Application supporting

the hotel reservation system

1. Dokumentacja wymagań i architektury oprogramowania

1.a. Krótki opis koncepcji projektu

System Hotel Reservation System to aplikacja webowa w architekturze MVC, wspierająca zarządzanie rezerwacjami hotelowymi. Pozwala na przeglądanie pokoi, dokonywanie rezerwacji, zarządzanie statusem rezerwacji, zarządzanie gośćmi, obsługę płatności Stripe i generowanie raportów.

1.b. Dwa mierzalne cele systemu

1. **Skrócenie czasu obsługi rezerwacji** o minimum 30% w porównaniu do obecnego systemu rejestracji telefonicznej lub mailowej.

2. **Zwiększenie liczby skutecznych rezerwacji** (tzn. finalnie potwierdzonych) o co najmniej 20% w ciągu pierwszych sześciu miesięcy działania systemu.

1.c. Dwóch interesariuszy systemu

1. **Kierownik hotelu** – oczekuje raportów dotyczących zajętości pokoi, przychodów oraz statystyk rezerwacji.

2. **Recepcjonista** – pracownik hotelu zajmujący się rezerwacjami i check-in/check-out na co dzień, potrzebuje intuicyjnego interfejsu do szybkiej obsługi gości.

1.d. Trzech aktorów projektowanego systemu

1. **Gość (Klient)** – osoba korzystająca z systemu w celu rezerwacji pokoju.

2. **Recepcjonista** – w systemie tworzy i modyfikuje rezerwacje, zarządza danymi gości.

3. **Kierownik hotelu** – ma dostęp do raportów i statystyk w celach analitycznych.

1.e. Pięć obiektów projektowanego systemu

1. **Rezerwacja** – przechowuje informacje o dacie przyjazdu, dacie wyjazdu, statusie i przypisanym pokoju.

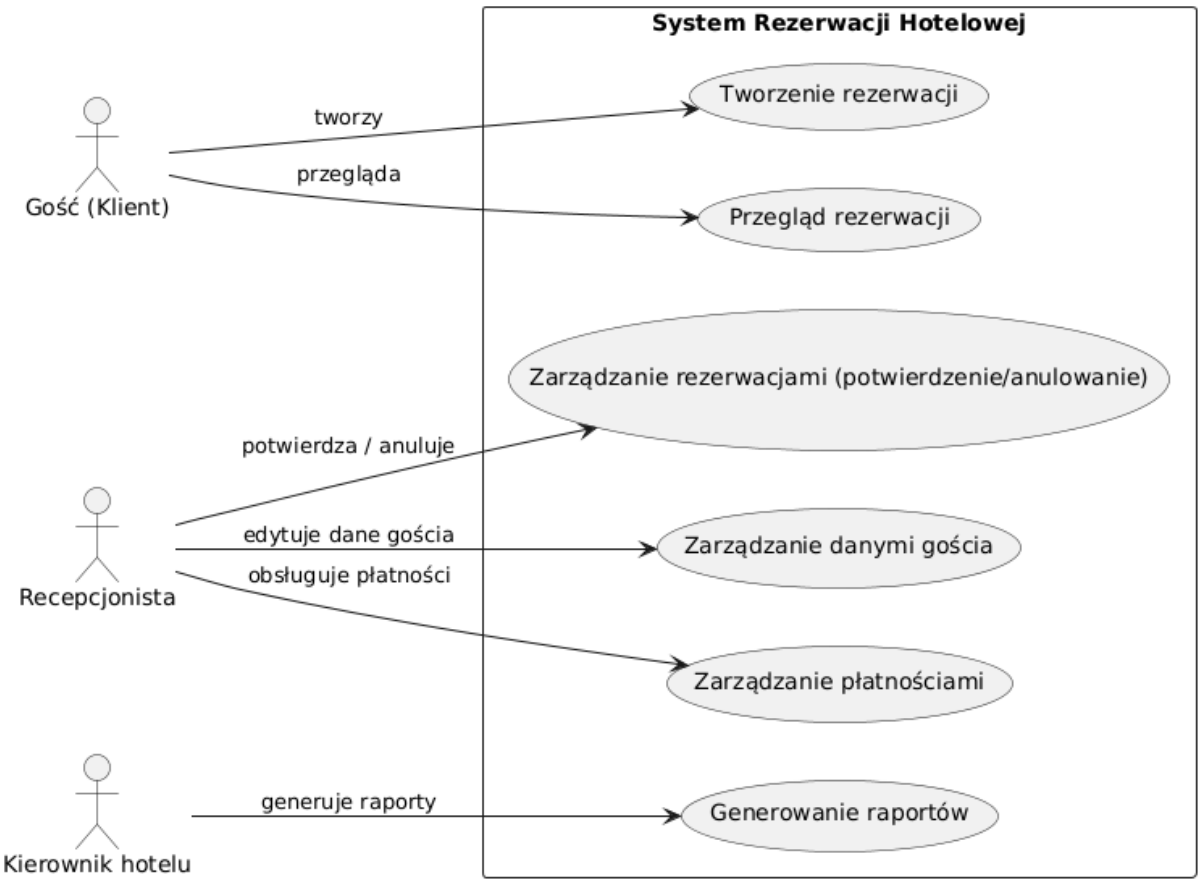
2. **Pokój** – zawiera numer, rodzaj (np. jednoosobowy, dwuosobowy), cenę oraz status dostępności.

3. **Gość (dane personalne)** – obejmuje informacje o danych osobowych gościa (imię, nazwisko, dane kontaktowe).

4. **Płatność** – opisuje sposób i status płatności (zapłacono/oczekuje/odrzucono).

5. **Raport** – generowane zbiorczo dane dotyczące rezerwacji, wpływów oraz statystyk wykorzystania pokoi.

1.f. Diagram kontekstowy



- **Gość** może tworzyć i sprawdzać „Rezerwację”.
- **Rezerwacja** jest związana z „Pokojem” i „Płatnością”.
- **Recepcjonista** zarządza „Rezerwacją” (potwierdza/anuluje) oraz może modyfikować dane gościa.
- **Kierownik hotelu** generuje „Raporty” oparte na danych z rezerwacji, płatności i dostępności pokoi.

1.g. Wymagania funkcjonalne wraz z kryteriami akceptacji

1. **Rejestracja rezerwacji**
    - **Wymaganie**: System musi umożliwić Gościowi utworzenie nowej rezerwacji poprzez formularz rezerwacyjny.
    - **Kryterium akceptacji**: Po podaniu daty przyjazdu, daty wyjazdu oraz danych osobowych system utworzy nowy rekord rezerwacji ze statusem „Oczekuje na potwierdzenie”.
2. **Potwierdzenie rezerwacji**
    - **Wymaganie**: Recepcjonista musi mieć możliwość zmiany statusu rezerwacji na „Potwierdzona”.
    - **Kryterium akceptacji**: Po zmianie statusu w panelu recepcjonisty w systemie, rezerwacja jest oznaczona jako „Potwierdzona” i może pojawiać się w raporcie zajętych pokoi.
3. **Anulowanie rezerwacji**
    - **Wymaganie**: Recepcjonista musi mieć możliwość anulowania rezerwacji wraz z podaniem powodu anulacji.
    - **Kryterium akceptacji**: Po wybraniu przycisku „Anuluj rezerwację” system ustawia status na „Anulowana” i rezerwacja nie jest widoczna w liście aktywnych rezerwacji.
4. **Sprawdzenie dostępności pokoi**

- **Wymaganie**: System musi umożliwiać Gościowi sprawdzenie dostępności pokoi w wybranym przedziale dat.

- **Kryterium akceptacji**: Po wybraniu terminu przyjazdu i wyjazdu, system wyświetla listę dostępnych pokoi wraz z cenami.

5. **Zarządzanie danymi gościa**

   - **Wymaganie**: Recepcjonista musi mieć możliwość edycji danych gościa (np. zmiana numeru telefonu).

   - **Kryterium akceptacji**: Po zapisaniu zmian w panelu recepcjonisty, zaktualizowane informacje o Gościu są widoczne w kolejnych rezerwacjach i raportach.

6. **Zarządzanie płatnością**

   - **Wymaganie**: System musi rejestrować sposób płatności (np. karta, gotówka, przelew) i status (opłacono/oczekuje).

   - **Kryterium akceptacji**: Po wprowadzeniu danych płatności system zmienia status rezerwacji na „Opłacona”, jeśli płatność została dokonana.

7. **Generowanie raportów**

   - **Wymaganie**: Kierownik hotelu musi mieć możliwość wygenerowania raportu przedstawiającego liczbę rezerwacji w danym okresie.

   - **Kryterium akceptacji**: Po wybraniu zakresu dat w panelu kierownika, system wyświetla raport z danymi statystycznymi (liczba rezerwacji, liczba potwierdzonych, przychód).

1.h. Dwa wymagania niefunkcjonalne z kryteriami akceptacji

1. **Rozszerzalność Systemu**

   - **Wymaganie:** System musi być zaprojektowany z myślą o łatwej integracji nowych modułów funkcjonalnych, bez ingerencji w istniejący kod. Każda nowa funkcjonalność (np. dodatkowy moduł płatności czy program lojalnościowy) powinna być dodawana jako oddzielny komponent, który

korzysta z wcześniej zdefiniowanych interfejsów i kontraktów.

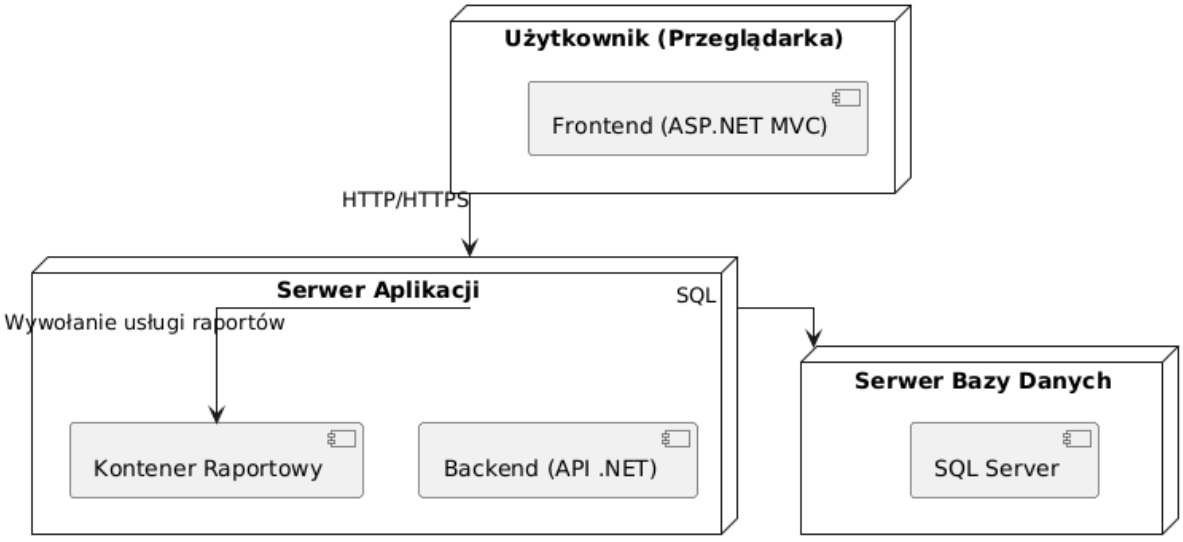
- **Kryterium akceptacji:** Nowa funkcjonalność (np. moduł ofert specjalnych) zostaje dodana jako osobny folder bez modyfikowania istniejących plików. Testy potwierdzają, że system działa poprawnie, a dotychczasowe funkcje pozostają bez zmian.

2. **Bezpieczeństwo**

   - **Wymaganie**: Dostęp do panelu Recepcjonisty i Kierownika hotelu ma być zabezpieczony mechanizmem autoryzacji i autentykacji – tylko zalogowane osoby z odpowiednimi uprawnieniami mogą wprowadzać zmiany w rezerwacjach.

   - **Kryterium akceptacji**: Brak możliwości nieautoryzowanego dostępu do panelu, a próba wejścia bez zalogowania przekierowuje na ekran logowania.

1.i. Diagram kontenera (C4) – wybrane kontenery



Powyższy diagram przedstawia strukturę kontenerów systemu rezerwacji hotelowej. Użytkownik końcowy komunikuje się z systemem przez przeglądarkę internetową, korzystając z interfejsu aplikacji opartego na ASP.NET MVC (Frontend). Wszystkie żądania HTTP/HTTPS trafiają do serwera aplikacji, który zawiera dwa główne kontenery:

- **Backend (API .NET)** – obsługuje logikę biznesową (rezerwacje, płatności, goście),

- **Kontener raportowy** – odpowiada za generowanie raportów dostępnych dla kierownika.

Aplikacja komunikuje się z relacyjną bazą danych Microsoft SQL Server, gdzie przechowywane są wszystkie dane związane z użytkownikami, rezerwacjami, pokojami, płatnościami oraz logami.

1.j. Diagram komponentów (C4) – szczegóły wybranego kontenera

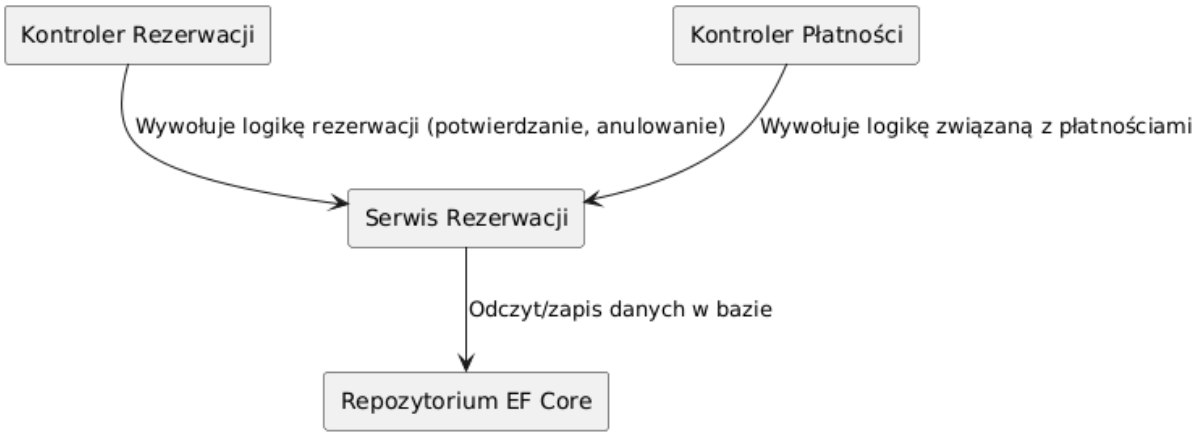
Przykładowo, dla **Backendu (API .NET)** wyróżniamy następujące komponenty:

1. **Kontroler Rezerwacji** – obsługuje żądania związane z tworzeniem, potwierdzaniem i anulowaniem rezerwacji.

2. **Kontroler Płatności** – zarządza obsługą i aktualizacją statusu płatności.

3. **Serwis Rezerwacji** – zawiera logikę biznesową związaną z zarządzaniem rezerwacjami (np. walidacja terminów).

4. **Repozytorium (np. EF Core)** – odpowiedzialne za komunikację z bazą danych (tabele: Rezerwacje, Pokoje, Płatności, Goście).



1.k. Stos technologiczny

1. **.NET 8 (C#)** – nowoczesne środowisko i język programowania, umożliwia szybkie tworzenie skalowalnych aplikacji webowych i desktopowych.

2. **MVC** - wzorzec architektoniczny (model-view-controller), który skupia w sobie logikę Frontendu w widokach aplikacji oraz Backend jako modele bazy danych i kontrolery z funkcjonalnościami

3. **Entity Framework Core** – narzędzie ORM upraszczające komunikację z bazą danych, co przyspiesza proces tworzenia i modyfikacji struktury danych.

4. **Microsoft SQL Server** – relacyjna baza danych dobrze zintegrowana z platformą .NET, zapewnia skalowalność i bezpieczeństwo.

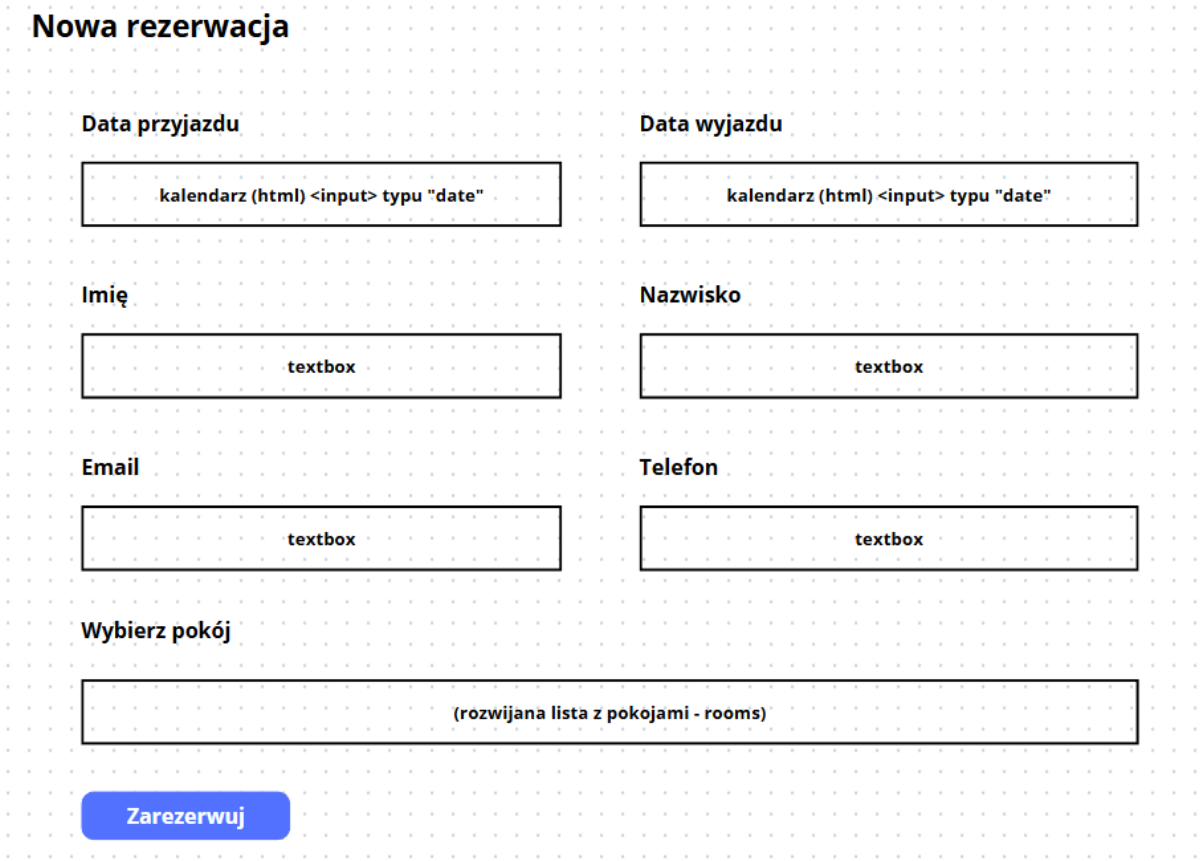
5. **Stripe API** - platforma płatności online zintegrowana z systemem w celu obsługi transakcji kartą płatniczą. Umożliwia tworzenie bezpiecznych płatności oraz śledzenie statusu transakcji.

6. **Bootstrap 5** – framework CSS wspierający szybkie tworzenie responsywnych i estetycznych interfejsów użytkownika. Ułatwia stylowanie formularzy, przycisków i tabel bez konieczności pisania dużej ilości własnego CSS.

7. **FluentValidation** – biblioteka do walidacji danych wejściowych w warstwie logiki biznesowej. Pozwala w prosty sposób definiować zasady walidacji (np. wymagane pola, format email) w sposób czytelny, skalowalny i niezależny od warstwy prezentacji.

2. Prototyp aplikacji

2.a. Interfejs użytkownika – makieta jednego ekranu



Powyższy prototyp przedstawia ekran formularza rezerwacji dostępny dla użytkownika (Gościa). Projekt został przygotowany w **Canva** i prezentuje intuicyjny układ pól formularza, który został wdrożony także w aplikacji.

Układ formularza zawiera następujące elementy:

- **Data przyjazdu / Data wyjazdu** – pola typu `date`, umożliwiające użytkownikowi wybór dat z kalendarza HTML.
- **Imię, Nazwisko, Email, Telefon** – podstawowe dane kontaktowe użytkownika, wprowadzone jako `textbox` (input).
- **Wybór pokoju** – rozwijana lista (`select`), która wyświetla dostępne pokoje wraz z ceną i typem.
- **Przycisk „Zarezerwuj”** – przesyła dane do systemu w celu zapisania rezerwacji.

2.b. Cztery argumenty uzasadniające przydatność interfejsu Intuicyjność:

Pola do wpisania dat przyjazdu i wyjazdu oraz przycisk „Sprawdź dostępne pokoje” są widoczne w centralnej części ekranu, co ułatwia początkującemu użytkownikowi znalezienie najważniejszej funkcjonalności.

1. **Szybkie działanie**: Po uzupełnieniu podstawowych danych i kliknięciu jednego przycisku użytkownik od razu widzi listę dostępnych pokoi – minimalna liczba kroków w procesie.

2. **Zrozumiała prezentacja wyników**: Lista pokazuje numer pokoju, typ oraz cenę za dobę, a także przycisk „Rezerwuj” obok każdego wariantu, co klarownie sygnalizuje użytkownikowi opcję zakupu.

3. **Czytelna nawigacja**: Nagłówek z nazwą hotelu i stopka/pasek informacyjny pozwalają na szybkie odnalezienie informacji ogólnych o hotelu i ewentualnych linków (np. regulamin, kontakt).

4. **Responsywność:** Interfejs dostosowuje się do różnych rozmiarów ekranu, dzięki czemu użytkownicy mogą wygodnie korzystać z aplikacji zarówno na komputerze, jak i urządzeniach mobilnych, np. podczas podróży.

3. Implementacja fragmentu funkcjonalności

Założenie: W ramach projektu wdrożymy przykładową funkcjonalność tworzenia i potwierdzania rezerwacji.

3.a. Zgodność implementowanej architektury z wybranym wzorcem

- Używany wzorzec architektoniczny: **Warstwowy (z rozdzieleniem na warstwę prezentacji, warstwę logiki biznesowej i warstwę dostępu do danych)**.

- **Warstwa prezentacji (np. ASP.NET MVC)** zawiera kontrolery obsługujące żądania HTTP i zwracające widoki.

- **Warstwa logiki biznesowej (usługi)** przechowuje zasady działania aplikacji, np. reguły tworzenia i potwierdzania rezerwacji.

- **Warstwa dostępu do danych (repozytoria)** komunikuje się z bazą danych przy użyciu Entity Framework Core.

3.b. Zgodność implementacji z wybranymi wymaganiami funkcjonalnymi

1. **Rejestracja rezerwacji**
2. **Potwierdzenie rezerwacji**
3. **Anulowanie rezerwacji**
4. **Sprawdzenie dostępności pokoi**

3.c. Zgodność implementacji z wybranymi wymaganiami niefunkcjonalnymi

Wdrożono dwa niefunkcjonalne wymagania (każde wycenione na 2,5 pkt):

1. **Rozszerzalność systemu:** Architektura aplikacji umożliwia łatwe dodawanie nowych funkcjonalności bez ingerencji w istniejący kod. Przykładowo, moduł „oferty specjalne" został dodany jako oddzielny zestaw plików w dedykowanym folderze, z wykorzystaniem istniejących interfejsów. System nadal działa poprawnie, a istniejące funkcje nie wymagają modyfikacji.
2. **Bezpieczeństwo**: Dodano autentykację (np. Identity w .NET) oraz autoryzację (role „Recepcjonista", „Kierownik") tak, by tylko zalogowane osoby mogły wykonać akcję potwierdzenia czy anulowania rezerwacji.

4. Scenariusz testowy

Przykład minimalnego scenariusza testowego dla funkcjonalności „Potwierdzenie rezerwacji":

- **Wymagania wstępne**:
    1. W systemie istnieje rezerwacja gościa ze statusem „Oczekuje" - na potwierdzenie.
    2. Użytkownik (Recepcjonista) jest zalogowany i posiada uprawnienia do potwierdzania rezerwacji.
- **Kroki**:
    1. Recepcjonista wchodzi w zakładkę „Rezerwacje" i wybiera rezerwację gościa.
    2. Recepcjonista klika przycisk „Potwierdź".

3. System wywołuje metodę `POST /Reservation/Confirm/{id}`.

- **Oczekiwany rezultat**:

    1. System ustawia status rezerwacji na „Potwierdzona”.
    2. Użytkownik otrzymuje komunikat „Rezerwacja potwierdzona.”.
    3. W bazie danych w tabeli `Rezerwacje` rekord o podanym **{id}** ma status „Potwierdzona”.

5. Wykorzystanie systemu kontroli wersji

5.a. Skonfigurowanie zdalnego repozytorium

Projekt został utworzony zarówno w serwisie GitHub, jak i lokalnie, przy użyciu systemu kontroli wersji Git oraz wiersza poleceń Git Bash, jako publiczne repozytorium. Zawiera ono podstawowe pliki aplikacji (MVC), frontend (widoki) + backend (kontrolery i modele itd.) jak i zarówno konfiguracyjne, np. apsettings.json.

5.b. Wykorzystanie GitHub

- **Brancha `main`** – zawiera stabilną, przetestowaną wersję kodu.
- **Brancha `develop`** – integruje bieżący rozwój aplikacji.
- **Funkcjonalności** (feature branches) – np. `feature/rejestracja-rezerwacji`, `feature/potwierdzanie-rezerwacji`.
- Po zakończeniu implementacji funkcjonalności tworzymy Pull/Merge Request do branchy `develop`, a przed publikacją wersji produkcyjnej – do `main`.