

Monika Wereszczyńska i Karolina Siwiak

I Informatyka

Magisterskie

Gr.3

Przedmiot: Projektowanie Aplikacji Internetowych i Rozproszonych

Projekt: Mikroserwis rezerwacji pokoi w hotelu

Rok akademicki 2020-21

Semestr letni

1. Koncepcja projektu:

Koncepcją naszego projektu jest mikroservis rezerwacji pokoju w hotelu.

Za jego pomocą będzie można sprawdzić:

- jakie pokoje są dostępne w konkretnym przedziale czasowym,
- dokonać rezerwacji pokoju,
- anulować rezerwację,
- dokonać edycji rezerwacji,
- sfinalizować rezerwację - zapłacić za wynajęcie pokoju przez gościa.

Mikroservis będzie umożliwiał także wykonywanie operacji CRUD na poszczególnych pokojach w hotelu:

- dodanie pokoju,
- edycję danych pokoju,
- usunięcie pokoju z oferty hotelu.

Operacje CRUD będą mogły być przeprowadzane na poszczególnych rezerwacjach:

- zarezerwuj pokój,
- edytuj rezerwację,
- anuluj rezerwację,
- zapłać za rezerwację - tworzymy rachunek do danej rezerwacji (operacja CREATE obiekt Rachunek).

Użytkownikami mikroservisu są pracownicy hotelu oraz klienci.

Pracownik hotelu będzie miał dostęp do wszystkich rachunków w bazie, rezerwacji, klientów i ich rezerwacji, a także do współpracowników i dokonanych przez nich rezerwacji.

2. Wymagania funkcjonalne

| Nazwa wymagania | Aktorzy | Opis wymagania |
|-----------------------|---------------------|--|
| Znajdź wolny pokój | Pracownik Klient | Sprawdzenie dostępności pokoi - zostanie zwrócona lista wolnych pokoi w określonym przedziale czasu. |
| Zarezerwuj pokój | Pracownik Klient | Rezerwacja pokoju na określony czas. |
| Edytuj rezerwację | Pracownik Klient | Edycja istniejącej rezerwacji - zmiana odstępu czasowego. |
| Anuluj rezerwację | Pracownik Klient | Anulowanie istniejącej rezerwacji. |
| Sfinalizuj rezerwację | Pracownik Klient | Zapłacenie za rezerwację - podsumowanie całego rachunku. |

| | | |
|------------------------------|---------------------|--|
| Lista pokoi | Pracownik Klient | Wyświetlanie wszystkich pokoi w mikroserwisie. |
| Dodaj pokój | Pracownik | Dodanie nowego pokoju do oferty hotelu. |
| Edytuj pokój | Pracownik | Edycja istniejącego pokoju. |
| Usuń pokój | Pracownik | Usunięcie istniejącego hotelu z oferty hotelu. |
| Twoje rezerwacje | Pracownik Klient | Zostaną wyświetlone wszystkie rezerwacje użytkownika. Pracownikowi zostanie wyświetlona lista rezerwacji, których dokonał osobiście w imieniu klienta. |
| Wszystkie rezerwacje w bazie | Pracownik | Zostaną wyświetlone wszystkie rezerwacje w mikroserwisie. |
| Wszystkie rachunki w bazie | Pracownik | Zostaną wyświetlone wszystkie rachunki w mikroserwisie. |
| Wszyscy klienci w bazie | Pracownik | Zostaną wyświetleni wszyscy klienci w mikroserwisie. |
| Wszyscy pracownicy w bazie | Pracownik | Zostaną wyświetleni wszyscy pracownicy w mikroserwisie. |
| Rezerwacje danego klienta | Pracownik | Zostaną wyświetlone wszystkie rezerwacje danego klienta. |
| Rezerwacje danego pracownika | Pracownik | Zostaną wyświetlone wszystkie rezerwacje, wykonane przez danego pracownika. |
| Rejestracja użytkownika | Przyszły klient | Tworzenie konta w mikroserwisie. |

3. Wymagania niefunkcjonalne

| |
|--|
| Klient do rejestracji potrzebuje swojego niepowtarzalnego e-maila i hasła. |
| Logowanie do systemu za pomocą e-maila i hasła. |
| Interfejsem systemu jest strona www. |
| System jest dostępny w języku polskim. |

4. Użyte technologie w projekcie

| | |
|---------------------------------|---|
| Baza danych | MongoDB |
| Pobieranie danych z bazy danych | C# .NET API |
| Interfejs | React |
| Protokół komunikacji | HTTP |
| Przewidywane żądania: | |
| GET | <ul style="list-style-type: none">• Get(Pokoje),• GetPokoje,• getVacancy,• getRachunek,• getRezerwacja(All),• getRezerwacje(Usera),• GetRezerwacjeDetails(Usera),• GetKlienci,• GetPracownicy,• GetUser. |
| POST | <ul style="list-style-type: none">• Create(Pokoje),• Create(Rezerwacja). |
| PUT | <ul style="list-style-type: none">• Update(Pokoje),• MakeBill(Rachunek),• Cancel(Rezerwacja),• Edit(Rezerwacja),• Edit(User). |
| DELETE | <ul style="list-style-type: none">• Delete(Pokoje). |

5. Przewidywana struktura bazy danych

| Nazwa tabeli | Pola w tabeli |
|--------------|---|
| Pokoje | <ul style="list-style-type: none">• id,• numer pokoju,• nazwa pokoju,• ile osób w pokoju,• cena za noc. |
| Rezerwacje | <ul style="list-style-type: none">• id,• id pokoju,• id klienta,• id rezerwującego (może być sam klient lub przez pracownika),• data rozpoczęcia pobytu,• data zakończenia pobytu,• przewidywany koszt,• czy rezerwacja była edytowana,• data ostatniej edycji,• czy anulowana,• czyOpłacona. |
| Rachunki | <ul style="list-style-type: none">• id,• id rezerwacji,• data zakończenia pobytu,• koszt rezerwacji,• czy uregulowany,• data uregulowania rachunku. |
| Klienci | <ul style="list-style-type: none">• id,• nazwisko,• imię,• numer telefonu,• e-mail. |
| Pracownicy | <ul style="list-style-type: none">• id,• nazwisko,• imię,• numer telefonu,• e-mail. |

6. Sposób logowania i zarządzania użytkownikami

Każdy klient będzie miał możliwość stworzenia swojego konta i wykonywania akcji, dostępnych dla klienta - przy wykonaniu 1 rezerwacji, jego dane zostaną zapisane w bazie MongoDB.

Każdy pracownik będzie miał możliwość utworzenia konta, ale musi on wybrać e-mail z bazy danych Pracownicy.

7. Podział prac

| Czynność | Wykonawca |
|---|---------------|
| Wypełnienie baz danych | Wereszczyńska |
| Wygenerowanie projektu .NET API | Wereszczyńska |
| Wykonanie interfejsów w REACT | Siwiak |
| Integracja MongoDB, .NET API oraz REACT | Siwiak |
| Implementacja kontrolerów | Wereszczyńska |
| Implementacja serwisów | Siwiak |

8. Opis implementacji

Projekt został zaimplementowany w środowisku Visual Studio.

W pliku appsettings.json zostały określone źródła danych, wykorzystywanych przez mikroserwis:

- Microsoft SQL Local Database - przechowuje dane użytkowników do logowania (linia 2 - 4),
- MongoDB (linia 5 - 12) - przechowuje informacje o :
 - klientach,
 - pracownikach,
 - rezerwacjach,
 - rachunkach,
 - pokojach.

Poniższy fragment kodu pochodzi z pliku, wspomnianego powyżej.

```
1. {
2.   "ConnectionStrings": {
3.     "DefaultConnection":
4.       "Server=(localdb)\\mssqllocaldb;Database=CoreDB;Trusted_Connection=True;MultipleActiveResultSets=true"
5.   },
6.   "MongoDBDatabaseSettings": {
7.     "PokojeCollectionName": "pokoje",
8.     "RezerwacjaCollectionName": "rezerwacje",
```

```

8.     "RachunekCollectionName": "rachunki",
9.     "KlientCollectionName": "klienci",
10.    "PracownikCollectionName": "pracownicy",
11.    "ConnectionString": "mongodb://localhost:27017",
12.    "DatabaseName": "bazaDanych"
13.    }

```

Poniższa klasa MongoDBDatabaseSettings ułatwia pobieranie informacji do połączenia z bazą MongoDB i kolekcji, wewnątrz bazy.

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Threading.Tasks;
5.
6. namespace ProjectMongoDBReact.Models
7. {
8.     public class MongoDBDatabaseSettings :
9.         IMongoDBDatabaseSettings
10.    {
11.        public string PokojCollectionName { get; set; }
12.        public string RezerwacjaCollectionName { get; set; }
13.        public string RachunekCollectionName { get; set; }
14.
15.        public string KlientCollectionName { get; set; }
16.        public string PracownikCollectionName { get; set; }
17.
18.        public string ConnectionString { get; set; }
19.        public string DatabaseName { get; set; }
20.    }
21.
22.    public interface IMongoDBDatabaseSettings
23.    {
24.        string PokojCollectionName { get; set; }
25.        string RezerwacjaCollectionName { get; set; }
26.        string RachunekCollectionName { get; set; }
27.
28.        string KlientCollectionName { get; set; }
29.        string PracownikCollectionName { get; set; }
30.
31.        string ConnectionString { get; set; }
32.        string DatabaseName { get; set; }
33.    }

```

Poniższy kod to fragment kodu klasy KlientService, której metody pobierają informacje z bazy danych MongoDB.

```
1.     private readonly IMongoCollection<Klient> _klient;
2.         private readonly IMongoCollection<Pracownik> _pracownik;
3.
4.     public KlientService(IMongoDBDatabaseSettings settings)
5.     {
6.         var client = new
MongoClient(settings.ConnectionString);
7.         var database =
client.GetDatabase(settings.DatabaseName);
8.
9.         _klient =
database.GetCollection<Klient>(settings.KlientCollectionName);
10.        _pracownik =
database.GetCollection<Pracownik>(settings.PracownikCollectionNam
e);
11.    }
```

Poniższa metoda z klasy KlientService edytuje istniejące rekordy w bazie danych MongoDB w kolekcji "klienci".

```
1. public void Edit(string id, Klient klient)
2. {
3.     _klient.ReplaceOne(p => p.Id == id, klient);
4. }
```

Poniższy kod przedstawia konstruktor w kontrolerze UserController - zadeklarowane zostają w nim obiekty KlientService i PracownikService, które poprzez właściwe metody tych obiektów, zwracają odpowiednie dane z MongoDB.

```
1. private readonly KlientService _klientService;
2.     private readonly PracownikService _pracownikService;
3.
4.     public UserController(KlientService klientService,
PracownikService pracownikService)
5.     {
6.         _klientService = klientService;
7.         _pracownikService = pracownikService;
8.     }
```

Poniższy kod przedstawia metodę w kontrolerze UserController. Poprzez klasę _pracownikService oraz _klientService kontroler pobiera dane z MongoDB.


```

1. [HttpGet(Name = "GetUser")]
2.     [Route("GetUser/{name}")]
3.     public object GetUser(string name)
4.     {
5.         var p = _pracownikService.GetUser(name);
6.
7.         if (p == null)
8.         {
9.             var k = _klientService.GetUser(name);
10.
11.             if (k == null)
12.             {
13.                 return null;
14.             }
15.
16.             return k;
17.         }
18.
19.         return p;
20.     }

```

Do aplikacji REACT, żądania HTTP są wysyłane przez AXIOS i FETCH.

Poniższa metoda z pliku Pokoje.js - za pomocą FETCH - wysyła żądanie HTTP GET do kontrolera, aby otrzymać listę pokoi. Autoryzacja użytkownika odbywa się poprzez token.

```

1. async populatePokojeData() {
2.     const token = await authService.getAccessToken();
3.     const response = await fetch('pokoje', {
4.         headers: !token ? {} : { 'Authorization': `Bearer ${token}` }
5.     });
6.     const data = await response.json();
7.     this.setState({ pokoje: data, loading: false });
8. }

```

Poniższy fragment kodu z pliku Pokoje.js przedstawia wysyłanie żądania HTTP DELETE za pomocą AXIOS - ma ona doprowadzić do usunięcia pokoju z bazy danych. Autoryzacja użytkownika odbywa się poprzez token.

```

1. axios.delete(`/pokoje/Delete/` + idPokoje + '/' + user.name,
2.     {
3.         headers: {
4.             Authorization: `Bearer ${token}`
5.         },
6.         data: {

```

```
7.         'id': idPokoj
8.     }
9.
10.    }).then(window.location.reload());
```