COMPANY CAR TOKEN HOMEWORK SPECIFICATION

MARCELL ANDRÁS GAJDOS

The smart contracts are designed to **handle and track the use of company cars**, and ensure the valid tracking of vehicles with the consensus provided by miners on the Ethereum blockchain.

Every car has a **unique, non-fungible token** and the contracts store the following:

- Constant parameters:
  - registration number
  - make
  - model ID (given by manufacturer, to distinguish between different variants of the same models. Ford Focus hatchback, sedan and estate for example)
  - date of purchase
  - cost of purchase
  - milage when purchased
  - category within fleet (every employee is assigned to a category by their contract, they can only book cars in this or any lower category, administrator category is 0, top clearance is 1, company can specify how many they need)
- Variables:
  - milage (in kilometers)
  - last pickup time
  - last owner (to deal with damage and lost belongings without needing to search the blockchain)
  - occupied (if the car is occupied, it cannot be picked up, must be returned first)

Employees have the following characteristics:

- name: stored as the key of the mapping
- clearance: the category clearance of the employee
- wallet ID: the walled ID of the employee
- hasCar: if the employee has a car picked up
- exists: if the employee exists

An employee **can only pick up the car if they meet the following criteria**:

- Does not have any unreturned cars on their wallet.
- Has a higher or equal level of clearance category compared to the car they wish to pick up.
- The car is not occupied.

Upon the pickup of the car:

- The milage is set to the current milage
- The last pickup time is set to the current time
- The last owner is set to the employee who just picked up the car (By this they confirm that the car is not damaged and there is nothing left in it. Any problems they can report to the fleet administrator.)
- The car is marked as occupied.
- The token representing the car is sent to the employee's wallet

An employee **can only drop the car off** if they **meet the following criteria**:

- The one who picked up the car and the one who wishes to drop it off is the same

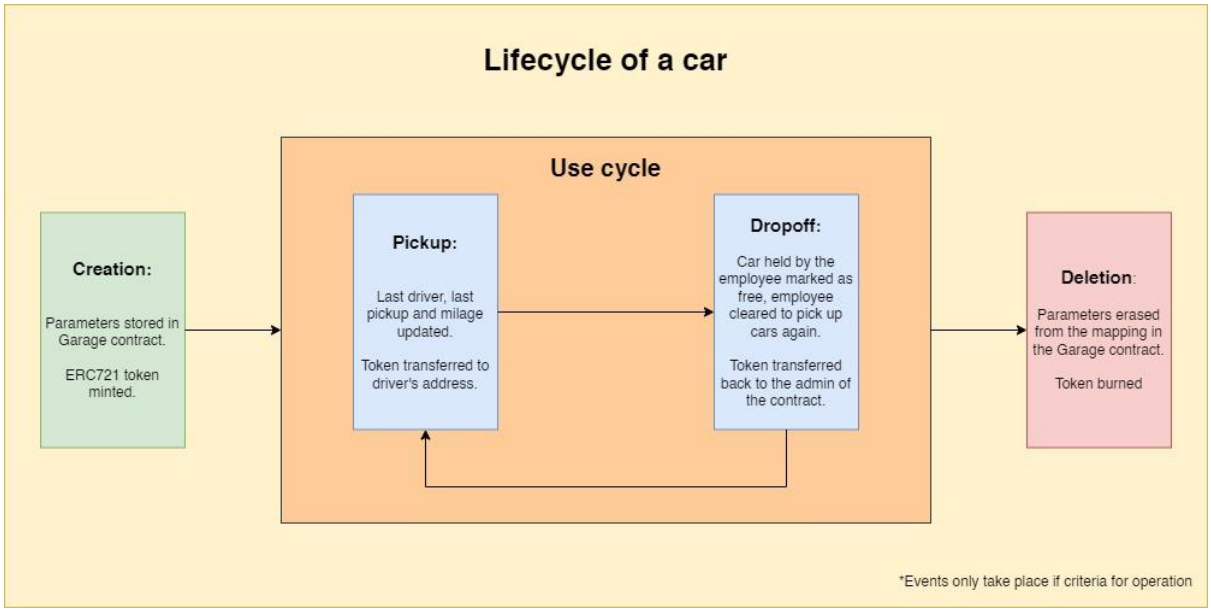
Upon the drop-off of the car:

- The token is transferred from the employee's wallet to the administrator's wallet
- The employee is cleared again to pick cars up
- The car is marked as unoccupied

Additions and removals to and from the fleet can **only be initiated by the fleet administrator**, who can create and destroy the tokens. The administrator's address is "burned" into the contract initially, but the admin can transfer it to another account. In this case, all cars in the garage needs to be transferred to the new admin manually.

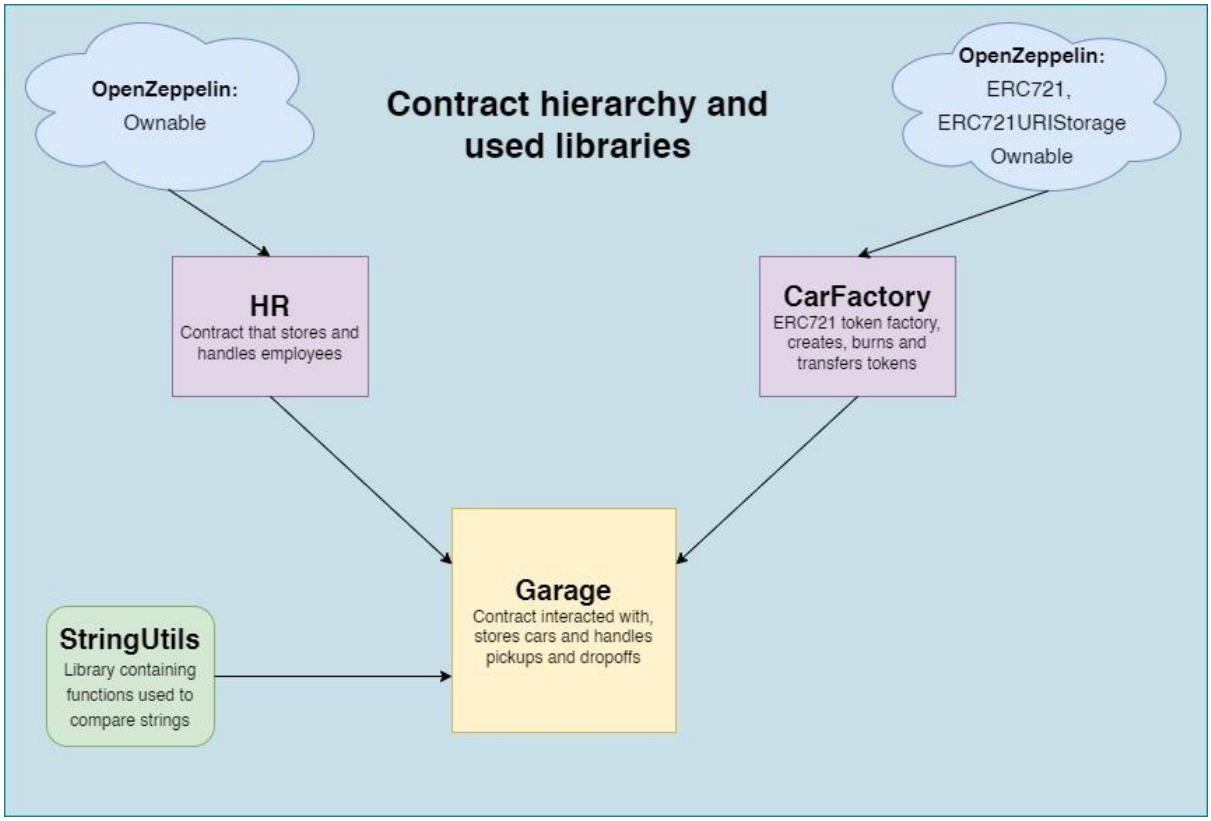
The contracts only **track ownership** of the car and **does not handle the booking.**

Pickups and drop-offs **require the name of the employee**, which is **validated** with some sort of ID, but not implemented in the current code.



Data model:

- *HR* contract:
    - For storing employees and their parameters
    - A mapping, that maps the names to the structs of employee parameters containing the following: clearance, wallet address, hasCar(if the employee has a car picked up), exists(if the employee exists)
    - Add, Modify, Delete functions
- *CarFactory* contract:
    - For ERC721 compliant token handling
    - safeMint, TransferCar, DestroyCar functions to create, transfer and burn tokens
- *Garage* Contract:
    - The main contract, that both stores car parameters and also the main interface to interact with.
    - A mapping, that maps the registration numbers to the structs, containing the parameters of the cars specified earlier and the ERC-721 token IDs.
    - Add, Delete, Pickup and Dropoff functions
- *StringUtils* library
    - Used for comparing strings in the *Garage* contract



Test cases (names speak for themselves):

1. Employee addition, modification and removal tested with the corresponding functions
    a. Add employee (every parameter checked)
    b. Modify employee (every new parameter checked)
    c. Delete employee (check if employee exists after deletion)
2. Car addition, removal, pickup and drop-off also tested with the functions, filtering the failure to meet the restrictions listed earlier
    a. Add car (every parameter checked)
    b. Remove car
    c. Pick car up, should not be able to pick up car when:
        i. clearance improper
        ii. employee has a car picked up
        iii. car already occupied
        iv. employee does not exist
    d. Drop car off

Bootstrapping:

1. Node.js and truffle should be installed
2. Open Node.js command prompt and navigate to the Project folder
3. truffle commands can be run here (develop, compile, test, deploy etc.)