POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

PROJEKT Z BAZ DANYCH 2

Sklep internetowy z elektroniką użytkową

Prowadzący:
Dr inż. Roman Ptak

Termin: Wtorek 13:15
Grupa nr 7:
Filip Gajewski, 236597
Andrzej Gierlak, 236411
Patryk Skowroński, 237454

Spis treści

1 Wstęp

Celem naszego projektu było zaprojektowanie oraz implementacja aplikacji webbowej, która realizowałaby funkcje sklepu internetowego z elektroniką użytkową, wspierającą mały, fizyczny sklep zatrudniający maksymalnie kilka osób. Po przez przeglądarkę internetową użytkownik naszego sklepu będzie mógł zarejestrować się w sklepie, przeglądać ofertę produktów dostępnych w sklepie oraz składać na nie zamówienia, które będzie mógł odbierać osobiście jak i zamówić z wysyłką kurierem.

2 Analiza wymagań

2.1 Opis zasobów ludzkich

Sklep funkcjonuje jako małe przedsiębiorstwo, mieszczące się w lokalu pełniącym rolę punktu sprzedaży oraz magazynu. Sprzedaż do tej pory była prowadzona stacjonarnie oraz za pośrednictwem serwisów aukcyjnych, a dane dotyczące zamówień i użytkowników były zarządzane przy pomocy arkuszy MS Excel. Zleceniodawca nie planuje bezpośredniej migracji posiadanych danych do nowego systemu. Sklep mieszczący się w lokalu dalej realizuje sprzedaż towarów, jednak również oferuje zakup, wysyłkę przez internet i odbiór osobisty w lokalu.

Pracownik sklepu może dodawać do katalogu produktów nowe pozycje oraz zmieniać ceny już dodanych produktów, w ramach czasowych zniżek lub na stałe. Może on dodatkowo modyfikować dane o produktach, zniżkach.

Klient może utworzyć swoje konto w bazie użytkowników sklepu oraz może on przeglądać całą ofertę sklepu. Klient może sprawdzić, ile łącznie wynosi wartość całego zamówienia oraz wartość poszczególnych produktów z uwzględnieniem przysługujących mu rabatów. Po złożeniu zamówienia otrzymuje ono swój unikalny numer identyfikacyjny. W razie ewentualnych trudności z realizacją zamówienia pracownik poinformuje o tym klienta.

2.2 Przepisy i strategia firmy

Pracownik odpowiada za zgodność bazy danych z realnym zasobem produktów. Do jego obowiązków należy dbanie o poprawność danych (aktualne ceny) oraz obecny stan na magazynie (czy dany produkt jest dostępny). Klient po utworzeniu konta, może złożyć zamówienia, które realizuje pracownik. Klient w trakcie realizacji zamówienia ma do wyboru dokonanie płatności z góry na konto bankowe sklepu lub zapłatę przy odbiorze towaru w fizycznym sklepie. Jeżeli zdecyduje się na płatność z góry to system wygeneruje mu dane do przelewu na konto sklepu, to jest poda mu numer konta bankowego, adres firmy oraz jej pełną nazwę i tytuł płatności. Po realizacji danego zamówienia, pracownik zobowiązany jest do aktualizacji danych (zmiana ilości dostępnego produktu). Pracownik ma dostęp do konta bankowego sklepu, na którym sprawdza czy wpłynęła opłata za złożone zamówienie. Jeśli tak to pracownik zmienia w bazie danych status zamówienia na "Zapłacone" oraz przygotowuje przesyłkę do wysyłki klientowi lub do odbioru osobistego w sklepie. W bazie danych sklepu będą przechowywane:

- Wszystkie dane klienta podane w procesie rejestracji jego konta to jest imię, nazwisko, data urodzenia, numer telefonu, adres email, nazwa użytkownika.
- Dane transakcji takie jak data, kwota do zapłaty, identyfikator klienta który złożył zamówienie, forma płatności, zawartość koszyka, status transakcji.

- Dane produktu zawierające jego nazwę, nazwę producenta, cenę, okres gwarancyjny producenta, nazwę kategorii, opis słowny.

2.3 Dane techniczne

System jest realizowany w formie strony internetowej sklepu. Będzie ona znajdowała się na wykupionym przez sklep zewnętrznym hosting. W ofercie sklepu znajduje się około 500 różnych rodzajów produktów których dostępność musi być ręcznie aktualizowana przez pracowników co wymusza przeprowadzanie częstych inwentaryzacji. Analiza statystyk pochodzących z serwisu aukcyjnego wykazuje iż liczbę użytkowników korzystających jednocześnie z aplikacji może być około 1000. Najliczniejszą z tabel w naszej bazie danych będzie tabela przechowująca informacje o złożonych w sklepie zamówienia. Zgodnie z prawem muszą być one przechowywane przez 5 lat od momentu zapłaty. Zakładając, że dziennie w systemie będzie składane około 30 zamówień to w przeciągu roku wygeneruje to prawie 55 tysięcy nowych rekordów. Przewidujemy, że sprzedaż wzrośnie dwukrotnie w związku z wprowadzeniem możliwości internetowych zamówień. Baza musi posiadać bezpieczny margines na dodatkowe rekordy w razie ewentualnych wyprzedaży zatem będzie przystosowana do przechowywania maksymalnie 200 tysięcy rekordów w tabeli zamówień.

2.4 Lista wymagań funkcjonalnych

1. Klient i pracownik może przeglądać produkty dostępne w ofercie sklepu.
2. Przeglądanie produktów o wybranych przez klienta lub pracownika parametrach.
3. Klient dodaje produkty do koszyka oraz edytuje jego zawartość.
4. Klient składa zamówienie na produkty znajdujące się w koszyku.
5. Klient ma możliwość założenia konta w sklepie.
6. Pracownik dodaje i usuwa produkty z bazy, w oparciu o aktualny stan magazynu.
7. Pracownik modyfikuje parametry produktów, ich aktualną cenę, dostępność oraz poprawia ewentualne błędy w bazie danych(wynikłe np. ze złego wprowadzenia danych).
8. Klient ma możliwość edycji swoich danych osobowych konta użytkownika.
9. Pracownik ma możliwość wprowadzenia okresowych, określonych zniżek na wybrane produkty.
10. Pracownik może oglądać i edytować zamówienia użytkownika.

2.5 Lista wymagań niefunkcjonalnych

1. Aplikacja internetowa w architekturze MVC (trójwarstwowej).
2. Wyłączny dostęp pracownika do edycji bazy produktów oraz użytkowników.
3. Wykorzystanie technologii ASP.NET oraz relacyjnych baz danych MySQL.
4. Planowana ilość różnych rodzajów (klas) produktów, dostępnych w sklepie wynosi 500.

5. Estetyczny oraz intuicyjny interfejs graficzny.
6. Automatyczne obliczanie należnej kwoty z uwzględnieniem przysługujących zniżek.
7. Generowanie historii zamówień.

3 Projekt systemu

3.1 Projekt bazy danych

Na potrzeby naszego projektu stworzyliśmy dwa modele bazy danych: konceptualny oraz fizyczny. Oba zostały przygotowane w programie Visual Paradigm, dzięki czemu kod SQL niezbędny do utworzenia tabel bazy danych został wygenerowany automatycznie na podstawie modelu fizycznego.

Oprócz tabel oraz relacji między nimi musieliśmy też zaprojektować dla bazy danych indeksy, widoki, procedury składowe, sekwencje oraz wyzwalacze. W MySQL nie można w prost zdefiniować sekwencji, jednak w naszej bazie są one realizowane na kluczach głównych i obcych poprzez nadanie im atrybutu AUTO_INCREMENT.

Indeksy założone w bazie danych:

- Invoice na kolumnę NIP
- Order, pojedyncze indeksy na kolumny SaleDate, Status oraz CustomerId
- Customer, pojedynczy indeksy na kolumne Login oraz podówjny indeks odpowiednio na kolumny LastName oraz FirstName
- Discount, indeks na na kolumnę EndDate
- Category, indeks na kolumnę Name
- ProductsInStore, pojedyncze indeksy na kolummny Status oraz ProductId
- Product, pojedyncze indeksy na kolumny Price, Name, CategoryId
- Address, indeks na kolumne PostCode

Widoki w bazie danych

- top_products - przechwouje najpopularniejsze produkty sklepu sprzedane do tej pory
- employee_discount - informuje, które zniżki zostały wprowadzone przez którego pracownika
- orders_in_progress - pokazuje, które zamówienia są właśnie realizowane tzn. że ich status to "Realizowane"
- products_sell_out - przechowuje informacje których produktów brak na magazynie
- returned_products - widok z produktami, które zostały zwrócone do sklepu

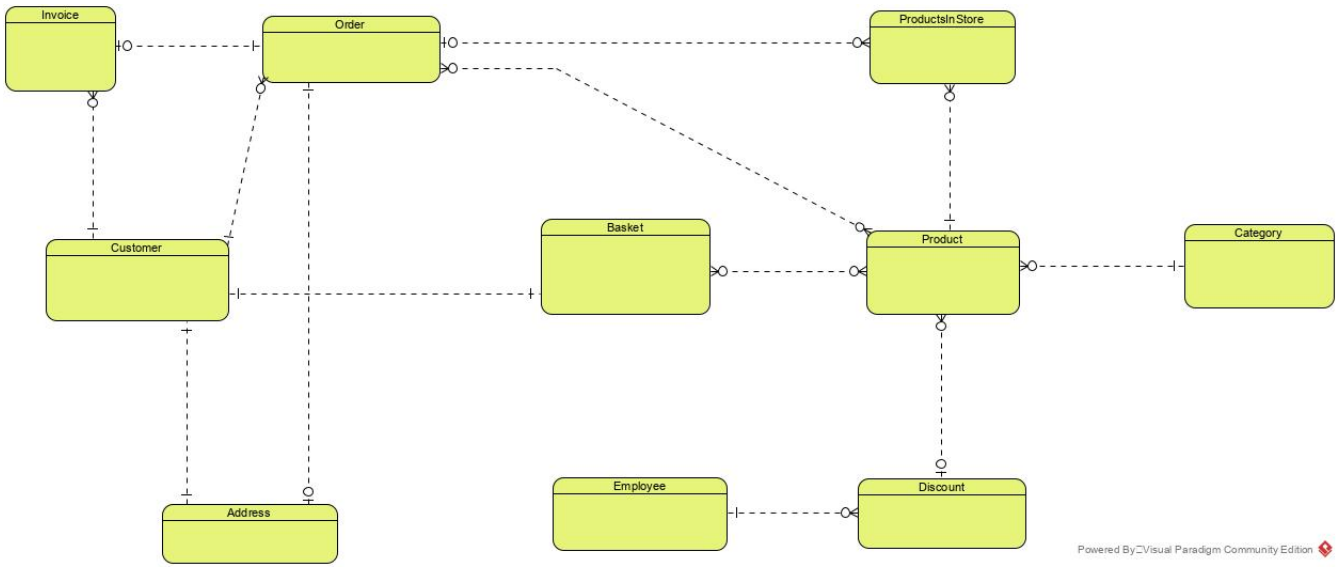
Procedury składowe w bazie danych

- make_discount - procedura przeceniająca produkty, które podlegają podanej jako argument zniżce
- basket_to_order - procedura przenosząca produkty z koszyka klienta do nowo utworzonego zamówienia.

Wyzwalacze w bazie danych

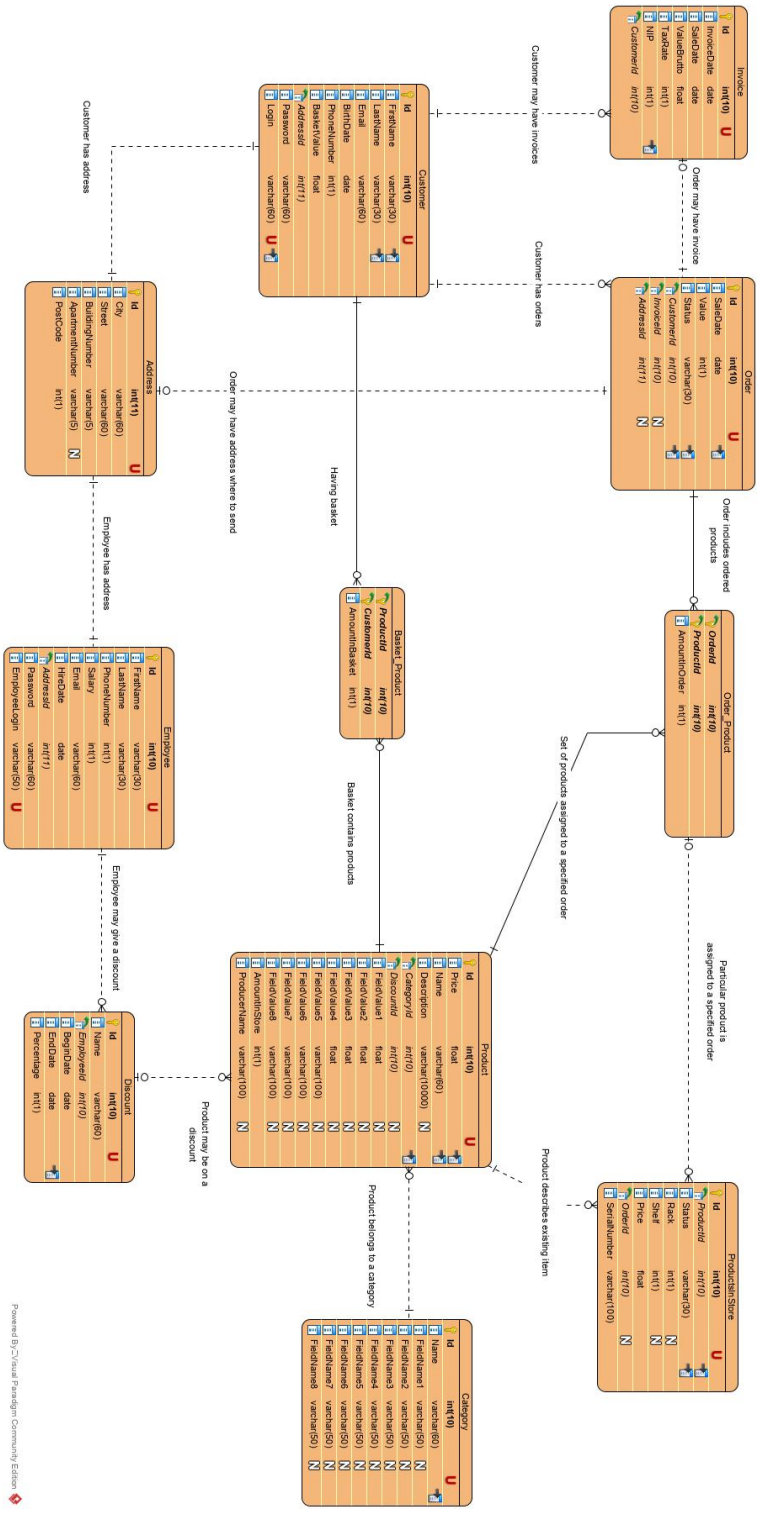
- add_to_basket - powiększa wartość koszyka w chwili dodania do niego nowego produktu
- remove_from_basket - zmniejszający wartość koszyka po usunięciu z niego całego rekordu produktu
- update_basket - modyfikuje wartości koszyka po modyfikacji ilości produktów w zamówieniu.
- add_to_order - powiększa wartość zamówienia w chwili dodania do niego nowego produktu
- remove_from_order - zmniejsza wartość zamówienia po usunięciu całego rekordu produktu z zamówienia
- update_order - modyfikuje wartość zamówienia po modyfikacji ilości produktów w zamówieniu
- remove_from_store - aktualizuje stanu magazynu po dodaniu produktu do zamówienia
- remove_from_store_after_update - aktualizuje stanu magazynu po zmianie ilość produktów w zamówieniu
- after_return_to_store - aktualizacja stanu magazynu po zwrocie towaru

3.1.1 Model konceptualny



Rysunek 1: Model konceptualny

3.1.2 Model fizyczny



Rysunek 2: Model fizyczny

3.1.3 Zabezpieczenia na poziomie bazy danych

3.2 Zabezpieczenia

Zagrożenia:

- Niezabezpieczone hasłem aplikacje Web do zarządzania
- Informacje o sieci i zabezpieczeniach na stronach Web.
- Nieświadomi pracownicy
- Luki w bazie SQL w postaci możliwych SQL injection, niezabezpieczony Listener

Sposoby na zabezpieczenie bazy danych:

- Korzystanie z comboboxów w celu ograniczenia możliwości SQL injection
- Unikanie struktury konkatenacji- przekazywanie wartości przez argumenty
- Zabezpieczenie hasłem kont użytkowników i pracowników
- Zabezpieczenie bazy danych silnym hasłem
- Zabezpieczenie aplikacji Web silnym hasłem
- Zmiana wszystkich domyślnych haseł na silne, nowe
- Usunięcie dostępu do domyślnych użytkowników(np. root), lub usunięcie ich z bazy(np. anonymous accounts)
- Kodowanie haseł?
- Wyłączenie podglądu plików lokalnych(disable “LOAD DATA LOCAL INFILE” command)
- Nadanie wyłącznie wymaganych/koniecznych uprawnień (pozostawienie file privilege, process, super wyłącznie na koncie admin)
- Wyłączenie lub zabezpieczenie komendy ”SHOW DATABASES”
- Zmienić domyślny port mySQL server
- Posprzątać po starcie- usunąć bazę testową, wyczyścić historię, zablokować do niej dostęp

3.3 Uprawnienia

W bazie przewidujemy dwa typy kont: konto Klient, konto Pracownik, gdzie konto Pracownik przy aktualnych założeniach(pełen dostęp do bazy danych) pełni też rolę administracyjną- posiada wszystkie uprawnienia i dostępy.

Podział na trzy Role: anonimowy klient, zalogowany klient, pracownik Uprawnienia nie będą przypisywane bezpośrednio do użytkowników, a za pomocą mechanizmu ról.

Dostęp będzie przydzielany do poszczególnych propert- użytkownik nie będzie miał dostępu do całej tabeli/encji, a tylko do poszczególnych pól lub będzie upoważniony tylko do wywołania przypisanych mu procedór

Property wymagane dla anonimowego klienta:
Select:

- Produkt
- Category
- Discount
- Customer(ściśle ograniczony)

Insert:

- Customer
- Address

Property wymagane dla zalogowanego klienta:
Select:

- Invoice
- Order (tylko dla siebie)
- Full customer(edycja danych)
- OrderProduct
- Product (przeglądanie)
- Category
- BasketProduct
- Discount(w zależności od dalszej implementacji w aplikacji)
- Address

Insert:

- Basketproduct
- Orderproduct
- Order
- Invoice
- Address

Update:

- Customer(np zmaina email)
- Basketproduct(zmiana ilości w zamówieniu)

- Order.status
- ProductsInStore(jeśli decydujemy się na aktualizację ilości zaraz po złożeniu zamówienia)
- Address

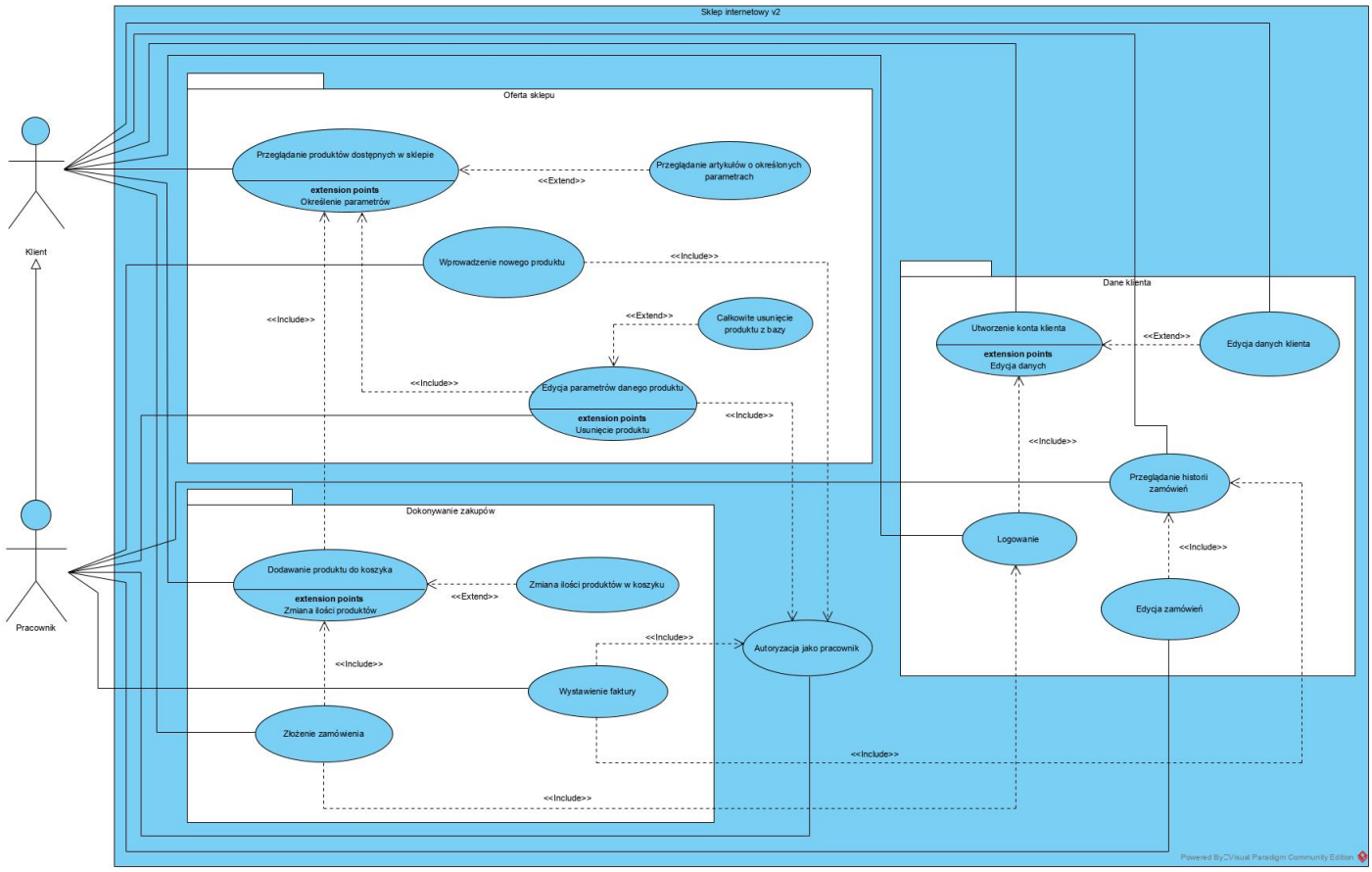
Delete:

- Orderproduct
- Basketproduct

Dostęp będzie weryfikowany i poprawiany w trakcie tworzenia aplikacji- po sprwadzeniu czego potrzebujemy do zapytań, w celu określenia minimalnego potrzebnego dostępu.

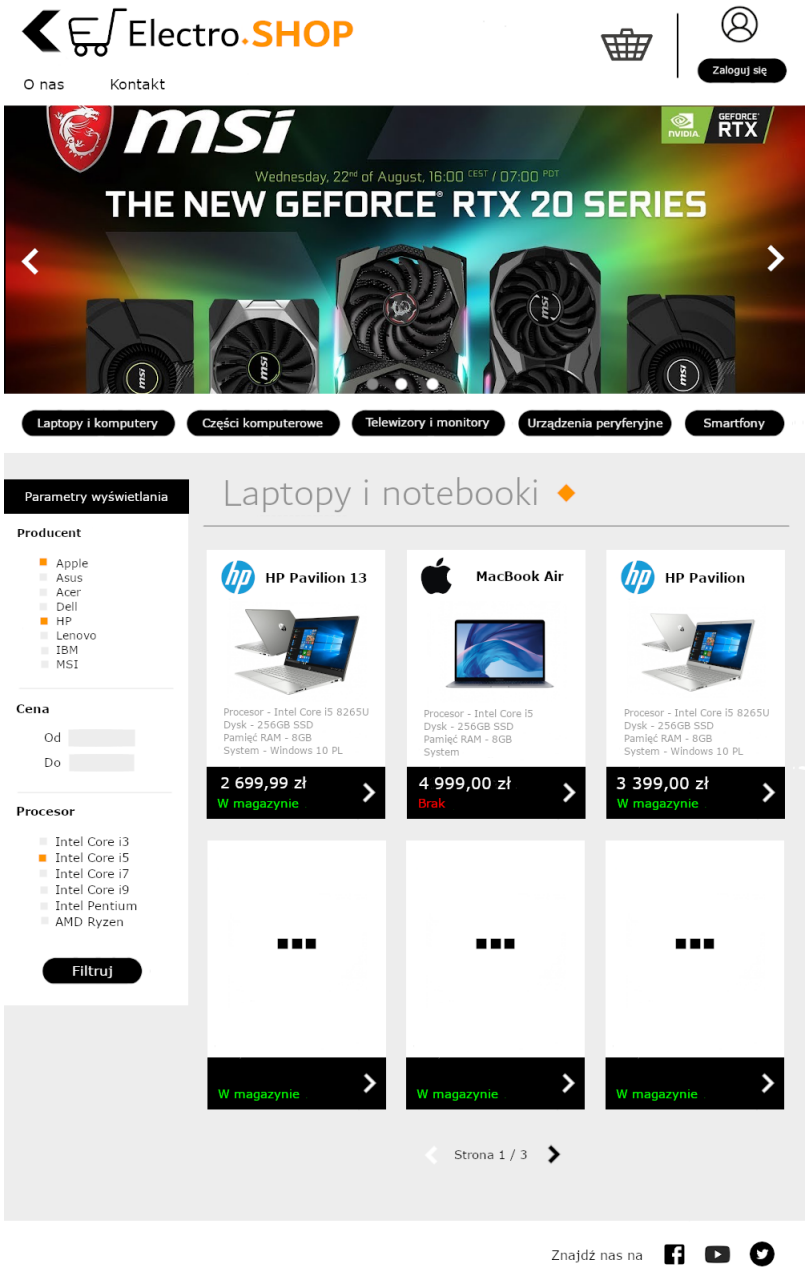
3.4 Projekt aplikacji użytkownika

3.4.1 Przypadki użycia

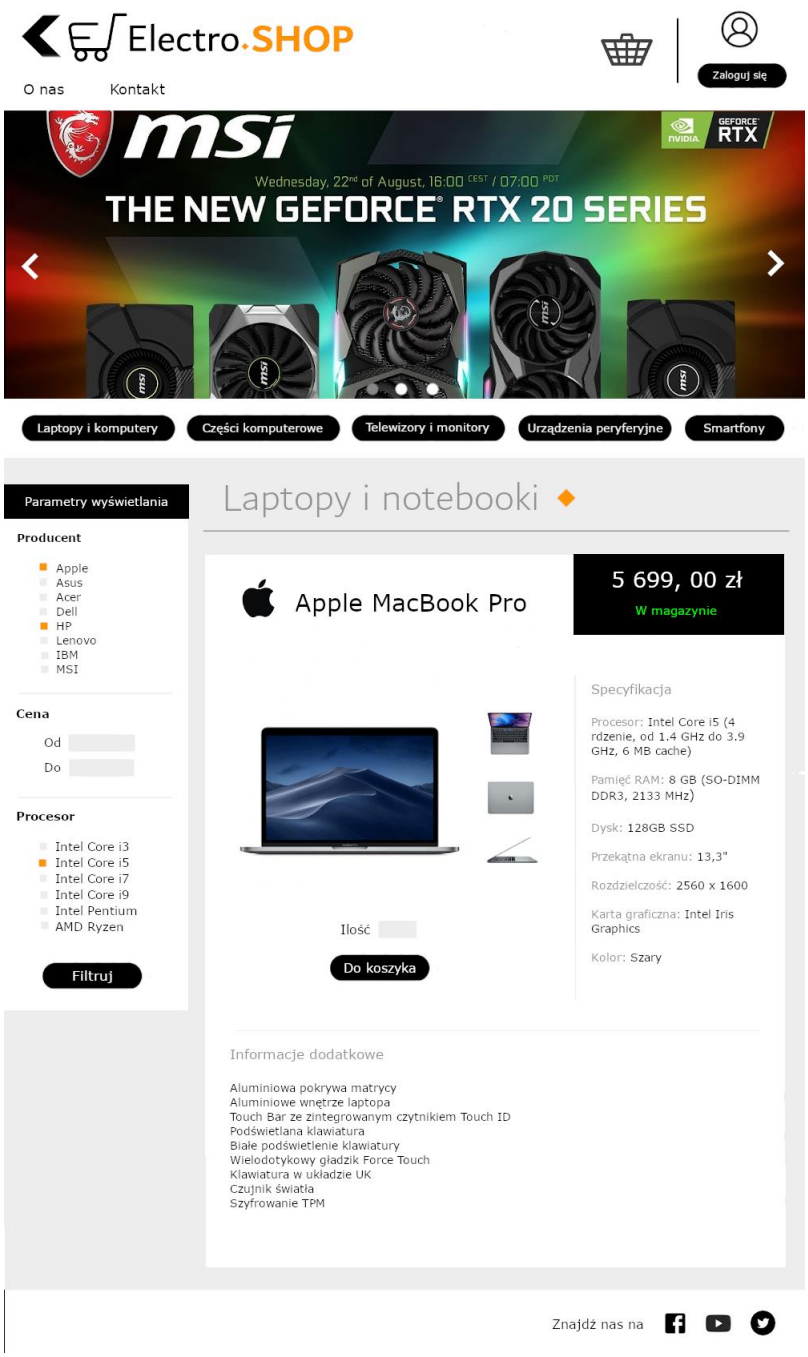


Rysunek 3: Diagram przypadków użycia

3.4.2 Mock-up'y interfejsu użytkownika



Rysunek 4: Widok główny sklepu



Rysunek 5: Karta produktu

3.4.3 Zabezpieczenia na poziomie aplikacji

Zgodnie z przewidywaniami wdrażamy system logowania do systemu realizowany poprzez wcześniej ustalone role ROLE-CLIENT, ROLE-WORKER, ROLE-START(do obsługi niezalogowanych użytkowników- opcje logowania oraz tworzenie konta) Dodatkowo ROLE-ADMIN jako podstawowa

rola do zarządzania systemem(przypisywana tylko do kont administracyjnych).

System ciągle się rozwija, testy dopiero powstały- kolejnym krokiem będzie dobranie uprawnień koniecznych do wykonania wszystkich akcji przypisanych do roli. Poniżej obecny wstęp do tworzenia kont użytkowników: Utworzenie w bazie Ról i nadanie im odpowiednich uprawnień.

```
--Test systemu rol----------------------
use sklepinternetowy
create role role_looker;
grant show databases on *.* to role_looker;
create user 'looker' identified by 'lookerpass';
grant role_looker to looker;
set default role role_looker for looker;
grant select on sklepinternetowy.* to 'role_looker';
mysql -u looker -plookerpass sklepinternetowy;

INSERT INTO Address (City, Street, BuildingNumber,
ApartmentNumber, PostCode)
VALUES -> ('ImoWroc aw', 'aleja Czewona', 11,15, 13231);
--denied - wszystko dziala okey
-------------------------KONIEC TESTU

CREATE ROLE ROLE_START, ROLE_CLIENT, ROLE_WORKER, ROLE_ADMIN, ROLE_TEST;
--Rola test zostala utworzona do testow,
--aby nie niszczyc gotowych koneceptow-
-- po przejsciu tej fazy, akceptujemy role i wdrazamy jako glowna

CREATE USER 'ANDRZEJ_ADM' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'PATRYK_ADM' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'FILIP_ADM' IDENTIFIED BY 'CZARNAOWCA';

CREATE USER 'ANDRZEJ_CLI' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'PATRYK_CLI' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'FILIP_CLI' IDENTIFIED BY 'CZARNAOWCA';

CREATE USER 'ANDRZEJ_WOR' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'PATRYK_WOR' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'FILIP_WOR' IDENTIFIED BY 'CZARNAOWCA';

CREATE USER 'ANDRZEJ_STA' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'PATRYK_STA' IDENTIFIED BY 'CZARNAOWCA';
CREATE USER 'FILIP_STA' IDENTIFIED BY 'CZARNAOWCA';

GRANT SHOW DATABASES ON *.* TO ROLE_ADMIN;
GRANT ALL ON QCOM.* TO ROLE_ADMIN;
GRANT ROLE_ADMIN TO ANDRZEJ_ADM, PATRYK_ADM, FILIP_ADM;
```

```
SET DEFAULT ROLE ROLE_ADMIN FOR ANDRZEJ_ADM, PATRYK_ADM, FILIP_ADM;
--skonfigurowano role administracyjne i przylaczono je do kont

GRANT ROLE_CLIENT TO ANDRZEJ_CLI, PATRYK_CLI, FILIP_CLI;
SET DEFAULT ROLE FOR ANDRZEJ_CLI, PATRYK_CLI, FILIP_CLI;

GRANT EXECUTE ON PROCEDURE QCOM.bascet_to_order TO ROLE_CLIENT;
GRANT SELECT ON QCOM.PRODUCT, QCOM.CATEGORY, QCOM.DISCOUNT;
--wstepny zarys uprawnien klienta

GRANT ROLE_WORKER TO ANDRZEJ_WOR, PATRYK_WOR, FILIP_WOR;
SET DEFAULT ROLE FOR ANDRZEJ_WOR, PATRYK_WOR, FILIP_WOR;
GRANT EXECUTE ON PROCEDURE QCOM.make_discount TO ROLE_WORKER;
GRANT UPDATE ON QCOM.PRODUCT, QCOM.PRODUCTINSTORE,
QCOM.DISCOUNT TO ROLE_WORKER;
GRANT SELECT ON QCOM.* TO ROLE_WORKER;
--wstepny zarys uprawnien pracownika- bedzie zdecydowanie inaczej wygladac
--po kolejnych konsultacjach z analitykiem grupy

--rola typu START jest jeszcze niezdefiniowana nawet wstepnie-
--potrzeban analiza czego dok adnie potrzebuje
--jak b dziemy realizowa  te mechanizmy (funkcje, procedury,
--jak wyslac zapytanie o utworzenie uzytkownika,
-- co jest potrzebne do logowania, opcja przegladania, potrzebny
--dodatkowy research, konsultacja z reszta grupy
```

Utworzyliśmy użytkowników testowych.

Przykładowy scenariusz- test:

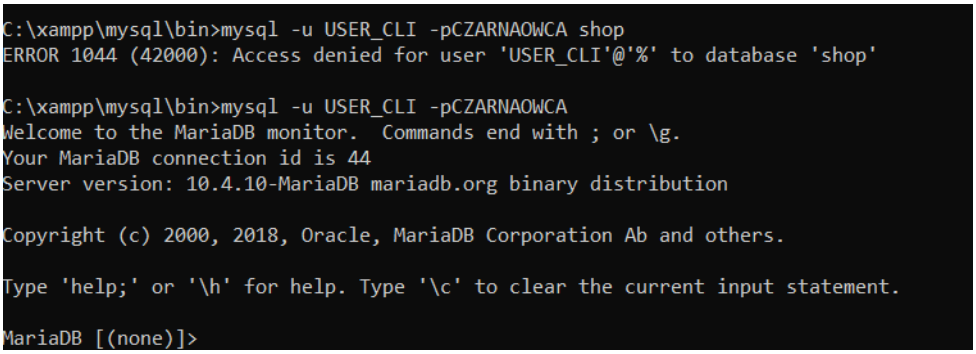
- Utwórz użytkownika Klient(USER_CLI)

  ```
  CREATE USER 'USER_CLI' IDENTIFIED BY 'CZARNAOWCA';
  ```

- Stwórz rolę do kont typu Klient

  ```
  CREATE ROLE ROLE_CLIENT;
  ```

- Domyślnie konto nie posiada żadnych uprawnień -spróbujmy sprawdzić czy możemy cokolwiek zobaczyć w systemie -próba zalogowania



Rysunek 6: Logowanie

- Nie mamy nawet dostępu do samej bazy sklepu
- Logujemy się z powrotem na konto root i przypisujemy rolę do konta

```
GRANT ROLE_CLIENT TO USER_CLI;
SET DEFAULT ROLE ROLE_CLIENT FOR USER_CLI;
```

- Nadajmy uprawnienia do przeglądania dostępnych produktów

```
GRANT SELECT ON shop.products TO ROLE_CLIENT;
FLUSH PRIVILEGES;
```

- Powinniśmy być w stanie zobaczyć zawartość products -i tylko jej, sprawdźmy

```
MariaDB [(none)]> exit
Bye

C:\xampp\mysql\bin>mysql -u USER_CLI -pCZARNAOWCA shop
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 83
Server version: 10.4.10-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [shop]> SELECT current_role();
+----------------+
| current_role() |
+----------------+
| ROLE_CLIENT    |
+----------------+
1 row in set (0.000 sec)

MariaDB [shop]> show tables;
+----------------+
| Tables_in_shop |
+----------------+
| products       |
+----------------+
1 row in set (0.001 sec)

MariaDB [shop]> SELECT * FROM products;
+----+---------+------------------------------+-------------+------------+------------+-------------+-------------+----
-------------------------------------------------------+--------------------------+---------------------------------+--
| Id | Price   | Name                         | Description | CategoryId | DiscountId | FieldValue1 | FieldValue2 | Fie
FieldValue6                                            | FieldValue7              | FieldValue8                     | A
+----+---------+------------------------------+-------------+------------+------------+-------------+-------------+----
-------------------------------------------------------+--------------------------+---------------------------------+--
| 17 |  199.00 | Death Stranding              | NULL        |          2 |       NULL |        2019 |          16 |
akcji, przygodowe                                      | polska i angielska       | Sony Interactive Entertainment  |
| 18 |  199.00 | STAR WARS JEDI: Upadły Zakon | NULL        |          2 |       NULL |        2019 |          16 |
akcji, przygodowe                                      | polska i angielska       | Electronic Arts                 |
| 19 |  179.00 | Gran Turismo Sport           | NULL        |          2 |       NULL |        2018 |           3 |
wyścigi                                                | polska i angielska       | Sony Computer Entertainment     |
| 20 | 4299.00 | TUF Gaming FX505DU           | NULL        |          1 |       NULL |          16 |          16 |
AMD Ryzen 7 3750H                                      | NVIDIA GeForce GTX 1660Ti | Brak                           |
| 21 | 3799.00 | Inspiron G3                  | NULL        |          1 |       NULL |           8 |          16 |
```

Rysunek 7: Akcje do których mamy uprawnienia

```
MariaDB [shop]> select * from customers;
ERROR 1142 (42000): SELECT command denied to user 'USER_CLI'@'localhost' for table 'customers'
MariaDB [shop]> INSERT INTO Products (Price, Name, CategoryId, FieldValue1, FieldValue2, FieldValue3,
    -> (199, 'Death Stranding', 2, 2019, 16, NULL, NULL, 'PlayStation4', 'akcji, przygodowe', 'polska
ERROR 1142 (42000): INSERT command denied to user 'USER_CLI'@'localhost' for table 'products'
MariaDB [shop]> UPDATE products SET Price=99999 WHERE Id='30';
ERROR 1142 (42000): UPDATE command denied to user 'USER_CLI'@'localhost' for table 'products'
MariaDB [shop]>
```

Rysunek 8: Akcje do których nie mamy dostępu

- Mechanizm kont i ról sprawnie radzi sobie z zabezpieczeniami

4 Implementacja systemu baz danych

Do zaimplementowania wcześniej zaprojektowanej przez nas bazy danych posłużyliśmy się otwartoźródłowym systemem zarządzania relacyjnymi bazami danych MySQL. Kod SQL został wygenerowany przez program Visual Paradigm na podstawie wcześniej wykonanego fizycznego diagramu bazy danych. Po poprawkach w wygenerowanym kodzie jak na przykład założeniem indeksów na dwie kolumny na raz, czego nie oferował Visual Paradigm mogliśmy podstawić naszą bazę danych. Do zarządzania naszą bazą wykorzystaliśmy darmowy program phpMyAdmin.

4.1 Implementacja wybranych tabel bazy danych

Tabela *Customer*

Przechowuje informacje na temat zarejestrowanego w naszym sklepie klienta. Dane są podawane przez nowego klienta w trakcie procesu rejestracji. Tabela przechowuje też informacje na temat aktualnego koszyka danego klienta.

```
CREATE TABLE Customers (
        Id int(10) NOT NULL AUTO_INCREMENT,
        FirstName varchar(30) NOT NULL,
        LastName varchar(30) NOT NULL,
        Email varchar(60) NOT NULL,
        BirthDate date NOT NULL,
        PhoneNumber varchar(20) NOT NULL,
        BasketValue decimal(8, 2) DEFAULT 0 NOT NULL,
        AddressId int(10) NOT NULL,
        Password varchar(60) NOT NULL,
        Login varchar(60) NOT NULL,
        PRIMARY KEY (Id),
        INDEX (LastName, FirstName),
        UNIQUE INDEX (Login)
        );

ALTER TABLE Customers
        ADD CONSTRAINT 'Customer has address'
        FOREIGN KEY (AddressId)
        REFERENCES Addresses (Id);
```

Tabela *Product*

Przechowuje informacje o abstrakcyjnej klasie produktu. Opisuje jego cechy oraz przechowuje takie parametry jak jego cena. Na podstawie tej tabeli generwoany jest widok produktu w sklepie.

```
CREATE TABLE Products (
        Id int(10) NOT NULL AUTO_INCREMENT,
        Price decimal(8, 2) NOT NULL,
        Name varchar(60) NOT NULL,
        Description varchar(10000),
        CategoryId int(10) NOT NULL,
        DiscountId int(10),
```

```
    FieldValue1 int(4),
    FieldValue2 int(4),
    FieldValue3 int(4),
    FieldValue4 int(4),
    FieldValue5 varchar(100),
    FieldValue6 varchar(100),
    FieldValue7 varchar(100),
    FieldValue8 varchar(100),
    AmountInStore int(10) DEFAULT 0 NOT NULL,
    ProducerName varchar(100),
    PRIMARY KEY (Id),
    INDEX (Price),
    INDEX (Name),
    INDEX (CategoryId)
    );

ALTER TABLE Products
    ADD CONSTRAINT 'Product belongs to a category'
    FOREIGN KEY (CategoryId)
    REFERENCES Categories (Id);
```

Tabela *Order*

Przechowuje informacje zarówno o zrealizowanych zamówieniach jak i będących w trakcie realizacji, przyporządkowanych do danego klienta.

```
CREATE TABLE Orders (
    Id int(10) NOT NULL AUTO_INCREMENT,
    SaleDate date NOT NULL,
    Value int(1) DEFAULT 0 NOT NULL,
    Status varchar(30) NOT NULL,
    CustomerId int(10) NOT NULL,
    InvoiceId int(10),
    AddressId int(10),
    PRIMARY KEY (Id),
    INDEX (SaleDate),
    INDEX (Status),
    INDEX (CustomerId)
    );

ALTER TABLE Orders
    ADD CONSTRAINT 'Order may have address where to send'
    FOREIGN KEY (AddressId)
    REFERENCES Addresses (Id);

ALTER TABLE Orders
    ADD CONSTRAINT 'Order may have invoice'
    FOREIGN KEY (InvoiceId)
```

```
REFERENCES Invoices (Id);
```

4.2 Implementacja sekwencji w MySQL

System zarządzania bazami danych MySQL nie oferuje wprost mechanizmu wprowadzania własnych sekwencji. W naszej bazie danych rolę sekwencji pełni automatyczna inkrementacja kluczy głównych w każdej tabeli zrealizowane komendą 'AUTO_INCREMENT'.

4.3 Implementacja wybranych zapytań SQL

Widoki:

- top_products

```
CREATE VIEW top_products AS
SELECT Product.Id, Product.Name, Product.ProducerName,
 Product.AmountInStore as In_store,
SUM(Order_Product.AmountInOrder) as Sold
FROM Product
JOIN Order_Product ON Product.Id = Order_Product.ProductId
GROUP BY Order_Product.ProductId
ORDER BY SUM(Order_Product.AmountInOrder) desc;
```

- employee_discount

```
CREATE VIEW employee_discount AS
SELECT Employee.Id, Employee.LastName, Discount.Name
FROM Employee
JOIN Discount ON Employee.Id = Discount.EmployeeId
ORDER BY Employee.LastName;
```

Procedury składowe:

- basket_to_order

```
delimiter //
CREATE PROCEDURE basket_to_order
(IN customerID int(10), IN orderID int(10))
BEGIN
    INSERT INTO Order_Product(OrderId, ProductId,AmountInOrder)
    SELECT orderID, Basket_Product.ProductId,
     Basket_Product.AmountInBasket FROM Basket_Product
    WHERE
     Basket_Product.CustomerId = customerID;
    DELETE FROM Basket_Product WHERE Basket_Product.CustomerId = customerID;
END;//
delimiter ;
```

Wyzwalacze:

- add_to_basket

```
delimiter //
CREATE TRIGGER add_to_basket AFTER INSERT ON Basket_Product
FOR EACH ROW BEGIN
    UPDATE Customer SET Customer.BasketValue = Customer.BasketValue +
    NEW.AmountInBasket*(SELECT Product.Price FROM Product
    WHERE Product.Id = NEW.ProductId)
    WHERE Customer.Id = NEW.CustomerId;
END;//
delimiter ;
```

- update_basket

```
delimiter //
CREATE TRIGGER update_basket AFTER UPDATE ON Basket_Product
FOR EACH ROW BEGIN
    UPDATE Customer SET Customer.BasketValue = Customer.BasketValue +
    (NEW.AmountInBasket-OLD.AmountInBasket)*(SELECT Product.Price
    FROM Product WHERE
    Product.Id = OLD.ProductId)
    WHERE Customer.Id = OLD.CustomerId;
END;//
delimiter ;
```

4.4 Testowanie bazy danych na przykładowych danych

4.4.1 Czas dostępu do danych

Średni czas dostępu do danych został wyznaczony jako średnia arytmetyczna z dziesięciu wywołań zapytania SELECT. Na zapytanie składało się wyszukanie pięciu rekordów w bazie danych. Tabele były przeszukiwane jak i sortowane według tych samych kolumn. Dla każdej było to następująco:

- Klienci - Nazwisko i imię
- Adresy - Kod pocztowy
- Zamówienia - wartość zamówienia

Ilość rekordów	Średni czas[s]	Założono indeks
1 000	0.0033	Nie
10 000	0.0072	Nie
100 000	0.0258	Nie
1 000	0.0023	Tak
10 000	0.0034	Tak
100 000	0.0126	Tak

Tabela 1: Wyszukiwanie klientów

Ilość rekordów	Średni czas[s]	Założono indeks
1 000	0.0037	Nie
10 000	0.0124	Nie
100 000	0.0816	Nie
1 000	0.0018	Tak
10 000	0.0021	Tak
100 000	0.0032	Tak

Tabela 2: Wyszukiwanie adresów

Ilość rekordów	Średni czas[s]	Założono indeks
1 000	0.0013	Nie
10 000	0.0015	Nie
100 000	0.0020	Nie
1 000	0.0013	Tak
10 000	0.0014	Tak
100 000	0.0021	Tak

Tabela 3: Wyszukiwanie zamówień

4.4.2 Czas sortowania danych

Z określonych tabel zostało wybranych po sto rekordów posortowanych według podanej kolumny. Średni czas sortowania został wyznaczony jako średnia arytmetyczna z dziesięciu powtórzeń danego sortowania.

Ilość rekordów	Średni czas[s]	Założono indeks
1 000	0.0025	Nie
10 000	0.0126	Nie
100 000	0.0818	Nie
1 000	0.0025	Tak
10 000	0.0127	Tak
100 000	0.0149	Tak

Tabela 4: Sortowanie klientów

Ilość rekordów	Średni czas [s]	Założono indeks
1 000	0.0028	Nie
10 000	0.0137	Nie
100 000	0.1013	Nie
1 000	0.0026	Tak
10 000	0.0136	Tak
100 000	0.0188	Tak

Tabela 5: Sortowanie adresów

Ilość rekordów	Średni czas[s]	Założono indeks
1 000	0.0014	Nie
10 000	0.0071	Nie
100 000	0.0513	Nie
1 000	0.0012	Tak
10 000	0.0066	Tak
100 000	0.0106	Tak

Tabela 6: Sortowanie zamówień

4.4.3 Czas wykonania złożonych zapytań do bazy danych

W bazie danych mamy zaimplementowane złożone zapytania SQL, których wpisywanie za każdym razem było by bardzo uciążliwe dla użytkownika. Dlatego zostały one zapisane w bazie danych jako widoki oraz procedury składowe znacznie usprawniając korzystanie z zawartości bazy danych.

Identyfikator zapytania	Średni czas wykonania [s]
top_products	0.0033
orders_in_progres	0.0025
returned_products	0.0027
basket_to_order	0.0029

Tabela 7: Czas wykonania złożonych zapytań

4.4.4 Podsumowanie wyników

Warto zauważyć że wszystkie powyższe testy odbywały się z wyłączonym zapamiętywaniem wielokrotnie powtarzanych zapytań do cache'a za pomocą komendy SQL_NO_CACHE. Po włączeniu tej opcji zmienia nam się czas dostępu do danych, na które jest założony indeks. Na takich zapytaniach średnio spadał on do około 0.0020 sekundy.

Podsumowując nasza baza danych nawet przy obłożeniu znaczną ilością danych cały czas zapewnia użytkownikowi swobodny dostęp bez jakiegokolwiek znaczącego oczekiwania. Założenie indeksów na dane, które będą wielokrotnie przeglądane przez użytkownika znacząco zmniejszają ten czas, dzięki czemu nawet przy liczności danych powyżej 100 000 w bazie czas dostępu do nich nie będzie odczuwalny dla użytkownika. W wykorzystaniu praktycznym nasza baza będzie też korzystała z cache'u co jeszcze bardziej obniży czas jej przeszukiwania.

5 Implementacja aplikacji

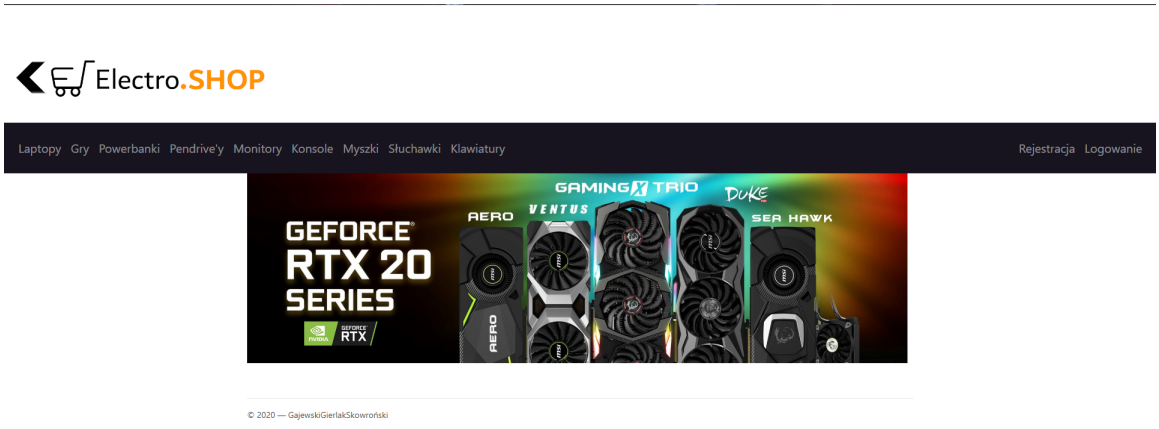
5.1 Instrukcja dla użytkownika

Aplikacja oferuje funkcjonalności dla dwóch typów użytkowników: anonimowego oraz zarejestrowanego i zalogowanego. Po założeniu konta zalogowany użytkownik zyskuje względem anonimowego możliwość: dodawania produktów do koszyka, przeglądanie i modyfikowanie zawartości koszyka, podgląd oraz modyfikowanie swoich danych, złożenie zamówienia na aktualną zawartość koszyka.

Oba rodzaje użytkowników mają dostęp do przeglądania oferty produktów sklepu. Klient anonimowy nie posiada dostępu do żadnych funkcjonalności do jakich zalogowany użytkownika nie ma dostępu, za wyjątkiem możliwości zarejestrowania się.

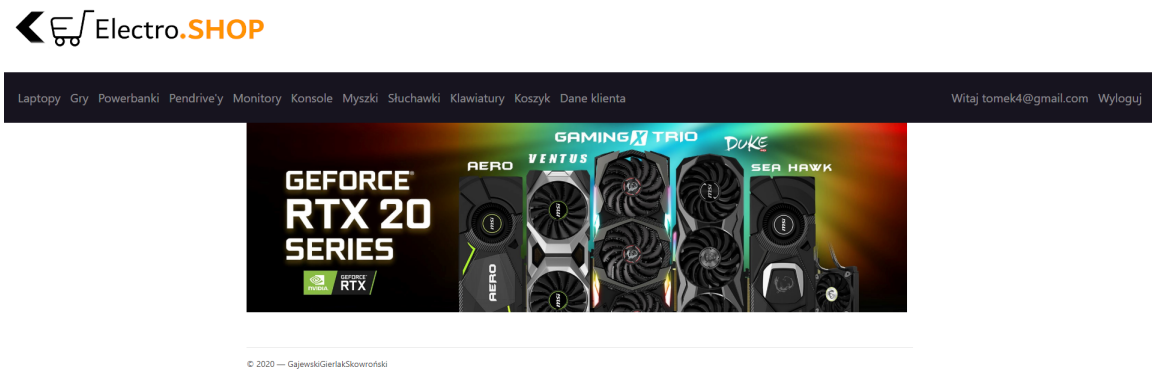
Główna strona aplikacji prezentuje się następująco:

Dla anonimowego użytkownika:



Rysunek 9: Menu główne

Dla zalogowanego użytkownika:

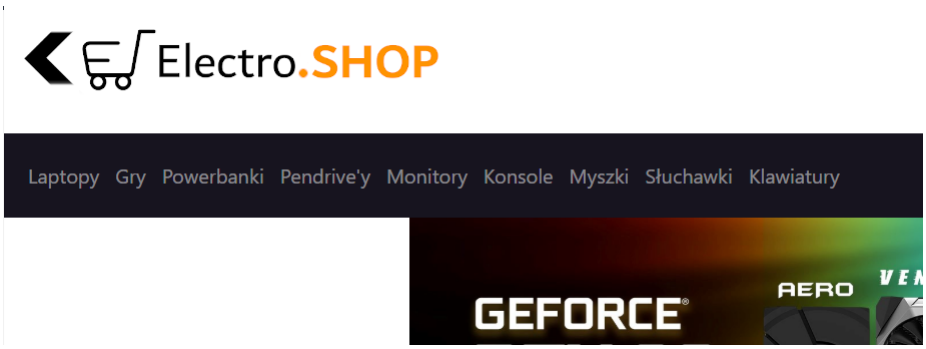


Rysunek 10: Menu główne

W prawym górnym rogu widać że dla niezalogowanego użytkownika mamy możliwość rejestracji lub zalogowania się na wcześniej utworzone konto. Natomiast zalogowany użytkownik może się w tym miejscu wylogować albo klikając na powitanie przejść do zakładki z modyfikowaniem danych.

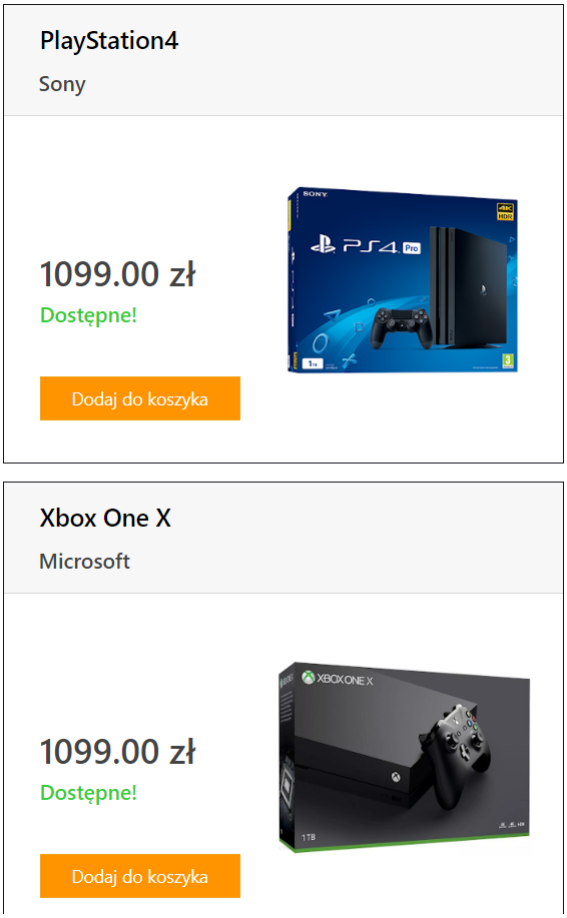
5.1.1 Przegląd dostępnych funkcjonalności dla użytkownika

1. Przeglądanie oferty sklepu
   Po przejściu na stronę sklepu użytkownik może wybrać interesujące go produkty według dostępnych kategorii.



Rysunek 11: Dostępne w sklepie kategorie

Po kliknięciu myszką w interesującą klienta kategorię zostanie on przeniesiony na podstronę, na której znajdują się wszystkie produkty z danej kategorii.

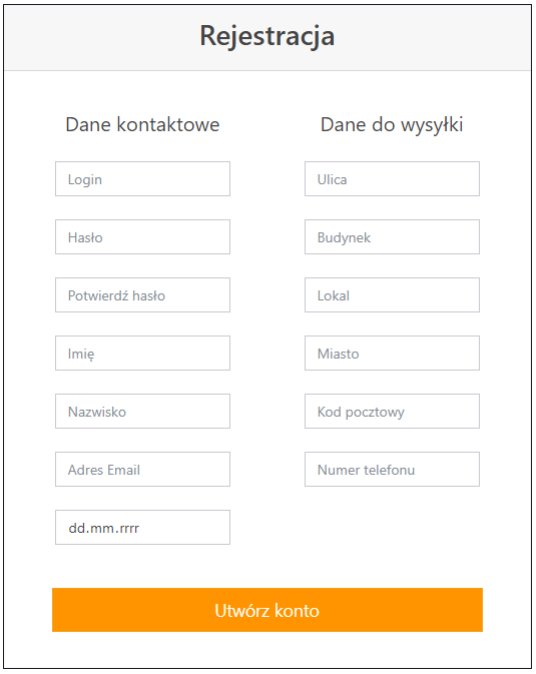


Rysunek 12: Wylistowane produkty

Po kliknięciu w wybrany przez nas produkt otwiera się jego karta katalogowa z informacjami na jego temat.

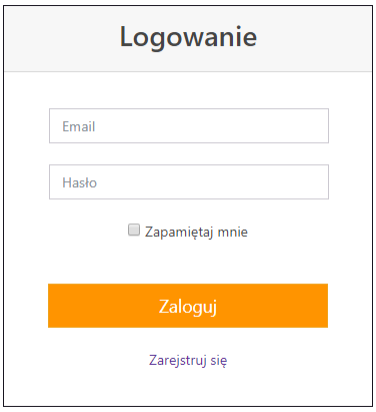
Rysunek 13: Informacje o produkcie

2. Rejestracja
   Po wybraniu opcji rejestracji z prawego górnego rogu użytkownikowi wyświetli się formularz do wypełnienia swoimi danymi. Po kliknięciu przycisku "Utwórz konto" jego dane zostaną zapisanie w bazie danych a konto założone w sklepie do którego od tej pory będzie się mógł logować. Po utworzeniu konta użytkownik zostaje do niego automatycznie zalogowany.



Rysunek 14: Formularz rejestracyjny

3. Logowanie
   Uprzednio zarejestrowany użytkownik może zalogować się na swoje konto wypełniając następujący formularz, po kliknięciu przycisku logowanie:

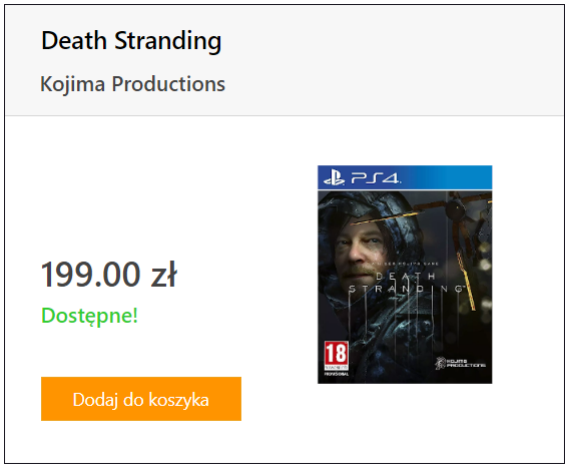


Rysunek 15: Formularz logowanie

Do logowanie użytkownik może użyć swojego adresu email albo loginu podanego w trakcie rejestracji. Użytkownika ma również możliwość zapamiętania po odhaczeniu "Zapamiętaj mnie", przy następnych odwiedzinach sklepu nie będzie się musiał powtórnie logować. Jeżeli

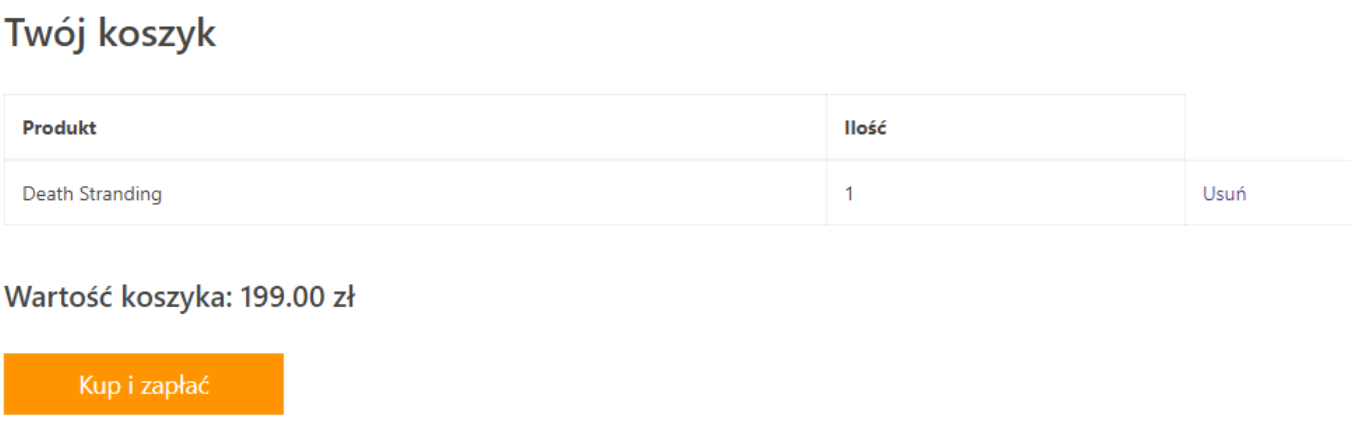
użytkownik nie ma jeszcze konta to może się przenieść do formularzu rejestracji klikając w link "Zarejestruj się".

4. Dodanie produktu do koszyka, edycja jego zawartości oraz złożenie zamówienia na jego zawartość
   Każdy produkt ma w swojej karcie przycisk "Dodaj do koszyka", który po kliknięciu dodaje jego jedną sztukę do koszyka użytkownika.



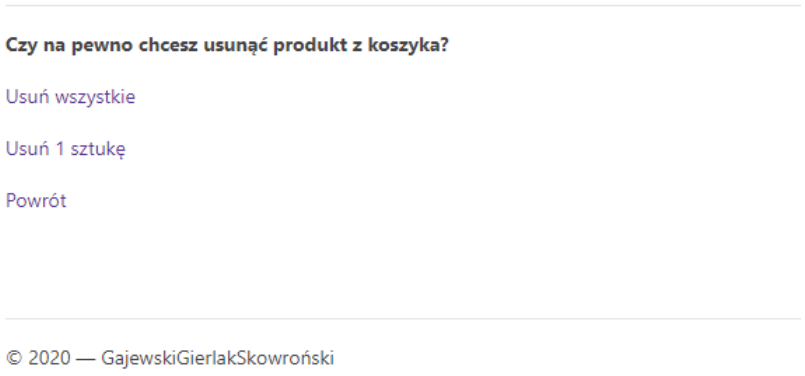
Rysunek 16: Dodawanie do Koszyka

Zalogowany użytkownik może podejrzeć aktualny stan swojego koszyka klikając na pasku menu opcje "Koszyk".



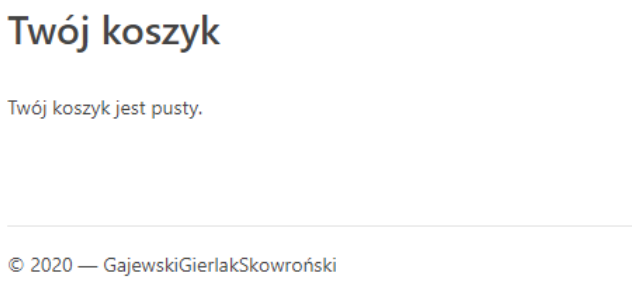
Rysunek 17: Zawartość koszyka

Po wybraniu opcji usuń obok, któregoś z produktów w koszyku użytkownik może albo usunąć jedną jego sztukę z koszyka albo całą instancję danego produktu.



Rysunek 18: Usuwanie z koszyka

Jeżeli użytkownik w widoku swojego koszyka wybierze opcje Kup i zapłać jego koszyk zostanie opróżniony co ma symulować złożenie przez niego zamówienia. W tym momencie powinien być przenoszony na stronę obsługującą płatność za zamówienie lecz w obecnej wersji zostaje na stronie informującej go że jego koszyk jest pusty po opróżnieniu.

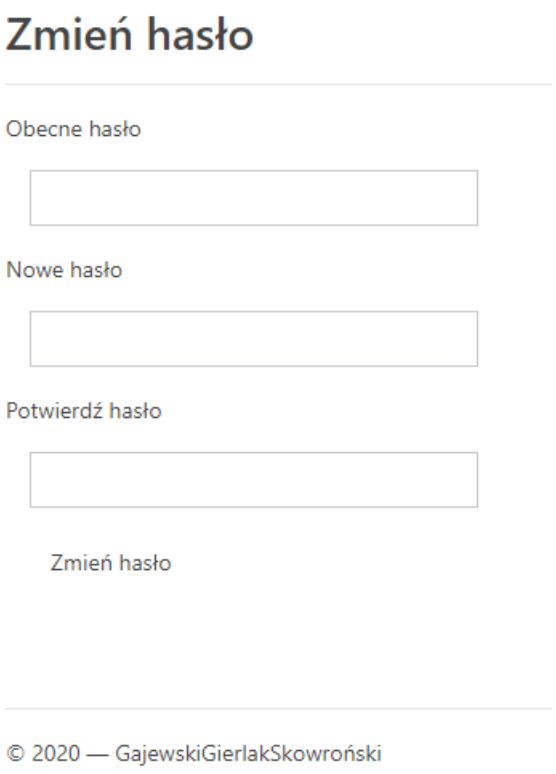


Rysunek 19: Widok pustego koszyka

5. Edycja danych użytkownika
   Po kliknięciu przez zalogowanego użytkownika wiadomości w prawej części paska menu, witającej go, z jego adresem email, zostanie on przekierowany do zakładki zarządzania swoim kontem, na którym może wybrać czy chce zmienić hasło do swojego konta czy edytować swoje dane użytkownika.

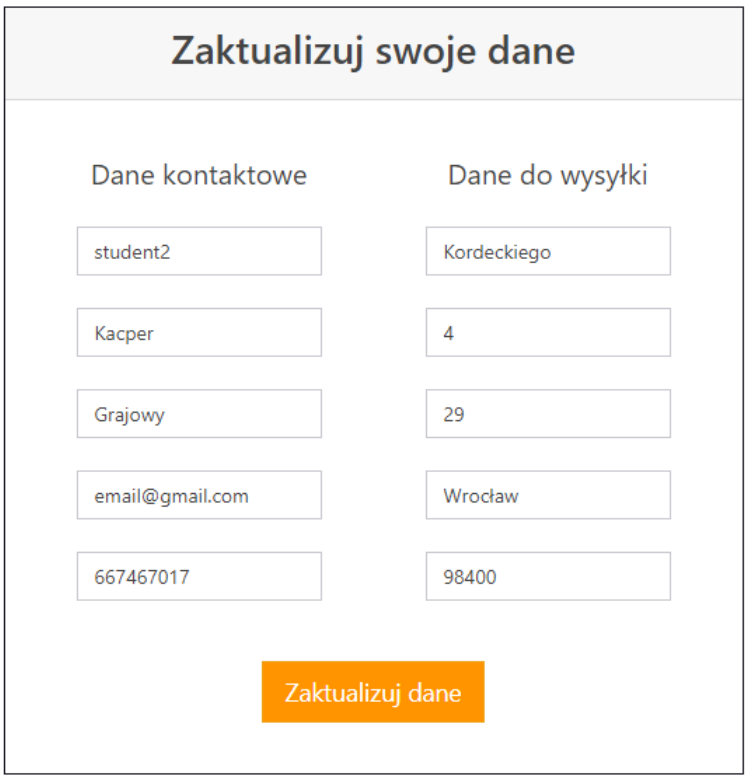
Rysunek 20: Zarządzanie kontem

Po wyborze opcji zmiany hasła będzie on proszony o podanie starego hasła oraz nowego razem z jego potwierdzeniem.



Rysunek 21: Zmiana hasła

Po wyborze opcji zmiany danych użytkownika zostanie wyświetlony formularz wypełniony jego aktualnymi danymi. Użytkownik może je zamienić na nowe po czym klikając przycisk "Zaktualizuj dane" nadpiszą one jego aktualne dane w bazie danych.



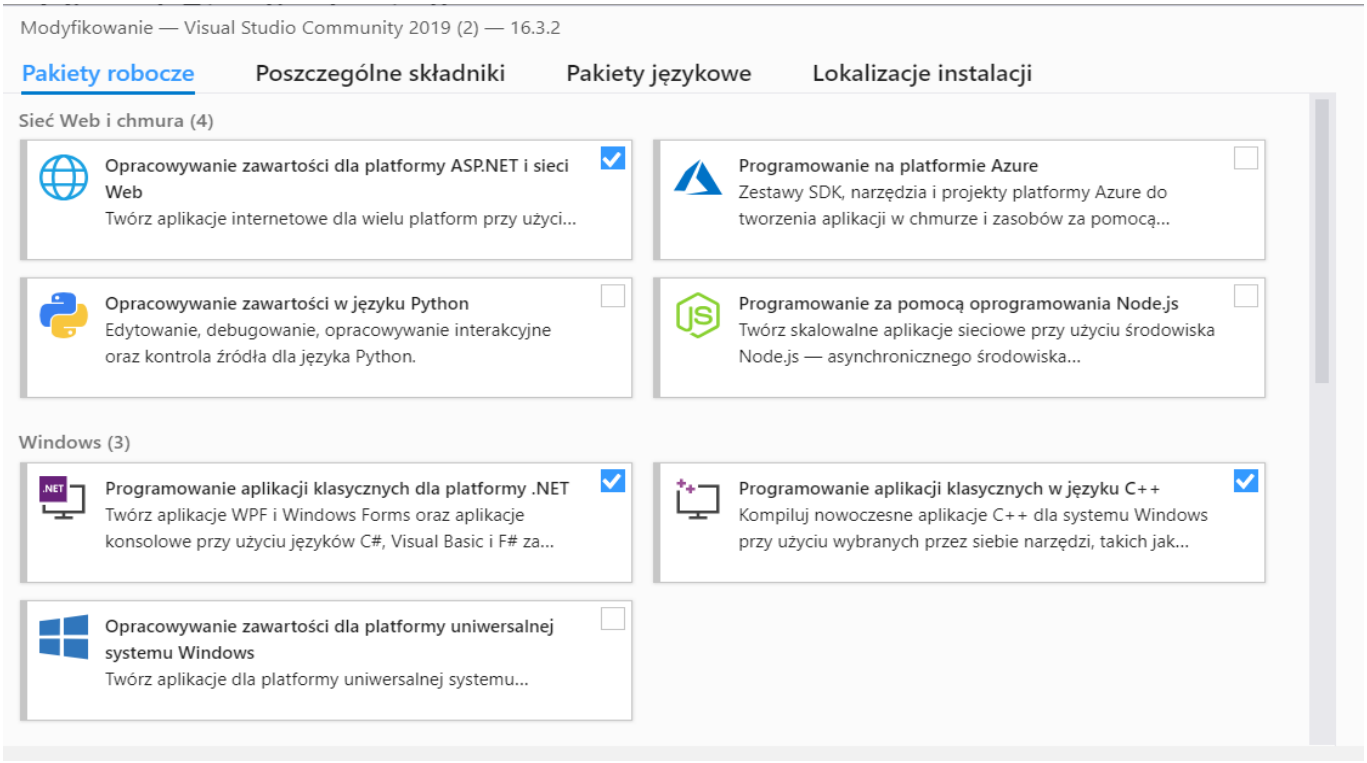
Rysunek 22: Aktualizowanie danych użytkownika

5.2 Instrukcja dla developera

Fragment dotyczy tego, jak zainstalować potrzebne oprogramowanie i dalej współtworzyć aplikację.

```
https://visualstudio.microsoft.com/pl/
```

- Pobierz i zainstaluj Visual Studio Community 2019 z oficjalnej strony producenta(najlepiej przez visual studio installer, dzięki czemu zawsze będziesz mógł doinstalować ewentualnie brakujące elementy).
- Upewnij się, że masz zaznaczone wymagane elementy.

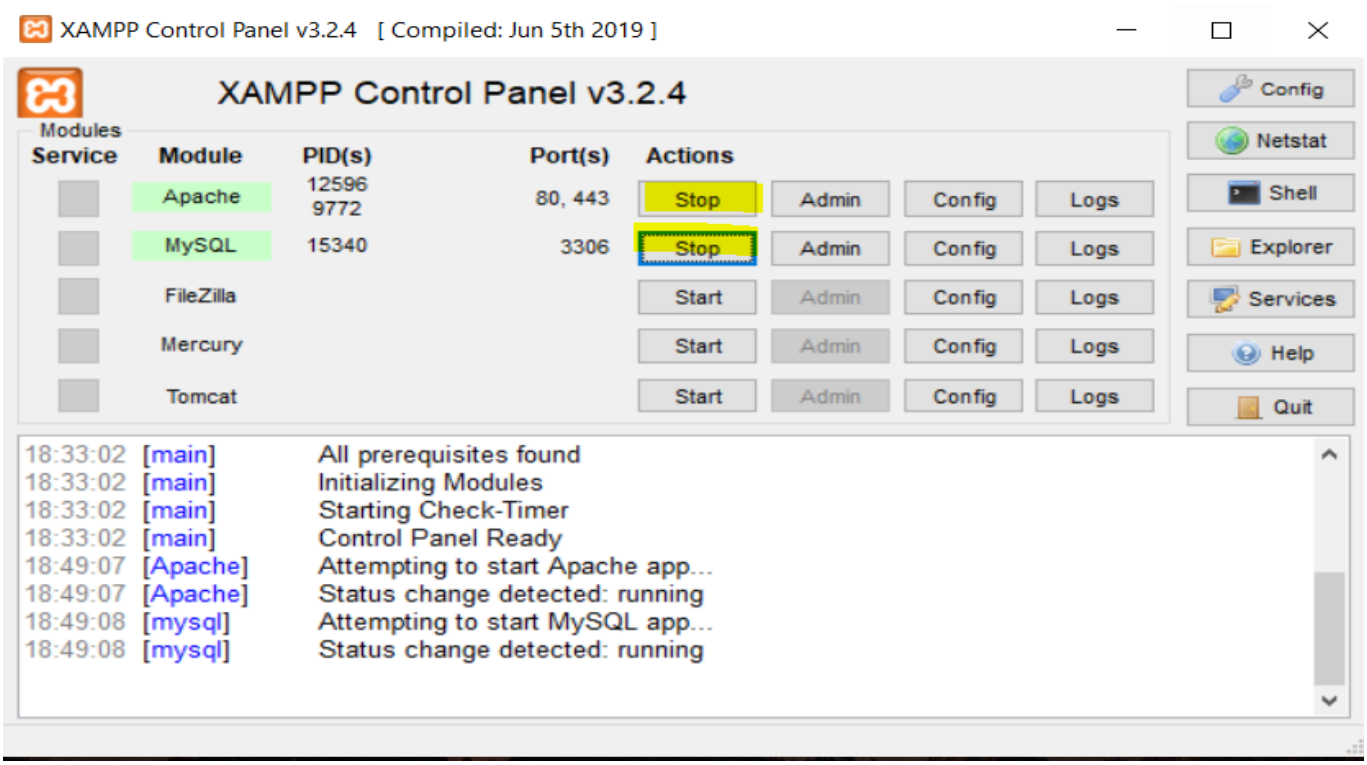


Rysunek 23: Instalator Visual Studio

- Zainstaluj lokalny serwer w celu testowania aplikacji.
  - Pobierz Xampp Apache ze strony:

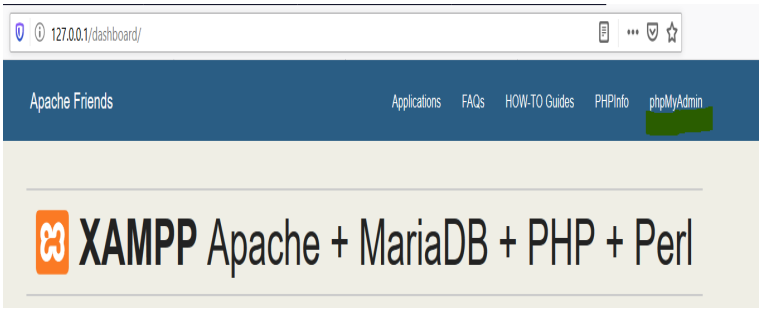
    `https://www.apachefriends.org/pl/index.html`

  - Przeprowadź pełną instalację(wszystkie ustawienia pozostaw domyślnie).
  - Uruchom panel kontrolny Xampp'a i kliknij start przy Apache i Mysql.



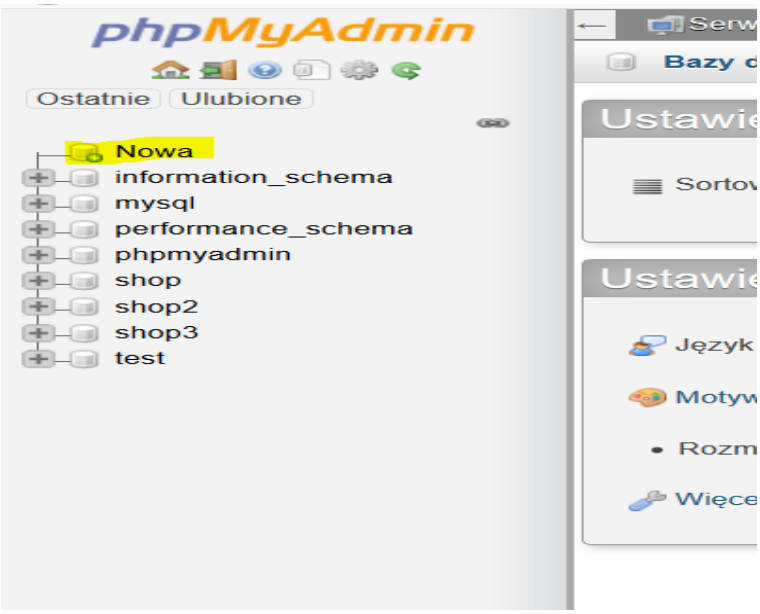
Rysunek 24: Panel kontrolny Xampp'a

– Otwórz przeglądarkę i wpisz adres 127.0.0.1, przejdź na phpMyAdmin.



Rysunek 25: phpMyAdmin

– Utwórz nową bazę danych (wpisz nazwę: shop2 , kliknij utwórz).



Rysunek 26: Nowa baza

- Idź do:

  `https://github.com/pskowronski97z/Sklep-internetowy-BD2`

- Pobierz folder APP2. Znajduje się tam kod aplikacji, jak również pliki do testowej bazy danych.
- Uruchom solucję ShopLogin.
- W pliku Web.config upewnij się, że masz odpowiednią nazwę bazy danych:

  `</connectionString> ... database=shop2; ...`

- Otwórz phpMyAdmin i zaimportuj bazę shop2 z folderu SQL, do bazy którą wcześniej utworzyłeś (zaznacz nowo utworzoną bazę, a następnie kliknij import i wybierz plik).

- Aby testować Klienta, utwórz nowe konto (zarejestruj się w aplikacji, jako nowy użytkownik).
- Aby testować Administratora, zaloguj się za pomocą:

  ```
  login Admin@admin.pl
  hasło: Pa$$w0rd
  ```

- W Package Manager Console (lewy dolny róg) wpisz: update-database
- Wciśnij F5, aplikacja powinna wystartować, do prawidłowego działania wymagane jest uruchomienie przeglądarki na pełnym ekranie (przystosowanie do skalowalności interfejsu planowane jest dopiero w kolejnych etapach rozwoju aplikacji).

- Wczytywanie danych najłatwiej realizować za pomocą phpMyAdmin, gdzie trzeba wprowadzić bezpośrednio SQL z danymi(folder SQL(git) zawiera przykładowe wypełnienia).

6 Wybrane fragmenty kodu aplikacji

6.1 Technologia

Projekt realizowaliśmy w technologii C# ASP.NET z modelem MVC. To w głównej mierze wpłynęło na całokształt pracy. Jest to technologia tworzona specjalnie pod budowę aplikacji internetowych. Samo wykorzystanie silnika ASP.NET udostępniło funkcje takie, jak:

- Mechanizmy zarządzania stanem aplikacji.
- Mechanizmy uwierzytelniania i autoryzacji (cały wcześniejszy system ról przełożyliśmy na implementację współgrającą z silnikiem).
- Mechanizm rejestracji- wbudowany wystarczyło zmodyfikować, by odpowiadał naszym wymaganiom.
- Obsługa plików cache, dzięki czemu użytkownik nie musi logować się za każdym razem, kiedy wygaśnie mu sesja.

6.2 MVC - Model, View, Controller

Model w naszym projekcie odnosił się do reprezentacji danych i odpowiedniego podziału logiki w aplikacji, to on odpowiadał za przekazywania danych pomiędzy pozostałymi elementami. Controller'y były u nas najbardziej złożonymi fragmentami kodu, gdyż miały obsługiwać wszystkie akcje, które toczyły się "pod spodem". View- widok, był tym co konkretnie było wyświetlane użytkownikowi, korzystającemu z aplikacji- zastosowaliśmy pewien rodzaj widoków dynamicznych, które w zależności od stanu, uprawnień zalogowanego użytkownika mogą wyświetlać i ukrywać odpowiednie elementy, dzięki temu oszczędziliśmy na dorabianiu kolejnych podstron, które różniłyby się od siebie tylko pod paroma względami (również nie przesadziliśmy z tym mechanizmem, dzięki czemu wszystko jest podzielone w sposób czytelny i logiczny).

6.3 Przykładowy model

Modele służą do reprezentacji tabel bazodanowych w kodzie aplikacji. Poniżej przykładowy model reprezentujący tabelę przechowujący dane o produkcie w naszej aplikacji.

```
public class Product
    {
        public int Id { get; set; }

        [Required]
        [Index]
        [Range(0, 99999.99)]
        public decimal Price { get; set; }
```

```
        [Required]
        [Index]
        public String Name { get; set; }

        public String Description { get; set; }

        [Required]
        [Index]
        public int CategoryId { get; set; }
        public Category Category { get; set; }

        public float? FieldValue1 { get; set; }

        public float? FieldValue2 { get; set; }

        public float? FieldValue3 { get; set; }

        public float? FieldValue4 { get; set; }

        public String FieldValue5 { get; set; }

        public String FieldValue6 { get; set; }

        public String FieldValue7 { get; set; }

        public String FieldValue8 { get; set; }

        public String PictureId { get; set; }

        [Required]
        public int AmountInStore { get; set; }

        public String ProducerName { get; set; }

        public ICollection<Basket> Baskets { get; set; }
    }
```

W powyższym kodzie wszystkie kolumny tabeli programujemy jako atrybuty klasy. Nad odpowiednimi atrybutami w nawiasach kwadratowych opisujemy wszystkie wymagania odnośnie danej kolumny za pomocą adnotacji.

6.3.1 Przykładowe funkcje w kontrolerach

Kontrolery to klasy zawierające funkcje odpowiedzialne za logikę aplikacji dla poszczególnych modeli.

Poniższa funkcja służy do wyświetlania produktów dla zadanej mu kategorii. Na podstawie

otrzymania Id konkretnej kategorii pobiera ona właściwe produkty z bazy danych a następnie zapisuje je do listy. Tak przygotowaną listę zwraca do widoku odpowiedzialnego za jej wyświetlenie.

```
public ViewResult Index(int catId)
{
        var products = _context.Products.Include(p => p.Category).ToList();
        List<Product> filteredProducts = new List<Product>();


                foreach (var product in products)
                {
                        if (product.CategoryId == catId)
                        {
                                filteredProducts.Add(product);
                        }
                }

        return View(filteredProducts);
}
```

Funkcja SignInByLoginOrEmail służy do rozpoznania czy użytkownik chce się zalogować do aplikacji przy pomocy swojego loginu czy adresu email. W pierwszej kolejności jest sprawdzane czy w bazie istniej użytkownik o podanym loginie jeśli nie to jest wyszukiwany użytkownik któremu podana wartość pasuje do adresu email. Jeżeli któreś z wymienionych zostanie zlokalizowane to następuje próba logowania dla danej wartości oraz podanego hasła.

```
private async Task<SignInStatus> SignInByLoginOrEmail(LoginViewModel model)
{
        Customer customer = _context.Customers.FirstOrDefault(c =>
        c.Login == model.Email);
        if(customer != null)
        {
                ApplicationUser user = UserManager.FindById(customer.UserId);
                return await SignInManager.PasswordSignInAsync(user.Email,
                model.Password, model.RememberMe, shouldLockout: false);
        }

        return await SignInManager.PasswordSignInAsync(model.Email,
        model.Password, model.RememberMe, shouldLockout: false);
}
```

7 Podsumowanie i wnioski

W trakcie naszej pracy nad projektem odświeżyliśmy i wykorzystaliśmy całość wiedzy zdobytej przez nas w trakcie kursu Bazy Danych 1 z zeszłego semestru. Oprócz tego nabyliśmy nową wiedzę na temat wzorca architektonicznego Model-View-Controller według, którego zaprogramowaliśmy naszą aplikację webbową.

Do zaprogramowania naszej aplikacji posłużyliśmy się technologią ASP.NET oraz Entity Framework, których to uczyliśmy się na bieżąco w trakcie trwania zajęć projektowych. Wymienione technologię okazały się dla nas nad wyraz przyjazne w obsłudze oraz poznawanie ich nie stanowiło ogromnych trudności. Finalna wersja naszej aplikacji wraz z bazą danych stanowią dobry grunt do dalszego rozwijania i dodawania do nich nowych funkcjonalności.

8 Literatura

- https://stackoverflow.com/questions/700166/allow-multiple-roles-to-access-controller-action
- https://www.udemy.com/course/the-complete-aspnet-mvc-5-course/
- https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/various-ways-to-pass-data-from-controller-to-view-in-mvc/
- http://kurs.aspnetmvc.pl/
- https://devblogs.microsoft.com/aspnet/customizing-profile-information-in-asp-net-identity-in-vs-2013-templates/
- https://stackoverflow.com/questions/42471866/how-to-create-roles-in-asp-net-core-and-assign-them-to-users
- https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-3.1
- https://pl.wikipedia.org/wiki/ASP.NET_MVC