Projekt aplikacji

Software Architecture Design (SAD)

Spis treści

1. Wstęp

Dokument opisuje architekturę aplikacji do zarządzania placówką medyczną, systemu umożliwiającego użytkownikom zarejestrowanie i przeprowadzenie wizyty lekarskiej. Aplikacja pozwala na dodawanie nowych pacjentów do bazy danych, do wyszukiwania już stałych pacjentów, na rejestracje wizyty do wybranego lekarza, na prowadzenie wizyty przez lekarza oraz na wystawianie skierowań.

2. Cel i zakres dokumentu

Celem dokumentu jest przedstawienie szczegółowej architektury aplikacji BookSwap, w tym logicznego projektu aplikacji, struktury bazy danych oraz interakcji między komponentami systemu. Zakres dokumentu obejmuje projekt architektury oprogramowania, projekt interfejsóużytkownika, oraz struktury danych.

3. Definicja architektury aplikacji

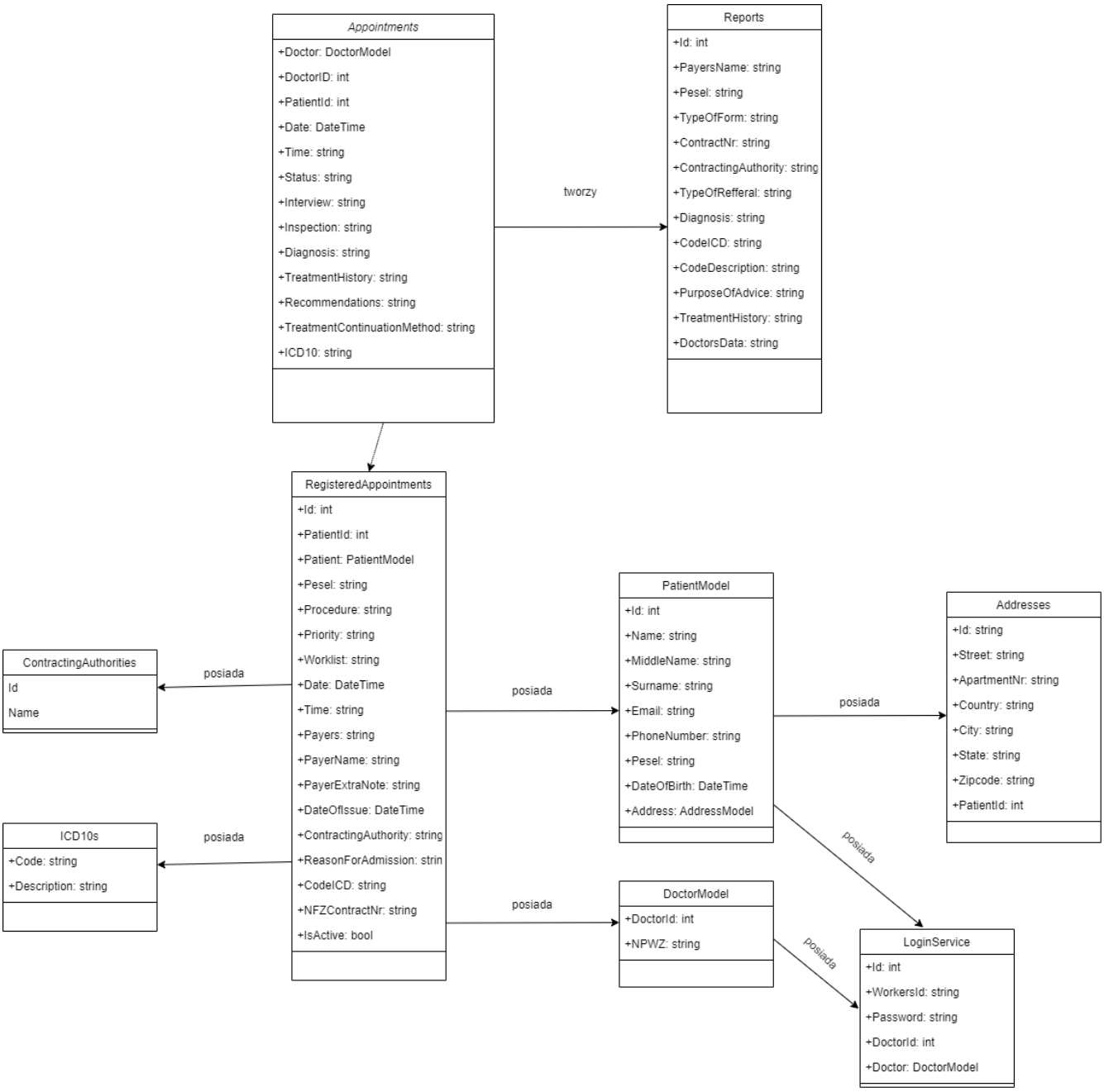
Aplikacja została napisana w języku programowania C#. Interfejs aplikacji został zaimplementowany z zastosowaniem technologii WPF oraz z wykorzystaniem wzorca projektowego MVVM. Baza danych została stworzona poprzez wykorzystanie platformy Entity Framework Core. Pomiędzy dwoma warstwami aplikacji zostało też stworzone Web API do transferu I modyfikacji danych podczas korzystania z aplikacji.

4. Cele i ograniczenia architektury

Celem architektury jest zapewnienie bezpieczeństwa i łatwości użytkowania. Ograniczenia to między innymi zapewnienie ochrony danych osobowych oraz zapewnienie wysokiej dostępności systemu.

5. Obraz logiczny aplikacji

5.1 Diagram klas aplikacji



5.2 Specyfikacja funkcji i metod aplikacji

Każdy model widoku posiada zestaw funkcji do pobierania, zapisywania i modyfikacji danych, zadaniem których jest robienie żądań na odpowiednie endpoint'y. Jak na listngu poniżej można zobaczyć przykład metody do ładowania wszystkich wizyt do głównego widoku aplikacji. Każda metoda CRUD danej aplikacji jest zdefiniowana w podobny sposób.

```
public async Task LoadAppointmentsAsync()
{
    try
    {
        var response = await httpClient.GetAsync("api/registeredAppointment");
        response.EnsureSuccessStatusCode();

        var appointmentsFromApi = await response.Content.ReadAsAsync<List<RegisteredAppointment>>();

        var registrationModel = appointmentsFromApi.Select(r => new RegistrationModel
        {
            Id = r.Id,
            PatientId = r.PatientId,
            PayerName = r.PayerName,
            Pesel = r.Pesel,
            Worklist = r.Worklist,
            Date = (DateTime)r.Date,
            Time = r.Time,
            Procedure = r.Procedure,
            Priority = r.Priority,
            ContractingAuthorities = r.ContractingAuthorities,
            DateOfIssue = ((DateTime)r.DateOfIssue).Date,
            ReasonForAdmission = r.ReasonForAdmission,
            CodeICD = r.CodeICD,
            NFZContractNr = r.NFZContractNr,
            IsActive =(bool) r.IsActive
        }).ToList();

        var filteredAppointments = registrationModel.Where(a => a.IsActive == true);

        RegisteredAppointments = new ObservableCollection<RegistrationModel>(filteredAppointments);




    }
    catch (Exception ex)
    {
        MessageBox.Show("An error occurred: " + ex.Message);
    }
}

public async Task UpdateAppointmentAsync(int appId, RegisteredAppointment appointment)
{
    try
    {
        var jsonAppointment = JsonConvert.SerializeObject(appointment);
        var content = new StringContent(jsonAppointment, Encoding.UTF8, "application/json");
        var apiUrl = $"https://localhost:7057/api/registeredAppointment/api/registeredAppointment/
{appId}";
        HttpResponseMessage response = await httpClient.PutAsync(apiUrl, content);
        if (response.IsSuccessStatusCode)
        {
            string responseBody = await response.Content.ReadAsStringAsync();
            MessageBox.Show("Updated");
            await LoadAppointmentsAsync();
        }
        else
        {
            MessageBox.Show("API call failed. Status code: " + response.StatusCode);
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show("An error occurred: " + ex.Message);
    }
}
```

6. Dynamiczny obraz modelowanej aplikacji

6.1 Diagram sekwencji UML dla obiektów

6.2 Diagram aktywności UML dla obiektów

Diagram aktywności logowanie.

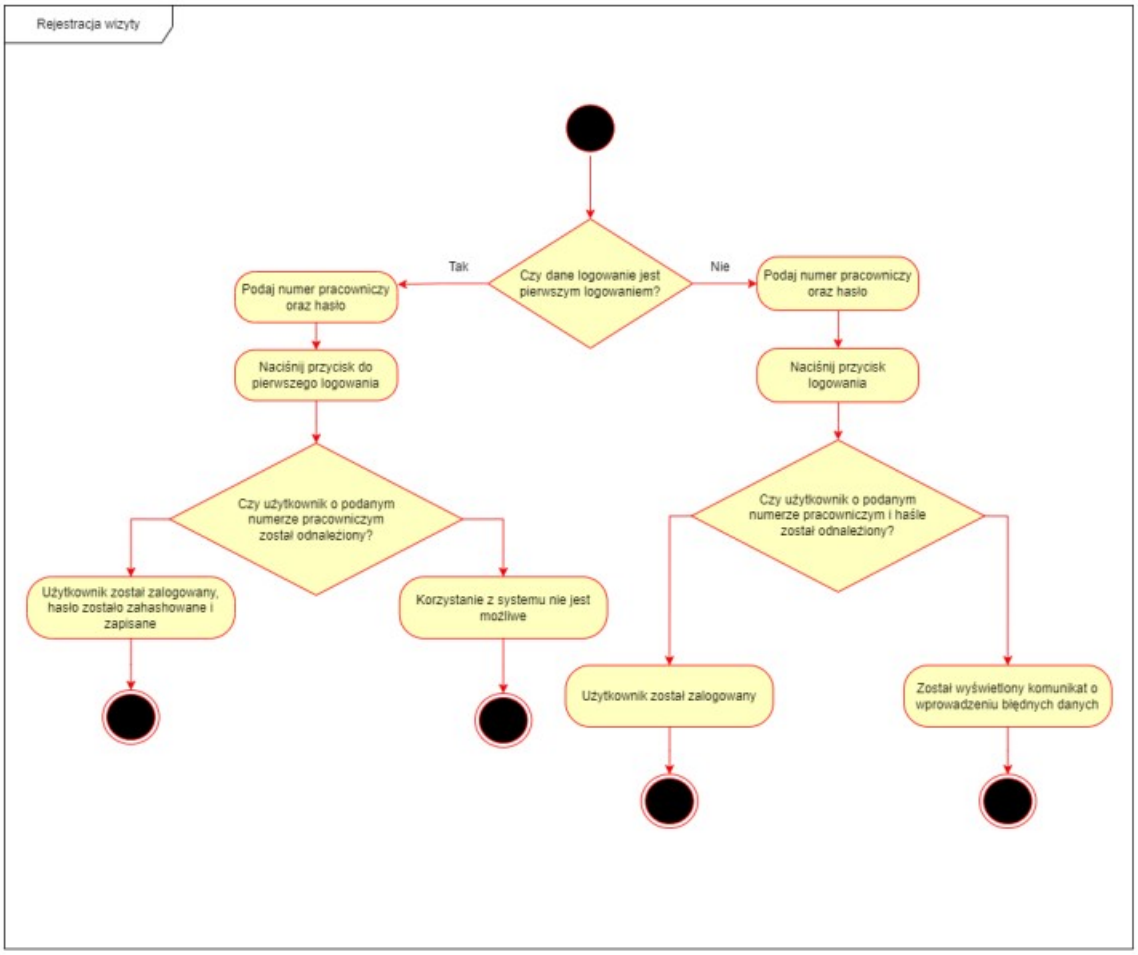


Diagram aktywności rejestracja wizyty.

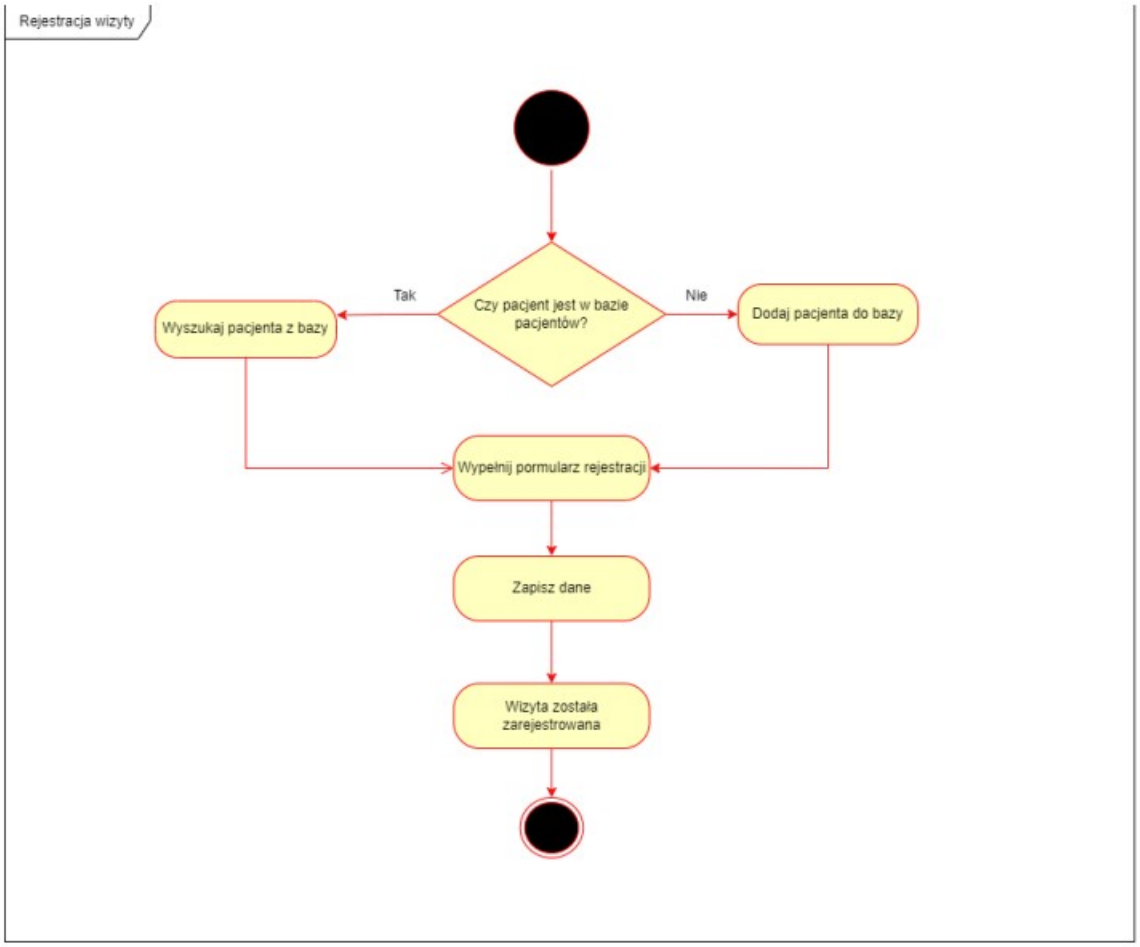


Diagram aktywności wypełnienie formularza wizyty.

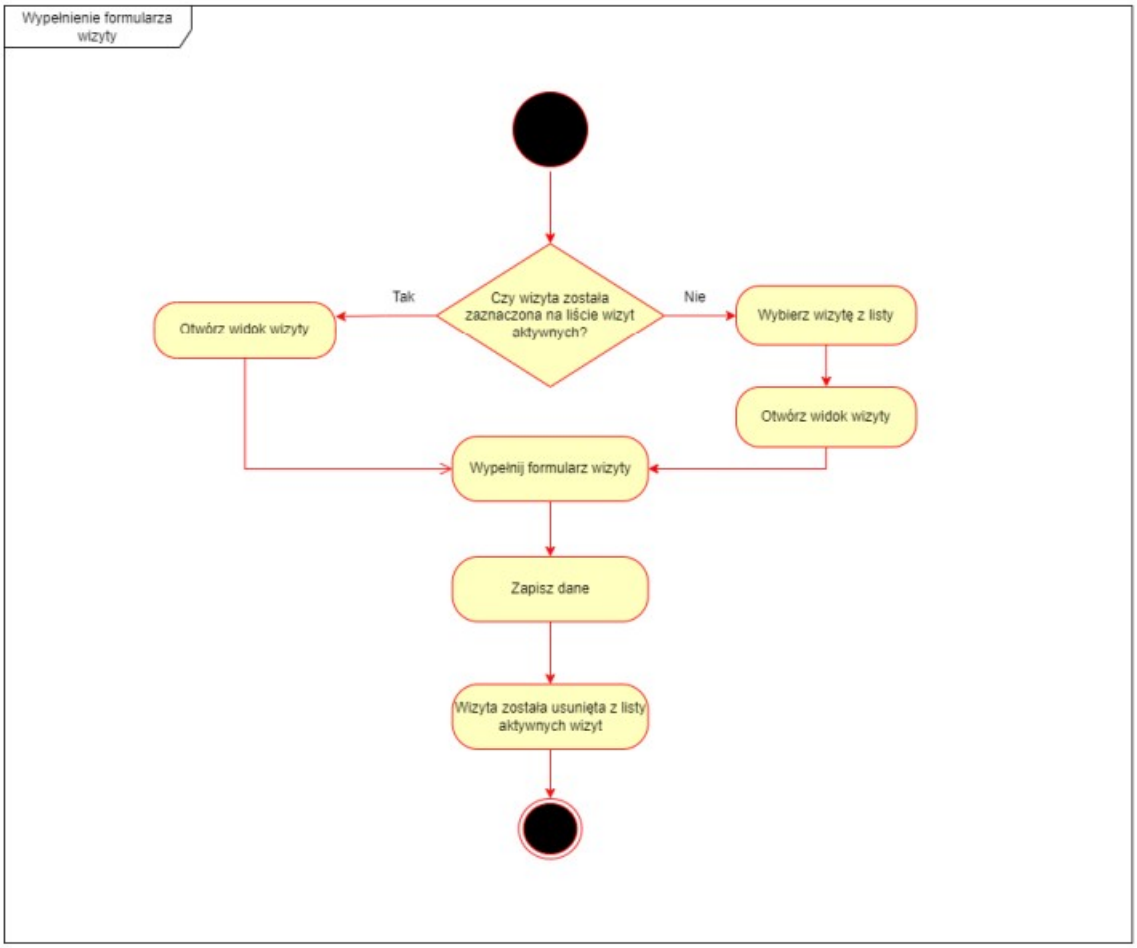
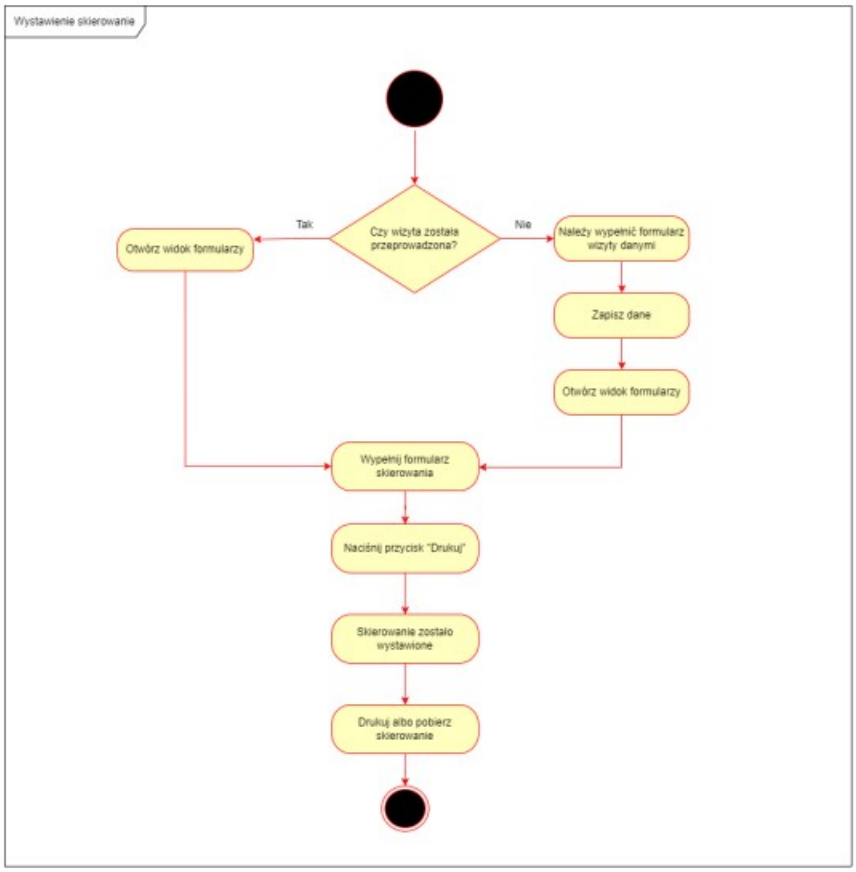


Diagram aktywności wystawienie skierowania.



7. Projekt bazy danych