|  |  |  |
| --- | --- | --- |
| [Politechnika ¦l±ska](http://www.polsl.pl/) | Politechnika Śląska  Wydział Inżynierii Materiałowej  i Metalurgii | |
| Temat:  Projekt: SmartShop  Dokumentacja | | |
| Komentarze: | | |
| Data:  Podpis: | | Aleksandra Miękina,  Tomasz Szostak,  Damian Kotulski,  Krzysztof Kurkiewicz |
| IPP-30PP |

Spis treści

[1. Założenia projektu//Krzysztof 3](#_Toc485926945)

[2. Wykorzystane technologie//Krzysztof 3](#_Toc485926946)

[a. Spis i opis technologii//Krzysztof 3](#_Toc485926947)

[b. Wtyczki wspomagające //Krzysztof 3](#_Toc485926948)

[3. Diagramy 5](#_Toc485926949)

[a. Diagram przypadków użycia//Krzysztof 5](#_Toc485926950)

[b. Diagram klas programu//Aleksandra 6](#_Toc485926951)

[c. Diagram encji bazy danych//Tomasz 6](#_Toc485926952)

[4. Baza Danych//Tomasz 6](#_Toc485926953)

[a. Opis//Tomasz 6](#_Toc485926954)

[b. Opis procedur//Tomasz 6](#_Toc485926955)

[5. PluginLogIn//Krzysztof 6](#_Toc485926956)

[a. Opis projektu//Krzysztof 6](#_Toc485926957)

[b. Najważniejsze kody wraz z komentarzem//Krzysztof 6](#_Toc485926958)

[6. SmartShop.CommunicateToWebService 8](#_Toc485926959)

[a. Opis projektu 8](#_Toc485926960)

[b. Najważniejsze kody wraz z komentarzem 8](#_Toc485926961)

[7. SmartShop.Models//Krzysztof 10](#_Toc485926962)

[a. Opis projektu//Krzysztof 10](#_Toc485926963)

[b. Najważniejsze kody wraz z komentarzem//Krzysztof 10](#_Toc485926964)

[8. SmartShop.Tests//Krzysztof 14](#_Toc485926965)

[a. Opis projektu//Krzysztof 14](#_Toc485926966)

[b. Najważniejsze kody wraz z komentarzem//Krzysztof 14](#_Toc485926967)

[9. SmartShopWebApp 18](#_Toc485926968)

[a. Opis projektu 18](#_Toc485926969)

[b. Najważniejsze kody wraz z komentarzem 18](#_Toc485926970)

[10. SmartShopWpf//Krzysztof + Ola 26](#_Toc485926971)

[a. Opis projektu//Krzysztof 26](#_Toc485926972)

[b. Najważniejsze kody wraz z komentarzem//Krzysztofa+Ola 26](#_Toc485926973)

[11. Podział obowiązków//Sami sie dopisujemy 32](#_Toc485926974)

[a. Tomasz Szostak 32](#_Toc485926975)

[b. Krzysztof Kurkiewicz 32](#_Toc485926976)

[c. Aleksandra Miękina 32](#_Toc485926977)

[d. Damian Kotulski 32](#_Toc485926978)

# Założenia projektu//Krzysztof

Założeniem projektu było wykonać nowoczesną aplikację, która potrafiłaby zastąpić kasę fiskalną dla sklepu. Aplikacja ma mieć pełną fukncjonalność pozwalającą na obsłużenie klienta przez sprzedawcę, jednocześnie mając prosty i intuuicyjny wygląd, aby mogła być obsłużona przez niewykwalifikowany w sposób techniczny personel. W tych funkcjonalnościach powinno mieścić się minimum:

1. możliwość logowania się na własne, spersonalizowane konto
2. możliwość wybrania produktu z listy wszystkich produktów
3. możliwość wpisania kodu danego produktu(imitacja skanowania kodów kreskowych)
4. możliwość wpisywania i edycji ilości sztuk zakupionego przedmiotu
5. możliwość usuwania przedmiotów z listy zakupionych
6. możliwość dokonywania zwrotów
7. możliwość drukawania paragonów
8. możliwość drukowania raportów w wybranym odstępie czasowym
9. możliwość nadawania zniżek całościowych oraz pojedyńczych, złotówkowych oraz procentowych
10. możliwość zdjęcia vat z produktu
11. możliwość przeglądania wszystkich transakcji

# Wykorzystane technologie//Krzysztof

# Spis i opis technologii//Krzysztof

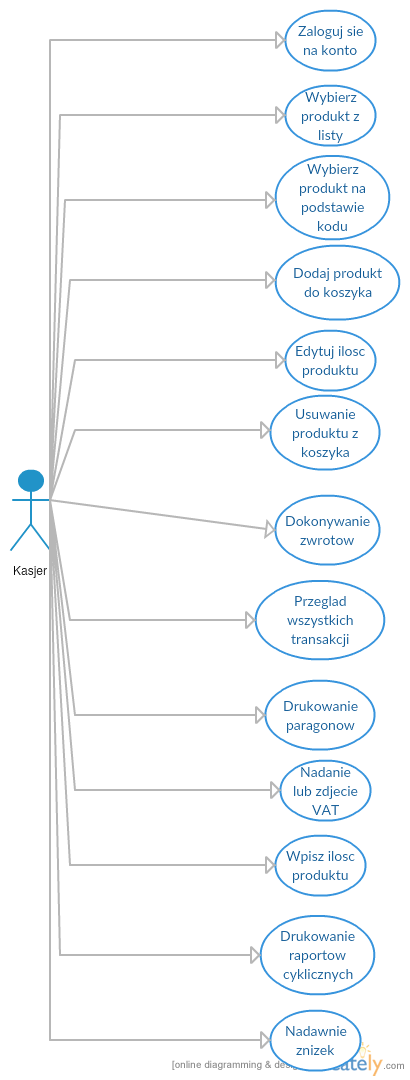
* + 1. Baza danych – MariaDB postawiona na zewnętrznych serwerze
    2. Środowisko do połączenią się z bazą - HeidiSQL
    3. Aplikacja kliencka – Windows Presentation Foundation w języku C#
    4. Webservice – w architekturze REST, napisany w języku C# w aplikacji typu ASP.NET, użyty ORM to Entity Framework
    5. Środowisko – Microsoft Visual Studio 2015

# Wtyczki wspomagające //Krzysztof

* + 1. MySQL.Data oraz MySql.Data.Entity.EF6 – wtyczki pozwalające na połączenie technologii .NET z bazą napisaną na silniku MySQL
    2. Newtonsoft.Json – wtyczka pozwalająca na serializacje i deserializacje informacji w formacie Json
    3. NUnit 3 – wtyczka pozwalające na pisanie testów jednostkowych
    4. Moq – wtyczka pozwalająca na mokowanie zewnętrznych zależności do testów
    5. RestSharp – wtyczka ułatwiająca połączenie z webservicem o architekturze REST
    6. Owin – wtyczka do wymuszenia autoryzacji na bazie tokenu w kontrolerach
    7. CodeMaid – wtyczka pozwalająca zachować czystość i zwięzłość w kodzie
    8. ReSharper – wtyczka rozszerzająca możliwości Visual Studio
    9. Spire.PDF – do drukowania paragonów oraz raportów w postaci PDF

# Diagramy

# Diagram przypadków użycia//Krzysztof



# Diagram klas programu//Aleksandra

# Diagram encji bazy danych//Tomasz

# Baza Danych//Tomasz

# Opis//Tomasz

# Opis procedur//Tomasz

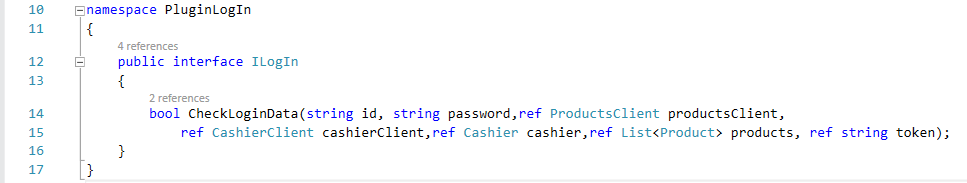
# PluginLogIn//Krzysztof

# Opis projektu//Krzysztof

Projekt ma za zasadnie spełnić umożliwić zalogowanie do aplikacji. Pluginy stosuję się w celu umożliwienia rozszerzenia aplikacji, bez zmieniania kodu. Na samym początku Plugin sprawdzał tylko poprawność loginu i hasła dla jednego zahardkorowanego przypadku. Następnie ten plugin zamieniliśmy pluginem w obecnej postaci, bez zmiany kodu, a dodając rzeczywiste logowanie.

# Najważniejsze kody wraz z komentarzem//Krzysztof

* + 1. Interface, jego zadaniem jest dostarczenie metody dla klasy implementującej. Duża liczba nadpisywanych zmiennych wynika z faktu zapotrzebowania na nie w kodzie.



* + 1. Implementacja interfejsu wykorzystująca do logowania token generowany przez odpowiedni projekt.



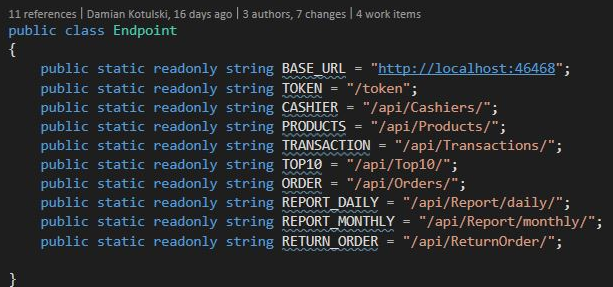
# SmartShop.CommunicateToWebService

# Opis projektu

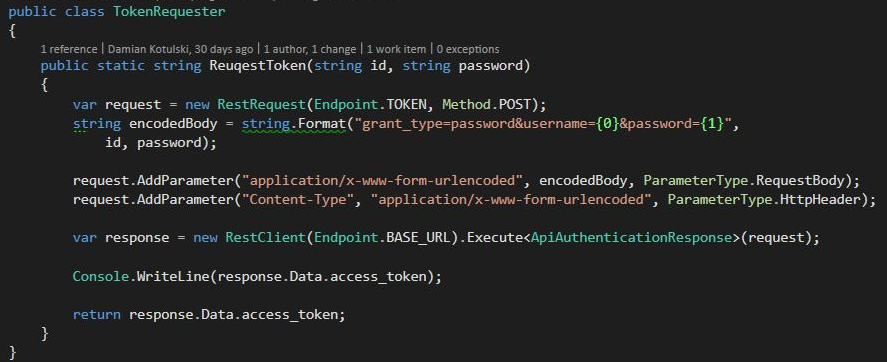
Projekt zawiera klasy odpowiedzialne za komunikację aplikacji z WebService’m.

# Najważniejsze kody wraz z komentarzem

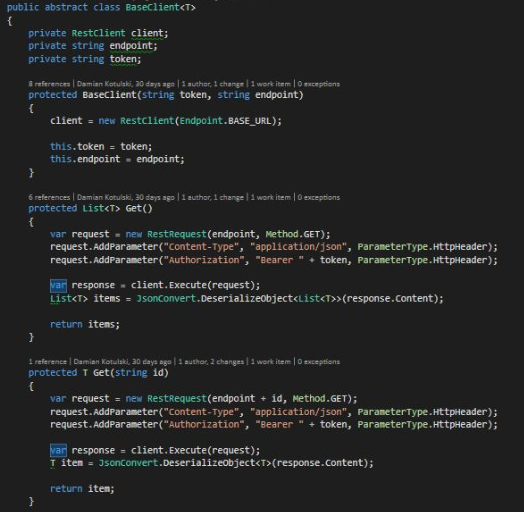
Klasa Enpoint – zawiera statyczne pola, będące adresami kontrolerów usług

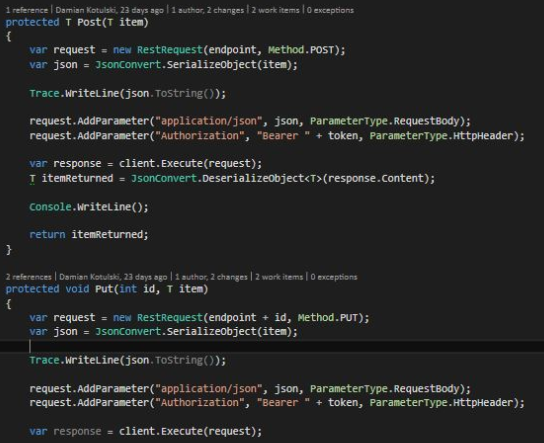


Klasa TokenRequester – zawiera metodę odpowiedzialną za uzyskanie tokenu, niezbędnego do dalszej komunikacji



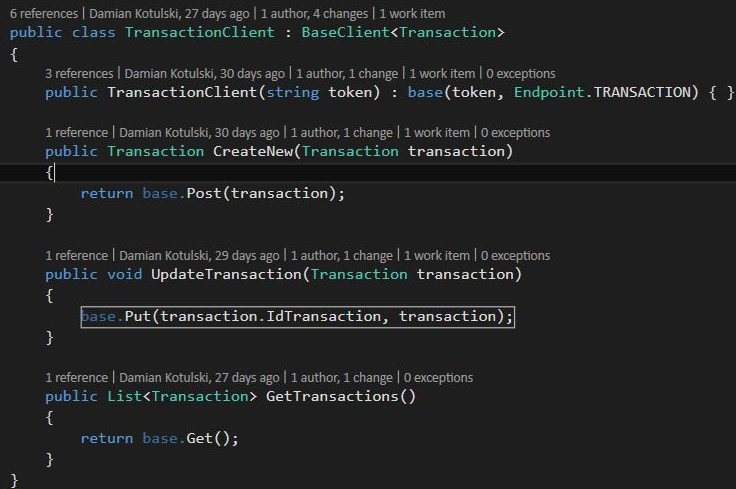
Klasa BaseClient – bazowy klient odpowiedzialny za komunikację, klasa abstrakcyjna zawierająca generyczne metody do wykonywania zapytań http





Pozostałe klasy stanowią rozszerzenia klasy BaseClient i służą do obsługi dostępu do usług konkretnych kontrolerów WebService’u. Przykład:

Klasa TransactionClient – obsługuje utworzenie, aktualizację i pobranie transakcji:



# SmartShop.Models//Krzysztof

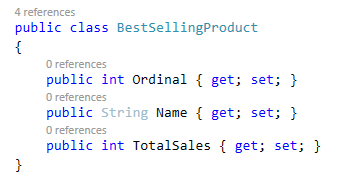
# Opis projektu//Krzysztof

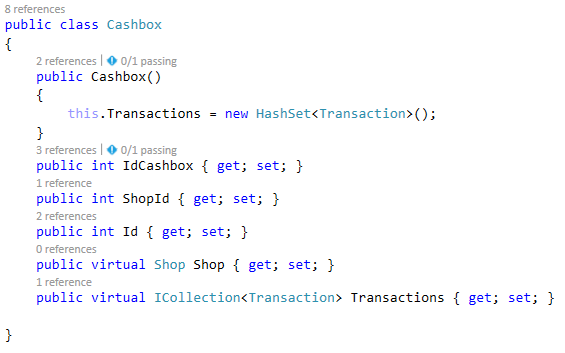
Stworzenie tego projektu miało na celu wyodrębnienie modeli klas z których będziemy korzystać w pozostałych projektach, aby nie tworzyć każdego z nich wieloktrotnie.

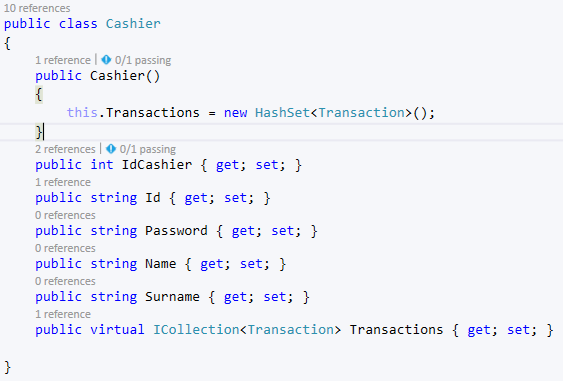
# Najważniejsze kody wraz z komentarzem//Krzysztof

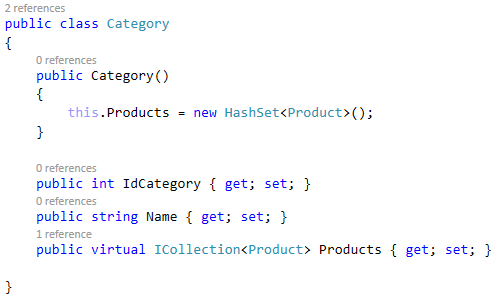
Projekt składa się z pojedyńczych klas pełniących role pojedyńczych modeli. Zostały one stworzone na podstawie wygenerowanych klas EntityFramework w webserviscie.

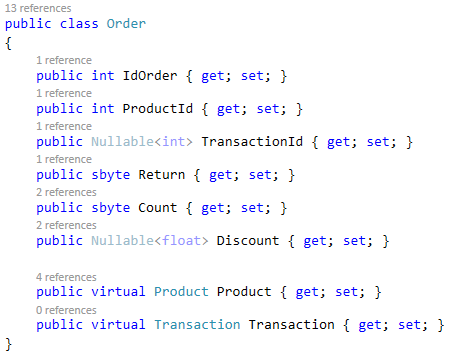
Kody poszczególnych modeli:

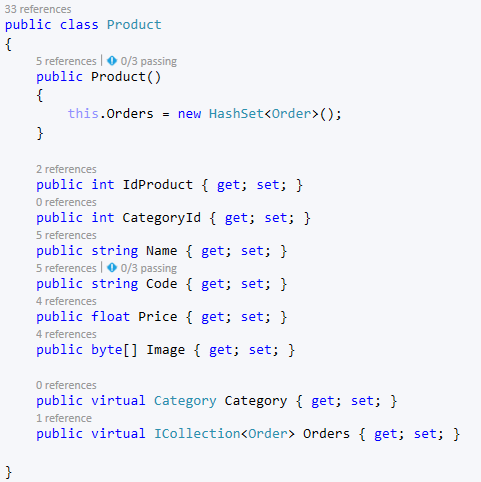




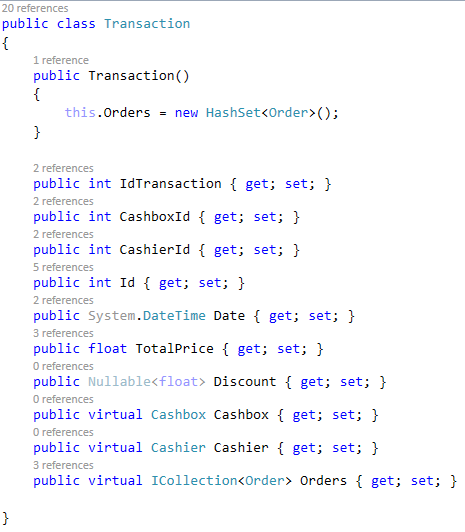












# SmartShop.Tests//Krzysztof

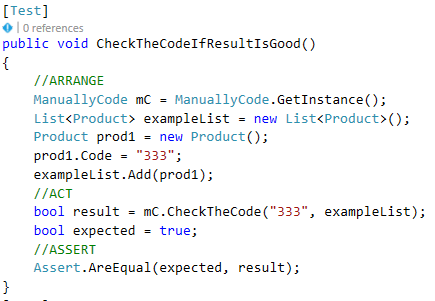
# Opis projektu//Krzysztof

Projekt został wykonany, z powodu potrzeby zrobienia testów jednostkowych dla całej solucji. Zostały w nich wykonanych siedem testów jednostkowych, wszystkie przechodzą. Do ich utworzenia wykorzystano bibliotekę NUnit w wersji 3.

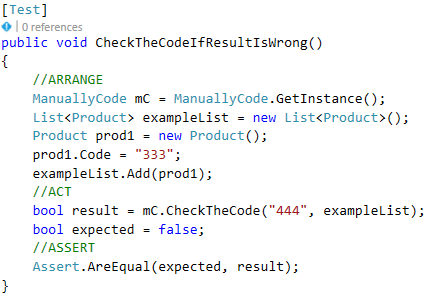
# Najważniejsze kody wraz z komentarzem//Krzysztof

Przetestowane zostały dwie klasy z solucji, wybrane z nich metody.

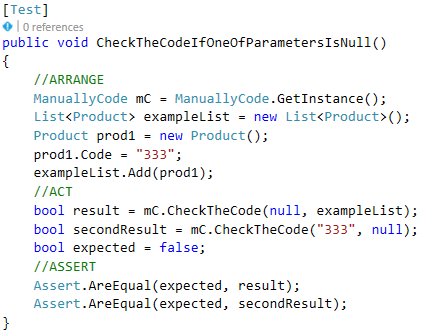
* + 1. ManuallyCodeTests



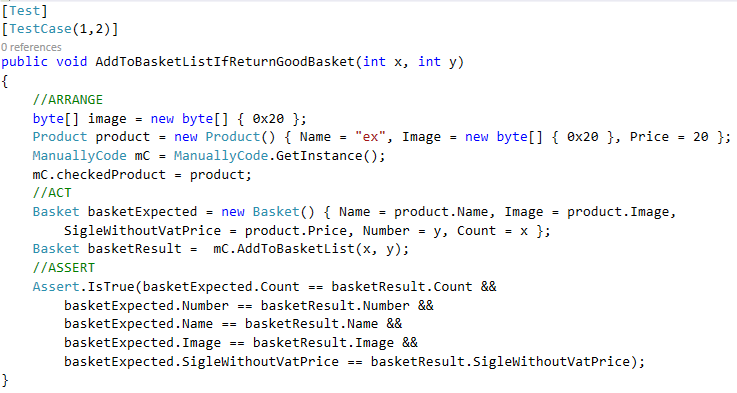
Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. znajdzie kod produktu w liście.



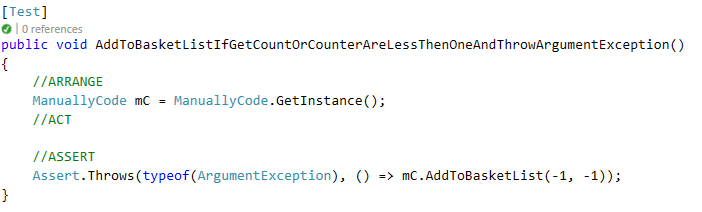
Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. nie znajdzie kodu produktu w liście.



Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. nie jeden z parametrów wchodzącym do metody jest nullem.

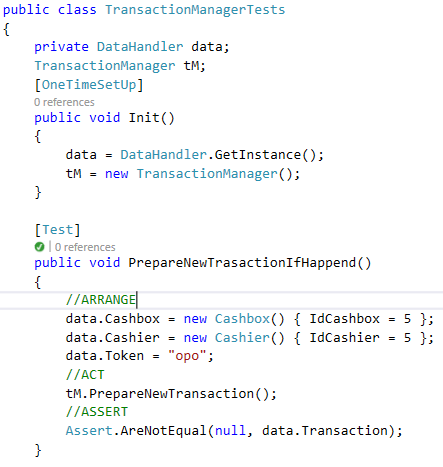


Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. zwraca obiekt klasy Basket z odpowiednimi polami. Wykorzystuje także TestCasy.



Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. zwraca błąd jeśli wprowadzone dane są nieprawidłowe, mniejsze niż zero.

* + 1. TransactionManagerTests



Powyższy test sprawdza czy metoda zwraca odpowiednią wartość jeśli spełni swoje zadanie, tzn. sprawdza czy transakcja została utworzona. Co więcej, test przechodzi tylko jako Run Selected, z powodu nie błędu testu, a zależności wystepującej w źle przeprojektowanej klasie.

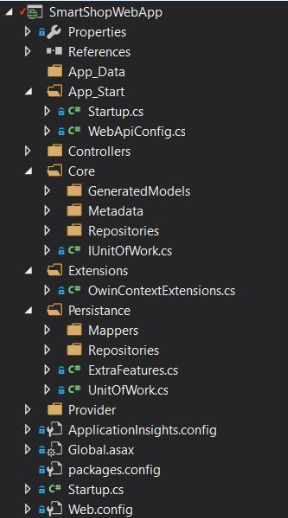
# SmartShopWebApp

# Opis projektu

Projekt zawiera warstwę WebService’u, zaimplementowaną na bazie wzorca REST, zabezpieczoną mechanizmem autoryzacji opartym o standard OAuth 2.0. Wykorzystano EntityFramework i podejście database first oraz wzorce Repozytorium i UnitOfWork w celu łatwiejszego i ujednoliconego dostępu do danych z modeli podczas wykorzystywania ich w kontrolerach.

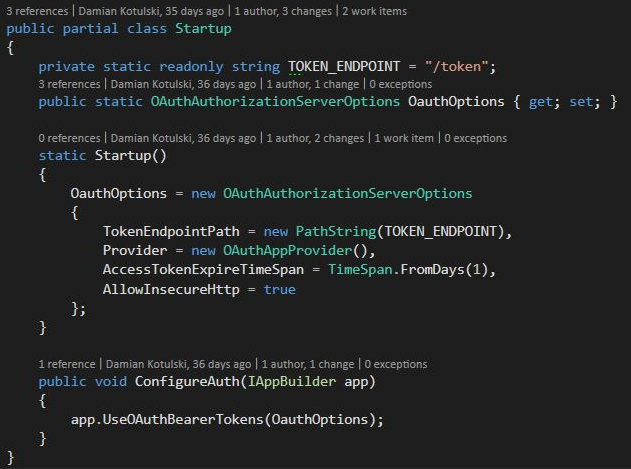
# Najważniejsze kody wraz z komentarzem

* Struktura projektu



* Omówienie najważniejszych komponentów

**App\_Start:** klasa Startup.cs



- konfiguracja mechanizmu uzyskiwania tokena (ścieżka dostępu, czas wygaśnięcia) do komunikacji zgodnej ze standarem OAuth 2.0

**App\_Start:** klasa WebApiConfig.cs

- konfigurację usług i reguły routingu

**Controllers:** klasy będące kontrolerami usług, reagują na zapytania http i zwracają odpowiedzi w formacie json

Specyfikacja najważniejszych dostępnych usług:

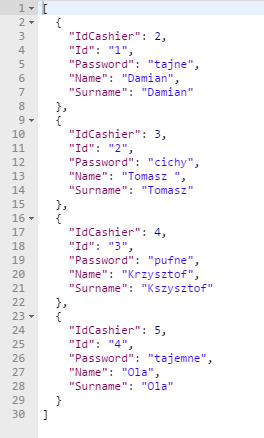
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pobieranie  GET | Dodawanie  POST | Modyfikacja  PUT | Usuwanie  DELETE |
| Shop | X  (w Cashboxes) | X | X | X |
| Cashboxes | **TAK** | X | X | X |
| Cashiers | TAK | X | TAK  (hasło) | X |
| Transactions | TAK | TAK | TAK | X |
| Orders | TAK | X  (w Transactions) | TAK | X |
| Products | TAK | X | X | X |
| Categories | X  (w Products) | X | X | X |

Cashiers

* GET

**ENDPOINT**: /api/cashiers

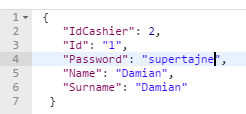
PRZYKŁADOWA ODPOWIEDŹ:



* PUT

**ENDPOINT**: /api/transactions/{idCashier}

PRZYKŁADOWE BODY:



ODPOWIEDŹ:

Bez odpowiedzi, ale status informuje, że wszystko przebiegło pomyślnie:

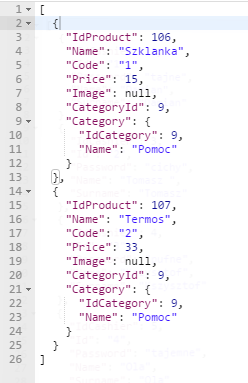


Products

* GET

**ENDPOINT**: /api/products

PRZYKŁADOWA ODPOWIEDŹ:



Cashboxes

* GET

**ENDPOINT**: /api/cashboxes I /api/ cashboxes /{id} dla konkretnego idCashbox

PRZYKŁADOWA ODPOWIEDŹ:



Orders

* GET

**ENDPOINT**: /api/orders

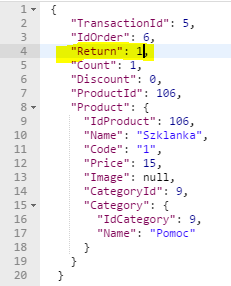
PRZYKŁADOWA ODPOWIEDŹ:



* PUT

**ENDPOINT**: /api/transactions/{idTransaction}

PRZYKŁADOWE BODY: (zwracamy produkt)



ODPOWIEDŹ:

Bez odpowiedzi, ale status informuje, że wszystko przebiegło pomyślnie:

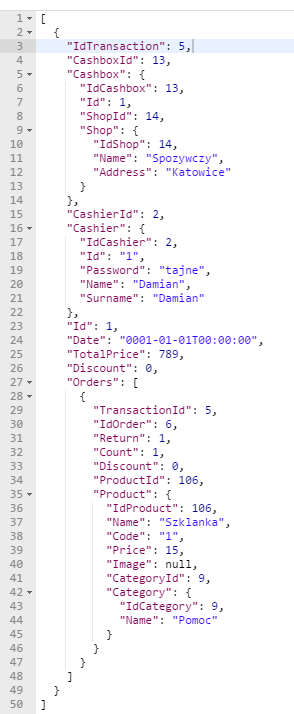


Transactions

* GET

**ENDPOINT**: /api/transactions I /api/transactions/{id} dla konkretnego idTransaction

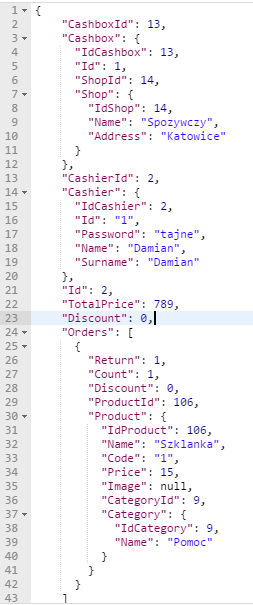
PRZYKŁADOWA ODPOWIEDŹ:



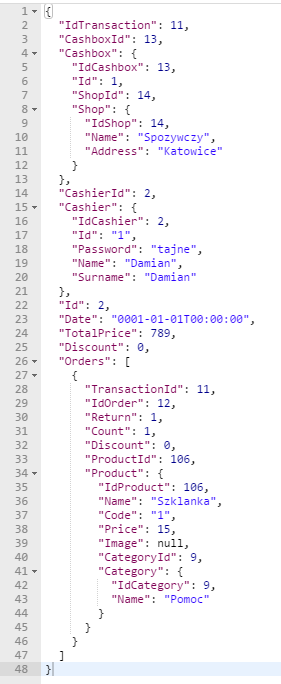
* POST

**ENDPOINT**: /api/transactions

PRZYKŁADOWE BODY: (nie ma idtransaction i idorder)



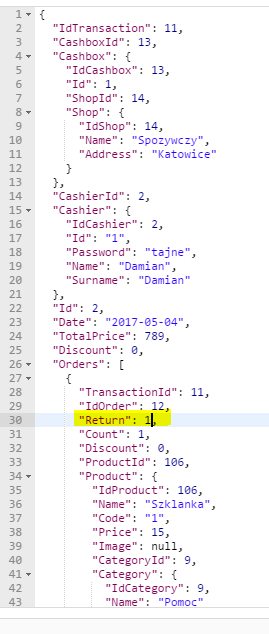
PRZYKŁADOWA ODPOWIEDŹ: (dostajemy odpowiedź z wygenerowanym idTransaction i idOrder dla każdego z zamówienia)



* PUT

**ENDPOINT**: /api/transactions/{idTransaction}

PRZYKŁADOWE BODY: (zwracamy produkt)



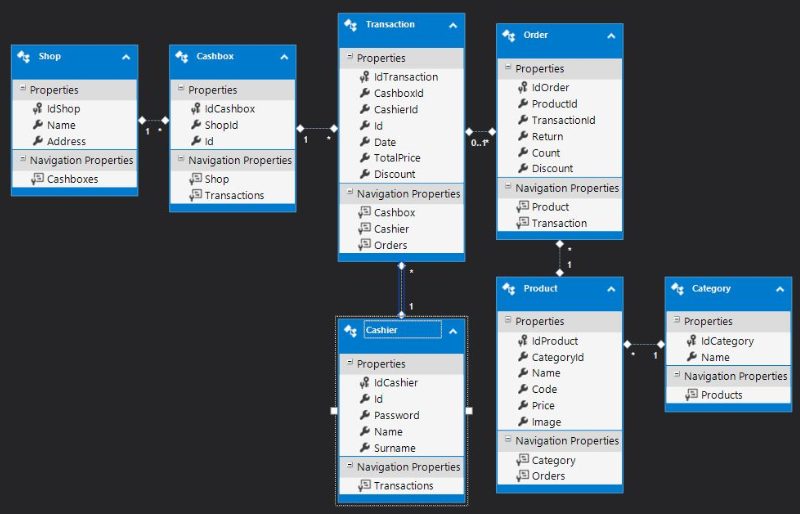
ODPOWIEDŹ:

Bez odpowiedzi, ale status informuje, że wszystko przebiegło pomyślnie:



**Core/GeneratedModels:** klasy będącemodelami danych, wygenerowane przez Entity Framework na podstawie istniejących modeli w bazie danych

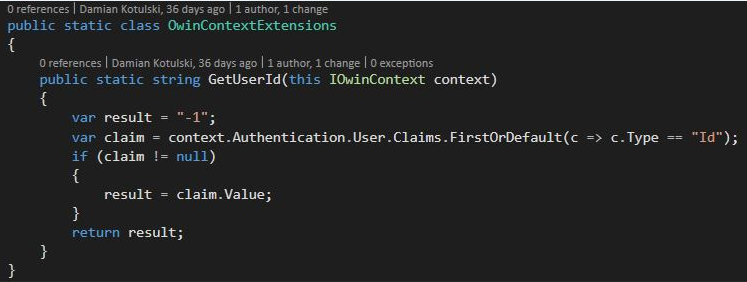
Diagram klas modeli

****

**Core/Metadata:** klasy partial dla modeli, będące klasami typu Metadata (przechowują dane o danych), zawierają atrybuty określające kolejność serializacji i metody pozwalające wyłączyć serializację właściwości nawigacyjnych. Wszelkie zmiany na modelach muszą być dokonywane w tych klasach, ponieważ klasy wygenerowane przez Entity Framework mogą zostać nadpisane podczas wykonania update modelu w przypadku gdy zmianie ulegnie struktura bazy.

**Core/Repositories:** warstwa abstrakcyjna (interfejsy) wzorca Repozytorium i UnitOfWork

**Extensions:** klasaOwinContextExtensions

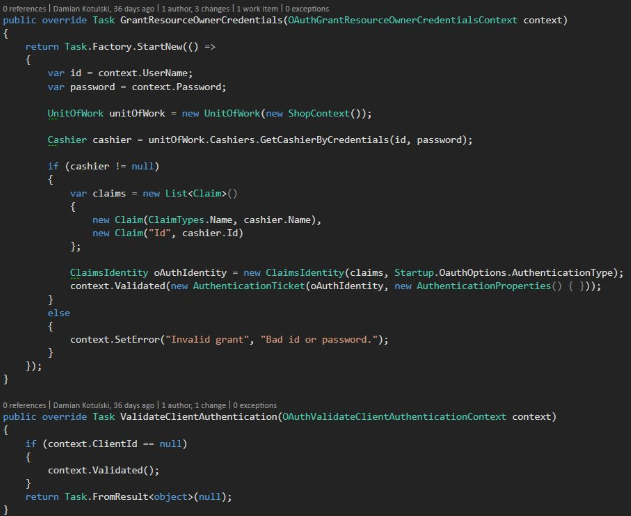


**-** klasa zawierająca metodę rozszerzającą, która obsługuje mechanizm przyznawania uprawnień dostępu do usług WebService’u dla konkretnych użytkowników (kasjerów)

**Persistance/Mappers:** klasy będące modelami danych dla komponentów wymaganych do obługi procedur składowanych z bazy danych

**Persistance/Repositories:** implementacja interfejsów (warstwy abstrakcyjnej) wzorca Repozytorium i UnitOfWork

**Provider**: klasa OAuthAppProvider



**-** asynchroniczna obsługa mechanizmy dawania uprawnień do dostępu do usług WebService’u dla kasjera posiadającego aktualny i poprawny token

# SmartShopWpf//Krzysztof + Ola

# Opis projektu//Krzysztof

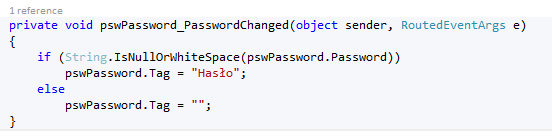
Główny projekt zawierający nasz program. Należą do niego widoki oraz część backendowa porozmieszczana w osobnych klasach. To ten projekt odpowiada za działanie aplikacji.

# Najważniejsze kody wraz z komentarzem//Krzysztofa+Ola

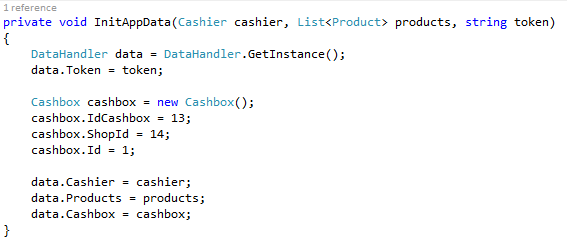
* + 1. LoginWindow



Powyższy event odpowiedzialny jest za poprawne zalogowanie się na użytkownika. Działanie logowania wykonuje sie w pluginie, który zwraca true/false, zależnie od opcji event reaguje przejściem do następnego widoku, błąd błedem. Wczytywanie pluginu jest zrobione asynchroniczne, dla szybszego działania.

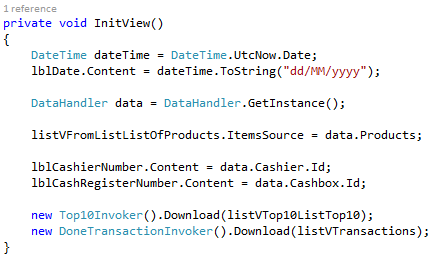


Powyższy event odpowiedzialny jest za poprawne oznaczenie tagiem passwordboxa, zależnie od jego zawartości.

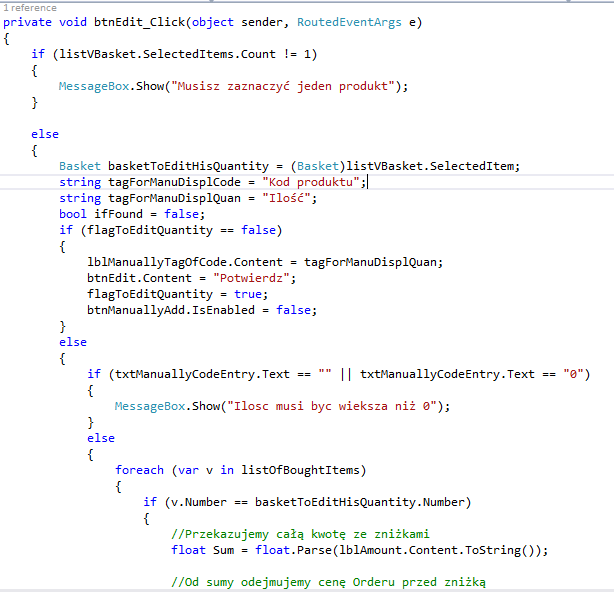


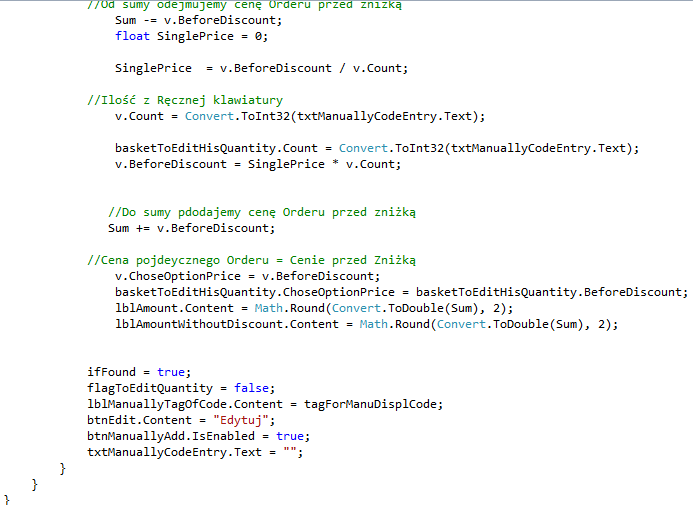
Powyższa metoda służy do inicjalizacji danych początkowych.

* + 1. MainWindow

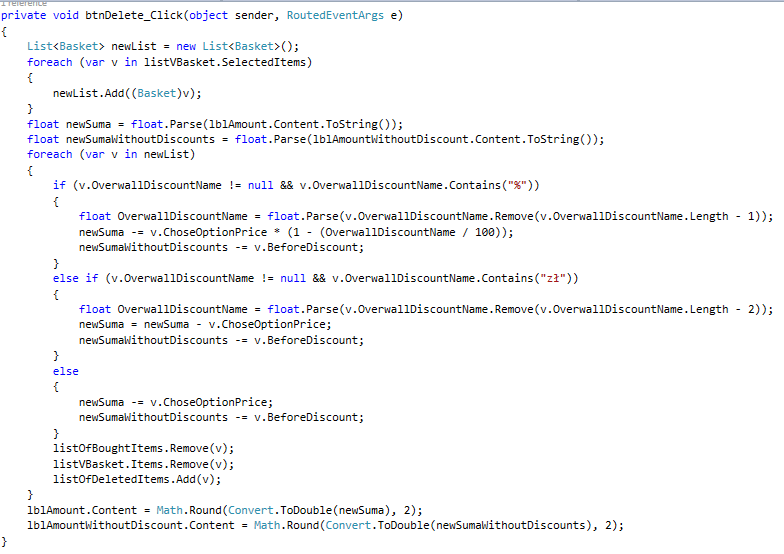


Powyższa metoda służy do inicjalizacji danych w oknie.

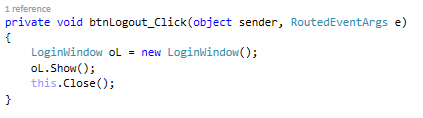




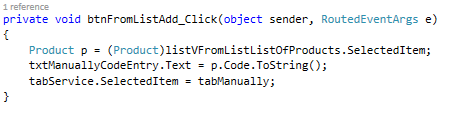
Powyższy przycisk służy do edycji ilości przedmiotów w koszyku.



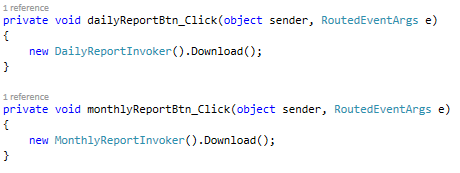
Powyższy przycisk służy do usuwania przedmitów z koszyka.



Powyższy przycisk służy do wylogowania się i przejścia do LoginWindow



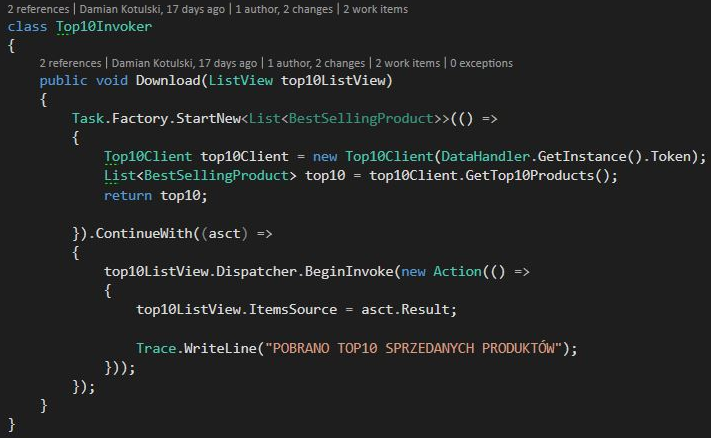
Powyższy przycisk służy do wybierania przedmiotu z listy przedmiotów.



Powyższe metody służą do drukowania raportów.

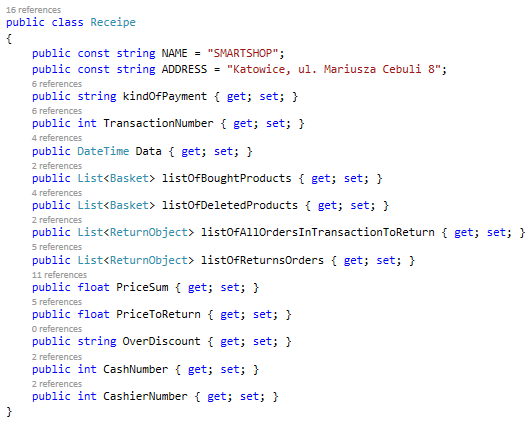
* + 1. DiscountWindow
    2. PaymentWindow
    3. Asynchronous folder

Folder zawiera klasy odpowiedzialne za asynchroniczne przetwarzanie długotrwałych operacji. Wykorzystaliśmy bibliotekę zadań równoległych (TPL), aby wyeliminować przerwy w możliwości korzystania z graficznego interfejsu. Wszystkie klasy opierają się na schemacie utworzenia zadania (Task) poprzez wbudowaną w system fabrykę, wykonaniu go oraz zwróceniu lub wyświetleniu rezultatu w aplikacji. Przykład kodu:



Asynchroniczne pobranie i wyświetlenie 10 najczęściej sprzedawanych produktów.

* + 1. Data folder



Powyższa klasa służy do stworzenia rachunku, zbiera wszystkie potrzebne informacje z przedmiotów w koszyku, które są potrzebne do wydrukowania paragonu.

* + 1. ReceipeMethods folder

Folder składa się z czterech klas odpowiedzialnych za drukowanie: Paragonu, Zwrotu, Dniowego oraz Miesiecznego raportu. Klasy te między sobą różnią się tylko drukowanymi informacjami. Ze względu na obszerność kodu nie umieszczamy go tutaj. Pdf’y tworzy sie jako pustą stronę. Nastepnie po określeniu rozmiaru strony, możemy drukować przy pomocy ustawiania współrzędnych X oraz Y. Strony są dodawane dynamicznie, jeśli zajdzie taka potrzeba, czyli spełni się warunek, że współrzędna Y przekroczyła limit.

* + 1. ViewModel folder

# Podział obowiązków//Sami sie dopisujemy

# Tomasz Szostak

# Krzysztof Kurkiewicz

1. Obsługa przycisków Edytuj/Usuń/Wyloguj w MainWindow
2. Wyszukiwanie produktow z listy w MainWindow
3. Generowanie paragonów/raportów w formacie .pdf
4. Plugin MockLogowania, a następnie przypisanie nowego pluginu PlugLogIn
5. Użycie pluginów logowania w oknie logowania
6. Zapewnienie asynchronicznego działania pewnych funkcji
7. Wykonanie UnitTestow
8. Wykonanie logo dla projektu
9. Wyodrębnienie modeli do SmartShop.Models

# Aleksandra Miękina

# Damian Kotulski

1. Organizowanie pracy zespołu, tworzenie zadań i nadzorowanie rozwoju projektu na platformie github

2. Konfiguracja, zarządzanie i wdrożenie zespołu w system kontroli wersji git

3. Utworzenie WebService’u:

* stworzenie modeli danych odpowiadających tabelą w bazie danych
* implementacja wzorców Repozytorium i Unit of Work
* stworzenie kontrolerów obsługujących metody http spełniające założenia aplikacji
* obsługa procedur składowanych w bazie danych
* implementacja standardu autoryzacji OAuth 2.0

4. Utworzenie modułu odpowiedzialnego za komunikację aplikacji głównej z WebService’m

5. Wyświetlanie listy dostępnych produktów w aplikacji głównej

6. Implementacja mechanizmu odpowiedzialnego za obsługę transakcji - tworzenie, aktualizowanie, dodawanie do bazy

7. Wyświetlanie wszystkich przeprowadzonych transakcji w aplikacji głównej

8. Wyświetlanie top10 produktów w aplikacji głównej

9. Implementacja asynchronicznej komunikacji aplikacji głównej z WebService’m