

---

# TEMAT NR 5 - MAGAZYN CZĘŚCI ZAMIENNYCH

Bazy Danych

DR HAB. INŻ. GRZEGORZ MZYK

1 czerwca 2021

---

Autorzy:

EMILIA STARCZYK 249005

MICHAŁ KALETA 248976

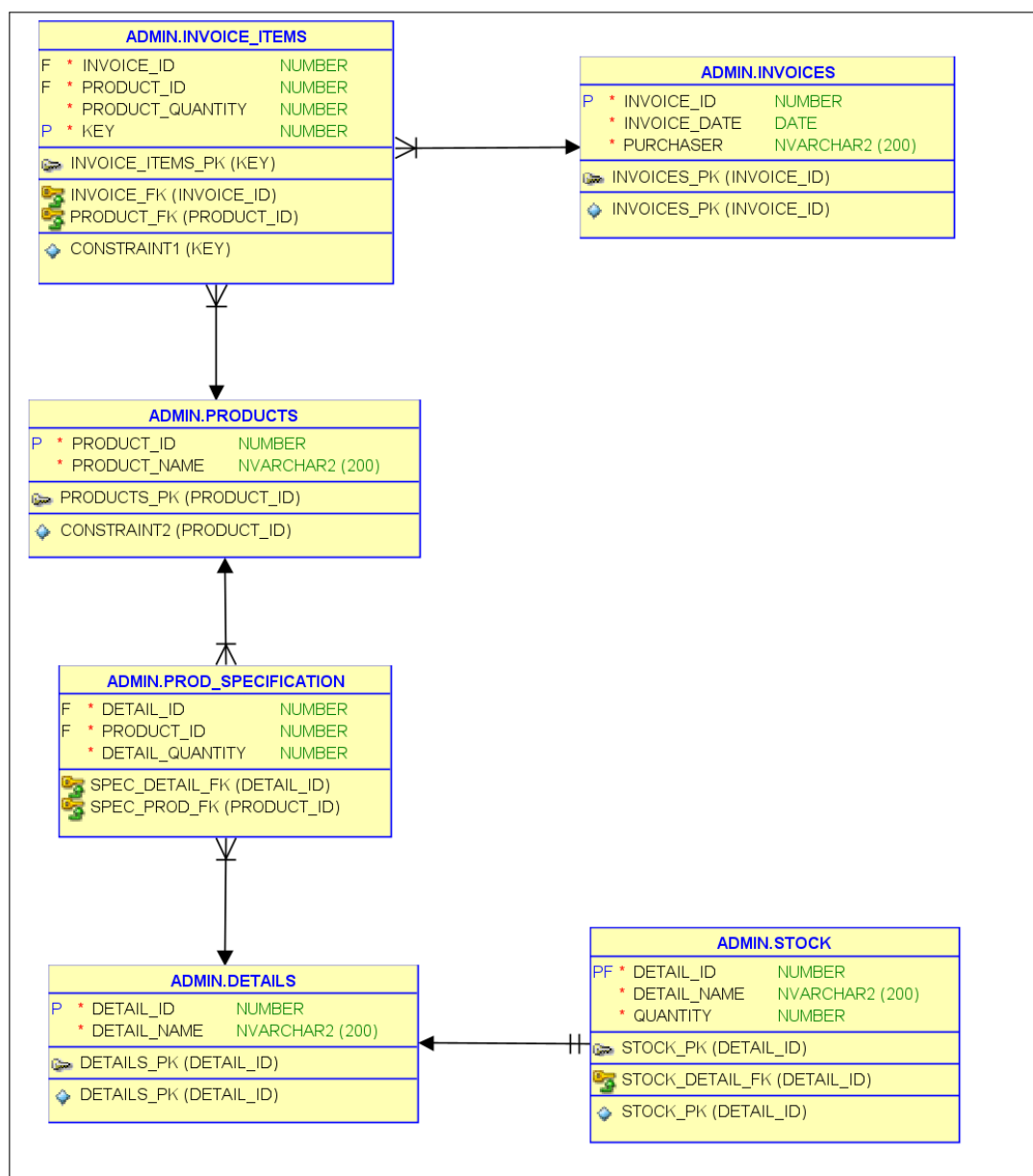
## 1. Cel projektu

Celem projektu jest stworzenie aplikacji pozwalającej na składanie zamówień przez użytkownika. Administrator/właściciel może przeglądać części zamienne do danego produktu oraz może zamówić poszczególne części.

## 2. Założenia projektowe

Projekt zostanie wykonany przy użyciu technologii dostępnych w ramach frameworku ASP.NET w języku C#. Jako baza danych posłuży nam serwer Oracle Database XE 18C.

## 3. Tabele



Rysunek 1: Projekt koncepcyjny bazy danych

### 3.1. Tabela produktów

Products		
PK	product_id	int NOT NULL
	product_name	varchar(200) NOT NULL

Tabela przechowuje wszystkie produkty możliwe do zamówienia. Każdy produkt ma swój unikalny ID.

### 3.2. Tabela faktur

Invoices		
PK	invoice_id	int NOT NULL
	invoice_date	varchar(200) NOT NULL
	purchaser	varchar(200) NOT NULL

Tabela zawiera informacje o tym, kiedy zostało złożone zamówienie. Kolumna purchaser zawiera informacje o nazwie podmiotu, który złożył zamówienie - może być to firma lub osoba fizyczna. Dodanie faktury do tabeli powoduje dodanie produktów zamówionych w ramach tej faktury do tabeli Invoice\_Items.

### 3.3. Tabela pozycji na fakturach

Invoice_Items		
FK	invoice_id	int NOT NULL
FK	product_id	int NOT NULL
	product_quantity	int NOT NULL
PK	key	int NOT NULL

Tabela zawiera informacje o tym, jakie produkty *product\_id* i jaka ich ilość *product\_quantity* została zamówiona w ramach danej faktury *invoice\_id*. Ze względu na użytą technologię wymagane było dodanie klucza głównego do tej tabeli.

### 3.4. Tabela części

Details		
PK	detail_id	int NOT NULL UNIQUE
	detail_name	varchar(200) NOT NULL UNIQUE

Tabela przechowuje wszystkie możliwe detale potrzebne w firmie.

### 3.5. Tabela stanu magazynowego

Stock		
PK,FK	detail_id	int NOT NULL UNIQUE
	detail_name	varchar(200) NOT NULL UNIQUE
	quantity	int NOT NULL

Tabela zawiera informacje o stanie magazynowym. Jeśli dany detal *detail\_id*, jest dostępny na magazynie, to istnieje on w tej tabeli wraz z liczbą dostępnych elementów. Jeśli jego liczba została zredukowana do 0 poprzez produkcję, zostaje on automatycznie usunięty z magazynu.

### 3.6. Tabela specyfikacji produktu

Prod_Specification		
PK,FK	detail_id	int NOT NULL
FK	product_id	int NOT NULL
	detail_quantity	int NOT NULL

Tabela przechowuje detale potrzebne do stworzenia produktu. Jeśli produkt ma więcej niż jedną część, to tabela zawiera więcej rzędów o takim samym product\_id.

## 4. Relacje między tabelami

1. Tabela Invoice\_Items zawiera kolumnę invoice\_id z tabeli Invoices. Relacja między tabelami to 1:N, ponieważ jedna faktura z tabeli Invoices może mieć jeden lub więcej odnośników w tabeli Invoice\_Items.
2. Tabela Invoice\_Items zawiera w sobie również kolumnę product\_id z tabeli Products. Relacja pomiędzy tymi tabelami to również relacja jeden do wielu, ponieważ w wielu fakturach produkty mogą się powtarzać. Zatem Tabela Invoice\_Items może zawierać jeden lub więcej odnośników do jednego wiersza w tabeli Products.
3. Tabela Prod\_Specification zawiera w sobie kolumnę product\_id z tabeli Products. Tutaj również zastosowana jest relacja 1:N, ponieważ jeden produkt może mieć jeden lub więcej elementów.
4. Dodatkowo tabela Prod\_Specification zawiera kolumnę detail\_id z tabeli Details. Relacja pomiędzy tymi tabelami to 1:N, ponieważ w specyfikacji produktów będzie wielokrotnie występował dany element.
5. Tabela Stock zawiera kolumnę detail\_id z tabeli Details. Relacja pomiędzy tymi tabelami to jeden do jednego, ponieważ jeden wiersz w tabeli stock będzie odpowiadał jednemu wierszowi z tabeli Details.

## 5. Funkcjonalności oraz prawa dostępu poszczególnych grup użytkowników

1. Grupa Zamawiających
  - Wgląd w listę dostępnych produktów.
  - Składanie zamówienia na dane produkty.
2. Grupa administratorów
  - Wgląd w listę dostępnych produktów.
  - Podgląd stanów magazynowych.
  - Podgląd złożonych zamówień.
  - Ręczne zamawianie części do magazynu.
  - Podgląd specyfikacji produktów.
3. Inne funkcjonalności:
  - Automatyczne zamawianie części do magazynu, dwa razy dziennie, jeśli występują zamówienia, w których brakuje części.
  - Automatyczne zamawianie części do magazynu (co tydzień) zgodnie z założonymi wartościami.

## 6. Początek prac nad aplikacją

Pracę nad aplikacją rozpoczęliśmy od przemyślenia funkcjonalności aplikacji. Następnie przeszliśmy do zaprojektowania tabel i połączeń między nimi, co zostało zaprezentowane w punktach 3 oraz 4. Kolejnym krokiem było zapoznanie się z narzędziem pracy od firmy Oracle oraz składnią języka SQL.

### 6.1. Tworzenie bazy danych oraz tabel

Stworzenie bazy danych odbywa się poprzez wydanie komendy *CREATE DATABASE nazwa\_bazy\_danych*. Tworzenie tabel w języku SQL odbywa się poprzez korzystanie z komendy *CREATE TABLE nazwa\_tabeli*. Poszczególne kolumny tabeli wymienia się po otwarciu nawiasów. Należy określić rodzaj danych w poszczególnych kolumnach oraz klucze tabeli. Dodatkowo można określić ograniczenia dla poszczególnych kolumn tabeli.

Poniżej znajdują się komendy potrzebne do stworzenia poszczególnych tabel wraz z komendami użytymi do stworzenia wyzwalaczy ustawiających klucz główny każdego kolejnego rzędu o jeden większy od poprzedniego. Zostały do tego użyte sekwencje zaprezentowane w kolejnym punkcie.

#### 6.1.1. Tworzenie tabeli faktur

```
CREATE TABLE INVOICES
(INVOICE_ID NUMBER NOT NULL ,
INVOICE_DATE DATE NOT NULL ,
PURCHASER NVARCHAR2(200) NOT NULL ,
CONSTRAINT INVOICES_PK PRIMARY KEY (INVOICE_ID)
)
```

```
CREATE OR REPLACE TRIGGER INVOICES_TRIGGER1
BEFORE INSERT ON INVOICES
FOR EACH ROW
BEGIN
    SELECT INVOICE_ID_SEQUENCE.NEXTVAL
    INTO :NEW.INVOICE_ID
    FROM DUAL;
END;
/
```

#### 6.1.2. Tworzenie tabeli produktów zamówionych w ramach faktur

```
CREATE TABLE INVOICE_ITEMS
(INVOICE_ID NUMBER NOT NULL ,
PRODUCT_ID NUMBER NOT NULL ,
PRODUCT_QUANTITY NUMBER NOT NULL ,
KEY NUMBER NOT NULL ,
CONSTRAINT INVOICE_ITEMS_PK PRIMARY KEY (KEY) ,
CONSTRAINT INVOICE_FK FOREIGN KEY (INVOICE_ID)
REFERENCES INVOICES (INVOICE_ID) ,
CONSTRAINT PRODUCT_FK FOREIGN KEY (PRODUCT_ID)
REFERENCES PRODUCTS (PRODUCT_ID)
)
```

```

CREATE OR REPLACE TRIGGER INVOICE_ITEMS_TRIGGER1
BEFORE INSERT ON INVOICE_ITEMS
FOR EACH ROW
BEGIN
    SELECT INVOICE_ITEM_SEQUENCE.NEXTVAL
    INTO :NEW.KEY
    FROM DUAL;
END;
/

```

#### 6.1.3. Tworzenie tabeli produktów

```

CREATE TABLE PRODUCTS
(
    PRODUCT_ID NUMBER NOT NULL ,
    PRODUCT_NAME NVARCHAR2(200) NOT NULL ,
    CONSTRAINT PRODUCTS_PK PRIMARY KEY (PRODUCT_ID)
)

```

```

CREATE OR REPLACETRIGGER PRODUCTS_TRIGGER1
BEFORE INSERT ON PRODUCTS
FOR EACH ROW
BEGIN
    SELECT PRODUCT_ID_SEQUENCE.NEXTVAL
    INTO :NEW.PRODUCT_ID
    FROM DUAL;
END;
/

```

#### 6.1.4. Tworzenie tabeli części

```

CREATE TABLE DETAILS
(
    DETAIL_ID NUMBER NOT NULL ,
    DETAIL_NAME NVARCHAR2(200) NOT NULL ,
    CONSTRAINT DETAILS_PK PRIMARY KEY (DETAIL_ID)
)

```

```

CREATE OR REPLACE TRIGGER DETAILS_TRIGGER1
BEFORE INSERT ON DETAILS
FOR EACH ROW
BEGIN
    SELECT DETAIL_ID_SEQUENCE.NEXTVAL
    INTO :NEW.DETAIL_ID
    FROM DUAL;
END;
/

```

#### 6.1.5. Tworzenie tabeli specyfikacji produktów

```
CREATE TABLE PROD_SPECIFICATION
(
    DETAIL_ID NUMBER NOT NULL ,
    PRODUCT_ID NUMBER NOT NULL ,
    DETAIL_QUANTITY NUMBER NOT NULL ,
    CONSTRAINT SPEC_PROD_FK FOREIGN KEY (PRODUCT_ID)
        REFERENCES PRODUCTS (PRODUCT_ID) ,
    CONSTRAINT SPEC_DETAIL_FK FOREIGN KEY (DETAIL_ID)
        REFERENCES DETAILS (DETAIL_ID)
)
```

#### 6.1.6. Tworzenie tabeli magazynu

```
CREATE TABLE STOCK
(
    DETAIL_ID NUMBER NOT NULL ,
    DETAIL_NAME NVARCHAR2(200) NOT NULL ,
    QUANTITY NUMBER NOT NULL ,
    CONSTRAINT STOCK_PK PRIMARY KEY (DETAIL_ID) ,
    CONSTRAINT STOCK_DETAIL_FK FOREIGN KEY (DETAIL_ID)
        REFERENCES DETAILS (DETAIL_ID)
)
```

### 6.2. Tworzenie sekwencji

Użycie sekwencji zapewnia nam unikalność klucza głównego, który będzie zaczynał się od wartości 1 i zwiększał się z każdym kolejnym rzędem o 1. Stworzyliśmy 4 sekwencje, które zostały użyte w wyzwalaczach w poszczególnych tabelach.

```
CREATE SEQUENCE DETAIL_ID_SEQUENCE
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

### 6.3. Dodawanie danych do tabeli

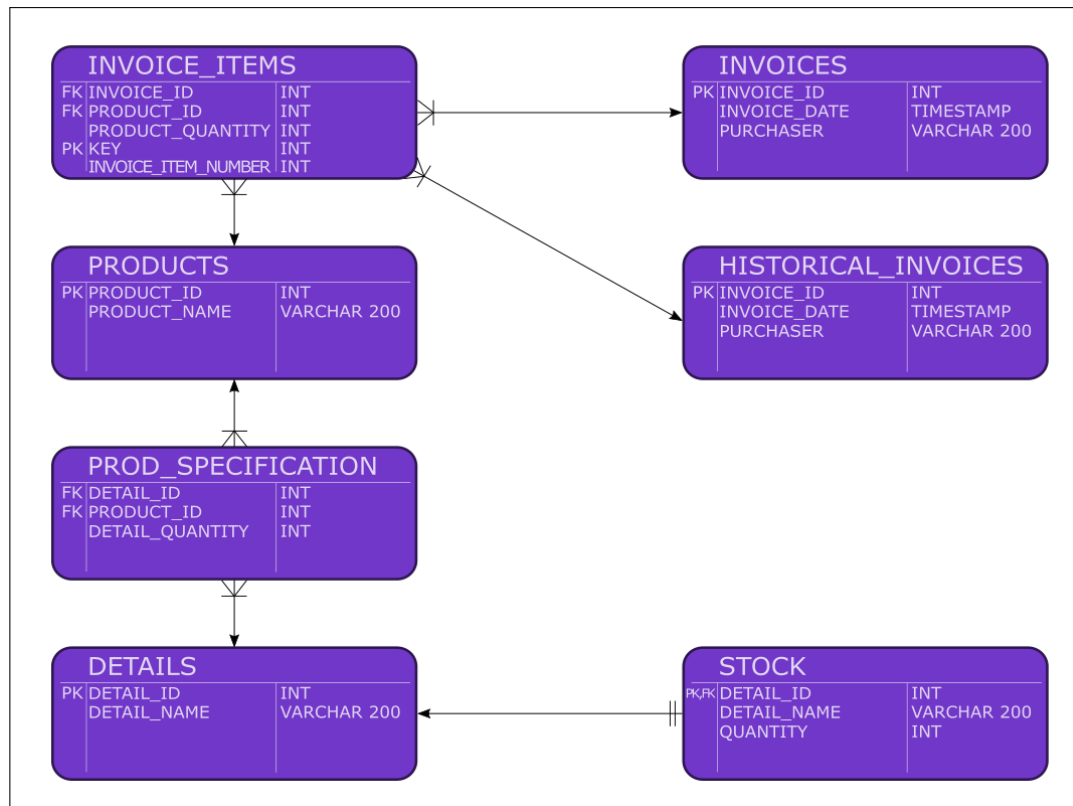
Dodawanie danych do tabeli odbywa się poprzez korzystanie z komendy *INSERT INTO nazwa\_tabeli (kolumna1, kolumna2) VALUES (wartość1, wartość2)*. Komenda ta nie musi zawierać wypisanych nazw kolumn, pod warunkiem, że wpychamy do tabeli całe wiersze danych. Poniżej przedstawione jest przykładowe dodanie produktu do tabeli Products.

```
INSERT INTO Products (product_name) VALUES ( 'odkurzacz ' );
```

## 7. Implementacja

Pracę nad implementacją rozpoczęliśmy od połączenia aplikacji z naszą bazą danych. Skorzystaliśmy z mapowania bazy danych do przestrzeni obiektowej za pomocą Entity Framework, by móc odnosić się do poszczególnych elementów z pozycji programu. W kolejnym kroku zaczęliśmy pracę nad implementowaniem założonych funkcjonalności systemu. Rozpoczęliśmy od możliwości zamawiania produktów, ponieważ potrzebna ona była do pozostałych funkcjonalności. Początkowo system zamawiania polegał na wybraniu produktu z rozwijanej listy i wpisaniu ilości produktu, jaką chcemy zamówić. Okno wpisywania zostało zabezpieczone zgodnie z zasadami walidacji danych. Następnym krokiem było utworzenie strony wyświetlającej potwierdzenie zamówienia oraz strony wyświetlającej części potrzebne do produkcji danego przedmiotu. W tym kroku stworzyliśmy również podział na widoki: widok administratora oraz widok zwykłego użytkownika. Strona wyświetlająca części zamienne została przypisana do widoku administratora,

zaś strona z potwierdzeniem została przypisana do widoku zwykłego użytkownika. Następnie dodaliśmy przeglądanie części znajdujących się w magazynie oraz dodaliśmy możliwość przeglądania zamówień, realizacji zamówień oraz kasowania zamówień. W kolejnym kroku zaimplementowaliśmy widok logowania do panelu administratora. Do panelu administratora dodaliśmy również możliwość ręcznego zamawiania części do magazynu. Administrator uzyskał również możliwość podglądu szczegółów dotyczących faktur oraz części potrzebnych do konkretnego zamówienia. Po pierwszych testach wprowadziliśmy zmiany w panelu użytkownika. Zamiast zamawiania produktów z rozwijanej listy wprowadziliśmy przeglądanie produktów kolejnymi stronami. Do produktów dodaliśmy zdjęcia oraz przycisk dodawania do koszyka wraz z konkretną ilością. W pasku nawigacyjnym umieściliśmy koszyk. Zaimplementowaliśmy dla niego osobną stronę prezentującą zawartość koszyka oraz zajmującą się ostatecznym składaniem zamówienia. Po konsultacji, wprowadziliśmy dodatkową tabelę w bazie danych. Tabela służy do przechowywania historii zamówień. Poniżej przedstawiona jest ostateczna struktura bazy danych znajdującej się w naszym projekcie.



Rysunek 2: Struktura bazy danych

Historical Invoices		
PK	invoice_id	int NOT NULL
	invoice_date	varchar(200) NOT NULL
	purchaser	varchar(200) NOT NULL

Tabela zawiera informacje o tym, kiedy było złożone zamówienie. Kolumna purchaser zawiera informacje o nazwie podmiotu, który złożył zamówienie - może być to firma lub osoba fizyczna. Kolumna z numerem id faktury (invoice\_id) pozwoliła nam na dostanie informacji o szczegółach historycznego zamówienia potrzebnych do implementacji funkcji automatycznego zamawiania części. Oprócz dodania tabeli historycznej dodaliśmy również kolumnę w tabeli "Invoice\_Items". Służy ona do przypisania pozycji produktu na fakturze. Kolejnym krokiem była implementacja systemu automatycznego zamawiania. Funkcja ta jest oparta o odpowiednie komendy w programie Cron. Służy on do ustalania harmonogramu wykonywania danych zadań, przykładowo do cotygodniowego wykonania uzupełnienia magazynu. Ostatnim krokiem w implementacji projektu była implementacja zabezpieczeń bazy danych przed jednoczesnym zapisem użytkowników do tabel. Zabezpieczenie jest o tyle istotne, ponieważ zakładamy w naszym projekcie wielu administratorów/realizatorów zamówień.



## 7.1. System automatycznego zamawiania

System automatycznego zamawiania składa się z dwóch elementów:

- funkcji wywołującej się raz w tygodniu, sprawdzającej, czy na magazynie są odpowiednie ilości części zgodnie z założoną wartością,
- funkcji wywołującej się 2 razy dziennie od poniedziałku do piątku, sprawdzającej, czy ilości produktów na stanie wystarczają do zrealizowania wszystkich obecnie złożonych zamówień.

Obie funkcje zrealizowane są w ramach kodu w języku C#. Ich wykonywanie się jest bazowane na programie Cron, który służy do harmonogramowania zadań. W przypadku ASP-Net skorzystaliśmy z dodatkowej paczki Cronos, aby zrealizować wywoływanie obu funkcji. Pierwsza z funkcji wykonuje sprawdzanie w kilku krokach:

- Pobranie listy części z tabeli Details
- Dla każdej części z tabeli Details, wyszukanie odpowiednika w magazynie(tabeli Stock)
  - jeśli odpowiednik jest na magazynie to: domówienie części (różnicy), jeśli istnieje potrzeba
  - jeśli odpowiednik nie istnieje na magazynie to: zamówienie wskazanej ilości danej części do magazynu

Kod 1: Funkcja tygodniowego sprawdzania magazynu

```
public static void CheckWeeklyStock()
{
    ModelContext m_context = new ModelContext();
    List<Detail> allDetails = m_context.Details.ToList();
    foreach(Detail detail in allDetails)
    {
        Stock stock =
            m_context.Stocks.Where(x => x.DetailId == detail.DetailId).First();
        if (stock!=null)
        {
            if(stock.Quantity<howManyToBe)
            {
                stock.Quantity = howManyToBe;
            }
        }
        else
        {
            m_context.Stocks.Add(new Stock(){
                DetailId = detail.DetailId,
                DetailName = detail.DetailName,
                Quantity = howManyToBe
            });
        }
    }
    m_context.SaveChanges();
    m_context.Dispose();
}
```

Założyliśmy, że tygodniowe sprawdzanie stanu magazynowego będzie wykonywało się w każdy piątek o godzinie 17. Stworzyliśmy zadanie, które wywołuje w sobie funkcję CheckWeeklyStock(). Zadanie to zostało dodane do harmonogramu zadań za pomocą wywołania:

## Kod 2: Dodanie zadania do harmonogramu

```
services.AddCronJob<CronJob1>(c =>
{
    c.TimeZoneInfo = TimeZoneInfo.Local;
    c.CronExpression = @"0 17 * * FRI";
});
```

Druga funkcja opiera się głównie na znalezieniu wszystkich części i ich ilości potrzebnych do realizacji wszystkich zamówień. Podobnie jak poprzednia funkcja, również i ta jest zrealizowana w kilku krokach:

- Pobranie listy wszystkich obecnie złożonych zamówień
- Utworzenie listy wszystkich produktów występujących w zamówieniach
- Dla każdego zamówienia znalezienie wszystkich produktów w nim zawartych
  - jeśli produkt wcześniej występował w liście wszystkich produktów to: zsumowanie ilości tych produktów
  - jeśli produkt wcześniej nie występował to: dodanie go do listy
- Utworzenie listy wszystkich części występujących w zamówieniach
- Dla każdego produktu w liście wszystkich produktów znalezienie wszystkich części
  - jeśli część wcześniej występowała w liście wszystkich części to: zsumowanie ilości tych części( z uwzględnieniem ilości występowania produktu)
  - jeśli część wcześniej nie występowała to: dodanie jej do listy z odpowiednią ilością
- Dla każdej części z listy, wyszukanie odpowiednika w magazynie(tabeli Stock)
  - jeśli odpowiednik jest na magazynie to: domówienie części (różnicy), jeśli istnieje taka potrzeba
  - jeśli odpowiednik nie istnieje na magazynie to: zamówienie wskazanej ilości danej części do magazynu

## Kod 3: Funkcja sprawdzania magazynu wywoływana dwa razy dziennie

```
public static void CheckTwiceADay()
{
    ModelContext m_context = new ModelContext();
    List<Stock> neededDetails = GetAllDetailsFromInvoices();
    foreach (Stock stockDetail in neededDetails)
    {
        Stock stock =
            m_context.Stocks.Where(x => x.DetailId == stockDetail.DetailId).First();
        if (stock != null)
        {
            if (stock.Quantity < stockDetail.Quantity)
            {
                stock.Quantity = stockDetail.Quantity;
            }
        }
        else
        {
            m_context.Stocks.Add(new Stock()
            {
                DetailId = stockDetail.DetailId,
                DetailName = stockDetail.DetailName,
                Quantity = stockDetail.Quantity
            });
        }
    }
    m_context.SaveChanges();
    m_context.Dispose();
}
```

W powyższym kodzie wywoływana jest funkcja GetAllDetailsFromInvoices(). Funkcja znajduje wszystkie potrzebne części do realizacji wszystkich obecnie złożonych zamówień. Kod funkcji przedstawiony jest poniżej.

Kod 4: Funkcja GetAllDetailsFromInvoices()

```
private static List<Stock> GetAllDetailsFromInvoices()
{
    ModelContext m_context = new ModelContext();
    List<InvoiceItem> allInvoiceItems = new List<InvoiceItem>();
    List<Invoice> invoiceList = m_context.Invoices.ToList();
    foreach (Invoice item in invoiceList)
    {
        List<InvoiceItem> tempList =
            m_context.InvoiceItems.Where(x => x.InvoiceId == item.InvoiceId).ToList();
        foreach (InvoiceItem invoiceItem in tempList)
        {
            if (allInvoiceItems.Any(x => x.ProductId == invoiceItem.ProductId))
            {
                var itemToChange =
                    allInvoiceItems.Where(x => x.ProductId == invoiceItem.ProductId).
                        FirstOrDefault();
                itemToChange.ProductQuantity += invoiceItem.ProductQuantity;
            }
            else
            {
                allInvoiceItems.Add(invoiceItem);
            }
        }
    }
    List<Stock> details = new List<Stock>();
    foreach (InvoiceItem item in allInvoiceItems)
    {
        List<ProdSpecification> prodSpecList =
            m_context.ProdSpecifications.Where(x => x.ProductId == item.ProductId).
                ToList();
        foreach (ProdSpecification prodDetail in prodSpecList)
        {
            if (details.Any(x => x.DetailId == prodDetail.DetailId))
            {
                var detailItem =
                    details.Find(x => x.DetailId == prodDetail.DetailId);
                detailItem.Quantity +=
                    (int)(prodDetail.DetailQuantity * item.ProductQuantity);
            }
            else
            {
                details.Add(new Stock() {
                    DetailName =
                        m_context.Details.Where(x => x.DetailId == prodDetail.DetailId).
                            FirstOrDefault().DetailName,
                    DetailId = (int)prodDetail.DetailId,
                    Quantity = (int)(prodDetail.DetailQuantity * item.ProductQuantity)
                });
            }
        }
    }
    return details;
}
```

W przypadku drugiej funkcji sprawdzającej magazyn założyliśmy, że będzie się ona wykonywała od poniedziałku do piątku o godzinie 8.00 oraz 16.00 czasu lokalnego. Ponownie utworzyliśmy zadanie oraz dodaliśmy je do harmonogramu za pomocą wywołania:

Kod 5: Dodanie zadania do harmonogramu

```
services.AddCronJob<CronJob2>(c =>
{
    c.TimeZoneInfo = TimeZoneInfo.Local;
    c.CronExpression = @"0 8,16 * * MON-FRI";
});
```

## 7.2. Zabezpieczenie operacji na bazie danych w przypadku wielu użytkowników

W celu zabezpieczenia bazy danych przed jednoczesnym zapisem przez wielu użytkowników wymagana była najpierw identyfikacja miejsc, w których do takiej sytuacji może dojść. Operacje dodawania danych do tabel są domyślnie obsługiwane przez bazę danych Oracle poprzez stosowanie ekskluzywnej blokady rekordu - oznacza to, że inne operacje wykonywane na tej samej tabeli nie mogą zablokować jej na wyłączność. Taka operacja w połączeniu z uzyskiwaniem klucza głównego z sekwencji utworzonej w bazie danych pozwala na dodawanie wielu produktów w tym samym czasie.

Jednym z miejsc, w których wymagana jest większa kontrola nad dostępem do bazy danych przez poszczególnych użytkowników jest moment wydawania gotowych zamówień. Taka procedura składa się z następujących kroków:

1. Sprawdzenie, czy na magazynie znajduje się wystarczająca ilość detali (wyrażenie “SELECT ...”),
  - (a) Jeśli produktów jest za mało, wyświetlamy komunikat o błędzie, i nie zmieniamy nic w bazie danych,
  - (b) Jeśli produktów jest wystarczająca ilość, przechodzimy do kroku 2.
2. Blokujemy dostęp do tabel w trybie ekskluzywnym (Jeśli nikt inny tego wcześniej nie zrobił),
3. Wywołujemy procedurę “COMPLETE\_ORDER”, która pomniejsza stany magazynowe poszczególnych detali o wymaganą ilość, przenosi zamówienie do tabeli historycznych zamówień oraz usuwa z magazynu wszystkie produkty, których ilość jest równa 0.
4. Zwalniamy ekskluzywną blokadę na tabelach.

Poniżej przedstawione są dwie procedury, które korzystają z blokowania tabel w celu wykonania operacji UPDATE oraz DELETE

Kod 6: Procedura COMPLETE\_ORDER

```
PROCEDURE "COMPLETE_ORDER" (
"ORDER_INVOICE_ID" IN NUMBER) AS

BEGIN
LOCK TABLE STOCK IN EXCLUSIVE MODE;
FOR i in (Select PRODUCT_ID,PRODUCT_QUANTITY From INVOICE_ITEMS Where Invoice_ID=
ORDER_INVOICE_ID) LOOP
FOR j in (Select DETAIL_ID,DETAIL_QUANTITY From PROD_SPECIFICATION Where
PRODUCT_ID = i.product_id) LOOP
UPDATE STOCK SET QUANTITY = QUANTITY - (i.product_quantity*j.detail_quantity)
Where DETAIL_ID=j.detail_id;
END LOOP;
END LOOP;
ARCHIVE_INVOICE(ORDER_INVOICE_ID);
DELETE FROM STOCK WHERE QUANTITY = 0;
COMMIT;
END;
```

#### Kod 7: Procedura ARCHIVE\_INVOICE

```

PROCEDURE "ARCHIVE_INVOICE" (
  "OLD_INVOICE_ID" IN NUMBER ) AS
BEGIN
  LOCK TABLE INVOICES, HISTORICAL_INVOICES IN EXCLUSIVE MODE;
  INSERT INTO historical_invoices SELECT * FROM INVOICES WHERE invoices.invoice_id =
    OLD_INVOICE_ID;
  DELETE FROM INVOICES WHERE invoices.invoice_id = OLD_INVOICE_ID;
  COMMIT;
END ARCHIVE_INVOICE;

```

#### Kod 8: Kod aplikacji wywołujący procedurę COMPLETE\_ORDER

```

if (IsEnoughToComplete)
{
    using (var connection = m_context.Database.GetDbConnection())
    {
        connection.Open();
        var cmd = connection.CreateCommand() as OracleCommand;
        cmd.CommandText = "COMPLETE_ORDER";
        cmd.CommandType = System.Data.CommandType.StoredProcedure;
        OracleParameter param1 = new OracleParameter("ORDER_INVOICE_ID", InvoiceID);
        cmd.Parameters.Add(param1);
        cmd.ExecuteNonQuery();
        connection.Close();
    }
}
try
    m_context.SaveChanges();
catch
    IsEnoughToComplete = false;

```

Dodatkowe zabezpieczenie zostało zaimplementowane na poziomie samej aplikacji. W klasie odpowiedzialnej za dostęp do bazy danych dodane zostało publiczne pole statyczne *IsCompletingOrder*, które jest sprawdzane przed wykonaniem procedury "COMPLETE\_ORDER". Taka implementacja zapewnia ochronę przed niepoprawnym zapisem danych na poziomie samej aplikacji, jak i również bazy danych. Implementacja pola statycznego pozwala również na uniknięcie sytuacji dead-lock.

```

while (ModelContext.IsCompletingOrder)
{
    Thread.Sleep(20);
    counter++;
    if (counter == 10)
    {
        isTimeout=true;
        break;
    }
}

```

## 8. Testowanie

### 8.1. Testowanie procedur w bazie danych

Testowanie procedur zostało przeprowadzone wprost z konsoli. Procedura archiwizacji jest wywoływana wprost z procedury kompletowania i kończenia zamówienia, dlatego były one testowe jednocześnie. Wywołanie odbyło się przez komendę:

**EXECUTE COMPLETE\_ORDER(33);**

Zgodnie z założeniami, stany magazynowe zostały pomniejszone o odpowiednią ilość detali oraz samo zamówienie zostało przeniesione do tabeli zamówień historycznych.

## 8.2. Testowanie interfejsu użytkownika

Testowanie interfejsu zamawiania produktów rozpoczęliśmy od sprawdzenia walidacji wszystkich pól, które muszą zostać wypełnione przez użytkownika. Pozwoliło to wyeliminować wszelkie błędy związane z niepoprawnie wprowadzonymi danymi.

### 8.2.1. Strona “Produkty” oraz dodawanie produktów do koszyka

Pole ilości zamawianych produktów musi zawierać tylko liczby większe od 0, co przetestowaliśmy próbując wpisać wartości ujemne lub litery i znaki. Taka walidacja jest przeprowadzana zarówno po stronie klienta, jak i serwera. W razie przekazania w formularzu ilości produktów mniejszej od 0 lub w innej postaci, polecenie nie jest wykonywane.

### 8.2.2. Koszyk

Następną testowaną funkcjonalnością był koszyk, z którego bezpośrednio składamy zamówienia. Wymagana jest walidacja dwóch rzeczy:

- W koszyku znajduje się co najmniej jeden produkt,
- Pole “Zamawiający” nie jest puste.

W przypadku, gdy koszyk jest pusty, przycisk “Zamów” jest nieaktywny, co pozwoliło zabezpieczyć system przed potencjalnie pustymi zamówieniami. Pole “Zamawiający” zostało oznaczone jako *required*, co sprawiło, że strona nie może zostać zatwierdzona tak długo dopóki w formularzu nie zostanie uzupełnione pole “Zamawiający”. W przypadku ingerencji w kod strony, to samo sprawdzenie wykonywane jest po stronie serwera przed ostatecznym złożeniem zamówienia. W przypadku, gdy któreś sprawdzenie walidacyjne nie zostanie potwierdzone, następuje natychmiastowe porzucenie aktualnego koszyka oraz przeniesienie do strony głównej.

### 8.2.3. Zamawianie części zamiennych przez administratorów

Ostatnim miejscem, w którym należało przeprowadzać walidację wprowadzanych danych jest panel domawiania części przez administratorów, aby uzupełnić stany magazynowe. Typ zamawianego produktu jest wybierany z rozwijanej listy dostępnych produktów, co pozwala uniknąć walidacji tego pola ze względu na brak możliwości przekazania innych danych. Pole ilości zamawianych części musi zawierać liczbę większą od 0. Oznacza to, że nie mogą się tam znaleźć żadne litery ani znaki. Walidacja w tym wypadku również jest przeprowadzana po stronie klienta, w postaci ograniczeń wartości wpisywanych do pola oraz po stronie serwera już po przesłaniu formularza.

## 8.3. Testowanie funkcjonalności systemu automatycznego zamawiania

W celu przetestowania działania obu funkcji automatycznego zamawiania wyczyściliśmy tabelę Stock w bazie danych. Następnie zmodyfikowaliśmy czasy wywołania obu funkcji, tak aby nie musieć czekać przykładowo do piątku (godz. 17). Ilość danej części, jaka powinna się znajdować w magazynie została ustawiona na 50. W kolejnym kroku uruchomiliśmy program i zaczęliśmy na wywołanie pierwszej funkcji. Poniżej zaprezentowany jest efekt przed i po.

Strona główna	Produkty	Przeglądaj zamówienia	Zamów części dodatkowe	Stany magazynowe	Wyloguj
---------------	----------	-----------------------	------------------------	------------------	---------

Stan magazynowy	
Element	Ilość
© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21	
Autorzy: Emilia Starczyk i Michał Kaleta	

Strona główna	Produkty	Przeglądaj zamówienia	Zamów części dodatkowe	Stany magazynowe	Wyloguj
---------------	----------	-----------------------	------------------------	------------------	---------

Stan magazynowy	
Element	Ilość
śruba 1	50
Obudowa odkurzacza	50
dioda LED	50
obudowa do telefonu	50
ekran dotykowy 6"	50
LCD 40"	50
LCD 30"	50
bateria	50
ekran dotykowy 5"	50
półka do lodówki	50
kompresor	50
wtyczka	50
termostat	50
Obudowa suszarka	50
plastik żelazko	50
złącze microUsb	50
złącze aux	50
obudowa lodówka	50
Kabel zasilający	50
plastik głośnik P	50

Rysunek 3: Stan magazynu przed i po wywołaniu funkcji

Aby przetestować działanie drugiej funkcji sprawdzającej dodaliśmy większe zamówienie, przykładowo na 50 laptopów, tak aby wymagane były większe stany magazynowe niż po 50 części. Poniżej przedstawione jest zdjęcie szczegółów zamówienia.

Strona główna	Produkty	Przeglądaj zamówienia	Zamów części dodatkowe	Stany magazynowe	Wyloguj
---------------	----------	-----------------------	------------------------	------------------	---------

Szczegóły zamówienia	
Zamówione produkty	
Zamówiony produkt	Ilość
Laptop	50

Wymagane części		
Detal	Wymagana ilość	Stan magazynowy
śruba 2	600	50
śruba 3	500	50
membrana 1	50	50
membrana 2	50	50
Kabel zasilający	50	50
złącze aux	100	50
Matryca	50	50

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21	Autorzy: Emilia Starczyk i Michał Kaleta
--	--

Rysunek 4: Wymagana ilość części do zrealizowania zamówienia

Na powyższym zdjęciu wyraźnie widzimy, że ilości części na magazynie nie wystarczają do realizacji zamówienia. Każdej części jest po 50 sztuk - stan magazynowy po wywołaniu poprzedniej funkcji. Następnie wywołana została druga funkcja sprawdzająca. Uzupełniła ona poprawnie stany magazynowe, nie tylko dla powyższego zamówienia ale i dla pozostałych zamówień. Poniżej przedstawiony jest efekt działania funkcji.

Strona główna	Produkty	Przeglądaj zamówienia	Zamów części dodatkowe	Stany magazynowe	Wyloguj
---------------	----------	-----------------------	------------------------	------------------	---------

Szczegóły zamówienia	
Zamówione produkty	
Zamówiony produkt	Ilość
Laptop	50

Wymagane części		
Detal	Wymagana ilość	Stan magazynowy
śruba 2	600	626
śruba 3	500	504
membrana 1	50	56
membrana 2	50	52
Kabel zasilający	50	53
złącze aux	100	104
Matryca	50	50

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21

Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 5: Stan magazynowy konkretnych części w zamówieniu

wtyczka	50
termostat	50
Obudowa suszarka	50
plastik żelazko	50
złącze microUsb	50
złącze aux	104
obudowa lodówka	50
Kabel zasilający	53
plastik głośnik P	50
plastik głośnik L	50
nausznik	50
membrana 2	52
membrana 1	56
śruba 4	50
śruba 3	504
śruba 2	626
obudowa pralka	50
Matryca	50

Rysunek 6: Stan magazynowy po wywołaniu funkcji sprawdzającej

## 8.4. Testowanie dostępu wielu osób jednocześnie

W celu przetestowania dostępu wielu osób do bazy danych jednocześnie skorzystaliśmy z narzędzia dostępnego na stronie *loader.io*. Pozwala ono na testowanie odpowiedzi serwerów pod różnym obciążeniem. Testy zostały

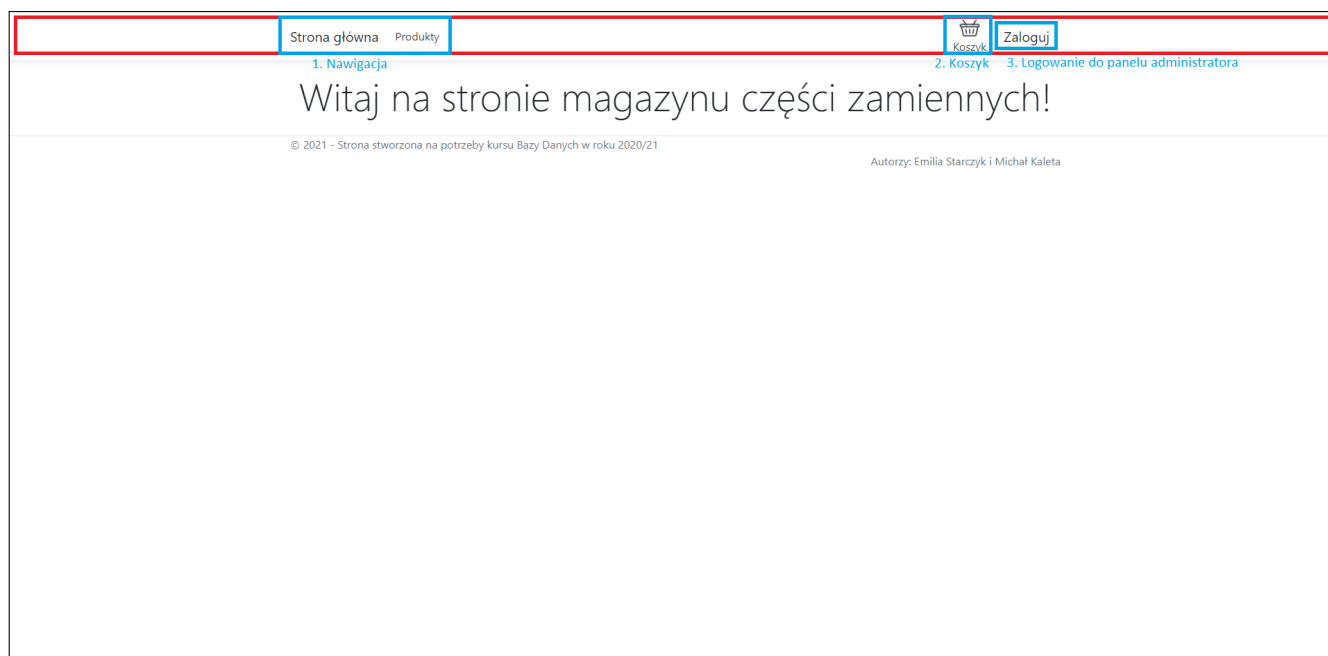


przeprowadzone na stronie z produktami, w taki sposób, aby każde zapytanie wyszukiwało produkty w bazie danych po ich nazwie. Wymagało to odpowiedzi ze strony bazy danych do każdego zapytania. Testy zostały przeprowadzone od 10 do 100 zapytań jednocześnie na przestrzeni 1 minuty. Wszystkie odpowiedzi były poprawne.

## 9. Prezentacja ostatecznego wyglądu aplikacji

### 9.1. Widok zwykłego użytkownika

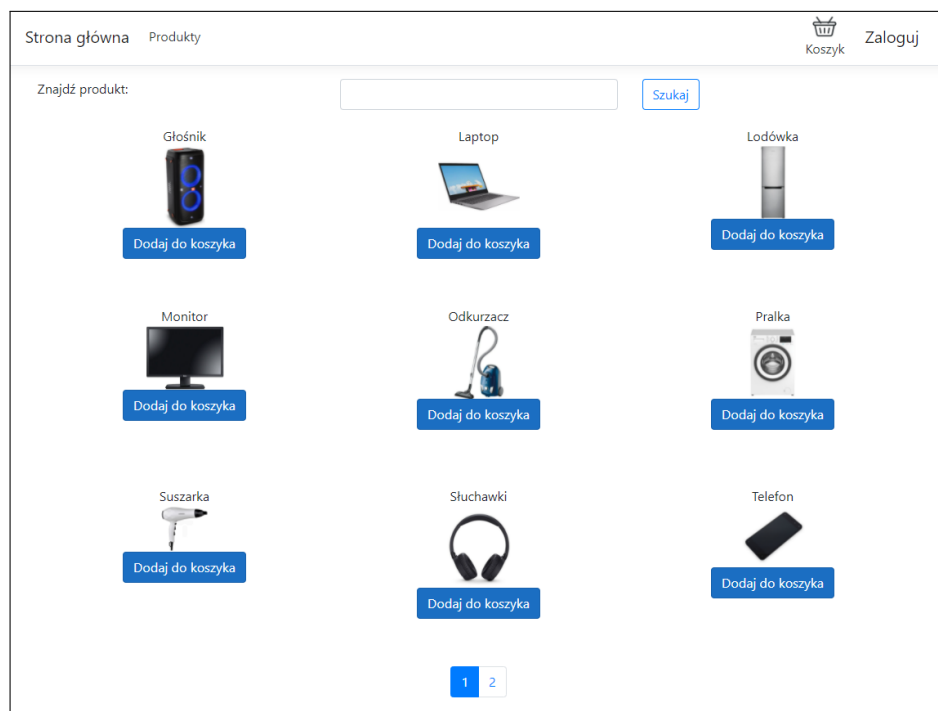
#### 9.1.1. Strona główna



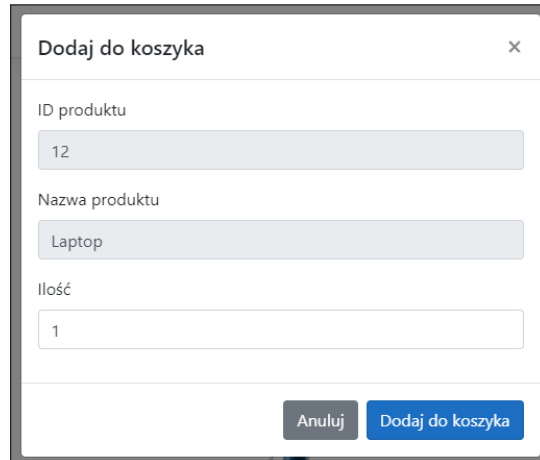
Rysunek 7: Strona główna - strona powitania

Strona główna przedstawia widok, jaki widzimy zaraz po wejściu na stronę www naszego projektu. Na stronie widnieje powitanie. Poruszanie się po stronie zapewnione jest przez nas za pomocą paska nawigacyjnego. Numerem jeden w niebieskiej obramówce oznaczone jest miejsce nawigacji pomiędzy poszczególnymi stronami. Pod numerem dwa znajduje się przycisk przejścia do strony z koszykiem. Numer 3 jest przyciskiem logowania do panelu administratora.

### 9.1.2. Produkty



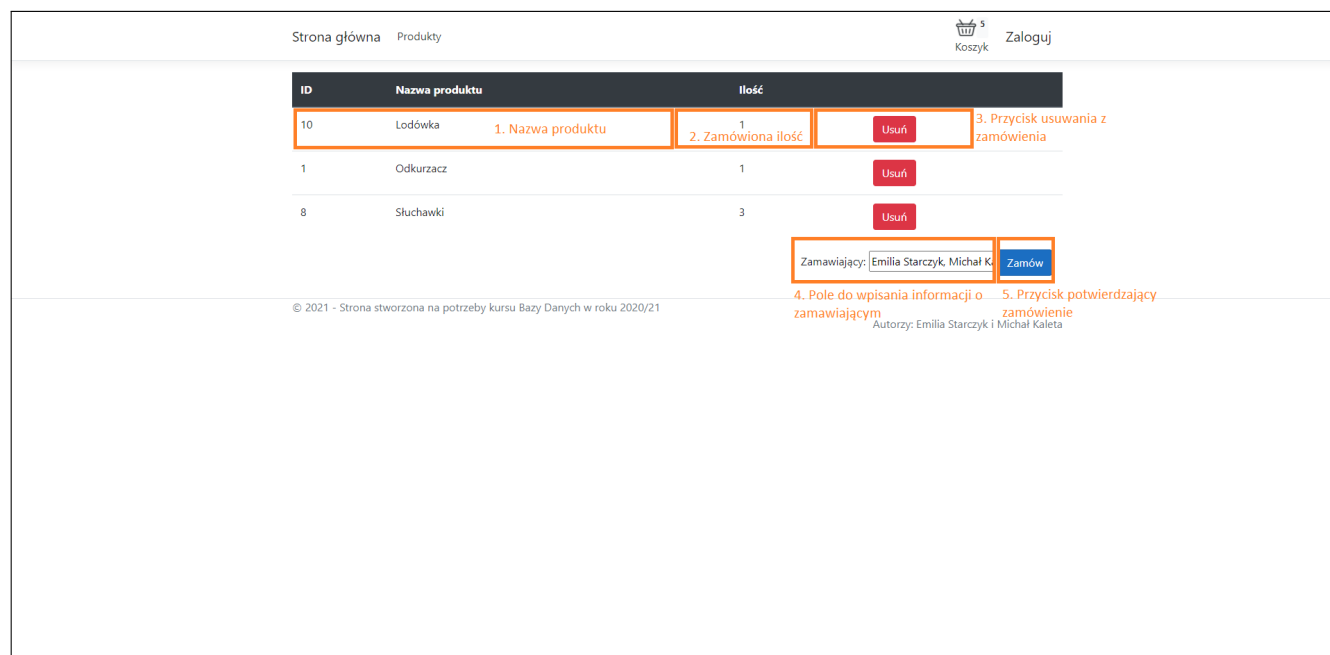
Rysunek 8: Przegląd dostępnych produktów



Rysunek 9: Okno dodawania produktu do koszyka

Strona pozwalająca na przeglądanie produktów dostępnych do zakupu stanowi kluczową część naszego projektu. Potencjalni klienci mogą oglądać jakie produkty są dostępne do zamówienia oraz dodać je do koszyka z tego samego miejsca. Dodawanie do koszyka odbywa się na tej samej stronie, za pomocą okna modalnego, w którym możemy wybrać ilość produktów którą chcemy dodać do koszyka.

### 9.1.3. Koszyk




Rysunek 10: Koszyk

Powyższe zdjęcie przedstawia stronę z zawartością koszyka. Składa się ona z kilku elementów:

1. Nazwa produktu - nazwa zamówionego przez użytkownika produktu.
2. Pole z zamówioną ilością.
3. Przycisk do usuwania produktu z zamówienia.
4. Pole Zamawiający - pole do wpisania informacji o zamawiającym.
5. Przycisk zamów - przycisk potwierdzenia zamówienia. Po wciśnięciu tego przycisku następuje przejście do strony potwierdzenia z informacją o zamówionych produktach.

#### 9.1.4. Potwierdzenie zamówienia

[Strona główna](#) [Produkty](#)  [Zaloguj](#)

Witaj Emilia Starczyk, Michał Kaleta! Twoje zamówienie zostało złożone!

Podsumowanie:

Nazwa produktu	Zamówiona ilość
Lodówka	1
Odkurzacz	1
Słuchawki	3

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21


Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 11: Strona zawierająca potwierdzenie zamówienia

Na powyższym zdjęciu przedstawiona jest zawartość strony potwierdzającej zamówienie produktów. W pomarańczowej ramce przedstawiona jest lista zamówionych produktów.

## 9.2. Widok administratora

### 9.2.1. Zaloguj

[Strona główna](#) [Produkty](#)  [Zaloguj](#)

Panel logowania

Nazwa użytkownika

Hasło

[Zaloguj się](#) [Anuluj](#)

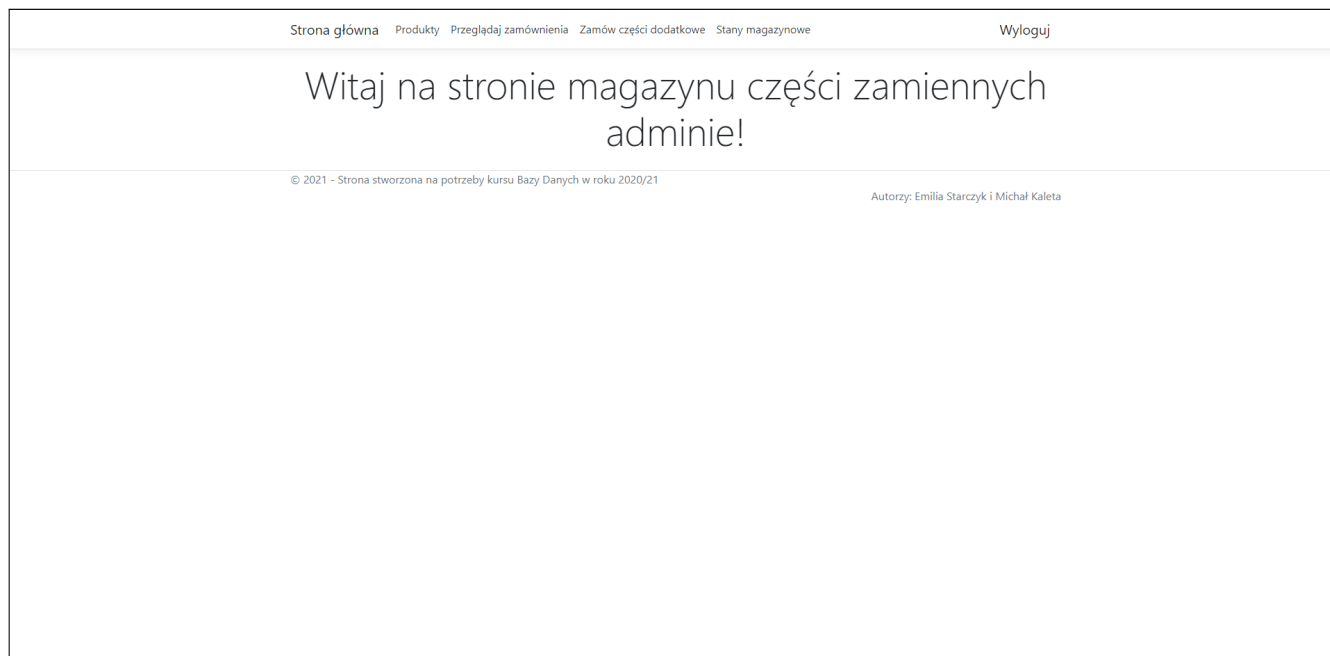
© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21

Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 12: Panel logowania do widoku administratora

W miejsce “Nazwa użytkownika” oraz “Hasło” administrator musi podać prawidłowe dane. Po uzupełnieniu danych i naciśnięciu przycisku “Zaloguj się” następuje przekierowanie do pełnego widoku administratora.

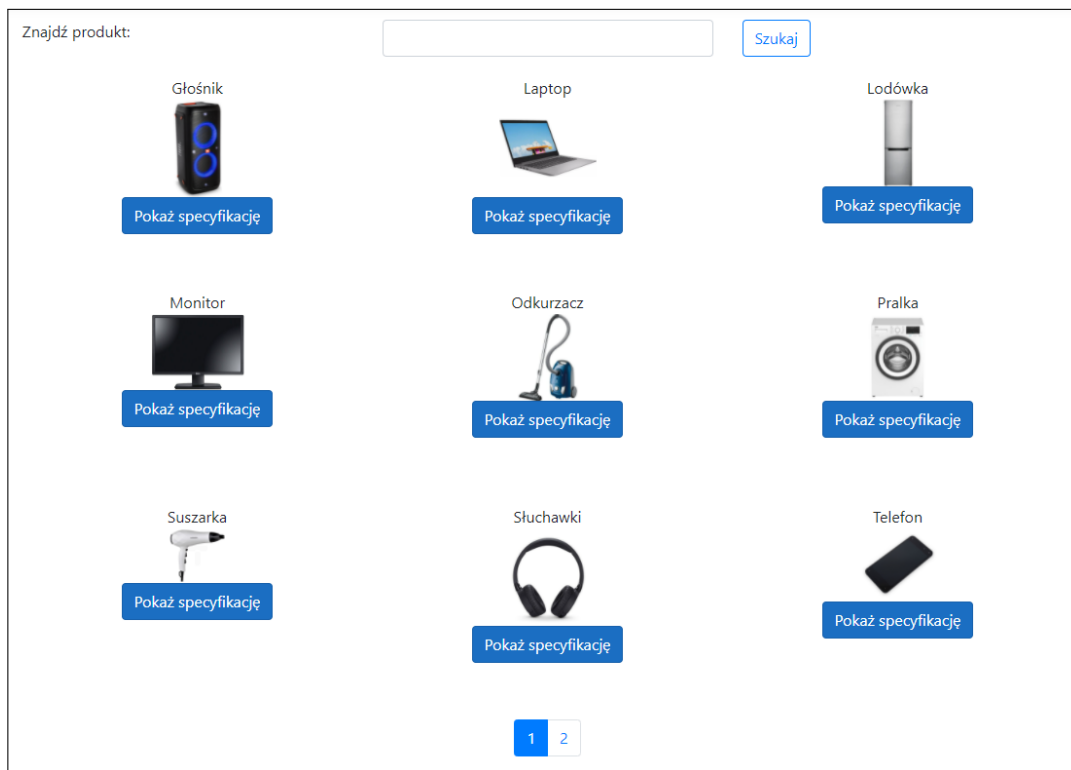
### 9.2.2. Strona główna



Rysunek 13: Strona główna widoku administratora

Widok ten zostaje wyświetlony tuż po zalogowaniu. Jest to widok rozszerzony w stosunku do widoku zwykłego użytkownika. Poruszanie się również zapewnione jest poprzez pasek nawigacyjny. Dodatkowe funkcje/widoki, przyznane administratorowi opisane będą w następnych punktach.

### 9.2.3. Produkty



Rysunek 14: Widok strony podglądu produktów w układzie administratora

Strona wyświetlająca produkty w widoku administratora jest bardzo podobna do zwykłego podglądu - różnicą jest przycisk znajdujący się pod każdym produktem, który w przypadku administratora przekieruje nas na stronę specyfikacji produktu.

### 9.2.4. Specyfikacja produktu

Nazwa części	Potrzebna ilość
śruba 2	12
śruba 3	10
membrana 1	1
membrana 2	1
Kabel zasilający	1
złącze aux	2
Matryca	1

Rysunek 15: Podgląd specyfikacji pojedynczego produktu

Strona wyświetlająca specyfikację produktu pozwala sprawdzić jakie części są wymagane dla danego produktu.

### 9.2.5. Przeglądaj zamówienia

Strona główna   Produkty   Przeglądaj zamówienia   Zamów części dodatkowe   Stany magazynowe   Wyloguj			
Lista zamówień			
Data	Zamawiający		
5/20/2021 5:38:02 PM	Michał Kaleta	Szczegóły zamówienia   Usuń zamówienie	Zrealizuj zamówienie
5/30/2021 6:15:57 PM	Emilia Starczyk, Michał Kaleta	Szczegóły zamówienia   Usuń zamówienie	Zrealizuj zamówienie
© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21			
Autorzy: Emilia Starczyk i Michał Kaleta			

Rysunek 16: Strona zapewniająca podgląd listy złożonych zamówień

Widok tej strony pojawia się po wciśnięciu odpowiedniego przycisku w pasku nawigacyjnym. Administrator ma tutaj dostęp do listy złożonych zamówień. Oprócz tego może podejrzeć szczegóły każdego zamówienia naciskając przycisk “Szczegóły zamówienia”. Dodatkowo wyposażyliśmy administratora w możliwość usuwania złożonego zamówienia oraz możliwość realizacji zamówienia. Naciskając niebieski tekst “Usuń zamówienie” spowodujemy usunięcie konkretnego zamówienia z bazy danych. Naciśnięcie przycisku “Zrealizuj zamówienie” powoduje wywołanie procedury sprawdzającej wymagany stan magazynowy, i w przypadku wystarczających ilości są one usuwane z magazynu a samo zamówienie jest przenoszone do tabeli zamówień historycznych.

- Panel “Szczegóły zamówienia”

Strona główna

Produkty

Przeglądaj zamówienia

Zamów części dodatkowe

Stany magazynowe

Wyloguj

Szczegóły zamówienia

Zamówione produkty

Zamówiony produkt	Ilość
Łódzka	1
Odkurzacz	1
Słuchawki	3

Wymagane części

Detal	Wymagana ilość	Stan magazynowy
kompresor	1	0
połka do lodówki	4	0
obudowa lodówki	1	0
Kabel zasilający	2	1
wtyczka	2	0
diody LED	6	0
środa 4	10	0
środa 3	4	5
Obudowa odkurzacza	1	0
membrana 1	6	1
nausznik	6	0
złącze aux	3	2
złącze mikroUSB	3	1
środa 2	18	10
środa 1	12	3

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21

Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 17: Szczegóły pojedynczego zamówienia

W panelu ze szczegółami zamówienia administrator może podejrzeć ilości zamówionych produktów. Oprócz tego wyświetlana jest lista wszystkich wymaganych części do konkretnego zamówienia w zestawieniu z ilością dostępnych części na magazynie.

### 9.2.6. Zamów części dodatkowe

Strona główna Produkty Przeglądaj zamówienia Zamów części dodatkowe Stany magazynowe Wyloguj

**Dodaj elementy do zamówienia:**

1. Rozwijana lista 2. Okno ilości

śruba 1 ilość: 200

membrana 1 ilość: 100

3. Przyciski dodawania lub odejmowania kolejnych produktów

4. Przycisk do zamawiania części

Zamów

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21

Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 18: Widok strony pozwalającej na ręczne zamawianie części do magazynu

Panel ten składa się z kilku elementów:

1. Rozwijana lista - lista wyboru części do zamówienia
2. Okno ilości - pole pozwalające wpisać ilość produktu do zamówienia
3. Przyciski dodawania lub odejmowania kolejnych części. Po wciśnięciu przycisku “+” dodajemy kolejne pole dla kolejnej części. Po wciśnięciu przycisku “-” zmniejszamy ilość rozwijanych list z której wcześniej wybraliśmy produkt.
4. Przycisk “Zamów”. Po wciśnięciu przycisku produkty zostają zamówione do magazynu i następuje automatyczne przekierowanie na stronę “Stany magazynowe”.



### 9.2.7. Stany magazynowe

Strona główna

Produkty

Przeglądaj zamówienia

Zamów części dodatkowe

Stany magazynowe

Wyloguj

Stan magazynowy


Element	Ilość
membrana 2	2
plastik głośnik L	1
plastik głośnik P	1
Kabel zasilający	1
złącze aux	2
złącze microUsb	1
śruba 2	10
śruba 1	3
śruba 3	5
membrana 1	1
Matryca	1

© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21

Autorzy: Emilia Starczyk i Michał Kaleta

Rysunek 19: Widok strony przedstawiającej zawartość magazynu producenta przedmiotów

### 9.2.8. Wyloguj

Strona główna	Produkty	 Koszyk	Zaloguj
Zostałeś wylogowany! Zaraz nastąpi przekierowanie na stronę główną...			
© 2021 - Strona stworzona na potrzeby kursu Bazy Danych w roku 2020/21			
Autorzy: Emilia Starczyk i Michał Kaleta			

Rysunek 20: Widok strony po wciśnięciu przycisku “Wyloguj” w pasku nawigacyjnym

## 10. Efekty końcowe

Pod poniższym linkiem znajduje się nagranie działania całej aplikacji. W nagraniu widać również przejścia między wszystkimi stronami internetowymi zawartymi w projekcie.