This is structured as the user will experience it in the webpage:

## When LOGIN – MENU-Dashboard

When you enter your credentials, *http://localhost:3000/* **Stats.js** component opens (*/client-jb/src/pages/dashboard/Stats.js*).

**Stats.js** collects all the info needed to plot and sends it to the child components, which are in charge of displaying it. **Stats.js**. is formed of 4 *useEffects ():*

- 1 collects user info from local storage.
- 1 collects score data from local storage.
- When user and score data are collected, it calls *getAllRecData()* from database to get list of all recordings of this user.
- When recordings are loaded, it calculates some needed info for the child components, explained below.

The child components of stats and the info they need are the following:

- **StatsGeneral.js** () : receives and displays number of total recordings, unread messages and pending tasks to complete
- **StatsPercentageStars.js** : displays performance by showing user the % of the level they have completed. It receives as input the total stars each level has (among all available scores *3 stars per score) and the achieved number of stars by the user (sum of the stars achieved by the user in each score, if one score has several recordings, it takes the best mark, per level)
- **StatsAreaChart.js**: chart displaying the activity of the user in the last week. It receives 2 arrays containing the dates and the levels, respectively, of the user recordings. Inside this component, for each level (seeing the array of level data) we count how many repetitions there are of each day (in the dates array) to know how many recordings the user made each day.
- **StatsRecentRecordings.js** : displays a table (scrolling table) where you see all the recordings order from most recent to oldest, allowing the user to go see it(click on eye icon, takes to you **ProgressPlayFileVisual.js**) or delete it(trash icon, calls *deleteRecording()*). It receives as input the list of all recoding data and a callback function (so that when the user deletes a recording, we tell Stats.js to "reload").
- **StatsTasksSection.js** : displays the most recent assignment the teacher posted. It has an "open" button that directly takes you to the "assignments" page. It does not receive any inputs. Everything is calculated inside:
  - User info is get from local storage
  - Once user info is available, we know who their teacher is, so we get from DB teacher info with *getProfileData(teacherId)* call.
  - Once teacher and student info are loaded, we call *getLatestAssignment(userData.id)* to load latest assignment to display.

**NOTE. All of the above components have their respective css file named:*
*Same-name.module.css*

## MENU – ALL LESSONS

When user clicks on AllLessons the component **AllLessons.js** is opened. This component basically gets the scoreData from local storage and displays it.

Additionally, there are listeners *handleNodeMouseOver()* and *handleNodeMouseOut()* that when mouse is over one score, it puts *selectedNodeActive*, provoking the component OpenSheetMusicDisplayPreview.js to appear, showing a preview of the score. This component receives as input the *path to the xml file* of the score.

Also, in each score item there is a <*Link*> object (imported from *react-router-dom* library) so that when you click it takes you to *http://localhost:3000/all-lessons/xml-file-name* that opens **ProgressPlayFile.js**.

**ProgressPlayFile.js** is one of the most complicated components. It is the one in charge of the recording, playing, *osmd*, audio, control bar with buttons… The best way to start to explain how it works is to go down in the page and see the *return()*, this is what is being returned:

- **OpenSheetMusicDisplay.js**: *osmd* object. Receives as input bpm, zoom, play… information.
- **MetronomeCountdown.js**: countdown object that only shows up when variable *showTimer=true* (which is activated when play/record buttons are clicked. It receives as input the bpm (to determine the speed of 4,3,2,1) and a callback *"onComplete"* that tells **ProgressPlayFile.js** when the countdown is finished to proceed with other tasks( showTimer=false, start playing/recording…)
- **ControlBar.js** (for *practice mode*) and **ControlBarRecord.js***** (for *recording mode*). Depending on which mode is active, one or the other will appear. Object in charge of buttons. It receives as input:
  - information about the cursor (because *osmd* detects cursor finishing→tells **ProgressPlayFile.js** about it→ informs **ControlBar** in order for it to reset its buttons to initial state)
  - callback function so that the child component, **ControlBar,** can tell **ProgressPlayFile.js**, parent, when the user has clicked a button, so that it can act over its other child, *OSMD*, telling it to stop playing or whatever other action…
- **PopupWindow.js**: window in charge of asking user if they want to save or delete the recording they just did (therefore it will only come up in recording mode, when the cursor reaches the end, meaning that recording is fully done, or when the user stops recording clicking on button, meaning that they want to stop the recording). It only shows up then *showPopUpWindow=true*. It has a callback so that this child component, popupwindow.js, can tell its parent **ProgressPlayFile.js** what has been the answer of the user.
- **ModeToggle.js**: is the switch icon that we see in the upper-right part of the screen that allows you to switch between one mode or the other

*The other code inside **ProgressPlayFile.js** just contributes to the correct functionality of it and the inter-communication between this component and its childs (a parent component communicates with childs by providing inputs, and the only way a child can communicate with a parent is by callback functions)*

*** In **ControlBarRecord.js** we can see a flag button that takes you to see your recordings associated with the current score you where in. This opens component ListRecordings.js.
This component receives throuth the URL which XML file we want (the score we were in) and retrieves from local storage the user data. With this, it calls *getRecData(userData.id, scoreID)* to get the list of recordings and display them (from oldest to newest).

**MENU – MY RECORDINGS**

This page loads **ListAllRecordings.js** component, which is very simple and very similar to **ListRecordings.js**. The only difference is that instead of retrieving the recordings associated to a *userId* and to a *scoreId*, we retrieve all the recordings from a *userId* calling *getAllRecData(userId),* and display them in the same way as **ListRecordings.js**.

## MENU – ASSIGNMENTS

When user clicks on Assignments page it opens **assignments.js** component. This component is in charge of showing the assignments submitted by the teacher and controlling the chat.
Again it is a bit complex, the easiest way is to focus on the *return()* at the end of the script.

```
return (
  <div className={AssignmentsCSS.container}>

      <div className={AssignmentsCSS.left} >⋯
      </div>
      <div className={AssignmentsCSS.right}>⋯
      </div>
      {popUpWindowGrade?<PopUpWindowGrades handlerBack={handleSeeGrades} comment={taskComment} grade={taskGrad
      {popUpWindowRecordings?<PopUpWindowRecordings handlerBack={handleSelectRecording} scoreId={selectedScore

  </div>
);
```

As you can see we have "left" and "right".

**Left side:**
It is the assignments' part. Everything related to this part is processed in the **assignments.js** script. We find 3 *useEffect()*:

- One to retrieve user info from local storage, followed by retrieving teacher data.
- One to retrieve scores data from local storage.
- Once we have the user and scores data, we retrieve the assignments associated to this student with *getAllAssignments(userId)*. Then they are displayed.
    - Those assignments that have no tasks associated they show a message "no tasks associated", those which DO have a task associated, they show the tasks.
    - Those tasks that have not been answered by the user, present two buttons. "Record vinyl icon" to take you to **ProgressPlayFile.js**, to record the corresponding task/score, or "Archive" icon to display you with **PopUpWindowAssignment.js** a list with your available recordings of that score, for you to select one.
    - Those tasks that have been answered present two other buttons. "eye" icon to take you to **ProgressPlayFileVisual.js** to see your recording. Or a "book" icon to see your grade calling **PopUpWindowGrade.js.**

**Right side:**
On the right side, **assignment.js** calls **messages.js**, to show the chat. It provides as inputs the user and teacher data. When **messages.js** is called, meaning, when chat shows up, as the user and teacher data have already been provided, the component carries out 2 tasks:

- Get chat messages: *getMessages(userId,teacherId)* and display them
- Tell DB that user has read all messages received by teacher, meaning, when user reads chat, we need to tell DB that *seen=true* for all messages sent by the teacher, with call *updateMessageSeen(userId, teacherId).*

There are two buttons in the chat:

- One for sending messages: putMessage() to DB
- One for reloading messages to check for new messages: *getMessages(userId,teacherId)*