

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Technologie Internetowe i aplikację mobilne

**Projekt**

*Dokumentacja do wykonanych zadań na laboratorium*

**Prowadzący:**

mgr. Inż. Tomasz Rajzer

**Student:**

Ihor Yaremchuk

**Nr. Albumu:**

W61863

Spis treści

[**Wstęp** 3](#_Toc136940723)

[**1.** **Diagram przypadków użycia** 4](#_Toc136940724)

[**2.** **Diagram klas** 7](#_Toc136940725)

[**3.** **Diagram stanów** 11](#_Toc136940726)

[**4.** **Paymentprocessor diagram** 12](#_Toc136940727)

[**5.** **Migracje, encje api** 13](#_Toc136940728)

[**Wniosek:** 16](#_Toc136940729)

**Wstęp**

Link do głównego repozytorium GitHub: https://github.com/Tuxuy3Dom/Inzynieria\_Oprogramowania/tree/master

Repozytorium posiada dwie gałęzie:

1. Gałąź "master" zawiera laboratorium pierwsze oraz laboratorium związane z pracą Angulara.
2. Gałąź "LAB3" zawiera większość wykonanych prac laboratoryjnych. Link do tej gałęzi: https://github.com/Tuxuy3Dom/Inzynieria\_Oprogramowania/tree/LAB3

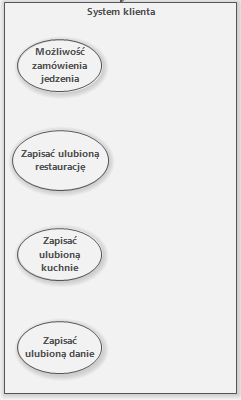
Podczas pierwszych zajęć skupiliśmy się na metodach budowy testów modułowych przy użyciu dedykowanych narzędzi do testowania jednostkowego. Proces ten obejmował inspekcję kodu źródłowego. Wykorzystaliśmy środowisko IDE: Visual Studio. Aby móc pracować w tym środowisku, zainstalowaliśmy .NET SDK, a także dodatkowe narzędzia takie jak nuget, XUnit i Bogus.

W kolejnych zajęciach laboratoryjnych zapoznaliśmy się z różnymi metodami tworzenia diagramów, takimi jak: diagram przypadków użycia, diagram klas, diagram stanów oraz diagram paymentprocessor. Wykorzystaliśmy do tego celu program o nazwie Software Ideas Modeler - narzędzie case do tworzenia diagramów, projektowania i analizy oprogramowania.

Dodatkowo, pracowaliśmy z frameworkiem ABP, który integruje się z frameworkiem Angular, oraz korzystaliśmy z bazy danych SQL Server.

1. **Diagram przypadków użycia**

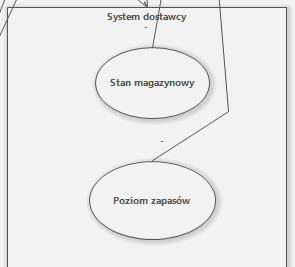
Pracę rozpoczęliśmy od tworzenia Systemu zarządzania zamówieniami, do którego dodałem moduły, w których przypisywałem funkcjonalności, które nasza aplikacja będzie zawierać. Utworzyłem cztery moduły dla czterech różnych ról: Moduł klienta, Moduł restauratora, Moduł dostawcy i Moduł sanepid.



Rysunek 1 System klienta

Źródło: Software Ideas Modeler program

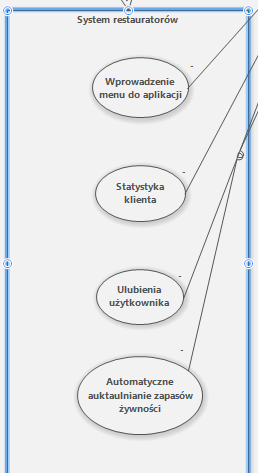
System klienta zawiera różne możliwości, takie jak: zamawianie jedzenia, złożenia zamówienia dla dostawy; dodawanie restauracji do ulubionych dla łatwego dostępu; zapisywanie ulubionej kuchni i dań.



Rysunek 2 System dostawcy

Źródło: Software Ideas Modeler program

System dostawcy oferuje dwie funkcje: stan magazynowy - informujący o ilości dostępnych produktów na stanie oraz poziom zapasów - informujący dostawcę, na przykład, gdy produkt jest całkowicie wyprzedany lub przekroczył minimalny poziom, co wskazuje na potrzebę dostawy konkretnego produktu.

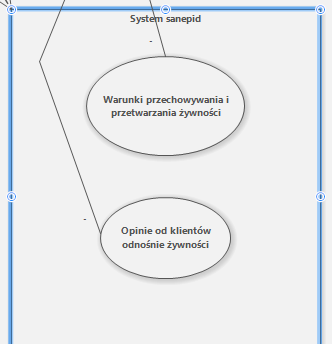


Rysunek 3 System restauratorów

Źródło: Software Ideas Modeler program

System restauratorów posiada następujące funkcje: wprowadzanie menu do aplikacji, wraz z możliwością modyfikacji wprowadzonego menu; statystyki klientów, które pokazują preferencje klientów w danym miejscu; polubienia, które wyświetlają wszystkie polubienia dokonane przez użytkowników w danej restauracji; automatyczne aktualizowanie zapasów żywności, które oblicza ilość zrealizowanych zamówień oraz pozostałą dostępną ilość na podstawie aktualnych danych dotyczących zapasów.

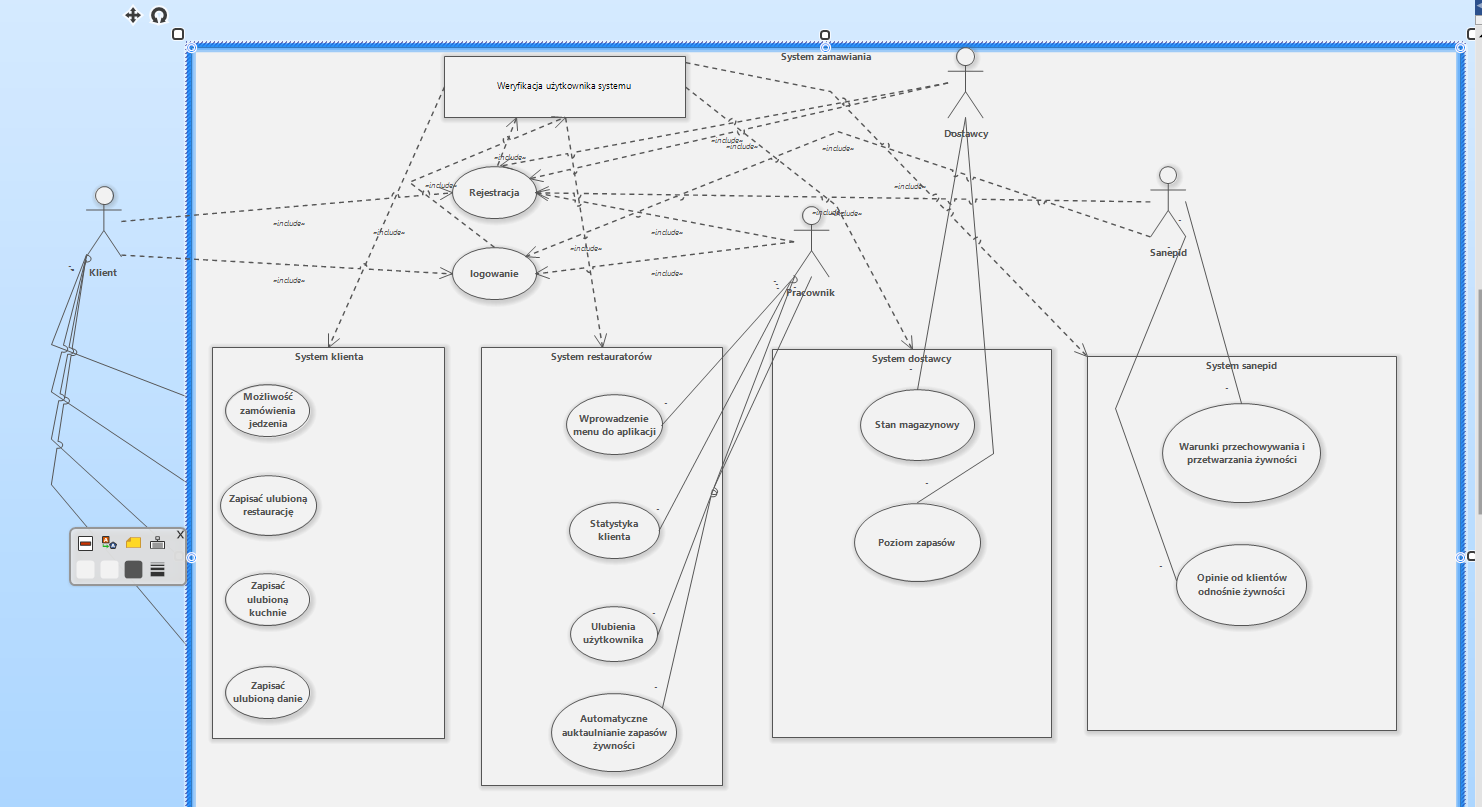
System, sanepidowi przekazuje informacje dotyczące warunków przechowywania i przetwarzania żywności w konkretnej restauracji; dostępne są również opcje otrzymywania zgłoszeń od klientów oraz opinii na temat żywności w lokalach.



Rysunek 4 System sanepidowy

Źródło: Software Ideas Modeler program

Przed uzyskaniem dostępu do wspomnianych wyżej systemów, każdy użytkownik rozpoczyna od procesu logowania do systemu. Podczas logowania użytkownik jest weryfikowany, aby określić, do jakiego systemu ma dostęp, a także przypisaną mu rolę użytkownika. W zależności od przypisanej roli, odpowiedni system zarządzania aplikacją jest wyświetlany.

Poniżej przedstawiamy cały system zarządzania zamówieniami.

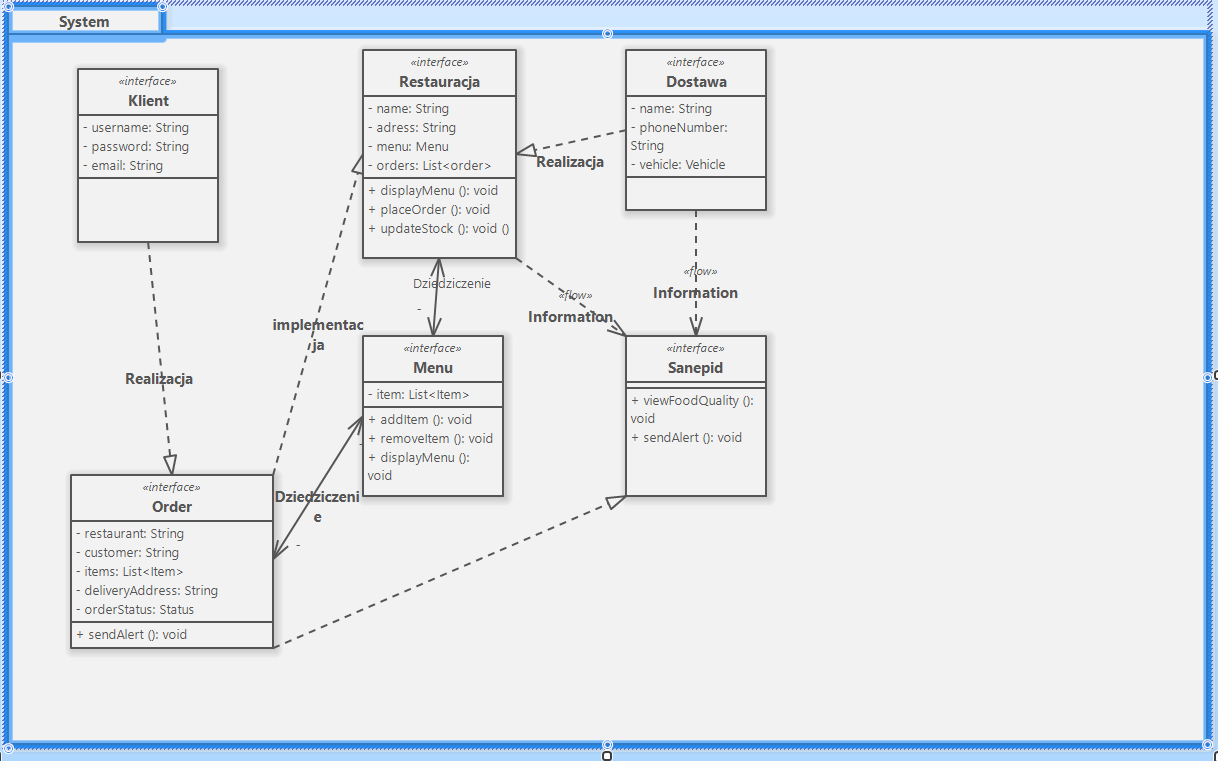
1. **Diagram klas**

Stworzyłem diagram klas, który zawiera sześć klas: Klient, Restauracja, Dostawa, Zamówienie, Menu i Sanepid. Klasy te są powiązane między sobą i posiadają odpowiednie relacje.

Klasa Klient umożliwia przekazanie zamówienia do realizacji, które trafia do klasy Restauracja, gdzie rozpoczyna się proces realizacji zamówienia.

Klasa Restauracja ma dostęp do menu i posiada możliwość dodawania, usuwania i wyświetlania poszczególnych elementów menu. Menu jest zsynchronizowane z zamówieniem dla danego zamówienia i konkretnej restauracji.

Klasa Dostawa otrzymuje informacje o zamówieniu i realizuje dostawę odpowiednim środkiem transportu. Dodatkowo, przekazuje informacje o sposobie przeprowadzonej dostawie do klasy Sanepid. Klasa Sanepid przyjmuje te informacje i podejmuje decyzje na ich podstawie. Ma również możliwość przyjmowania zgłoszeń od klientów, którzy mogą zgłaszać skargi dotyczące konkretnego zamówienia.

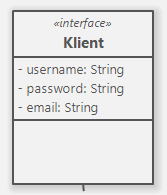


Rysunek 5 System klasowy ogólny

Źródło: Software Ideas Modeler program

W klasie Klient znajdują się następujące elementy:

* username: login użytkownika,
* password: hasło użytkownika do systemu,
* email: adres e-mail, na który będą przesyłane komunikaty dotyczące dostaw oraz innych informacji.

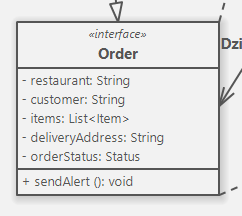


Rysunek 6 Klas Klient

Źródło: Software Ideas Modeler program

Klasa 'Order' zawiera informacje dotyczące zamówienia wraz z danymi restauracji, a także zawiera następujące elementy:

* restaurant: wybór restauracji
* customer: przypisanie zamówienia do klienta
* items: lista dań, które można zamówić,
* deliveryAddress: adres, na który ma być zrealizowana dostawa,
* orderStatus: informuje o aktualnym statusie zamówienia.
* sendAlert: odesłanie skargi do konkretnego zamówienia



Rysunek 7 Klas Order

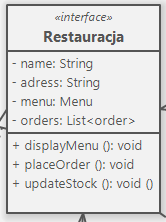
Źródło: Software Ideas Modeler program

Klasa 'Restauracja' zawiera następujące elementy:

* name: nazwa restauracji,
* address: adres restauracji,
* menu: lista potraw dostępnych w restauracji,
* orders: lista zamówień od klientów.

Klasa Restauracja zapewnia następujące funkcje:

* displayMenu: wyświetlanie karty dla restauratora,
* placeOrder: składanie zamówienia na dostawę produktów do restauracji (Dostawa),
* updateStock: aktualizowanie stanu zapasów w restauracji (Dostawa).

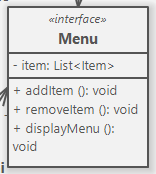


Rysunek 8 Klas Restauracja

Źródło: Software Ideas Modeler program

Klas „Menu” zawiera:

* item – stworzona lista potraw znajdujących się w restauracji,
* addItem: dodawać elementy do listy,
* removeItem: usuwać elementy z listy,
* displayMenu: wyświetlać całą listę.

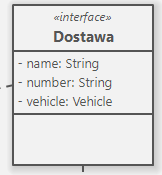


Rysunek 9 Klas Menu

Źródło: Software Ideas Modeler program

Klas „Dostawa” zawiera:

* name – nazwę restauracji,
* numer – numer zamówienia,
* vehicle – pojazd którym będzie zrealizowana dostawa.

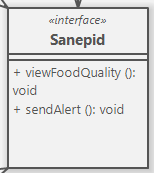


Rysunek 10 Klas Dostawa

Źródło: Software Ideas Modeler program

Klas „Sanepid” zawiera:

* viewFoodQuality – informację związaną z jakością żywności, która trafia do restauracji,
* sendAlert – dostęp do komunikacji z klientem, który informuję o naruszeniu norm sanepidu.

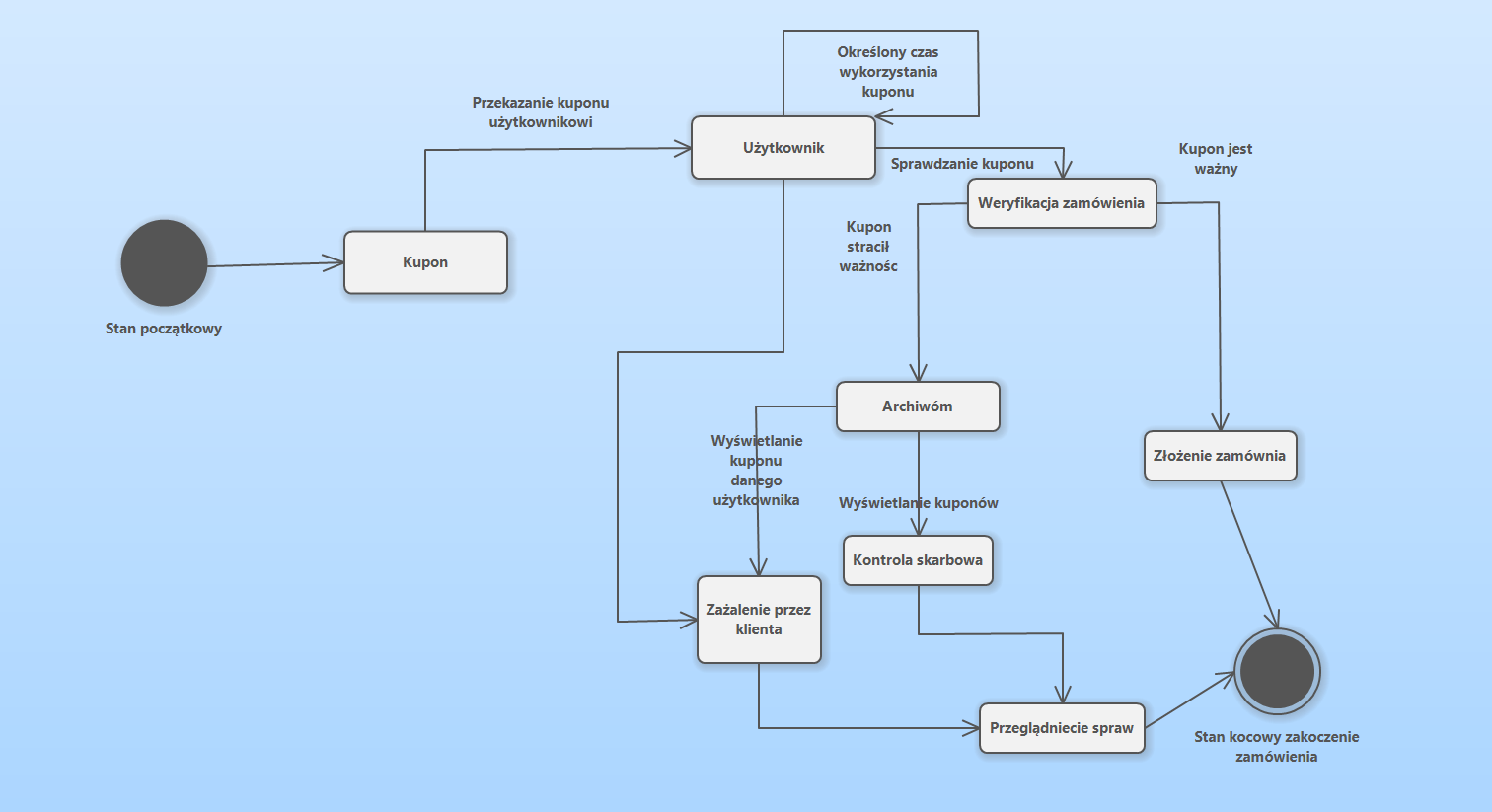


Rysunek 11 Klas Sanepid

Źródło: Software Ideas Modeler program

1. **Diagram stanów**

Za pomocą tego diagramu dodawaliśmy do naszej aplikacji funkcjonalność związaną z kuponami w branży gastronomicznej. Gdzie założenie było takie że kupon zostaw wystawiony przez restauratora i przekazany do użytkownika. Po czym użytkownik miał określoną ilość czasu do wykorzystania jego. Nasz serwis weryfikował ważność kuponu kiedy trafiał do weryfikacji zamówienia, jeżeli ważny przechodził jeżeli nie trafiał do archiwum na wypadek skargi lub kontroli.



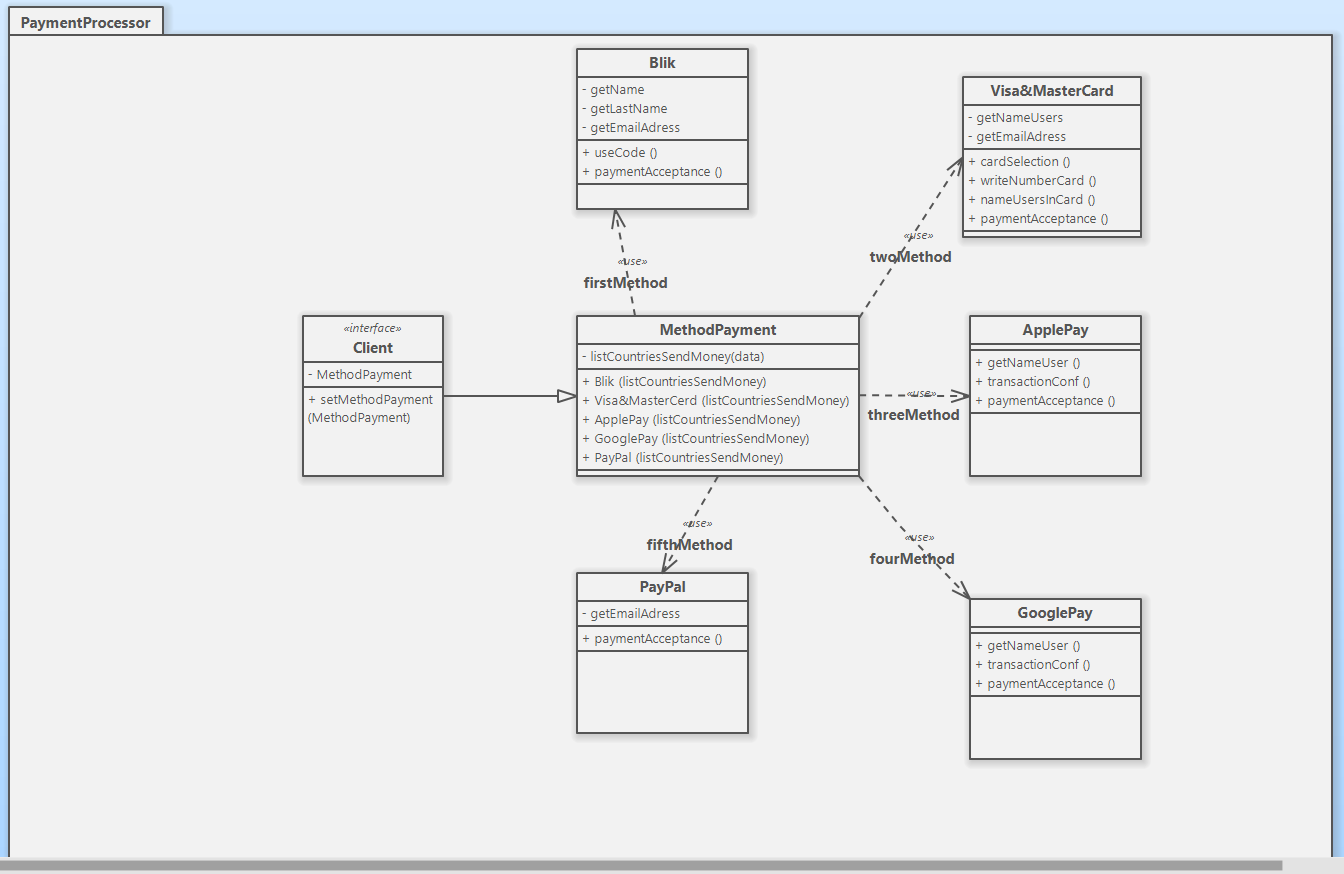
Rysunek 12 Ogólny diagram stanów

Źródło: Software Ideas Modeler program

Stan początkowy który otworzyłem jest kupon który został utworzony i został przekazany do użytkownika. Dalej zrobiłem pętle czasową która by sprawdzała ile jeszcze pozostało ważności kuponu, po tym założyłem że użytkownik już złożył zamówienie i chcę dodać kupon. W tym momencie sprawdzam ważność danego kuponu, jeżeli jest ważny to idzie złożenie zamówienia. Czyli poinformowanie że złożenie udało się, po czym dochodzimy do stanu końcowego. Jeśli okaże się że kupon stracił ważność to jest przerzucony do Archiwum, z którego jest sygnał do pokazania kuponu klientowi pod czas zażalenia i udowodnienie tego że klient nie dotrzymał się ustalonych jemu wymóg czasowych, lub w przypadku kontroli wyświetlenie że rzeczywiście był taki kupon, po czym jest przeglądnięcie sprawy i zakończenie czyli stan końcowy z zrealizowanym zamówieniem lub przerwanym przez brak ważnego kuponu.

1. **Paymentprocessor diagram**

Diagram PaymentProcessor zawiera pięć metod płatności, które są dostępne klientowi. Wybrałem takie metody płatności jak Blik, Visa&MasterCard, ApplePay, GooglePay, PayPal. Client ma metodę wyboru rodzaju płatności, gdzie odbyła się dziedziczenie z MethodPayment, główną klasą która zawiera listę metod płatności oraz dziedziczy z każdą metodą płatności.



Rysunek 13 Ogólny system metod płatności

Źródło: Software Ideas Modeler program

Metoda płatności "Blik" pobiera dane takie jak imię, nazwisko oraz pocztę od użytkownika. Następnie prosi o wprowadzenie kodu wygenerowanego w systemie bankowym i akceptację płatności.

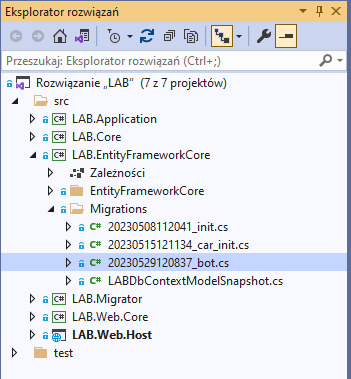
Metoda "Visa&MasterCard" pobiera login użytkownika oraz email. Użytkownik musi wybrać rodzaj karty (Visa lub MasterCard), podać numer karty bankowej oraz imię i nazwisko widoczne na karcie. Na końcu jest wymagane zaakceptowanie transakcji.

Metody "ApplePay" i "GooglePay" działają podobnie. Pobierają dane użytkownika, przetwarzają transakcję i wymagają akceptacji.

Metoda "PayPal" zakłada, że użytkownik ma już konto na tej platformie. Wystarczy kliknąć przycisk opłaty i zaakceptować transakcję. Następnie użytkownik otrzyma powiadomienie na swoją pocztę informujące o udanej transakcji.

1. **Migracje, encje api**

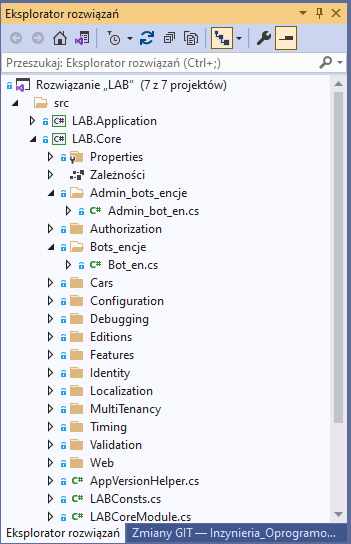
Tworzenie migracji robiliśmy w programie Visual Studio 2019. Oraz dodawaliśmy to do bazy danych Microsoft SQL Server Management Studio. Migracje można zrobić za pomocą narzędzia konsoli pakietu Entity Framework Core (EF Core). Aby rozpocząć projekt, konieczne jest ustawienie "LAB.Web.Host" jako projektu głównego, a następnie można uruchomić projekt za pomocą komendy "run".



Rysunek 14 Stworzony plik migracyjny

Źródło: Visual Studio 2019

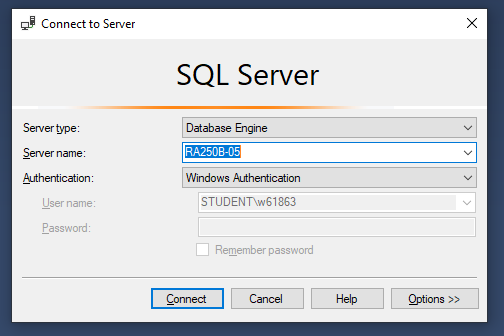
Encje w projekcie reprezentują strukturę danych, które są mapowane na tabele w bazie danych. Dodałem dwie encji do swojego projektu: Admin\_bots\_encje oraz Bots\_encje – gdzie Admin\_bots\_encje jest powiązana z encją Bots\_encje. Wiem że popełniłem błąd w notacji Encji miało być Camel Case a użyłem Snake Case, ponieważ pracuje z Pythonem a tam zazwyczaj stosuje się notacji Snake Case.



Rysunek 15 Dwie stworzone Encje

Źródło: Visual Studio 2019

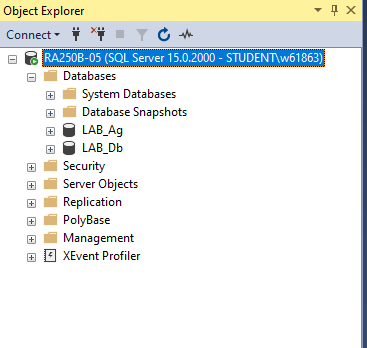
Po tym stworzyłem dla tych encji modele DTO oraz serwisy. Również została napisana metoda GetAll i dla jednej z niej była ukryta jej widoczność w API. Dla Sprawdzenie czy zostały poprawnie stworzone encje, logujemy się w SQL Server do naszej bazy danych.



Rysunek 16 Logowanie się do SQL Servera

Źródło: SQL Serwer Menedżer

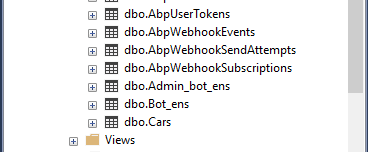
W mojej bazie danych stworzyłem bazę pod nazwą: LAB\_Ag



Rysunek 17 Podłączona baza danych

Źródło: SQL Serwer Menedżer

I do tej bazy do sekcji tablic robiłem migracje moich encji: pod nazwą dbo.Admin\_bot\_ens oraz dbo.Bot\_ens. Za pomocą konsoli menedżerów pakietów: w LAB.EntityFrameworkCore **–** za pomocą polecenia: add-migration i żeby było widoczne w naszej bazie to piszemy polecenie: update-database.



Rysunek 18 Stworzone encje w bazie danych

Źródło: SQL Serwer Menedżer

**Wniosek:**

W ramach dokumentacji przedstawiono informacje dotyczące projektu Inżynieria Oprogramowania, które obejmują repozytorium GitHub oraz przegląd przeprowadzonych działań podczas zajęć laboratoryjnych.

W projekcie stworzyłem repozytorium GitHub, które składa się z dwóch gałęzi. Gałąź "master" zawiera laboratorium pierwsze oraz laboratorium związane z pracą Angulara. Z kolei gałąź "LAB3" zawiera większość wykonanych prac laboratoryjnych.

W początkowych zajęciach skupiliśmy się na budowie testów modułowych przy użyciu dedykowanych narzędzi testowania jednostkowego. Proces ten obejmował inspekcję kodu źródłowego. Do pracy wykorzystaliśmy środowisko IDE: Visual Studio oraz zainstalowane narzędzia takie jak .NET SDK, nuget, XUnit i Bogus.

W dalszej części laboratoriów skoncentrowaliśmy się na tworzeniu różnych rodzajów diagramów, takich jak diagram przypadków użycia, diagram klas, diagram stanów oraz diagram paymentprocessor. Do tego celu wykorzystaliśmy narzędzie Software Ideas Modeler, które jest case narzędziem do tworzenia diagramów, projektowania i analizy oprogramowania.

Dodatkowo, podczas prac laboratoryjnych zetknęliśmy się z frameworkiem ABP, który integruje się z frameworkiem Angular, oraz korzystaliśmy z bazy danych SQL Server.

Dokumentacja zawiera podstawowe informacje, które pozwalają na zrozumienie projektu Inżynieria Oprogramowania oraz środowiska pracy, które zostało wykorzystane.