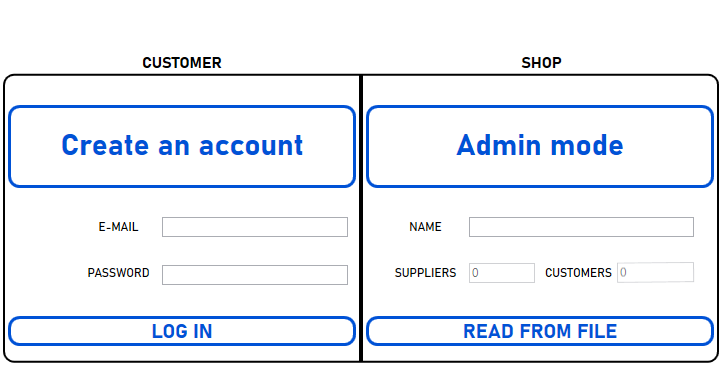
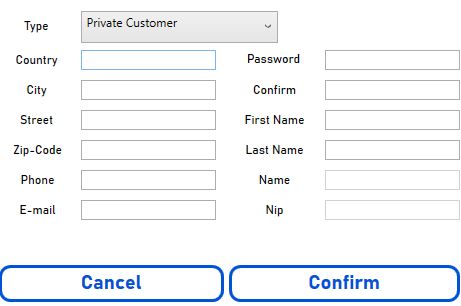
# Sprawozdanie z projektu C#

## Wstęp

Tematem naszego projektu jest sklep komputerowy, a dokładniej aplikacja obsługująca sklep internetowy z poziomu pracownika oraz potencjalnego nabywcy. Nasza grupa jest dwuosobowa, a po wstępnych ustaleniach zdecydowaliśmy się na pracę na Github.

# Działanie naszej aplikacji (TUTORIAL)

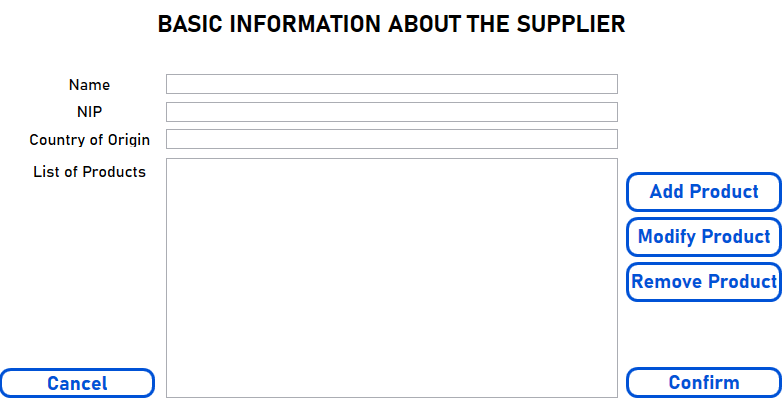
Postawiliśmy na realizację aplikacji obsługującej sklep komputerowy. Do sklepu komputerowego można przejść z poziomu klienta oraz z poziomu administratora, co pozwala na symulację zarządzania sklepem, jak i na symulację zakupów.

Aby przejść do aplikacji z poziomu klienta należy założyć konto lub takowe posiadać i zwyczajnie się do niego zalogować za pomocą e-mailu oraz hasła. Klientem może być osoba prywatna lub firma. Z poziomu klienta można dokonywać zakupów, tzn. pakować koszyk produktów i zgłaszać go do realizacji.

Ważnym jest fakt, że osoba prywatna nie musi podawać NIP’u oraz nazwy firmy, dlatego po wybraniu typu „Private Customer” okna Name oraz Nip będą nieaktywne – firma również nie ma Imienia i Nazwiska dlatego analogicznie nie może ona wypełnić tychże pól. Niezależnie od typu klienta logujemy się tak samo – poprzez e-mail i hasło.

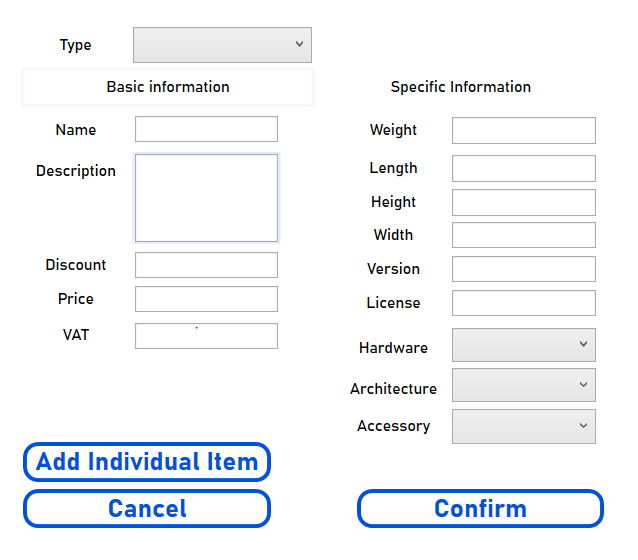


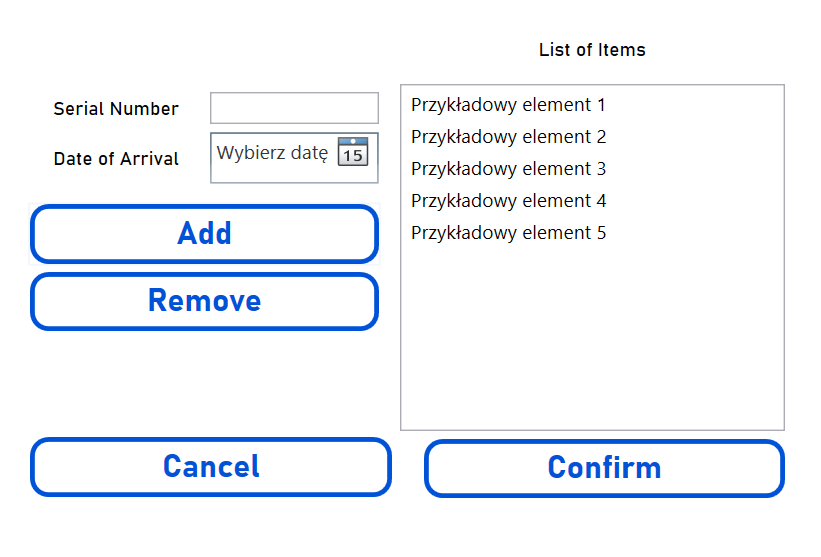
Aby przejść do aplikacji z poziomu administratora wystarczy kliknąć na przycisk „Admin mode”. Nie ma wymagań dotyczących logowania, niemniej jednak można wcześniej załadować sklep z pliku (serializacja JSON). Aplikacja domyślnie pokaże nazwę odczytanego sklepu oraz jego liczbę dostawców i klientów – jest to sposób weryfikacji poprawności wczytanego pliku. Z poziomu administratora można dodawać dostawców oraz produkty na stanie naszego sklepu.

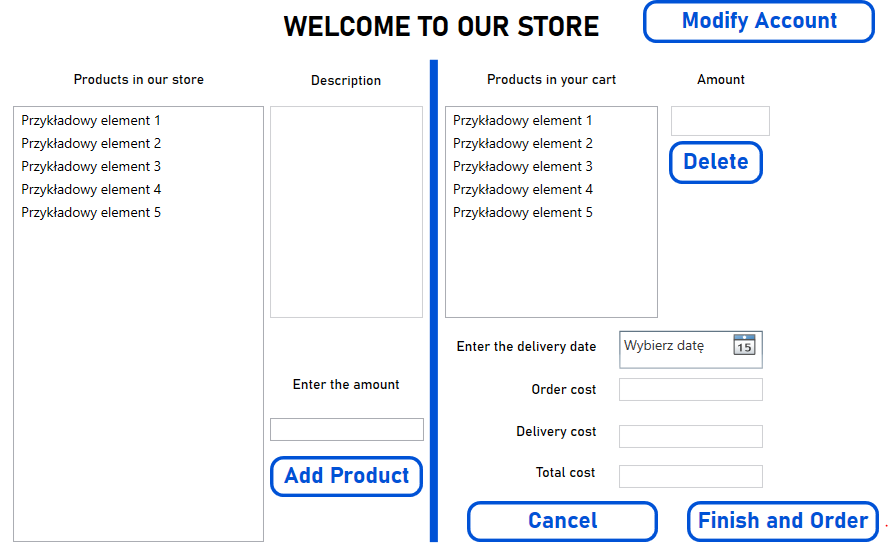
Z poziomu administratora możemy dodawać dostawców oraz ich modyfikować. Ważne jest to, że dodanie produktu możliwe jest po wybraniu dostawcy, tzn. aby dodać/modyfikować produkt należy wybrać dostawcę, który za niego odpowiada.

Zarządzanie produktami

Z poziomu administratora można również spojrzeć na zamówienia w sklepie dokonane przez klientów. Można również zobaczyć szczegóły zamówień – w tym miejsce dostawy. Można je usunąć lub zrealizować.

Dodawanie produktu odbywa się w sposób podobny do tworzenia nowego klienta – w zależności od typu produktu aktywują się różne okna – inne właściwości mają produkty typu hardware, a inne typu software. Co jest ważne, to dodanie produktu nie znaczy, że jest on możliwy do kupienia – należy dodać indywidualny egzemplarz danego produktu. Przechodzi się do tego z modify product -> add individual item.

Aby dodać egzemplarz należy wpisać numer seryjny oraz dzień dostawy danego egzemplarza do sklepu. Egzemplarze można usuwać. To one również podlegają zakupom – użytkownik wybiera produkt i go kupuje, ale tak naprawdę przypisywany jest do zamówienia dany egzemplarz produktu, a gdy egzemplarzy nie ma, to i nie można kupić danego produktu, mimo że jest on na liście



Kupowanie w sklepie jest proste – wybieramy produkty z listy, wpisujemy liczbę produktów i dodajemy produkt do koszyka. Koszyk znajduje się po prawej stronie sklepu i wyświetla on parametry naszego zamówienia – produkty, koszt produktów, cenę dostawy i całkowity koszt. Produkty można usunąć z koszyka. Dodatkowo klient może modyfikować swoje dane osobowe.

Dodatkowo, aplikacja, dokumentacja oraz sam kod pisane są po angielsku – uznaliśmy, że tak będzie nieco uniwersalniej. Dokumentacja generowana jest w XML, a diagram UML załączony jest jako plik PNG, niemniej jednak jest również kod generujący diagram UML realizowany za pomocą wtyczki do Visual Studio Code – plant UML.

**Program sprawdza wszystkie wpisywane dane dlatego nie przepuści błędnego numeru PESEL oraz NIP’u.** Wszystko sprawdzane jest pełnymi algorytmami dlatego należy wpisać możliwy istniejący PESEL (można np. swój) oraz NIP jakiejkolwiek firmy (np. 5272677009 – Ringier Axel Springer). Dodatkowo większość danych musi zaczynać się z dużej litery (np. imię), dlatego **należy dbać o poprawność wprowadzanych danych**, bo program może później wyświetlać komunikaty o błędach.

## Zrealizowane aspekty projektu

W miejscach, gdzie była możliwość wyboru (np. pomiędzy XML, a JSON, to zostawione zostało zwyczajnie to, co zostało przez nas wybrane).

**Na ocenę 3 należy oddać:** (ocena wystawiana jest po obronie projektu)

* Kod klas w języku C#, zawierający wykorzystanie jednej z omawianych na zajęciach kolekcji.
* Odczyt danych do/z pliku (serializacja JSON).
* Prosty interfejs graficzny (GUI) pokazujący działanie wybranego systemu.

**Na ocenę 3,5 dodatkowo:**

* Kod powinien korzystać z mechanizmu interfejsów oraz klas abstrakcyjnych.
* Tworzenie własnych wyjątków.

**Na ocenę 4 dodatkowo:**

* Wykorzystanie interfejsów **IClonable**, **IComparable, IEquatable** – kopiowanie, sortowanie, porównywanie danych.
* Diagram UML dla klas występujących w systemie.

**Na ocenę 4,5 dodatkowo:**

* Testy jednostkowe podstawowych klas systemu.
* Rozbudowany interfejs graficzny (GUI), pozwalający na łatwą obsługę kluczowych elementów systemu.
* Wykorzystanie delegatów.

## Podział pracy

Z racji, że nasza grupa jest tylko dwuosobowa, to zdecydowaliśmy się na realizowanie większości zadań razem, tzn. trudno ustalić podział pracy w części czystego kodowania, gdyż razem nanosiliśmy poprawki i stosowaliśmy nasze własne, jednoosobowe peer review. Niemniej jednak niektóre zadania wykonywaliśmy głównie sami, a podział jest następujący:

* **Wiktoria Arendarczyk**
  + Implementacja UML <plant UML>,
  + Dokumentacja,
  + Sprawozdanie,
  + Wygląd GUI.
* **Cezary Moskal**
  + Testy jednostkowe,
  + Implementacja GUI.

Resztę zadań, czyli wstępne projektowanie obiektowe, pisanie naszej Class Library więc trudno ustalić tu jakiś sensowny podział.