**KANJI JISHO**

**Projet DAPM**

**Etudiants : Craioveanu Andrada**

**Matreata Alexandra**

**Groupe 1240F**

**SOMMAIRE**

1. **Introduction**
   1. Problèmes résolues par l’application
   2. Mode de travail en équipe
   3. Technologie utilisée
2. **Implémentation**

2.1 Spécifications de l’application

2.2 Design de l’application

2.3 Architecture de l’application

2.4 Fonctionnalités de l’application

2.5 Structure code

1. **Conclusions et perspectives**
2. **Références**
3. **INTRODUCTION**

L’application “Kanji Jisho” est une application de type dictionnaire japonais- anglais qui aide l’utilisateur à mieux apprendre à lire et écrire avec les symboles japonais, les kanji.

Les kanji sont des caractères généralement empruntés au système d'écriture de l'ethnie chinoise Han. Ces caractères sont les éléments d'un des trois ensembles de caractères de l'écriture japonaise, avec les hiragana et les katakana, ces deux derniers étant regroupés sous le terme de kanas.

Les kanji, eux, sont des caractères idéographiques, ou idéogrammes. Ils représentent à la fois un sens et un son. Contrairement au chinois où chaque idéogramme ne comporte généralement qu'un seul sens et qu'une seule prononciation, en japonais, ils peuvent comporter plusieurs sens et plusieurs prononciations en fonction du contexte.

L’application donne à l’utilisateur la possibilité de chercher des symboles désirés par deux méthodes : la méthode directe avec la caisse de recherche, et la méthode à l’aide des critères de recherche variés.

L’application se propose ainsi d’aider aussi l’utilisateur intéressé seulement d’apprendre la signification d’un symbole quelconque que celui qui veut bien maitriser la langue japonaise a un niveau académique.

Pour réaliser cette application, le travail en équipe a été fait d’une manière équitable, en partageant aussi la partie technologique (de recherche des termes technologiques nécessaires pour le développement de l’application et l’écriture de code) et aussi la partie de documentation. Cette dernière, a été fait premièrement plus ample par l’un des membres de l’équipe, Alexandra, qui a des connaissances de langue Japonaise et près le fin aussi par l’autre membre de l’équipe, cette collaboration ayant le but de mettre le données ensemble.

Le développement de l’application a été réalisé en utilisant la technologie Swift pour l’écriture des applications iOS pour des dispositifs Apple.

Le programme qu’on a utilisé pour l’édition du code Swift est XCode.

1. **IMPLEMENTATION**
   1. **Spécification de l’application**

L’application Kanji Jisho donne l’utilisateur la possibilité de chercher des symboles kanji soit par la méthode directe avec la caisse de recherche soit par la méthode à l’aide des critères de recherche variés.

L’utilisateur peut insérer dans la caisse de recherche des mots anglais ou même des romaji (des symboles japonais écrits en suivant la prononciation avec les lettres de l’alphabet latin).

Les critères disponibles pour le deuxième type de recherche seront (apart le sens en anglais, les onyomi ou les kunyomi qu’il peut introduire dans la caisse de recherche) un panneau des radicaux (morceaux qui composent le kanji) d’où l’utilisateur peut choisir un ou plusieurs signes qui font partie du symbole recherché et le nombre de lignes utilisées dans son écriture.

En fonction de la langue choisit par l’utilisateur, le mot peut être cherché soit en Japonaise soit en Anglais.

La structure de la page des résultats nous permet de voir les sens du kanji cherché, le symbole de celui-ci et la prononciation.

* 1. **Design de l’application**

En ce qui concerne le design de l’application, pour la structurer, on a utilisé 5 écrans : un pour la première page, pour choisir le type de recherche, deux pour chaque type ( soit pour introduire en utilisant des lettres le symbole cherché pour un Search direct, soit pour choisir un Search customisé) et deux autres pour la visualisation des résultats.

Faisant la corrélation avec le langage de programmation utilisée, Swift, comme design on a utilisé des views de type ViewController, TableView Controller, et des éléments comme ScrollView, Label, TextField, etc.

Les ViewControllers sont un lien vital entre les données de l’application et son aspect visuel. Chaque fois qu'une application iOS affiche une interface utilisateur, le contenu affiché a été géré par un ViewController.

Par conséquent, les ViewControllers ont fourni le cadre du squelette sur lequel on a construit l’application.

Pour l’affichage des résultats recherchés, on a utilisé un TableView Controller. La classe UITableViewController crée un objet controller qui gère une TableView. Pour notre application, on a initialisé ce TableView avec un nombre de lignes égal au nombre de kanji qui seront retournés.

En ce qui concerne le mode d’affichage des radicaux pour la recherche customisé des kanji, on a utilisé un ScrollView. La classe UIScrollView fournit un support pour l'affichage du contenu qui est supérieure à la taille de la fenêtre de l'application. Il permet aux utilisateurs de faire défiler sein de ce contenu en faisant des gestes de glissage, et pour zoomer à partir de portions du contenu en faisant des gestes de pincement. Pour notre application, le ScrollView a été initialisé avec la dimension correspondante aux nombres de radicaux stockés dans la mémoire.

Pour la contribution de l’utilisateur, on a utilisé des éléments UI comme des TextFields (UITextField) et des buttons (UIButton).

* 1. **L’architecture de l’application**

En ce qui concerne l’architecture et la structure de l’application Kanji Jisho, on peut dire que celle-ci est une architecture MVC (Model – View - Controller) avec des rôles combinées.

Pour les applications Swift, les composantes ViewController mettent ensemble les éléments Controller et View de la forme originale de ce type de structure (MVC).

C’est-à-dire que, dans notre cas, les ViewController fonctionnent aussi comme des entités pour afficher contenu, mais aussi pour envoyer des notifications, pour valider les dates introduites par l’utilisateur (par exemple pour s’assurer que le chaines de caractères cherchés n’est pas vide), pour communiquer avec des autres ViewControllers qui seront responsables d’analyser les dates d’utilisateur.

Pour le dictionnaire de kanji, les ViewControllers vont communiquer entre eux pour prendre l’input de l’utilisateur d’un écran et le passer comme paramètre dans l’écran des résultats (pour la recherche directe).

En ce qui concerne le stockage des données, les radicaux et les kanji sont stockés dans un fichier XML avec des éléments kanji qui ont la structure suivante :

[<kanji>](file:///C:\Users\craiovea\Desktop\kanjiXML.xml)

<semn>亜</semn>

<strokes>7</strokes>

<on>- </on>

<translation>rank next, come after, Asia, sub-, -ous (inacids)</translation>

<kun>a</kun>

<radical>二</radical>

</kanji>

L’alphabet des syllabes, hiragana, est aussi stocké dans un fichier XML, utilisé pour transformer l’input de l’utilisateur dans le TextField de l’ecran « Direct Search » pour la recherche avec des mots en japonais de romaji en hiragana.

Le flow de l’application se fait dans la manière suivante : du premier écran, en fonction du choix d’utilisateur, on passera vers SearchView pour la recherche simple, ou vers le CustomView pour la recherche customisé. Après avoir complété les UserInputs correspondants a chaque ecran et avoir appuye sur un boutton de recherche, un TableView est affiche avec les résultats de la recherche. Si l’utilisateur veut savoir plus sur un des résultats, il peut cliquer sur la cellule le contenant et ainsi ouvrir le dernier View contenant les détails du kanji respectif.

* 1. **Fonctionnalités**

Le premier écran de l'application contient deux boutons, <<Custom Search>> et <<Search>>. Si on touche le bouton <<Search>>, un deuxième écran nous présente deux champs texte ou on peut écrire à l'aide d'un clavier en alphabet latin le mot cherché en anglais et Japonaise.

Si on veut faire un recherche customisé, le premier bouton, celui de <<Custom Search>> va ouvrir un nouvel écran avec les éléments suivants: un champ texte ou on peut introduire le mot ; un champ texte ou l'utilisateur peut introduire le numéro de lignes du kanji recherché (si vide, on va chercher tous les kanji, n'importe leur numéro de lignes), un élément ScrollView avec les radicaux d'où on peut sélectionner un radical qui fait partie du kanji recherché et un bouton <<Search>>.

Les boutons <<Search>> du premier et deuxième écran vont appeler une fonction qui va parcourir un fichier XML avec des kanji et leurs propriétés (comme nous avons montré avant) et va afficher en utilisant un tableView tous les kanji qui remplissent toutes les conditions de la recherche. Pour la recherche directe, la méthode prendra l’input de l’utilisateur qui sera comparé avec les sous-éléments *onyomi* ou *kunyomi* ou des mots présents dans l’élément XML « translation ».

En cliquant sur un des éléments de TableView, on verra un autre écran avec tous les sous-éléments (détailles) de l’élément kanji sélecté du fichier XML.

* 1. **Structure du code**

Le code est structuré en respectant un pattern adapté du Model-View-Controller :

* Premièrement, on a un directeur **Controllers** qui contient :
* SearchViewController (pour le premier écran ou on choisit le type de recherche)
* ViewController (pour l’écran de la recherche directe)
* CustomViewController (pour l’écran de la recherche customisé)
* TableViewController (pour l’affichage des tous les kanji qui remplissent les conditions de recherche pour les deux types de recherche)
* DetailViewController (pour l’écran avec les détails d’un des kanji retournés par la recherche)
* Un directeur **SupportingFiles**, pour les **Ressources** qui contient :
* Les deux fichiers XML avec les kanji et avec l’alphabet des syllabes Hiragana
* Les images
* Les deux classes pour parser les fichiers XML
* La classe principale, **AppDelegate**, crée automatiquement en XCode

Quelques méthodes utilisées :

La méthode pour la transformation de l’input du romaji en hiragana :

     @IBAction func searchButton(sender: AnyObject) {

        if(tf.text != "")

        {

            tfText = tf.text

            j = dataArray.count - 1

            scrisfinal = tf.text

            for i in 0...j

            {

                aux = dataArray.objectAtIndex(i).objectForKey("semn") as String

                cheie = dataArray.objectAtIndex(i).objectForKey("citire") as String

                var match = aux.rangeOfString(myRegex, options: .RegularExpressionSearch)

                var substaux = aux.substringWithRange(match!)

                match = cheie.rangeOfString(myRegexcitire, options: .RegularExpressionSearch)

                var substcheie = cheie.substringWithRange(match!)

                scrisfinal = scrisfinal.stringByReplacingOccurrencesOfString(substcheie, withString: substaux, options: nil, range: nil)

            }

            tf.text = scrisfinal

        }

Cette méthode utilise deux expressions régulières, myRegex = "(.){1}" et myRegexcitire = "[a-z]{1,3}". La première des deux extrait le signe de la syllabe hiragana du fichier XML sans les espaces qui auront pu être insérés dans le fichier XML par accident et la deuxième sépare l’input dans des syllabes qui peuvent avoir des correspondants en hiragana. Après avoir trouvé les signes des syllabes correspondantes, la méthode remplace dans la variable de type String « scrisfinal » chaque occurrence du syllabe avec son signe correspondant.

La methode pour l’affichage du panneau avec les radicals :

    func buttonMake()

    {

        var edeja = false

        var caleKanji = NSBundle.mainBundle().pathForResource("kanjiXML-2", ofType: "xml")

        var urlKanji:NSURL = NSURL (fileURLWithPath: caleKanji!)!

        var parserKanji : kanjiXMLParser = kanjiXMLParser.alloc().initWithURL(urlKanji) as kanjiXMLParser

        //verification du radical dans le xml

        dataArray = parserKanji.posts

        j = dataArray.count - 1

        for i in 0...j

        {

            aux = dataArray.objectAtIndex(i).objectForKey("radical") as String

            if(edeja)

            {edeja = false}

            if(radicals.count>=2)

            {for k in 0...radicals.count - 1

            {

                if radicals[k] as NSString == aux

                {edeja = true}

                }}

            if (!edeja)

            {

                radicals.append(aux)

            }

        }

        for i in 0...radicals.count-1

            {   aux = radicals[i]

                let button = UIButton.buttonWithType(UIButtonType.System) as UIButton

                button.frame = CGRectMake(x, y , 50, 50)

                button.backgroundColor = UIColor.lightGrayColor()

                button.setTitle(aux, forState: UIControlState.Normal)

                button.setTitleColor(UIColor.blackColor(), forState: UIControlState.Normal)

                button.addTarget(self, action: "radicalSelected:", forControlEvents: UIControlEvents.TouchUpInside)

                self.scr.addSubview(button)

                if(x<=185)

                {x = x+50}

                else

                {y = y+50

                    x = 35}

        }

    }

Cette méthode cherche parmi les données du fichier XML tous les radicals et les enregistre dans un vecteur « radicals », tenant compte du fait que les radicals doivent apparaitre chacun une seule fois. Puis, en utilisant une boucle for, la méthode itère le vecteur et pour chacun de ses éléments, construit un élément de design de type UIButton.

1. **CONCLUSIONS ET PERSPECTIVES**

Travailler pour ce projet a été une opportunité intéressante pour nous d'apprendre et de travailler avec un nouveau langage de programmation et un nouveau système d'exploitation.

Même si c’était aussi un peu difficile et défiant, quand l’application a gagné contour, la satisfaction et la curiosité d'acquérir plus d'informations a aussi augmenté.

C’est pourquoi, en ce qui concerne les perspectives d'avenir pour cette application, nous avons pensé à la possibilité de mettre en œuvre un système de authentification, en ordre pour chaque utilisateur d'être en mesure de créer un dictionnaire de kanji personnalisé, partager leur dictionnaire et voir des autres. Aussi, pour mieux aider les gens apprenant la langue, on essayera de mettre en œuvre un système de recherche sur le niveau JLPT des kanji recherchés et ainsi essayer de résoudre un problème « persistante » dans la vie de l’étudiant de la langue japonaise.

1. **REFERENCES**

[*https://github.com/swiftdetut*](https://github.com/swiftdetut)

*https://developer.apple.com*

[*https://github.com/johnno1962*](https://github.com/johnno1962)

[*https://www.learnswift.tips*](https://www.learnswift.tips)

[*http://www.saiga-jp.com/kanji\_dictionary.html*](http://www.saiga-jp.com/kanji_dictionary.html)