



PRACA DYPLOMOWA

INSTYTUT PODSTAW INFORMATYKI

POLSKA AKADEMIA NAUK

STUDIA PODYPLOMOWE

PROGRAMOWANIE NA PLATFORMIE .NET



ELEKTRONICZNA KSIĄŻKA SERWISOWA

Streszczenie

Celem pracy dyplomowej jest stworzenie aplikacji okienkowej pełniącej funkcję elektronicznej książki serwisowej do samochodu. Podstawowym zadaniem aplikacji jest umożliwienie zapisywania i przeglądania historii serwisowej samochodu użytkownika. Kluczowym walorem aplikacji jest prostota użytkowania. Aplikacja została dostarczona w technologii WPF przy wykorzystaniu wzorca projektowego MVVM.

Autor: Michał Niesłuchowski

email: m.niesluchowski@protonmail.com

telefon: 600 380 919

Promotor: dr inż. Grzegorz Glonek



OŚWIADCZENIE

Oświadczam, że złożoną pracę końcową pod tytułem:

ELEKTRONICZNA KSIĄŻKA SERWISOWA

Napisałem samodzielnie. Jednocześnie oświadczam, że praca ta:

1. nie narusza praw autorskich osób trzecich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. 2006 nr 90, poz. 631 z późn. zm.);
2. nie zawiera informacji i danych uzyskanych w sposób nielegalny, a w szczególności informacji zastrzeżonych przez inny podmiot albo stanowiących tajemnicę przedsiębiorstwa;
3. nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów/świadectw lub tytułów zawodowych wyższych uczelni. Nadto oświadczam, że treści zawarte w pisemnym egzemplarzu pracy oraz w egzemplarzu tej pracy w formie elektronicznej, złożonych przeze mnie, są jednobrzmiące.

Przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przeze mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy, lub ustalenia naukowego, właściwy organ stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 193 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym, tekst jednolity Dz.U. z 2012 r., poz. 572 z późn. zm.).

.....

data i podpis



Spis treści

I. Założenia i cele pracy	6
II. Analiza wymagań.....	8
Wprowadzenie	8
Przypadki użycia	8
III. Instrukcja użytkowania.....	10
Wprowadzenie	10
Uruchomienie aplikacji.....	10
Menu główne aplikacji	10
Formularz dodawania nowego auta	11
Główny widok auta.....	12
Formularz edycji danych auta	14
Formularz dodawania naprawy do auta	15
Formularz edycji naprawy auta	16
IV. Dokumentacja techniczna	18
Wprowadzenie	18
Zastosowane technologie	18
Ogólna architektura	19
Obsługa danych	19
Framework MVVM	19
Podejście do testowania	19
V. Dokumentacja frameworku MVVM	20
Dependency injection.....	20
Podłączanie ViewModeli	20
Nawigacja pomiędzy widokami.....	20
Wymiana danych między widokami	21
Komunikacja z użytkownikiem	21
Walidacja danych	21
RelayCommand	21
Przykładowe dane do projektowania widoku	21
VI. Podsumowanie.....	22



I. Założenia i cele pracy

Celem pracy jest stworzenie aplikacji służącej do prowadzenia elektronicznej książki serwisowej samochodu. Podstawowym zadaniem aplikacji jest umożliwienie użytkownikowi zapisywania i przeglądania historii serwisowej auta. Dzięki temu użytkownik ma możliwość sprawdzenia, jakie naprawy były już wykonywane w samochodzie i kiedy, co ma duże znaczenie podczas wizyt w serwisie i planowania kolejnych napraw. Dodatkowo użytkownik, może śledzić koszty utrzymania samochodu i bardziej świadomie podejmować decyzję, kiedy opłaca się wymienić auto na nowe albo przynajmniej nowsze.

Podobne rozwiązania są stosowane przez autoryzowane stacje serwisowe. Użytkownik może poprosić o wydruk z systemu i w ten sposób ma dostęp do historii serwisowej. W praktyce większość kierowców korzysta jednak z nieautoryzowanych serwisów, w związku z czym takie rozwiązanie nie jest dostępne i stąd zapotrzebowanie na aplikację, której dotyczy niniejsza praca dyplomowa.

Kluczowym walorem aplikacji ma być prostota użytkowania, tak by uniknąć szybkiego zniechęcenia użytkownika. W ocenie autora pracy akceptacja aplikacji przez użytkownika jest jedną z kluczowych kwestii przy każdym wdrożeniu nowego systemu informatycznego, dlatego zachowanie prostoty „Elektronicznej Książki Serwisowej” jest jednym z fundamentalnych założeń pracy.

Odbiorcą aplikacji jest osoba prywatna posiadająca prywatny samochód. Odbiorca jest zainteresowany kwestią serwisowania auta na poziomie średnim. Dla użytkownika o eksperckim podejściu do utrzymania auta, aplikacja będzie zbyt prosta, a dla użytkownika, o niskim zainteresowaniu tematyką serwisowania samochodu, aplikacja będzie wydawać się zbyt prosta. Samochód użytkownika docelowego ma 10 albo więcej lat. Samochody w wieku do 3 lat najczęściej serwisowane są autoryzowanych serwisach (ze względu na gwarancję producenta), a potem w okresie 4-10 roku użytkowania nie wymagają zbyt częstych napraw. Najczęściej po 10 latach użytkowania wizyty u mechanika stają się coraz częstsze oraz zasadne staje się pytanie „czy bardziej nie opłaca się wymienić auto na nowsze”, i od tego momentu aplikacja „Elektroniczna Książka Serwisowa” jest najbardziej przydatna.

Aplikacja zostanie dostarczona w formie aplikacji okienkowej (pełen klient) przy wykorzystaniu technologii WPF. Tym samym będzie ona mogła być uruchomiona na dowolnym komputerze z zainstalowanym systemem Windows. Aplikacja powstanie w nowoczesnym podejściu MVVM (Model, View, ViewModel). Implementacja tego podejścia zostanie przeprowadzona w oparciu o autorski framework stworzony przez autora pracy na potrzeby aplikacji „Elektroniczna Książka Serwisowa”. W zakresie dostępu do danych wykorzystany zostanie standardowy model repozytorium. Dane będą zapisywane i odczytywane w plikach XML przechowywanych lokalnie na lokalnym dysku twardym komputera użytkownika aplikacji. Wysoka jakość aplikacji zostanie zapewniona poprzez zastosowanie szczegółowych testów jednostkowych z naciskiem na klasy z obszaru „ViewModel”.



II. Analiza wymagań

Wprowadzenie

Analiza wymagań aplikacji została przeprowadzona w oparciu o przypadki użycia (user stories). Wyodrębnionych zostało osiem przypadków użycia. Ze względu na niski stopień skomplikowania aplikacji analiza wymagań składa się z przypadków użycia zapisanych jako jednozdaniowe sformułowania zgodne z podejściem „Jako <użytkownik> chcę <potrzeba>, żeby <cel do osiągnięcia>.”.

Przypadki użycia

Analiza wymagań zawiera następujące przypadki użycia:

- 001. Jako użytkownik potrzebuję dodać mój samochód do aplikacji, żeby mógł zacząć korzystać z aplikacji.
- 002. Jako użytkownik chcę zobaczyć listę napraw w moim samochodzie, żebym mógł zacząć korzystać z aplikacji.
- 003. Jako użytkownik chcę dodać naprawę do mojego samochodu, żebym mógł prowadzić historię serwisową samochodu.
- 004. Jako użytkownik chcę zmienić dane samochodu, ponieważ podczas tworzenia samochodu zrobiłem literówkę.
- 005. Jako użytkownik chcę mieć możliwość zapisu i odczytu danych samochodu z naprawami, żebym nie musiał zaczynać pracy od nowa za każdym razem, kiedy uruchamiam aplikację.
- 006. Jako użytkownik chcę zmienić dane naprawy, ponieważ podczas tworzenia naprawy zrobiłem literówkę.
- 007. Jako użytkownik chcę usunąć naprawę, ponieważ utworzyłem ją przez przypadek.
- 008. Jako użytkownik chcę, żeby program zapytał mnie o potwierdzenie, zanim przez nieuwagę przejdę do menu głównego i stracę wszystkie niezapisane dane samochodu.



III. Instrukcja użytkowania

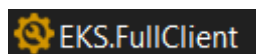
Wprowadzenie

Instrukcja użytkownika została napisana w oparciu o widoki aplikacji. Każdy widok (ekran) aplikacji został opisany w oddzielnym podrozdziale instrukcji. Opis zawiera wytłumaczenie wszystkich pól i przycisków. Aplikacja zawiera następujące widoki:

1. Menu główne
2. Formularz dodawania nowego auta
3. Główny widok auta
4. Formularz edycji danych auta
5. Formularz dodawania naprawy do auta
6. Formularz edycji danych naprawy

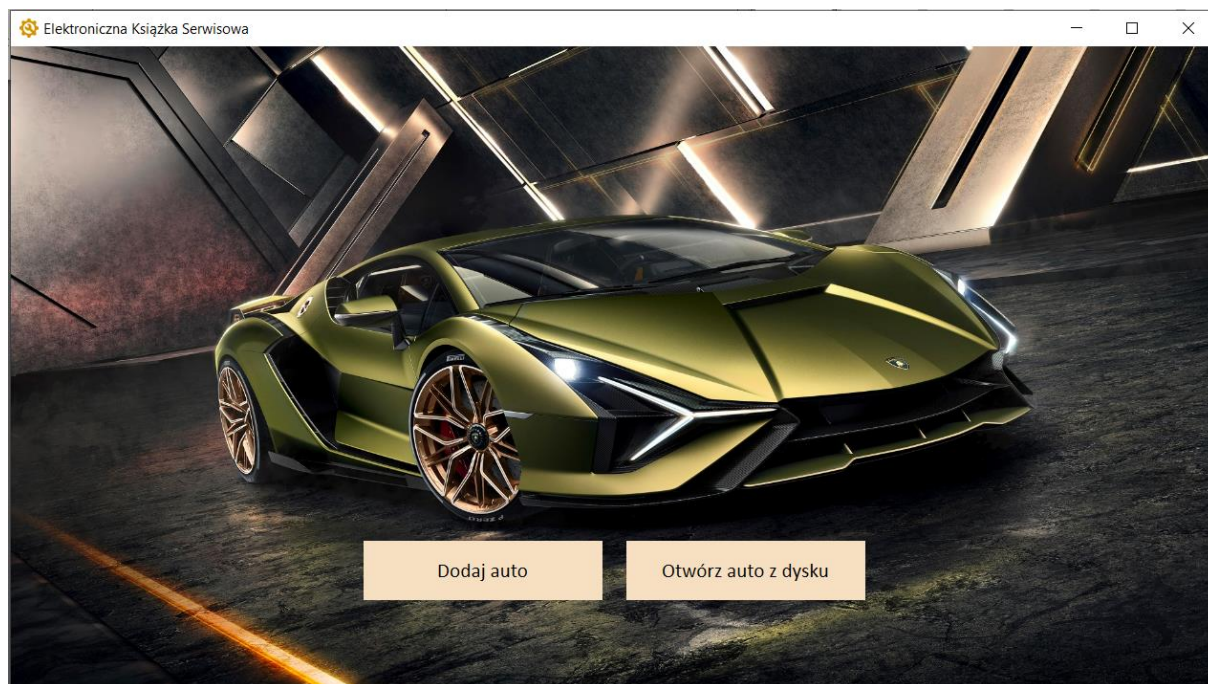
Uruchomienie aplikacji

Aplikacja nie wymaga instalacji. Folder z aplikacją należy przekopiować na dysk twardy użytkownika, a następnie uruchomić poprzez dwukrotnie kliknięcie na plik „EKS.FullClient” albo skrótu „Uruchom program”. Oba pliki są łatwo rozpoznawalne poprzez zastosowanie dedykowanej ikony zaprezentowanej poniżej



Po uruchomieniu aplikacji pojawia się menu główne aplikacji.

Menu główne aplikacji



Menu główne aplikacji jest podstawowym widokiem aplikacji umożliwiającym wykonywanie podstawowych zadań przez użytkownika. Dostępne są przede wszystkim dwie opcje –

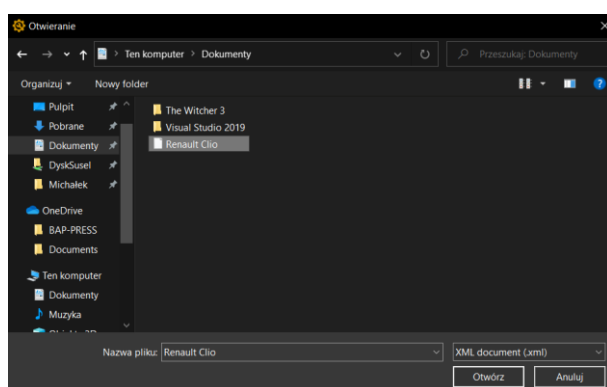
rozpoczęcie pracy z programem poprzez stworzenie nowego auta albo poprzez otworenie istniejącego auta z dysku.

Dodaj auto

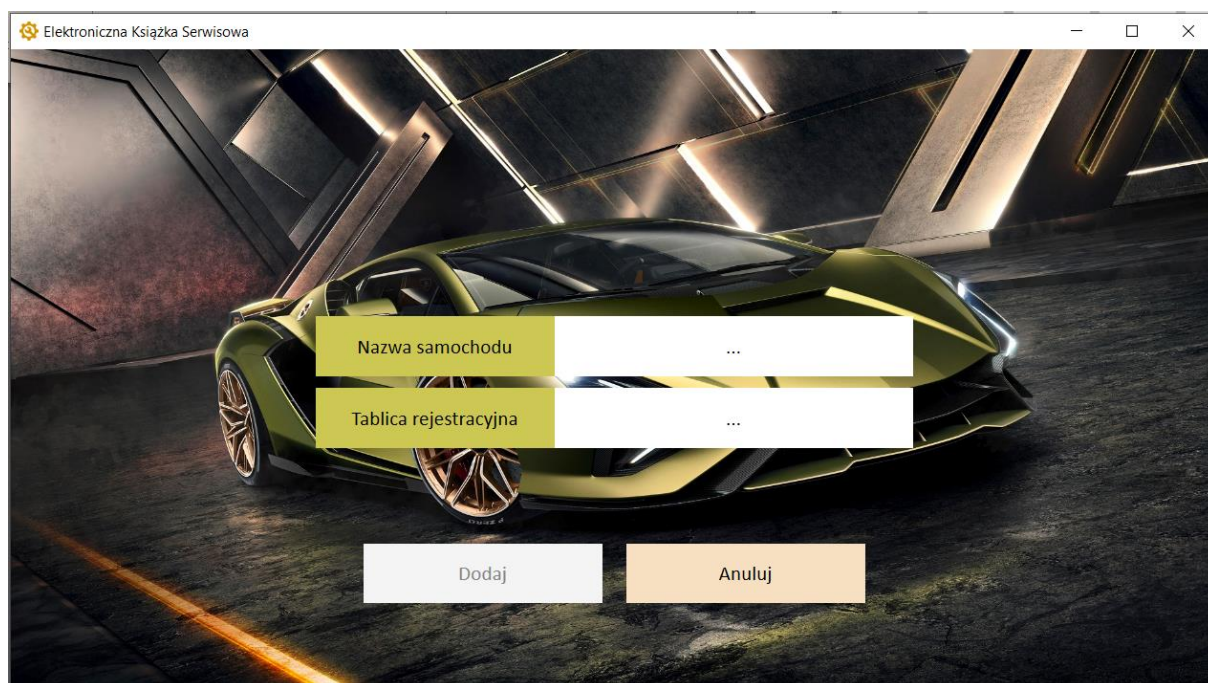
Przycisk „Dodaj auto” umożliwia stworzenie nowego auta w aplikacji. Po wybraniu przycisku wyświetla się formularz dodawania nowego auta. Skorzystanie z tego przycisku jest konieczne, jeżeli użytkownik nie posiada zapisanego na dysku innego auta.

Otwórz auto z dysku

Przycisk „Otwórz auto z dysku” umożliwia otworenie auta z dysku. Do skorzystania z tej opcji konieczne jest wcześniejsze zapisanie auta na dysku. Po wybraniu tego przycisku pojawia się okno dialogowe Windows, w którym należy wybrać plik z autem, co zostało zobrazowane poniżej.



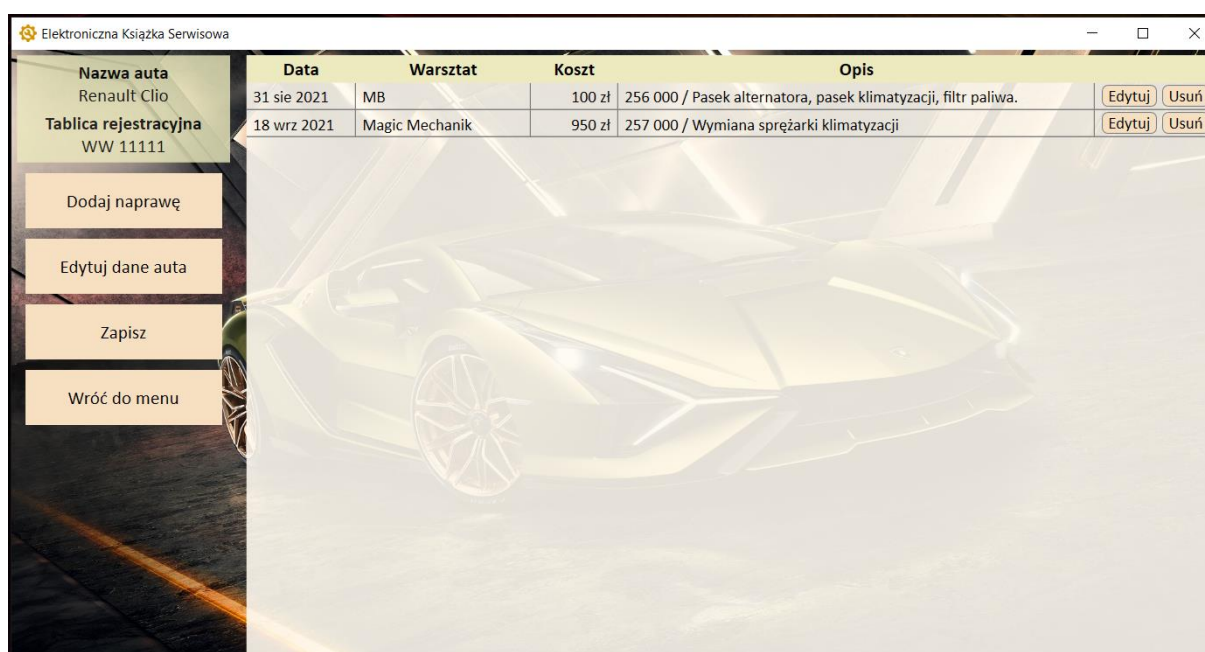
Formularz dodawania nowego auta



Formularz dodawania nowego auta umożliwia rozpoczęcie pracy z programem poprzez stworzenie nowego auta użytkownika. Przycisk „Dodaj” jest nieaktywny, dopóki użytkownik nie wprowadzi poprawnych danych auta.

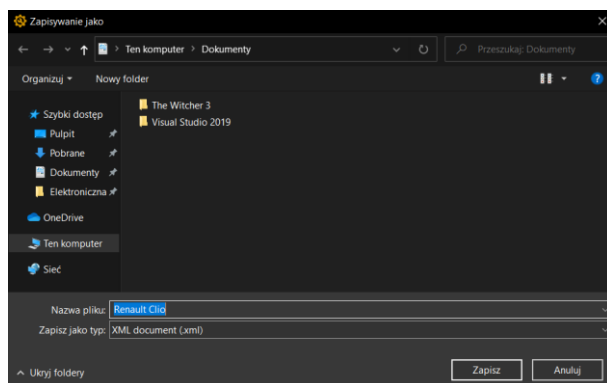
Nazwa samochodu	...	Pole do wprowadzenia nazwy samochodu. Należy wprowadzić tekst.
Tablica rejestracyjna	...	Pole do wprowadzenia numeru rejestracyjnego samochodu. Należy wprowadzić tekst.
Dodaj		Przycisk służący do stworzenia nowego auta do programu. Przycisk jest aktywny jedynie po wprowadzeniu nazwy i tablicy rejestracyjnej pojazdu. Po wybraniu przycisku auto zostaje utworzone i program przechodzi do głównego widoku auta.
Anuluj		Przycisk służący do anulowania operacji. Po wybraniu przycisku operacja dodania nowego auta jest anulowana i program powraca do menu głównego.

Główny widok auta

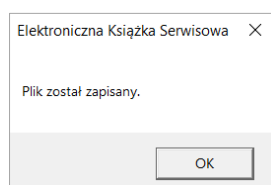


Główny widok aplikacji, w którym zaprezentowane są wszystkie naprawy wykonane w aucie oraz dostępne są wszystkie operacje dostępne do wykonania na aucie. Jest to podstawowe okno, w którym użytkownik spędza większość czasu podczas pracy z programem.

Dodaj naprawę	Umożliwia dodanie naprawy do auta. Wybranie przycisku otwiera formularz dodawania naprawy do auta.
Edytuj dane auta	Umożliwia zmianę danych auta tzn. nazwy i tablicy rejestracyjnej. Wybranie przycisku otwiera formularz edycji danych auta.
Zapisz	Umożliwia zapis danych auta na dysku. Po wybraniu tego przycisku pojawia się okno dialogowe Windows, w którym należy wybrać miejsce zapisu pliku oraz nazwę pliku, co zostało zobrazowane poniżej. Domyślna nazwa pliku to nazwa auta. Plik zapisywany jest w formacie XML.

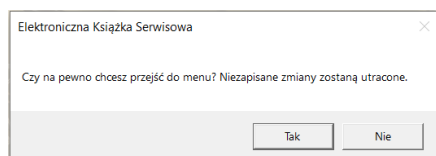


Po poprawnym zapisaniu danych na dysku aplikacja wyświetla komunikat, przedstawiony poniżej.



Wróć do menu

Umożliwia powrót do menu głównego aplikacji. Po wybraniu przycisku aplikacja ostrzega przed możliwą utratą danych i prosi o potwierdzenie przez użytkownika. W przypadku wybrania kliknięcia „Tak” aplikacja przechodzi do ekranu głównego. W przypadku kliknięcia przycisku „Nie” aplikacja pozostaje w głównym widoku auta i tym samym umożliwia zapisanie danych.



Data	Warsztat	Koszt	Opis		
31 sie 2021	MB	100 zł	256 000 / Pasek alternatora, pasek klimatyzacji, filtr p:	Edytuj	Usuń
18 wrz 2021	Magic Mechanik	950 zł	257 000 / Wymiana sprężarki klimatyzacji	Edytuj	Usuń

Główna tabelka przedstawiające wszystkie naprawy wykonane w aucie. Pierwsza kolumna zawiera datę wykonania naprawy, kolumna „Warsztat” to miejsce wykonania naprawy, „Koszt” to poniesiony koszt w polskich złotych, a „Opis” zawiera spis wykonanych napraw oraz inne informacje takie jak przebieg auta. Tabela może zawierać dowolną liczbę napraw. Każda naprawa zawiera się w jednym wierszu. W przypadku, gdy zawartość kolumny „Opis” nie jest w pełni widoczna należy zwiększyć okienko programu. Gdy cała tabela zostanie wypełniona naprawami, po prawej stronie tabeli pojawi się pasek do przewijania. Przy każdej naprawie dostępny jest przycisk „Edytuj” oraz przycisk „Usuń”. Przycisk „Edytuj” umożliwia edycję naprawy. Po jego wybraniu aplikacja przechodzi do formularza edycji naprawy auta. Przycisk „Usuń” umożliwia usunięcie naprawy. Po jego wybraniu aplikacja prosi użytkownika o potwierdzenie operacji, co przedstawia poniższy zrzut ekranu. Po potwierdzeniu (opcja „Tak”) naprawa jest usuwana i aplikacja powraca do głównego widoku auta. W przypadku

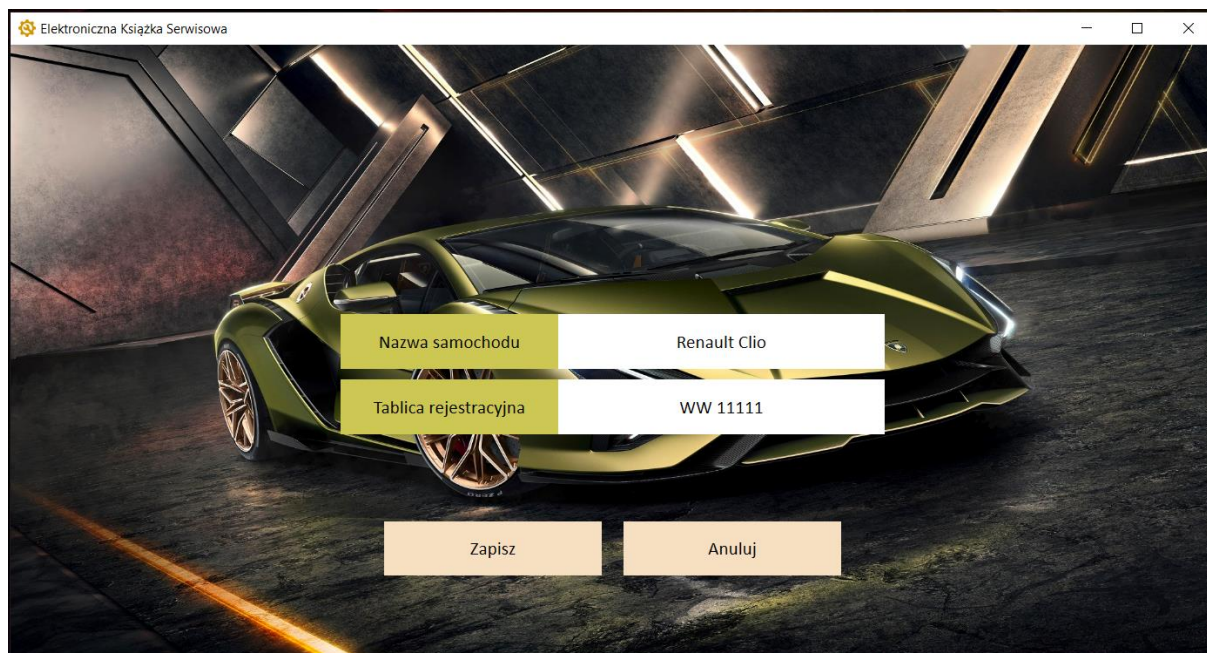
wybrania opcji „Nie” aplikacja powraca do głównego widoku auta, a naprawa nie jest usuwana.

Elektroniczna Książka Serwisowa ×

Czy chcesz usunąć naprawę?

Formularz edycji danych auta

Elektroniczna Książka Serwisowa



Nazwa samochodu

Renault Clio

Tablica rejestracyjna

WW 11111

Formularz służy do zmiany danych auta tzn. nazwy oraz tablicy rejestracyjnej samochodu. Przycisk „Zapisz” jest dostępny jedynie po wprowadzeniu poprawnych danych auta. W szczególności wyczyszczenie zawartości któregoś z pól powoduje dezaktywację przycisku.

Nazwa samochodu

Renault Clio

Pole do edycji nazwy samochodu. Należy wprowadzić tekst.

Tablica rejestracyjna

WW 11111

Pole do edycji numeru rejestracyjnego samochodu. Należy wprowadzić tekst.

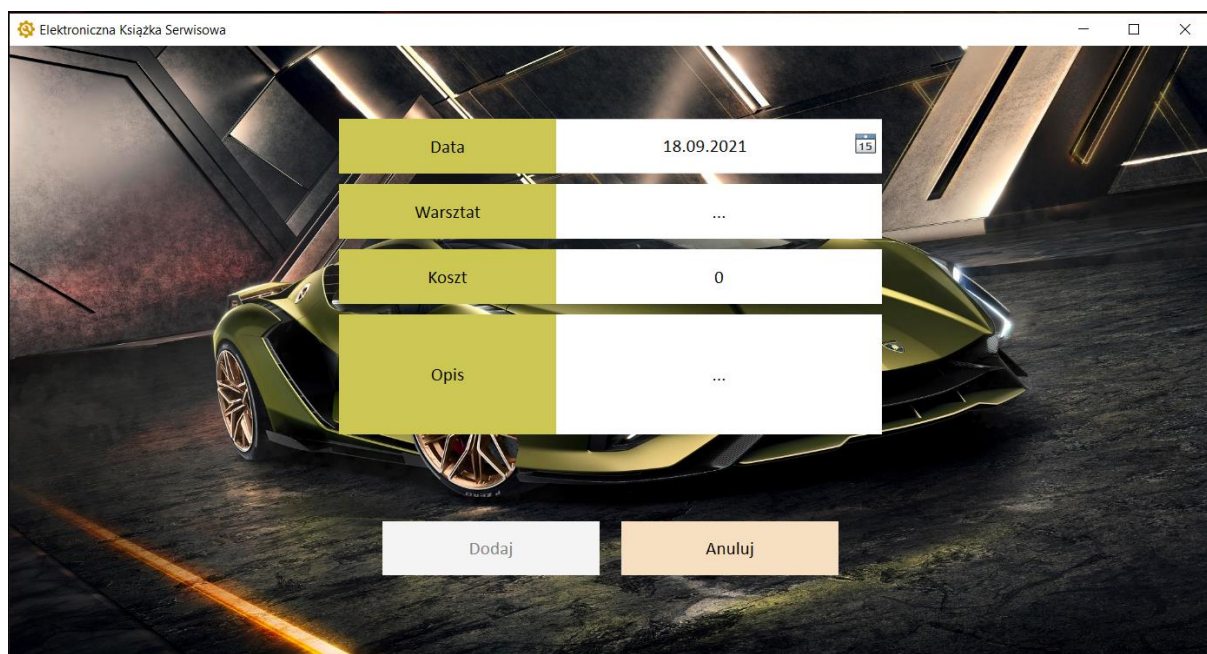
Zapisz

Przycisk służący do zapisania wprowadzonych zmian. Przycisk jest aktywny jedynie po wprowadzeniu poprawnej nazwy i tablicy rejestracyjnej pojazdu. Po wybraniu przycisku dane auta zostają zapisane i program powraca do głównego widoku auta.

Anuluj


Przycisk służący do anulowania operacji. Po wybraniu przycisku operacja zmiany danych auta jest anulowana i program powraca do głównego widoku auta.

Formularz dodawania naprawy do auta



Formularz służy do dodania nowej naprawy do auta. Przycisk „Dodaj” aktywuje się w momencie wprowadzenia poprawnie wszystkich danych na formularzu.

Data	18.09.2021
------	------------

Pole do wprowadzenia daty wykonania serwisu. Domyślnie program wpisuje bieżący dzień. Datę wybiera się w mini kalendarzu, który uruchamia się poprzez kliknięcie ikony kalendarza: .

Warsztat	...
----------	-----

Pole do wprowadzenia nazwy warsztatu, w którym została przeprowadzona naprawa. Należy wprowadzić tekst.

Koszt	0
-------	---

Pole do wprowadzenia kosztu naprawy. Należy wprowadzić liczbę. Program automatycznie sformatuje wartość na polskie złote. W przypadku wprowadzenia niepoprawnej wartości program wyświetli informację o błędzie: **Błąd**.

Opis	...
------	-----

Pole do wprowadzenia opisu naprawy. Opis powinien zawierać w szczególności opis wykonanych czynności w serwisie oraz przebieg samochodu w momencie naprawy. Należy wprowadzić tekst.

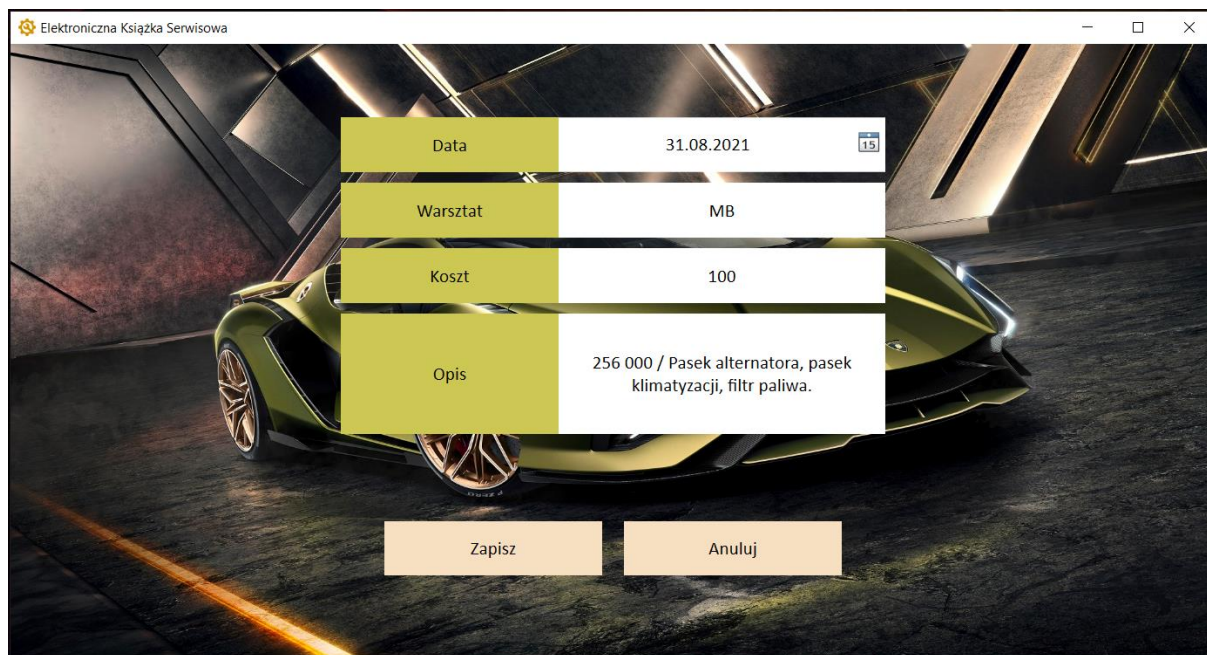
Dodaj

Przycisk służący do dodania nowej naprawy do auta. Przycisk jest aktywny jedynie po wprowadzeniu poprawnych danych w formularzu. Po wybraniu przycisku naprawa zostaje dodana do auta i program przechodzi do głównego widoku auta.

Anuluj


Przycisk służący do anulowania operacji. Po wybraniu przycisku operacja dodania nowej naprawy jest anulowana i program powraca do menu głównego.

Formularz edycji naprawy auta



Formularz służy do edycji istniejącej naprawy przypisanej do auta. Przycisk „Zapisz” jest aktywny tylko, jeżeli wszystkie dane w formularzu są poprawne.

Data	31.08.2021
------	------------

Pole do edycji daty wykonania serwisu. Datę wybiera się w mini kalendarzu, który uruchamia się poprzez kliknięcie ikony kalendarza: .

Warsztat	MB
----------	----

Pole do edycji nazwy warsztatu, w którym została przeprowadzona naprawa. Należy wprowadzić tekst.

Koszt	100
-------	-----

Pole do edycji kosztu naprawy. Należy wprowadzić liczbę. Program automatycznie sformatuje wartość na polskie złote. W przypadku wprowadzenia niepoprawnej wartości program wyświetli informację o błędzie : **Błąd**.

Opis	256 000 / Pasek alternatora, pasek klimatyzacji, filtr paliwa.
------	--

Pole do edycji opisu naprawy. Opis powinien zawierać w szczególności opis wykonanych czynności w serwisie oraz przebieg samochodu w momencie naprawy. Należy wprowadzić tekst.

Zapisz

Przycisk służący do zapisania zmiany w edytowanej naprawie. Przycisk jest aktywny jedynie, gdy wszystkie dane wprowadzone w formularzu są

poprawne. Po wybraniu przycisku zmiany w naprawie zostają zapisane i program przechodzi do głównego widoku auta.

Anuluj

Przycisk służący do anulowania operacji. Po wybraniu przycisku operacja edycji naprawy jest anulowana i program powraca do menu głównego.

IV. Dokumentacja techniczna

Wprowadzenie

Na potrzeby sporządzenia dokumentacji technicznej aplikację można podzielić na kilka kluczowych obszarów:

1. Zastosowane technologie
2. Ogólna architektura
3. Obsługa danych
4. Framework MVVM
5. Podejście do testowania

Każdy z wymienionych wyżej obszarów został szerzej opisany w dedykowanych podrozdziale.

Zastosowane technologie

Wykaz zastosowanych technologii wraz z przeznaczeniem oraz sposobem dostarczenia został zaprezentowany w tabeli poniżej. Wykaz zawiera jedynie kluczowe, w ocenie autora, technologie.

Nazwa	Wersja	Przeznaczenie	Dostarczenie
.NET Framework	4.7.2	Biblioteki bazowe	Wbudowana w VS
WPF	4.7.2	Front-end klienta okienkowego	Wbudowana w VS
Ninject	3.3.4	Obsługa dependency injection w implementacji frameworku MVVM	NuGet
MSTest	2.1.2	Framework do testów jednostkowych	Wbudowana w VS
Moq	4.16.1	Tworzenie atrap (mocków) obiektów na potrzeby testów jednostkowych	NuGet
Castle.Core	4.4.0	Biblioteka zależna wykorzystywana przez Moq	NuGet (automatycznie po instalacji Moq)
System.Xml.Serialization	-	Obsługa zapisu/odczytu danych dla plików XML.	Część .NET Framework

Na podstawie przedstawionej tabeli autor pracy przedstawił wniosek, że aplikacja w zdecydowanej większości opiera się na bazowych bibliotekach Microsoftu dostarczanych wraz z Visual Studio i tym samym jedynie w niewielkim stopniu opiera się na dodatkowych bibliotekach zewnętrznych. Wykorzystana została paczka Ninject, jako szybkie i sprawdzone rozwiązanie do zarządzania dependency injection oraz paczka Moq, która jest najbardziej

popularną biblioteką wykorzystywaną w testach jednostkowych do tworzenia atrap (mocków) klas.

Ogólna architektura

Solucja zawierająca aplikację została podzielona na trzy projekty:

1. BackEnd
2. FullClient
3. FullClient.Test

Projekt BackEnd składa się z dwóch głównych elementów – warstwy obsługi dostępu do danych oraz z modelu. Warstwa obsługi dostępu do danych (DAL) została oparta na wzorcu repozytorium. Model zawiera implementacje logiki biznesowej w podejściu domenowym.

Projekt FullClient składa się przede wszystkim z widoków aplikacji napisanych w XAML (folder Views), klas z kontekstem dla widoku (folder ViewModels) oraz frameworku do obsługi wzorca MVVM (folder Framework). Projekt FullClient dostarcza implementację aplikacji w postaci okienkowej.

Projekt FullClient.Test zawiera testy jednostkowe dla wszystkich klas typu ViewModel wykorzystanych w projekcie. Każdy ViewModel jest testowany w oddzielnej dedykowanej klasie.

Obsługa danych

W projekcie wszystkie dane są zapisywane i odczytywane przy wykorzystaniu plików XML. Zapis danych odbywa się poprzez serializację klas z modelu (domen) do pliku XML, a odczyt analogicznie tzn. poprzez deserializację z pliku XML do klas modelowych.

Framework MVVM

Na potrzeby realizacji projektu autor pracy napisał własny framework do obsługi wzorca MVVM. Szczegółowa dokumentacja frameworku znajduje się w oddzielnym rozdziale.

Podejście do testowania

Testowanie projektu zostało oparte na testach jednostkowych. Testy jednostkowe zostały przygotowane dla wszystkich klas stosowanych jako ViewModel. Celem testów było zapewnienie wysokiej jakości klienta okienkowego aplikacji. Założeniem do testowania było szerokie pokrycie wszystkich elementów klasy (konstruktory, komendy, metody, właściwości) przy jednoczesnej koncentracji na podstawowych sytuacjach (nieuwzględnienie corner case'ów).

V. Dokumentacja frameworku MVVM

Na potrzeby projektu autor pracy stworzył autorski framework do obsługi wzorca MVVM. Framework można podzielić na kilka elementów:

- Dependency injection
- Podłączanie ViewModeli
- Nawigacja pomiędzy widokami
- Wymiana danych między widokami
- Komunikacja z użytkownikiem
- Walidacja danych
- RelayCommand
- Przykładowane dane do projektowania widoku

Dependency injection

Fundamentem frameworku jest implementacja dependency injection. Za implementację odpowiadają klasy umieszczone w folderze loc: klasa loc.Configuration zawiera konfigurację wszystkich klas wykorzystywanych do dependency injection, a klasa locKernel jest wykonawcą iniekcji. Dependency injection jest inicjalizowane w kodzie startowym aplikacji (App.xaml.cs).

Podłączanie ViewModeli

Wszystkie ViewModele są dostarczone do klas typu View za pośrednictwem klasy agregującej ViewModelLocator. Klasa ViewModelLocator udostępnia poszczególne ViewModele jako właściwości. Od drugiej strony można powiedzieć, że wszystkie widoki aplikacji są podpięte pod ViewModelLocator, z którego „wybierają” odpowiedni ViewModel jako właściwość. ViewModele są dostarczone poprzez Dependency Injection w formie „Transient” (za każdym razem wstrzykiwana jest nowa instancja).

Nawigacja pomiędzy widokami

Nawigacja między widokami została zaimplementowana jako mikroservis w folderze Navigation. Nawigacja odbywa się przy wykorzystaniu eventów. Główne okno aplikacji (Main Window) wyświetla poszczególne widoki jako elementy User Control. Przy wykorzystaniu serwisu do nawigacji każdy ViewModel może wywołać event zmiany widoku (User Control), co powoduje zmianę User Control wyświetlanego przez główne okno. Spójność i łatwość wywołań poszczególnych User Control jest zapewniona przez zastosowanie klasy wyliczeniowej ControlsRegister, która zawiera listę wszystkich widoków aplikacji. Serwis jest dostarczany do ViewModeli w konstruktorze jako dependency injection w formie singleton.

Wymiana danych między widokami

Wymiana danych między widokami została zaimplementowana jako mikroservis w folderze TempData. Serwis umożliwia wczytywanie, zapisywanie i kasowanie danych przechowywanych jako klasy modelu. Serwis jest dostarczany do ViewModeli w konstruktorze jako dependency injection w formie singleton.

Komunikacja z użytkownikiem

Komunikacja z użytkownikiem została zaimplementowana jako mikroservis w folderze UserDialog. Serwis zawiera obsługę komunikacji z użytkownikiem poprzez MessageBoxy oraz FileDialogi. Serwis jest dostarczany do ViewModeli w konstruktorze jako dependency injection w formie singleton.

Walidacja danych

Walidacja danych jest implementowana w kodzie XAML jako walidacja przy bindingu (`<Binding.ValidationRules>`). Framework zawiera klasy walidacyjne wykorzystywane w tym podejściu. Klasy znajdują się w folderze Validations i implementują interfejs ValidationRule.

RelayCommand

Standardowa implementacja klasy rekomendowana przez MVVM Foundation. Klasa wykorzystywana do podpinania metod do właściwości typu Command we ViewModelu.

Przykładowe dane do projektowania widoku

Dane na potrzeby projektowania w XAML są podpinane jako DataContext dla Design Instance (komenda `d:DataContext="{d:DesignInstance ViewModel, IsDesignTimeCreatable=True}"`). W takim przypadku wykorzystywany jest bezparametrowy konstruktor ViewModelu. W bezparametrowym konstruktorze wykorzystywana jest klasa DesignDataService (folder DesignData) zapewniająca dane „z kodu” i wypełniającą właściwości ViewModelu przykładowymi wartościami.

VI. Podsumowanie

W ocenie autora wytworzona aplikacja całkowicie spełnia przyjęte założenia i cele. Aplikacja umożliwia użytkownikowi zapisywanie i przeglądania historii serwisowej auta. Ocena prostoty użytkowania została przeprowadzona poprzez testy z użytkownikami z najbliższego otoczenia autora. Testerzy zgodnie przyznali, że aplikacja jest bardzo prosta w obsłudze, a korzystanie z niej zapewnia znacznie lepsze doświadczenia niż np. prowadzenie zapisów w pliku typu Excel albo edytor tekstu. Atrakcyjny wygląd i prostota programu zachęcają do korzystania i prowadzenia historii serwisowej auta.

Od strony technicznej autor również ocenia prace pozytywnie. Obsługa danych przy pomocy XML pozwoliła na uniknięcie konieczności budowy bazy danych. Dzięki temu nie jest konieczne ustanawianie serwera w architekturze klient-serwer albo innego zbliżonego rozwiązania. Użytkownik może uruchomić aplikację bez instalacji i wystarczy przegranie plików na niemalże dowolny komputer z zainstalowanym Windowsem 10.

Aplikacja nie jest skomplikowana od strony technologicznej. Projekt w znikomym stopniu korzysta z bibliotek zewnętrznych, co w opinii autora należy uznać za zaletę. Zastosowany autorski framework obsługi MVVM jest lekki i prosty w obsłudze. Wykorzystanie dependency injection pozwoliło na uniknięcie anty-wzorca singleton. Wszystkie zależności klas są jednoznacznie wstrzykiwane w konstruktorze, co zwiększa przejrzystość i umożliwia testy jednostkowe. Sama implementacja obsługi dependency injection została przeprowadzana w oparciu o wzorzec singleton, jednak w ocenie autora jest to akceptowalne ze względu na prostotę frameworku. Dodatkowo implementacja dependency injection jest jawnie wywoływana w kodzie aplikacji (App.xaml.cs), więc nie ma mowy o ukrywaniu zależności projektu.

Zastosowanie testów jednostkowych pozwoliło na dostarczenie projektu w wysokiej jakości. Podczas testów użytkowników nie zostały wykryte żadne błędy.

Perspektywy dalszego rozwoju aplikacji są w ocenie autora duże. Do najciekawszych pomysłów na nowe funkcjonalności należą przygotowanie raportu z historii serwisowej oraz przygotowanie podsumowania kosztów utrzymania auta. Implementacja tych funkcjonalności nie powinna nastręczać trudności pod kątem zmian w interfejsie użytkownika oraz nie powinna negatywnie wpłynąć na prostotę i intuicyjność z użytkowania programu, a z drugiej strony wyraźnie podniosłaby atrakcyjność programu.