CTP App

CTP Cluj-Napoca is managing its fleet of vehicles using a mobile app. The employees are able to register a new vehicle, query the status, or view reports about them.

On the server-side at least the following details are maintained:

- Id - the internal vehicle identifier. Integer value greater than zero.
- Name - The vehicle name. A string of characters.
- Status - Vehicle current status. A string of characters. Eg. “new”, “working”, “damaged”, “old”, etc.
- Passengers - An integer value representing the number of passengers.
- Driver - The name of the driver. A string of characters.
- Paint - The color of the vehicle. A string of characters.
- Capacity - The trunk volume. An integer number.

The application should provide at least the following features:

- Registration Section (separate activity)
  a. (0.5p) Record a vehicle. Using **POST /vehicle** call by specifying all the vehicle details. Available online and offline.
  b. (1p) View all the vehicles in the system. Using **GET /all** calls, the user will retrieve all the vehicles. If offline, the app will display an offline message and a way to retry the connection and the call. Once retrieved, the vehicle details should **always** be available, no other server calls are needed.
- Manage Section (separate activity)
  a. (0.5p) View all the available vehicle paints in the system. Using **GET /paint** calls, the user will retrieve all the vehicle paint names. Available only online.
  b. (0.5p) View all the available vehicles for the selected color in a list. Using **GET /vehicles** calls, the user will retrieve all the available vehicles of the selected paint. Available only online.
  c. (0.5p) Delete a vehicle, the user will be able to delete the **selected** vehicle. Using **DELETE /vehicle** call, by sending the vehicle id. Available online only.
- Reports Section (separate activity)
  a. (0.5p) View the top 10 vehicles, in a list containing the vehicle name, status, passenger capacity, and driver. Using the same **GET /all** call. The list should present the result in descending order by their passenger’s capacity value. Note that the list received from the server is not ordered.
  b. (1p) View the top 10 drivers, in a list containing the driver name, and the number of vehicles. Using the same **GET /all** call. The list should present the result in descending order by their number of vehicles. Note that the list received from the server is not ordered.
  c. (1p) View the top 5 biggest vehicles by capacity. Using the same **GET /all** call. The list should present the vehicle name and its capacity in descending order by the capacity.
- Driver Section (separate activity)
  a. (0.5p) Record the driver’s name in application settings. Persisted to survive app restarts.

b. (0.5p) View all the vehicles of the persisted driver, in a list that presents the vehicle name, status, and passenger capacity. Using **GET /my** call by specifying the driver.
c. (0.5p) Display all the details of a selected vehicle, from the previous list. The details should be presented on a separate screen/dialog.

(1p) On the server-side, once a new vehicle is added in the system, the server will send, using a WebSocket channel, a message to all the connected clients/applications with the new vehicle object. Each application, that is connected, will display the received vehicle name, size, and driver values, in human form (not JSON text or toString) using an in-app “notification” (like a snackbar or toast or a dialog on the screen).

(0.5p) On all server/DB operations a progress indicator will be displayed.

(0.5p) On all server/DB interactions, if an error message is generated, the app should display the error message using a toast or snackbar. On all interactions (server or DB calls), a log message should be recorded.