Planes App

An airport is managing its fleet using a mobile app. The employees are able to register a new plane, query the status or view reports about them.

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

- Id - the internal plane identifier. Integer value greater than zero.
- Name - The plane name. A string of characters.
- Status - Plane current status. A string of characters. Eg. “new”, “working”, “damaged”, “private”, etc.
- Size - An integer value representing the number of passager places.
- Owner - The name of the owner. A string of characters.
- Manufacturer - The name of the manufacturer. A string of characters.
- Capacity - The cargo capacity. An integer number.

The application should provide at least the following features:

- Registration Section (separate activity)
  a. **(1p)**(0.5p) Record a plane. Using **POST /plane** call by specifying all the plane details. Available online and offline.
  b. **(2p)**(1p) View all the planes in the system. Using **GET /all** calls, the user will retrieve all the planes. If offline, the app will display an offline message and a way to retry the connection and the call. Once retrieved, the plane details should always be available, no other server calls are needed.
- Manage Section (separate activity)
  a. **(1p)**(0.5p) View all the available plane manufacturers in the system. Using **GET /types** calls, the user will retrieve all the manufacturer names. Available only online.
  b. **(1p)**(0.5p) View all the available planes in the selected location in a list. Using **GET /planes** calls, the user will retrieve all the available planes of the selected manufacturer. Available only online.
  c. **(1p)**(0.5p) Delete a plane, the user will be able to delete the selected plane. Using **DELETE /plane** call, by sending the plane id. Available online only.
- Reports Section (separate activity)
  a. **(1p)**(0.5p) View the top 10 planes, in a list containing the plane name, status, size, and owner. 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 owners, in a list containing the owner name, and the number of planes. Using the same **GET /all** call. The list should present the result in descending order by their number of planes. Note that the list received from the server is not ordered.
  c. (1p) View the top 5 biggest planes by capacity. Using the same **GET /all** call. The list should present the plane name and its capacity in descending order by the capacity.
- Owner Section (separate activity)
  a. (0.5p) Record the owner’s name in application settings. Persisted to survive app restarts.
  b. (0.5p) View all the planes of the persisted owner, in a list that presents the plane name, status, and size. Using **GET /my** call by specifying the owner.

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

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