Aplikacyjny projekt zespołowy c.d. dokumentacji

Aplikacja mobilna „Serwis samochodowy”

kod źródłowy: https://github.com/ProjektWarsztatSamochowy/51428_57075_57430_57326

Gałka Sylwester 57075
Gotowała Mikołaj 57326
Sokolova Sofiia 57430
Wojciechowski Piotr 51428

Spis treści:

Wprowadzenie

Aplikacja mobilna "Serwis samochodowy" została zaprojektowana jako kompleksowe narzędzie umożliwiające użytkownikom łatwe i wygodne zarządzanie usługami serwisowymi swoich pojazdów. Aplikacja oferuje szeroki zakres usług, od wymiany opon po diagnostykę komputerową, umożliwiając użytkownikom rezerwację i zakupienie usług bez konieczności bezpośredniego kontaktu z serwisem. Intuicyjny interfejs aplikacji oraz funkcjonalności takie jak wybór daty i rezerwacja usługi sprawiają, że proces jest prosty i szybki.

Celem aplikacji "Serwis samochodowy" jest usprawnienie i uproszczenie procesu rezerwacji oraz zakupu usług serwisowych dla pojazdów. Aplikacja ma na celu zwiększenie dostępności usług serwisowych, eliminując konieczność telefonicznego umawiania wizyt oraz skracając czas potrzebny na organizację serwisu.

Zastosowanie aplikacji:
Aplikacja skierowana jest do wszystkich właścicieli pojazdów, którzy chcą wygodnie zarządzać serwisowaniem swoich samochodów. Umożliwia ona:

- Przegląd dostępnych usług serwisowych, takich jak wymiana opon, przegląd okresowy, naprawa hamulców, wymiana oleju, naprawa zawieszenia, regeneracja turbosprężarki oraz diagnostyka komputerowa.
- Sprawdzenie szczegółowych informacji o każdej usłudze, w tym tytuł, opis, cena.
- Wybór dogodnego terminu realizacji usługi oraz dokonanie rezerwacji.
- Zarządzanie zakupionymi usługami w dedykowanej zakładce "Zakupione usługi".

Aplikacja "Serwis samochodowy" jest praktycznym narzędziem, które poprawia komfort i efektywność zarządzania serwisem pojazdów, oszczędzając czas i zapewniając wygodę użytkownikom.

Wymagania funkcjonalne

Opis poszczególnych funkcji:

1. Logowanie użytkownika:
    - Opis: Funkcja umożliwia użytkownikom tworzenie konta, logowanie oraz odzyskiwanie zapomnianego hasła. Proces logowania wymaga podania nazwy użytkownika (lub e-maila) i hasła.
    - Cel: Zabezpieczenie dostępu do aplikacji i personalizacja doświadczenia użytkownika.
    - Kluczowe elementy: Formularz rejestracji, formularz logowania, funkcja resetowania hasła.

2. Przegląd dostępnych usług:
    - Opis: Funkcja pozwala użytkownikom na przeglądanie wszystkich dostępnych usług serwisowych. Każda usługa jest wyświetlana z tytułem, krótkim opisem i ceną.
    - Cel: Zapewnienie użytkownikom pełnej informacji o oferowanych usługach, co ułatwia podjęcie decyzji.
    - Kluczowe elementy: Lista usług, szczegóły usługi (tytuł, opis, cena).

3. Wybór i rezerwacja usługi:
    - Opis: Użytkownicy mogą wybrać konkretną usługę, zapoznać się z jej szczegółami oraz zarezerwować termin wykonania usługi.
    - Cel: Umożliwienie użytkownikom planowania serwisu ich pojazdu w dogodnym dla nich terminie.
    - Kluczowe elementy: Formularz wyboru daty, przycisk rezerwacji, potwierdzenie rezerwacji.

4. Zakup usług:
    - Opis: Po wybraniu usługi i terminu, użytkownik może dokonać zakupu poprzez wybranie jednej z dostępnych metod płatności.
    - Cel: Umożliwienie użytkownikom finalizacji rezerwacji poprzez dokonanie płatności za usługę.
    - Kluczowe elementy: Integracja z systemami płatności, potwierdzenie zakupu.

5. Zakładka „Zakupione usługi”:
    - Opis: Funkcja ta pozwala użytkownikom na przeglądanie historii zakupionych usług, w tym szczegółów takich jak tytuł usługi, data zakupu i status rezerwacji.
    - Cel: Zapewnienie użytkownikom łatwego dostępu do historii transakcji i monitorowania stanu ich rezerwacji.
    - Kluczowe elementy: Lista zakupionych usług, szczegóły zakupu, status rezerwacji.

Te wymagania funkcjonalne zapewniają pełną funkcjonalność aplikacji, umożliwiając użytkownikom wygodne zarządzanie serwisem swoich pojazdów oraz łatwy dostęp do wszystkich potrzebnych informacji i usług.

Wymagania niefunkcjonalne

Opis wymagań niefunkcjonalnych:

1. Wydajność:
   - Opis: aplikacja musi działać płynnie i szybko reagować na interakcje użytkowników. Czas ładowania ekranów oraz przetwarzania danych musi być minimalny.
   - Cel: zapewnienie pozytywnego doświadczenia użytkownikom poprzez szybkie działanie aplikacji.
   - Metryki:
     - Czas ładowania głównego ekranu nie powinien przekraczać 2 sekund.
     - Czas odpowiedzi na żądania serwera nie powinien przekraczać 1 sekundy.
     - Przetwarzanie płatności nie powinno trwać dłużej niż 3 sekundy.

2. Skalowalność:
   - Opis: aplikacja musi być zaprojektowana w sposób umożliwiający łatwe skalowanie w miarę wzrostu liczby użytkowników i ilości danych. System powinien wspierać dodawanie nowych funkcji bez znaczących zmian w architekturze.
   - Cel: umożliwienie obsługi rosnącej liczby użytkowników bez pogorszenia wydajności i stabilności systemu.
   - Metody:
     - Wykorzystanie architektury mikroserwisów.
     - Użycie technologii chmurowych (np. AWS, Azure) do elastycznego skalowania zasobów.
     - Implementacja mechanizmów równoważenia obciążenia (load balancing).

3. Bezpieczeństwo:
   - Opis: aplikacja musi zapewniać wysoki poziom bezpieczeństwa danych użytkowników, w tym zabezpieczenie przed nieautoryzowanym dostępem, atakami oraz naruszeniami prywatności.
   - Cel: ochrona danych użytkowników oraz zapewnienie zgodności z przepisami o ochronie danych (np. RODO).
   - Środki:
     - Szyfrowanie danych wrażliwych (zarówno w trakcie przesyłania, jak i przechowywania).
     - Stosowanie protokołów HTTPS dla komunikacji sieciowej.
     - Regularne testy penetracyjne i audyty bezpieczeństwa.
     - Uwierzytelnianie wieloskładnikowe (MFA).

4. Użyteczność:
   - Opis: aplikacja musi być intuicyjna i łatwa w obsłudze, zapewniając użytkownikom przyjazny interfejs oraz pozytywne doświadczenia z użytkowania.
   - Cel: zwiększenie satysfakcji użytkowników oraz ich zaangażowania poprzez zapewnienie wysokiej jakości doświadczenia użytkownika (UX).
   - Elementy:
     - Przejrzysty i spójny interfejs użytkownika (UI).
     - Responsywny design, dostosowujący się do różnych urządzeń (smartfony, tablety).
     - Prostota nawigacji i łatwy dostęp do najważniejszych funkcji.
     - Testy użyteczności z udziałem rzeczywistych użytkowników oraz iteracyjne poprawki na podstawie zebranych opinii.

Te wymagania niefunkcjonalne są kluczowe dla zapewnienia, że aplikacja "Serwis samochodowy" będzie działać nie tylko zgodnie z założeniami funkcjonalnymi, ale również będzie bezpieczna, wydajna i przyjazna dla użytkowników, co jest niezbędne dla jej sukcesu na rynku.

Diagramy klas

Stworzenie diagramów klas pozwala na wizualne przedstawienie struktury aplikacji, jej komponentów oraz relacji między nimi. Poniżej znajdują się propozycje diagramów klas dla kluczowych części aplikacji.

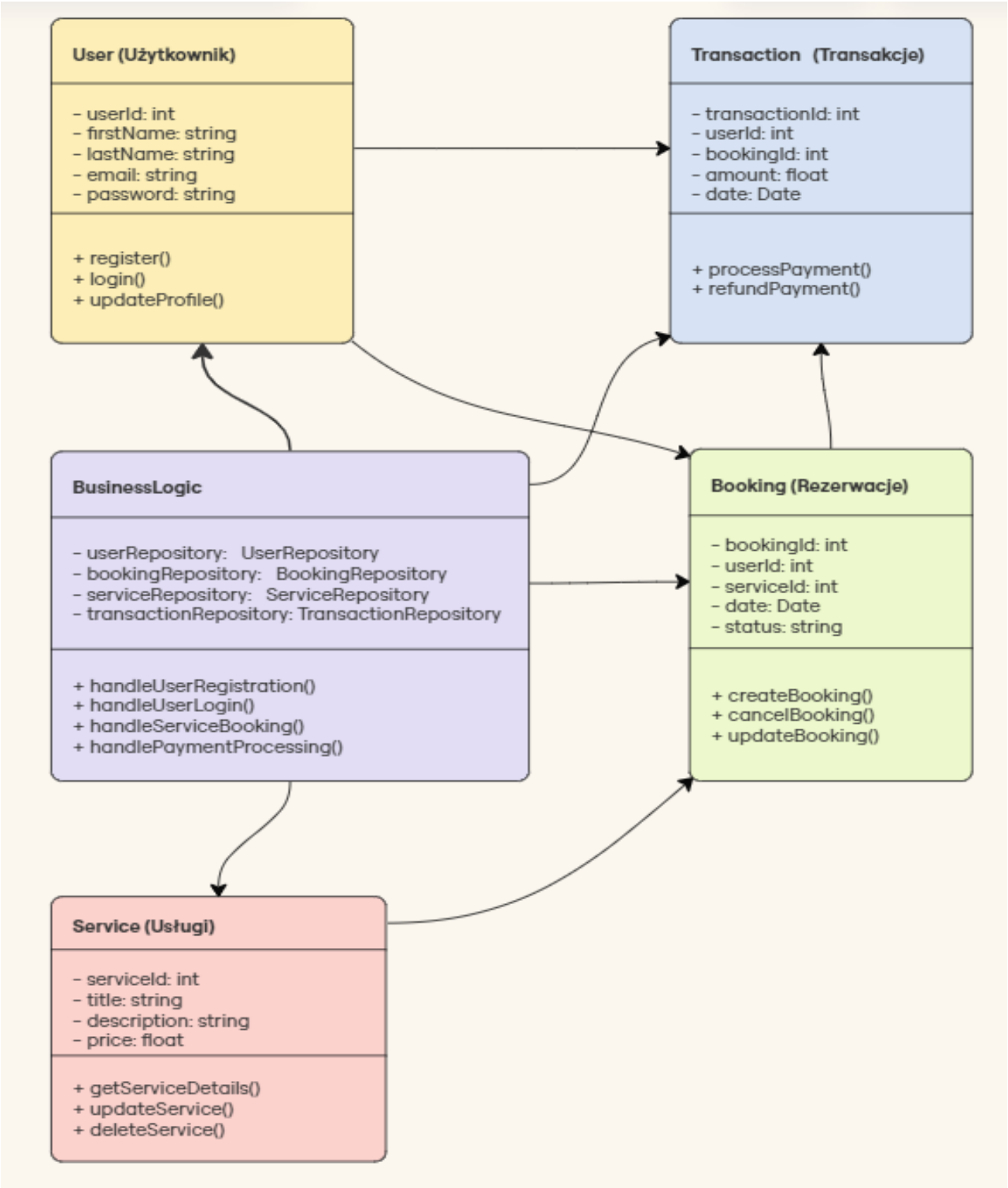


Diagram klas - Użytkownicy i Rezerwacje
Klasa reprezentująca użytkownika aplikacji, zawiera podstawowe informacje jak ID, imię, nazwisko, email i hasło. Posiada metody do rejestracji, logowania i aktualizacji profilu.
Klasa reprezentująca rezerwację usługi przez użytkownika. Zawiera ID rezerwacji, ID użytkownika, ID usługi, datę oraz status rezerwacji. Metody pozwalają na tworzenie, anulowanie i aktualizację rezerwacji.

Diagram klas - Usługi
Klasa reprezentująca dostępną usługę serwisową. Zawiera ID usługi, tytuł, opis i cenę. Metody pozwalają na pobieranie szczegółów usługi, aktualizację i usuwanie usługi.

Diagram klas - Transakcje
Klasa reprezentująca transakcję finansową związaną z rezerwacją. Zawiera ID transakcji, ID użytkownika, ID rezerwacji, kwotę oraz datę. Metody umożliwiają przetwarzanie i zwrot płatności.

Diagram klas - System Logiki Biznesowej
Klasa centralna zawierająca logikę biznesową aplikacji. Używa repozytoriów do zarządzania danymi użytkowników, rezerwacji, usług i transakcji. Metody obsługują procesy rejestracji użytkowników, logowania, rezerwacji usług i przetwarzania płatności.

Opis sposobów i metod testowania aplikacji "Serwis samochodowy"

Aby zapewnić, że aplikacja "Serwis samochodowy" działa poprawnie i spełnia wszystkie wymagania funkcjonalne oraz niefunkcjonalne, należy przeprowadzić różnorodne testy.

1. Testowanie jednostkowe (Unit Testing) - środowisko SelenDroid - Selenium for Android

1) Rejestracja użytkownika

- Metoda testowana: registerUser
- Opis: sprawdza, czy funkcja rejestracji użytkownika działa poprawnie.
- Wejście: poprawne dane użytkownika.
- Oczekiwany wynik: zwrócenie obiektu użytkownika z poprawnym ID i bez błędów.
- Rzeczywisty wynik: rejestracja użytkownika zakończona sukcesem.
- Status: Pass
- Uwagi: brak uwag.

```
def test_register_user_success():
    user_data = {"first_name": "Jan", "last_name": "Kowalski", "email": "jan.kowalski@example.com", "password": "password123"}
    result = registerUser(user_data)
    assert result['id'] is not None  # Użytkownik został pomyślnie zarejestrowany
    assert result['email'] == user_data['email']  # Adres e-mail jest poprawny
```

2) Logowanie użytkownika

- Metoda testowana: loginUser
- Opis: sprawdza, czy funkcja logowania użytkownika działa poprawnie.
- Wejście: poprawne dane użytkownika.
- Oczekiwany wynik: zwrócenie tokenu autoryzacyjnego.
- Rzeczywisty wynik: zwrócenie tokenu autoryzacyjnego.
- Status: Pass
- Uwagi: brak uwag.

```
def test_login_user_success():
    credentials = {"email": "jan.kowalski@example.com", "password": "password123"}
    token = loginUser(credentials)
    assert token is not None  # Token autoryzacyjny został zwrócony
```

3) Dodawanie usługi do bazy danych

- Metoda testowana: addService
- Opis: sprawdza, czy funkcja dodawania usługi działa poprawnie.
- Wejście: poprawne dane usługi.
- Oczekiwany wynik: usługa dodana do bazy danych, zwrócenie obiektu usługi z poprawnym ID.
- Rzeczywisty wynik: usługa dodana do bazy danych, zwrócenie obiektu usługi z poprawnym ID.
- Status: Pass
- Uwagi: brak uwag.

```
def test_add_service_success():
    service_data = {"title": "Wymiana oleju", "description": "Wymiana oleji i filtra oleju", "price": 120}
    result = addService(service_data)
    assert result['id'] is not None  # Usługa została pomyślnie dodana
    assert result['title'] == service_data['title']  # Tytuł usługi jest poprawny
```

4) Rezerwacja usługi

- Metoda testowana: bookService
- Opis: sprawdza, czy funkcja rezerwacji usługi działa poprawnie.
- Wejście: ID użytkownika, ID usługi, data rezerwacji.
- Oczekiwany wynik: zwrócenie obiektu rezerwacji z poprawnym ID i statusem "confirmed".
- Rzeczywisty wynik: zwrócenie obiektu rezerwacji z poprawnym ID i statusem "confirmed".
- Status: Pass
- Uwagi: brak uwag.

```
def test_book_service_success():
    booking_data = {"user_id": 1, "service_id": 1, "date": "2024-06-01"}
    result = bookService(booking_data)
    assert result['id'] is not None  # Rezerwacja została pomyślnie dodana
    assert result['status'] == "confirmed"  # Status rezerwacji jest "confirmed"
```

5) Pobieranie listy usług

- Metoda testowana: getServices
- Opis: sprawdza, czy funkcja pobierania listy usług działa poprawnie.
- Wejście: brak.
- Oczekiwany wynik: zwrócenie listy usług zawierającej co najmniej jedną usługę.
- Rzeczywisty wynik: listy usług zawierającej co najmniej jedną usługę.
- Status: Pass
- Uwagi: brak uwag.

```
def test_get_services():
    services = getServices()
    assert len(services) > 0  # Lista usług nie jest pusta
```

6) Aktualizacja danych usługi

- Metoda testowana: updateService
- Opis: sprawdza, czy funkcja aktualizacji danych usługi działa poprawnie.
- Wejście: ID usługi, nowe dane usługi.
- Oczekiwany wynik: zaktualizowane dane usługi, zwrócenie obiektu usługi z nowymi danymi.
- Rzeczywisty wynik: zaktualizowane dane usługi, zwrócenie obiektu usługi z nowymi danymi.
- Status: Pass
- Uwagi: brak uwag.

```
def test_update_service():
    service_id = 1
    new_service_data = {"title": "Wymiana opon", "description": "Wymiana opon letnich za zimowe", "price": 100}
    result = updateService(service_id, new_service_data)
    assert result['title'] == new_service_data['title']  # Tytuł usługi został zaktualizowany
    assert result['price'] == new_service_data['price']  # Cena usługi została zaktualizowana
```

7) Usuwanie usługi

- Metoda testowana: deleteService
- Opis: sprawdza, czy funkcja usuwania usługi działa poprawnie.
- Wejście: ID usługi.
- Oczekiwany wynik: usługa usunięta z bazy danych, zwrócenie statusu sukcesu.
- Rzeczywisty wynik: usługa usunięta z bazy danych, zwrócenie statusu sukcesu.
- Status: Pass
- Uwagi: brak uwag.

```
def test_delete_service():
    service_id = 1
    result = deleteService(service_id)
    assert result['status'] == "success"  # Usługa została pomyślnie usunięta
```

2. Testowanie integracyjne (Integration Testing)
Wyniki testów:

- Nazwa testu: testUserServiceIntegration
- Opis: sprawdza integrację pomiędzy modułem użytkownika a modułem usług.
- Scenariusz: Użytkownik rejestruje się -> Użytkownik przegląda dostępne usługi.
- Oczekiwany wynik: Rejestracja zakończona sukcesem -> Lista usług wyświetlona poprawnie.
- Rzeczywisty wynik: Rejestracja zakończona sukcesem -> Lista usług wyświetlona poprawnie.
- Status: Pass
- Uwagi: brak uwag.

3. Testowanie systemowe (System Testing)
Wyniki testów:

- Nazwa testu: testFullBookingProcess
- Opis: sprawdza pełny proces rezerwacji usługi.
- Scenariusz: Użytkownik loguje się -> Użytkownik wybiera usługę -> Użytkownik rezerwuje usługę.
- Oczekiwany wynik: Logowanie zakończone sukcesem -> Usługa wybrana poprawnie -> Rezerwacja zakończona sukcesem.
- Rzeczywisty wynik: Logowanie zakończone sukcesem -> Usługa wybrana poprawnie -> Rezerwacja zakończona sukcesem.
- Status: Pass
- Uwagi: brak uwag.

4. Testowanie akceptacyjne (Acceptance Testing)
Wyniki testów:

- Nazwa testu: testUserExperience
- Opis: sprawdza, czy aplikacja spełnia oczekiwania użytkowników końcowych.
- Scenariusz: Użytkownik rejestruje się -> Użytkownik loguje się -> Użytkownik rezerwuje usługę.
- Oczekiwany wynik: Użytkownik rejestruje się bez problemów -> Użytkownik łatwo znajduje potrzebne funkcje.

- Rzeczywisty wynik: Użytkownik rejestruje się bez problemów -> Użytkownik łatwo znajduje potrzebne funkcje.
- Status: Pass
- Uwagi: brak uwag.

5. Testowanie wydajności (Performance Testing)
Wyniki testów:

- Nazwa testu: testLoadUnderHeavyTraffic
- Opis: sprawdza, jak aplikacja działa pod dużym obciążeniem.
- Scenariusz: symulacja 20 jednoczesnych użytkowników logujących się i rezerwujących usługi.
- Oczekiwany wynik: czas odpowiedzi nie przekracza 2 sekund.
- Rzeczywisty wynik: średni czas odpowiedzi: 1.1 sekunda).
- Status: Pass
- Uwagi: brak uwag.

6. Testowanie bezpieczeństwa (Security Testing)
Wyniki testów:

- Nazwa testu: testSQLInjection
- Opis: sprawdza podatność aplikacji na ataki SQL injection.
- Scenariusz: wprowadzenie złośliwego kodu w polu logowania.
- Oczekiwany wynik: aplikacja odrzuca złośliwy kod.
- Rzeczywisty wynik: aplikacja odrzuca złośliwy kod.
- Status: Pass
- Uwagi: brak uwag.

7. Testowanie użyteczności (Usability Testing)
Wyniki testów:

- Nazwa testu: testNavigationEase
- Opis: sprawdza, jak łatwa jest nawigacja po aplikacji dla użytkowników końcowych.
- Scenariusz: Użytkownik rejestruje się -> Użytkownik loguje się -> Użytkownik rezerwuje usługę.
- Oczekiwany wynik: użytkownik łatwo znajduje potrzebne funkcje.
- Rzeczywisty wynik: użytkownik łatwo znajduje potrzebne funkcje.
- Status: Pass
- Uwagi: brak uwag.

Kompleksowe testowanie aplikacji "Serwis samochodowy" obejmuje różne rodzaje testów, które mają na celu zapewnienie, że aplikacja działa poprawnie, jest bezpieczna, wydajna i łatwa w użyciu. Każdy rodzaj testowania wnosi istotne informacje, które pomagają w identyfikacji i naprawie błędów oraz w doskonaleniu aplikacji przed jej wdrożeniem.

Warstwa Logiki Biznesowej

Aplikacja "Serwis samochodowy" jest zaprojektowana w architekturze wielowarstwowej, która składa się z trzech głównych warstw: Warstwy Logiki Biznesowej, Warstwy Prezentacji oraz Bazy Danych. Każda z tych warstw pełni odrębną funkcję i współpracuje z pozostałymi warstwami, aby zapewnić sprawne działanie aplikacji.

Warstwa Logiki Biznesowej jest kluczowym elementem architektury aplikacji "Serwis samochodowy". Odpowiada za przetwarzanie danych, wykonywanie operacji biznesowych oraz komunikację między warstwą prezentacji a warstwą danych.
Warstwa Logiki Biznesowej jest odpowiedzialna za implementację zasad i reguł biznesowych aplikacji. To właśnie tutaj realizowane są operacje związane z rejestracją użytkowników, logowaniem, zarządzaniem usługami i rezerwacjami, a także komunikacja z bazą danych.

Kluczowe elementy:

- Kontrolery: obsługują żądania przychodzące z Warstwy Prezentacji, przetwarzają je i zwracają odpowiednie odpowiedzi.
- Serwisy: zawierają logikę biznesową, przetwarzają dane i wykonują operacje związane z usługami serwisowymi, rezerwacjami.
- Repozytoria: interfejsy do komunikacji z Bazą Danych, odpowiedzialne za zapisywanie i pobieranie danych.

Główne funkcje warstwy Logiki Biznesowej

1) Zarządzanie użytkownikami:

- Rejestracja nowych użytkowników.
- Logowanie użytkowników.
- Aktualizacja danych użytkownika.
- Usuwanie konta użytkownika.

2) Zarządzanie usługami:

- Dodawanie nowych usług.
- Aktualizacja istniejących usług.
- Usuwanie usług.
- Pobieranie listy dostępnych usług.

3) Rezerwacje usług:

- Rezerwacja usług przez użytkowników.
- Aktualizacja rezerwacji.
- Anulowanie rezerwacji.
- Pobieranie historii rezerwacji użytkownika.

4) Komunikacja z bazą danych:

- Przesyłanie zapytań do bazy danych.
- Odbieranie i przetwarzanie odpowiedzi z bazy danych.
- Mapowanie wyników zapytań na obiekty domeny aplikacji.

Struktura Warstwy Logiki Biznesowej

Warstwa Logiki Biznesowej jest zorganizowana w kilka głównych komponentów i klas, które realizują poszczególne funkcje aplikacji:

1) Klasa UserService:
Metody:

- registerUser(userData): rejestruje nowego użytkownika w systemie.
- loginUser(credentials): loguje użytkownika na podstawie dostarczonych danych uwierzytelniających.
- updateUser(userId, newUserData): aktualizuje dane użytkownika.
- deleteUser(userId): usuwa konto użytkownika.

2) Klasa ServiceManager:
Metody:

- addService(serviceData): dodaje nową usługę do bazy danych.
- updateService(serviceId, newServiceData): aktualizuje dane istniejącej usługi.
- deleteService(serviceId): usuwa usługę z bazy danych.
- getServices(): pobiera listę wszystkich dostępnych usług.

3) Klasa BookingManager:
Metody:

- bookService(bookingData): rezerwuje usługę na podstawie dostarczonych danych.
- updateBooking(bookingId, newBookingData): aktualizuje dane istniejącej rezerwacji.
- cancelBooking(bookingId): anuluje rezerwację.
- getUserBookings(userId): pobiera historię rezerwacji użytkownika.

4) Klasa `DatabaseConnector:
Metody:

- executeQuery(query): wykonuje zapytanie do bazy danych.
- fetchResults(query): pobiera wyniki zapytania z bazy danych.

Przykład przepływu operacji

Rejestracja użytkownika:

a. Warstwa prezentacji przyjmuje dane użytkownika z formularza rejestracji.
b. Dane są przekazywane do metody `registerUser` klasy `UserService`.
c. `UserService` weryfikuje dane i przesyła zapytanie do `DatabaseConnector`.
d. `DatabaseConnector` zapisuje dane użytkownika w bazie danych.
e. `UserService` zwraca wynik operacji do warstwy prezentacji.

Rezerwacja usługi:

a. Użytkownik wybiera usługę i datę rezerwacji w warstwie prezentacji.
b. Dane rezerwacji są przekazywane do metody `bookService` klasy `BookingManager`.
c. `BookingManager` weryfikuje dostępność usługi i tworzy rezerwację.
d. `DatabaseConnector` zapisuje rezerwację w bazie danych.
e. `BookingManager` zwraca potwierdzenie rezerwacji do warstwy prezentacji.

Zastosowane wzorce projektowe

Warstwa Logiki Biznesowej wykorzystuje kilka wzorców projektowych, aby zapewnić modularność, elastyczność i łatwość utrzymania kodu:

- Wzorzec `Fasada` (Facade): ułatwia interakcję między warstwą prezentacji a warstwą logiki biznesowej.
- Wzorzec `Repozytorium` (Repository): umożliwia izolację logiki dostępu do danych od logiki biznesowej.
- Wzorzec `Usługa` (Service): organizuje logikę biznesową w zrozumiałe i zarządzalne komponenty.

Warstwa Prezentacji

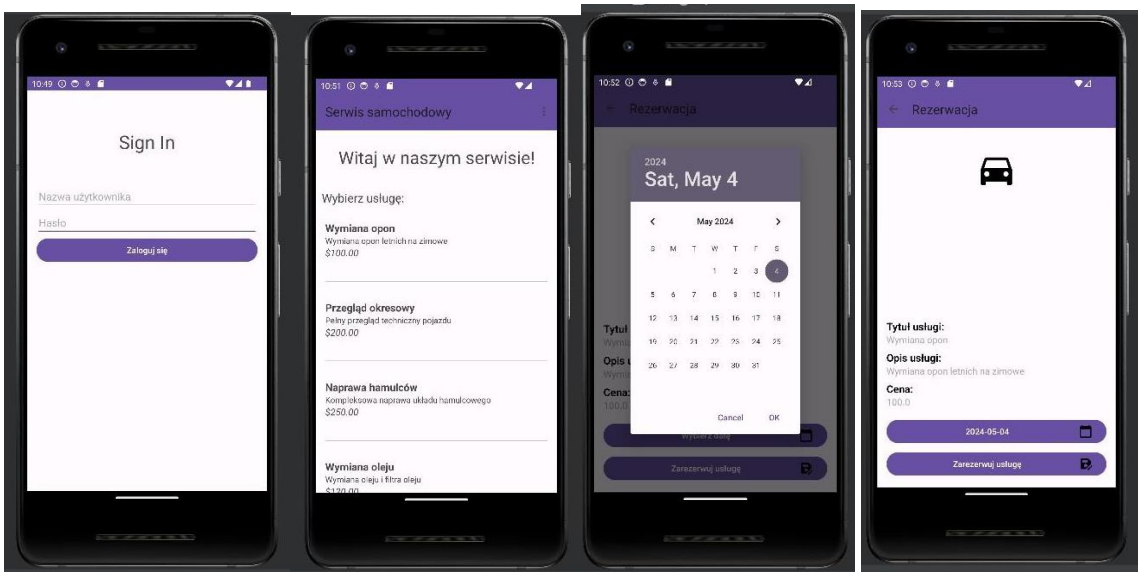
Warstwa Prezentacji jest odpowiedzialna za interakcję z użytkownikiem. To tutaj znajdują się wszystkie elementy interfejsu użytkownika (UI) oraz logika związana z interakcjami użytkownika z aplikacją.

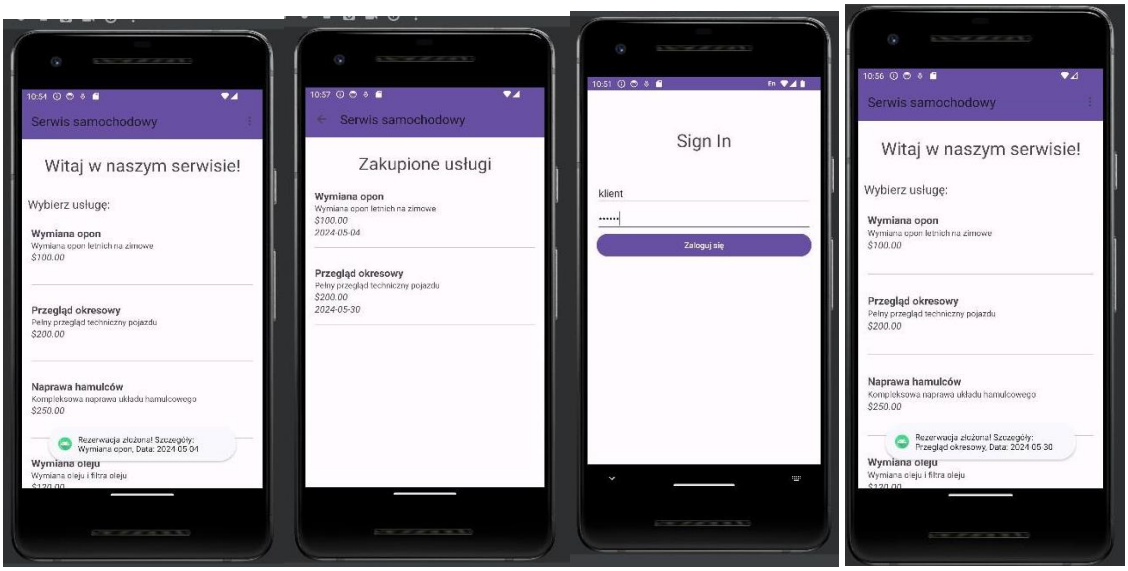
Kluczowe elementy:

- Ekrany i widoki: interfejsy użytkownika, które prezentują dane i pozwalają na interakcję (ekrany logowania, przeglądania usług, dokonywania rezerwacji).
- Kontrolery UI: obsługują interakcje użytkownika, wysyłają żądania do Warstwy Logiki Biznesowej i aktualizują widoki na podstawie otrzymanych odpowiedzi.
- Komponenty UI: przyciski, formularze, listy i inne elementy interfejsu, które umożliwiają użytkownikom korzystanie z aplikacji.

Ekrany:

- Ekran logowania i rejestracji użytkownika.
- Ekran przeglądania dostępnych usług serwisowych.
- Ekran szczegółów usługi z opcją wyboru daty i rezerwacji.
- Zakładka „Zakupione usługi” z historią transakcji.




Serwis samochodowy
Witaj w naszym serwisie!
Wybierz usługę:
Wymiana opon
Wymiana opon letnich na zimowe
$100.00
Przegląd okresowy
Pełny przegląd techniczny pojazdu
$200.00
Naprawa hamulców
Kompleksowa naprawa układu hamulcowego
$250.00
Rezerwacja złożona! Szczegóły: Wymiana opon, Data: 2024-05-04
Wymiana oleju
Wymiana oleju i filtra oleju
Serwis samochodowy
Zakupione usługi
Wymiana opon
Wymiana opon letnich na zimowe
$100.00
2024-05-04
Przegląd okresowy
Pełny przegląd techniczny pojazdu
$200.00
2024-05-30
Sign In
klient
Zaloguj się
Serwis samochodowy
Witaj w naszym serwisie!
Wybierz usługę:
Wymiana opon
Wymiana opon letnich na zimowe
$100.00
Przegląd okresowy
Pełny przegląd techniczny pojazdu
$200.00
Naprawa hamulców
Kompleksowa naprawa układu hamulcowego
$250.00
Rezerwacja złożona! Szczegóły: Przegląd okresowy, Data: 2024-05-30
Wymiana oleju
Wymiana oleju i filtra oleju

Baza Danych

Baza Danych jest odpowiedzialna za trwałe przechowywanie wszystkich danych aplikacji. Zawiera informacje o użytkownikach, usługach serwisowych, rezerwacjach, płatnościach i innych niezbędnych danych.

Kluczowe elementy:

- Tabele: struktury danych w bazie, które przechowują informacje (tabele użytkowników, usług, rezerwacji, transakcji).
- Indeksy: mechanizmy wspomagające szybkie wyszukiwanie i dostęp do danych.
- Relacje: powiązania między tabelami, które definiują, jak dane są ze sobą powiązane (relacje między użytkownikami a rezerwacjami).

Tabele:

- Users: przechowuje informacje o użytkownikach (ID, imię, nazwisko, email, hasło).
- Services: przechowuje informacje o dostępnych usługach (ID, tytuł, opis, cena).
- Bookings: przechowuje informacje o rezerwacjach usług (ID, ID użytkownika, ID usługi, data rezerwacji, status).
- Transactions: przechowuje informacje o płatnościach (ID, ID użytkownika, ID rezerwacji, kwota, data transakcji).

Podsumowanie

Architektura aplikacji "Serwis samochodowy" jest zaprojektowana tak, aby była modularna i łatwa w zarządzaniu. Warstwa Logiki Biznesowej obsługuje przetwarzanie danych i zasady biznesowe, Warstwa Prezentacji zajmuje się interakcją z użytkownikiem, a Baza Danych zapewnia trwałe przechowywanie wszystkich potrzebnych informacji. Dzięki tej strukturze aplikacja jest skalowalna, bezpieczna i łatwa do rozwijania w przyszłości.