POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka (INF)
SPECJALNOŚĆ: Inżynieria systemów informatycznych (INS)

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja webowa wspomagająca zarządzanie flotą samochodów

A web application supporting cars fleet management

AUTOR:
Jan Pajdak

PROWADZĄCY PRACĘ:
dr inż. Jarosław Mierzwa, K-9

OPIEKUN:
dr hab. inż. Olgierd Unold Prof. nadzw. PWr, K-9

OCENA PRACY:

WROCŁAW 2018

Spis treści

Rozdział 1

Wprowadzenie

1.1 Wstęp

Celem niniejszej pracy dyplomowej jest opracowanie projektu, implementacja oraz wdrożenie systemu umożliwiającego zarządzanie flotą samochodów. Pierwszym etapem projektu jest zebrane wymagań funkcjonalnych i niefunkcjonalnych oraz określenie zakresu pracy. Drugi etap projektu to wybór technologii i projekt architektury. Ostatnim celem jest implementacja systemu.

Temat projektu został wybrany ze względu na chęć wykorzystania wiedzy z dziedziny motoryzacji w celu stworzenia aplikacji ułatwiającej zarządzanie pojazdami. Z uwagi na rosnącą popularność rozwiązań związanych z wypożyczaniem samochodów celem projektu jest system, który można opisać jako wewnątrzfirmową wypożyczalnie umożliwiająca jak największe wykorzystanie dostępnej floty pojazdów przez pracowników, którzy nie mają potrzeby posiadania firmowego samochodu na wyłączność.

1.2 Cel i zakres pracy

Celem projektu jest stworzenie aplikacji umożliwiającej wypożyczanie oraz zarządzanie flotą samochodów. Aplikacja jest skierowana do firm które nie mają potrzeby lub wystarczających środków by zapewnić pracownikom samochody na wyłączność. Przykładowym przypadkiem użycia systemu może być jednorazowa potrzeba odwiedzenia klienta lub wyjazd na szkolenie. Typowe rozwiązania dla firm obecne na rynku skierowane są do firm świadczących usługi spedycyjne — aplikacje posiadają warstwę śledzenia ładunków oraz tworzenia zadań przewozowych dla kierowców; programy służące do obsługi komercyjnych wypożyczalni pomijają proces autoryzacji wypożyczenia — zwykle sprawdzana jest zdolność wypożyczającego do zapłaty.

Projekt utworzony w ramach tej pracy łączy mechanikę z komercyjnych wypożyczalni z dodatkową warstwą biznesową pozwalającą kontrolować sposób używania pojazdów.

Zakres pracy obejmuje utworzenie systemu spełniającego wymagania postawione w rozdziale 3 oraz przygotowanie projektu do wdrożenia, poprzez np. konteneryzacje.

1.3 Układ pracy

W rozdziale pierwszym zawarto wstęp oraz krótki opis celu projektu. Drugi rozdział porównuje istniejące rozwiązania do aplikacji będącej celem projektu. Rozdział trzeci zawiera

wymagania funkcjonalne oraz niefunkcjonalne. W kolejnym, czwartym rozdziale znajduje się opis wybranych technologii oraz narzędzi, wraz z uzasadnieniem.

Rozdział 2

Istniejące rozwiązania

Rozdział 3

Wymagania funkcjonalne i niefunkcjonalne

3.1 Wymagania funkcjonalne

Wymagania zostały opisane według poniższego wzorca:

Numer	Numer wymagania
Nazwa	Krótka nazwa
Opis	Dokładny opis
Aktor	Grupa użytkowników
Kryterium spełnienia	Funkcjonalność która musi zostać zaimplementowana by wymaganie można było uznać jako spełnione
Ograniczenia	Ograniczenia funkcjonalności, jeżeli takie istnieją

Rozróżniane są dwa rodzaje aktorów:

- Kierowca - użytkownik korzystający z funkcjonalności tworzenia i przeglądania historii wypożyczeń
- Kierownik - użytkownik z pełnym dostępem do systemu

Kierownik posiada wszelkie prawa i możliwości Kierowcy.

Dodatkowe pojęcia związane z modelami świata biznesowego:

- **Model Pojazdu** - model opisujący specyfikacje techniczną wspólną dla pewnego zbioru pojazdów
- **Pojazd** - model opisujący informacje unikatowe dla pewnego przedstawiciela zbioru Modeli Pojazdów.

Numer	1
Nazwa	Zarządzanie modelami pojazdów
Opis	System powinien pozwalać na dodawanie i edycję modeli pojazdów; specyfikacji technicznej dla danego modelu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może dodawać nowe modele samochodów. Informacje mogą zostać w późniejszym czasie zmodyfikowane lub usunięte.
Ograniczenia	Model pojazdu może zostać usunięty wyłącznie gdy nie ma żadnych pojazdów

Numer	2
Nazwa	Zarządzanie pojazdami
Opis	System powinien pozwalać na dodawanie i edycję pojazdów będących egzemplarzami modeli z wymagania #2; pojazd zawiera informacje unikalne dla danego egzemplarza, takie jak numer rejestracyjny.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może dodawać nowe pojazdy dla wybranego modelu. Informacje mogą zostać w późniejszym czasie zmodyfikowane lub usunięte.
Ograniczenia	Pojazd nie może być modyfikowany gdy jest obecnie wypożyczony. Pojazd który był wypożyczany nie może zostać usunięty — może zostać oznaczony jako wycofany z użycia.

Numer	3
Nazwa	Zarządzanie ubezpieczeniami pojazdu
Opis	System powinien umożliwiać wprowadzanie informacji związanych z ubezpieczeniami danego pojazdu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać historię ubezpieczeń danego pojazdu oraz wprowadzać nowe dane. System bierze pod uwagę obecny stan pojazdu podczas tworzenia wypożyczenia; pojazd nie może zostać wypożyczony w okresie gdy nie ma aktywnego ubezpieczenia.
Ograniczenia	

Numer	4
Nazwa	Zarządzanie serwisami pojazdu
Opis	System powinien umożliwiać wprowadzanie informacji związanych z serwisami danego pojazdu.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać historię napraw danego pojazdu oraz wprowadzać nowe dane. System rozróżnia różne rodzaje serwisowania takie jak regularny przegląd, zdarzenie wyjątkowe czy naprawa powypadkowa. System bierze pod uwagę obecny stan pojazdu podczas tworzenia wypożyczenia; pojazd nie może zostać wypożyczony gdy jest obecnie naprawiany.
Ograniczenia	

Numer	5
Nazwa	Tworzenie wypożyczeń
Opis	System powinien umożliwiać przeglądanie dostępnych pojazdów (dostępność określana jest na podstawie informacji z wymagania #2) i tworzenie wypożyczeń wraz z niezbędnymi danymi takimi jak okres czasu i potrzeba stojąca za wypożyczeniem
Aktor	Kierowca
Kryterium spełnienia	Kierowca może utworzyć wypożyczenie
Ograniczenia	Kierowca nie może utworzyć wypożyczenia dla innego użytkownika

Numer	6
Nazwa	Kontrola wypożyczeń
Opis	System umożliwia kontrolowanie stanu wypożyczenia. Wypożyczenie uznane jest za obowiązujące dopiero po akceptacji przez uprawnioną do tego osobę.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może przeglądać wypożyczenia utworzone przez użytkowników systemu oraz zmieniać ich obecny stan po ocenie zasadności wypożyczenia
Ograniczenia	Kierownik nie może akceptować własnych wypożyczeń

Numer	7
Nazwa	Zbieranie informacji o kosztach wypożyczeń
Opis	System umożliwia śledzenie kosztów utrzymania floty na podstawie raportów wprowadzanych przez wypożyczających.
Aktor	Kierowca
Kryterium spełnienia	Kierowca może wprowadzić informację związane z wypożyczeniem (zużyte litry paliwa, przejechane kilometry, całkowity koszt) po oddaniu samochodu.
Ograniczenia	

Numer	8
Nazwa	Zbieranie informacji o kosztach utrzymania
Opis	System umożliwia śledzenie kosztów utrzymania floty związanych z ubezpieczeniami oraz naprawami.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może wprowadzić koszty związane z ubezpieczeniem/serwisem pojazdu.
Ograniczenia	

Numer	9
Nazwa	Wyświetlanie informacji o kosztach utrzymania
Opis	System jest w stanie wygenerować plik kompatybilny z programem *Excel* zawierający dane na temat kosztów floty.
Aktor	Kierownik
Kryterium spełnienia	Kierownik może wywołać utworzenie pliku ze statystykami
Ograniczenia	

Numer	10
Nazwa	Przechowywanie informacji audytowych
Opis	System zapisuje informacje o dacie i użytkowniku dokonującym wprowadzenia nowych danych lub modyfikacji istniejących.
Aktor	Kierownik
Kryterium spełnienia	Informacje o dacie i użytkowniku modyfikowane są w trakcie zapisu do bazy danych. Kierownik może przeglądać dane audytowe.
Ograniczenia	

3.2 Wymagania niefunkcjonalne

3.2.1 Interfejs użytkownika

- Wygląd powinien być prosty i nowoczesny
- Elementy strony powinny być rozmieszczone w intuicyjny sposób
- Struktura widoków powinna być ułożona zgodnie z zależnościami między wyświetlanymi danymi
- Aplikacja być wygodna w użyciu na ekranach komputerów o rozdzielczości HD (1366x768 pikseli) lub większej

3.2.2 Interfejs programistyczny

- System powinien wymagać niewielkich modyfikacji w przypadku integracji z istniejącymi zasobami firmy (np. baza danych pracowników)
- Komunikacja powinna opierać się na otwartych i uniwersalnych standardach, np. dane w postaci *JSON* lub *XML* przesyłane protokołem *HTTP*

- Interfejs programistyczny powinien być niezależny od platformy tak by w przyszłości mógł zostać wykorzystany przez inne aplikacje

3.2.3 Bezpieczeństwo

System powinien być zabezpieczony zarówno po stronie interfejsu użytkownika (np. blokada przed przejściem do podstrony) oraz po stronie interfejsu programistycznego (ignorowanie zapytań od nieupoważnionych aplikacji). Zabezpieczenie powinno obsługiwać różne poziomy autoryzacji w zależności od roli użytkownika.

Rozdział 4

Zastosowane technologie i narzędzia

4.1 Zastosowane technologie

Interfejs użytkownika wykorzystuje platformę *Angular 7*. Podstawowymi elementami w *Angular* są komponenty [1], każdy z nich złożony z: pliku klasy *TypeScript* zawierającej logikę, wzorca *htm* opisującego wygląd widoku oraz opcjonalnego stylu *css*; w przypadku jego braku styl brany jest z komponentu-rodzica. Warto zwrócić uwagę na język programowania wykorzystywany przez platformę *Angular* — *TypeScript* [2], będący rozszerzeniem języka *JavaScript*. *TypeScript* dodaje silniejsze typowanie i kładzie większy nacisk na programowanie obiektowe, jednocześnie pozostając w pełni kompatybilnym z *JavaScript*, do którego jest kompilowany i następnie uruchamiany jest w przeglądarce. Proces kompilacji pozwala na usunięcie wielu błędów, które w przypadku *JavaScript* zostałyby zauważone dopiero po uruchomieniu aplikacji.

Jednym z ważniejszych komponentów aplikacji jest *Bootstrap* - framework interfejsu użytkownika pozwalający w prosty sposób tworzyć estetyczne strony internetowe. Dodatkowo, w projekcie wykorzystano motywy *Bootswatch*.

Interfejs programistyczny oparty został na technologii *ASP.NET Core 2.1* — jest to nowoczesna platforma oferująca działanie na wielu systemach operacyjnych oraz większa wydajność względem starszych rozwiązań firmy Microsoft. Wykorzystany język programowania to obiektowy, kompilowany i statycznie typowany *C# 7.3*. Bardzo ważnym elementem tej części projektu jest *EF (Entity Framework) Core 2.1* [3], framework ORM (Object-Relational Mapping) pozwalający na konwersję miedzy tabelami bazy danych a klasami *C#*. Jedną z najważniejszych funkcjonalności *EF Core* jest wykorzystana w niniejszym projekcie możliwość utworzenia bazy danych przy użyciu konwencji *Code First*; baza danych jest automatycznie generowana na podstawie klas C# znajdujących się w projekcie. *EF Core* współpracuje z większością popularnych baz danych; na potrzeby tego projektu wykorzystano *MS SQL Server*.

4.2 Wykorzystane narzędzia

W trakcie realizacji projektu wykorzystane zostały narzędzia najczęściej używane przy wybranych technologiach.

Do zarządzania kodem został wykorzystany system kontroli wersji *Git*. Lokalna kopia projektu była synchronizowana ze zdalnym, prywatnym repozytorium znajdującym się na serwisie GitHub. Wykorzystane rozwiązanie pozwala na łatwy dostęp do wcześniejszych wersji projektu oraz zmniejsza ryzyko utraty kodu, gdyż nie jest on przechowywany tylko

w jednym miejscu.

Ze względu na wykorzystane technologie, kod był rozwijany z pomocą narzędzi Microsoft, oferujących najlepsze wsparcie dla *TypeScript* oraz *C#*. Aplikacja klienta była rozwijana przy użyciu *Visual Studio Code 1.28*, nowoczesnego edytora który sprawdza się znakomicie przy tworzeniu interfejsów użytkownika ze względu na zintegrowaną konsolę pozwalającą na łatwe zarządzanie paczkami oraz łatwość dostosowywania do potrzeb użytkownika. W trakcie pracy wykorzystano wiele rozszerzeń, najważniejsze z nich to *TSLint*, linter wykrywający błędy w kodzie *TypeScript* oraz *GitLens* — rozszerzenie wspomagające zarządzanie repozytorium *Git*. Do rozwoju serwisów wykorzystano *Visual Studio 2017* pozwalające na łatwe debugowanie kodu oraz analizę aspektów takich jak wykorzystanie zasobów przez program. *Visual Studio* zostało wzbogacone o narzędzie *JetBrains ReSharper* automatycznie formatujące pliki projektu według zadanego wzorca, zapewniając spójność i przejrzystość kodu.

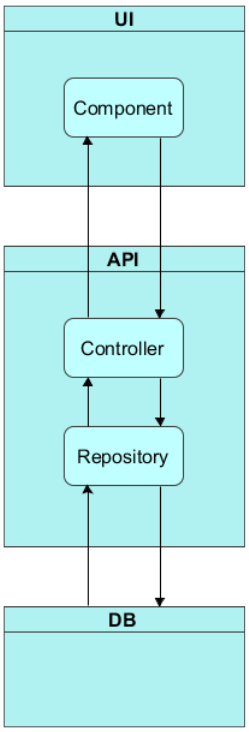
Interfejs programistyczny testowany był przy pomocy *Postman 6.5.2*, aplikacji pozwalającej na wysyłanie oraz zarządzanie zapytaniami HTTP.

Rozdział 5

Projekt

5.1 Architektura

System został stworzony przy użyciu klasycznej architektury w której można wyodrębić trzy moduły - interfejs użytkownika (UI), interfejs programistyczny (API) oraz bazę danych (DB).



Rysunek 5.1 Uproszczony schemat architektury z wyodrębnionymi najważniejszymi rodzajami klas

System został zaprojektowany tak, by mógł zostać zintegrowany z istniejącymi zasobami firmy — jedyne dane jakie przechowuje dotyczą logiki biznesowej związanej z wymaganiami funkcjonalnymi; wynika to z faktu, że większość firm ma już własne bazy danych przechowujące informacje o pracownikach więc duplikacja danych jest niepożądana ze względu na zużycie zasobów oraz możliwe problemy z synchronizacją. Dane związane z użytkownikami (np. imię, nazwisko, e-mail i numer telefonu) czy lokacjami firmy (np. adres) mogą zostać pobrane z innej bazy danych; ponadto interfejs użytkownika nie

umożliwia wprowadzania lub edycji takich danych. Implementacja opisana w dalszej części niniejszej pracy przechowuje przykładowe dane użytkowników do celów testowych w tej samej bazie danych, jednakże konfiguracja systemu tak by korzystał z innej, nie stanowi większego problemu.

W architekturze można rozróżnić trzy najważniejsze składowe, dwie pierwsze w interfejsie programistycznym i trzecią w interfejsie użytkownika:

- Kontroler (*Controller*) to klasa odpowiadająca za obsługę żądań *HTTP* [4].
- Repozytorium (*Repository*) zawiera logikę pośredniczącą w komunikacji między *API* a bazą danych.
- Komponent (*Component*) to podstawowy element definiujący działanie widoku w *Angular* [1].

W celu zapewnienia przejrzystości kodu, nazewnictwo wszystkich elementów oraz dokumentacja kodu są zgodne ze standardową konwencją danego języka. Kod jest napisany w całości w języku angielskim.

Język	Typy	Pliki	Zmienne prywatne	Inne zmienne
C#	PascalCase	SomeExampleClass.cs	camelCase	PascalCase
TypeScript	PascalCase	some-example.class.ts	camelCase	camelCase

Tablica 5.1 Najważniejsze konwencje nazewnicze

5.2 Bezpieczeństwo

5.3 Konteneryzacja

Rozdział 6

Testy

Rozdział 7

Podsumowanie

7.1 Wnioski

7.2 Możliwości rozwoju

Bibliografia

[1] Angular Docs. *Introduction to components*, 2018. URL `https://angular.io/guide/architecture-components`. Dostęp 03.12.2018.

[2] MSDN. *TypeScript - Understanding TypeScript*, 2015. URL `https://msdn.microsoft.com/en-us/magazine/dn890374.aspx`. Dostęp 03.12.2018.

[3] MSDN. *Entity Framework Core*, 2016. URL `https://docs.microsoft.com/en-us/ef/core`. Dostęp 03.12.2018.

[4] MSDN. *Build web APIs with ASP.NET Core*, 2018. URL `https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-2.1`. Dostęp 03.12.2018.

Spis rysunków

Spis tablic