
# Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

Raport końcowy z przedmiotu TAB z projektu pt.

## „Aquapark”

**Skład sekcji:**

Leszek Kaliściak – lider

Jakub Antonowicz

Kamil Ceglarski

Bartosz Malinowski

Artur Szaboń

# 1. Treść zadania

Projekt AquaparkApp ma na celu stworzenie kompleksowego systemu informatycznego, który wspiera codzienną pracę obiektów rekreacyjnych typu aquapark. Zadanie obejmuje:

- Uwierzytelnianie i rejestracja użytkowników – implementację bezpiecznego procesu rejestracji nowych klientów, weryfikację adresu e-mail oraz mechanizm logowania z opcją dwuetapowej autoryzacji.
- Prezentację oferty – opracowanie interfejsu pozwalającego na przeglądanie:
    - dostępnych atrakcji (z opisami, limitami uczestników oraz dodatkowymi opłatami),
    - cen biletów i karnetów wraz z obowiązującymi promocjami i zniżkami,
    - prognozy pogody na stronie Weather, co wpływa na planowanie wizyt.
- Obsługę koszyka zakupowego – zapewnienie logiki wyboru ofert, naliczania zniżek, obliczania końcowej ceny oraz wygenerowanie pozycji płatności.
- Proces płatności – integrację prostego formularza płatności (symulacja, bez rzeczywistego połączenia z systemem zewnętrznym), rejestrację transakcji i jej pozycji w bazie.
- Kontrolę dostępu i rejestrację wizyt – przydzielanie opaski RFID, symulacja bramek wejściowych/wyjściowych, rejestracja eventów w systemie i monitorowanie czasu pobytu.
- Zarządzanie danymi – panel Sprzedawcy i Administratora umożliwiający:
    - CRUD klientów, ofert cenowych, opasek, przeglądanie logów dostępu, historii wizyt oraz naliczonych kar,
    - generowanie raportów podstawowych statystyk (np. liczba wizyt dziennie).
- Architekturę warstwową i testowalność – rozdzielenie warstwy prezentacji (Blazor), logiki biznesowej (serwisy), dostępu do danych (EF Core) oraz modeli DTO.

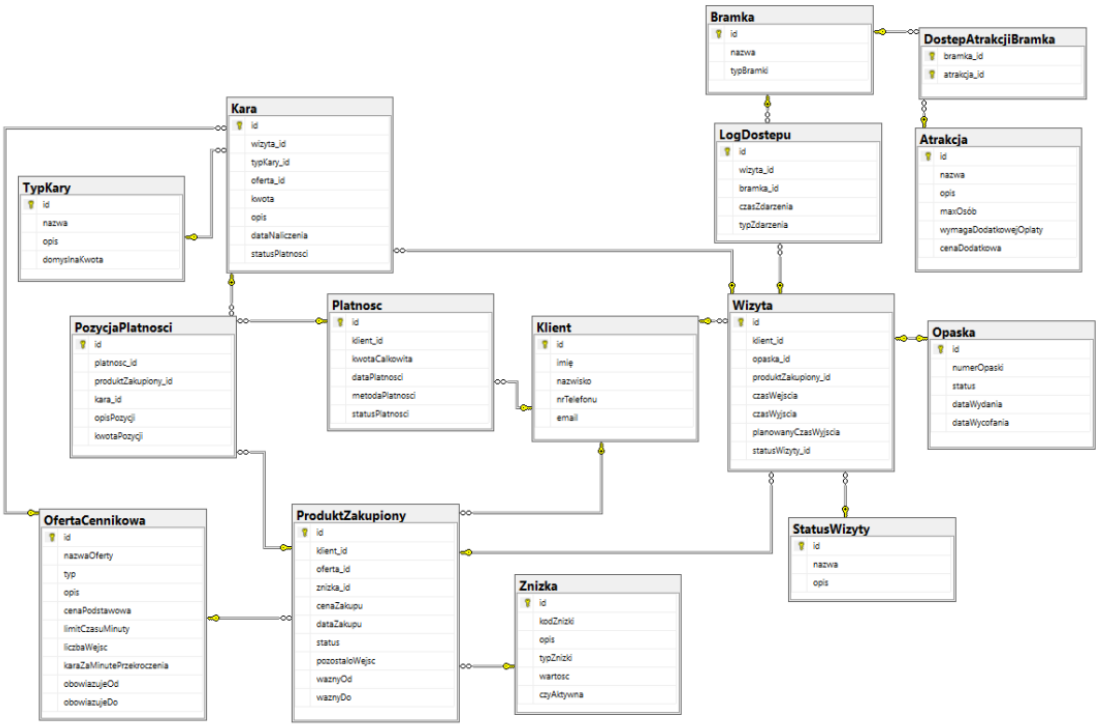
Celem jest nie tylko funkcjonalne odwzorowanie procesów operacyjnych aquaparku, lecz także zapewnienie skalowalności, bezpieczeństwa i czytelnej architektury, która ułatwi dalszy rozwój systemu.

# 2. Analiza i omówienie

W fazie analizy przeprowadzono szczegółowe badanie wymagań funkcjonalnych i niefunkcjonalnych, co pozwoliło na wyodrębnienie głównych obszarów systemu:

- Moduł autoryzacji i zarządzania kontem:
    - Wymagania: bezpieczna rejestracja, logowanie, zmiana i reset hasła, zarządzanie danymi osobowymi.

    - Rozwiązanie: wykorzystanie ASP.NET Core Identity z rozszerzeniami dla dwuskładnikowego uwierzytelniania i potwierdzania e-mail.
- Prezentacja oferty i cennika:
    - Wymagania: wyświetlanie listy atrakcji z opisami oraz parametrów (maksymalna liczba osób, dodatkowe opłaty), przeglądanie cennika z obowiązującymi promocjami i rabatami.
    - Rozwiązanie: strony Blazor (Atrakcje.razor, Cennik.razor), dynamiczne ładowanie danych z serwisu IAtrakcjaService i IOfertaCennikowaService.
- Obsługa koszyka i procesu zakupowego:
    - Wymagania: dodawanie ofert do koszyka, naliczanie zniżek, obliczanie ceny końcowej, symulacja płatności.
    - Rozwiązanie: wzorzec DTO (PozycjaKoszykaDto), serwis TransakcjaService, automatyczne zapisywanie transakcji w tabelach Platnosc i PozycjaPlatnosci.
- Zarządzanie wizytami i kontrola dostępu:
    - Wymagania: przypisywanie opasek identyfikacyjnych (RFID), rejestrowanie zdarzeń przejścia przez bramki, monitorowanie czasu pobytu, naliczanie kar za przekroczenia czasu.
    - Rozwiązanie: serwis WizytaService oraz LogDostepu, symulacja bramek w SymulatorBramek.razor, tabele Opaska, Wizyta, LogDostepu i Kara w bazie danych.
- Panel administracyjny i raportowanie:
    - Wymagania: CRUD dla klientów, ofert, opasek oraz przeglądanie historii wizyt i logów.
    - Rozwiązanie: komponenty Blazor (ZarzadzajKlientami.razor, ZarzadzajOferta.razor, ZarzadzajOpaskami.razor), role użytkowników (Admin, Sprzedawca), dostępne pod PanelSprzedawcy.




## Omówienie rozwiązania

**Architektura warstwowa:** aplikacja podzielona na warstwy prezentacji (Blazor), serwisów (logika biznesowa) oraz dostępu do danych (EF Core), co ułatwia testowanie i rozwój.

**Bezpieczeństwo:** zastosowanie ASP.NET Core Identity zapewniło wydajne mechanizmy uwierzytelniania i autoryzacji, w tym obsługę dwuskładnikowego logowania.

**Elastyczność i skalowalność:** dzięki użyciu EF Core i migracji, schemat bazy danych można łatwo rozszerzać, a serwisy można poddawać mockowaniu w testach jednostkowych.

**Doświadczenie użytkownika:** Blazor Server umożliwia tworzenie interaktywnych stron bez przeładowania, a wspólne layouty (MainLayout, AccountLayout) dbają o spójność interfejsu.

# 3. Wybrane technologie oraz opis architektury systemu

## 3.1. Uzasadnienie doboru technologii

- **Blazor Server:** Wybrano Blazor Server, ponieważ umożliwia tworzenie nowoczesnych, interaktywnych aplikacji webowych w całości w C#, bez konieczności pisania JavaScript. Architektura SignalR zapewnia wydajną komunikację między klientem a serwerem, co skutkuje niskim opóźnieniem przy aktualizacji interfejsu. Blazor Server dobrze integruje się z ASP.NET Core oraz mechanizmami Identity, co przyspiesza implementację funkcji uwierzytelniania.
- **ASP.NET Core Identity:** Zespół zdecydował się na wbudowane rozwiązanie Identity ze względu na gotowe mechanizmy bezpiecznego zarządzania użytkownikami, hasłami oraz obsługę ról. Rozszerzalność komponentów (np. dwuskładnikowe uwierzytelnianie, potwierdzanie e-mail) umożliwiła szybkie wdrożenie funkcji bezpieczeństwa zgodnych z dobrymi praktykami.
- **.NET 8:** Stabilna, długoterminowo wspierana platforma z optymalizacjami wydajności i niskimi wymaganiami zasobów. Użycie najnowszej wersji .NET gwarantuje wsparcie dla aktualnych bibliotek i funkcji językowych (np. rekordy, wzorce matching), zwiększając czytelność i zwięzłość kodu.
- **Entity Framework Core (Code First):** Model Code First pozwala definować encje jako klasy C# i automatycznie generować schemat bazy. EF Core oferuje łatwe migracje, zapytania LINQ i śledzenie zmian obiektów. Dzięki temu zmiany w modelu danych są proste do wdrożenia, a warstwa dostępu do danych pozostaje czytelna.
- **MS SQL Server:** Relacyjna baza danych o sprawdzonej wydajności i funkcjonalnościach (transakcje, procedury składowane, indeksy). Zapewnia spójność danych i bezpieczeństwo, co jest kluczowe w systemie finansowym oraz rejestracji wizyt.
- **Dependency Injection (DI):** ASP.NET Core natywnie wspiera wstrzykiwanie zależności, co sprzyja luźnemu powiązaniu komponentów, łatwości testowania (mockowanie serwisów) i czytelności kodu.

## 3.2. Architektura systemu

System został zaprojektowany w oparciu o architekturę wielowarstwową (ang. layered architecture), co ułatwia utrzymanie, rozwój oraz testowanie aplikacji.

**Wartstwa prezentacji (UI)**

- Składa się z komponentów Blazor umieszczonych w katalogach Pages i Shared.

- Layouty (MainLayout.razor, AccountLayout.razor) definiują wspólne elementy (menu, nagłówek, stopkę).
- Routing jest skonfigurowany w Routes.razor, natomiast dostęp do stron jest kontrolowany przez autoryzację atrybutami [Authorize].

**Warstwa serwisów (Business Logic Layer)**

- Serwisy realizują operacje domenowe (rejestracja wizyty, obsługa koszyka, naliczanie kar).
- Każdy serwis implementuje interfejs (np. IAtrakcjaService), co umożliwia zamienniki w testach jednostkowych.
- Wstrzykiwanie zależności serwisów odbywa się w Program.cs przy użyciu services.AddScoped<...>().

**Warstwa dostępu do danych (Data Access Layer)**

- ApplicationDbContext dziedziczy po IdentityDbContext<ApplicationUser> i zawiera zestaw DbSet<T> dla każdej encji.
- Migracje EF Core w katalogu Migrations umożliwiają ewolucję schematu bazy bez manualnej ingerencji.
- Relacje między tabelami (1:N, N:M) odzwierciedlone zostały w konfiguracji modeli (atrybuty i Fluent API).

**DTO i mapowanie**

- Klasa PozycjaKoszykaDto służy do przenoszenia danych między UI a warstwą serwisów, odciążając bezpośrednie użycie encji EF w interfejsie.

**Bezpieczeństwo i autoryzacja**

- ASP.NET Core Identity zarządza użytkownikami i rolami.
- Dodatkowe komponenty (IdentityRevalidatingAuthenticationStateProvider) zapewniają automatyczne odświeżanie stanu uwierzytelnienia.

**Rozszerzalność**

- Dzięki jasno zdefiniowanym warstwom i interfejsom, można łatwo dodać nowe moduły (np. integracja z zewnętrznym systemem płatności, IoT do bramek RFID).

System został zaprojektowany tak, aby zapewnić czytelność kodu, łatwość testowania oraz skalowalność, co jest kluczowe przy rozwoju rozległych aplikacji bazodanowych.

# 4. Instrukcja użytkownika

## 4.1. Uruchomienie środowiska developerskiego

**Pobranie kodu źródłowego:**

git clone https://github.com/KaliPolsl/AquaparkApp.git

cd AquaparkApp

**Konfiguracja środowiska:**

Zainstaluj:

- .NET SDK 6.0 lub wyższy (zalecane .NET 8)
- MS SQL Server (lokalnie lub zdalnie)
- Visual Studio 2022 lub Visual Studio Code

**W plikach appsettings.json i appsettings.Development.json ustaw połączenie do bazy:**

"ConnectionStrings": {

"DefaultConnection":
"Server=.;Database=AquaparkDb;Trusted_Connection=True;MultipleActiveResultSets=true"

}

**Migracje bazy danych:**

Uruchom w terminalu:

dotnet ef database update

i zweryfikuj utworzenie tabel.

**Uruchomienie aplikacji:**

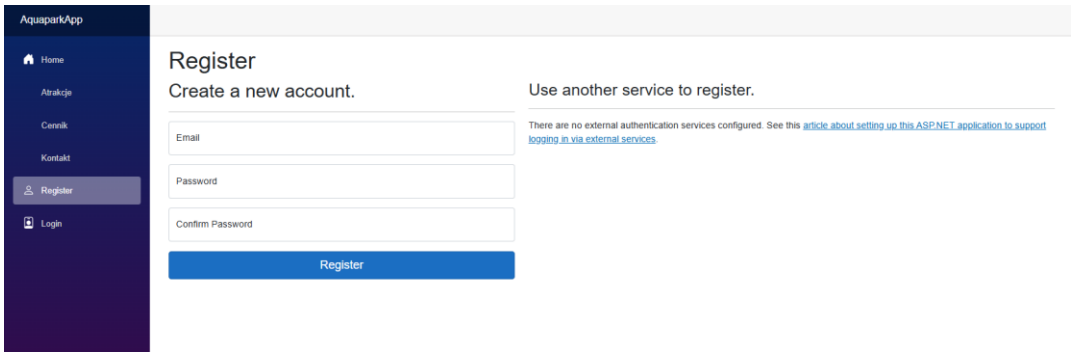
- W Visual Studio: wybierz projekt AquaparkApp jako startowy i naciśnij F5.
- Lub w terminalu:

```
dotnet run --project AquaparkApp/AquaparkApp.csproj
```

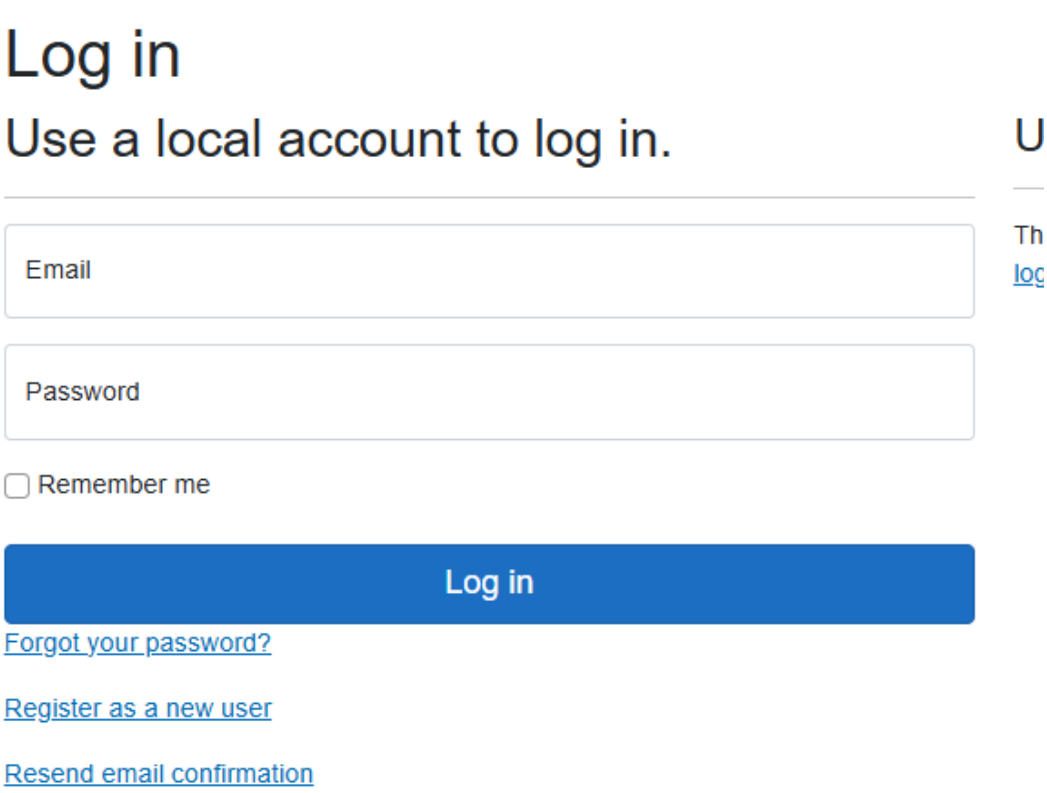
Aplikacja powinna być dostępna pod adresem https://localhost:5001.

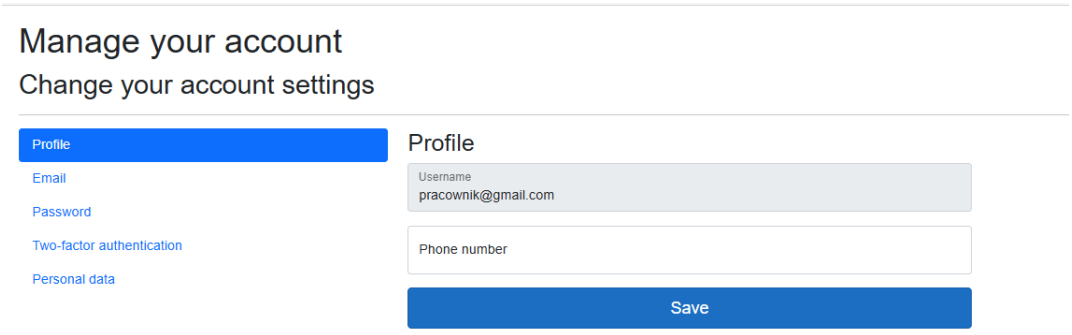
## 4.2 Instrukcja użytkowania aplikacji (wysoki poziom)

**Rejestracja i logowanie**

- Rejestracja:
  - W menu wybierz Zarejestruj się.
  - Wypełnij formularz (e-mail, hasło).
  - Zatwierdź
  - 

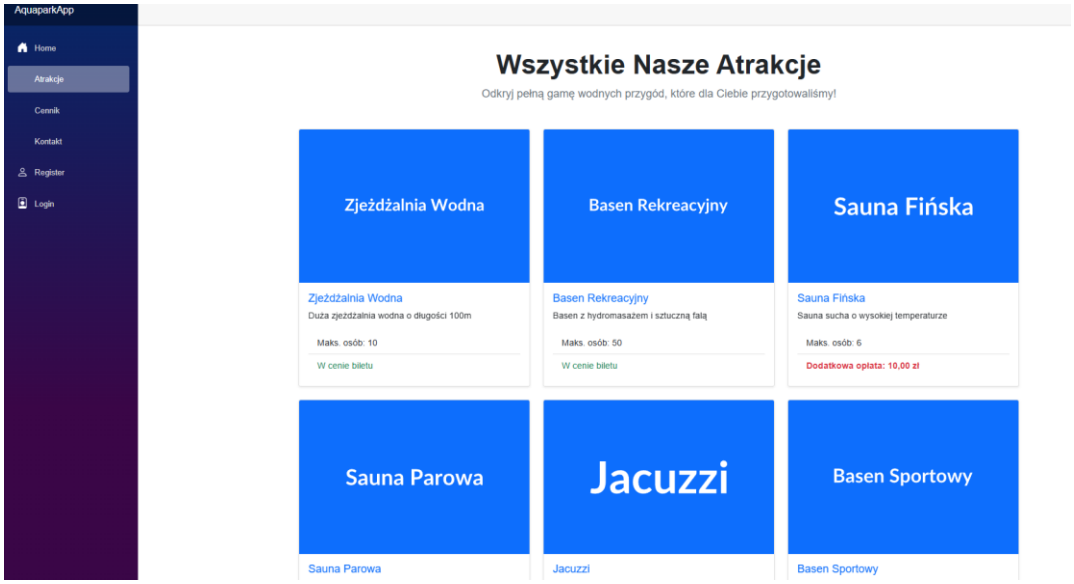
- Logowanie:
  - Wybierz Zaloguj się
  - Wpisz email i hasło
  - Zatwierdź

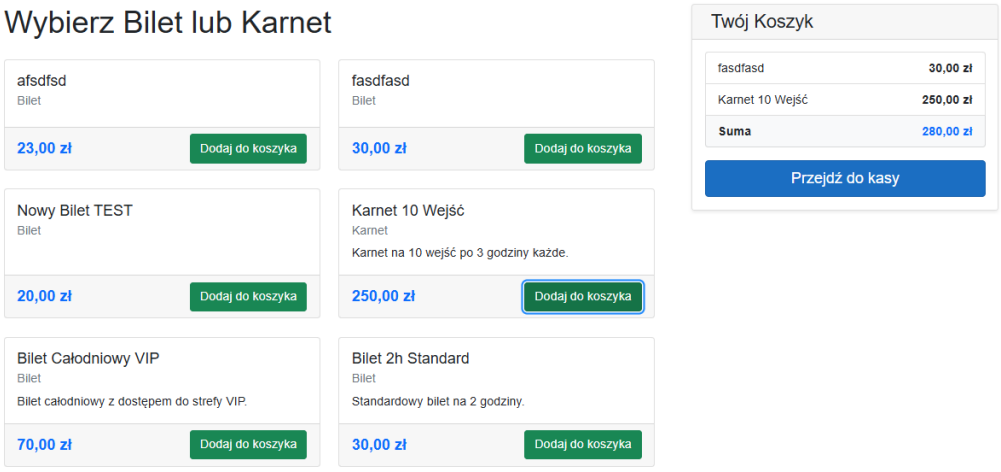


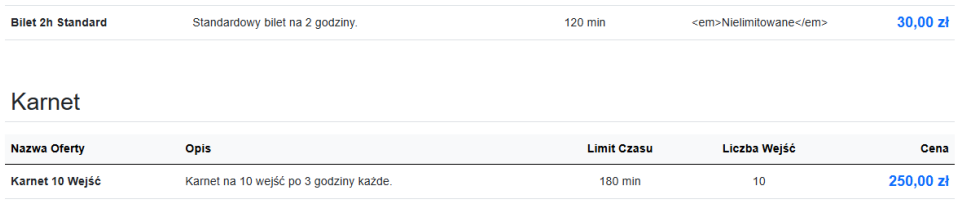

- 


**Przeglądanie oferty i cennika**

- Atrakcje:
  - Z menu głównego wybierz Atrakcje.
  - Kliknij wybraną atrakcję, aby zobaczyć szczegóły.

- 

- Cennik:
  - Przejdź do zakładki Cennik.
  - Wybierz ofertę (bilet/karnet), wskaż liczbę sztuk i kliknij Dodaj do koszyka.
  - 

  - 

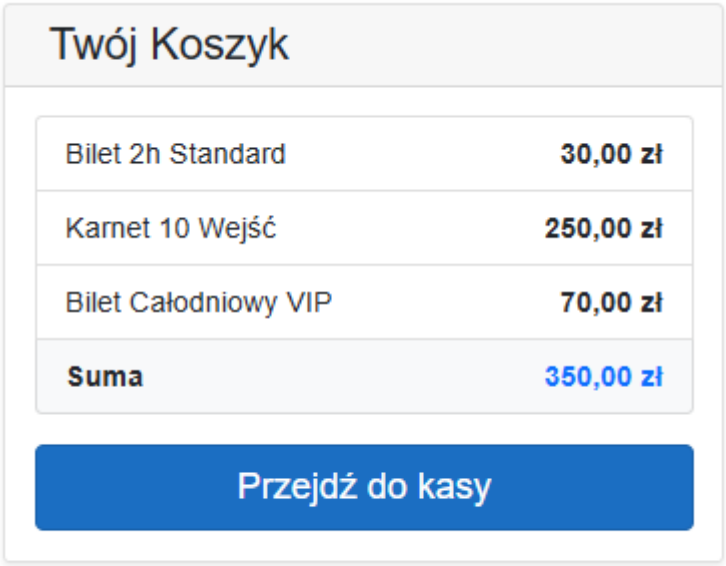
| | | | | |
|---|---|---|---|---|
| **Bilet 2h Standard** | Standardowy bilet na 2 godziny. | 120 min | Nielimitowane | 30,00 zł |

Karnet

| Nazwa Oferty | Opis | Limit Czasu | Liczba Wejść | Cena |
|---|---|---|---|---|
| **Karnet 10 Wejść** | Karnet na 10 wejść po 3 godziny każde. | 180 min | 10 | 250,00 zł |

**Obsługa koszyka i płatności**

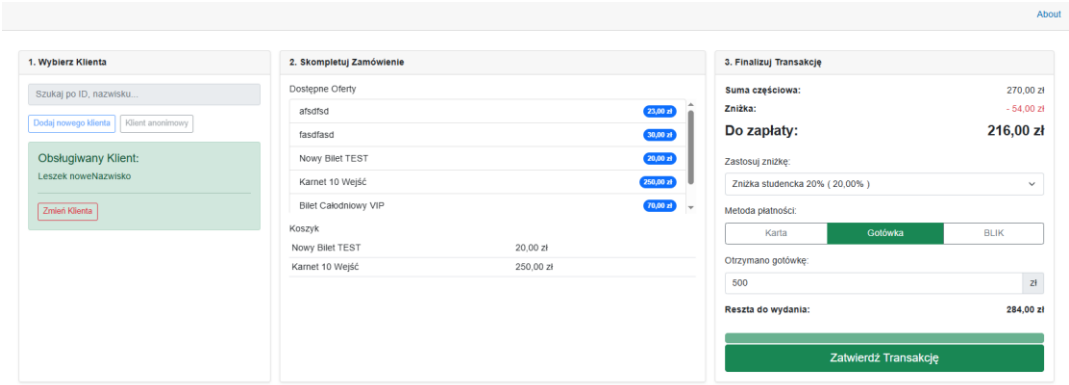
- Koszyk:

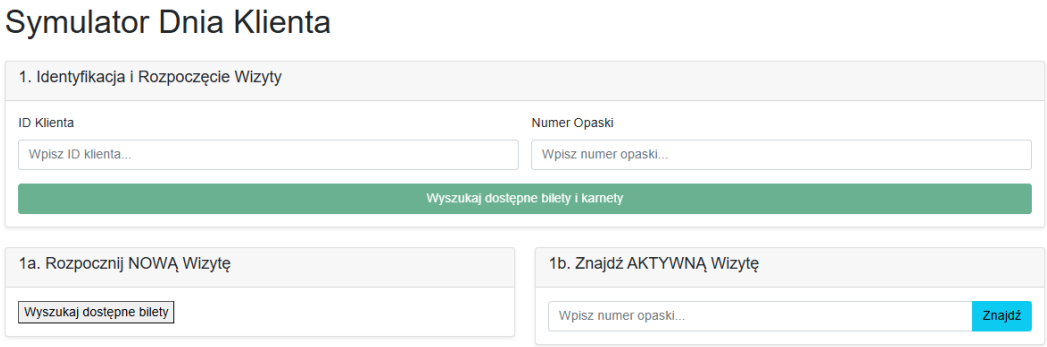



- 
- Płatność:
  - Kliknij Przejdź do płatności.
  - Wprowadź dane kartowe (symulacja) i zatwierdź.
  - Otrzymasz potwierdzenie transakcji na ekranie.

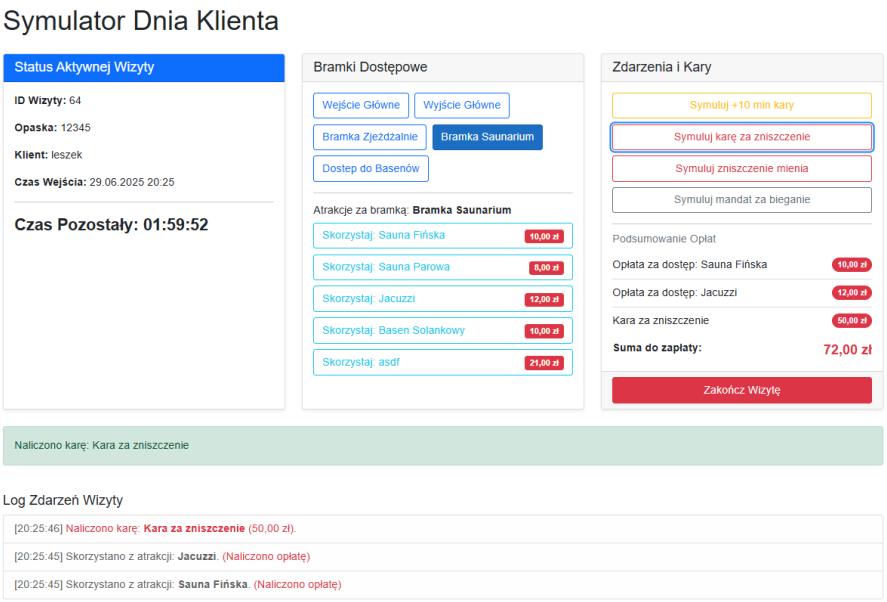
**Kontrola dostępu i rejestracja wizyt**

- Panel Sprzedawcy:
  - Zaloguj się jako Sprzedawca.
  - Przejdź do Zarządzaj Opaskami, wybierz klienta i przypisz mu opaskę.



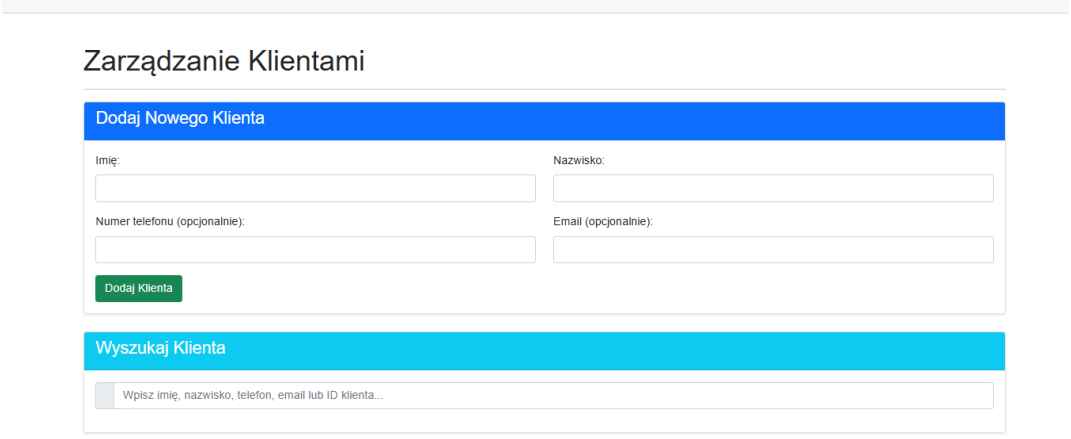



- Symulator Bramki:
  - Otwórz Symulator Bramki.
  - Wprowadź numer opaski, wybierz typ zdarzenia (wejście/wyjście) i zatwierdź.
  - Monitoruj logi w tabeli Log Dostępu.

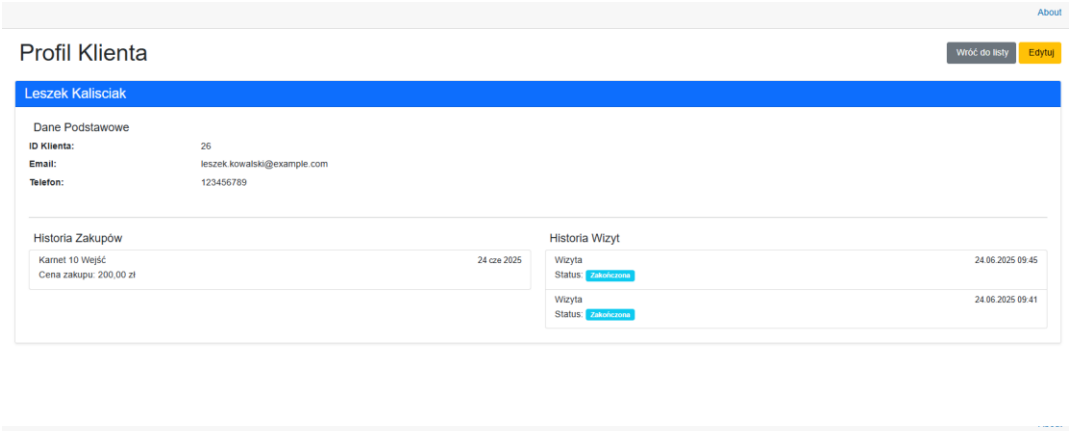


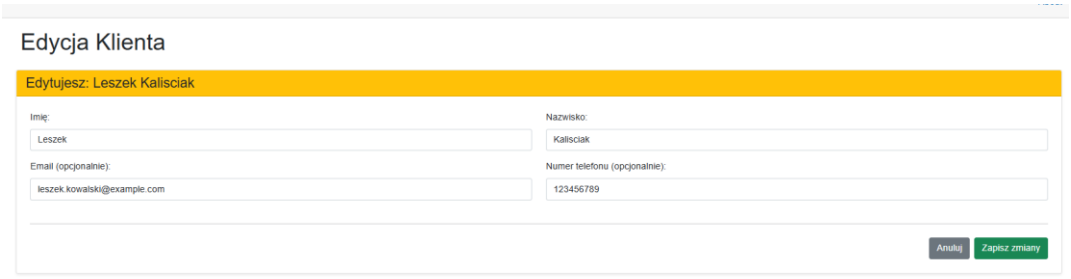

**Panel administracyjny (Admin)**

- Zarządzanie klientami:
  - W Zarządzaj Klientami wyszukaj, edytuj lub usuń rekordy klientów.

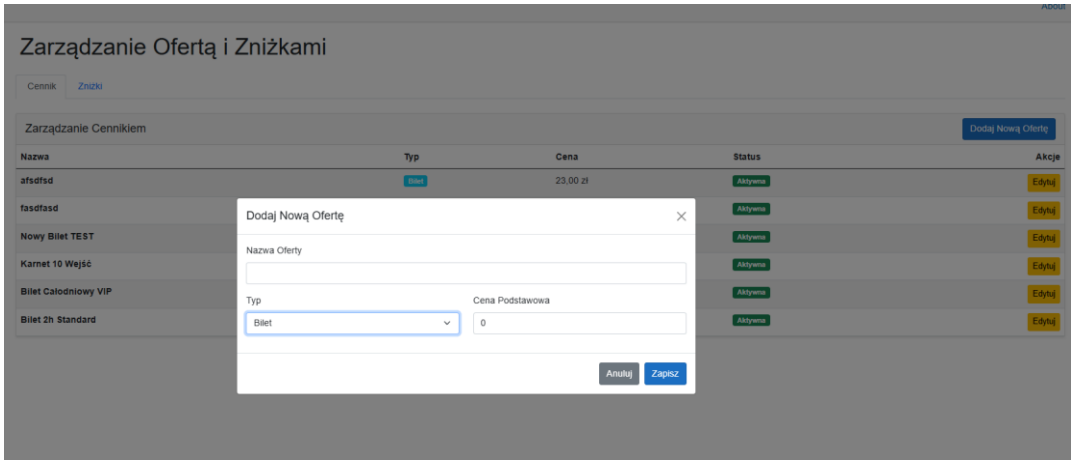


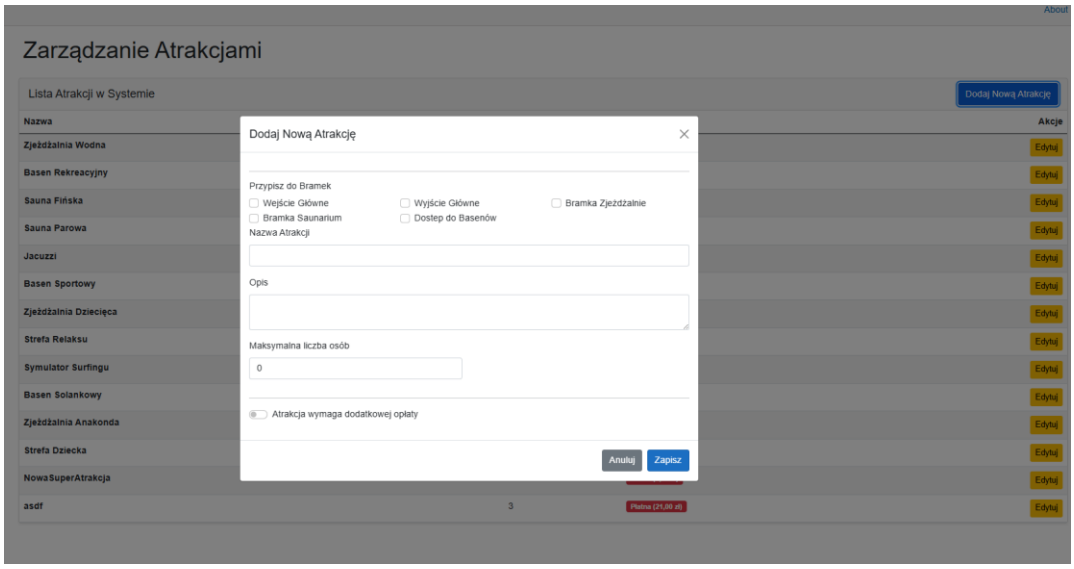

- 

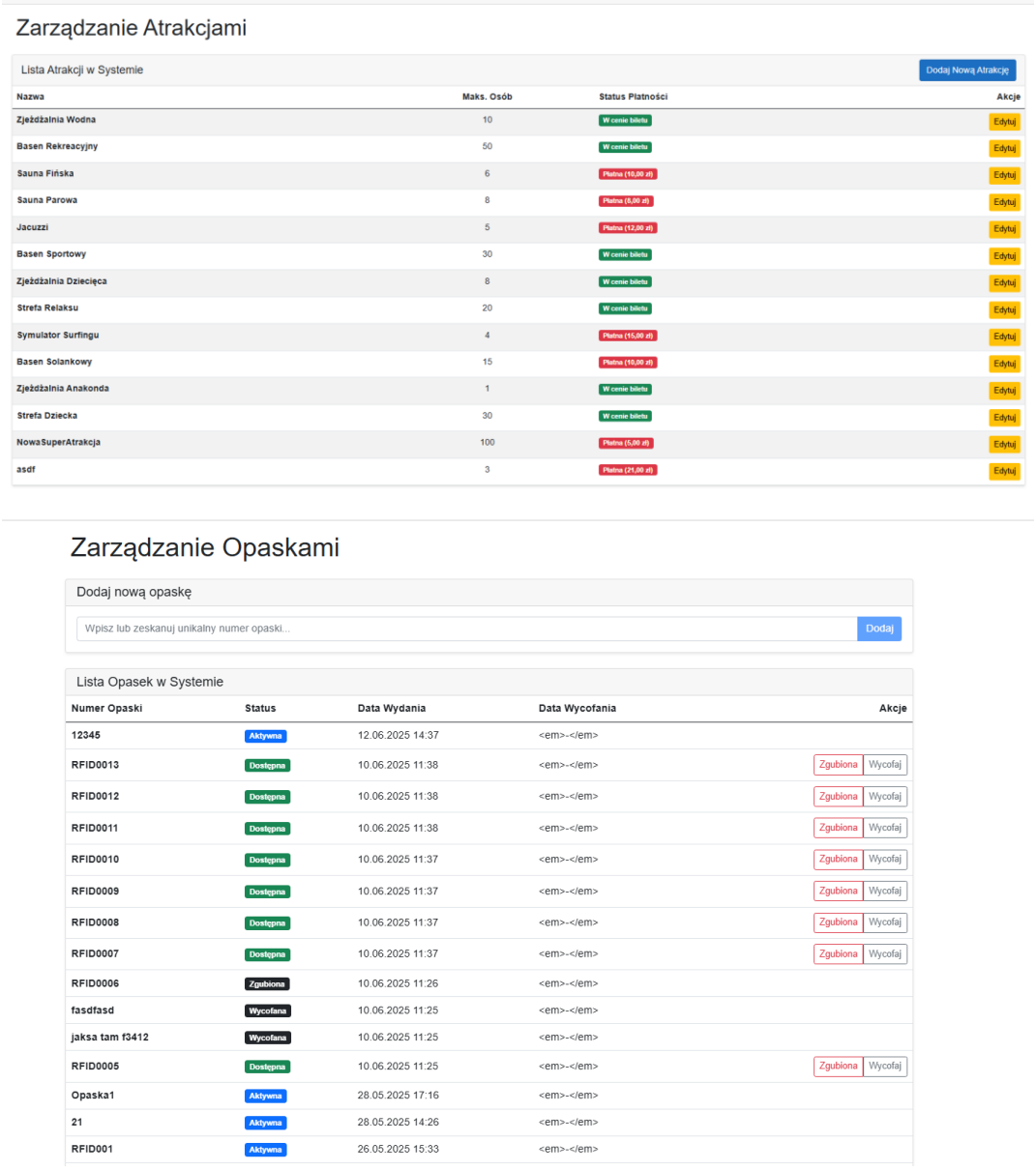



- Zarządzanie ofertami:
  - W Zarządzaj Oferta dodawaj i edytuj oferty cenowe.

- 
About
Zarządzanie Ofertą i Zniżkami
Cennik
Zniżki
Zarządzanie Cennikiem
Dodaj Nową Ofertę
Nazwa
Typ
Cena
Status
Akcje
afsdfsd
Bilet
23,00 zł
Aktywna
Edytuj
fasdfasd
Nowy Bilet TEST
Karnet 10 Wejść
Bilet Całodniowy VIP
Bilet 2h Standard
Dodaj Nową Ofertę
Nazwa Oferty
Typ
Bilet
Cena Podstawowa
0
Anuluj
Zapisz


About
Zarządzanie Atrakcjami
Lista Atrakcji w Systemie
Dodaj Nową Atrakcję
Nazwa
Akcje
Zjeżdżalnia Wodna
Basen Rekreacyjny
Sauna Fińska
Sauna Parowa
Jacuzzi
Basen Sportowy
Zjeżdżalnia Dziecięca
Strefa Relaksu
Symulator Surfingu
Basen Solankowy
Zjeżdżalnia Anakonda
Strefa Dziecka
NowaSuperAtrakcja
asdf
3
Edytuj
Dodaj Nową Atrakcję
Przypisz do Bramek
Wejście Główne
Wyjście Główne
Bramka Zjeżdżalnie
Bramka Saunarium
Dostep do Basenów
Nazwa Atrakcji
Opis
Maksymalna liczba osób
0
Atrakcja wymaga dodatkowej opłaty
Anuluj
Zapisz







# 5. Wnioski

**Zrozumienie procesów biznesowych:**

Praca nad aplikacją AquaparkApp pozwoliła zespołowi dogłębnie przeanalizować i odwzorować rzeczywiste procesy operacyjne aquaparku, od rejestracji klienta, przez sprzedaż biletów i kontrolę dostępu, aż po naliczanie kar za przekroczenia czasu. Szczegółowa analiza wymagań wymusiła precyzyjne zdefiniowanie ról użytkowników (klient, sprzedawca, administrator) i ich uprawnień, co zaowocowało spójnym modelem danych.

**Architektura i separacja odpowiedzialności:**

Przyjęta warstwowa architektura (UI / serwisy / dostęp do danych) okazała się kluczowa dla czytelności kodu i łatwości jego rozwijania. Wyodrębnienie interfejsów usług (I*Service) oraz ich implementacji ułatwiło testowanie jednostkowe i wprowadzanie poprawień bez ryzyka naruszenia innych obszarów aplikacji. Zastosowanie DTO pomogło uniknąć wycieków szczegółów implementacyjnych bazy danych do warstwy prezentacji.

**Entity Framework Core i migracje:**

Code-First w EF Core znacząco przyspieszyło prace nad modelem danych oraz umożliwiło automatyczne generowanie i modyfikowanie schematu bazy. Migracje pozwoliły na płynne przenoszenie zmian między środowiskami (developerskim, testowym, produkcyjnym), eliminując konieczność ręcznej ingerencji w strukturę SQL. Wyzwaniem okazało się jedynie precyzyjne odwzorowanie złożonych relacji N:M (tabela DostepAtrakcjiBramka) oraz konfiguracja zasad usuwania kaskadowego.

**Bezpieczeństwo i Identity:**

Wykorzystanie ASP.NET Core Identity ułatwiło wdrożenie solidnych mechanizmów uwierzytelniania i autoryzacji. Dzięki dwuskładnikowemu uwierzytelnianiu oraz potwierdzaniu e-mail, system spełnia podstawowe wymagania bezpieczeństwa w aplikacjach webowych. W przyszłości warto rozważyć dodanie polityk bezpieczeństwa haseł (np. wymagania długości i znaków specjalnych) oraz monitorowanie prób logowania (alerty, blokada konta).

**Interaktywność i UX dzięki Blazor Server:**

Blazor Server zapewnił płynne przejścia między stanami aplikacji bez przeładowania strony, co podniosło komfort użytkownika. Użycie wspólnych layoutów i komponentów znacznie przyspieszyło tworzenie nowych stron. Jedyną zauważalną wadą jest nieco wyższe zużycie zasobów serwera w dużym obciążeniu, w przyszłości warto przetestować migrację do Blazor WebAssembly lub hybrydowego modelu Server-Prerendered.

**Wyzwania związane z symulacją sprzętu:**

Zaimplementowany „Symulator Bramki” udowodnił koncepcję rejestracji zdarzeń RFID, ale do pełnego wdrożenia systemu w realnym aquaparku konieczna byłaby integracja z fizycznymi czytnikami i sterownikami bramek. W praktycznym rozwinięciu projektu warto

zastosować rozwiązania IoT oraz komunikację MQTT lub REST API między urządzeniami a aplikacją.

**Możliwości rozwoju i skalowalność:**

- Integracja systemów zewnętrznych: płatności online (Stripe, PayU), systemy lojalnościowe, kalendarze rezerwacji.
- Raportowanie i analityka: rozbudowane pulpity menedżerskie, wizualizacje statystyk (wykresy liczby wizyt, przychody dzienne, źródła zniżek).
- Wydajność: skalowanie serwera Blazor (SignalR), cache'owanie zapytań, optymalizacja zapytań LINQ.
- Rozbudowa modułu opasek: geolokalizacja stref, limity jednoczesnych użytkowników, dynamiczne alerty (np. przekroczenie maksymalnej liczby osób w strefie).

**Podsumowanie edukacyjne:**

Realizacja AquaparkApp była świetnym ćwiczeniem integrującym wiele obszarów technologii .NET: od projektowania bazy danych, przez API serwerowe, po interaktywny interfejs użytkownika w Blazorze. Projekt wzbogacił zespół o doświadczenie w zarządzaniu kodem, prowadzeniu migracji, testowaniu warstwowym oraz w pracy z repozytorium GitHub. Zdobyte umiejętności z pewnością przełożą się na przyszłe projekty komercyjne o podobnym zakresie.