Wypożyczalnia hulajnóg elektrycznych

Bazy Danych 2023

Autorzy: Michał Skałka, Igor Swat, Jakub Płowiec

Technologie (backend): Oracle Database, Node.js, Express

Spis treści

Funkcje w systemie

Opis użytkowników:

- **Klient** - możliwości rejestracji i logowania do aplikacji, czasowe wypożyczenia dostępnych pojazdów, dostęp do statystyk użytkownika i historii wypożyczenia
- **Administrator** – ustawianie statusu dostępności i innych danych dotyczących pojazdów, dodawanie nowych pojazdów, wyświetlanie rezerwacji i zarządzanie hasłami użytkowników

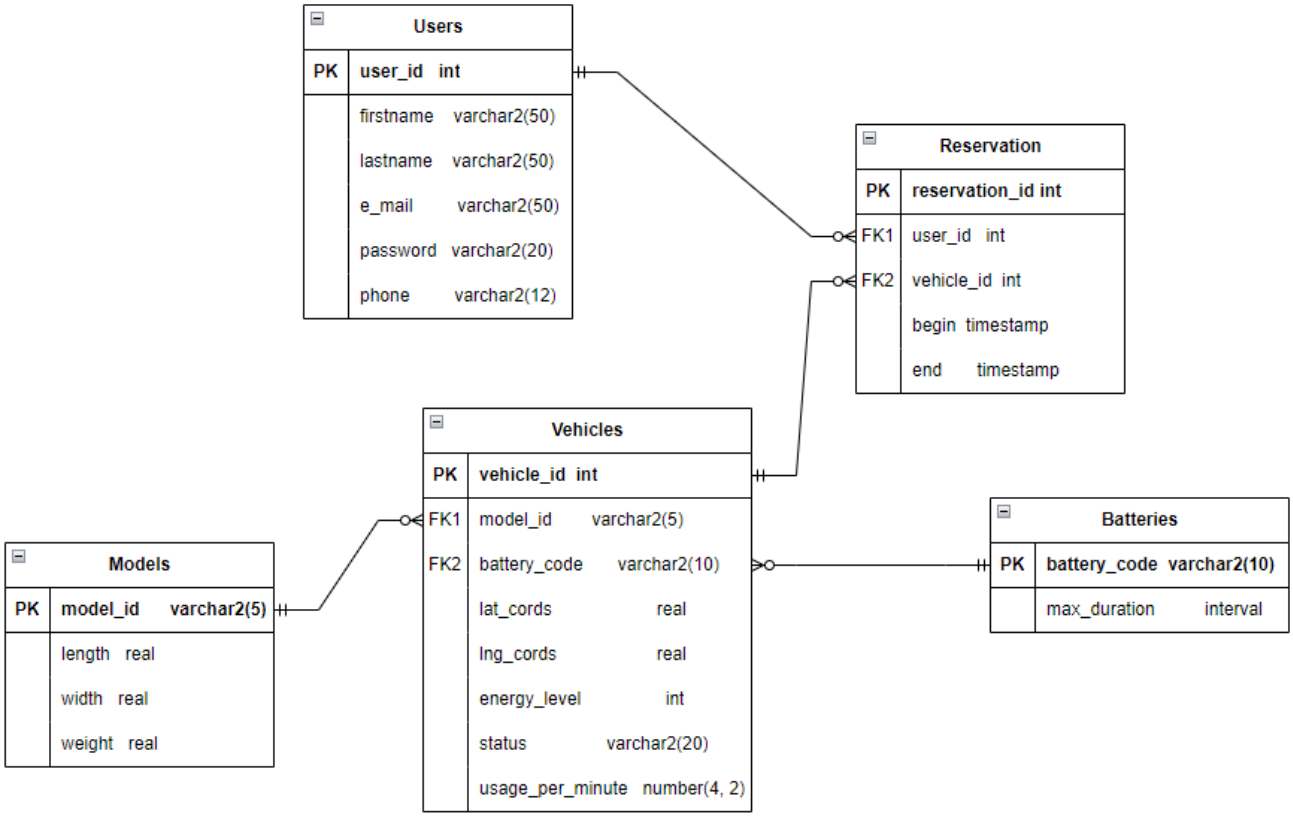
Funkcje użytkowników:

- **Klient:**
  1. Rejestracja w systemie
  2. Logowanie do aplikacji
  3. Przegląd dostępnych pojazdów
  4. Rezerwacja czasowa wybranego pojazdu
  5. Wygenerowanie statystyk własnych rezerwacji
- **Administrator:**
  1. Zmiana statusu dostępności wskazanego pojazdu
  2. Aktualizacja danych pojazdu
  3. Dodanie nowego pojazdu do bazy danych
  4. Wyświetlenie listy wszystkich rezerwacji
  5. Zmiana hasła użytkownika

Funkcje systemowe:

1. Walidacja danych podanych w procesie logowania
2. Wskazanie aktualnie dostępnych pojazdów w zadanym obszarze
3. Obliczenie kosztu rezerwacji pojazdu
4. Aktualizacja stanu energii pojazdu
5. Obliczenie statystyk użytkownika

Schemat bazy danych



Opis tabel

Tabela Users

Przechowuje dane użytkowników zarejestrowanych w systemie.

```
create table USERS
(
    USER_ID   NUMBER generated as identity
        constraint USERS_PK
            primary key,
    FIRSTNAME VARCHAR2(50) not null,
    LASTNAME  VARCHAR2(50) not null,
    E_MAIL    VARCHAR2(50) not null
        constraint USERS_UN1
            unique
        constraint USERS_CHK1
            check (e_mail like '%@%'),
    PHONE     VARCHAR2(12)
        constraint USERS_CHK2
            check (regexp_like(PHONE, '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')),
    PASSWORD  VARCHAR2(20)
        constraint USERS_CHK3
            check (LENGTH(PASSWORD) > 5)
)
```

- **Klucz główny:** USER_ID
- **Imię użytkownika:** FIRSTNAME
- **Nazwisko użytkownika:** LASTNAME
- **Adres e-mail:** E_MAIL – wykorzystywany podczas autentykacji
- **Numer telefonu:** PHONE
- **Hasło:** PASSWORD – wykorzystywane podczas autentykacji

Tabela Reservation

Przechowuje dane o rezerwacjach użytkowników na poszczególne pojazdy.

```
create table RESERVATION
(
    RESERVATION_ID NUMBER generated as identity
        constraint RESERVATION_PK
            primary key,
    USER_ID        NUMBER not null
        constraint RESERVATION_FK1
            references USERS,
    VEHICLE_ID     NUMBER not null
        constraint RESERVATION_FK2
            references VEHICLE,
    R_BEGIN        DATE   not null,
    R_END          DATE   not null,
    constraint RESERVATION_CHK1
        check (R_BEGIN < R_END)
)
```

- **Klucz główny:** RESERVATION_ID
- **ID użytkownika:** USER_ID
- **ID rezerwowanego pojazdu:** VEHICLE_ID
- **Początek rezerwacji:** R_BEGIN
- **Koniec rezerwacji:** R_END

Tabela Vehicle

Przechowuje ogólne parametry pojazdu wpływające na kwestie rezerwacji, takie jak aktualny stan enegii, lokalizacja pojazdu czy koszt za minutę rezerwacji.

```
create table VEHICLE
(
    VEHICLE_ID      NUMBER generated as identity
        constraint VEHICLES_PK
            primary key,
    MODEL_ID        VARCHAR2(5)  not null
        constraint VEHICLES_FK1
            references MODEL,
    BATTERY_CODE    VARCHAR2(10) not null
        constraint VEHICLES_FK2
            references BATTERY,
    LAT_CORDS       FLOAT(63)    not null
        constraint VEHICLES_CHK1
            check (LAT_CORDS between -90 and 90),
    LNG_CORDS       FLOAT(63)    not null
        constraint VEHICLES_CHK2
            check (LNG_CORDS between -180 and 180),
    STATUS          VARCHAR2(20) not null
        constraint VEHICLES_CHK4
            check (STATUS in ('Available', 'Not available')),
    ENERGY_LEVEL    NUMBER       not null
        constraint VEHICLES_CHK3
            check (ENERGY_LEVEL between 0 and 100),
    COST_PER_MINUTE NUMBER(4, 2) not null
        constraint VEHICLES_CHK5
            check (COST_PER_MINUTE >= 0)
)
```

- **Klucz główny:** VEHICLE_ID
- **ID modelu pojazdu:** MODEL_ID
- **ID baterii / akumulatora:** BATTERY_CODE
- **Szerokość geograficzna:** LAT_COORDS
- **Długość geograficzna:** LNG_COORDS

- **Aktualny status dostępności pojazdu:** STATUS – jest to odgórny status ustawiany przez administratora, nie informuje o aktualnych rezerwacjach
- **Poziom energii:** ENERGY_LEVEL (0 -100 [%])
- **Koszt minuty rezerwacji:** COST_PER_MINUTE

Tabela Model

Przechowuje informacje o fizycznym rozmiarze i rodzaju danego pojazdu.

```
create table MODEL
(
    MODEL_ID  VARCHAR2(5) not null
        constraint MODEL_PK
            primary key,
    LENGTH_M  FLOAT(63)   not null
        constraint MODEL_CHK1
            check (LENGTH_M > 0),
    WIDTH_M   FLOAT(63)   not null
        constraint MODEL_CHK2
            check (WIDTH_M > 0),
    WEIGHT_KG FLOAT(63)   not null
        constraint MODEL_CHK3
            check (WEIGHT_KG > 0)
)
```

- **Klucz główny:** MODEL_ID – etykieta modelu
- **Długość pojazdu:** LENGTH_M [m]
- **Szerokość pojazdu:** WIDTH_M [m]
- **Waga pojazdu:** WEIGHT_KG [kg]

Tabela Battery

Reprezentuje modele baterii / akumulatorów i przechowuje informacje o ich maksymalnej żywotności

```
create table BATTERY
(
    BATTERY_CODE VARCHAR2(10)                    not null
        constraint BATTERY_PK
            primary key,
    MAX_DURATION INTERVAL DAY(2) TO SECOND(6) not null
        constraint BATTERY_CHK1
            check (max_duration > interval '0' second)
)
```

- **Klucz główny:** BATTERY_CODE
- **Maksymalny czas korzystania z baterii:** MAX_DURATION – na bazie tej wartości i procentowego stanu energii pojazdu obliczany jest pozostały czas działania pojazdu bez ładowania

Widoki

Widok vehicles_to_charge

Widok udostępnia listę pojazdów wymagających ładowania (o zerowym stanie energii).

```
create view VEHICLES_TO_CHARGE as
select VEHICLE_ID, BATTERY_CODE, LAT_CORDS, LNG_CORDS
    from VEHICLE
    where ENERGY_LEVEL = 0
```

Widok vehicles_to_charge

Widok stanowi połączenie informacji z tabel Vehicle, Battery i Model, prezentując kompletne informacje o pojazdach.

```
create view VEHICLES as
select v.VEHICLE_ID, v.MODEL_ID, m.LENGTH_M, m.WIDTH_M, m.WEIGHT_KG,
          b.BATTERY_CODE, b.MAX_DURATION, v.COST_PER_MINUTE, v.STATUS
    from VEHICLE v
    join MODEL m
    on v.MODEL_ID = m.MODEL_ID
    join BATTERY b
    on v.BATTERY_CODE = b.BATTERY_CODE
```

Widok current_reservations

Prezentuje listę aktualnie trwających rezerwacji wraz z danymi użytkowników będących właścicielami rezerwacji.

```
create view CURRENT_RESERVATIONS as
select r.RESERVATION_ID, u.USER_ID, u.FIRSTNAME, u.LASTNAME, u.E_MAIL, r.VEHICLE_ID, r.r_BEGIN, r.r_END
    from RESERVATION r
    join USERS u
    on r.USER_ID = u.USER_ID
    where sysdate between r.R_BEGIN and r.R_END
```

Widok available_vehicles

Widok wyświetla wszystkie aktualnie dostępne do rezerwacji pojazdy – pomijane są pojazdy o statusie niedostępnym, pojazdy o zerowym stanie energii oraz pojazdy aktualnie zarezerwowane przez innych użytkowników

```
create view AVAILABLE_VEHICLES as
select v.VEHICLE_ID, v.MODEL_ID, v.LAT_CORDS, v.LNG_CORDS, b.MAX_DURATION * v.ENERGY_LEVEL / 100 as duration,
       v.ENERGY_LEVEL, v.COST_PER_MINUTE
    from VEHICLE v
    join BATTERY b
    on v.BATTERY_CODE = b.BATTERY_CODE
    left join CURRENT_RESERVATIONS cr
    on v.VEHICLE_ID = cr.VEHICLE_ID
    where cr.VEHICLE_ID is null and v.STATUS like 'Available' and v.ENERGY_LEVEL > 0
```

Widok unavailable_vehicles

Wyświetla listę pojazdów zablokowanych przez administratora.

```
create or replace view UNAVAILABLE_VEHICLES as
select v.VEHICLE_ID, v.MODEL_ID, v.LAT_CORDS, v.LNG_CORDS, b.MAX_DURATION * v.ENERGY_LEVEL / 100 as duration,
       v.ENERGY_LEVEL, v.COST_PER_MINUTE
    from VEHICLE v
    join BATTERY b
    on v.BATTERY_CODE = b.BATTERY_CODE
    left join CURRENT_RESERVATIONS cr
    on v.VEHICLE_ID = cr.VEHICLE_ID
    where cr.VEHICLE_ID is null and v.STATUS like 'Not available'
```

Widok reservations

Widok prezentuje informacje o wszystkich rezerwacjach z uwzględnieniem obliczonego kosztu każdej rezerwacji.

```
create view RESERVATIONS as
select r.RESERVATION_ID, r.USER_ID, r.VEHICLE_ID, r.R_BEGIN, r.R_END,
          ((extract(day from CAST(r.R_END as timestamp)) -
               extract(day from CAST(r.R_BEGIN as timestamp))) * 24 * 60 +
          (extract(hour from CAST(r.R_END as timestamp)) -
               extract(hour from CAST(r.R_BEGIN as timestamp))) * 60 +
          (extract(minute from CAST(r.R_END as timestamp)) -
               extract(minute from CAST(r.R_BEGIN as timestamp)))) * v.COST_PER_MINUTE as cost
    from RESERVATION r
    join VEHICLE v
    on r.VEHICLE_ID = v.VEHICLE_ID
```

Widok users_stats

Wyświetla statystyki każdego użytkownika dotyczące liczby i łącznego kosztu wszystkich dokonanych przez niego rezerwacji.

```
create view USERS_STATS as
select u.USER_ID, count(r.USER_ID) as no_reservations, nvl(sum(r.COST), 0) as total_cost
    from USERS u
    left join RESERVATIONS r
    on u.USER_ID = r.USER_ID
    group by u.USER_ID
```

Widok users_info

Statystyki rezerwacji użytkowników wzbogacone o podstawowe dane użytkownika.

```
create view USERS_INFO as
select u.USER_ID, u.FIRSTNAME, u.LASTNAME, u.E_MAIL, u.PHONE, us.NO_RESERVATIONS, us.TOTAL_COST
    from USERS u
    join USERS_STATS us
    on u.USER_ID = us.USER_ID
```

Widok vehicle_models

Prezentuje statystyki dotyczące liczby wykorzystywanych modeli wśród dostępnych pojazdów.

```
create view VEHICLE_MODELS as
select m.MODEL_ID, m.LENGTH_M, m.WIDTH_M, m.WEIGHT_KG, count(v.VEHICLE_ID) as total_vehicles
    from MODEL m
    left join VEHICLE v
    on m.MODEL_ID = v.MODEL_ID
    group by m.MODEL_ID, m.LENGTH_M, m.WIDTH_M, m.WEIGHT_KG
```

Widok rented_vehicles

Wyświetla aktualnie wypożyczone pojazdy.

```
create or replace view RENTED_VEHICLES as
SELECT USER_ID, v.VEHICLE_ID, v.MODEL_ID, v.LAT_CORDS, v.LNG_CORDS, b.MAX_DURATION * v.ENERGY_LEVEL / 100 as duration,
v.ENERGY_LEVEL, v.COST_PER_MINUTE FROM CURRENT_RESERVATIONS INNER JOIN VEHICLE V
on CURRENT_RESERVATIONS.VEHICLE_ID = V.VEHICLE_ID join BATTERY b
on v.BATTERY_CODE = b.BATTERY_CODE
```

Funkcje

Funkcja users_exist_by_id

Zwraca true, gdy użytkownik o zadanym numerze ID istnieje w bazie danych oraz false w przeciwnym przypadku

```
create function user_exist_by_id(u_id int)
    return boolean
as
    found int;
begin
    select count(*) into found from USERS where USER_ID = u_id;
    if found = 0 then
        return false;
    else
        return true;
    end if;
end;
```

Funkcja users_exist_by_auth

Zwraca true, gdy użytkownik o podanym adresie e-mail i haśle istnieje w bazie danych oraz false w przeciwnym przypadku

```
create function user_exist_by_auth(email varchar2, paswd varchar2)
    return boolean
as
    found int;
begin
    select count(*) into found from USERS where E_MAIL = email and PASSWORD = paswd;
    if found = 0 then
        return false;
    else
        return true;
    end if;
end;
```

Funkcja get_user_id

Zwraca numer id użytkownika o podanym adresie e-mail i haśle

```
create function get_user_id(email varchar2, paswd varchar2)
    return int
as
    u_id int;
begin
    select USER_ID into u_id from USERS where E_MAIL = email and PASSWORD = paswd;
    return u_id;
exception
    when NO_DATA_FOUND then
        raise_application_error(-20001, 'user not found');
        return -1;
end;
```

Funkcja get_user_info

Zwraca dane podanego użytkownika i jego statystyki rezerwacji

```
CREATE OR REPLACE FUNCTION get_user_info(u_id INT)
    RETURN USER_STATS_TABLE
AS
    result USER_STATS_TABLE;
BEGIN
    select USER_STATS(u.USER_ID, u.FIRSTNAME, u.LASTNAME, u.E_MAIL, u.PHONE, u.NO_RESERVATIONS, u.TOTAL_COST)
    bulk collect into result
    from USERS_INFO u
    where USER_ID = u_id;
    return result;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'user not found');
        RETURN null;
END;
```

Funkcja wykorzystuje zdefiniowane typy danych USER_STATS oraz USER_STATS_TABLE:

```
create or replace type user_stats as object
(
    USER_ID int,
    FIRSTNAME varchar2(50),
    LASTNAME varchar2(50),
    E_MAIL varchar2(50),
    PHONE varchar2(12),
    NO_RESERVATIONS int,
    TOTAL_COST float
)
/

create or replace type USER_STATS_TABLE as table of user_stats
```

Funkcja get_user_all_reservations

Zwraca historię rezerwacji podanego użytkownika

```
create function get_user_all_reservations(u_id int)
    return USER_RESERVATION_TABLE
as
    result USER_RESERVATION_TABLE;
begin
    select USER_RESERVATION(r.USER_ID, r.RESERVATION_ID, r.VEHICLE_ID, r.R_BEGIN, r.R_END, r.COST)
    bulk collect into result
    from RESERVATIONS r
    where USER_ID = u_id;
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych USER_RESERVATION oraz USER_RESERVATION_TABLE:

```
create type user_reservation as object
(
    user_id int,
    reservation_id int,
    vehicle_id int,
    r_begin date,
    r_end date,
    cost real
)
```

```
create type USER_RESERVATION_TABLE as table of USER_RESERVATION
```

Funkcja get_user_current_reservations

Zwraca aktualnie trwające rezerwacje podanego użytkownika

```
create function get_user_current_reservations(u_id int)
    return USER_RESERVATION2_TABLE
as
    result USER_RESERVATION2_TABLE;
begin
    select USER_RESERVATION2(cr.RESERVATION_ID, cr.VEHICLE_ID, cr.R_BEGIN, cr.R_END)
    bulk collect into result
    from CURRENT_RESERVATIONS cr
    where cr.USER_ID = u_id;
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych USER_RESERVATION2 oraz USER_RESERVATION2_TABLE:

```
create type user_reservation2 as object
(
    reservation_id int,
    vehicle_id int,
    r_begin date,
    r_end date
)
```

```
create type USER_RESERVATION2_TABLE as table of USER_RESERVATION2
```

Funkcja get_reservations_by_vehicle

Zwraca historię rezerwacji dla podanego pojazdu.

```
create function get_reservations_by_vehicle(v_id int)
    return USER_RESERVATION_TABLE
as
    result USER_RESERVATION_TABLE;
begin
    select USER_RESERVATION(r.USER_ID, r.RESERVATION_ID, r.VEHICLE_ID, r.R_BEGIN, r.R_END, r.COST)
    bulk collect into result
    from RESERVATIONS r
    where VEHICLE_ID = v_id;
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych USER_RESERVATION oraz USER_RESERVATION_TABLE (patrz str. 21).

Funkcja vehicle_exist

Zwraca true w przypadku gdy pojazd o podanym numerze ID istnieje w bazie danych oraz false w przeciwnym przypadku.

```
create function vehicle_exist(v_id int)
    return boolean
as
    result int;
begin
    select count(*) into result from VEHICLE where VEHICLE_ID = v_id;
    if result = 0 then
        return false;
    else
        return true;
    end if;
end;
```

Funkcja vehicles_in_area

Zwraca listę aktualnie dostępnych pojazdów w obrębie zadanego obszaru geograficznego.

```
create function vehicles_in_area(lat_min real, lng_min real, lat_max real, lng_max real)
    return VEHICLE_DATA_TABLE
as
    result VEHICLE_DATA_TABLE;
begin
    select VEHICLE_DATA(av.VEHICLE_ID, av.MODEL_ID, av.LAT_CORDS, av.LNG_CORDS, av.DURATION,
        av.ENERGY_LEVEL, av.COST_PER_MINUTE) bulk collect into result
    from AVAILABLE_VEHICLES av
    where (av.LAT_CORDS between lat_min and lat_max)
        and (av.LNG_CORDS between lng_min and lng_max);
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych VEHICLE_DATA oraz VEHICLE_DATA_TABLE:

```
create type vehicle_data as object
(
    vehicle_id int,
    model_id varchar2(5),
    lat_cords real,
    lng_cords real,
    duration interval day to second,
    energy_level int,
    cost_per_minute real
);
```

```
create type vehicle_data_table is table of vehicle_data;
```

Funkcja vehicles_by_model

Zwraca listę wszystkich aktualnie dostępnych pojazdów w podanym modelu

```
create function vehicles_by_model(m_id varchar2)
    return VEHICLE_DATA_TABLE
as
    result VEHICLE_DATA_TABLE;
begin
    select VEHICLE_DATA(av.VEHICLE_ID, av.MODEL_ID, av.LAT_CORDS, av.LNG_CORDS, av.DURATION,
        av.ENERGY_LEVEL, av.COST_PER_MINUTE) bulk collect into result
    from AVAILABLE_VEHICLES av
    where av.MODEL_ID = m_id;
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych VEHICLE_DATA oraz VEHICLE_DATA_TABLE (patrz str. 24).

Funkcja vehicles_by_price

Funkcja zwraca listę wszystkich aktualnie dostępnych pojazdów w zadanym przedziale cenowym za minutę rezerwacji.

```
create function vehicles_by_price(price_min real, price_max real)
    return VEHICLE_DATA_TABLE
as
    result VEHICLE_DATA_TABLE;
begin
    select VEHICLE_DATA(av.VEHICLE_ID, av.MODEL_ID, av.LAT_CORDS, av.LNG_CORDS, av.DURATION,
        av.ENERGY_LEVEL, av.COST_PER_MINUTE) bulk collect into result
    from AVAILABLE_VEHICLES av
    where av.COST_PER_MINUTE between price_min and price_max;
    return result;
end;
```

Funkcja wykorzystuje zdefiniowane typy danych VEHICLE_DATA oraz VEHICLE_DATA_TABLE (patrz str. 24).

Funkcja vehicle_info

Funkcja zwraca szczegółowe informacje o wskazanym pojeździe.

```
create function vehicle_info(v_id int)
    return VEHICLES%rowtype
as
    result VEHICLES%rowtype;
    exc1 exception;
begin
    if not VEHICLE_EXIST(v_id) then
        raise exc1;
    end if;
    select * into result
    from VEHICLES v
    where v.VEHICLE_ID = v_id;
    return result;
exception
    when exc1 then
        raise_application_error(-20002, 'vehicle not found');
        return null;
end;
```

Funkcja get_user_currently_rented_vehicles

Funkcja wyświetla wszystkie obecnie zarezerwowane pojazdy przez wskazanego użytkownika.

```
create or replace function get_user_currently_rented_vehicles(u_id int)
    return VEHICLE_DATA_TABLE
as
    result VEHICLE_DATA_TABLE;
begin
    select VEHICLE_DATA(r.VEHICLE_ID, r.MODEL_ID, r.LAT_CORDS, r.LNG_CORDS, r.duration, r.ENERGY_LEVEL, r.COST_PER_MINUTE)
    bulk collect into result
    from RENTED_VEHICLES r
    where USER_ID = u_id;
    return result;
end;
```

Funkcja get_reservation_cost

Zwraca koszt rezerwacji podanego pojazdu na zadany czas

```
create or replace function get_reservation_cost(v_id int, p_duration interval day to second)
    return real
as
    minutes int;
    cost real;
begin
    if not VEHICLE_EXIST(v_id) then
        raise_application_error(-20002, 'vehicle not found');
    end if;
    minutes := (extract(day from p_duration) * 24 * 60 +
                extract(hour from p_duration) * 60 +
                extract(minute from p_duration));
    if extract(second from p_duration) > 0 then
        minutes := minutes + 1;
    end if;
    select v.COST_PER_MINUTE into cost from VEHICLE v where v.VEHICLE_ID = v_id;
    return cost * minutes;
end;
```

Procedury

Procedura add_user

Procedura służy dodawaniu nowych użytkowników do bazy danych.
Wykorzystywana jest podczas rejestracji.

```
create procedure add_user(fname varchar2, lname varchar2, email varchar2, paswd varchar2, phone_num varchar2)
as
begin
    if email not like '%@%' then
        raise_application_error(-20003, 'incorrect e-mail address');
    end if;
    insert into USERS(firstname, lastname, e_mail, phone, password)
    values (fname, lname, email, phone_num, paswd);
exception
    when DUP_VAL_ON_INDEX then
        raise_application_error(-20004, 'e-mail address already in use');
end;
```

Procedura set_vehicle_status

Ustawia wskazany status pojazdu.

```
create procedure set_vehicle_status(v_id int, p_status varchar2)
as
begin
    if not VEHICLE_EXIST(v_id) then
        raise_application_error(-20002, 'vehicle not found');
    end if;
    if p_status not in ('Available', 'Not available') then
        raise_application_error(-20005, 'incorrect status');
    end if;
    update VEHICLE v
    set v.STATUS = p_status
    where v.VEHICLE_ID = v_id;
end;
```

Procedura add_reservation

Dodaje do tabeli Reservation nową rezerwację, w tej wersji nie jest aktualizowany stan energii pojazdu.

```
create procedure add_reservation(u_id int, v_id int, p_duration interval day to second)
as
begin
    if not USER_EXIST_BY_ID(u_id) then
        raise_application_error(-20001, 'user not found');
    end if;
    if  not VEHICLE_EXIST(v_id) then
        raise_application_error(-20002, 'vehicle not found');
    end if;
    insert into RESERVATION(user_id, vehicle_id, r_begin, r_end)
    values (u_id, v_id, current_date, current_date + p_duration);
end;
```

Procedura set_energy_by_value

Ustawia poziom energii pojazdu wg podanej wartości procentowej.

```
create procedure set_energy_by_value(v_id int, e_lvl int)
as
begin
    if not VEHICLE_EXIST(v_id) then
        raise_application_error(-20002, 'vehicle not found');
    end if;
    if not e_lvl between 0 and 100 then
        raise_application_error(-20006, 'invalid energy level');
    end if;
    update VEHICLE v
    set v.ENERGY_LEVEL = e_lvl
    where v.VEHICLE_ID = v_id;
end;
```

Procedura set_energy_by_duration

Aktualizuje poziom energii pojazdu pomniejszając o tyle, ile ubyło w podanym czasie działania baterii.

```
create procedure set_energy_by_duration(v_id int, p_duration interval day to second)
as
    real_duration interval day to second;
    m_duration interval day to second;
    new_energy int;
    vehicle_data AVAILABLE_VEHICLES%rowtype;
begin
    select * into vehicle_data
    from AVAILABLE_VEHICLES av
    where av.VEHICLE_ID = v_id;
    if vehicle_data.DURATION <= p_duration then
        real_duration := interval '0 00:00:00' day to second;
    else
        real_duration := vehicle_data.DURATION - p_duration;
    end if;
    select v.MAX_DURATION into m_duration
    from VEHICLES v
    where v.VEHICLE_ID = v_id;
    new_energy := 100 * (extract(day from real_duration) * 24 * 60 * 60 +
                         extract(hour from real_duration) * 60 * 60 +
                         extract(minute from real_duration) * 60 +
                         extract(second from real_duration)) / (extract(day from m_duration) * 24 * 60 * 60 +
                           extract(hour from m_duration) * 60 * 60 +
                           extract(minute from m_duration) * 60 +
                           extract(second from m_duration));
    update VEHICLE
    set ENERGY_LEVEL = new_energy
    where VEHICLE_ID = v_id;
exception
    when NO_DATA_FOUND then
        raise_application_error(-20002, 'vehicle not found');
end;
```

Procedura add_reservation_with_update

Podobnie jak add_reservation dodaje wpis o nowej rezerwacji, jednak ta wersja przeprowadza jednocześnie aktualizację stanu baterii pojazdu pomniejszając o czas trwania rezerwacji.

W rzeczywistych warunkach aktualizacja taka odbywałaby się poprzez pobranie wskazania energii z pojazdu, z racji iż czas wypożyczenia niekoniecznie musi równać się czasowi korzystania z pojazdu. Ponieważ jednak jest to tylko projekt studencki i nie mamy dostępu do rzeczywistych pojazdów, zdecydowaliśmy się na pewne uproszczenie.

```
create procedure add_reservation_with_update(u_id int, v_id int, p_duration interval day to second)
as
begin
    if not USER_EXIST_BY_ID(u_id) then
        raise_application_error(-20001, 'user not found');
    end if;
    if  not VEHICLE_EXIST(v_id) then
        raise_application_error(-20002, 'vehicle not found');
    end if;
    SET_ENERGY_BY_DURATION(v_id, p_duration);
    insert into RESERVATION(user_id, vehicle_id, r_begin, r_end)
    values (u_id, v_id, current_date, current_date + p_duration);
end;
```

Procedura add_vehicle

Dodaje nowy pojazd.

```
create procedure add_vehicle(modelID varchar2, batteryCode varchar2, lat float, lng float, vehicleStatus varchar2, energyLvl int, costPerMinute float) as
b_count pls_integer;
begin
    select count(*) into b_count from BATTERY B where BATTERY_CODE = batteryCode;
    if b_count = 0 then
        raise_application_error(-20004, 'No such a battery');
    end if;

    insert into VEHICLE(MODEL_ID, BATTERY_CODE, LAT_CORDS, LNG_CORDS, STATUS, ENERGY_LEVEL, COST_PER_MINUTE)
    values (modelID, batteryCOde, lat, lng, vehicleStatus, energyLvl, costPerMinute);
end;
```

Procedura toggle_vehicle_status

Zmienia status pojazdu na przeciwny.

```
create PROCEDURE toggle_vehicle_status(
    p_vehicleId IN NUMBER
)
AS
    v_vehicleState VARCHAR2(20);
BEGIN
    SELECT STATUS
    INTO v_vehicleState
    FROM vehicles
    WHERE vehicle_id = p_vehicleId;

    IF v_vehicleState = 'Available' THEN
        UPDATE vehicles
        SET STATUS = 'Not available'
        WHERE vehicle_id = p_vehicleId;
    ELSE
        UPDATE vehicles
        SET STATUS = 'Available'
        WHERE vehicle_id = p_vehicleId;
    END IF;
END;
```

Endpointy

Endpoint register

Połączenie z funkcją add_user – rejestracja użytkownika.

```
const register = async (req: Request, res: Response) => {
  try {
    const { firstName, lastName, phoneNumber, email, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);

    const query = `begin add_user('${firstName}','${lastName}','${email}','${hashedPassword}','${phoneNumber}'); end;`;
    const conn = await oracle.connect();
    conn?.execute(query, [], { autoCommit: true }, (error, result) => {
      if (error) {
        return res.status(500).json({
          message: error.message,
          error,
        });
      } else {
        return res.status(201).json(result);
      }
    });
  } catch (err) {
    res.status(500).json(err);
  }
};
```

Endpoint login

Logowanie do aplikacji.

```
const login = async (req: Request, res: Response) => {
  try {
    const { email, password } = req.body;
    if (!(email && password)) throw new Error('Passed invalid values');
    const query = `SELECT * FROM users WHERE E_MAIL = '${email}'`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [],
      { autoCommit: true },
      async (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        } else if (result) {
          const users = result.rows;
          if (users && users?.length > 0) {
            if (await bcrypt.compare(password, users[0][5].toString())) {
              const user = users[0];
              const userObj = {
                userId: user[0],
                firstName: user[1],
                lastName: user[2],
                email: user[3],
                phoneNumber: user[4],
                password: user[5],
              } as IUser;

              const accessToken = await generateAccessToken(userObj);
              const refreshToken = await generateRefreshToken(userObj);
              refreshTokens.push(refreshToken);
              return res
                .status(201)
                .json({ accessToken, refreshToken, ...userObj });
            } else return res.status(400).json('Wrong e-mail or password');
          }
          return res.status(404).json('Cound not find user with passed email');
        }
      }
    );
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint getAllUsers

Pobiera listę wszystkich użytkowników z bazy.

```
const getAllUsers = async (req: Request, res: Response) => {
  try {
    const query = `SELECT USER_ID, firstname FROM users`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [],
      { autoCommit: true },
      async (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        }
        return res.status(201).json(
          result.rows?.map(item => ({
            userId: item[0],
            firstName: item[1],
          }))
        );
      }
    );
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint getUserStats

Pobiera statystyki użytkownika o danym numerze ID.

```
const getUserStats = async (req: Request, res: Response) => {
  try {
    const { userId } = req.query;
    if (!userId) throw new Error('Invalid argumnet');
    const query = `SELECT * FROM TABLE(get_user_info(${userId}))`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(query, [], {}, async (error, result) => {
      if (error) {
        return res.status(500).json({
          message: error.message,
          error,
        });
      } else {
        const users = result.rows;
        if (users && users?.length == 1) {
          const user = users[0];
          return res.status(201).json({
            noReservations: user[5],
            totalCost: user[6],
          });
        } else return res.status(401).json('Cound not get current user data');
      }
    });
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint getUserReservations

Pobiera z bazy historię rezerwacji danego użytkownika.

```
const getUserReservations = async (req: Request, res: Response) => {
  try {
    const { userId } = req.query;
    if (!userId) throw new Error('Invalid argumnet');
    const query = `SELECT * FROM TABLE(get_user_all_reservations(${userId}))`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [],
      { autoCommit: true },
      async (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        } else {
          return res.status(201).json(
            result.rows
              ?.map(item => ({
                userId: item[0],
                reservationId: item[1],
                vehicleId: item[2],
                r_begin: item[3],
                r_end: item[4],
                cost: item[5],
              }))
              .sort((a, b) =>
                new Date(a.r_begin).getTime() - new Date(b.r_begin).getTime() >
                0
                  ? -1
                  : 1
              )
          );
        }
      }
    );
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint getUserReservations

Pobiera z bazy aktualne rezerwacje danego użytkownika.

```
const getUserCurrentReservations = async (req: Request, res: Response) => {
  try {
    const { userId } = req.query;
    if (!userId) throw new Error('Invalid argumnet');
    const query = `SELECT * FROM TABLE(get_user_current_reservations(${userId}))`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(query, [], {}, async (error, result) => {
      if (error) {
        return res.status(500).json({
          message: error.message,
          error,
        });
      } else {
        const reservations = result.rows;
        if (reservations && reservations?.length > 0) {
          return res.status(201).json(
            result.rows?.map(item => ({
              reservationId: item[0],
              vehicleId: item[1],
              r_begin: item[2],
              r_end: item[3],
            }))
          );
        } else {
          return res.status(202).json([]);
        }
      }
    });
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint addReservation

Tworzy nową rezerwację.

```
const addReservation = async (req: Request, res: Response) => {
  try {
    const { userId, vehicleId, duration } = req.body;
    const query = `begin add_reservation_with_update('${userId}', '${vehicleId}', '${duration}'); end;`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [],
      { autoCommit: true },
      (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        } else {
          return res.status(201).json(result);
        }
      }
    );
  } catch (err) {
    return res.status(500).json(err);
  }
};
```

Endpoint getAllAvailableVehicles

Pobiera z bazy listę wszystkich aktualnie dostępnych pojazdów.

```
const getAllAvailableVehicles = async (req: Request, res: Response) => {
  try {
    const query = `SELECT VEHICLE_ID, MODEL_ID, LAT_CORDS, LNG_CORDS, TO_CHAR(DURATION), ENERGY_LEVEL, COST_PER_MINUTE FROM available_vehicles`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [],
      { autoCommit: true },
      async (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        }
        return res.status(201).json(
          result.rows?.map(item => ({
            vehicleId: item[0],
            modelId: item[1],
            latCords: item[2],
            lngCords: item[3],
            duration: item[4],
            energyLevel: item[5],
            costPerMinute: item[6],
          }))
        );
      }
    );
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint getAllCurrentlyRentedVehicles

Pobiera z bazy listę wszystkich aktualnie zarezerwowanych pojazdów.

```
const getAllCurrentlyRentedVehicles = async (req: Request, res: Response) => {
  try {
    const { userId } = req.query;
    const query = `SELECT VEHICLE_ID, MODEL_ID, LAT_CORDS, LNG_CORDS, TO_CHAR(DURATION), ENERGY_LEVEL, COST_PER_MINUTE FROM TABLE(get_user_currently_rented_vehicles(${userId}))`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(query, [], {}, async (error, result) => {
      if (error) {
        return res.status(501).json({
          message: error.message,
          error,
        });
      }
      return res.status(201).json(
        result.rows?.map(item => ({
          vehicleId: item[0],
          modelId: item[1],
          latCords: item[2],
          lngCords: item[3],
          duration: item[4],
          energyLevel: item[5],
          costPerMinute: item[6],
        }))
      );
    });
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint updateVehiclePosition

Aktualizacja pozycji pojazdu.

```
const updateVehiclePosition = async (req: Request, res: Response) => {
  try {
    const { vehicleId, lat, lng } = req.body;
    if (!(vehicleId && lat && lng)) throw new Error('Invalid argumnets');
    const query = `
    UPDATE VEHICLE
    SET LAT_CORDS = :1, LNG_CORDS = :2
    WHERE VEHICLE_ID = :3`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [lat, lng, vehicleId],
      {
        autoCommit: true,
      },
      async error => {
        if (error) {
          return res.status(501).json({
            message: error.message,
            error,
          });
        }
        return res.status(201).json({});
      }
    );
  } catch (error) {
    if (error instanceof Error)
      return res.status(500).json({
        message: error.message,
        error,
      });
  }
};
```

Endpoint changePassword

Zmiana hasła użytkownika.

```
const changePassword = async (req: Request, res: Response) => {
  try {
    const { userId, newPassword } = req.body;
    const hashedPassword = await bcrypt.hash(newPassword, 10);
    const query = `UPDATE users SET PASSWORD = :1 WHERE USER_ID = :2`;
    const conn = await oracle.connect();
    conn?.execute<(string | number)[]>(
      query,
      [hashedPassword, userId],
      { autoCommit: true },
      (error, result) => {
        if (error) {
          return res.status(500).json({
            message: error.message,
            error,
          });
        } else {
          return res.status(201).json(result);
        }
      }
    );
  } catch (err) {
    return res.status(500).json(err);
  }
};
```

Endpoint addVehicle

Dodanie nowego pojazdu.

```
const addVehicle = async (req: Request, res: Response) => {
  try {
    const {
      modelID,
      batteryCode,
      lat,
      lng,
      vehicleStatus,
      energyLvl,
      costPerMinute,
    } = req.body;
    const query = `begin add_vehicle('${modelID}','${batteryCode}',${lat},${lng},'${vehicleStatus}',${energyLvl},${costPerMinute}); end;`;
    const conn = await oracle.connect();
    conn?.execute(query, [], { autoCommit: true }, (error, result) => {
      if (error) {
        return res.status(500).json({
          message: error.message,
          error,
        });
      } else {
        return res.status(201).json(result);
      }
    });
  } catch (err) {
    res.status(500).json(err);
  }
};
```

Endpoint toggleVehicleStatus

Zmiana statusu pojazdu na przeciwny.

```
const toggleVehicleStatus = async (req: Request, res: Response) => {
  try {
    const { vehicleId } = req.body;
    const query = `begin TOGGLE_VEHICLE_STATUS('${vehicleId}'); end;`;
    const conn = await oracle.connect();
    conn?.execute(query, [], { autoCommit: true }, (error, result) => {
      if (error) {
        return res.status(500).json({
          message: error.message,
          error,
        });
      } else {
        return res.status(201).json(result);
      }
    });
  } catch (err) {
    res.status(500).json(err);
  }
};
```