Cab App

An big taxi company is using a mobile app to manage their fleet. The employees are able to manage all their cabs and create simple reports.

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

- Id - the internal cab id. Integer value greater than zero.
- Name - the cab name. A string of characters representing the taxi name.
- Model - the cab model. A string of characters representing the cab model. Ie. Audi, Bmw, Skoda, Seat, Toyota.
- Status - the cab status. Eg. “free”, “busy”.
- Seats - the number of seats. Eg. 4.
- Rides - the number of rides that the taxi has made. Eg. 10000, 1000.

The application should provide at least the following features:

- Driver Section (separate activity - available offline too)
  a. **(1p)**(0.5p**)** View all the available cabs. Using **GET /cabs** call, the driver will receive the list of free cabs available in the system. The list will be persisted on the device. If offline the app will display the local persisted list and a way to retry the connection.
  b. **(1p)**(0.5p) Edit cab details. The name, status and the seats should be available to be updated using a **POST /change** call, by using a valid cab id. Available both online and offline. If offline the information will be saved locally and once connected the app will synchronize the local information with the server.
  c. **(1p)**(0.5p) Add more rides. Using **POST /rides** call with the cab id the driver will be able to specify the number of rides that should be added. The server will respond with the cab object with the total number of rides.
  d. **(0.5p)** Remove the local persisted information.
  e. (0.5p) Display the persisted information, collected while offline, in a list format.
  f. (0.5p) When selecting a persisted offline entry, allow the user to update the entry.
  g. (0.5p) Add new entries to the offline persisted list.
- Employee Section (separate activity - available only online)
  a. **(1p)**(0.5p) Using **POST /new** call by specifying a cab name, model and a seats, a new cab will be added.
  b. **(1p)**(0.5p) Remove a cab. Using **DELETE /cab** by specifying the cab id.
  c. **(1p)**(0.5p) List the busy cabs descending by the total number of rides and seats. Using the **GET /busy** call (note that the server is **NOT** maintaining the list of cabs sorted).
  d. (1p) List all the entries received by the websocket, while the app was connected.
  e. (0.5p) Delete all the entries received by the websocket.

(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 add the new cab in the local persisted list of cabs.

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

(0.5p) On all server interactions, if an error message is received, the app should display the error message using a toast or snackbar.

(0.5p) On all interactions (server or db calls), a log message should be recorded.

If your laboratory grade is bigger than 5, only the **bold** requirements should be addressed. Otherwise all the requirement are mandatory!