## Important!!

This document contains a few guidelines for the skynote code. It's a work in progress, so if you notice something outdated or missing, feel free to make any corrections or additions. The original file I used is HERE :)

## 1. Main .js files

Here is a list of the main .js files, in the order the user would see them when using skynote for the first time.

### Landing.js

When the user arrives at the site for the first time, they see “Landing.js”. In here they can navigate to “Demos.js”, “OurTeam.js”, “Research.js”, Interest Form(redirects to a Google Form), and “Register.js”.

### Register.js

When the user clicks the submit button, “Register.js” makes use of appContext.js

```
const { user, isLoading, showAlert, displayAlert, setupUser } =
useAppContext();
```

In here, “appContext.js” is used to log in the user, and store some relevant information locally. This is done through “addUserToLocalStorage” and “getAllScoreData2”. I believe that the original intention was to use JWT tokens to store all this information, but this is currently not implemented.

Once the user is logged in, “Stats.js” is loaded. (This file is explained in the document that Amaia wrote before leaving, and should be merged into this one. It contains info about the Dashboard, All lessons, My recordings and Assignments menus)

Other than that, there are 3 extra menus:

- Profile: “Profile.js” is a simple file that allows to change some of the user data from the database. It has one useEffect() to get the user data from the database, and another one to register every change made in the form.
- Sound visualization: “TimbreVisualization.js” is called here. The user can use their mic to analyse some aspects of their playing like the pitch or the dynamic stability.
- API testing: “apitesting.js” contains a few functions for trying some of the database calls.
    - toggletimer: allows to activate a timer. The idea behind this is to try and record the amount of time a user is actually using skynote.
    - Create task: this one allows for the creation of tasks, in order to help with testing. It does that by creating a popup window (PopUpWindowAssignments.js) At the moment the list of students and the teacher Id are fixed

        - The teacher part if the site is not done yet, so we set a fixed teacherId just for testing purposes.
        - The list of students should be retrieved from the database, but we haven't decided on how we should store the relationship between student and teacher. We could assign a teacherId to every student and then search all the students that match a specific teacherId, or we could add a "contacts" field to every user, and that way we could ask for the info of one specific database object, rather than going over all of them. This also allows for future features like adding friends, but we are currently not focused on that yet.
    - Grade task: this hasn't been coded yet, so right now it only shows an empty popup window ("PopUpWindowGrading.js"). The focus is not on this at the moment, so it's probably not worth investing time into this, and code it once the teacher side is created.
    - postMessage: This one posts a prewritten message in the chat from the current user to our default teacherId (in this case '5d34c59c098c00453a233bf3')
    - getMessages: This one gets the latest messages in descending date order. The limit of messages can be chosen, but it's currently fixed to 12.

## 2. DB calls

In order to create new database calls, the file "flow" is file, then methods, then routes, then controller and finally model. An example of this would be:

- The file "ListAllRecordings.js" uses a function called "getAllRecData" to retrieve all of the recordings from a given user (studentId).
- The "getAllRecData" function is in "studentRecordingMethods.js". It is an async function that makes an axios.get request to the URL endpoint "/api/v1/recordings/getAllRecData".
- The "recordings/getAllRecData" route is made in "recordingRoutes.js". When a GET request is made to this route, it is handled by the "getAllRecData" function, which is imported from "recordingController.js"
- In "recordingController.js", the function "getAllRecData" takes the mongoose schema from "StudentRecordings.js" (in the models folder), as "student_recordings" and makes all the necessary database calls.

There are models for every DB collection, and the same goes for controllers and routes:

| Route | Controller | Model | Collection |
|---|---|---|---|
| assignmentRoutes.js | assignmentController.js | Assignments.js | assignments |

| | | | |
|---|---|---|---|
| messageRoutes.js | messageController.js | Message.js | messages |
| scoreRoutes.js | scoreController.js | xmlScoreModel.js | scores |
| recordingRoutes.js | recordingController.js | StudentRecordings.js | student_recordings |
| authRoutes.js | authController.js | User.js | users |

## 3. Styling

Styling is done though the CSS module files with the same name as the javascript files. For example, the styling for ‘MainMenu.js’ is done in ‘MainMenu.module.css’. We changed to this way of styling not so long ago, so there is a chance that some of the older files don’t have a ‘.module.css’ file. If that is the case, a new ‘.module.css’ file should be eventually created.
Other than that, some other rules are being applied that come from the “normalize.css” package, from Bootstrap and from the “index.css” file.
I’ve found a few inconsistencies in the form of redundant rules, or rules that are applied at the wrong time (for example, rules from ListRecordings.module.css applied to the assignments page). A few things to keep in mind:

- The .module.css files should not make changes to tags like <body>, as this cause the css to affect the whole page rather than specific areas. This should be done in index.css. The .module.css files should only make changes by className.

In order to keep consistency, this is a set of colours we’ve been using, if the colour you’re looking for is not here, feel free to add it to the list, but make sure you’re not assigning a new colour to something that already exists (there is also a list of colours in “index.css”, but I believe some of the newer .module.css files don’t necessarily take that list into account):

- Back button : #9E9E9E
- Backbutton hover: #717171
- Borders: #CADDFB
- Box-shadow: #CADDFB
- Containers (aka background colour): #FFFFFF
- Fonts: #000000
- Icons: #939393
- Icon button hover: #D2E8E1(this one needs to be reviewed)
- Star complete: #E8D73F
- Star incomplete: #909090