

Boats App

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

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

- Id - the internal boat id. Integer value greater than zero.
- Name - the boat name. A string of characters representing the boat name.
- Model - the boat model. A string of characters representing the boat model. Ie. Serenity, Orion, Whisper.
- Status - the boat status. Eg. "free", "busy".
- Seats - the number of seats. Eg. 4.
- Rides - the number of rides that the boat has made. Eg. 10000, 1000.

The application should provide at least the following features:

- Owner Section (separate activity - available offline too)
 - a. **(1p)(0.5p)** View all the available boats. Using **GET /boats** call, the owner will receive the list of free boats 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 boat details. The name, status and the seats should be available to be updated using a **POST /change** call, by using a valid boat 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 boat id the owner will be able to specify the number of rides that should be added. The server will respond with the boat 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 boat name, model and a seats, a new boat will be added.
 - b. **(1p)(0.5p)** Remove a boat. Using **DELETE /boat** by specifying the boat id.
 - c. **(1p)(0.5p)** List the busyboats descending by the total number of rides and seats. Using the **GET /busy** call (note that the server is **NOT** maintaining the list of boats 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 boat is added in the system, the server will send, using a websocket channel, a message, to all the connected clients/applications, with the new boat object. Each application, that is connected, will add the new boat in the local persisted list of boats.

(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!