Educational App

A school is managing its students using a mobile application. The teachers and their students are able to manage and view their quizzes and marks.

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

- Id - the internal student id. An integer value greater than zero.
- Name - the student name. A string of characters representing the student name.
- Group - the student group. A string of characters representing the group. Eg. 911,912,931,932, etc.
- Year - the enrollment year. An integer value greater than zero.
- Status - student status. A string of characters. Eg. free, paid.
- Credits - the student accumulated credits. An integer value greater or equal to zero.

The application should provide at least the following features, in separate activities:

- Teacher Section
  a. (1p) View the students available in the system in the form of a list, ordered by group and name. Using **GET /students** call, the user will retrieve the list of all the users found in the system. Note that from the server the list is retrieved unsorted. If offline, the app will display an offline message and a way to retry the connection and the call. Once retrieved it should be **available offline**. If the list is already available offline it should always be used, no new server calls of this type are needed anymore. The user should be able to trigger a manual refresh if needed.
  b. (0.5p) View all the details related to the selected student. By selecting an student from the previous list, the user will be able to see all the details in a separate screen. In order to get the student details **GET /student** call should be used. Available only if online.
  c. (0.5p) In the details screen from above, using **POST /student** call by specifying the student object with a valid student id, the user will be able to update the student details. Available online only. The operation should be reflected on the main list too.
  d. (0.5p) In the details screen from above, using **DELETE /student** call by specifying the student id, the user will be able to delete the selected student. Available online only. The operation should be reflected on the main list too.
  e. (0.5p) While on the main list screen the user should have the option to add a new student. Using **POST /create** call by specifying the student object, the user will be able to add a new student into the system. The operation should be **available offline** too. Once online the app should detect and resume the operation. The content should survive even if the application is terminated.
  f. (0.5p) While on the main list screen the user should have the option to update also the student credit. Using **POST /updateCredit** call by specifying the student id and the new credit, the user will be able to update only the student credit. The operation should be available online only.
- Student Section
  a. (0.5p) Give the user the option to specify and store the student id. The student id should be persisted in the local database and available for further operations.

b. (0.5p) View all the available lectures. Using **GET /lectures** call, the user will be able to see the list of all the available lectures found in the system. Available **both online and offline**. Once retrieved it should be used from the local storage only.
c. (0.5p) Once an lecture is selected the user should be able to register to the lecture. Using **POST /register** call by specifying the lecture id and the student id, the one that was stored in the local database.

- Statistics Section
  a. (0.5p) Give the user the option to specify and store the lecture name. The name should be persisted in the local database and available for further operations.
  b. (0.5p) View all the students associated with the persisted name. Using the **GET /byLecture** call, the application should retrieve and display the list of students associated with the specified lecture. The operation should be available only online. The list should display the student name and their credits in ascending order by the student name.
  c. (0.5p) View the top ten lectures by registered students. Using the same **GET /students** call, the application should retrieve and display the list of lectures with most of the registered students. The operation should be available only online. The list should display the lecture name and the number of registered students in descending order by the number of students.
  d. (0.5p) View the top ten students by credit. Using the same **GET /students** call, the application should retrieve and display the top ten students by their credits. The operation should be available only online. The list should display the student name, group, year and credits in descending order by credits and group.

(1p) On the server side, once a new student is added in the system, the server will send, using a web socket channel, a message to all the connected clients/applications with the new student object. Each application, that is connected, will add the new student in their main list, if visible, or add the details in the local database to be used later. A notification message should be displayed too regardless of the open activity.

(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. On all interactions (server or DB calls), a debug log message should be recorded.