LECHEVALLIER Maxime
BUI David
OUNISSI Sara

RealTimeSubtitles report

Situation :

The department of the disabled students, at the UGA, is aiming to help disabled students to follow a course autonomously. To do so, they participate regularly to various projects as consultant, to ensure the accessibility of workspaces and of education facilities or to improve the ergonomics of educational interfaces used by people with all types of disabilities. In this context, we have been ask to develop an application with GoogleAPI speech to transcript teacher speech to subtitles.

Topic :

The aim of our project is to make easier the understanding of a course by partially deaf students. In order to achieve this, using GoogleAPI Speech, we had to develop an app that transcrip in live a teacher speech on his slides as subtitles. However, the subtitles are not always accurate, due to the API. To correct this problem, we had to set a collaborative HCI that allows students to correct misinterpreted words. We save the subtitles in a final file to keep track of the course linked to the slides.

Goals :

The main target of the application is to find the good way to implement the collaborative HCI for the correction of the transcript, by setting a scoring system. The word that have the highest score is more likely to be true. In addition, we also have to manage the two different interfaces : the teacher’s one and the student’s one. To avoid an abusive use of

the application by disruptive students, we have to restrict permissions for unlogged people.

Technologies used :

- Google API Speech to transcript in live teacher's comments. It is a free API Speech designed by Google. There are some good tutorials online for new users. However, the API isn't designed for long talk, but more for short queries. After different tests in various conditions we noticed that the API isn't reliable and crashes easily.

- Reveal.js to program the teacher presentation, it allows teachers to do elegant and simple slides in HTML. We can get events easily using javascript. We can add reveal sections in an HTML page and follow the course while correcting the subtitles on the right side.

- HTML5, CSS3 and JavaScript to program the basis of the app. These are three programming language that we must use for web app. The structure of our app needs HTML, the dynamic part is implemented due to JavaScript and the layout with CSS.

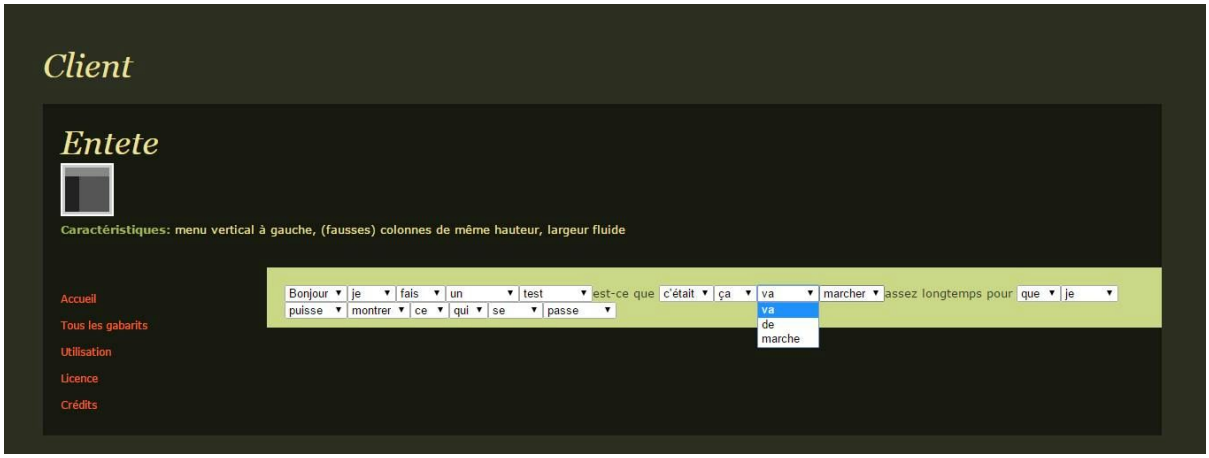
- Socket.io used at first place to communicate between the API and the corrective collaborative HCI used by students. For example socket.io allows users to concurrently edit a document and see each other's changes.

- Meteor, a framework allowing us to have dynamic pages and to communicate between the different interfaces much more easier than using socket.io. Many packages exist for meteor, we found them on https://atmospherejs.com/ , as accounts-ui for login form ...

- Bootstrap, a package included in meteor, very useful for layout. We are not expert in CSS, so Bootstrap help us a lot to have a nice interface. Navbar, buttons, caroussel...

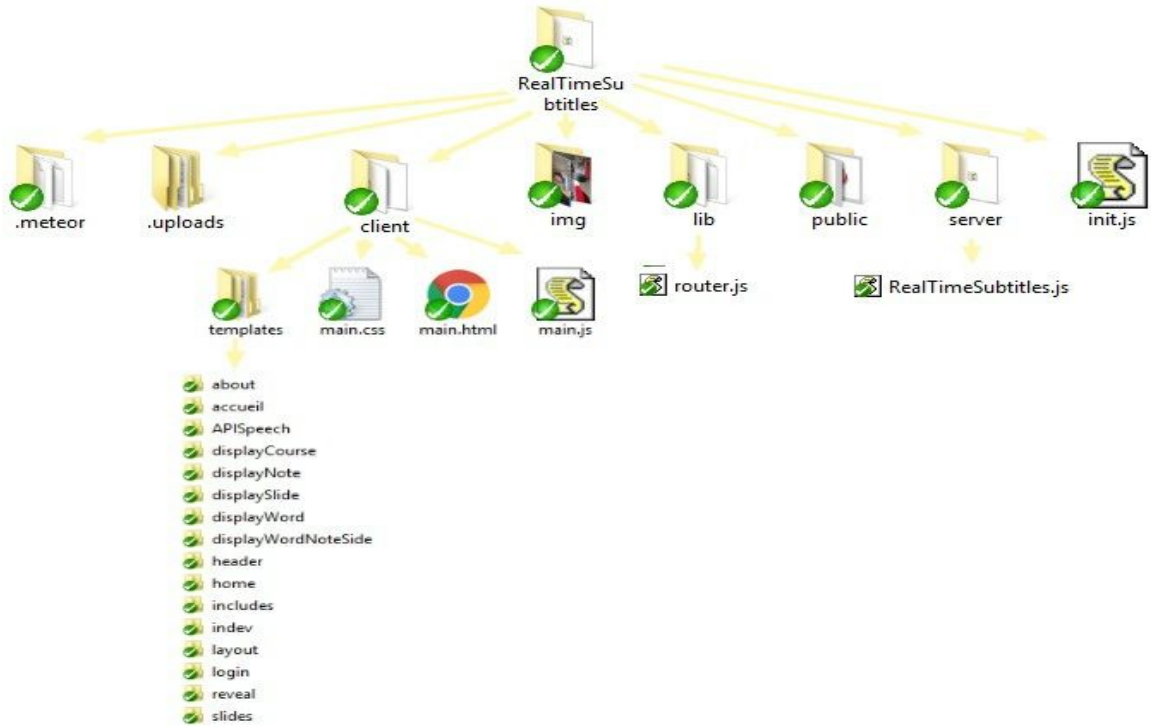
Work done :

From the start to half-time project we used Jquery, Socket.io and Reveal.js for the whole project. We had the speech recognized by Google Speech parsed into the app and could send it to a single user. When we had to implement MVC pattern and make a distributed app, it became suddenly less easy. The following picture is what we had at the half-time project, the interface is minimalist but we could at that moment get the results of Google Speech.



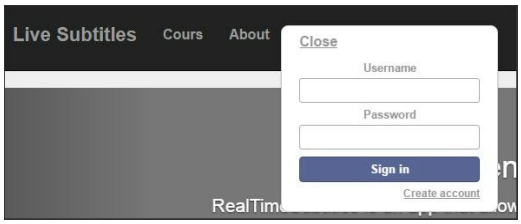
With our basics knowledge of database learnt at Polytech, we couldn't figure out how to make a database and manipulate the data. From there we have decided to start from scratch and use Meteor.

The next picture is showing how we manage to separate the work and make it faster to develop. Meteor allows a lot of things because everything is dynamic.



Everything is dynamic with meteor that is why we had to adapt our habits and learn about how to produce dynamic code. No more static HTML, no more long CSS, everything is handle by meteor (which is convenient at first). Then is it easy to add packages that we need for the project : accounts, reveal, css framework ...etc

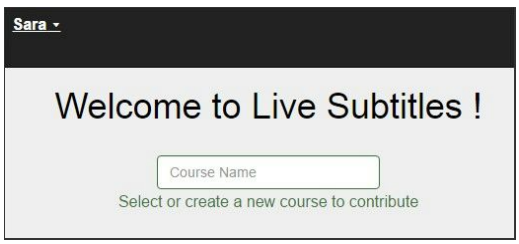
From that point, we tried to add as many features as possible in a short time : handle session login, teacher/student version, edit subtitles, collaboration on correcting subtitles. Here's what we have achieved :



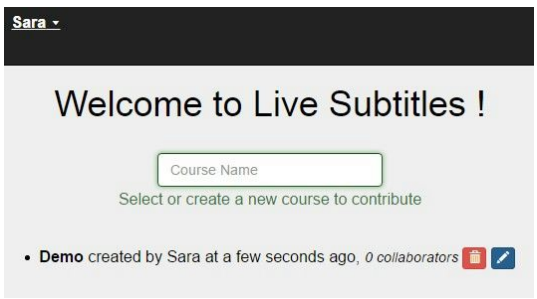
login form using package : accounts-ui and accounts-password. It is really easy to set, and accounts-ui give us all the function needed to catch the name of the user, or if he is connected or not…

{{#if currentUser}}

We set some conditions to our app, features like adding a course, correcting notes and so one are allowed only for logged users. If the user isn't logged he can only see the presentation of the app, and the section about us.

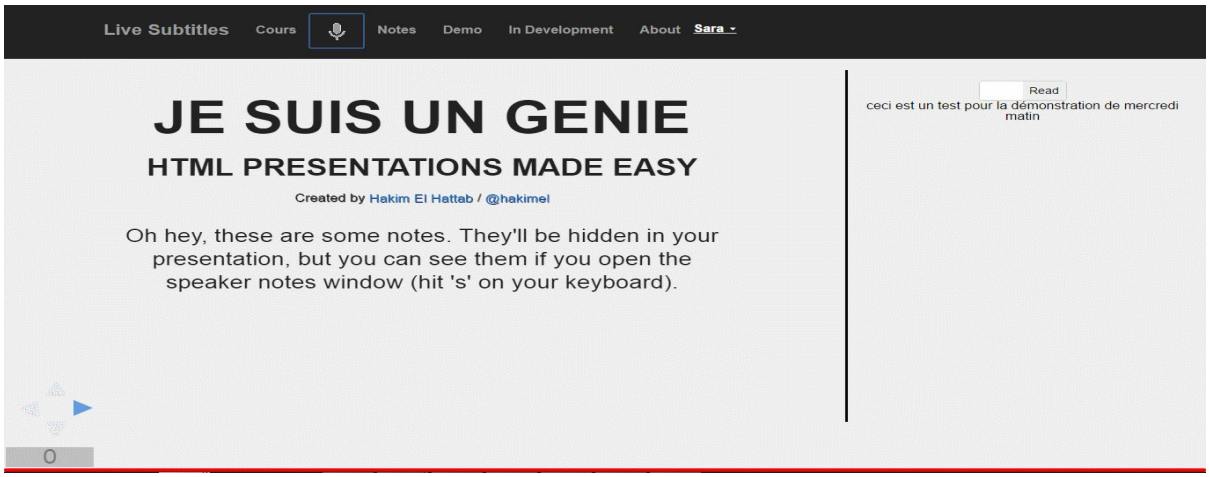


The main goal was to upload the teacher file corresponding to the presentation he wants to do. But first we just set a collection with names of courses. The user can add a course (name) and the courses created by him or by the other students are displayed as a list.

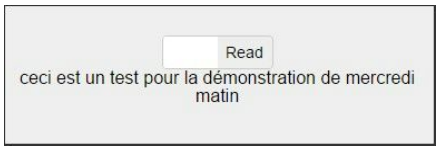


The user can join a course and become a collaborator (blue pencil)

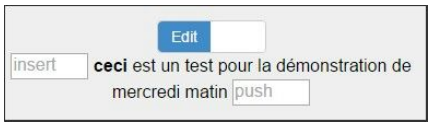
To removing a course (red bin), the user has to be the one who created it (authorization required), others can just join the course but not removing it.



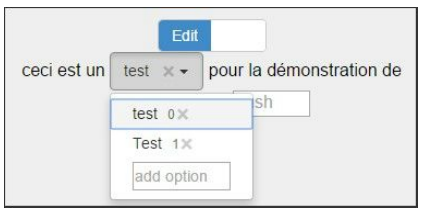
This is how we displayed the presentation reveal.js, with the subtitles on the right side. The user that have created the course can launch GoogleSpeech and start talking. The API transcript the speech and display the results next to the current slide. The other users have to join the course follow it and to collaborate.



Users can either watch slides without editing, in the “Read” mode.



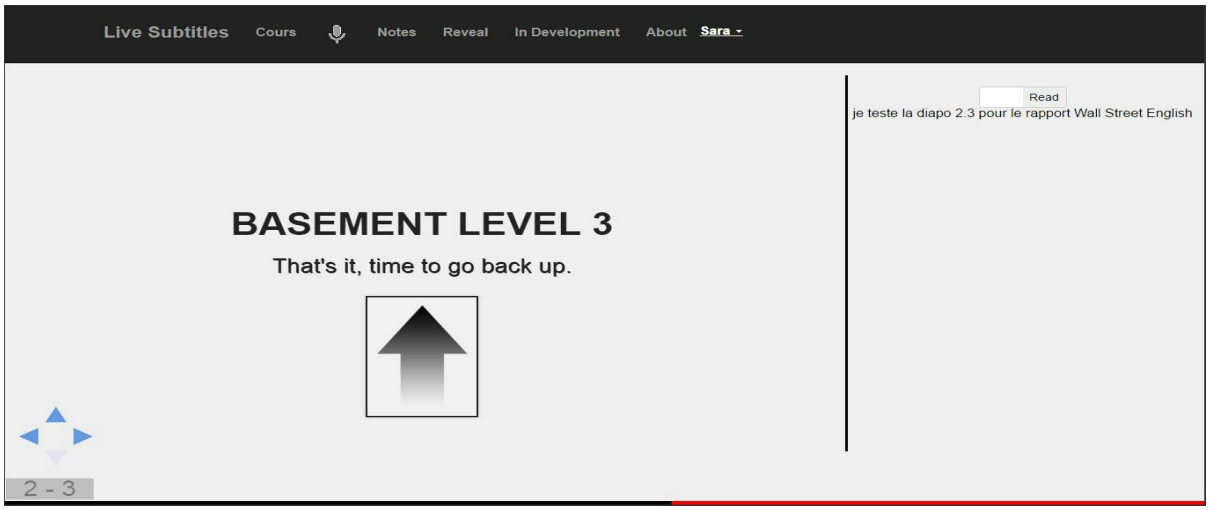
Or they can turn on the editing mode and start correcting the subtitles. They can insert a word at any place. If a word is false, they can click on it and add an option to it, then chose this option.



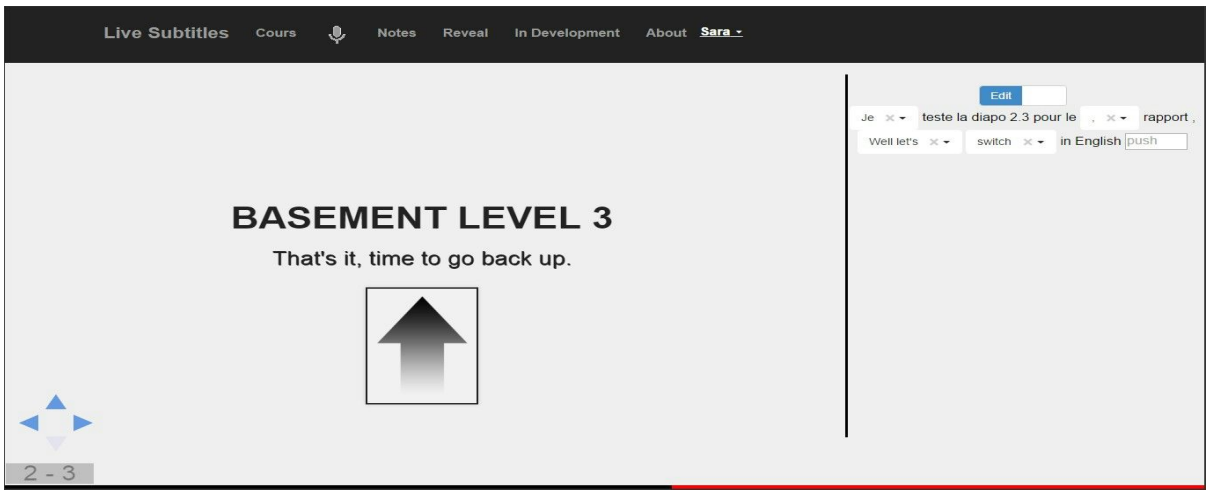
We can have different correction of the same word, depending on users. To deal with that, we set up a scored correction and display the more accurate result. The more a word is selected in an option form for example, to more likely it is to be true. So when we go back to the mode “Read” after editing we see the sentence corrected with the highest probability.

Demo :

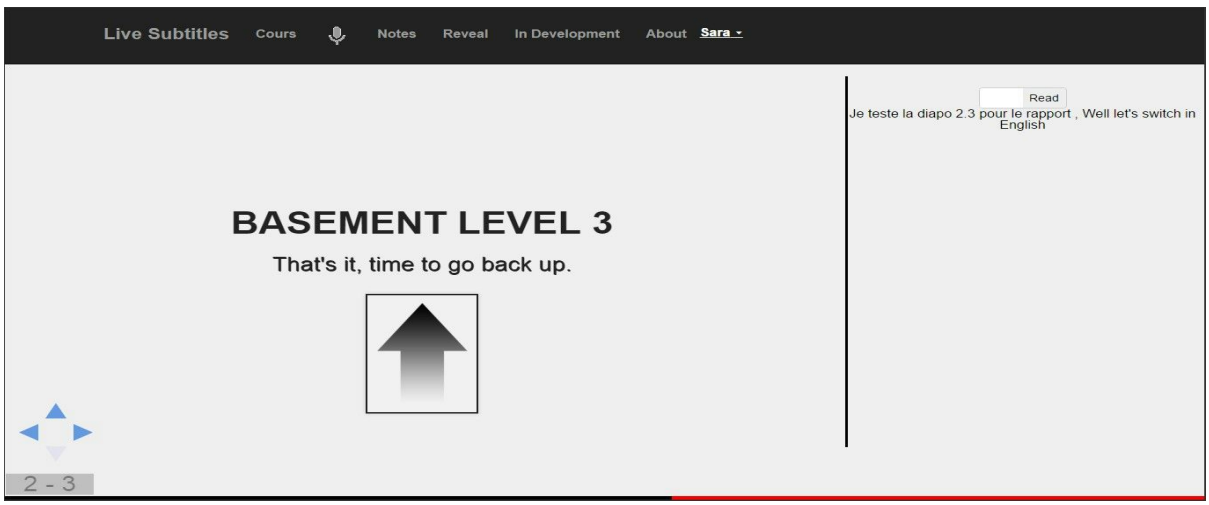
Here is an example of presentation. The “Teacher” talked and the sentence has been written on the right side. The mode “Read” is activated so we just see the sentence.



Then we switch to “Edit” mode, and we correct the sentence by adding words and correcting others.



Finally when we get back to the “Read” mode, we can read the correct sentence.



Problems faced :

Reading documentation take us a long time. We had to understand all the technologies we were using because we didn't have any experience in any of them.

First problem we had to deal with was the instability of GoogleSpeech. Often the transcript was really slow, or stopped completely after just displaying few words. The API take always into account the context of the sentence so it can correct the transcript while going on the vocal recognition. When the contexte become to long the API stopped. The use of the API is more adapted to short sentences lire queries, and not for a continuous course. Moreover, when there is some noise in the room or near the microphone, the API crashed too.

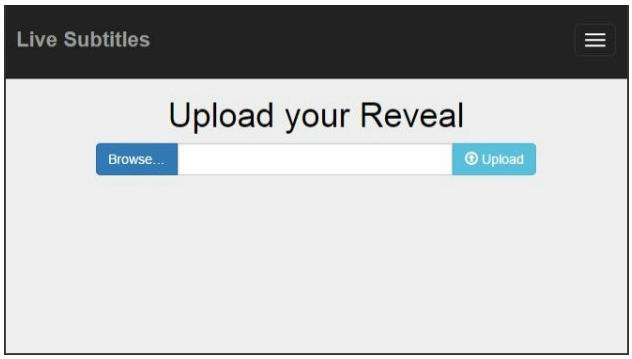
After that we faced a new problem : how to send the results from the API to the student's interface for correction? Our supervisor told us to look at socket.io. After weeks of reding and testing our program using socket.io we faced new problems. First of them, socket.io was hard to use, nevertheless we had a functional release of our project using socket.io. Secondly, at this step of the project we had to think about a way to save our data. We were handling table of table of table, which was heavy. We also thought about having a login form, with a database for users… We started to draw some database but our knowledge in SQL were too limited so we lost some time on it.

To find a solution to our problem we asked for some advices to our teacher Didier Donzer. He recommended us to look at meteor, which is a framework easy to use, and it will help us to communicate between the two interface and to settle the correction of a student on the other students. Furthermore, Meteor had its own database MongoDB, and other packages downloadable on atmosphere. Thus, we had to change the main technology we used in our project three weeks before its end. We first had to spend some time reading about meteor and testing some function. After that, we had to program again our app with meteor. It was much more easy than using socket.io, and a lot of useful packages were available to help us. However, it was hard for us to handle this technology in a short period of time. We spent a lot of time on small bugs. We couldn't achieve all our objectives.

The last problem we want to talk about is a problem of management. The relation between the customer and the developer. We didn't had the time to see our supervisor regularly, we asked for some meetings but we worked a lot alone on our understanding of the subject. But, almost at the end of the time we had for the project, in a meeting with our supervisor he realized that our app doesn't appear as he wanted to. The problem wasn't on the running of the app but more on the interface with the user. During the last week we have tried to match with his expectations with some revisions. The thing is that we will remember, in our futures projects, to meet regularly with the customer to show him the progress of our work. This will avoid a bad surprize at the end of the project if it is too far different from what it was expected.

Potential improvements

Security issue : we tried to add security mechanics to the app but obviously not enough to deploy the app to the public. Links between pages are not secured and there is no backup for the database.



A crucial improvement could be to add a File Uploader to show it with Reveal. Actually, the slides are pure HTML and not .pdf or .jpeg. This is not usable for lamba users, the HCI at this moment is not optimized. An easy way to make it possible is to parse the file into multiple images so Reveal.js can interpret them to slides then add the subtitles properly.

We have to use a terminal to use our app for now. Meteor is rebuilding each time we launch the app. To make it usable for anyone, we have to deploy it. This part need a little bit more time to accomplish since it implies having a distant server for the Database.

A major and necessary upgrade for the actual app would be using another Speech Recognition System because we are very limited by Google Speech : time listened is short, accuracy is low, modding is hard.

Conclusion

We had good times working in group on this project. It was the first time for all of us designing a web app, but more than this it was the first time we worked on a concrete project with a real and useful purpose. We have made our code well organized allowing us, easy review or improvement, whether by us or by any other interested group. It is a project we really enjoyed, and we hope to continue working on it to make it totally functional and effective. Furthermore, we already know what we can improve and we have found some suggestions to do so.

This first experience in Web application development will definitely help us in our future dev app projects.