

Dokumentacja

Projekt:

“Hurtownia/magazyn sprzętu rowerowego”

Programowanie Obiektowe

Politechnika Śląska

Wydział Elektryczny

Przygotowali:

Kacper Czerwiński

Igor Hałas

Błażej Janus

Spis Treści

1. Ogólne założenia projektu
2. Prezentacja architektury oprogramowania
3. Omówienie zastosowanych technologii
4. Struktura obiektowa
 - 4.1 Interfejsy
 - 4.1.1 IDataSource
 - 4.1.2 IItem
 - 4.2 Klasy
 - 4.2.1 Koszyk
 - 4.2.2 Klasy implementujące IDataSource
 - 4.2.3 Klasy implementujące IItem
 - 4.2.4 Dodatkowe klasy abstrakcyjne i ich klasy potomne
 - 4.2.5 Klasa GUI (Formularz)
 - 4.2.6 Klasa Koszyk_GUI (Formularz)
 - 4.2.7 Klasa FormDodawanie (Formularz)
 - 4.2.8 Klasa Zamówienie
 - 4.2.9 Klasa ZawartoscLst
5. Struktura danych
 - 5.1 Podstawowe struktury
 - 5.2 Typy wyliczeniowe
6. Klasy narzędziowe
 - 6.1 Klasa tostr
 - 6.2 Klasa separator
 - 6.2 Klasa Test
7. Struktura bazy danych
8. Harmonogram
9. Wnioski

1. Ogólne założenia projektu

Projekt ma obejmować system zarządzania magazynem części rowerowych.

Konieczna jest realizacja następujących zadań:

- możliwość kupna towaru przez klienta (GUI)
- obsługa koszyka dla klienta (GUI)
- możliwość sprzedaży całego roweru złożonego z części
- możliwość złożenia roweru z części dostępnych w magazynie
- obsługa dostaw towaru do magazynu

Projekt ma wykorzystywać bazę danych do przechowywania informacji o stanie magazynu.

Dostęp do bazy danych ma być realizowany za pomocą odpowiedniej klasy.

Konieczna jest kontrola nad stanem magazynu, np. klient nie powinien móc zamówić więcej towaru niż jest w magazynie.

Dla klienta sklepu przygotowany zostanie interfejs graficzny, w którego skład wchodzi:

Strona z kategoriami:

- kategorie główne (Przedmioty, Rowery, Akcesoria, Części)
- kategorie podrzędne (np. Ramy, Koła, Odblaski)
- sortowanie
- dostępność/ilość produktu
- Strona koszyka:
 - możliwość dodawania usuwania towaru
 - możliwość zmiany ilości towaru
 - możliwość kupienia towaru
 - możliwość wyczyszczenia koszyka
 - koszyk musi wyświetlać sumaryczną cenę, aktualizowaną samoczynnie po zmianie zawartości koszyka
- Strona składania roweru:
 - możliwość dobrania ramy
 - możliwość dobrania podzespołów kompatybilnych z ramą
 - konieczność dodania wszystkich niezbędnych elementów
 - możliwość, ale nie konieczność dodania opcjonalnych elementów

2. Prezentacja architektury oprogramowania

Aplikacja do przechowywania danych ma wykorzystywać bazę danych. Za komunikację z bazą danych jest odpowiedzialna klasa `DataSourceSQL`, implementująca interfejs `IDataSource`.

Prezentacja danych klientowi ma być realizowana przy użyciu Windows Forms.

Podstawą dla typów danych reprezentujących poszczególne części jest interfejs `IItem`. Wszystkie ww typy danych implementują ten interfejs.

Ze względu na taki sposób przechowywania informacji o towarze, naturalnym staje się zastosowanie list generycznych `list<IItem>`.

Rozwiązanie takie umożliwia przechowywanie i przekazywanie między poszczególnymi funkcjami informacji o elementach w spójny, prosty i wygodny sposób.

W programie jest używane wiele typów wyliczeniowych, reprezentujących wartości poszczególnych parametrów właściwych podzespołów.

3. Omówienie zastosowanych technologii

W naszym programie wykorzystaliśmy technologie takie jak:

- SQLite - system relacyjnych baz danych SQL.
- Windows Forms - GUI
- Dapper - narzędzie do uproszczonego wykonywania operacji na Bazach Danych (Object-Relational Mapping).

SQLite jest stosunkowo prostą, nie wymagającą wielu zasobów implementacją silnika relacyjnej bazy danych SQL.

Biblioteka ta została napisana w języku C.

Jest dostępna na licencji domeny publicznej.

W celu zapewnienia lepszych możliwości współpracy, umożliwienia wygodnej asynchronicznej pracy na współdzielonych zasobach wykorzystaliśmy w naszej pracy system kontroli wersji Git.

Do hostowania projektu wykorzystaliśmy platformę GitHub.

Projekt napisany został w języku C# przy użyciu środowiska Microsoft Visual Studio 2019 Community.

4. Struktura obiektowa

Projekt został zrealizowany zgodnie z zasadami obiektowego paradygmatu programowania. Wykorzystane struktury tworzą spójną, wzajemnie powiązaną i zależną całość. Zastosowanie takiego paradygmatu zapewnia hermetyzację danych, spójność i jednoznaczność wykonywanych działań, oraz ułatwia rozszerzanie programu o nowe funkcje.

Zastosowanie dziedziczenia przy projektowaniu klas reprezentujących poszczególne podzespoły ułatwia dodawanie kolejnych, nowych elementów, ułatwia zapis i przekazywanie danych pomiędzy różnymi częściami kodu.

4.1 Interfejsy

W projekcie można wyszczególnić dwa główne interfejsy, omówione poniżej. Interfejsy te są podstawą struktury dziedziczenia zastosowanej w projekcie.

4.1.1 IItem

Interfejs, z którego dziedziczą wszystkie klasy reprezentujące poszczególne podzespoły. Wymusza na wszystkich implementujących go klasach implementację wspólnych dla wszystkich elementów właściwości.

Właściwości wymuszone przez interfejs:

- int ID - unikalny numer identyfikacyjny
- string Nazwa - nazwa konkretnego elementu
- double Cena - cena
- enum Producent - producent konkretnego elementu
- int Ilość - ilość towaru w magazynie.

Metody wymuszone przez interfejs:

- public void Wypisz()

4.1.1 IDataSource

Interfejs implementowany przez klasę DataSourceSQL.

Zawiera metody:

List<IItem> WczytajDane() - metoda pobierająca wszystkie przedmioty z bazy danych, zwraca je w postaci listy generycznej typu IItem.

void UstawIlosc(IItem obj, int ilosc) - metoda ustawiająca podaną ilość dla podanego argumentu (kupowanie i dodanie przedmiotu).

void DodajElement(IItem obj) - metoda dodająca do bazy danych nowy element podany w argumencie, tworzony jest nowy rekord.

List<Zamowienie> WyszwietlZamowienia() - metoda wyświetlająca wszystkie zamówienia.

void DodajZamowienie(Zamowienie obj) - dodawanie nowego zamówienia do bazy danych.

4.2 Klasy

Klasy reprezentują w projekcie bardziej złożone obiekty, posiadające możliwość przechowywania danych oraz umożliwiające wykorzystywanie tych danych, jak również inne właściwe im metody.

4.2.1 Koszyk

Koszyk jest klasą odpowiedzialną za przechowywanie i manipulowanie towarem dodanym do koszyka przez klienta.

Konstruktory:

```
public Koszyk(GUI AppGUI, bool display = false)
```

```
private Koszyk()
```

Pola klasy koszyk:

List<ZawartoscLst> zawartość - lista przechowująca zawartość koszyka

Metody klasy koszyk:

```
public void Dodaj(IItem)
```

- dodaje element do koszyka

```
public void Usuń(IItem)
```

- usuwa element z koszyka

```
public void Wyczyść()
```

- usuwa całą zawartość koszyka

`private void Aktualizuj()`

- metoda wewnętrzna, używana w celu zaktualizowania sumarycznej ceny produktów w koszyku po każdej zmianie jego zawartości.

`public void value_update_GUI(object sender, int ID, int zmiana)`

- może jej używać tylko GUI koszyka. Stosowana przy zmianie ilości produktów w formularzu koszyka.

`private Item GetItem(int ID)`

- zwraca element zawartości o wskazanym ID, jeśli nie znajdzie takiego zwraca null

`public void Widocznosc(bool widocznosc)`

- zmienia widoczność GUI koszyka

`public List<ZawartoscLst> PobierzZawartoscLst()`

- zwraca zawartość koszyka w formie listy obiektów klasy `ZawartoscLst`

`public List<Item> PobierzZawartoscItem()`

- zwraca zawartość koszyka w formie listy obiektów `Item`

4.2.2 Klasa `DataSourceSQL`

Klasa służąca do zarządzania bazą danych. Umożliwia ona obustronną komunikację (zapis, odczyt) z bazą danych. W tej klasie został zaimplementowany wzorzec projektowy Singleton, dzięki któremu może istnieć tylko jedna instancja tej klasy w programie.

W klasie `DataSourceSQL` jest statyczne pole prywatne przechowujące instancje tej klasy (`instance`). Jego wartość można odczytać poprzez publiczną właściwość `Instance` (tylko do odczytu), która zwraca wartość pola `instance`. Gdy pole nie ma przypisanej wartości (`null`) to jest do niego przypisywany nowy obiekt klasy `DataSourceSQL`.

Metody klasy:

Publiczne (opisane w `IDataSource`):

```
List<Item> WczytajDane(),  
void UstawIlosc(Item obj, int ilosc),  
void DodajElement(Item obj),  
List<Zamowienie> WyszukajZamowienia(),
```

```
void DodajZamowienie(Zamowienie obj),
```

Prywatne:

string LoadConnectionString(string name = "Default") - metoda wczytująca łańcuch połączeniowy (Connection String) zawierający informacje o danym połączeniu (Default) z pliku konfiguracji projektu, łańcuch ten jest zwracany.

Do otwierania i zamykania połączenia z bazą wykorzystywana jest klauzula using {}, która automatycznie otwiera połączenie i zamyka je po wykonaniu zadanej operacji. Połączenie jest automatycznie tworzone i usuwane.

4.2.3 Klasy implementujące Item

Każda z klas reprezentujących jakikolwiek podzespół oraz akcesorium musi implementować ten interfejs.

Implementują go wszystkie klasy dziedziczące po klasach Akcesoria i Czesci.

4.2.4 Dodatkowe klasy abstrakcyjne i ich klasy potomne

Klasa przechowująca informacje o akcesoriach rowerowych niebędących podzespołami roweru to abstrakcyjna klasa "Akcesoria".

Właściwości klasy "Akcesoria":

```
public int ID
public string Nazwa
public double Cena
public Producents Producent (enum)
public int Ilosc
```

Metody klasy "Akcesoria":

```
public virtual List<string> Wypisz()
```

Klasy dziedziczące po klasie Akcesoria

W każdej klasie dodano odpowiednie właściwości reprezentujące parametry konkretnego podzespołu. Każda z poniższych klas przeciąża metodę Wypisz().

Klasa "Kask"

Właściwości klasy “Kask”:

```
public Rozmiary Rozmiar  
public Plcie Plec (enum)  
public Materiał materiał (enum)  
public double Masa  
public Kolory Kolor (enum)
```

Klasa “Narzedzia”

Właściwości klasy “Narzedzia”:

```
Materialy Material (enum)
```

Pola klasy “Narzedzia”:

```
private List<RodzajNarzedzi> narzedzia - przechowuje listę  
narzędzi.
```

Metody klasy “Narzedzia” (obie metody dodają narzędzia do listy):

```
public void DodajNarzedzia(List<RodzajNarzedzi> narzędzia)  
public void DodajNarzedzia(RodzajNarzedzi narzedzie)
```

Klasa posiada konstruktory umożliwiające dodanie pojedynczego/listy narzędzi przy tworzeniu instancji klasy. Konstruktory wywołują ww metody.

Klasa “Oswietlenie”

Właściwości klasy “Oswietlenie”:

```
public Materialy Material
```

Pola klasy “Oswietlenie”:

```
private List<Kolory> kolor_swiecenia;  
private List<TrybSwiecenia> tryb_swiecenia;
```

Metody klasy “Oswietlenie”:

```
public void UstawKolorSwiecenia(Kolory Kolor)  
public void UstawKolorSwiecenia(List<Kolory> kolory)  
public void UstawTrybSwiecenia(TrybSwiecenia tryb)  
public void UstawTrybSwiecenia(List<TrybSwiecenia> tryby)
```

Klasa posiada konstruktory umożliwiające dodanie pojedynczego/listy kolorów i pojedynczego/listy trybów świecenia przy tworzeniu instancji klasy. Konstruktory wywołują ww metody.

Klasa “Bidon”

Właściwości klasy “Bidon”:

```
public double Pojemnosc
public Materiały Material
public double Masa
public Kolory Kolor
```

Klasa “Licznik”

Właściwości klasy “Licznik”:

```
public IPs IP
public double Czas_pracy
public Kolory Kolor
```

Pola klasy “Licznik”:

```
public List<FunkcjeLicznika> funkcje
```

Metody klasy “Licznik”:

```
public void DodajFunkcje(FunkcjeLicznika funkcja)
public void DodajFunkcje(List<FunkcjeLicznika> funkcje)
```

Klasa posiada konstruktory umożliwiające dodanie pojedynczego/listy funkcji przy tworzeniu instancji klasy. Konstruktory wywołują ww metody.

Klasa “Lusterko”

Właściwości klasy “Lusterko”:

```
public Polozenie Rodzaj_lusterka
public Kolory Kolor
public Materiały Material
```

Klasa “Pokrowiec”

Właściwości klasy “Pokrowiec”:

```
public TypPokrowca Typ
```

```
public Kolory Kolor  
public Materiały Material
```

Klasa “Rekawiczka”

Właściwości klasy “Rekawiczka”:

```
public Kolory Kolor  
public Materiały Material  
public Plcie Plec
```

Klasa “Stopka”

Właściwości klasy “Stopka”:

```
public Kolory Kolor  
public Materiały Material  
public RodzajStopki Rodzaj_stopki
```

Klasa przechowująca informacje o częściach rowerowych będącymi podzespołami roweru, implementuje interfejs IItem:

Wirtualna klasa ”Czesci”

Właściwości klasy “Czesci”:

```
public int ID  
public string Nazwa  
public double Cena  
public Producent Producent (enum)  
public int Ilosc
```

Metody klasy “Czesci”:

```
public virtual List<string> Wypisz()
```

Klasy dziedziczące po klasie Czesci

posiadają one dodatkowe właściwości, które opisują dokładnie parametry danej części oraz przeciążoną metodę Wypisz(), która zamienia wartości właściwości na listę string.

Klasa “Rama”

Właściwości klasy “Rama”:

public Kolory Kolor
public Typy Typ
public Materiały Material
public Rozmiary Rozmiar
public double Waga
public double SrednicaSztocy
public double RozmiarKola

Publiczna Klasa “Siodelko”

Właściwości klasy “Siodelko”:

public Kolory Kolor
public Typy Typ
public Plcie Plec
public Materiały Material

Publiczna Klasa “Przerzutka”

Właściwości klasy “Przerzutka”

public Kolory Kolor
public int IloscPrzelozen
public Polozenia Polozenie

Publiczna Klasa “Kolo”

Właściwości klasy “Kolo”:

public Kolory Kolor
public Materiały Material
public double Rozmiar
public Typy Typ

Publiczna Klasa “Kierownica”

Właściwości klasy “Kierownica”:

public Kolory Kolor
public Typy Typ
public Materiały Material
public double Szerokosc

Publiczna Klasa “Hamulec”

Właściwości klasy “Hamulec”:

```
public TypyHamulca TypHamulca  
public Polozenia Polozenie  
public Typy Typ  
public Kolory Kolor
```

Publiczna Klasa “AmortyzatorTylny”

Właściwości klasy “AmortyzatorTylny”

```
public Kolory Kolor  
public RodzajeAmortyzacji RodzajAmortyzacji  
public double Skok  
public double RozmiarKola
```

Publiczna Klasa “AmortyzatorPrzedni”

Właściwości klasy “AmortyzatorPrzedni”:

```
public Kolory Kolor  
public Materialy Material  
public RodzajeAmortyzacji RodzajAmortyzacji  
public double Skok  
public double RozmiarKola  
public double SrednicaSztocy
```

4.2.5 Klasa GUI

Klasa GUI jest klasą dziedziczącą po klasie systemowej Form, klasa ta jest odpowiedzialna za tworzenie i wyświetlanie interfejsu graficznego dla użytkownika. W skład tej klasy wchodzi, poza domyślnymi:

Obiekty:

- button_kup - przycisk klasy Button do kupowania wybranych rzeczy,
- button_doKoszyka - przycisk klasy Button do dodawania wybranych rzeczy do koszyka.
- treeView_kategorie - obiekt klasy TreeView, zawiera rozwijaną listę z wszystkimi kategoriami przedmiotów dostępnych w hurtowni.

- listView_container - obiekt klasy ListView, służy do wyświetlania danych znajdujących się w bazie, jego kolumny są dynamicznie zmieniane w zależności od wybranego elementu. Służy on także do wybierania danych poprzez zaznaczenie.
- pictureBox_koszyk - obiekt klasy PictureBox, służy do otwierania i zamykania koszyka.
- textBox_ilosc - obiekt klasy TextBox, wpisujemy do niego ilość towaru jaką chcemy zakupić.
- label_ilosc - etykieta dla pola tekstowego z ilością.
- button_szukaj - przycisk klasy Button do wyszukiwania danych na podstawie podanego tekstu.
- button_dodaj - przycisk klasy Button, służy do otwierania formularz do dodawania nowych przedmiotów do bazy danych.
- button_orders - przycisk klasy Button, służy do wyświetlania okna z aktualnymi zamówieniami.
- textBox_orders - pole tekstowe do wyświetlania zamówień, znajdujących się w bazie danych.
- textBox_szukaj - obiekt klasy TextBox, wpisujemy do niego szukane wyrażenie.
- button_zwieksz - obiekt klasy Button, do zwiększania ilości towaru.

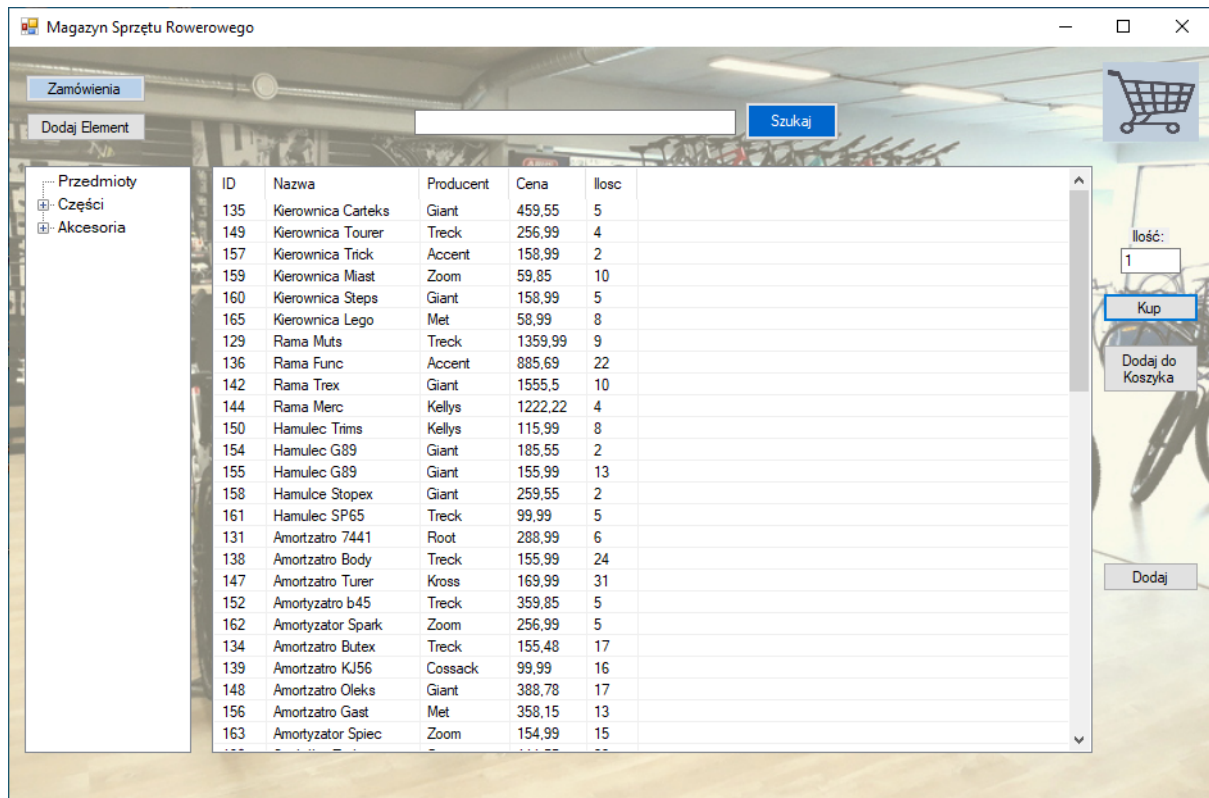
Metody:

- DodajBazowe() - metoda dodająca bazowe kolumny do obiektu klasy ListView, kolumny te są one zgodne z właściwościami interfejsu IItem.

- `treeView_kategorie_AfterSelect()` - metoda, która jest aktywowana po wybraniu elementu z obiektu `treeView_kategorie`, w zależności od wybranej kategorii dodaje ona odpowiednie kolumny do obiektu klasy `ListView` oraz wpisuje do niej dane z wcześniej pobrane z bazy dane.
- `pictureBox_koszyk_Click()` - metoda wywoływane po kliknięciu na zdjęcie koszyka (obektu `pictureBox_koszyk`), jest ona odpowiedzialna za tworzenie (jeśli nie jest stworzony) i wyświetlanie koszyka.
- `button_kup_Click()` - metoda wywoływana po kliknięciu przycisku `button_kup`. Jej zadaniem jest kupowanie wybranego produktu, wykorzystuje do tego metodę `UstawIlosc(Item obj, int ilosc)`. Dodatkowo sprawdza ona czy można kupić dany produkt. Można kupić więcej niż jeden przedmiot na raz, po wykonaniu operacji zakupy jest wyświetlany odpowiedni komunikat.
- `button_kup_doKoszyka()` - metoda wywoływana po kliknięciu na przycisk `button_doKoszyka`, służy ona do dodawania wybranych elementów z `listView_container` do koszyka, wykorzystuje do tego metodę `Dodaj(Item obj)` z klasy `Koszyk`. Po wykonaniu operacji jest wyświetlany odpowiedni komunikat (powodzenie lub błąd).
- `button_szukaj_Click()` - metoda wywoływana po kliknięciu na przycisk `button_szukaj`.
- `button_dodaj_Click()` - metoda wywoływane po kliknięciu na przycisk `button_dodaj`, otwiera ona nowy formularz `FormDodawanie`.
- `button_orders_Click()` - metoda wywoływane po kliknięciu na przycisk `button_szukaj`, ukrywa ona wszystkie elementy poza przyciskiem zamówienia i odkrywa pole tekstowe z zamówieniami (wpisywane są do niego wartości odczytane z bazy danych). Ponowne kliknięcie odwraca tą operację.

- `button_zwieksz_Click` - metoda wywoływana po kliknięciu przycisku `button_zwieksz`, zwiększa ilość wybranego elementu o wybraną liczbę.

Wygląd GUI:



4.2.6 Klasa Koszyk_GUI (Formularz)

Klasa “`koszyk_GUI`” służy do wyświetlania informacji o produktach w koszyku, a także do manipulowania produktami tam się znajdującymi.

Użytkownik może wyczyścić zawartość koszyka, kupić wszystkie produkty w koszyku, czy też np. zmienić ilość konkretnego produktu.

W koszyku użytkownika może zobaczyć jakie produkty zostaną dodane do zamówienia i ile za nie zapłaci. “`Koszyk_GUI`” służy także do pośredniczenia między koszykiem (jego logiką), a resztą GUI. Waliduje on wprowadzone dane i zwraca wyjątki w razie napotkania błędów.

Klasa ta posiada 2 konstruktory:

```
public koszyk_GUI()
```

```
public koszyk_GUI(GUI AppGUI, bool display=false)
```


- przyjmuje ona odniesienie do głównego GUI aplikacji
- flaga display, umożliwia ustawienia czy nowo utworzony obiekt ma od razu zostać wyświetlony, czy też nie.

Metody klasy “Koszyk_GUI”:

public void Dodaj(Item przedmiot)

public void Dodaj(List<Item> przedmioty)

- obie metody korzystają z metody dodawania przedmiotu z klasy koszyk, sprawdzają poprawność danych wejściowych oraz możliwość wykonania operacji. W razie problemów generują wyjątek.

public void Usun(Item przedmiot)

public void Usun(List<Item> przedmioty)

- obie metody korzystają z metody usuwania przedmiotu z klasy koszyk, sprawdzają poprawność danych wejściowych oraz możliwość wykonania operacji. W razie problemów generują wyjątek.

public void Pokaz() - włącza widoczność koszyka

public void Wyswietl() - wyświetla zawartość koszyka

public void DokonanoZakupu(double cena) - wyświetla komunikat o dokonaniu zakupu

Funkcje odpowiedzialne za obsługę elementów GUI:

private void koszyk_button_wyczysc_Click(object sender, EventArgs e) - reaguje na kliknięcie przycisku wyczyść.

private void koszyk_button_kup_Click(object sender, EventArgs e) - reaguje na kliknięcie przycisku kup

private void koszyk_GUI_FormClosing(object sender, FormClosingEventArgs e) - uniemożliwia zamknięcie koszyka poprzez

kliknięcie przycisku X w prawym górnym rogu okna koszyka.
Zamknięcie spowodowałoby zniszczenie aktualnego koszyka, zamiast tego koszyk jest ukrywany.

Metody wewnętrzne, używane przez inne metody:

`private static List<ZawartoscLst> Convert2Struct(List<IItem> data)`

- kopiuje przekazaną `List<IItem>`
- sortuje skopiowaną listę po ID
- liczy wystąpienia każdej z części
- po każdej części o unikatowym ID dodaje nowy obiekt klasy `ZawartoscLst` do Listy, która zostanie zwrócona. Obiekt ten zawiera informację o każdym unikatowym obiekcie klasy `IItem` oraz liczbę jego wystąpień w przekazanej liście.
- Zwrócona lista zawiera więc listę występujących w koszyku przedmiotów wraz z ich ilością.

`public static void ConversionTest(List<IItem> data)`

- metoda służąca do testowania poprawności konwersji z `List<IItem>` na `List<ZawartoscLst>`.

Klasa wewnętrzna `ZawartoscLst`:

Konstruktor:

`public ZawartoscLst()` - inicjalizuje nowy obiekt, ustawia ilość na 0, a element na null

Właściwości:

`public int ilosc { get; set; }` //Unikatowy element
`public IItem element { get; set; }` //Ilość każdego elementu

Metody:

`public override string ToString()`
- umożliwia wygodną reprezentację obiektu tej klasy w formie czytelnie sformatowanego stringa.

Pola klasy:

```
private GUI AppGUI; - referencja do głównego GUI  
private Koszyk koszyk = null; - referencja do koszyka
```

Obiekty wchodzące w skład Formularza Koszyk_GUI:

Formularz Koszyk_GUI dziedziczy po Form.

Do Formularza dodano następujące obiekty:

Button:

koszyk_button_wyczysc - przycisk kup

koszyk_button_kup - przycisk wyczyść

DataGridView koszyk_zawartosc_lista, składająca się z kolumn:

DataGridViewTextBoxColumn koszyk_ds_lp;

DataGridViewTextBoxColumn koszyk_ds_name;

DataGridViewTextBoxColumn koszyk_ds_ilosc;

DataGridViewTextBoxColumn koszyk_ds_cena;

.DataGridViewTextBoxColumn koszyk_ds_wartosc;

Label koszyk_label_title - tytuł

Wygląd koszyka:

 Koszyk

×

Zawartość koszyka:

Lp.:	Nazwa:	Ilość:	Cena:	Wartość:
1.	Amortyzator b45	1	359,85	359,85
	Suma:	1	359,85	359,85

KUP

Wyczyść

4.2.7 Klasa FormDodawanie (Formularz)

Formularz do dodawania nowych elementów do bazy danych, zawiera listę z kategoriami przedmiotów, które można dodać oraz pole tekstowe, do którego wpisujemy parametry danego przedmiotu. Parametry oddzielone są przecinkiem, muszą być podawane w kolejności wyświetlonej nad polem tekstowym.

Obiekty

listBox_typ - obiekt klasy ListBox zawierający kategorie dla których możemy dodać nowy element.

label_wzor - obiekt klasy Label, zawiera wzór dla wprowadzanych danych w polu tekstowym.

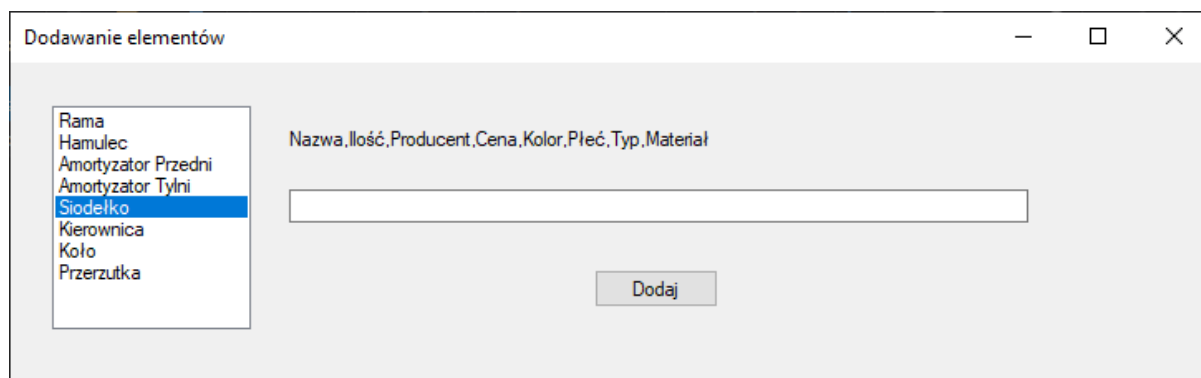
textBox_dane - obiekt klasy TextBox, służy do wprowadzania parametrów danego przedmiotu.

button_dodaj - obiekt klasy Button, po jego kliknięciu następuje sprawdzenie wprowadzonych danych, jeśli są poprawne to obiekt jest wprowadzany do odpowiedniej tabeli w bazie danych.

Metody:

listBox_typ_SelectedIndexChanged - metoda ustawia Tekst obiekt label_wzor w zależności od wybranego pola.

button_dodaj_Click - metoda pobiera dane z pola tekstowego i je przetwarza oraz sprawdza, następnie na ich podstawie tworzy wybrany obiekt i wpisuje go do bazy danych.



4.2.8 Zamówienie

Klasa przechowująca informacje o zamówieniu. Posiada listę typu `IItem`, w której przechowuje zakupione towary, listę `int` na ilość zakupionych przedmiotów, właściwość `ID` na numer zamówienia oraz właściwość `Data` przechowującą informację o dacie zakupu towarów.

Dodatkowo klasa `Zamówienie` posiada metodę `Wypisz()` do wypisania danych zamówienia w postaci łańcucha znaków.

4.2.9 Klasa `ZawartoscLst`

Prosta klasa zaprojektowana do przechowywania pojedynczego obiektu `IItem` oraz zmiennej `int` oznaczającej jego ilość.

Klasa została dodana w celu ułatwienia wyświetlania zawartości koszyka w jego GUI.

Konstruktory:

```
public ZawartoscLst()
```

- konstruktor ustawia ilość na 0, a element na `null`.

Właściwości:

```
public int ilosc { get; set; }
```

```
public IItem element { get; set; }
```

Metody:

```
public List<IItem> ToIItem()
```

- przetwarza obiekt struktury na listę `IItem`

```
public override string ToString()
```

5. Struktura danych

Omówienie typów danych stosowanych w projekcie.

5.1 Podstawowe struktury

Jak wyłożono wcześniej podstawą przechowywania i wymiany danych o podzespołach jest interfejs `IItem`.

Ze względu na łatwość przechowywania, przetwarzania i przesyłania większej

liczby podzespółów wykorzystuje się tablice generyczne List<Item>.

5.2 Typy wyliczeniowe

Typy wyliczeniowe stosowane są w projekcie w celu zapewnienia jednoznaczności danych i wygodniejszego przetwarzania zmiennych reprezentujących poszczególne parametry różnych elementów.

Zastosowane typy wyliczeniowe:

public enum FunkcjeLicznika:

zegar, dystans, prędkość, kalorie

public enum IP:

IP00, IP01, IP02, IP03, IP04, IP05, IP06, IP07, IP08, IP09,
IP10, IP11, IP12, IP13, IP14, IP15, IP16, IP17, IP18, IP19,
IP20, IP21, IP22, IP23, IP24, IP25, IP26, IP27, IP28, IP29,
IP30, IP31, IP32, IP33, IP34, IP35, IP36, IP37, IP38, IP39,
IP40, IP41, IP42, IP43, IP44, IP45, IP46, IP47, IP48, IP49,
IP50, IP51, IP52, IP53, IP54, IP55, IP56, IP57, IP58, IP59,
IP60, IP61, IP62, IP63, IP64, IP65, IP66, IP67, IP68, IP69

public enum Kolory:

Zółty, Zielony, Niebieski, Czerwony, Biały, Czarny,
Fioletowy, Pomarańczowy, Rozowy, Granatowy,
Szary, Grafitowy, Bezowy,

public enum Materiały:

Aluminium, Stal, Plastik, Włókno_Szklane, Tytan, Carbon, Guma,
Bawełna, Poliester, Skóra

public enum Płeć:

Męski, Damski, Dziecięcy, Uniwersalny

public enum Producent:

Accent, Andar_Bike, Baltica_Bicycles, Cossack, Zoom, Root, Kellys,
Giant, Lazer, Meteor, Met, Poc, Eneo, Lumix, Petzl, Timex, Dafi, Dural,
Stanley, Brugi, Iguana, Kross, Trek

public enum Polozenie:

prawe, prawa, lewe, lewa, uniwersalne, uniwersalna, centralna, przod, tyl, przedni, tylni, zestaw

public enum RodzajNarzedzi:

pompka, klucze_płaskie, klucze_oczkowe, srubokrety, lyzka_do_opon

public enum Rozmiary:

S, M, L, M, XL, XXL, XS, XXS

public enum TrybSwiecenia:

ciągły, migający, migający_dwukolorowy, sekwencyjny

public enum Typy:

Miejski, Gorski, Szosowy, Profesjonalny, BMX, MTB, Sportowy, Cross

public enum TypPokrowca:

na_rower, na_siodelko

public enum TypyHamulca:

Szczekowy, Tarczowy_Hydrauliczny, Tarczowy_Mechaniczny

public enum RodzajeAmortyzacji:

Brak, Sprezynowy, Gazowy, Olejowy

6. Klasy narzędziowe

W trakcie rozwoju projektu dodane zostały klasy narzędziowe różnego typu przeznaczone do pełnienia różnych funkcji. Klasy te mają w założeniu uprościć różne czynności, jak również pomóc w unikaniu powtórzeń kodu wykonującego te same operacje.

6.1 Klasa **tostr**:

Ze względu na podjętą decyzję, że wartością zwracaną przez funkcje wypisujące parametry poszczególnych części/akcesoriów zwracają `List<string>` konieczne jest powtarzanie podobnych operacji do każdej z klas

implementujących tę funkcję. Aby uniknąć powtarzania kodu, oraz ułatwić i przyspieszyć jego pisanie stworzono klasę `tostr`.

Klasa `tostr` wprowadza metody, które pozwalają na łatwą konwersję typów wyliczeniowych enum oraz list tych typów na czytelne i estetyczne ciągi znaków.

Klasa zawiera różne wersje (dla każdego niezbędnego typu wyliczeniowego) metody:

```
public static string Enum2str(Enum)
```

Metoda ta zwraca wartość typu wyliczeniowego enum w estetycznej formie, np. usuwa z wartości znaki takie jak “_”.

napisane wersje metody “2str”:

```
public static string TypPokrowca2str(TypPokrowca pokrowiec)
```

```
public static string Materiał2str(Materiał materiał)
```

```
public static string IP2str(IP ip)
```

```
public static string TrybŚwiecenia2str(TrybŚwiecenia tryb_świecenia)
```

Kolejną metodą zawartą w tej klasie są różne wersje metody `List2str` (dla każdego niezbędnego typu wyliczeniowego).

```
public static string List2str(List<Enum>)
```

Metoda ta zwraca pojedynczy ciąg, w którym każdy z elementów listy jest estetycznym ciągiem znaków rozpoczynającym się od tabulatora “\t”, a zakończonym nową linią “\n”, umożliwia to łatwe wyświetlanie na ekranie konsoli listy wartości różnych parametrów.

Napisane wersje metody “List2str”:

```
public static string List2str(List<FunkcjeLicznika> lista)
```

```
public static string List2str(List<RodzajNarzędzi> lista)
```

```
public static string List2str(List<Kolor> lista)
```

```
public static string List2str(List<TrybŚwiecenia> lista)
```

6.2 Klasa separator:

Klasa separator dostarcza metod do łatwej obróbki podstawowego elementu komunikacji jakim jest List<string>, jak i samych elementów, które mają wejść w skład takiej listy.

metody klasy “separator”:

```
public static void endl(List<string> lista)
public static void endl(string element)
```

Obie metody zostały stworzone w celu ułatwienia i przyspieszenia pisania funkcji wypisz w klasach potomnych klasy Akcesoria.

Każdy element przekazywanej listy powinien kończyć się znakiem nowej linii “\n”. Wyżej wymienione metody dodają taki znak do ciągu/wszystkich ciągów na liście, jeśli te już nie kończą się takim znakiem. Pozwoliło to zautomatyzować proces dodawania końca linii. Klasa separator używana jest przy testowaniu pracy programu w konsoli, bez GUI.

6.3 Klasa Test

Klasa Test nie jest częścią samego projektu, dlatego została dodana w podprzestrzeni nazw. Cały projekt dzieli przestrzeń nazw Projekt_PO. Klasa Test znajduje się w przestrzeni Projekt_PO.Test.

Klasa została napisana, aby przyspieszyć testowanie poprawności już napisanych elementów. Pierwotnie została napisana do wyświetlania w konsoli wyniku wywołania metody Wypisz() dla najbardziej złożonych klas akcesoriów.

Zawartość klasy “Test”:

```
public enum Tryb
    licznik, oswietlenie, bmx, koszyk
    - służy do wyboru trybu działania

public static void TestujObiekt(Tryb tryb, bool błąd=false)
    - tworzy i jeśli to możliwe wyświetla wynik funkcji Wypisz()

private static Licznik CreateLicznik()
    - tworzy przykładowy licznik

private static Oswietlenie CreateOswietlenie()
```

- tworzy przykładowe oświetlenie

```
private static void wypisz_liste(List<string> lista)
```

- wypisuje zawartość listy List<string> do konsoli

```
private static void TestKoszyk()
```

- tworzy listę kilku elementów i wywołuje dla nich funkcję zamiany na ZawartoscLst, wynik wpisuje w konsoli.

7. Struktura Bazy Danych

Baza danych (ProjektDB.db) składa się z głównej tabeli Przedmioty, która posiada kolumny zgodne z wartościami interfejsu IItem:

- ID (Klucz Główny - PK)
- Nazwa,
- Cena,
- Ilosc,
- Producent.

Dodatkowo tabela Przedmioty posiada kolumny, który służą do łączenia jej z tabelami posiadającymi bardziej szczegółowe parametry poszczególnych, są to kolumny:

- TypPrzedmiotu,
- IDparam (Klucz Obcy - FK).

Dodatkowe tabele z dokładniejszymi parametrami przedmiotów, posiadają kolumny zgodne z wartościami klas tych przedmiotów, tabele posiadają klucz główny IDparam. (Przykładowa klasa Rama na diagramie).

Tabela Zamówienia, przechowująca informacje o zamówieniach, posiada kolumny:

- ID (PK),
- Towar,
- Ilosc,
- Data.

Wartości w polach Towar i Ilosc są oddzielone przecinkami.

Aktualne tabele:

<div> <div>AmorP</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>RodzajAmortyzacji</div> <div>TEXT</div> </div> <div> <div>Skok</div> <div>TEXT</div> </div> <div> <div>RozmiarKola</div> <div>TEXT</div> </div> <div> <div>SrednicaSztycy</div> <div>TEXT</div> </div> </div> <div> <div>AmorT</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>RodzajAmortyzacji</div> <div>TEXT</div> </div> <div> <div>Skok</div> <div>TEXT</div> </div> <div> <div>RozmiarKola</div> <div>TEXT</div> </div> </div> <div> <div>Bidony</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Pojemnosc</div> <div>TEXT</div> </div> <div> <div>Masa</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> </div> <div> <div>Hamulce</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>TypHamulca</div> <div>TEXT</div> </div> <div> <div>Polozenie</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> </div>	<div> <div>Kaski</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Plec</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Masa</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Rozmiar</div> <div>TEXT</div> </div> </div> <div> <div>Kierownice</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Szerokosc</div> <div>INTEGER</div> </div> </div> <div> <div>Kola</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Rozmiar</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> </div> <div> <div>Liczniki</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>IP</div> <div>TEXT</div> </div> <div> <div>Czas_pracy</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>FunkcjeS</div> <div>TEXT</div> </div> </div>
<div> <div>Lusterka</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Rodzaj_lusterka</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> </div> <div> <div>Narzedzia</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Rodzaj</div> <div>TEXT</div> </div> </div> <div> <div>Oswietlenie</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>KolorS</div> <div>TEXT</div> </div> <div> <div>TrybS</div> <div>TEXT</div> </div> </div> <div> <div>Pokrowce</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> </div> <div> <div>Przedmioty</div> <div> <div>ID</div> <div>INTEGER</div> </div> <div> <div>Ilosc</div> <div>INTEGER</div> </div> <div> <div>Cena</div> <div>TEXT</div> </div> <div> <div>Producent</div> <div>TEXT</div> </div> <div> <div>Nazwa</div> <div>TEXT</div> </div> <div> <div>TypPrzedmiotu</div> <div>TEXT</div> </div> <div> <div>IDparam</div> <div>INTEGER</div> </div> </div>	<div> <div>Przerzutki</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>IloscPrzelozen</div> <div>TEXT</div> </div> <div> <div>Polozenie</div> <div>TEXT</div> </div> </div> <div> <div>Ramy</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Rozmiar</div> <div>TEXT</div> </div> <div> <div>Waga</div> <div>TEXT</div> </div> <div> <div>SrednicaSztycy</div> <div>TEXT</div> </div> <div> <div>RozmiarKola</div> <div>TEXT</div> </div> </div> <div> <div>Rekawiczki</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> <div> <div>Plec</div> <div>TEXT</div> </div> <div> <div>Rozmiar</div> <div>TEXT</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> </div> <div> <div>Siodelka</div> <div> <div>IDparam</div> <div>INTEGER</div> </div> <div> <div>Kolor</div> <div>TEXT</div> </div> <div> <div>Typ</div> <div>TEXT</div> </div> <div> <div>Plec</div> <div>TEXT</div> </div> <div> <div>Material</div> <div>TEXT</div> </div> </div>

▼	Stopki	
	IDparam	INTEGER
	Material	TEXT
	Rodzaj	TEXT
	Kolor	TEXT
▼	Zamowienia	
	ID	INTEGER
	IDtowarow	TEXT
	IloscTowaru	TEXT
	Data	TEXT

8. Harmonogram

	Igor Hałas	Kacper Czerwiński	Błażej Janus
12.04.2021	[T1] Wybranie tematu, dyskusja, burza mózgów.		
13.04.2021	Wybór metody składowania danych.	tworzenie potrzebnych Google docs.	
14.04.2021	Wybór narzędzi do obsługi Bazy Danych.	Szukanie informacji na temat GitHub.	Poszukiwania podobnych rozwiązań.
15.04.2021	Szukanie informacji o zarządzaniu DB.	stworzenie repozytorium GitHub.	
16.04.2021	Praca nad strukturą DB.		Zapoznanie z informacjami na temat technologii projektowania GUI.
17.04.2021			
18.04.2021	Wspólna praca nad dokumentacją i prezentacją. Ustalenie podziału obowiązków.		
19.04.2021	[T2] Przedstawienie postępu w pracy nad projektem.		
20.04.2021	Tworzenie Relacyjnej Bazy Danych.	Tworzenie interfejsów.	Prace nad klasą koszyk.
21.04.2021	Uzupełnianie Bazy danymi do testów.	Wymyślenie działania kreatora roweru.	Prace nad klasą koszyk.
22.04.2021	Praca nad połączeniem DB z programem.	Tworzenie typów wyliczeniowych.	Prace nad klasą koszyk.
23.04.2021	Tworzenie metod do komunikacji z DB.		
24.04.2021	Podsumowanie wykonanych prac, praca nad dokumentami.		
25.04.2021	Wolne/ostatnie poprawki.		
26.04.2021	[T3] Oddanie Prototypu 1 + plan na ten tydzień.		
27.04.2021	Tworzenie metod do komunikacji z DB.	Tworzenie GUI kreatora roweru.	Prace nad GUI koszyka.
28.04.2021	Praca nad obsługą danych pozyskanych z DB.	Tworzenie GUI kreatora roweru.	Prace nad GUI koszyka.
29.04.2021	Praca nad obsługą danych pozyskanych z DB.	Prace nad prototypem klasy kreatora roweru.	Prace nad GUI sklepu.
30.04.2021	Praca nad obsługą danych pozyskanych z DB.	Prace nad prototypem klasy kreatora roweru.	Prace nad GUI sklepu.
01.05.2021	Weekend majowy.		
02.05.2021			
03.05.2021			
04.05.2021			
05.05.2021	Łączenie gotowych elementów programu.	Dopracowanie kreatora rowerów.	Prace nad GUI sklepu.
06.05.2021	Łączenie gotowych elementów programu.	Poprawa wybranych klas (ustalenie 24.04)	Prace nad GUI sklepu.
07.05.2021	Testowanie programu/naprawa błędów.	Poprawa wybranych klas (ustalenie 24.04)	Dopracowywanie GUI.
08.05.2021	Podsumowanie wykonanych prac, praca nad dokumentami.		
09.05.2021	Wolne/ostatnie poprawki.		
10.05.2021	[T4] Oddanie Prototypu 2 + plan na ten tydzień.		
11.05.2021	Optimalizacja Bazy Danych.	Testowanie kreatora roweru.	Testowanie GUI sklepu.
12.05.2021	Implementacja dodatkowych funkcji.	Naprawa ewentualnych błędów.	Poprawki błędów GUI sklepu.
13.05.2021	Testowanie i naprawa błędów.		Testowanie i poprawki GUI koszyka.
14.05.2021	Czyszczenie kodu.	Kompilacja końcowego programu.	Dopracowywanie estetyki GUI.
15.05.2021	Przygotowywanie finalnej wersji prezentacji i dokumentacji.		
16.05.2021	Ostatnie poprawki w dokumentacji i prezentacji		
17.05.2021	[T5] Oddanie Projektu		

Harmonogram z dnia 18.04.2021.

9. Wnioski

Pierwszy tydzień:

- Praca w grupie przy projekcie programistycznym jest dużo bardziej skomplikowana niż nam się wydawało.
- Wymyślenie programu od podstaw nie jest prostym zadaniem. Trzeba rozważyć każdą klasę oraz interfejs oraz wiedzieć jakie funkcje będą potrzebne w programie.
- Błędy popełnione na etapie projektowania architektury obiektowej mogą nieść za sobą poważne konsekwencje na etapie realizacji programu i mogą być trudne do naprawienia.

Jako przykład można podać realizację czytania danych z bazy poprzez listę (`list <string>`). W pewnym momencie wydawało nam się to niewygodne i chcieliśmy to zmienić, jednak w większej części kodu (foldery akcesoria oraz części) było to zaimplementowane, więc zmiana nie wchodziła w grę.

Drugi tydzień:

- Trzymanie się harmonogramu nie jest łatwe, kiedy jest więcej zajęć oraz sprawozdań do napisania. W tamtym czasie do oddania mieliśmy średnio $\frac{3}{4}$ sprawozdania tygodniowo oraz prezentację i dokumentację na PO.
- Podczas projektowania, części kodu pisze się tylko po to, żeby za chwilę coś zmienić i w następstwie usunąć całą poprzednią pracę. Przykładem może być klasa rower oraz jej pochodne (BMX, Szosowy...). Nie zostały one wykorzystane nigdzie w projekcie, mimo tego że były one całkowicie skończone.
- Podział obowiązków jest bardzo ważny i ułatwia realizację projektu. Bez sztywnego ustalenia, kto jest odpowiedzialny za jaką część programu, praca nie rusza do przodu.

Trzeci/Czwarty tydzień:

- W początkowej fazie projektu, podział GitHub na pliki jest wygodny. W późniejszej lepiej zrobić jeden projekt bez podziałów.
- .gitignore bardzo ułatwia pracę nad projektem, nie ma potrzeby zmieniać plików, których nie zmodyfikowaliśmy w kodzie lub bazie danych.
- Podczas pracy w grupie nad dużym projektem bardzo łatwo się zagubić. Szczegółowy podział obowiązków na samym początku pomógłby w znacznym stopniu. Na starcie nie ustaliliśmy wszystkiego, gdyż nie wiedzieliśmy jeszcze jak bardzo projekt się rozbuduje, trzeba było ustalać harmonogram na bieżąco.
- Zrezygnowaliśmy z funkcji usuwania przedmiotów z Bazy Danych, jej zastosowania sprawiłoby, że nie dałoby się wyświetlić zamówień z usuniętym przedmiotem.

Baza Danych:

Zastosowanie bazy danych jako warstwy przechowującej dane, bez odpowiedniej wiedzy na temat ich wykorzystania w takich projektach było dość kłopotliwe, wymagało dużo researchu i uczenia się na błędach. Sama wybranie odpowiedniej struktury bazy, było ciężkie. Finalna struktura bazy też nie jest optymalna, ponieważ dodanie nowego przedmiotu wymaga stworzenie osobnej tabeli z jego parametrami oraz kolumna 'Typ' w tabeli przedmioty jest nadmiarowa (potrzebna podczas łączenia odpowiednich tabel). Można to było rozwiązać tworząc osobną tabelę dla każdego parametru (np. Rozmiar, Waga), takie tabele zawierałyby ID części oraz wartość. Struktura taka wymagałaby dużej ilości połączeń tabel oraz odpowiednich systemów do ich łączenia.

Narzędzia ORM takie jak wykorzystywany Dapper, ułatwiają pracę z Bazą Danych, lecz nie są one w stanie zrobić wszystkiego. Zastosowanie listy w klasie na podstawie której Dapper tworzy obiekty sprawiało, że były one puste. Wymagane było dodatkowe uzupełnienie listy.

Wystąpił też problem z jedną z bibliotek, w jej najnowszej wersji po jej automatycznym pobraniu, program jej nie wykrywał. Zmiana wersji tej biblioteki na starszą rozwiązała problem. Dodatkowo czasami pojawiały się problemy z samą bazą danych, które rozwiązywały się po ponownym utworzeniu bazy danych, co momentami było frustrujące.

GUI:

Praca z Windows Forms w trybie graficznym jest bardzo przyjemna i intuicyjna, kod do stworzonych graficznie formularzy jest generowany automatycznie co znacznie przyspiesza pracę. Ilość elementów możliwych do wykorzystania jest bardzo duża, a same elementy mogą być wykorzystywane na różne sposoby, co przekłada się na wiele możliwości wykorzystania tych elementów. Połączenie GUI z innymi częściami programu nie sprawiło żadnych problemów.