Obraz zawierający tekst, Czcionka

Opis wygenerowany automatycznie

***Mateusz Kapka***

***222459***

PRACA DYPLOMOWA INŻYNIERSKA

na kierunku Informatyka

**System internetowy w technologii .NET do obsługi wypożyczalni aut**

Katedra Mikroelektroniki i Technik Informatycznych

Promotor: dr inż. Paweł Marciniak

ŁÓDŹ 2023

**Streszczenie**

Niniejsza praca inżynierska koncentruje się na projektowaniu i implementacji systemu internetowego opartego na technologii .NET, w tym frameworku React, oraz bazie danych SQL Server, mającego na celu usprawnienie obsługi wypożyczalni aut. Współczesny rynek usług wynajmu samochodów wymaga innowacyjnych rozwiązań, które poprawią dostępność, efektywność i wygodę dla klientów.

Praca rozpoczyna się od analizy wymagań technicznych, co pozwala na stworzenie kompleksowego systemu obejmującego zarządzanie flotą pojazdów, rezerwacje online, obsługę klientów i monitorowanie stanu samochodów. Wykorzystanie technologii .NET, w tym React, gwarantuje skalowalność, bezpieczeństwo i łatwość utrzymania systemu. Baza danych SQL Server zapewnia niezbędną trwałość i wydajność przechowywanych danych.

W pracy szczegółowo opisano proces implementacji, skupiając się na kwestiach związanych z bezpieczeństwem danych, intuicyjnym interfejsem użytkownika oraz integracją z systemami płatności. Dzięki temu system może zapewnić nie tylko wydajność i dostępność, ale także bezpieczeństwo informacji klientów.

Wdrożenie tego systemu w wypożyczalniach aut może znacząco przyczynić się do poprawy jakości obsługi klientów i efektywności operacyjnej firm wypożyczających samochody. Opracowany system internetowy w technologii .NET z wykorzystaniem React i SQL Server stanowi innowacyjne rozwiązanie, które przynosi korzyści zarówno klientom, jak i przedsiębiorstwom wypożyczalni.

Słowa kluczowe: .NET framework, React, aplikacja internetowa, SQL Server, zarządzanie pojazdami

# Wprowadzenie

W dzisiejszym dynamicznym świecie mobilności, wypożyczalnie samochodów odgrywają kluczową rolę w ułatwianiu dostępu do pojazdów dla szerokiego grona użytkowników. W odpowiedzi na rosnące zapotrzebowanie na wygodne i efektywne rozwiązania z zakresu wypożyczania samochodów, ta praca inżynierska prezentuje projekt i implementację nowoczesnego systemu internetowego, który ma na celu usprawnienie procesu zarządzania wypożyczalnią samochodów.

## Cel i znaczenie pracy

Głównym celem tej pracy jest opracowanie kompleksowego systemu internetowego do obsługi wypożyczalni samochodów, który umożliwi zarówno klientom, jak i pracownikom obsługującym wypożyczalnię łatwy dostęp do usług oraz efektywne zarządzanie flotą pojazdów. System ten wykorzystuje nowoczesne technologie, takie jak platforma .NET, framework React oraz bazę danych SQL Server, aby dostarczyć wydajne, skalowalne i intuicyjne rozwiązanie.

Projekt ten ma na celu zaspokojenie potrzeb klientów, umożliwiając im łatwe przeglądanie dostępnych pojazdów, dokonywanie rezerwacji online i zarządzanie swoimi profilami. Jednocześnie, pracownikom wypożyczalni umożliwia sprawną administrację flotą samochodów, obsługę klientów oraz monitorowanie stanu pojazdów.

## Struktura pracy

Praca inżynierska jest podzielona na kilka głównych rozdziałów, które skupiają się na różnych aspektach projektu i implementacji systemu. W kolejnych rozdziałach zostaną omówione projektowanie bazy danych, implementacja w technologiach .NET i React, a także szczegółowe opisy funkcjonalności systemu.

Ponadto, praca zawiera podsumowanie osiągniętych celów, perspektywy rozwoju systemu oraz wnioski wynikające z projektu.

System ten ma na celu nie tylko zautomatyzowanie procesów wypożyczalni samochodów, ale także dostarczenie wygodnych narzędzi zarówno klientom, jak i pracownikom, aby zapewnić im efektywną i przyjemną interakcję z usługą. Praca ta stanowi próbę stworzenia innowacyjnego rozwiązania, które przyczyni się do usprawnienia obsługi wypożyczalni samochodów w erze cyfrowej mobilności.

# Narzędzia i środowisko pracy

W trakcie projektowania i implementacji systemu internetowego do obsługi wypożyczalni samochodów wykorzystano szeroki zakres narzędzi, które umożliwiły efektywne zarządzanie kodem źródłowym, testowanie aplikacji, a także tworzenie intuicyjnego interfejsu użytkownika. Poniżej przedstawiono kluczowe narzędzia i środowisko pracy, które odegrały istotną rolę w procesie tworzenia systemu.

## Git i GitHub

Git został wykorzystany jako system kontroli wersji, co umożliwiło śledzenie zmian w kodzie źródłowym, zarządzanie gałęziami (branchami), oraz współpracę nad projektem. GitHub jako platforma internetowa do hostowania repozytoriów Git, pozwolił na przechowywanie i udostępnianie kodu, a także śledzenie problemów (issues) i prośb o ściągnięcie (pull requests).

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Rys.1 Projekt na platformie GitHub

## Postman

Postman stanowił nieocenione narzędzie podczas testowania interfejsu API mojej aplikacji. Umożliwił mi tworzenie zapytań HTTP, testowanie różnych ścieżek dostępu i sprawdzanie odpowiedzi serwera. Dzięki Postmanowi mogłem zapewnić, że moje API działa zgodnie z oczekiwaniami i spełnia wymagania funkcjonalne.

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Rys.1 Przykładowy żądanie HTTP do pobrania informacji o aucie

## Visual Studio 2022

Visual Studio 2022 stało się głównym środowiskiem programistycznym do tworzenia części backendowej mojej aplikacji opartej na technologii .NET. Dostarczyło mi narzędzi do tworzenia, debugowania i testowania kodu, a także zapewniło dostęp do szerokiej gamy rozszerzeń i bibliotek.

## Visual Studio Code

Visual Studio Code było używane do implementacji komponentów frontendowych w technologii React. Jego lekki charakter i obszerna społeczność rozszerzeń sprawiły, że

stanowiło idealne środowisko do tworzenia interfejsu użytkownika. Visual Studio Code zapewniło składniowe podpowiedzi, integrowane narzędzia do zarządzania projektem i wbudowany terminal do uruchamiania komend.

# Architektura projektu

Koncepcja architektury projektu opiera się na zasadach, które promują czytelność, skalowalność i utrzymanie kodu. Obejmuje trzy główne obszary:

* warstwa aplikacji – w obszarze warstwy aplikacji analizuje się organizację komponentów, strukturę kodu oraz obsługę żądań i odpowiedzi,
* warstwa bazy danych – warstwa bazy danych jest kluczowym elementem projektu. Obejmuje zarządzanie danymi, modelowanie struktury bazy danych oraz operacje CRUD,
* część kliencka – część kliencka to interfejs użytkownika i sposób, w jaki użytkownicy integrują z aplikacją.

## Warstwowa architektura aplikacji

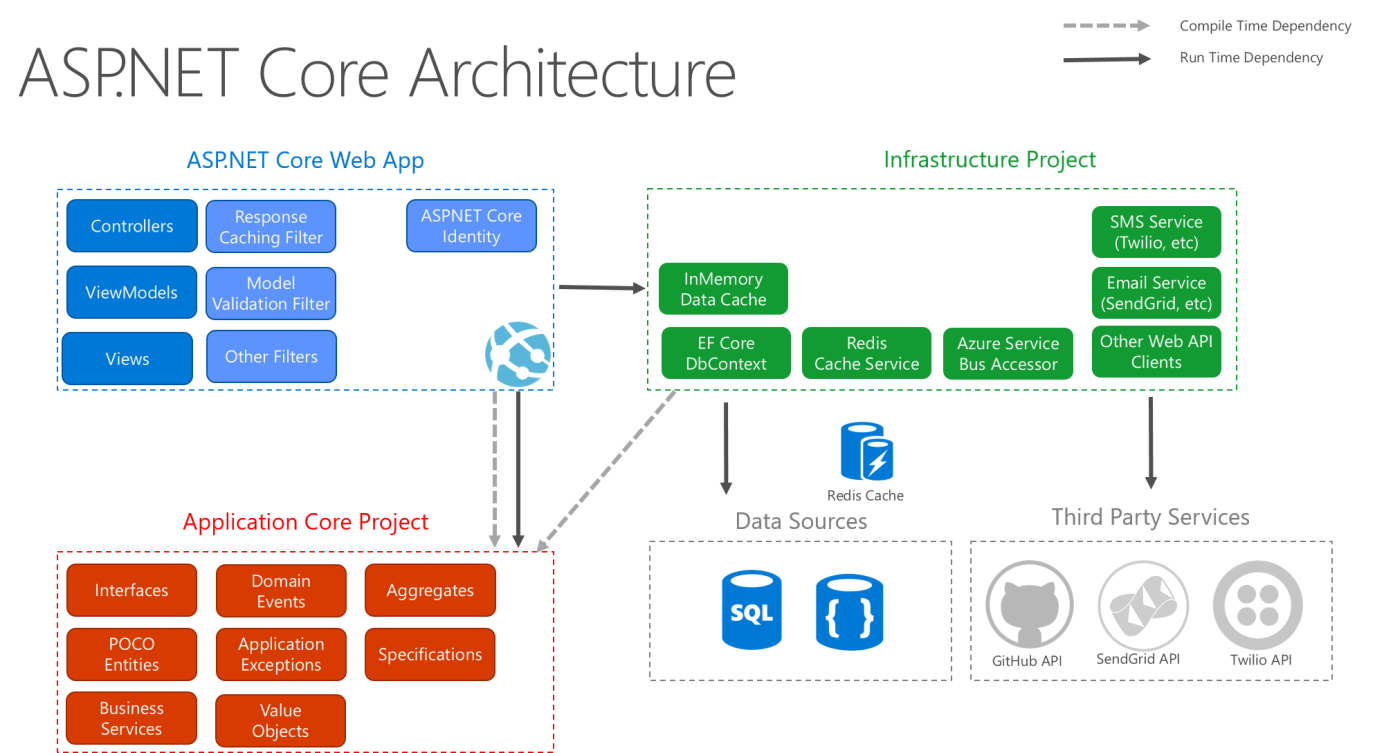
Aplikacja ASP.NET Web API 6 opiera się na „czystej architekturze kodu”, która pomaga w organizacji projektu i oddzielaniu odpowiedzialności. Oto pięć głównych warstw:

* warstwa API – warstwa API udostępnia naszą aplikację jako interfejs programistyczny. Tutaj definiujemy punkty końcowe, obsługę żądań HTTP i komunikację z klientami,
* warstwa prezentacji – warstwa prezentacji zajmuje się interfejsem użytkownika. W tym miejscu znajdują się widoki, kontrolery interfejsu użytkownika oraz komponenty odpowiedzialne za prezentację danych,
* warstwa infrastruktury – warstwa infrastruktury zawiera narzędzia, helpery i komponenty używane w różnych częściach projektu, w tym dostęp do bazy danych oraz inne aspekty infrastrukturalne,
* warstwa aplikacji – to tutaj skupiamy logikę biznesową aplikacji. Definiujemy przypadki użycia, przetwarzamy dane i wykonujemy operacje biznesowe. Warstwa aplikacji komunikuje się z innymi warstwami i dostarcza pełną funkcjonalność aplikacji,
* warstwa domeny – warstwa domeny zawiera modele biznesowe oraz reprezentacje encji i agregatów. W tej warstwie definiujemy strukturę danych i reguły biznesowe.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Struktura aplikacji



**Rys. 1** ASP.NET Core diagram architektury po czystej architekturze

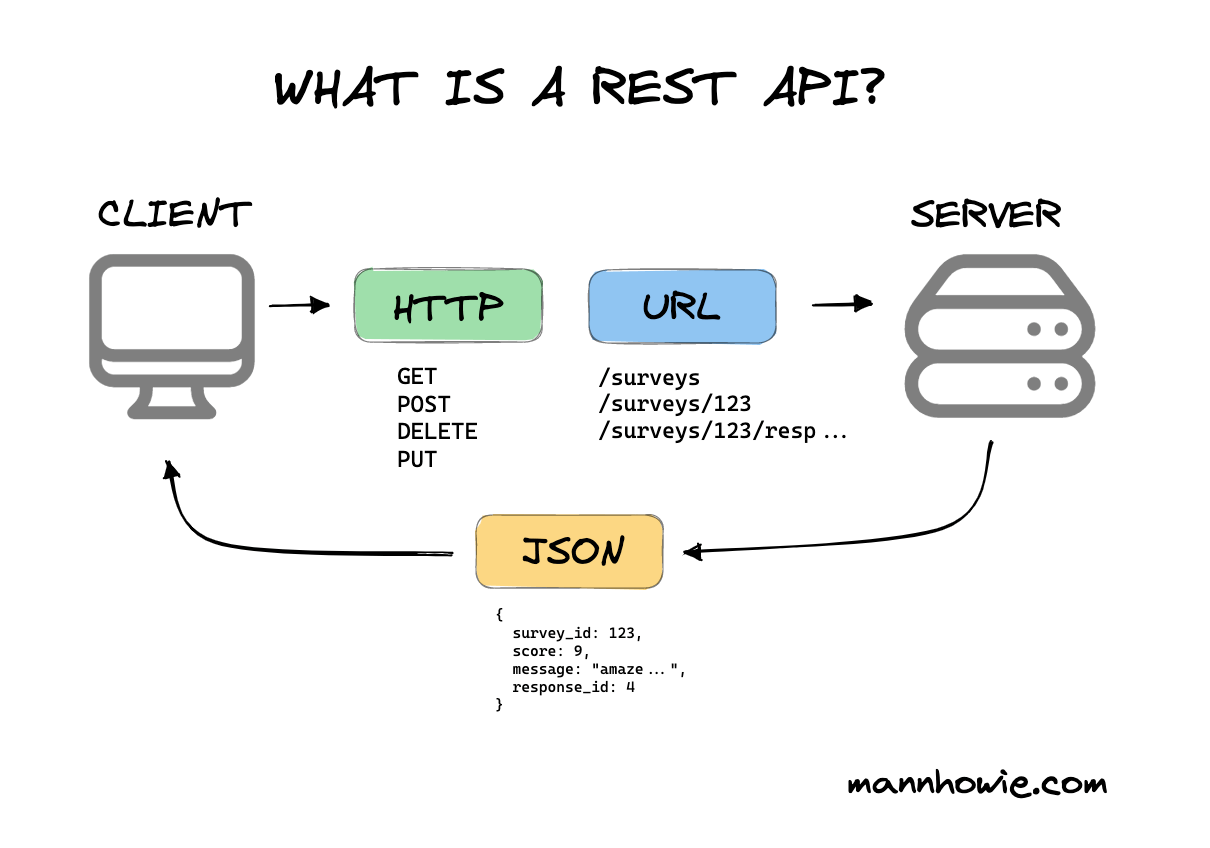
## Zarządzanie stanem

W aplikacji używa się mechanizmów zarządzania stanem, takich jak Dependency Injection (DI) i Inversion of Control Container. Pozwala to na wstrzykiwanie zależności i kontrolowanie ich cyklu życia w całej aplikacji.

## Komunikacja z aplikacja

Projekt wykorzystuje zestaw dostępnych punktów końcowych API w celu komunikacji z serwerem backendowym. Wykorzystywane są standardy takie jak HTTP oraz protokół RESTful do przekazywania żądań i odpowiedzi między klientem a serwerem. Można wymienić 5 podstawowych żądań HTTP:

* GET – w przypadku żądań typu GET, aplikacja odczytuje dane z zasobów w bazie danych lub innych źródeł danych,
* POST – żądania POST są używane do tworzenia nowych zasobów w aplikacji,
* PUT – żądania PUT służą do aktualizacji istniejących zasobów
* DELETE – żadania DELETE służą do usuwania zasobów z aplikacji
* PATCH – ten typ pozwala na częściową aktualizację istniejących zasobów



**Rys 1** Przykład RESTful API https://mannhowie.com/rest-api

Żądania HTTP, takie jak GET, POST, PUT, PATCH i DELETE, odgrywają kluczową rolę w zarządzaniu danymi w aplikacjach internetowych. Te żądania umożliwiają klientom komunikację z serwerem i wykonywanie operacji na danych. Są one ściśle powiązane z operacjami CRUD (Create, Read, Update, Delete).

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys 1. Operacje CRUD

## Swagger

W celu dokumentowania i testowania API, wykorzystuje się narzędzie Swagger. Swagger ułatwia zarządzanie i udostępnianie API w sposób przejrzysty i zrozumiały. Dzięki niemu można szybko zrozumieć, testować i korzystać z API, bez konieczności zagłębiania się w szczegóły implementacyjne. To także ułatwia proces tworzenia i utrzymania dokumentacji technicznej, co przyczynia się do lepszej komunikacji w ramach projektu.

Obraz zawierający tekst, linia, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Endpointy użytkownika

## Obsługa błędów

Aplikacja ma wbudowaną obsługę błędów, która jest odpowiedzialna za rejestrowanie i obsługę błędów podczas przetwarzania żądań HTTP. Błędy są odpowiednio logowane i obsługiwane, a klientowi zwracane są odpowiednie kody i komunikaty błędów.

Obraz zawierający tekst, zrzut ekranu, Czcionka, informacja

Opis wygenerowany automatycznie

Rys. 1 Zapis żądań w logu

## Bezpieczeństwo i uwierzytelnianie

Bezpieczeństwo aplikacji jest zapewnione poprzez mechanizmy uwierzytelniania i autoryzacji. Wykorzystujemy tokeny JWT (JSON Web Tokens) do uwierzytelniania użytkowników i kontrolowania dostępu do zasobów.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Dekodowanie tokenu w jwt.io

# Projekt bazy danych

Projekt bazy danych, który stanowi ważny element systemu internetowego do obsługi wypożyczalni samochodów. Baza danych obejmuje zarówno strukturę bazy danych, jak i relacje między tabelami, niezbędne do skutecznego przechowywania i zarządzania danymi.

## Model danych

Na początku procesu definiowaliśmy model danych, który reprezentuje główne encje naszego systemu, takie jak użytkownicy, samochody, wypożyczenia, komentarze itp. Model ten jest definiowany w kodzie źródłowym za pomocą klas C#. Nasz model danych obejmuje następujące główne encje:

* użytkownicy – informacje o użytkownikach, takie jak dane osobowe, dane kontaktowe, numer dowodu osobistego, numer prawa jazdy,
* poświadczenia użytkowników – zawierają dane uwierzytelniające użytkowników
* wypożyczenia – informacje o wypożyczeniach aut, o osobach które wypożyczyły auto, data wypożyczenia, data zwrotu
* samochody – dane dotyczące dostępnych samochodów w wypożyczalni
* blogi – zawartość blogów
* komentarze – komentarze użytkowników na blogach

## Konfiguracja encji

Każda klasa encji jest skonfigurowana przy użyciu atrybutów i konwencji Entity Framework. Określiliśmy relacje między encjami, klucze główne, klucze obce, indeksy i inne właściwości, które wpływają na strukturę bazy danych.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Encja użytkownika

## Migracje

Entity Framework umożliwia korzystanie z mechanizmu migracji, który pozwala na ewolucję struktury bazy danych w miarę rozwoju projektu. Tworzymy migracje, które zawierają zmiany w modelu danych, a Entity Framework automatycznie aktualizuje bazę danych na podstawie tych migracji. Dzięki temu utrzymujemy spójność między modelem danych a strukturą bazy danych.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. 1 Fragment klasy migracji

## Generowanie bazy danych

Po zdefiniowaniu modelu danych i stworzeniu odpowiednich migracji możemy generować bazę danych. Entity Framework automatycznie tworzy lub aktualizuje bazę danych SQL Server na podstawie naszego modelu danych.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

**Rys. 1** Diagram bazodanowy przedstawiający tabele i ich relacje

## Zarządzanie modelem

W trakcie rozwoju projektu, gdy zmienia się struktura danych lub dodaje nowe encje, aktualizujemy model danych w kodzie źródłowym. Następnie generujemy odpowiednie migracje i aktualizujemy bazę danych, aby odzwierciedlić te zmiany.

Dzięki podejściu "Code First" z użyciem Entity Framework jesteśmy w stanie efektywnie zarządzać bazą danych SQL Server, zachowując spójność między modelem a rzeczywistą strukturą danych w systemie.

## Rola DbContext w aplikacji

*DbContext* w aplikacjach opartych na Entity Framework w technologii .NET pełni kluczową rolę. Jest to klasa, która dziedziczy po klasie bazowej DbContext dostarczonej przez Entity Framework i umożliwia efektywną interakcję między aplikacją a bazą danych SQL Server.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Rys. 1 Klasa abstrakcyjna DbContext

Główne funkcje DbContext obejmują definiowanie modelu danych, konfigurację połączenia z bazą danych, wykonywanie operacji CRUD na danych, mapowanie obiektów na struktury tabel w bazie danych, obsługę migracji bazy danych, łączenie danych z różnych źródeł, zarządzanie stanem obiektów i wiele innych. Jest to nieodłączna część aplikacji .NET, która ułatwia pracę z bazą danych i zapewnia integralność danych oraz wydajność aplikacji.

## Hashowanie haseł

W celu zabezpieczenia haseł użytkowników, aplikacja wykorzystuje funkcje haszującą, taką jak SHA-256. Te funkcja przekształcają hasło użytkownika w unikalny ciąg znaków, który jest trudny do odwrócenia. Haszowanie hasła polega na przekształceniu hasła użytkownika w skrócony, nieodwracalny zapis, który jest przechowywany w bazie danych zamiast samego hasła.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Tabela użytkowników z zahashowanymi hasłami

Podczas procesu rejestracji lub zmiany hasła, aplikacja generuje nowy skrót hasła i zapisuje go w bazie danych. Podczas procesu uwierzytelniania, aplikacja porównuje hasz wprowadzonego przez użytkownika hasła z tym, który jest przechowywany w bazie danych. Jeśli hasze są zgodne, użytkownik jest autoryzowany.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. 1 Implementacja metody hashującej

To podejście zapewnia ochronę haseł przed dostępem osób nieuprawnionych, nawet w przypadku wycieku danych z bazy danych. Hashowanie haseł jest praktyką bezpieczeństwa powszechnie stosowaną w aplikacjach internetowych, aby zabezpieczyć poufne informacje użytkowników.

# Warstwa aplikacji

Podczas konfiguracji środowiska .NET przeprowadzono niezbędne dostosowania narzędzi, środowisk uruchomieniowych oraz bibliotek, umożliwiających efektywny rozwój i działanie aplikacji w technologii .NET. Dodatkowo, rozpoczęto implementację głównej logiki aplikacji w środowisku .NET, co obejmuje tworzenie i zarządzanie funkcjonalnościami systemu.

## Implementacja logiki biznesowej

Implementacja logiki biznesowej polegała na przetwarzaniu danych, wykonywaniu operacji biznesowych oraz zarządzaniu funkcjonalnościami aplikacji. W tym etapie zaimplementowano przypadki użycia, reguły biznesowe oraz operacje związane z procesem wypożyczania samochodów.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Rys. 1 Implementacja serwisu do logowania użytkowników

## Repozytorium

W ramach implementacji w technologii .NET, kluczowym elementem projektu jest wykorzystanie wzorca projektowego Repozytorium (Repository). Repozytorium jest warstwą abstrakcji pomiędzy aplikacją a bazą danych, co umożliwia efektywne zarządzanie dostępem do danych. Główne cele Repozytorium to izolacja logiki dostępu do danych oraz umożliwienie bardziej elastycznego korzystania z danych w różnych częściach aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 1 Fragment repozytorium samochodu

Repozytorium pełni rolę pośrednika między warstwą aplikacji a bazą danych SQL Server, co pozwala na oddzielenie operacji bazodanowych od logiki biznesowej. Dzięki temu, operacje takie jak odczyt, zapis i aktualizacja danych są bardziej czytelne i łatwiejsze do utrzymania. Ponadto, Repozytorium może zapewniać abstrakcyjne metody dostępu do danych, co ułatwia testowanie jednostkowe i zapewnia spójność w całej aplikacji.

## Kontroler

Kontroler pełni zasadniczą rolę w obsłudze żądań HTTP oraz zarządzaniu logiką przetwarzania tych żądań. Jest odpowiedzialny za przekazywanie żądań do odpowiednich miejsc w aplikacji i zwracanie odpowiedzi na te żądania. Oto kilka najważniejszych cech kontrolerów:

* przetwarzanie akcji – kontroler zawiera różne akcje, które odpowiadają na konkretne żądania HTTP, takie jak GET, POST, PUT, PATCH, DELETE itp. Każda akcja jest publiczną metodą kontrolera i jest oznaczona atrybutami, które określają rodzaj żądania, który jest obsługiwany,
* modelowanie danych – kontroler może przyjmować dane wejściowe od klienta, np. parametry żądania lub ciało żądania, i przekazywać je do odpowiednich miejsc w logice biznesowej. Może również formatować dane wyjściowe i tworzyć odpowiedzi, które zostaną wysłane z powrotem do klienta,
* obsługa tras – kontroler zarządza trasami, które określają, które akcje są wywoływane w odpowiedzi na określone żądania. Trasy są definiowane w konfiguracji aplikacji i określają, jakie URL-e są mapowane na konkretne akcje kontrolera,
* atrybuty działania – kontrolery mogą być zabezpieczone przy użyciu atrybutu *Authorize*, który określa, że dostęp do danej akcji lub kontrolera jest ograniczony i wymaga autentykacji użytkownika. Dodatkowo, atrybut *Roles* pozwala na określenie, które role użytkowników mają dostęp do danej akcji lub kontrolera,
* testowalność – kontrolery są często testowane, aby upewnić się, że poprawnie przetwarzają żądania i generują oczekiwane odpowiedzi. Testy jednostkowe i testy integracyjne są stosowane, aby zweryfikować funkcjonalność kontrolera.



Rys. 1 Kontroler do obsługi hasła użytkownika

## Zarządzanie DTO

W celu efektywnej komunikacji między frontendem a backendem, w projekcie wprowadzono mechanizm DTO (Data Transfer Object). DTO to struktury danych, które służą do przekazywania informacji między różnymi warstwami aplikacji. Implementacja DTO umożliwia odseparowanie modeli używanych w warstwie backendowej od modeli wykorzystywanych w warstwie frontendowej. Dzięki temu można precyzyjnie kontrolować dane przekazywane do klienta, zapewniając jednocześnie optymalną wydajność i bezpieczeństwo transmisji danych.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Rys. 1 Model DTO loginu

## Serializacja do formatu JSON

W celu przekazywania danych między klientem a serwerem, używamy formatu JSON (JavaScript Object Notation). JSON jest lekkim, czytelnym i łatwym do analizy formatem danych. W projekcie zaimplementowano mechanizmy serializacji i deserializacji danych do i z formatu JSON. To pozwala na skuteczną wymianę danych między warstwami aplikacji oraz z klientem, co jest szczególnie istotne w przypadku komunikacji asynchronicznej.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. Fragment odpowiedzi w formacie JSON o użytkowniku

## Wykorzystanie biblioteki AutoMapper

Biblioteka *AutoMapper* została wykorzystana do automatycznego mapowania danych między obiektami DTO a obiektami biznesowymi. Dzięki *AutoMapper* możliwe jest zdefiniowanie reguł mapowania i przekształcania danych bez konieczności pisania dużych ilości powtarzalnego kodu. To zwiększa efektywność i czytelność kodu oraz przyspiesza rozwijanie aplikacji.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Rys. 1 Fragment mapowania encji i DTOs

## Logowanie z użyciem NLog

W celu monitorowania działania aplikacji oraz identyfikacji błędów, w projekcie zaimplementowano mechanizm logowania przy użyciu biblioteki *NLog*. Logi aplikacji pozwalają na zbieranie informacji o wykonywanych operacjach, zgłaszanych błędach i wydajności systemu. Dzięki *NLog* możliwe jest elastyczne konfigurowanie sposobu rejestrowania logów oraz ich analizy.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. 1 Fragment pliku konfiguracyjnego nlog.config

## Walidacja danych z FluentValidation

W ramach projektów programistycznych, narzędzie FluentValidation pełni kluczową rolę w obszarze walidacji danych wejściowych. Jest to biblioteka umożliwiająca programistom precyzyjne definiowanie reguł walidacji w sposób deklaratywny oraz czytelny. Główne cechy FluentValidation obejmują intuicyjną składnię, zdolność do stosowania reguł walidacji do różnorodnych typów danych, personalizację komunikatów błędów oraz możliwość definiowania reguł na poziomie modelu danych.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. 1 Fragment klasy walidatora modelu CarDTO

FluentValidation stanowi fundament zabezpieczenia aplikacji przed błędnymi lub niezgodnymi danymi wejściowymi. To istotny aspekt zapewniający integralność danych oraz wzmacniający bezpieczeństwo systemu. Ponadto, narzędzie to ułatwia proces testowania reguł walidacji, co jest nieodłączną częścią tworzenia stabilnych i niezawodnych aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rys. 1 Metoda obsługująca dodawanie auta wraz z walidacja danych

## Middleware

Middleware w aplikacji ASP.NET Web API jest używane do przetwarzania żądań i odpowiedzi HTTP. Stanowi warstwę pośrednią między klientem a aplikacją, umożliwiającą różne operacje na żądaniach i odpowiedziach. Middleware jest aplikowany w kolejności rejestracji, co pozwala na tworzenie potoku przetwarzania żądań. Przykłady zastosowania Middleware kontroli dostępu obejmują:

* autentykacja użytkowników – middleware może być używane do autentykacji użytkowników przy użyciu różnych mechanizmów, takich jak tokeny JWT, ciasteczka sesji, uwierzytelnianie przez OAuth i wiele innych,
* autoryzacja dostępu do zasobów – middleware pozwala na definiowanie reguł autoryzacji, które określają, czy dany użytkownik ma dostęp do określonych zasobów lub funkcjonalności. Na przykład, administratorzy mogą mieć dostęp do zaawansowanych opcji, podczas gdy zwykli użytkownicy nie,
* kontrola dostępu do akcji API – middleware może kontrolować dostęp do konkretnych akcji API na podstawie ról użytkowników, uprawnień lub innych kryteriów,
* ograniczenia dostępu do danych – middleware pozwala na definiowanie ograniczeń dostępu do danych, co oznacza, że użytkownicy mogą widzieć tylko dane, do których mają uprawnienia.

Można także wymienić wiele rodzajów middleware, z tych ważniejszych to:

* middleware autentykacji (Authentication Middleware) – ten rodzaj middleware obsługuje proces autentykacji użytkowników. Przykłady obejmują middleware obsługujące tokeny JWT, uwierzytelnianie przez OAuth, autentykację przez ciasteczka sesji, itp.,
* middleware autoryzacji (Authorization Middleware) – middleware autoryzacji kontroluje dostęp użytkowników do zasobów lub funkcjonalności na podstawie określonych reguł i uprawnień,
* middleware obsługi wyjątków (Exception Handling Middleware) – ten rodzaj middleware przechwytuje błędy i wyjątki, które mogą wystąpić w trakcie przetwarzania żądania i generuje odpowiedzi błędów lub podejmuje odpowiednie działania reakcji na błąd,
* middleware kontroli dostępu (Access Control Middleware) – middleware kontrolujący dostęp odpowiedzialny jest za sprawdzanie czy użytkownik ma dostęp do określonych zasobów lub funkcji. Może opierać się na rolach użytkowników, uprawnieniach, tokenach, itp.,
* middleware obsługi żądań (Request Handling Middleware) – middleware obsługujące żądania jest odpowiedzialne za przetwarzanie i kierowanie żądań HTTP do odpowiednich akcji lub obsługujących funkcji,
* middleware do logowania żądań (Request Logging Middleware) – middleware logowania żądań służy do rejestrowania żądań HTTP i odpowiedzi generowanych przez aplikację. Pomaga w monitorowaniu działania aplikacji, diagnozowaniu problemów oraz zbieraniu informacji o wydajności,
* middleware do logowania wyjątków (Exception Logging Middleware) – middleware logowania wyjątków przechwytuje błędy i wyjątki, a następnie loguje je w określonym formacie lub przesyła do systemu monitoringu błędów. Pomaga to w identyfikowaniu i rozwiązywaniu problemów w aplikacji,

Te rodzaje middleware są kluczowe w procesie przetwarzania żądań w aplikacji ASP.NET Web API 6. Każdy z nich ma specyficzne zadanie i pomaga w zapewnieniu niezawodności, bezpieczeństwa i monitorowania działania aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Rys. 1 Fragment middleware obsługi wyjątków z rzucanymi kodami błędu

## Autentykacja

Autentykacja to proces potwierdzania tożsamości użytkownika w systemie. W ramach autentykacji użytkownik dostarcza dane, takie jak nazwa użytkownika i hasło, które pozwalają systemowi potwierdzić, czy jest on tym, za kogo się podaje. Inne metody autentykacji mogą obejmować wykorzystanie tokenów autentykacyjnych lub usług autentykacji zewnętrznych, takich jak logowanie za pomocą konta społecznościowego.

## Autoryzacja

Autoryzacja to proces kontrolowania dostępu użytkownika do określonych zasobów lub funkcjonalności w aplikacji. Oznacza to określenie, jakie czynności lub zasoby są dostępne dla użytkownika w oparciu o jego tożsamość i uprawnienia. Autoryzacja może być oparta na rolach użytkowników, uprawnieniach lub bardziej zaawansowanych politykach dostępu.

# Warstwa kliencka

Warstwa kliencka odgrywa kluczową rolę jako interfejs użytkownika aplikacji internetowej. Jest to miejsce, gdzie użytkownicy wchodzą w interakcje z aplikacją. W tym kontekście wykorzystano technologie takie jak React i Bootstrap, aby stworzyć zarówno atrakcyjny wizualnie, jak i funkcjonalny interfejs.

## React

React to biblioteka JavaScript, która jest szeroko wykorzystywana do tworzenia interaktywnych interfejsów użytkownika (UI) w aplikacjach internetowych. W kontekście projektu, React pełni rolę kluczowej technologii w warstwie klienckiej, umożliwiając tworzenie interfejsu użytkownika, zarządzanie stanem aplikacji i komunikację z serwerem backendowym. Oto główne aspekty związane z użyciem React:

* tworzenie Interfejsu użytkownika – *React* umożliwia programistom tworzenie interaktywnych i dynamicznych interfejsów użytkownika. Komponenty *React* reprezentują różne części interfejsu, takie jak przyciski, formularze, listy czy strony. Te komponenty są łatwe do utrzymania, ponieważ każdy z nich jest oddzielną jednostką, która może być wielokrotnie wykorzystywana w całej aplikacji,
* reaktywność – React wprowadza reaktywność do interfejsu użytkownika. Zmiany w stanie aplikacji automatycznie prowadzą do aktualizacji odpowiednich komponentów, co eliminuje konieczność ręcznej manipulacji DOM (Document Object Model). Dzięki temu interakcje użytkownika są płynne i natychmiastowe,
* komponenty funkcyjne i klasowe – React obsługuje zarówno komponenty funkcyjne, jak i klasowe. Komponenty funkcyjne to bardziej nowoczesna forma definicji komponentów, która jest bardziej zwięzła i czytelna. Komponenty klasowe, choć mniej popularne, pozwalają na korzystanie z pełnej gamy funkcji cyklu życia komponentów,
* stan komponentu – React umożliwia definiowanie i zarządzanie stanem komponentu. Stan jest miejscem, w którym przechowywane są dane, które mogą się zmieniać w trakcie działania aplikacji. Aktualizacja stanu prowadzi do ponownego renderowania komponentu, co pozwala na dynamiczne reagowanie na interakcje użytkownika,
* jednokierunkowy przepływ danych – w React dane przepływają w jednym kierunku, co ułatwia śledzenie, jakie zmiany wpływają na komponenty. Jest to znane jako jednokierunkowy przepływ danych (ang. "one-way data flow"). Dane są przekazywane od komponentu nadrzędnego do podrzędnego za pomocą właściwości (props),
* biblioteki i rozszerzenia – w ekosystemie React istnieje wiele dodatkowych bibliotek i narzędzi, które ułatwiają rozwijanie aplikacji. Na przykład, Redux służy do zarządzania stanem aplikacji, React Router do nawigacji między widokami, a Axios/Fetch do wykonywania żądań http,
* virtual DOM – React wykorzystuje wirtualny DOM (Virtual Document Object Model), co pozwala na minimalizację operacji na rzeczywistym DOM, co jest wydajniejsze. React porównuje wirtualny DOM z rzeczywistym i aktualizuje tylko te elementy, które zmieniły się.

## Bootstrap