

Pour assurer une rentabilité à notre projet, il nous faut le penser, le structurer en vue d'une large distribution sous de multiples formes.

Nous sommes dotés d'une identité porteuse de ce projet. Le groupe de travail est baptisé The SpeechApp Company. Visuellement, elle se constitue en premier lieu d'un logo : Un micro, élément central du projet, dont la tête est le logo de Mines ParisTech, signe de notre appartenance à l'école et de l'aide qu'elle nous a apporté dans le projet.

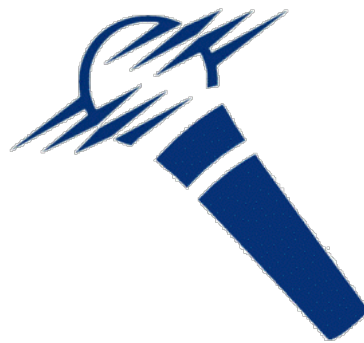


FIGURE 1 – Architecture proposée pour le projet The SpeechApp Company

0.1 Choix d'une architecture optimale pour notre projet

Distribuer notre projet tel quel présenterait à ce stade de nombreux défauts : - le coeur de notre technologie de reconnaissance vocale est directement accessible à tous. - une interface unique en ligne de commande constitue un blocage majeur pour la majorité des utilisateurs finaux et empêche une intégration large à des applications tierces.

Étudions l'opportunité d'adopter une architecture client/serveur pour ce projet.

Dans ce scénario, divers clients logiciels, potentiellement indépendant de The SpeechApp Company pourraient communiquer par requêtes/réponses (spécifiées par une API) avec les serveurs de The SpeechApp Company. Ces derniers seuls auraient accès au coeur algorithmique du projet, qui resterait ainsi exclusivement entre nos mains. Par leurs requêtes, les clients demanderaient l'analyse automatique de mots, l'ajout de nouveaux mots ainsi que toute autre opération pertinentes relative à l'analyse et la gestion d'une base de données de mots. L'accès à notre API serait monétisable forfaitairement ou à l'utilisation.

Les mots enregistrés par les clients seraient conservés dans des bases de données chez The SpeechApp Company. La location de ses bases de données hébergées serait monétisable. Alors, The SpeechApp Company pourrait prioritairement développer deux applications connectables au serveur : la première, SpeechCreator, permettrait l'enregistrement aisé de nouveaux mots dans les bases de données clients. La seconde, SpeechApp, permettrait, au travers d'une application Web riche, de tester la reconnaissance vocale en ligne.

Cette configuration permettrait aussi à une multitudes d'applications tierces d'utiliser notre technologie en ne voyant de l'extérieur qu'une API définissant le format des requêtes et réponses dans la communication entre clients logiciel et serveur.

Nous aboutirions alors à l'architecture représentée par le schéma suivant :

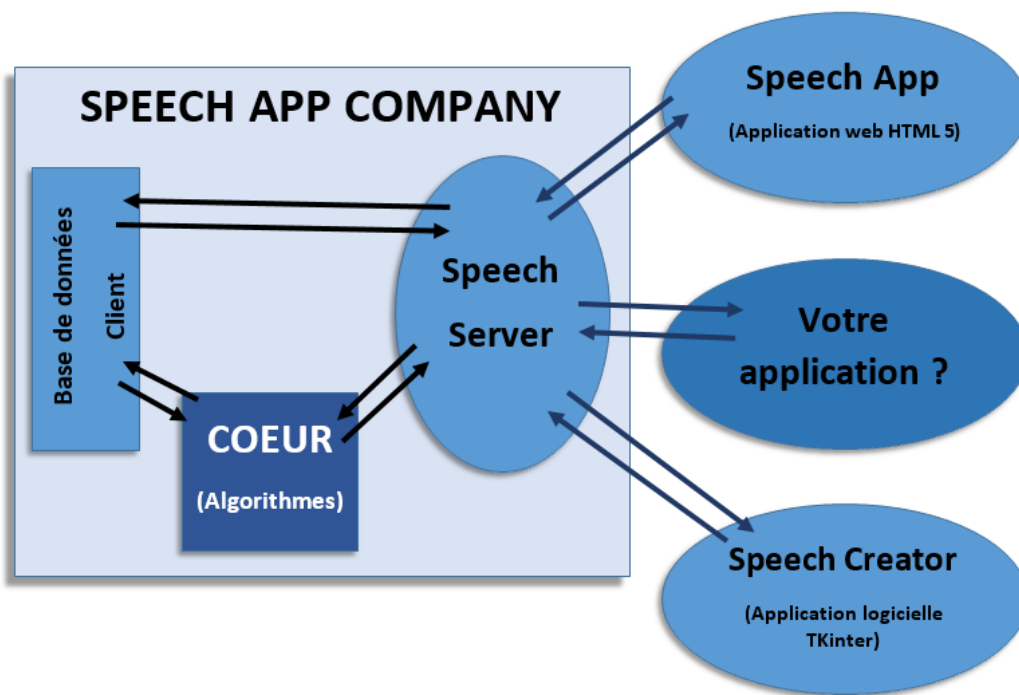


FIGURE 2 – Architecture proposée pour le projet The SpeechApp Company

Plus précisément dans le cadre des échanges entre le SpeechServer Les requêtes pourraient être traitées de la façon suivante : Le client (au sens logiciel toujours) envoie au SpeechServer une requête HTTP POST contenant un formulaire avec en particulier son identifiant, son mot de passe, la base de données qu'il veut utiliser, l'action qu'il veut faire effectuer au SpeechServer, et les données d'entrée qui lui sont associées. La requête analysée par le SpeechServer, les opérations adéquates ayant été réalisées par le coeur algorithmique, le SpeechServer répond au client par une réponse HTTP POST contenant des données au format XML. Le client peut alors lire et interpréter la réponse donnée par le SpeechServer.

Avec ses spécifications, nous obtiendrions le cycle suivant pour la reconnaissance d'un mot par SpeechApp :

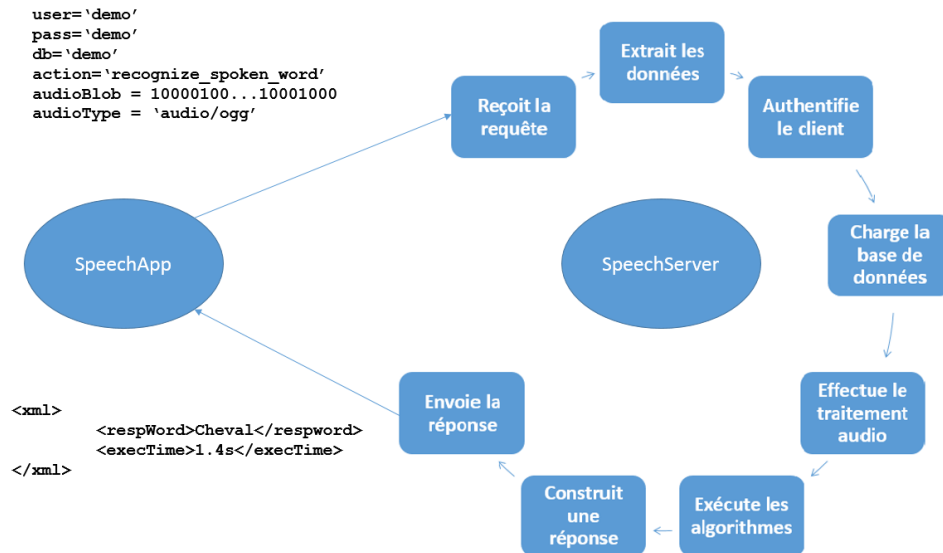


FIGURE 3 – Reconnaissance d'un mot par SpeechApp couplée au SpeechServer

L'architecture client/serveur proposée présenterai pour nous l'avantage de

- permettre la création d'un écosystème varié d'applications basées sur le coeur algorithmique de The SpeechApp Company via l'API de son SpeechServer, et générant ainsi des revenus
- conserver le coeur de notre travail entre nos mains et même de nous donner le contrôle sur toute la chaîne

L'architecture client/serveur proposée présenterai pour nos clients l'avantage de

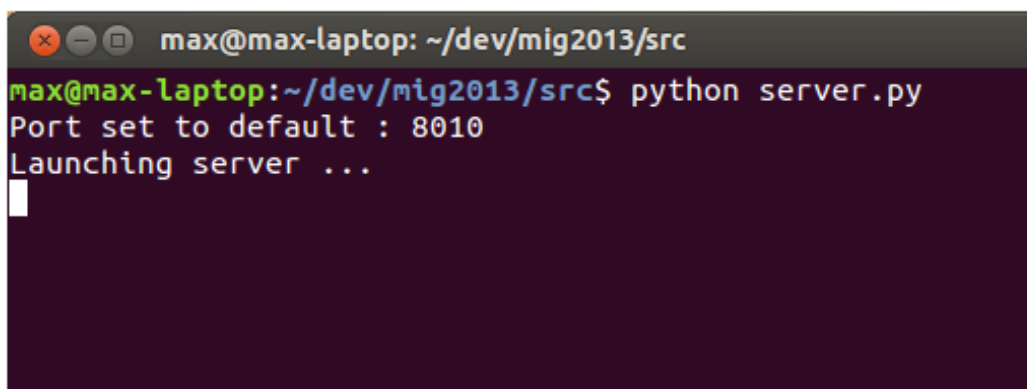
- ne pas se soucier du coeur algorithmique de la reconnaissance vocale, en n'y voyant que l'API de SpeechServer. Cette API peut offrir par ailleurs une grande liberté d'action
- n'avoir pas ou peu d'investissement initial de développement à effectuer, nos applications propriétaires SpeechApp et SpeechRecorder pouvant être intégrées sous forme de widgets aux applications tierces
- ne pas avoir à faire de lourds calculs eux-mêmes, ceux-ci étant réalisés par les machines de The SpeechApp Company
- les cycles de mises à jour seraient en majorité invisibles chez les clients, l'API restant immuables sur des cycles plus long (Long Term Support)

Au vu des nombreux avantages qu'elle présente, *Nous avons donc opté pour une architecture modulaire client/serveur pour notre projet.*

0.2 Réalisation du SpeechServer

Le SpeechServer a été codé en Python. Python a une librairie standard suffisamment riche pour n'avoir à traiter ce problème qu'à un haut niveau (en reception de requêtes selon leurs méthodes). De plus, ce choix facilite les interactions avec le coeur algorithmique : des imports et appels de fonctions depuis le SpeechServer suffisent.

Finalement, le SpeechServer prend seulement la forme d'un programme python à lancer sur un ordinateur.

A terminal window with a dark background and light-colored text. The title bar at the top reads 'max@max-laptop: ~/dev/mig2013/src'. The terminal content shows the command 'python server.py' being executed, followed by the output 'Port set to default : 8010' and 'Launching server ...'. A cursor is visible on the line following the last output.

```
max@max-laptop: ~/dev/mig2013/src
max@max-laptop:~/dev/mig2013/src$ python server.py
Port set to default : 8010
Launching server ...
```

FIGURE 4 – Le SpeechServer lancé

Il écoute alors les requêtes sur le port 8010 (par défaut) de l'ordinateur. Lorsqu'il en reçoit, il interagit avec le coeur algorithmique et le système de gestion de bases de données mis en place.

0.3 Système de Gestion de Base de Données (SGBD)

Le SGBD doit permettre de stocker et gérer les fichiers audio associés aux mots (au moins une dizaine d'enregistrement par mot), les modèles de markov cachés qui leurs sont associés ainsi que les données d'authentification des applications clientes.

Le standard actuel de gestion de bases de données est le modèle relationnel basé sur le langage SQL. Néanmoins, dans le cas précis de stockage de fichiers relativement lourds (> 0.1 Mo), la lecture/écriture des données directement sur le disque dur s'avère plus performante.

Nous avons donc fait le choix de stocker nos données sur le disque dur du serveur, en enregistrant les fichiers audio en format brut, et les autres données (modèles de Markov, données d'authentification) comme des objets Python, avec le module pickle de la librairie standard.

Un module python db.py a été développé par nos soins pour gérer efficacement nos fichiers

Comme les accès en lecture/écriture à la mémoire RAM sont bien plus rapide que les accès aux disques dur, *On pourrait obtenir un gain de vitesse significatif pour la reconnaissance vocale en chargeant l'intégralité des données en mémoire RAM au démarrage du SpeechServer* : la vitesse en lecture/écriture sur un disque dur est de l'ordre de 50 Mo / s. Sur la RAM, elle est de l'ordre de 1 Go / s, soit un gain d'un facteur 20 pour les opérations en mémoire.

Néanmoins, la quantité de mémoire RAM nécessaire serait très importante, croissant linéairement avec le nombre de mots enregistrés. Les coûts engendrés pourraient être importants.

Tâchons de dimensionner l'infrastructure serveur dont nous aurons besoin.

0.4 Dimensionnement de l'infrastructure de calcul de The Speech App Company

Il nous faut d'abord définir les variables relatives au fonctionnement commercial de The Speech App Company ainsi que leurs valeurs de référence.

0.4.1 Hypothèses de fonctionnement

Soit M le nombre de clients de The SpeechApp Company. La référence sera $M = 1000$. Soit N le nombre moyen de mots dans les bases de données de chaque client. Nous prenons pour référence $N = 2000$ mots : un dictionnaire comme le Petit Robert en contient 60000.

Soit J le nombre de requêtes par seconde. Nous prendrons pour référence $J = 1000$ requêtes / seconde

On vise le traitement des requêtes en 1s. On gère donc J requêtes en simultané.

Les fichiers audio bruts envoyés par les clients au serveur pèsent environ 100 ko chacun. Les Modèles de Markov Cachés (MMC) associés aux mots pèsent environ 50 ko pour chaque mot. Lors des traitements sur ces fichiers audios, on estime qu'on a besoin de créer 5 fichiers audios temporaires, d'environ 100 ko chacun.

0.4.2 Dimensionnement en mémoire RAM et espace disque

Les fichiers audios bruts (10 par mot par défaut) et les MMC sont conservés sur le disque dur. Il faut donc $(10 * 100 \text{ ko} + 50 \text{ ko}) * N * M = 2.100 \text{ To}$ d'espace sur le disque dur.

Par ailleurs, on charge les MMC en mémoire, soit un espace RAM nécessaire de $50 \text{ ko} * N * M = 100 \text{ Go}$

Lors des opérations, si les $5 * 100 \text{ ko}$ de fichiers temporaires sont créés en RAM, et qu'on gère environ $J = 1000$ requêtes en parallèle, il nous faut 500 Mo de RAM en plus, ce qui est marginal.

Au vu des capacités mémoire en informatique, toutes puissances de 2, *Dans le cadre de référence, il nous faut au moins 128 Go de RAM et 4 To d'espace disque*

0.4.3 Dimensionnement réseau

Pour la reconnaissance de mots, tâche la plus courante, Le serveur reçoit J requêtes de 100 ko (fichiers audio). Il faut donc recevoir 100 Mo / s de données. Le débit descendant (vers le serveur) doit donc être supérieur strictement à 100 Mo / s. Le serveur répond par des fichiers ne contenant que du texte, de taille négligeable devant celle des fichiers audio. Le débit montant (depuis le serveur) n'est donc pas un facteur discriminant dans le choix d'une connexion au réseau.

On veillera à avoir une connexion d'au moins 200 Mo / s)

0.4.4 Dimensionnement des éléments de calculs

Sur un ordinateur d'une puissance de calcul de 1 GFlops, on observe que lors de la reconnaissance d'un mot, l'unique opération dont la complexité dépend du nombre de mots en jeu, l'exécution de l'algorithme Forward (linéaire) prenait 0.005s sur une base de 100 mots.

Ainsi pour réaliser $J = 1000$ reconnaissances en simultannée sur des bases de $N = 1000$ mots avec un temps d'exécution de l'algorithme Forward de moins de 0.5s, *il nous faut une puissance de calcul d'au moins 100 Gflops.*

Les processeurs de dernière génération dédié au calcul atteignent ce niveau de performance. Le Intel Xeon E5-2670 atteint ainsi en théorie 330 Gflops

0.4.5 Choix de l'infrastructure et coûts liés

Connaissant les caractéristiques minimales du serveur : en termes d'espace RAM, disque dur, de connexion réseau et de puissance de calcul, nous pouvons choisir le serveur le plus adapté à nos besoins.

L'hébergeur OVH propose une gamme de serveurs de calcul pour les entreprises :

Modèle	SP-64	SP-128	MG-128	MG-256
Prix	81.99€ HT /Mois	131.99€ HT /Mois	202.99€ HT /Mois	302.99€ HT /Mois
Installation	99.99€ HT	99.99€ HT	99.99€ HT	99.99€ HT
Commander	Commander	Commander	Soon	Soon
Disponibilité				
Contrôle	IPMI / KVMolP	IPMI / KVMolP	IPMI / KVMolP	IPMI / KVMolP
Châssis	1U / T3	1U / T3	1U / T3	1U / T3
CPU	Intel Xeon E5-1620v2	Intel Xeon E5-1650v2	Intel Xeon 2x E5-2650v2	Intel Xeon 2x E5-2670v2
Cores / Threads	4c / 8t	6c / 12t	16c / 32t	20c / 40t
Fréquence / Burst	3.7 GHz+ / 3.9 GHz+	3.5 GHz+ / 3.9 GHz+	2.6 GHz+ / 3.4 GHz+	2.5 GHz+ / 3.3 GHz+
RAM	64 Go DDR3 ECC	128 Go DDR3 ECC	128 Go DDR3 ECC	256 Go DDR3 ECC
Disques Durs	2x 2 To SATA3 SSD ⁽¹⁾ / SAS ⁽¹⁾	2x 2 To SATA3 SSD ⁽¹⁾ / SAS ⁽¹⁾	2x 2 To SATA3 SSD ⁽¹⁾ / SAS ⁽¹⁾	2x 2 To SATA3 SSD ⁽¹⁾ / SAS ⁽¹⁾
RAID	SOFT HARD ⁽¹⁾	SOFT HARD ⁽¹⁾	SOFT HARD ⁽¹⁾	SOFT HARD ⁽¹⁾
Bande passante	300 Mbps ⁽¹⁾	300 Mbps ⁽¹⁾	400 Mbps ⁽¹⁾	500 Mbps ⁽¹⁾

FIGURE 5 – Gamme de serveur de calculs d'OVH

Nous devons disposer de 128 Go de RAM, de 4 To d'espace disque, de 100 Mo/s de connexion au réseau. Aussi le processeur Intel Xeon E5-1650 v2 ne dépasse pas les 70 Gflops et ne valide donc pas le critère de puissance de calcul. Le bi-ES-2650 atteint 140 Gflops, et le bi-ES-2670 330 Gflops.

Afin d'avoir une certaine marge, nous préférons prendre le processeur bi-Intel Xeon ES-2670. Nous sélectionons donc le serveur MG-256 de OVH pour 303 euros HT / mois qui valide nos critères de performance avec une marge conséquente. À des fins de redondance, nous aurions besoin de 2 serveurs identiques, pour donc 606 euros HT / mois.

Nous nous sommes contentés d'un stockage des données intégralement sur disque dur et de moyens bien plus réduits lors de la phase de développement.

Les spécifications du SpeechServer et du SGBD ayant été définis, il devient possible et nécessaire de construire des applications se fondant dessus.

0.5 SpeechRecorder

Il est nécessaire de proposer aux clients une interface plus simple à appréhender que la console. C'est pourquoi nous avons développé l'application logiciel SpeechRecorder, qui permet aux clients enregistrés dans nos bases de données d'authentification (s'acquittant d'une licence),

d'ajouter des mots à leur bases de données. Elle devrait permettre à terme, de gérer l'intégralité des bases de données clients.

Cette interface a été réalisée avec la librairie TKinter de Python, la librairie graphique Python la plus simple et la plus largement disponible : Elle est incluse dans les paquets de base de Python.



FIGURE 6 – Authentification d'un client au SpeechRecorder

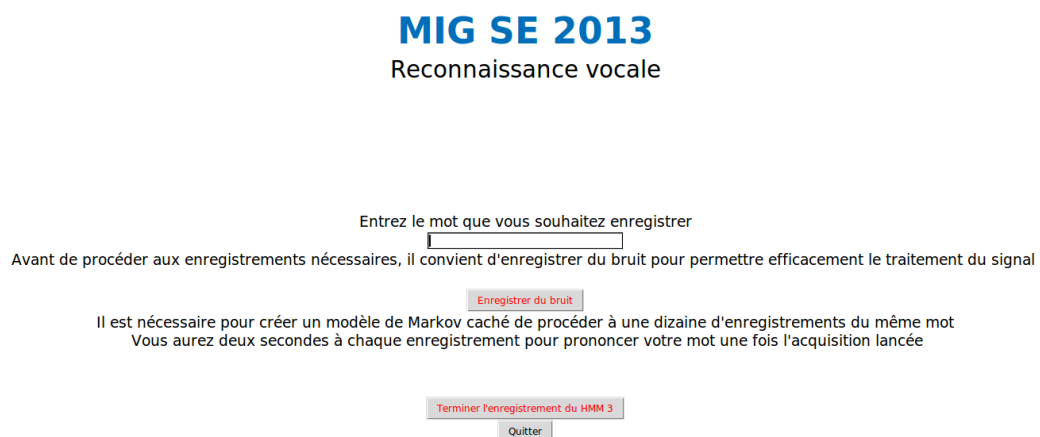


FIGURE 7 – L'interface du SpeechRecorder

Pour entrer un nouveau mot dans une base de données client, par défaut 10 enregistrements sonores sont pris par SpeechRecorder. La librairie additionnelle pyaudio est utilisée pour ce faire.

0.6 SpeechApp

SpeechApp est le démonstrateur principal de notre projet. Il s'agit d'une application web permettant au grand public de tester notre technologie de reconnaissance vocale de mots isolés. À l'aide des dernières APIs HTML5 (élaborées depuis le début d'année 2013), l'utilisateur peut s'enregistrer sans l'installation de logiciel auxiliaire. Son enregistrement audio est transmis au SpeechServer (selon le schéma spécifié plus haut) qui renvoie le mot trouvé. Pour ce démonstrateur, une application web a été choisie car elle fonctionne sur tout terminal doté d'un navigateur web récent sans nécessiter la moindre installation : nous l'avons conçue de façon à ce qu'elle soit adaptée aussi bien aux grands écrans d'ordinateur, qu'à ceux plus petits des tablettes et smartphones. On qualifie ce type de design de "Responsive".

Notre démonstrateur SpeechApp est disponible à l'adresse <http://speechapp.wumzi.info/>.

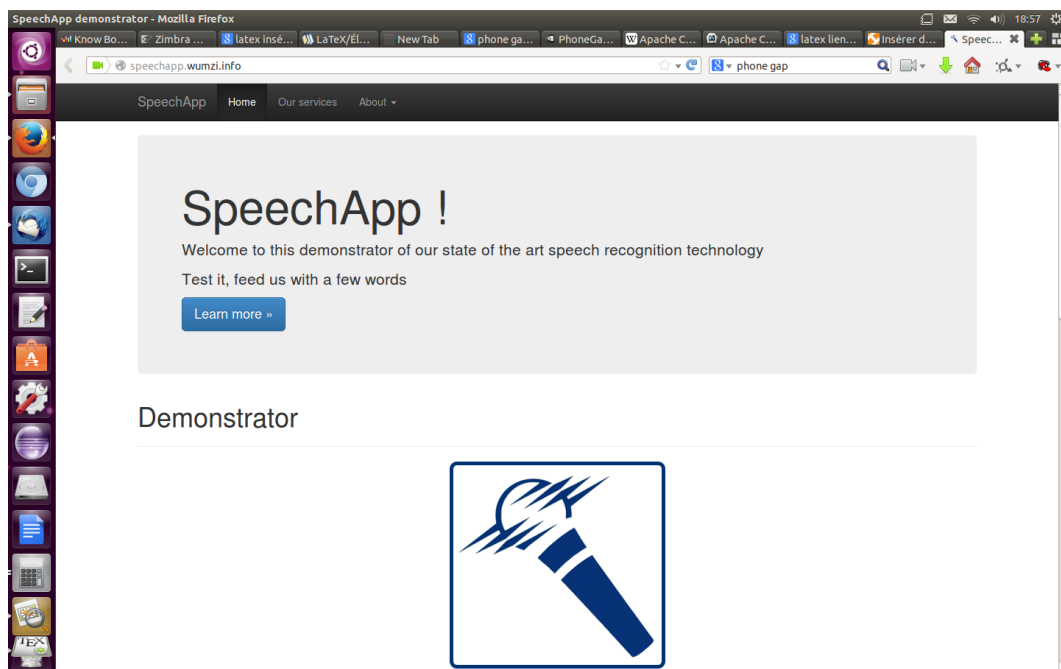


FIGURE 8 – Le démonstrateur SpeechApp sur ordinateur

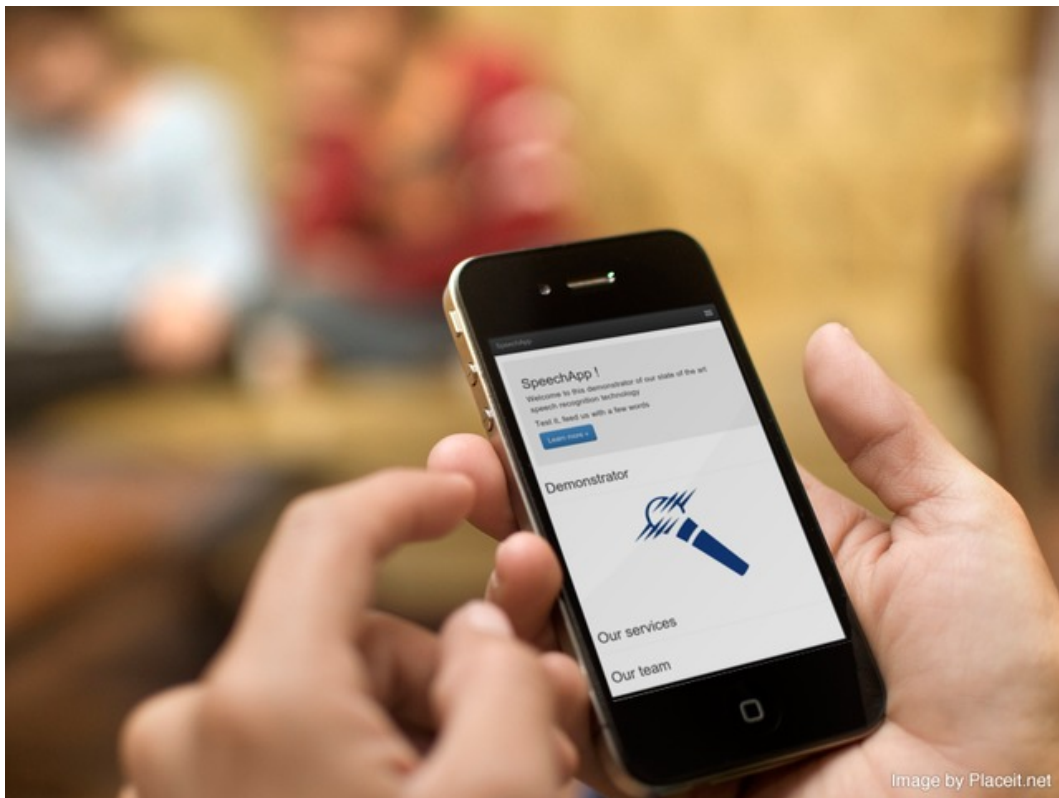


FIGURE 9 – Le démonstrateur SpeechApp sur iPhone

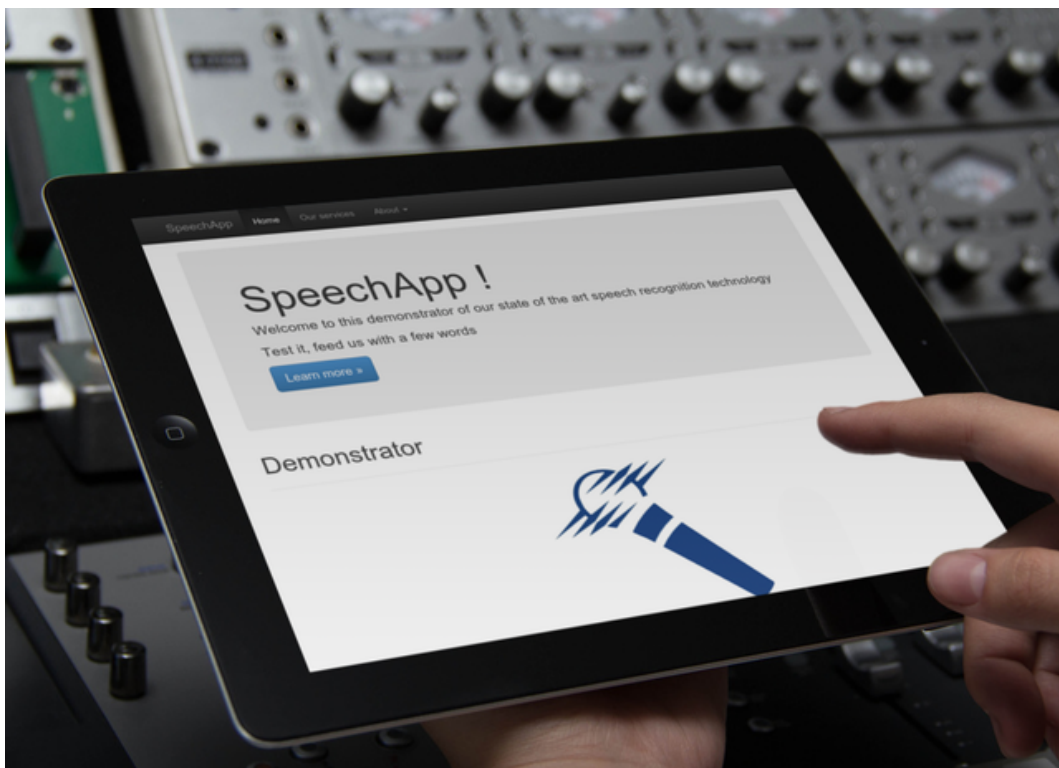


FIGURE 10 – Le démonstrateur SpeechApp sur iPad

Une difficulté néanmoins a été de rendre l'enregistrement audio fonctionnel sur la majorité des navigateurs. Nous assurons la compatibilité pour les moteurs Gecko et WebKit récents soit

les dernières versions de Firefox, Chrome et Safari ainsi que leurs éditions mobiles.

SpeechApp n'a en elle-même pas d'autre fonction que celle de démonstrateur, néanmoins ses modules peuvent être distribués aisément, sous forme de widgets intégrables n'importe où.

Aussi, une application web reprenant des modules de SpeechApp pourrait être transformée aisément en application native pour smartphone iOS, Android, Windows Phone, Firefox Mobile ou encore en application Windows 8. Des frameworks open-source font ce travail presque automatiquement (par exemple le framework PhoneGap développé par Adobe : <http://phonegap.com/>).