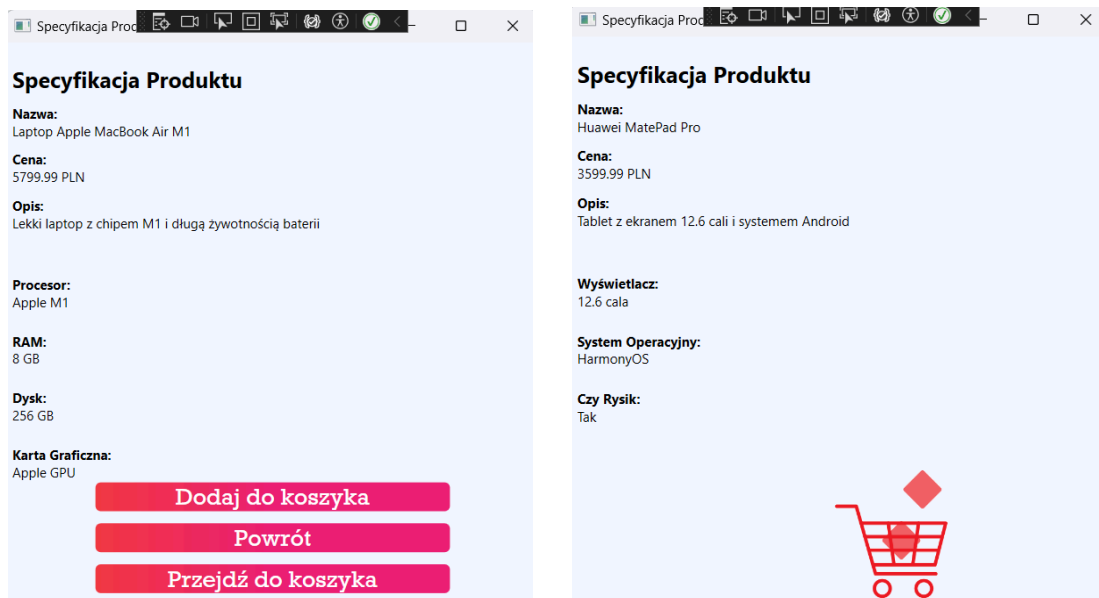
1. Tworzy nową listę obiektów typu **Produkt** o nazwie **p1**, która będzie przechowywała produkty z pełną informacją (nie tylko nazwy).
2. Wczytuje dane magazynu produktów z pliku XML przy użyciu metody **Magazyn.OdczytXml("magazyn.xml")**. Odczytane dane są zapisane w obiekcie **odczytanyMagazyn**.
3. Iteruje przez wszystkie nazwy produktów znajdujące się w przekazanej do metody liście **lista**.
4. Dla każdej nazwy produktu szuka odpowiadającego jej obiektu **Produkt** w odczytanym magazynie. Jeśli znajdzie taki produkt, dodaje go do listy **p1**.
5. Sortuje listę **p1** (zawierającą obiekty **Produkt**) za pomocą metody **Sort()**. Klasa **Produkt** posiada zdefiniowaną logikę sortowania (np. poprzez implementację interfejsu **IComparable**), która sortuje obiekty według ich ceny.
6. Czyści oryginalną listę **lista**, która przechowuje wyłącznie nazwy produktów.
7. Iteruje przez posortowaną listę **p1** i dodaje nazwy produktów do listy **lista**, odtwarzając ją w posortowanej kolejności.

Metoda **btnSortujCenaClick**:

Obsługuje kliknięcie przycisku "Sortuj po cenie" i wywołuje metodę sortującą po cenie.

1. Wywołuje metodę **SortujPoCenie**, przekazując do niej listę nazw produktów (**p**), która została wcześniej załadowana do tej listy w konstruktorze okna. Metoda ta sortuje produkty według ceny w sposób rosnący.
2. Odświeża zawartość listy widocznej w interfejsie (ListBox):
 - a. Najpierw ustawia **ItemsSource** kontrolki **listBoxProdukty** na **null**. Ten krok usuwa obecnie wyświetlane elementy z ListBoxa.
 - b. Następnie ponownie przypisuje posortowaną listę **p** jako źródło danych dla kontrolki **listBoxProdukty**, co powoduje, że produkty w ListBoxie są wyświetlane w nowej, posortowanej kolejności.

Specyfikacja.xaml



To okno wyświetla szczegóły produktu, takie jak jego nazwa, cena i opis. Użytkownik może zobaczyć również dodatkowe informacje o produkcie, w zależności od kategorii. Okno to umożliwia skorzystanie z trzech przycisków : dodania produktu do koszyka, powrotu do okna **ListaProduktow** oraz przejścia do koszyka.

Elementy interfejsu:

- Etykiety (TextBlock):
 - TextBlock 1: „Specyfikacja Produktu” – Nagłówek wyświetlający tytuł specyfikacji, wyróżniony pogrubioną czcionką o rozmiarze 20.
 - TextBlock 2: „Nazwa:” – Etykieta informująca o polu nazwy produktu, używa pogrubionego stylu.
 - TextBlock 3: **lblNazwa** – Etykieta wyświetlająca nazwę produktu.
 - TextBlock 4: „Cena:” – Etykieta informująca o polu ceny produktu, używa pogrubionego stylu.
 - TextBlock 5: **lblCena** – Etykieta wyświetlająca cenę produktu w formacie „x PLN”.
 - TextBlock 6: „Opis:” – Etykieta informująca o polu opisu produktu, używa pogrubionego stylu.
 - TextBlock 7: **lblOpis** – Etykieta wyświetlająca opis produktu.
- StackPanel (Dodatkowe Pola):
 - stackDodatkowePola: Panel zawierający dodatkowe informacje o produkcie (np. cechy). Dla każdego dodatkowego pola utworzona jest para etykiet, gdzie pierwsza zawiera nazwę pola, a druga wartość.
- Przyciski (Button):
 - **BtnDodaj**: Przycisk „Dodaj do koszyka” wyświetlany na dole ekranu. Po kliknięciu produkt zostaje dodany do koszyka, a przycisk staje się niewidoczny. Wykorzystywany jest również animowany obrazek oraz dźwięk, aby potwierdzić dodanie produktu.
 - **BtnPowrót**: Przycisk „Powrót” umożliwiający powrót do poprzedniego okna.
 - **BtnKoszyk**: Przycisk „Przejdź do koszyka”, który przenosi użytkownika do okna koszyka, gdzie można zobaczyć dodane produkty.

- Obrazek (Image):
 - gifImage: Obrazek (animowany GIF) symbolizujący dodanie produktu do koszyka. Początkowo jest niewidoczny, a po dodaniu produktu zostaje wyświetlony, aby pokazać użytkownikowi, że produkt trafił do koszyka.
- ScrollView:
 - ScrollView: Zawiera wszystkie szczegóły specyfikacji produktu. Umożliwia przewijanie zawartości, gdy nie mieści się ona na ekranie.

Logika okna:

Konstruktor klasy **Specyfikacja**:

Konstruktor klasy Specyfikacja inicjalizuje okno, ustawia odpowiednie wartości na podstawie przekazanych argumentów z okna **ListaProduktow** (nazwa, cena, opis, dodatkowe pola, wybrany produkt, koszyk).

1. **lblNazwa.Text**: Ustawia nazwę produktu na etykiecie **lblNazwa**.
2. **lblCena.Text**: Ustawia cenę produktu na etykiecie **lblCena** w formacie „cena PLN”.
3. **lblOpis.Text**: Ustawia opis produktu na etykiecie **lblOpis**.
4. Dodatkowe pola: Przechodzi przez kolekcję **dodatkowePola** (słownik) i dla każdego elementu dodaje etykietę zawierającą nazwę pola (np. „Kolor”) i wartość pola (np. „Czerwony”) do panelu **stackDodatkowePola**.
5. Animowany obrazek (GIF): Ustawia źródło animowanego GIFa na obrazek „shopping-cart-shopping.gif”, który zostanie wyświetlony później w celu potwierdzenia dodania produktu do koszyka.

Metoda **PlayAudioWithNAudio**:

1. `Task.Run`: Funkcja jest uruchamiana w tle, aby nie blokować głównego wątku UI.
2. Wykorzystanie biblioteki `NAudio`:
 - a. `AudioFileReader`: Otwiera plik audio (podany w `filePath`).
 - b. `WaveOutEvent`: Inicjalizuje urządzenie do odtwarzania dźwięku.
 - c. `outputDevice.Play()`: Odtwarza dźwięk.
 - d. Funkcja czeka, aż dźwięk się skończy, zanim kontynuuje działanie (blokując do tego momentu proces).

Metoda `btnDodajDoKoszykaClick`:

Jest wywoływana po kliknięciu przycisku „Dodaj do koszyka”. Dodaje produkt do koszyka i pokazuje potwierdzenie wizualne.

1. `_koszyk.DodajProdukt(_produkt)`: Dodaje aktualnie wybrany produkt (`_produkt`) do obiektu `Koszyk`.
2. `MessageBox`: Wyświetla okno dialogowe z informacją, że produkt został dodany do koszyka.
3. Zmiana widoczności przycisków:
 - a. Ukrywa przyciski Dodaj do koszyka, Powrót oraz Przejdź do koszyka.
 - b. Ustala widoczność animowanego GIFa (pokazuje go na 3 sekundy).
4. Odtwarzanie dźwięku: Wywołuje funkcję `PlayAudioWithNAudio`, aby odtworzyć dźwięk z pliku „sound.wav”.
5. Opóźnienie: Czeki przez 3 sekundy (używając `Task.Delay`), po czym przywraca pierwotną widoczność przycisków i ukrywa animowany GIF.

Metoda `btnPowrotClick`:

Funkcja wywoływana po kliknięciu przycisku „Powrót”. Zamykająca bieżące okno specyfikacji i przywracająca poprzedni ekran.

Metoda **btnKoszykClick**:

Funkcja wywoływana po kliknięciu przycisku „Przejdź do koszyka”. Otwiera okno koszyka z wszystkimi dodanymi produktami.

1. **Koszyk_produktów k = new(_koszyk)**: Tworzy nową instancję okna Koszyk_produktów i przekazuje do niego obiekt koszyka, zawierający dodane produkty.
2. **k.Show()**: Wyświetla nowe okno koszyka.
3. **this.Close()**: Zamknięcie bieżącego okna (specyfikacji produktu).

Dodanie gifu i dźwięku – Zostały one dodane do okna Specyfikacja, gdzie pojawia się przycisk Dodaj do koszyka. Do ich poprawnej funkcjonalności wykorzystano pakiety NuGet – NAudio i WpfAnimatedGif. Trwają 3 sekundy, a obrazek pokazuje się zamiast przycisków, które są na ten moment schowane.


Koszyk_produkciów.xaml

The screenshot shows a window titled "Koszyk" with a light blue background. At the top, there is a large red button with the text "Koszyk" in white. Below this, on the left, is a white rectangular box containing a list of products: "Laptop Apple MacBook Air M1", "iPad Air", and "Realme GT 2 Pro". To the right of this list are three red buttons stacked vertically: "Zwiększ liczbę", "Usuń z koszyka", and "Kupuj dalej". Below the product list, there is a section titled "Podsumowanie zakupów:". To the left of this section is a red button with the text "Zakończ zakupy". To the right, there is a table summarizing the cart's total.

Podsumowanie zakupów:	Cena przed rabatem:	12 599,97 zł
	Udzielony rabat:	1 260,00 zł
	Cena ostateczna:	11 339,97 zł

Informacja o koszyku danego klienta, jest przenoszona przez wszystkie okna aż do tego, aby w nim poprawnie wyświetlić wybrane przez klienta produkty. Przy wszelkich zmianach w stanie koszyka na bieżąco pomagają nam metody `OdswiezhListe()`, która jako nowe źródło danych podaje obecne produkty i na nowym koszyku oblicza ceny przed i po rabacie oraz wartość samego rabatu i użyciem metody `ZastosujRabat()`. Wszystkie te metody są zaczerpnięte z klasy `Koszyk` w logice aplikacji. Przycisk `Kupuj dalej` zamyka obecne okno i przenosi nas do miejsca wyboru kategorii dla klienta. Przycisk `Usuń` usuwa z listy w koszyku te, których nazwy są takie, jak te zaznaczone w `ListBoxie`. Gdy nic zostało zaznaczone, wyświetlany jest `MessageBox`. Zwiększenie liczby produktów polega na sklonowaniu produktu, który został zaznaczony na liście. Wykorzystana tu została nadpisana metoda `Clone()` z interfejsu `ICloneable`. Nowy produkt dodawany jest do listy w koszyku. Przycisk `Zakończ zakupy` wyłącza wszystkie okna i kończy działanie aplikacji.

Wersja dla admina:

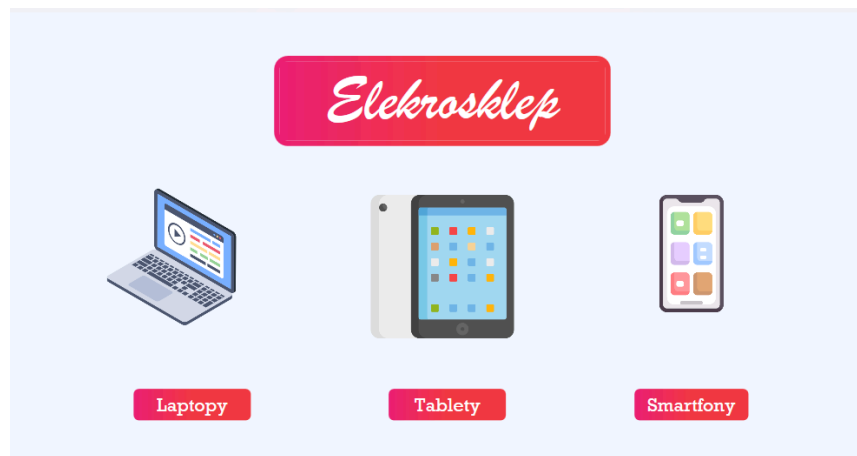


Wpisz login administratora:

Wpisz hasło:

Zaloguj się

Powrót



Admin_ListaProduktów.xaml

Produkty z kategorii: Tablety

Dodaj do magazynu

Usuń z magazynu

Aktualizuj dane

- Microsoft Surface Pro 8
- Lenovo Tab P11 Pro
- Amazon Fire HD 11
- Huawei MatePad Pro
- Xiaomi Pad 5
- iPad Air
- Samsung Galaxy Tab A7

Sortuj po cenie

Sortuj po nazwie

Powrót

Zakończ działanie

Jest to okno analogiczne do okna **ListaProduktów**, ale pojawia się tylko po zalogowaniu z konta administratora i wybraniu kategorii produktu, która determinuje zarówno tytuł na górze okna, ale także zawartość ListBoxa - z pliku XML pobierana jest lista dostępnych w magazynie produktów danej kategorii. Administrator również ma możliwość posortować produkty na liście według nazwy

(alfabetycznie) lub względem ceny (od najtańszych do najdroższych). W konstruktorze okna musi zostać podana kategoria produktu tak, aby następnie wyświetlały się odpowiednie elementy na liście i tytuł okna nad listą. To, w jakiej kategorii produktów jesteśmy jest przechowywane na przestrzeni całego okna w zmiennej **kategoria1**, żeby potem móc się do niej odwołać. Źródłem danych do **LitsBoxa** jest dla nas lista nazw produktów, którą pobieramy z pliku "magazyn.xml" za pomocą metody **PobierzProdukty(string kategoria)**. Jest ona taka sama jak w oknie **ListaProduktów**. Tak samo sytuacja wygląda z funkcją **lstBoxProdukty_SelectionChanged**. Odczytywanie danych z pliku "magazyn.xml" jest możliwe dzięki metodzie statycznej z klasy **Magazyn**. Następnie w kodzie pojawia się przycisk **Powrotu** i jego metoda **btnPowrotClick**, która sprawia, że wracamy z powrotem do okna **Kategoria_produktu** w wersji dla admina, a obecne okno się zamyka. Musimy nawet dla administratora przechowywać w pamięci koszyk, ponieważ konstruktory niektórych okien wspólnych dla obu użytkowników wymagają tego. Koszyk admina oczywiście nie jest nigdy wykorzystywany. Działanie przycisku sortowania po nazwie jest możliwe dzięki metodzie **SortujPoNazwie**, wykorzystującej wbudowaną metodę **Sort** dla listy. Do sortowania pobieramy sobie produkty za pomocą metody **PobierzProdukty** i po sortowaniu jako źródło danych do **ListBoxa** bierzemy posortowaną listę (źródło danych trzeba uprzednio wyczyścić). Przycisk do sortowania po cenie działa analogicznie, natomiast metoda **SortujPoCenie** jest skonstruowana w ten sposób, że najpierw tworzymy sobie listę obiektów klasy **Produkt**, które znajdujemy w odczytanym magazynie z pliku .xml na podstawie ich nazw, ponieważ na liście wcześniejszej mieliśmy zapisane tylko nazwy jako stringi. Nową listę sortujemy z użyciem nadpisanej metody **CompareTo** z interfejsu **IComparable**, który dla **Produktów** ma brać pod uwagę ich ceny. Z posortowanej listy produktów wyciągamy na koniec tylko nazwy, bo te mają być

znowu wyświetlane w ListBoxie. Przycisk usuwania produktów z magazynu działa w ten sposób, że na początku pobiera z pliku .xml Produkty, następnie jeśli żadne produkty na liście nie zostały zaznaczone, to wyświetla odpowiednią informację w MessageBoxie, a gdy coś jest zaznaczone to dodaje to do nowej listy (jako string). Z obiektu klasy Magazyn, na którym pracujemy, usuwane są wszystkie produkty, których nazwa jest taka sama jak tych zaznaczonych elementów. Aktualizacja danych na wyświetlanej liście przebiega analogicznie jak poprzednio. Dodawanie do magazynu polega na włączeniu okna **DaneProduktu** dla tej samej kategorii i z wersją "dodawanie". Po zakończeniu działań w nowym oknie tutaj magazyn jest odczytywany z xml jeszcze raz i produkty na liście odświeżane jak poprzednio. Aktualizowanie danych produktu przebiega tak, że na początku następuje sprawdzenie, czy cokolwiek w ListBoxie zostało zaznaczone. Jeśli nie, wyświetla się odpowiednia informacja. Nazwa zaznaczonego produktu jest wyszukiwana w odczytanym magazynie i z niego pobierany jest cały Produkt. Dla niego otwierane jest okno **DaneProduktu** z opcją "aktualizowanie". Po jego zamknięciu magazyn jest znów zapisywany do xml i jak poprzednio musi jeszcze zostać zaktualizowana lista wyświetlanych produktów. Gdy administrator kończy pracę, powinien do tego wykorzystać przycisk "Koniec działania", który zamyka wszystkie okna i wyłącza aplikację.

DaneProduktu.xaml

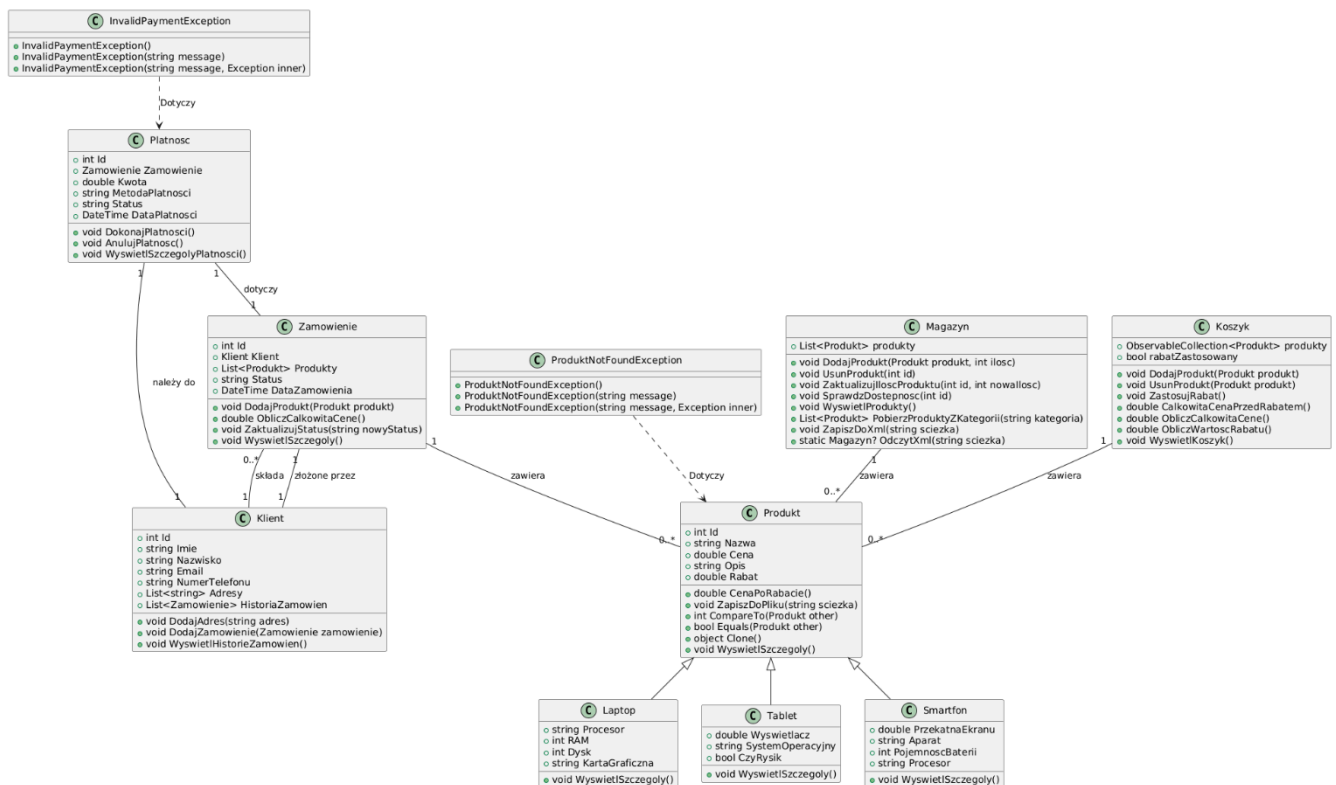
KATEGORIA	Tablety
NAZWA	Amazon Fire HD 11
CENA	699,99
OPIS	Tablet 10 cali z systemem Android
Wyświetlacz	10
System operacyjny	Fire OS
Czy rysik	nie
<div>Zatwierdź</div> <div>Anuluj</div>	

KATEGORIA	Tablety
NAZWA	
CENA	
OPIS	
Wyświetlacz	
System operacyjny	
Czy rysik	
<div>Zatwierdź</div> <div>Anuluj</div>	

Wygląd tego okna zależy od funkcji, jaką ma ono pełnić. Już w konstruktorze musimy podać, z jakiej kategorii produktów przychodzimy i jakich funkcji od okna oczekujemy oraz na jakim produkcie pracujemy. Na przestrzeni całego okna mamy dostępne obiekty klas Laptop, Tablet i Smartfon. Do wybranej już w konstruktorze przypisujemy ten produkt, który do nas przychodzi, chyba że jest on null, co znaczy, że dopiero będziemy dodawać nowy produkt. Następnie w zależności od kategorii dodajemy odpowiednie nazwy etykiet przy polach tekstowych. Dla kategorii "aktualizowanie" dodatkowo dla odpowiednich kategorii wpisujemy szczegółową specyfikację przestanego produktu. Przy dodawaniu tego oczywiście nie ma. Tablet ma o 1 mniej pole, dlatego w tej sytuacji ustawiamy je na niedostępne. Przycisk anuluj zamyka całe okno i znów widzimy tylko poprzednie, które nie zostało wcześniej zamknięte. Przycisk zatwierdź natomiast przypisuje do danych naszego produktu to, co w ostatniej chwili leżało w polach tekstowych. Jeśli produkt był tworzony od zera, wszystkie pola są nowe, dla istniejącego wcześniej produkty także wszystko się nadpisuje, ale część informacji mogła pozostać niezmienną. Konieczna jest w tym miejscu także konwersja na odpowiednie typy, ponieważ w polach tekstowych mamy stringi, a nasze pola często wymagały innych typów. Jeśli

produkt o danej nazwie nie był jeszcze na liście produkty, zostaje do niej dodany w magazynie. Jeśli nie, to jego dane zapisują się także, lecz dopiero czy ostatecznym zapisie do xml magazynu. Okno zostaje na koniec zamknięte.

Diagram klas (będzie dołączony jako Diagram.png do zadania przez słabą jakość w dokumencie Word)



Testy jednostkowe

Do rozwiązania dołączony jest także projekt z testami jednostkowymi. Utworzono 3 testy:

1. **TestNazwaLaptopa:**

Test sprawdza, czy nazwa nowo utworzonego obiektu klasy **Laptop** jest poprawnie ustawiona na wartość podaną podczas inicjalizacji. Porównuje oczekiwaną nazwę z faktyczną nazwą zapisaną w obiekcie.

2. **TestCompareTo:**

Test weryfikuje działanie metody **CompareTo** w klasie **Tablet**. Metoda ta powinna porównywać dwa obiekty **Tablet** na podstawie ceny. Test sprawdza, czy metoda zwraca **-1**, gdy pierwszy obiekt (**t1**) ma mniejszą cenę niż drugi obiekt (**t2**).

3. **wyjatek:**

Test sprawdza, czy metoda **SprawdzDostepnosc** klasy **Magazyn** poprawnie rzuca wyjątek **ProduktNotFoundException** w sytuacji, gdy nie znaleziono produktu o podanym identyfikatorze. Test jest poprawny, jeśli metoda wyrzuci oczekiwany wyjątek.