Tuition App

A university is managing its tuitions using a mobile application. The secretaries, students and their parents are able to manage and view the taxes related to them.

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

- Id - the internal tax id. An integer value greater than zero.
- Name - the student name. A string of characters representing the student name.
- Description - the tuition description. A string of characters.
- Type - the tuition type. A string of characters representing the type. Eg. per year, per quarter, per month, etc.
- Due - the due date. An integer value greater than zero.
- Status - tuition status. A string of characters. Eg. open, closed, late.
- Amount - the tuition amount. An integer value.

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

- Secretary Section
  a. **(1p)** View the tuitions available in the system in a form of a list, ordered by student name and status. Using **GET /tuitions** call, the user will retrieve the list of all the tuitions 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. **(1p)**(0.5p) View all the details related to selected tuition. By selecting a tuition from the previous list, the user will be able to see all the details in a separate screen. In order to get the tuition details **GET /tuition** call should be used. Available only if online.
  c. **(1p)**(0.5p) In the details screen from above, using **POST /tuition** call by specifying the tuition object with a valid tuition id, the user will be able to update the tuition details. Available online only. The operation should be reflected on the main list too.
  d. **(1p)**(0.5p) In the details screen from above, using **DELETE /tuition** call by specifying the tuition id, the user will be able to delete the selected tuition. 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 new tuitions. Using **POST /add** call by specifying the tuition object, the user will be able to add new tuition into the system. The operation should be available offline too. Once online the app should detect and resume the operation.
  f. (0.5p) While on the main list screen the user should have the option to update only the tuition status. Using **POST /updateStatus** call by specifying the tuition id and the new status, the user will be able to update only the tuition status. The operation should be available online only.
- Parent Section
  a. **(0.5p)** Give the user the option to specify and store the student name. The name should be persisted in the local database and available for further operations.
  b. **(1p)**(0.5p) View all the open tuitions for the current student name. Using **GET /myTuitions** call, the user will be able to see the list of all the available tuitions found

in the system for the specified name. Available both online and offline. Once retrieved it should be used from the local storage.

c. (0.5p) Once an open tuition is selected the user should be able to pay the tuition amount. Using **POST /pay** call by specifying the tuition id and the amount. Note that partial amount should be supported.

- Student Section
  a. **(0.5p)** Give the user the option to specify and store the student name. The name should be persisted in the local database and available for further operations.
  b. **(1p)**(0.5p) View all the open tuitions in the current year associated with his name. Using the same **GET /tuitions** call, the application should retrieve and display the filtered list by the year. The operation should be available only online. The list should display the tuition description, amount and year in ascending order by year.
  c. (0.5p) View all the close tuitions in the current year ordered by amount. Using the same **GET /tuitions** call, the application should retrieve and display the filtered list by the year for the current student. The operation should be available only online. The list should display the tuition description, year and amount in descending order by amount.
  d. (0.5p) View the top ten tuitions by amount. Using the same **GET /tuitions** call, the application should retrieve and display the top ten tuitions by rating. The operation should be available only online. The list should display the tuition description, year and amount in descending order by amount and year.

(1p) On the server side, once new open tuitions are added in the system, the server will send, using a web socket channel, a message to all the connected clients/applications with the new tuition object. Each application, that is connected, will add the new tuition 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.

(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 log message should be recorded.

NOTE: If your laboratory grade was above 4.5 you need to solve only the requirements that have the points in bold. If your laboratory mark is less then 4.5 than in order to compute the exam mark we are using the regular points (non-bold ones).