WSEI

Praca semestralna Programowanie Aplikacji BackEnd lato 2021/2022

Temat pracy:
Web Api dedykowane dla serwisów rowerowych.

Danylo Sydorchuk
13309

Lab8

Spis treści

Serwis Rowerowy

Opis aplikacji

Aplikacja „WebSerwisRowerowy” dedykowana dla serwisów rowerowych. Jest oprogramowaniem działającym w trybie on-line typu WebApi.
Aplikacja komunikuje się za pomocą metod GET, POST, PUT, DELETE. Wszystkie informacje są przechowywane w bazie danych.

Aplikacja umożliwia:

- Rejestracje i logowanie użytkowników,
- Dodawanie, usuwanie i modyfikacja zleceń serwisowych,
- Dodawanie, usuwanie i modyfikacja usług świadczonych przez serwis rowerowy,
- Dodawanie, usuwanie i modyfikacja pracowników w firmie i klientów,
- Wysyłkę maila do klienta przy zmianie statusu zlecenia.

Technologie użyte w aplikacji

Została dodana instancja DbContext, która reprezentuje sesję z bazą danych, która jest używana do zapytań oraz poleceń do bazy danych.

Do aplikacji została dodana funkcja Migracji, która umożliwia przyrostowe aktualizowanie schematu bazy danych w celu zachowania synchronizacji z modelem danych aplikacji przy jednoczesnym zachowaniu istniejących danych w bazie danych.

Wykonuje się insert przykładowych danych do tabel w bazie, jeżeli tabela jest pusta.

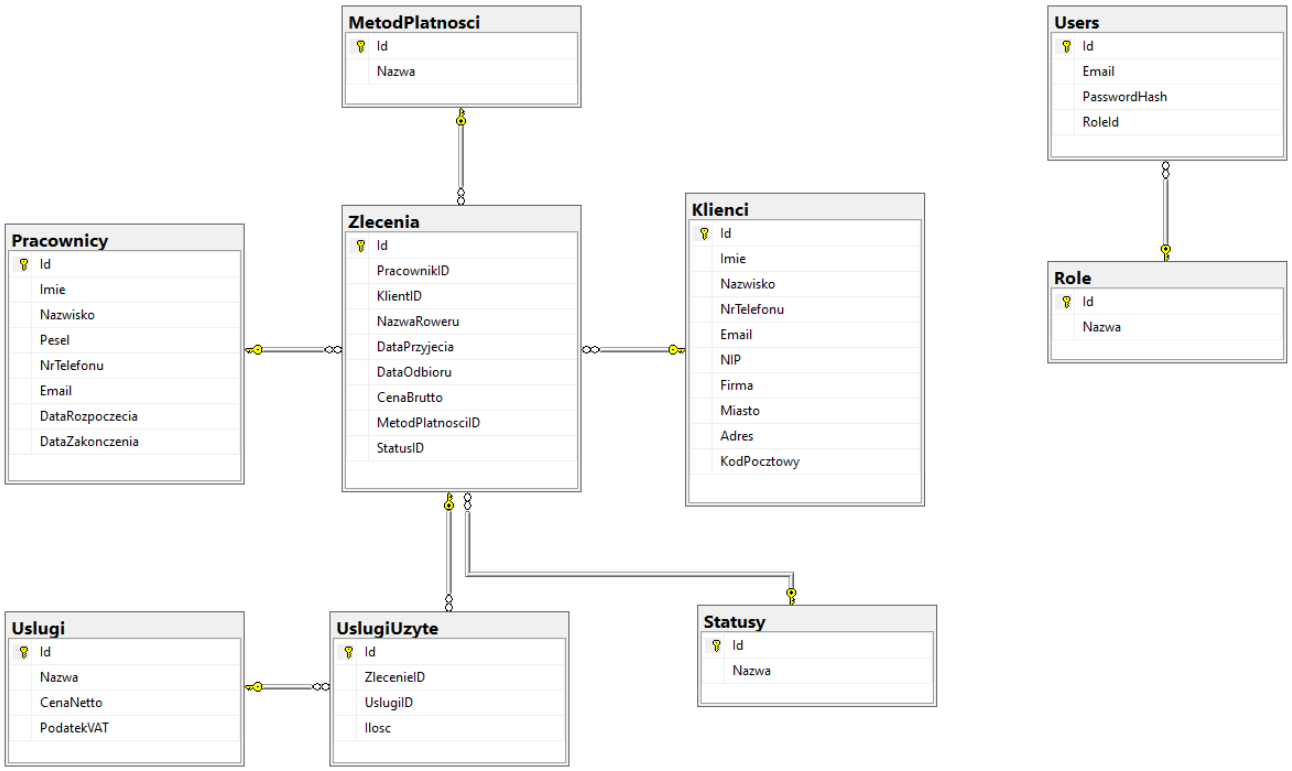
Do aplikacji została dodana obsługa wyjątków: BadRequestException i NotFoundException. Zaimplementowany interfejs IMiddleware i klasa ErrorHandlingMiddleware, która wyłapuje dane wyjątki.

Wysyłka maili następuje przy zmianie statusu zlecenia serwisowego. Komunikacja następuje za pomocą klasy SmtpClient, która służy do wysyłania wiadomości e-mail na serwer SMTP, i MailMessage, która reprezentuje wiadomość e-mail, którą można wysłać przy użyciu SmtpClient klasy.

Aplikacja generuje JWT (JSON Web Token), który definiuje sposób wymiany danych między stronami w bezpieczny sposób poprzez obiekt JSON. Token zawiera dane do logowania, rolę użytkownika i czas wygaśnięcia tokenu = 15 dni. Token jest zwracany po prawidłowym logowaniu użytkownika.

Do aplikacji został dodany AutoMapper, który pomaga skopiować dane z jednego obiektu do drugiego.

SCHEMAT BAZY DANYCH



PRZYKŁADOWE ZAPYTANIA

REJESTRACJA UŻYTKOWNIKA

Komunikat POST: /api/account/register

Oczekiwany parametr [FromBody]:

```
{
   "email": "admin@admin.pl",
   "password": "admin1",
   "confirmPassword": "admin1",
   "role":2
}
```

,gdzie

Role – identyfikator roli użytkownika

a) 1-User

b) 2-Admin

Domyślnie przypisuje się rola Usera.

Wymagania do rejestracji:

Email – nie pusty unikalny,
Password – długość minimum 6 znaków,
ConfirmPassword – taki sam jak password.

LOGOWANIE UŻYTKOWNIKA

Komunikat POST:/api/account/login
Oczekiwany parametr [FromBody]:

```
{
    "email": "admin@admin.pl",
    "password": "admin1"
}
```

,output: token, w którym mamy zawarty login, hasło i role użytkownika.
Wygenerowany token należy dołączać do nagłówka następnych zapytań do API.

ZLECENIECONTROLLER

Komunikat GET:/api/zlecenie

Zwraca wszystkie zlecenia świadczone przez firmę.
Zapytanie dostępne dla zalogowanych użytkowników.

Komunikat GET:/api/zlecenie/1

Zwraca zlecenie o id 1.
Zapytanie dostępne dla wszystkich użytkowników.
Zwraca strukturę:


```
{
    "id": 1,
    "pracownikID": 1,
    "klientID": 1,
    "nazwaRoweru": "Bianchi Szosa",
    "dataPrzyjecia": "2022-07-03T13:31:08.7507577",
    "dataOdbioru": null,
    "cenaBrutto": null,
    "metodPlatnosciID": null,
    "statusID": 3,
    "uslugiUzyte": []
}
```


,gdzie:
pracownikID – numer pracownika w tabeli Pracownicy
klientID – numer klienta w tabeli Klienci
statusID:
1 - Anulowane
2 - Do odbioru
3 - W realizacji
4 – Przyjęty

Komunikat POST:/api/zlecenie

Tworzy zlecenie serwisowe.

Zapytanie dostępne dla zalogowanych użytkowników.
Oczekiwany parametr [FromBody]:

```
{
  "pracownikID": 0,
  "klientID": 0,
  "nazwaRoweru": "string",
  "dataPrzyjecia": "2022-07-05T17:33:37.096Z",
  "dataOdbioru": "2022-07-05T17:33:37.096Z",
  "cenaBrutto": 0,
  "metodPlatnosciID": 0
}
```

, W przypadku poprawnego dodania, metoda zwraca strukturę zlecenia.

Komunikat PUT:/api/zlecenie/1

Wykonuje update na zleceniu 1.
Zapytanie dostępne dla zalogowanych użytkowników.
Oczekiwany parametr [FromBody]:

```
{
  "pracownikID": 0,
  "dataOdbioru": "2022-07-05T17:45:37.440Z",
  "cenaBrutto": 0,
  "metodPlatnosciID": 0,
  "statusID": 0
}
```

, W przypadku poprawnego update, metoda zwraca status 200 (OK).
Na każdym elemencie JSON można wykonać osobny update. Nieobowiązkowo przesyłać całą strukturę.

Komunikat DELETE:/api/zlecenie/1

Usuwa zlecenie o id 1.
Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.
Zapytanie zwraca status 204 (No content)

UslugaController

Komunikat GET:/api/usluga

Zwraca wszystkie usługi świadczone przez firmę.
Zapytanie dostępne dla wszystkich użytkowników.

Komunikat GET:/api/usluga/1

Zwraca usługę o id 1.
Zapytanie dostępne dla zalogowanych użytkowników.
Zwraca strukturę:

```
{
    "id": 1,
    "nazwa": "Montaż roweru",
    "cenaNetto": 119.99,
    "podatekVAT": 0.23
```

```
}
```

Komunikat POST:/api/usluga

Tworzy usługę dostępną w serwisie.
Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.
Oczekiwany parametr [FromBody]:

```
{
    "nazwa": "Testowa usługa",
    "cenaNetto": 99.99,
    "podatekVAT": 0.23
}
```

, W przypadku poprawnego dodania, metoda zwraca strukturę usługi.

Komunikat PUT:/api/usluga/1

Wykonuje update na usłudze 1.
Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.
Oczekiwany parametr [FromBody]:

```
{
    "nazwa": "Nowa nazwa usługi",
    "cenaNetto": 100.00,
    "podatekVAT": 0.23
}
```

, W przypadku poprawnego update, metoda zwraca status 200 (OK).
Na każdym elemencie JSON można wykonać osobny update. Nieobowiązkowo przesyłać całą strukturę.

Komunikat DELETE:/api/usluga/1

Usuwa usługę o id 1.
Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.
Zapytanie zwraca status 204 (No content)

KlientController

Komunikat GET:/api/klient

Zwraca listę dodanych klientów.
Zapytanie dostępne dla zalogowanych użytkowników.

Komunikat GET:/api/klient/1

Zwraca dane klienta o id 1.
Zapytanie dostępne dla zalogowanych użytkowników.
Zwraca strukturę:

```
{
    "id": 1,
    "imie": "Daniel",
    "nazwisko": "Sydorchuk",
    "nrTelefonu": "123123123",
    "email": "mail@gmail.com",
    "nip": "1231231233",
    "firma": "Firma",
```

```
    "miasto": "Kraków",
    "adres": "Ludwinowska",
    "kodPocztowy": "30-331"
}
```

Komunikat POST:/api/klient

Dodaje klienta do bazy.
Zapytanie dostępne dla zalogowanych użytkowników.
Oczekiwany parametr [FromBody]:

```
{
    "imie": "Mariusz",
    "nazwisko": "Kowalczyk",
    "nrTelefonu": "123123123",
    "email": "mail@gmail.com",
    "nip": "1231231233",
    "firma": "Firma",
    "miasto": "Kraków",
    "adres": "Bagatela",
    "kodPocztowy": "30-123"
}
```

, W przypadku poprawnego dodania metoda zwraca strukturę.

Komunikat PUT:/api/klient/1

Wykonuje update danych klienta o id 1.
Zapytanie dostępne dla zalogowanych użytkowników.
Oczekiwany parametr [FromBody]:

```
{
    "imie": "Mariusz",
    "nazwisko": "Kowalczyk",
    "nrTelefonu": "123123123",
    "email": "mail@gmail.com",
    "nip": "1231231233",
    "firma": "Firma",
    "miasto": "Kraków",
    "adres": "Bagatela",
    "kodPocztowy": "30-123"
}
```

, W przypadku poprawnego update metoda zwraca status 200 (OK).
Na każdym elemencie JSON można wykonać osobny update. Nieobowiązkowo przesyłać całą strukturę.

Komunikat DELETE:/api/klient/1

Usuwa klienta o id 1.
Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.
Zapytanie zwraca status 204 (No content)

PracownikController

Komunikat GET:/api/pracownik

Zwraca listę dodanych pracowników.

Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.

Komunikat GET:/api/pracownik/1

Zwraca dane pracownika o id 1.

Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.

Zwraca strukturę:


```
{
    "id": 1,
    "imie": "Dariusz",
    "nazwisko": "Kowalski",
    "pesel": "12312312312",
    "nrTelefonu": "123123123",
    "email": "poczta@gmail.com",
    "dataRozpoczecia": "2022-07-03T13:31:08.7227994",
    "dataZakonczenia": null
}
```


Komunikat POST:/api/pracownik

Dodaje pracownika do bazy.

Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.

Oczekiwany parametr [FromBody]:


```
{
    "imie": "Dariusz",
    "nazwisko": "Kowalski",
    "pesel": "12312312312",
    "nrTelefonu": "123123123",
    "email": "poczta@gmail.com"
}
```


, W przypadku poprawnego dodania metoda zwraca strukturę.

Komunikat PUT:/api/pracownik/1

Wykonuje update danych pracownika o id 1.

Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.

Oczekiwany parametr [FromBody]:


```
{
    "nrTelefonu": "123123123",
    "email": "poczta@gmail.com",
    "dataZakonczenia": null
}
```


, W przypadku poprawnego update metoda zwraca status 200 (OK).

Na każdym elemencie JSON można wykonać osobny update. Nieobowiązkowo przesyłać całą strukturę.

Komunikat DELETE:/api/pracownik/1

Usuwa pracownika o id 1.

Zapytanie dostępne dla zalogowanych użytkowników z rolą Admin.

Zapytanie zwraca status 204 (No content)

Dane do logowania na Gmail:
Mail: serwis.rowerowy.api@gmail.com
Hasło: serwis.rowerowy.2022