Taxi App

A company is managing its fleet of cars using a mobile app. The employees are able to register a new car, query the status or view reports about them.

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

- Id - the internal car identifier. Integer value greater than zero.
- Name - The car name. A string of characters.
- Status - Car current status. A string of characters. Eg. “new”, “working”, “damaged”, “private”, etc.
- Size - An integer value representing the number of passengers.
- Driver - The name of the driver. A string of characters.
- Color - The color name. 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. **(1p)**(0.5p) Record a cab. Using **POST /cab** call by specifying all the car details. Available online and offline.
  b. **(2p)**(1p) View all the cabs in the system. Using **GET /all** calls, the user will retrieve all the cabs. If offline, the app will display an offline message and a way to retry the connection and the call. Once retrieved, the cab details should always be available, no other server calls are needed.
- Manage Section (separate activity)
  a. **(1p)**(0.5p) View all the available cab colors in the system. Using **GET /colors** calls, the user will retrieve all the cab color names. Available only online.
  b. **(1p)**(0.5p) View all the available cabs for the selected color in a list. Using **GET /cabs** calls, the user will retrieve all the available cabs of the selected color. Available only online.
  c. **(1p)**(0.5p) Delete a cab, the user will be able to delete the selected cab. Using **DELETE /cab** call, by sending the cab id. Available online only.
- Reports Section (separate activity)
  a. **(1p)**(0.5p) View the top 10 cabs, in a list containing the cab name, status, size, and driver. Using the same **GET /all** call. The list should present the result in descending order by their size 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 cabs. Using the same **GET /all** call. The list should present the result in descending order by their number of cabs. Note that the list received from the server is not ordered.
  c. (1p) View the top 5 biggest cabs by capacity. Using the same **GET /all** call. The list should present the cab 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 cabs of the persisted driver, in a list that presents the cab name, status, and size. Using **GET /my** call by specifying the driver.

c. (0.5p) Display all the details of a selected cab, from the previous list. The details should be presented on a separate screen/dialog.

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

NOTE: If your laboratory mark is less then 4.5 than non-bold points will be used to compute the exam mark.