Automotive Fleet Management App

An automotive company is implementing a fleet management mobile app for its vehicles. The employees can use the app to register new vehicles, query their status, and view various reports.

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

- Id: Internal identifier for the vehicle. Integer value greater than zero.
- Model: The vehicle model. A string of characters.
- Status: Current status of the vehicle. A string of characters. Eg. “new”, “working”, “damaged”, “private”, etc.
- Capacity: An integer value representing the seating capacity of the vehicle.
- Owner: The name of the owner. A string of characters.
- Manufacturer: The name of the vehicle manufacturer. A string of characters.
- Cargo: The cargo capacity of the vehicle. An integer number.

The application should provide the following features (available without restarting the app):

● Registration Section (Separate Activity/Screen):

A. **(1p)**(0.5p) Record a Vehicle: Record a vehicle using a **POST /vehicle** call by specifying all vehicle details. Available online and offline.

B. **(2p)**(1p) View all Vehicles: View all vehicles in the system using a **GET /all** call. The list should include id, model, owner, and status. In offline mode, an offline message and retry option should be provided. The data should persist on the device after retrieval, regardless of online, offline, or restart conditions. Upon successful retrieval, since the data is available on the device, additional server calls should not be performed.

C. **(1p)**(1p) View Vehicle Details: By selecting a vehicle from the list, the user can view all the details. Using **GET /vehicle** call with the vehicle id, the data should be retrieved from the server each time and made available on the device.

C. (0.5p) Delete a Vehicle: Delete a vehicle using a **DELETE /vehicle** call by sending the vehicle ID. The action should be available in the details screen.

● Report Section (Separate Activity/Screen) - Available Online Only:

A. **(1p)**(0.5p) View Manufacturers: View all available vehicle manufacturers in a list using a **GET /types** call.

B. **(1p)**(0.5p) View Vehicles by Manufacturer: View all available vehicles from the selected manufacturer in a list using the **GET /vehicles** call, by specifying the selected manufacturer.

C. **(1p)**(0.5p) Top 10 Vehicles: View the top 10 vehicles in a list with details such as model, status, capacity, and owner. Use the **GET /all** call and present the result in descending order by capacity.

D. (0.5p) Top 5 Owners: View the top 5 owners in a list with the owner's name and the number of vehicles. Use the same **GET /all** call and present the result in descending order by the number of vehicles.

● Owner Section (Separate Activity/Screen) - Available Online Only:

A. (1p) Record Owner's Name: Record the owner’s name in application settings. Persisted to survive app restarts.

B. (1p) View Owner's Vehicles: View all vehicles of the persisted owner in a list showing vehicle name, status, and capacity. Use **GET /my** call by specifying the owner.

● **(1p)**(1p) On the server side, once a new vehicle is added to the system, the server will send, using a WebSocket channel, a message to all the connected clients/applications with the new object. Each application that is connected will display the received object fields, in a human form

(not JSON text or toString) using an in-app “notification” (e.g., using a snack bar, toast, or an on-screen dialog).
(0.5p)(0.5p) During all server or database operations, a progress indicator will be displayed.
(0.5p)(0.5p) On all server or DB interactions, if an error message is received, the app should display the error message using a toast or snackbar. A log message should be recorded for all interactions (server or DB calls).

NOTE: If your laboratory grade is at least 4.5, only the bold points will be used to compute the exam grade.