1. Tytuł projektu.

Projekt aplikacji do zarządzania wypożyczalnią samochodów.

1. Opis ogólny projektu.

Celem projektu jest stworzenie aplikacji do obsługi wypożyczalni samochodów, która umożliwia zarówno klientom, jak i pracownikom wygodne zarządzanie rezerwacjami, flotą pojazdów oraz historią użytkowania. System realizuje kluczowe procesy biznesowe, takie jak:

* przeglądanie dostępnych pojazdów,
* rezerwacja samochodu na określony termin,
* anulowanie lub modyfikacja rezerwacji,
* zwrot pojazdu,
* przetwarzanie płatności,
* zarządzanie flotą przez pracowników.

Projekt został zrealizowany w języku Python z użyciem podejścia obiektowego. System wspiera testowanie jednostkowe i dokumentację UML, aby zapewnić wysoką jakość kodu i przejrzystość architektury.

1. **Cele funkcjonalne i niefunkcjonalne**

Cele funkcjonalne:

* Rejestracja i logowanie klientów oraz pracowników
* Przeglądanie listy dostępnych pojazdów
* Tworzenie i anulowanie rezerwacji
* Zarządzanie samochodami przez pracowników
* Obsługa płatności oraz kalkulacja kosztów wypożyczenia

Cele niefunkcjonalne:

* Przejrzysty interfejs kodu (czytelność, rozdział klas)
* Modularna struktura systemu
* Łatwe testowanie i rozwój
* Kompatybilność z systemami kontroli wersji (GitHub)

1. Opis działania systemu

### Logowanie i uwierzytelnianie

System odróżnia użytkowników na podstawie typu konta (Klient / Pracownik). Po zalogowaniu użytkownik otrzymuje dostęp do odpowiednich funkcji.

### Rezerwacja samochodu

Klient wybiera samochód i określa daty. System weryfikuje dostępność pojazdu, oblicza koszt rezerwacji i tworzy wpis.

### Anulowanie rezerwacji

Użytkownik może anulować rezerwację przed datą rozpoczęcia – samochód wraca do puli dostępnych.

### Zwrot i przedłużenie

System pozwala na oznaczenie zwrotu pojazdu oraz na przedłużenie trwającej rezerwacji, jeśli pojazd nie jest już zarezerwowany.

### Zarządzanie flotą

Pracownicy mogą dodawać i usuwać samochody z listy. System uniemożliwia usunięcie pojazdu z aktywną rezerwacją.

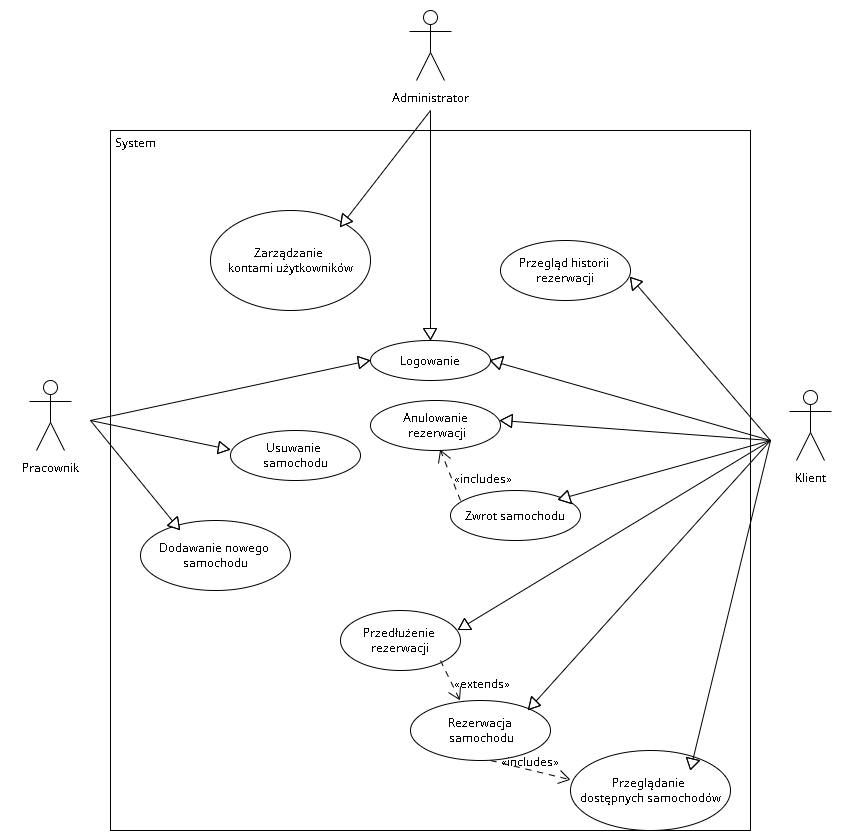
1. Diagramy UML

a) Diagram przypadków użycia

Przedstawia interakcje między aktorami systemu (Klient, Pracownik, Administrator) a funkcjami aplikacji:

* Klient: logowanie, przeglądanie aut, rezerwacje, historia
* Pracownik: dodawanie/usuwanie pojazdów
* Administrator: zarządzanie kontami

Zawiera zależności <<include>> oraz <<extend>>.

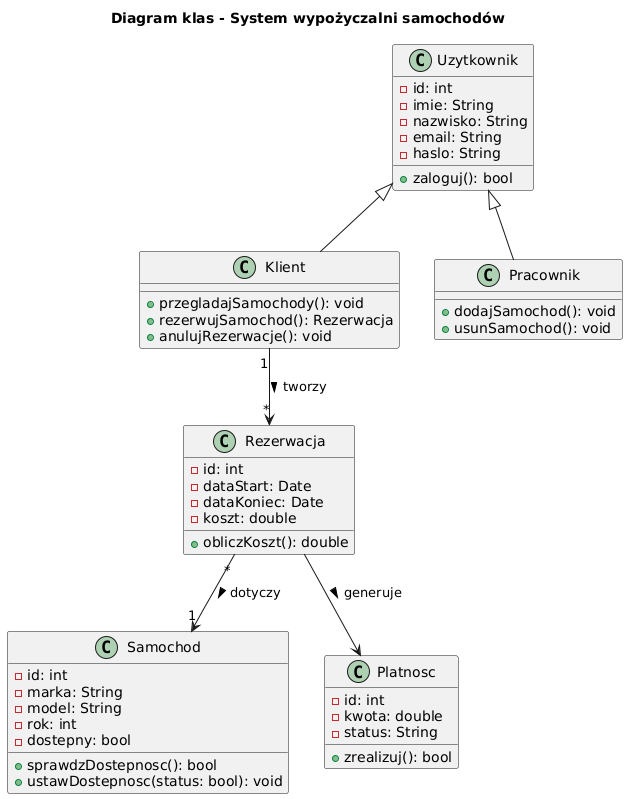


Rys.1 Diagram przypadków

b) Diagram klas

Przedstawia strukturę aplikacji:

* Klasy dziedziczące po Uzytkownik: Klient, Pracownik
* Samochod, Rezerwacja, Platnosc – powiązane relacjami asocjacji
* Metody i pola klas odzwierciedlają logikę systemu

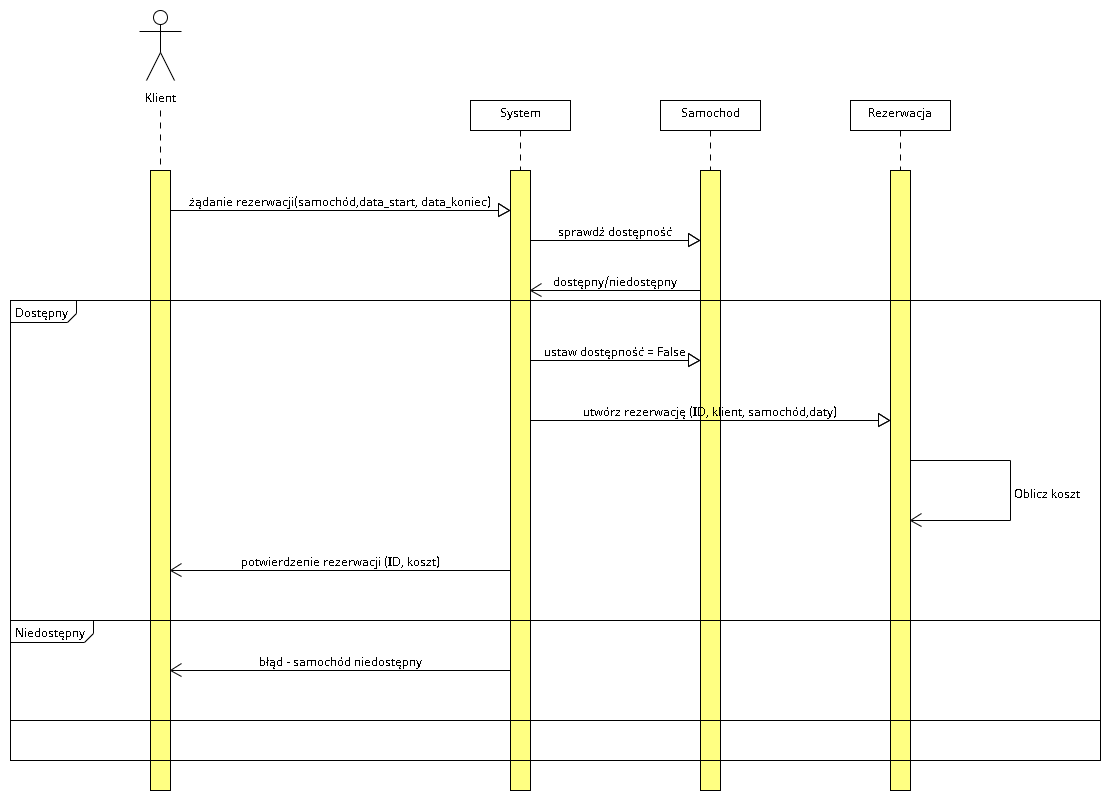


Rys.2 Diagram klas

c) Diagram sekwencji

Symuluje proces rezerwacji:

* Klient wysyła żądanie rezerwacji
* System sprawdza dostępność
* Tworzy obiekt rezerwacji i oblicza koszt
* Generuje potwierdzenie lub komunikat błędu

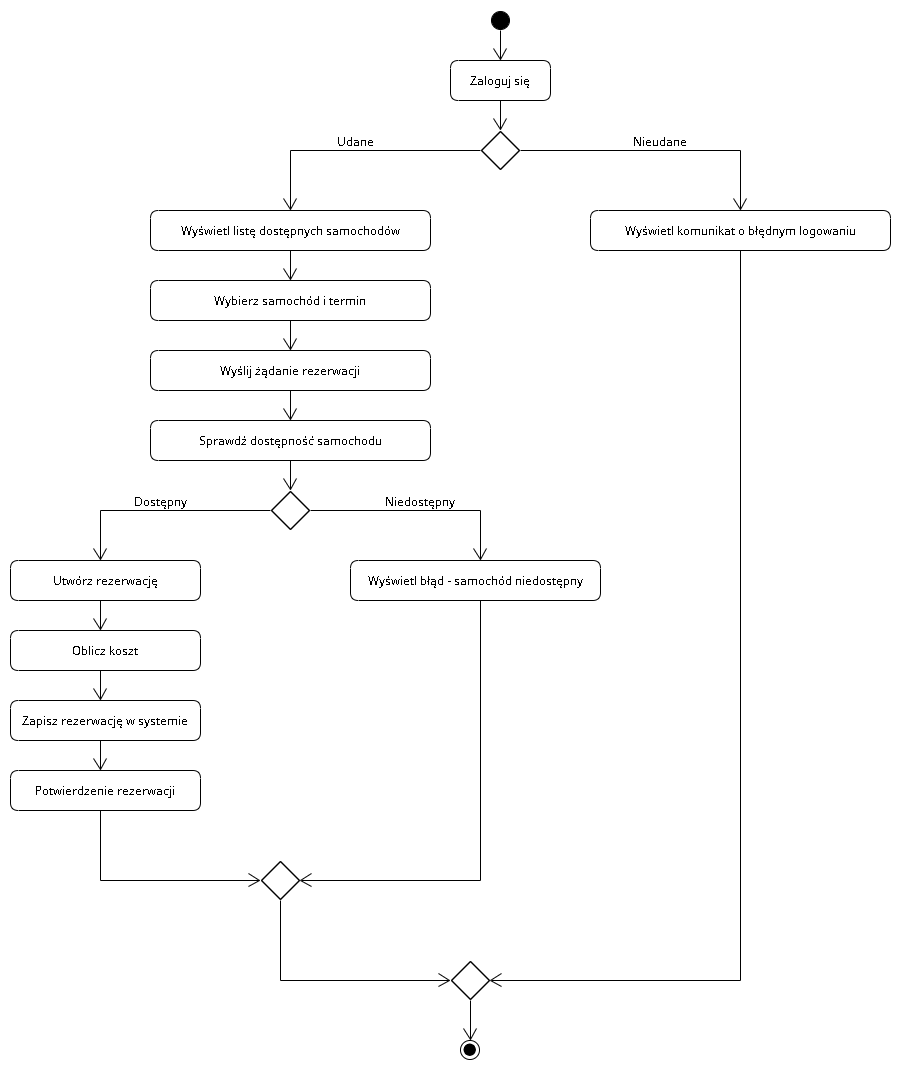


Rys.3 Diagram sekwencji

d) Diagram aktywnośći

Opisuje scenariusz działania użytkownika:

* Logowanie → przeglądanie aut → wybór auta → rezerwacja → potwierdzenie/błąd



Rys.4 Diagram aktywności

1. Opis funkcji i testów

Funkcja: dodaj\_samochod()

* Dodaje nowy samochód do floty
* Sprawdza unikalność ID

Testy:

1. Dodanie nowego auta
2. Próba dodania auta z istniejącym ID
3. Dodanie auta z danymi null
4. Dodanie kilku aut i sprawdzenie listy

Funkcja: usun\_samochod()

* Usuwa samochód z listy, jeśli nie ma aktywnej rezerwacji

Testy:

1. Usunięcie auta dostępnego
2. Usunięcie auta z rezerwacją – błąd
3. Usunięcie auta nieistniejącego
4. Lista po usunięciu

Funkcja: rezerwuj\_samochod()

* Tworzy nową rezerwację

Testy:

1. Rezerwacja dostępnego auta
2. Rezerwacja tego samego auta przez innego użytkownika – błąd
3. Rezerwacja z nieprawidłową datą
4. Obliczenie poprawnego kosztu

Funkcja: anuluj\_rezerwacje()

* Anuluje istniejącą rezerwację

Testy:

1. Anulowanie istniejącej rezerwacji
2. Próba anulowania już zakończonej rezerwacji
3. Próba anulowania cudzej rezerwacji
4. Samochód staje się dostępny

Funkcja: oblicz\_koszt()

* Oblicza cenę wypożyczenia na podstawie liczby dni

Testy:

1. Koszt 1-dniowej rezerwacji
2. Koszt wielodniowej rezerwacji
3. Rezerwacja weekendowa
4. Rezerwacja z błędną datą – błąd
5. Zastosowane technologie

| Technologia | Opis |
| --- | --- |
| Python | Główny język programowania |
| pytest | Framework testowy |
| PlantUML | Tworzenie diagramów |
| Git + GitHub | Zarządzanie kodem i wersjami |
| UMLetino | Narzędzie online do diagramów UML |

1. Harmonogram pracy.

| Etap | Opis |  |
| --- | --- | --- |
| Tydzień 1 | Inicjalizacja repozytorium, diagram przypadków użycia |  |
| Tydzień 2 | Implementacja klas i funkcji bazowych |  |
| Tydzień 3 | Testy jednostkowe i poprawki |  |
| Tydzień 4 | Diagramy UML, sprawozdanie |  |

1. Podział obowiązków.

Jurkiewicz: Projektowanie UML + Implementacja funkcji aplikacji Odpowiedzialny za projektowanie diagramów UML oraz implementację głównych funkcji aplikacji.

Jarończyk: Testowanie + Zapewnienie jakości Odpowiedzialny za tworzenie testów jednostkowych, testowanie funkcji oraz zapewnienie jakości kodu i optymalizację aplikacji.

Komorowski: Sprawozdanie + Prezentacja Odpowiedzialny za dokumentację projektu, przygotowanie prezentacji projektu oraz podsumowanie wyników pracy zespołu.

1. Wnioski i obserwacje.

Projekt pozwolił na praktyczne zastosowanie znajomości:

* programowania obiektowego w Pythonie,
* analizy systemu za pomocą UML,
* pracy zespołowej z GitHubem,
* testowania jednostkowego i dbałości o jakość kodu.

Projekt może być rozszerzony o interfejs webowy z użyciem Flask lub Django.