



**Politechnika
Śląska**

Dokumentacja projektowa

Programowanie Obiektowe i Graficzne

*Magazyn Liczników i modułów samochodowych -
Implementacja Aplikacji Desktopowej C# WPF z
Wykorzystaniem MVVM, Prism i SQLite*

Kierunek: Informatyka

Członkowie zespołu:

Mateusz Zaskórski

Jakub Kwaśniewski

Gliwice, 2024/2025

Spis treści

1	Wprowadzenie	2
1.1	Role w projekcie	2
1.2	Cel i Zakres Dokumentacji	2
1.3	Przegląd Technologii: C# WPF, MVVM, Prism, SQLite	2
1.4	Wykorzystane Pakiety	4
2	Start aplikacji	5
2.1	Panel Logowania	5
3	Główny widok aplikacji	6
3.1	Menu Główne	6
3.2	Logi użytkowników	8
3.3	Logi urządzeń	9
3.4	Historia napraw	10
3.5	Urządzenia	11
3.6	Konto	16
3.7	Wyloguj	16
4	Wnioski	18
5	Bibliografia	20

1 Wprowadzenie

1.1 Role w projekcie

- **Jakub Kwaśniewski** - pomysłodawca projektu, twórca dokumentacji, osoba odpowiedzialna za rozwój aplikacji po stronie back-endu, implementacja bazy danych do projektu i utworzenie rekordów w samej bazie, szyfrowanie haseł, twórca customowych okien informacyjnych;
- **Mateusz Zaskórski** - pomysłodawca stosu technologicznego użytego w projekcie, współtwórca dokumentacji, osoba odpowiedzialna za rozwój aplikacji po stronie front-endu (UI/UX), zaprojektowanie bazy danych i utworzenie kwerend wykorzystywanych w aplikacji;

1.2 Cel i Zakres Dokumentacji

Od początku planem było stworzenie aplikacji, która zostanie wykorzystana w naszej codziennej pracy. Pracujemy w jednej firmie, więc sam projekt, jak i komunikacja odnośnie projektu wydały nam się słuszne.

Naszym celem było stworzenie uniwersalnej aplikacji do obsługi stanów magazynowych w naszej firmie. Aktualnie jakiegokolwiek dane odnośnie stanów magazynowych były przechowywane w Excelu i bardzo rzadko aktualizowane. Dlatego postanowiliśmy uprościć proces przyjmowania, wydawania urządzeń oraz zapanować nad chaosem ich dodawania i usuwania w firmie. Programując, lubimy rozwiązywać realne problemy, stąd właśnie taki pomysł.

Ta dokumentacja powstała po to, żeby w prosty sposób pokazać, jak aplikacja została zbudowana i jak korzystać z programu.

1.3 Przegląd Technologii: C# WPF, MVVM, Prism, SQLite

Poniżej omówione zostaną cztery kluczowe technologie, które wspólnie tworzą spójny i efektywny stos technologiczny dla naszej aplikacji desktopowej:

- **C# WPF (Windows Presentation Foundation):** Jest to potężny framework firmy Microsoft, służący do tworzenia bogatych i interaktywnych interfejsów użytkownika w aplikacjach desktopowych dla platformy .NET. WPF oferuje elastyczne możliwości graficzne, zaawansowane mechanizmy powiązania danych (Data Binding) i stylizacji, co pozwala na budowanie wizualnie atrakcyjnych i responsywnych aplikacji.










- **MVVM (Model-View-ViewModel):** To wzorzec architektoniczny, który promuje czystą separację odpowiedzialności między warstwą danych (Model), logiką prezentacji (ViewModel) a interfejsem użytkownika (View). Stosowanie MVVM znacząco ułatwia rozwój, testowanie i utrzymanie aplikacji WPF, minimalizując kod-behind i zwiększając modularność.
- **Prism Framework:** Jest to framework do budowania modułowych, kompozycyjnych aplikacji WPF. Prism wspiera skalowalność i elastyczność w dużych projektach, umożliwiając dzielenie aplikacji na niezależne, luźno sprzężone komponenty (moduły), które mogą być rozwijane i wdrażane niezależnie.
- **SQLite Database:** To lekka, wbudowana baza danych, która jest idealna dla aplikacji desktopowych, które nie wymagają zewnętrznego serwera bazodanowego ani złożonej infrastruktury. SQLite przechowuje dane w pojedynczym pliku na dysku, co upraszcza wdrożenie i zarządzanie danymi.

Te technologie naprawdę dobrze ze sobą współpracują. WPF daje dużo możliwości graficznych i pozwala łatwo powiązać dane z interfejsem. MVVM korzysta z tych funkcji, żeby oddzielić logikę od wyglądu, co ułatwia testowanie i późniejszą rozbudowę aplikacji. Prism idzie o krok dalej – pozwala podzielić aplikację na moduły, które można rozwijać osobno. To ważne, gdy budujemy coś większego niż tylko prosty formularz. Od tego momentu, wszędzie gdzie wspomnimy o regionie, będziemy mieli na myśli wykorzystanie regionów, które oferuje Prism.

Do tego dochodzi SQLite – lekka, wbudowana baza danych, która świetnie pasuje do aplikacji desktopowych, bo nie trzeba ustawiać żadnego serwera. Wszystko razem tworzy solidny, nowoczesny zestaw narzędzi, który dobrze sprawdza się w praktyce i jest łatwy do utrzymania.

Dlatego taki zestaw technologii wydał nam się słuszny – pokazuje, jak budować aplikacje zgodnie z dobrymi zasadami.

1.4 Wykorzystane Pakiety

 Dapper <small>prince Sam Saffron, Marc Gravell, Nick Craver</small> A high performance Micro-ORM supporting SQL Server, MySQL, SQLite, SigCL, Firebird etc. Major Sponsor: Dapper Plus from ZZZ Projects.	2,166
 Dryloc.dll <small>prince Matsum Yokuu</small> Dryloc is fast, small, Full featured x64 Container for .NET	4,77 543
 Microsoft.Bcl.AsyncInterfaces <small>prince Microsoft</small> Provides the <code>IAsyncEnumerable<T></code> and <code>IAsyncDisposable</code> interfaces and helper types for .NET Standard 2.0. This package is not required starting with .NET Standard 2.1 and .NET Core 3.0.	9,01 9,05
 Microsoft.Xaml.Behaviors.Wpf <small>prince Microsoft</small> Easily and intuitively to your app using XAML Behaviors for WPF. Behaviors encapsulate reusable functionalities for elements that can be easily added to your XAML without the need for more imperative code.	1,131 1,135
 Prism.Core <small>prince Brian Lagunas, Dan Siegel</small> Prism is a fully open source version of the Prism guidance originally produced by Microsoft Patterns & Practices. Prism provides an implementation of a collection of design patterns that are helpful in writing well structured and maintainable XAML applications, including MVVM, dependency injection, commanding, event aggregation, and more. Prism's core functionality is a shared code base in a Portable Class Library targeting these platforms: WPF, and Xamarin Forms. Features that need to be platform specific are implemented in the respective libraries for the target platform.	8,197 9,0537
 Prism.DryIoc <small>prince Brian Lagunas, Dan Siegel</small> Use these extensions to build Prism applications based on DryIoc.	8,197 9,0537
 Prism.Wpf <small>prince Brian Lagunas, Dan Siegel</small> Prism is a fully open source version of the Prism guidance originally produced by Microsoft Patterns & Practices. Prism provides an implementation of a collection of design patterns that are helpful in writing well structured, maintainable, and testable XAML applications, including MVVM, dependency injection, commanding, event aggregation, and more. Prism's core functionality is a shared library targeting the .NET Framework and .NET Standard. Features that need to be platform specific are implemented in the respective libraries for the target platform (WPF, Uno Platform, and Xamarin Forms).	8,197 9,0537
 Stub.System.Data.SQLite.Core.NetFramework <small>prince SQLite Development Team</small> The official SQLite database engine for both x86 and x64 along with the ADO.NET provider.	10,119
 System.Data.SQLite.Core <small>prince SQLite Development Team</small> The official SQLite database engine for both x86 and x64 along with the ADO.NET provider.	10,119

Rysunek 1: Widok Login Page

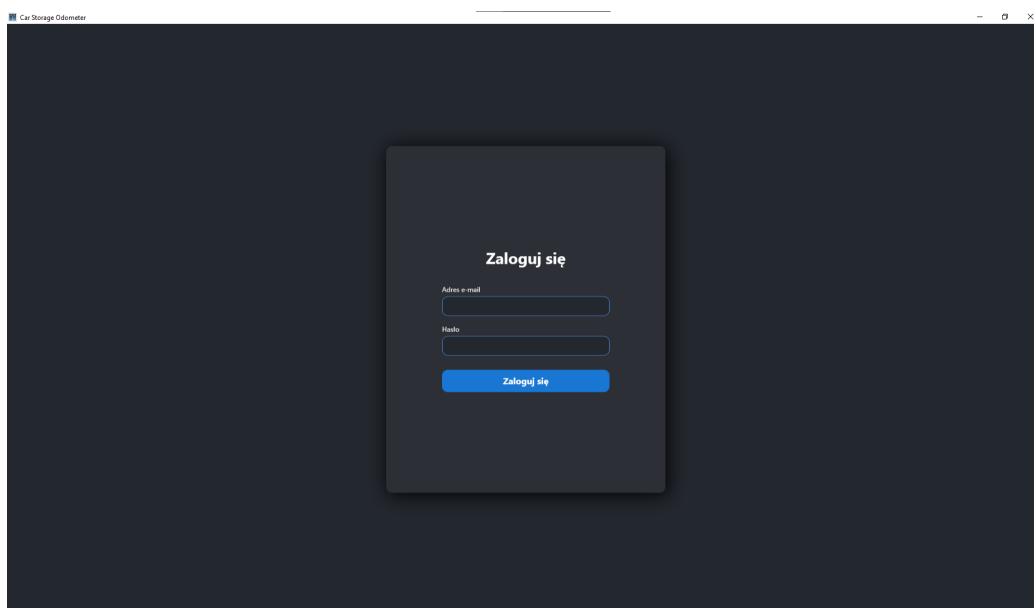
2 Start aplikacji

2.1 Panel Logowania

Dane do logowania na użytkownika testowego

- **Login/Email** - test@test.pl
- **Hasło** - testowehaslo

Po każdorazowym uruchomieniu aplikacji, powita nas Panel Logowania.



Rysunek 2: Podgląd Managera Pakietów NuGet

Na tym etapie aplikacji, nie przewidzieliśmy rejestracji użytkowników. Wychodząc z założenia, że aplikacja w przyszłości będzie wykorzystywana w naszym zakładzie pracy liczącym około 20 osób, założenie konta pracownikom korzystającym z naszego programu nie będzie problemem.

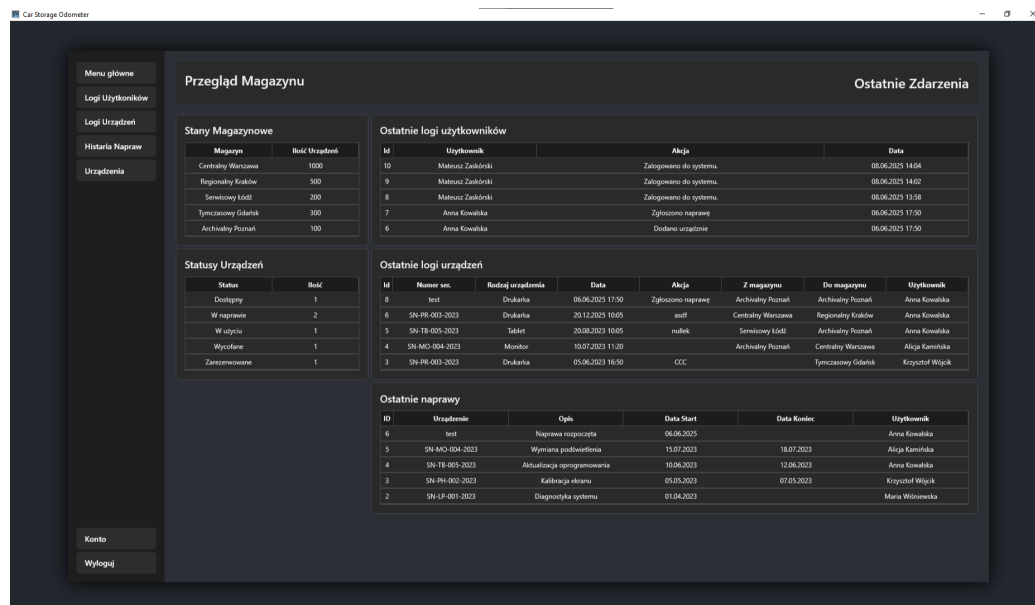
W tym widoku, aby przejść dalej, program wymaga od nas zalogowania się.

Adres e-mail oraz hasło w zahashowanej postaci **algorytmem SHA-256** przechowywane jest w bazie danych. Po podaniu poprawnych danych i naciśnięciu przycisku "Zaloguj się", program sprawdza wpisy w bazie i jeśli będą zgodne, nastąpi zamknięcie okna logowania i przejście do głównego widoku.

3 Główny widok aplikacji

3.1 Menu Główne

Po poprawnym zalogowaniu się do aplikacji, pojawi nam się główny widok aplikacji.



Rysunek 3: Region DashboardView

Po lewej stronie jest SideBar. Umożliwia on zmianę regionów w naszej aplikacji. Każdy z regionów zostanie omówiony **w osobnej podsekcji**.

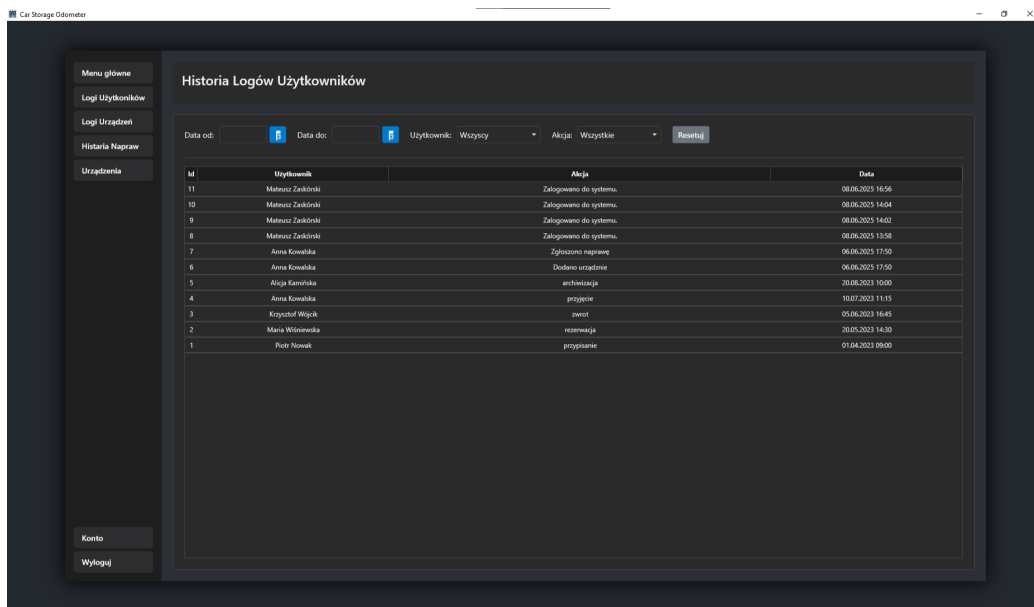
W regionie "Menu główne" zadaliśmy, aby ten pełnił rolę zbiorczych informacji. Stąd takie tabele jak:

- Stany magazynowe
- Statusy urządzeń
- Ostatnie logi użytkowników
- Ostatnie logi urządzeń
- Ostatnie naprawy

Użytkownik w ten sposób widzi, co w ostatnim czasie działo się w naszej aplikacji. Ostatnie logi użytkowników, urządzeń oraz ostatnie naprawy wyświetlają ostatnie 5 najnowszych wpisów w bazie. Stany magazynowe wyświetlają liczbę urządzeń (w naszym przypadku, zostało to zaimplementowane, aby zbudować trochę większą bazę danych pod inny przedmiot), a statusy urządzeń zliczają liczbę konkretnych statusów.

3.2 Logi użytkowników

Logi użytkowników to region, który wyświetla listę wszystkich logów w obrębie działań osób korzystających z aplikacji.



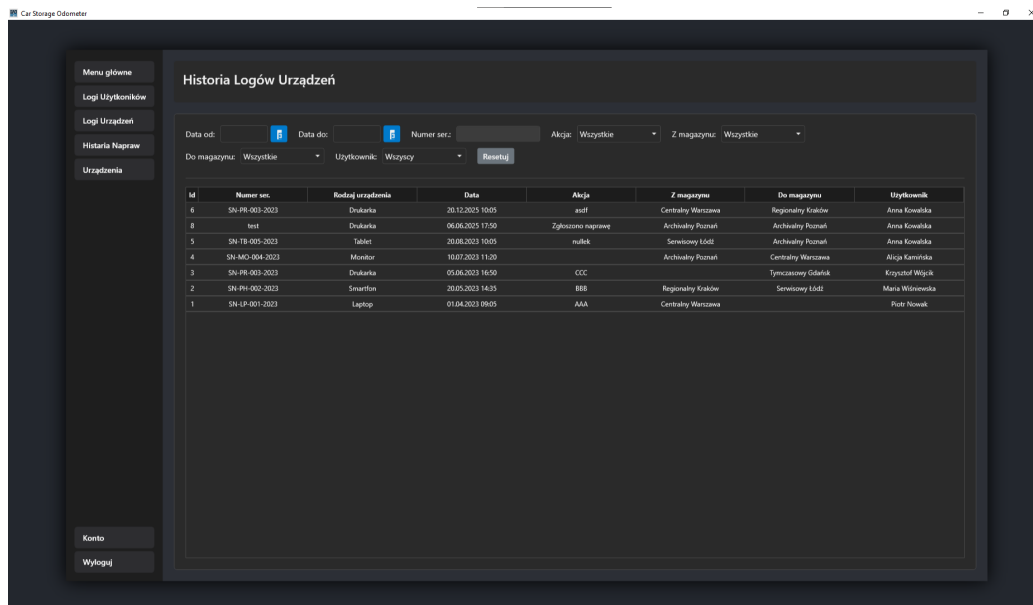
Rysunek 4: Region UserLogsView

Nad tabelą widnieje sekcja, gdzie możemy ustawić różne filtrowania:

- Zakresy dat od i do
- Czy chcemy filtrować po konkretnym użytkowniku, czy po wszystkich
- Czy chcemy filtrować po konkretnej akcji, czy po wszystkich
- przycisk do zresetowania filtru/ów

3.3 Logi urządzeń

Logi urządzeń to podobny region jak Logi Użytkowników.



Rysunek 5: Region DeviceLogsView

Umożliwia dodatkowo akcje takie jak filtrowanie:

- urządzeń po numerze seryjnym
- po konkretnych akcjach
- ze wszystkich lub do wszystkich magazynów

3.4 Historia napraw

Historia napraw również jest regionem bardzo zbliżonym do dwóch poprzednich. Pozwala podglądać naprawy konkretnych urządzeń.

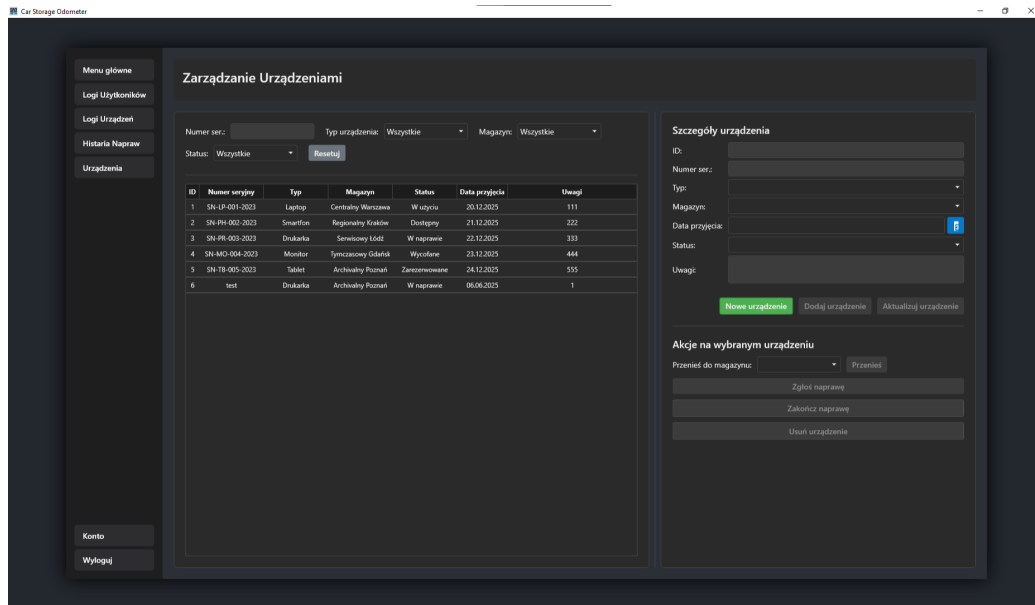
Tej sekcji potrzebowaliśmy również do powiększenia naszej bazy pod inny przedmiot.

Id	Numer seryjny	Opis	Data Start	Data Koniec	Użytkownik
6	test	Naprawa rozpoczęta	06.06.2023 17:50		Anna Kowalska
5	SN-MQ-004-2023	Wymiana podświetlenia	15.07.2023 00:00	18.07.2023 00:00	Alicja Karwiska
4	SN-TB-005-2023	Aktualizacja oprogramowania	10.06.2023 00:00	12.06.2023 00:00	Anna Kowalska
3	SN-PH-002-2023	Kalibracja ekranu	05.05.2023 00:00	07.05.2023 00:00	Krzysztof Wójcik
2	SN-LP-001-2023	Diagnostyka systemu	01.04.2023 00:00		Maria Wiśniewska
1	SN-PR-003-2023	Wymiana tonera	20.03.2023 00:00	21.03.2023 00:00	Piotr Nowak

Rysunek 6: Region HistoryOfRepair

3.5 Urządzenia

Z tej sekcji jesteśmy najbardziej dumni. Ponieważ, tak jak wspomnieliśmy wcześniej, chcemy rozwiązywać realne problemy, tutaj poświęciliśmy najwięcej czasu.

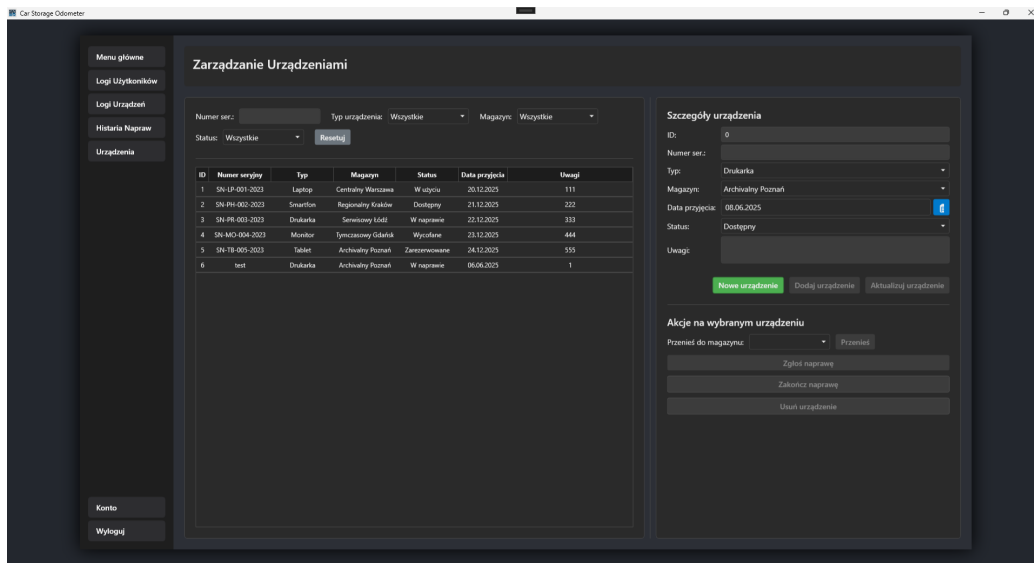


Rysunek 7: Region DevicesView

W tym regionie, pomijając już filtrowanie i wyświetlanie urządzeń, które działa tak samo jak we wcześniejszych sekcjach, możemy wykonać takie akcje jak:

- Dodanie nowego urządzenia
- Edycja istniejącego urządzenia
- Usunięcie istniejącego urządzenia
- Zgłoszenie, zakończenie i usunięcie naprawy
- przesuwanie między magazynami

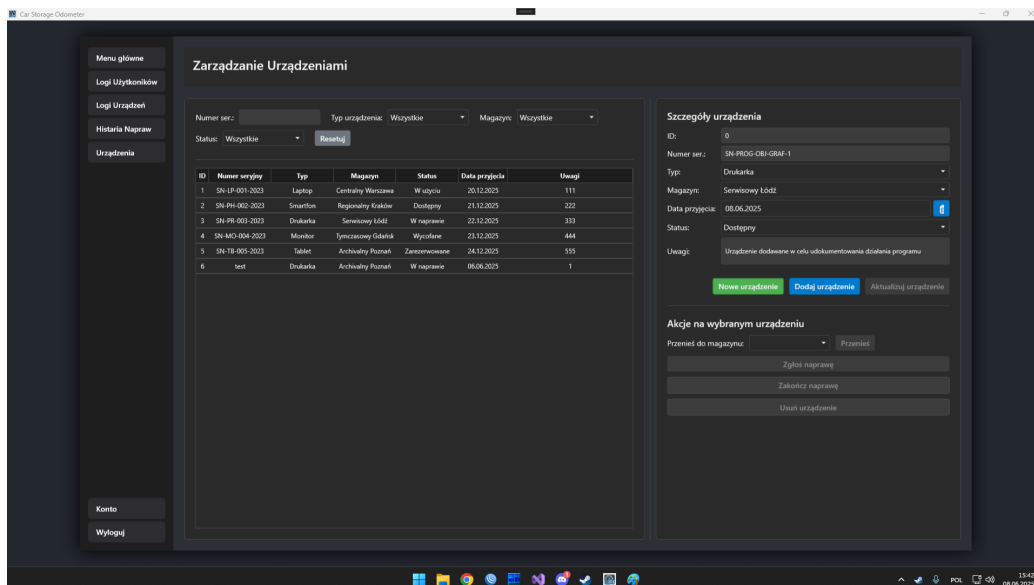
W celu **dodania nowego** urządzenia, najpierw naciskamy przycisk "Nowe urządzenie".



Rysunek 8: Dodawanie nowego urządzenia

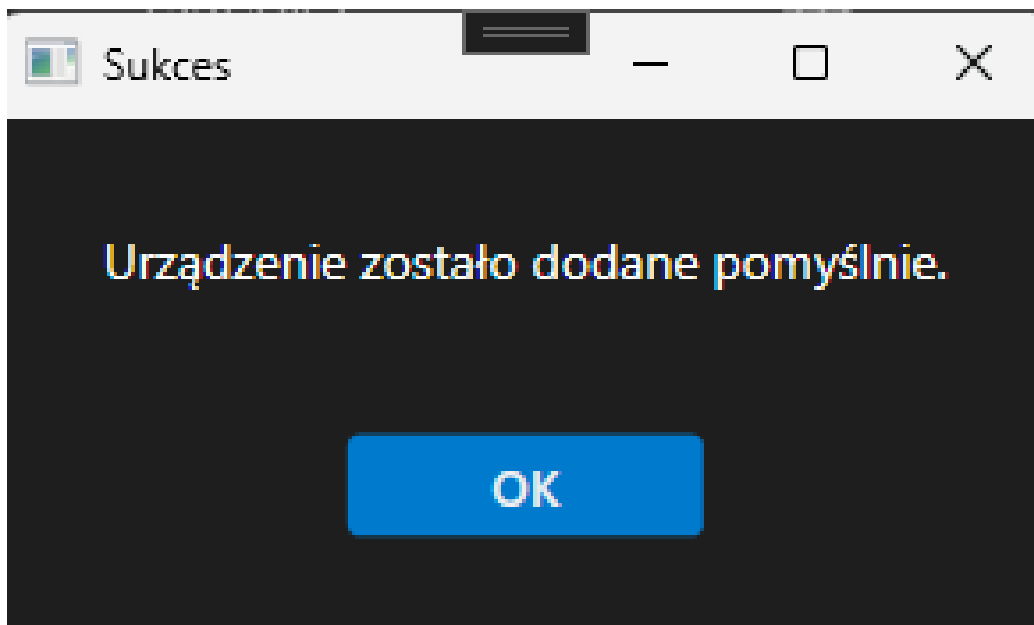
Dane w sekcji "Szczegóły urządzenia" zostaną wyczyszczone. ID urządzenia wyświetli "0" i nie jest to błąd, ponieważ ta zmienna jest inkrementowana sama, przy dodawaniu wpisu do bazy.

Aby móc dodać jakiegokolwiek nowe urządzenie, program wymaga od nas podania przynajmniej numeru seryjnego.



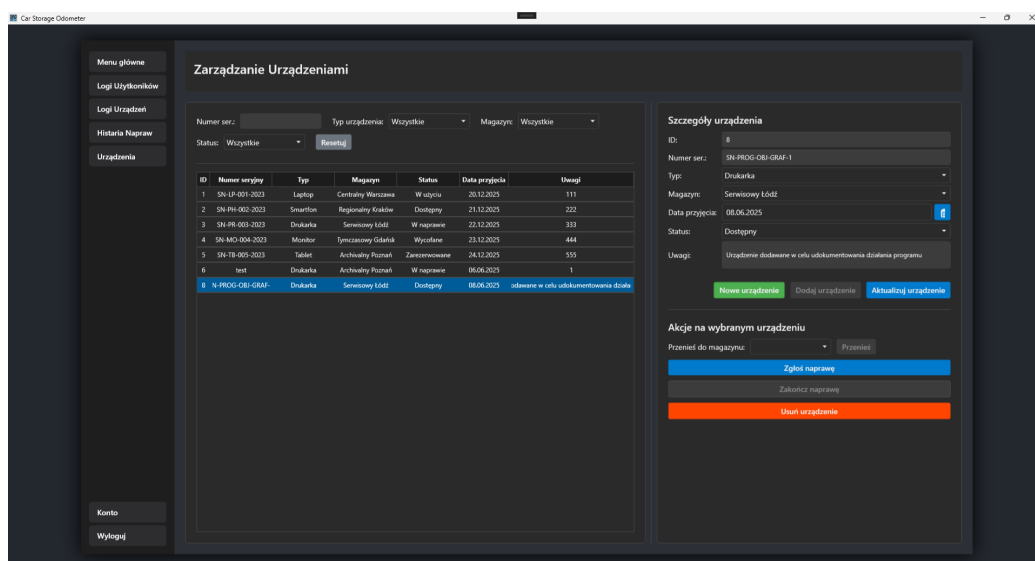
Rysunek 9: Uzupełnianie danych dla nowego urządzenia

Po podaniu odpowiednich danych, program umożliwia nam naciśnięcie przycisku "Dodaj urządzenie", a po dodaniu wyskakuje odpowiedni MessageBox, że urządzenie zostało pomyślnie dodane.

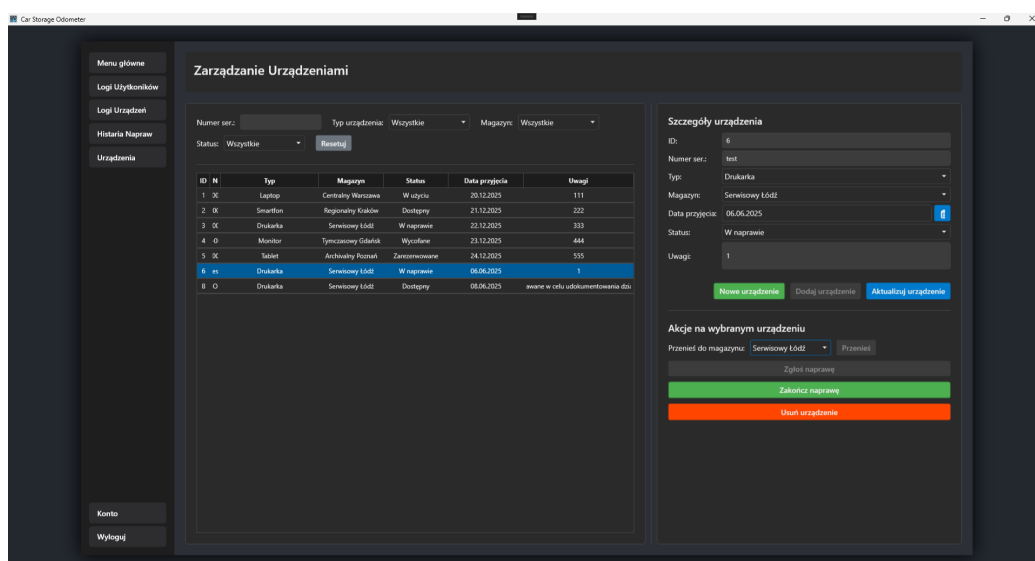


Rysunek 10: Pomyślnie dodano nowe urządzenie

Po poprawnym dodaniu i zaznaczeniu go, jesteśmy w stanie zaktualizować w nim dane (proces działa analogicznie, jak dodawanie nowego urządzenia), zgłosić do naprawy (ponieważ urządzenie ma status dostępny) oraz usunąć je z bazy.



Rysunek 11: Możliwe operacje na urządzeniu



Rysunek 12: Możliwe stany przycisków na wybranym urządzeniu

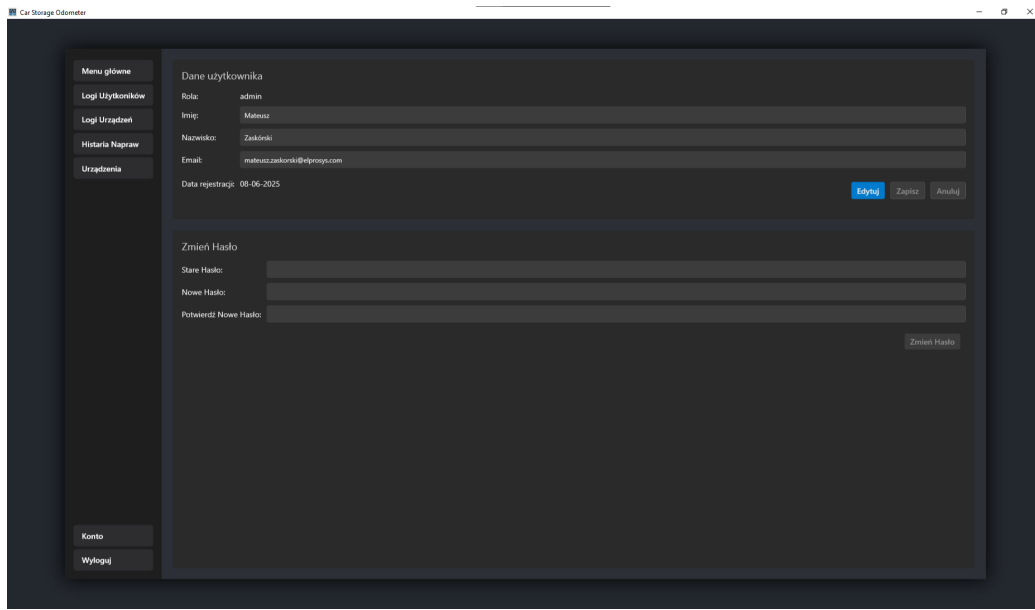
Nie będziemy tutaj pokazywać wszystkich możliwych stanów w GUI, jedynie nadmienimy, że zadbano o takie rzeczy jak:

- Przeniesienie na magazyn, w którym znajduje się zaznaczone urządzenie jest niemożliwe

- Urządzenia w naprawie nie można zgłosić ponownie do naprawy, należy najpierw zakończyć naprawę

3.6 Konto

W widoku "Konto" możemy podejrzeć dane naszego konta. Umożliwia on również zmianę danych, takich jak imię, nazwisko, adres e-mail oraz hasło.

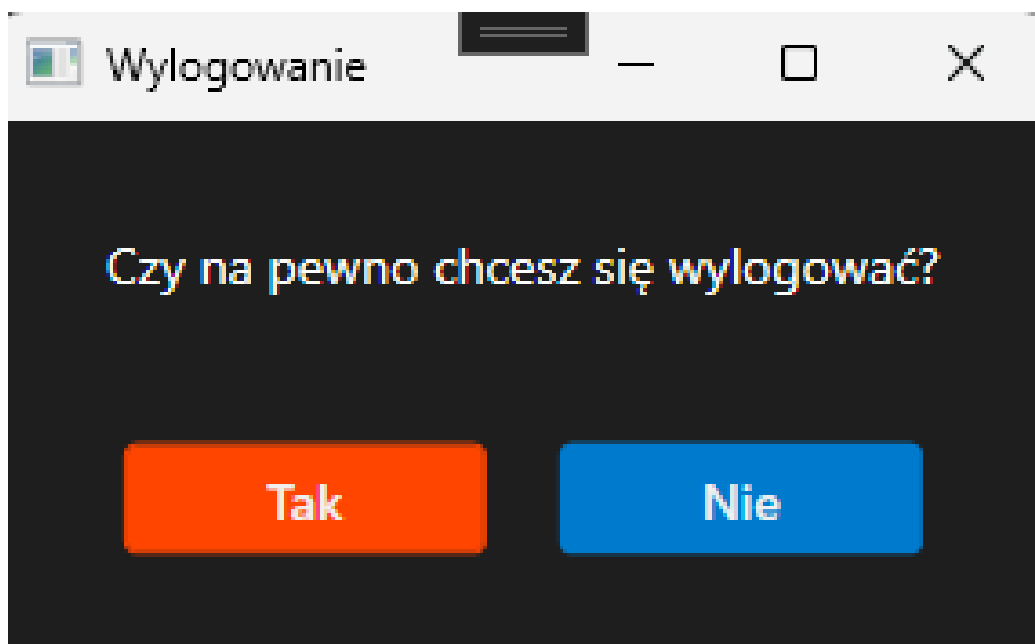


Rysunek 13: Region AccountView

Zadbaliśmy tutaj o poprawne zachowanie hashowania nowego hasła oraz tego, by użytkownik nie mógł sam zmienić swojej roli i daty rejestracji. Dodatkowo jak wcześniej wyskakują odpowiednie MessageBoxy.

3.7 Wyloguj

To po prostu przycisk, w którym zaimplementowaliśmy proces wylogowania użytkownika.



Rysunek 14: MessageBox po naciśnięciu przycisku ‘Wyloguj’

Oczywiście po naciśnięciu przycisku Tak nastąpi proces wylogowania i powrót do okna logowania, a po naciśnięciu przycisku Nie nastąpi powrót do okna z programem. Jeśli nie zamkniemy programu, a po wylogowaniu zalogujemy się ponownie, to aplikacja wróci do ostatnio przeglądane regionu.

4 Wnioski

- *Spostrzeżenia*
 - Wybrano przemyślany i solidny stos technologiczny (C# WPF, MVVM, Prism, SQLite), co zapewnia skalowalność, łatwość utrzymania i testowania aplikacji.
 - Zastosowanie wzorca MVVM i frameworku Prism promuje modularność i czystą separację odpowiedzialności, ułatwiając rozwój i przyszłe rozszerzenia.
 - Interfejs użytkownika wydaje się być intuicyjny i logicznie uporządkowany, z łatwym dostępem do kluczowych funkcji i informacji.
 - Wdrożono podstawowe mechanizmy bezpieczeństwa (hashowanie haseł) oraz walidację danych wejściowych, co świadczy o dbałości o integralność systemu.
 - Sekcja zarządzania urządzeniami jest kompleksowa, oferując szeroki zakres operacji niezbędnych do efektywnej pracy magazynu.
- *Osiągnięcia*
 - Skutecznie zaprojektowano i wdrożono aplikację, która efektywnie centralizuje zarządzanie magazynem, eliminując zależności od ręcznych arkuszy kalkulacyjnych.
 - Zbudowano aplikację w oparciu o nowoczesną i skalowalną architekturę (MVVM, Prism), co jest znaczącym osiągnięciem inżynierijnym.
 - Dostarczono kluczowe funkcjonalności do ewidencji i śledzenia urządzeń, zarządzania logami użytkowników i urządzeń, oraz obsługi napraw.
 - Zapewniono podstawową kontrolę dostępu poprzez system logowania, poprawiając bezpieczeństwo i śledzenie działań użytkowników.
 - Aplikacja zapewnia wysoką transparentność operacji dzięki szczegółowym logom i historii napraw, co ułatwia audyt i analizę.

- *Potencjał rozwoju*
 - Rozważenie wdrożenia bardziej zaawansowanego zarządzania rolami i uprawnieniami użytkowników, umożliwiającego granularną kontrolę dostępu do funkcji.
 - Możliwość rozbudowy o moduł raportowania i analityki, dostarczający głębszych spostrzeżeń na temat stanów magazynowych i przepływów urządzeń.
 - Potencjalna integracja z innymi systemami biznesowymi firmy (np. ERP, systemy księgowe) w celu dalszej automatyzacji procesów.
 - Wprowadzenie systemu powiadomień (np. o niskich stanach magazynowych lub wymaganych przeglądach) dla proaktywnego zarządzania.
 - Rozważenie optymalizacji bazy danych (przeniesienie jej w lokalizację sieciową dostępną dla korzystających z niej użytkowników) lub migracji do bardziej skalowalnych rozwiązań (np. SQL Server) w przypadku znacznego wzrostu ilości danych lub liczby użytkowników.
 - Dodanie opcji personalizacji widoków tabel przez użytkowników dla zwiększenia komfortu pracy.
- *Napotkane trudności*
 - Pierwszą poważną trudnością był dobór odpowiedniej wersji Prism'a, gdyż nowsze wersje nie zawierają części kluczowych dla naszych funkcji. Najwyższą poprawnie działającą wersją jest 8.1.97
 - Potrzeba aktualizacji logów podczas życia aplikacji spowodowała pewne trudności, ale skuteczne okazało się przerobienie synchronicznych funkcji na ich asynchroniczne wersje
 - Przechowywanie UserId, aby dodawane logi dotyczyły konkretnie zalogowanego użytkownika
 - System logowania i wylogowywania również nie był prosty, ale po dłuższym doczytaniu dokumentacji udało nam się usprawnić ten system wykorzystując mechanizm Prism - Eventy
 - Problemy ze standardowymi MessageBoxami, musieliśmy wdrożyć własne implementacje nadpisujące wyskakujące okna.

5 Bibliografia

1. Repozytorium GitHub
2. Kurs języka C#
3. Oficjalna dokumentacja Microsoft: Wprowadzenie do WPF (Windows Presentation Foundation)
4. Wzorzec architektoniczny MVVM (Model-View-ViewModel)
5. Oficjalna dokumentacja i wiki Prism Library
6. Oficjalna dokumentacja SQLite Database
7. Oficjalna dokumentacja języka C#
8. Dokumentacja dotycząca algorytmów hashujących w .NET