Rapport de Développement mobile

Frédérick Fabre Ferber , L3 Informatique

12 avril 2019

Résumé

Le but de ce rapport est de présenter l'élaboration d'une application mobile sur deux systèmes différents. L'un sous **Android** en **Java** et l'autre sous **iOS** en **Swift**.

L'application décrite ici est un **Mastermind** où l'on devra deviner une combinaison de 4 couleurs aléatoires pour ensuite pouvoir inscrire son nom et le temps que l'on aura mis pour deviner la combinaison dans une liste de scores.

*Durant ce rapport nous verrons les détails de l'application pour les deux **plateformes**.*

1 Fonctionnement général de l'application

1.1 Principe et règles du Mastermind

Mastermind est un jeu de société à deux joueurs créé 1972. Le premier joueur va élaborer une combinaison de 4 couleurs parmi les différents pions de couleurs disponible. Le deuxième joueur doit alors trouver cette combinaison c'est-à-dire les bonnes **couleurs** aux bonnes **positions**.

Pour cela, il va introduire 4 pions sur une rangée, viens ensuite *l'arbitrage* du second joueur, en effet celui-ci va déterminer si les pions de la rangée sont conformes à la combinaison en plaçant des pions dans la grille d'arbitrage :

— Un pion **Blanc** si la couleur du pion n'est pas dans la combinaison.
— Un pion **Rouge** si la couleur du pion est dans la combinaison mais est mal placée.
— Un pion **Noir** si le pion est bien placé et à la bonne couleur.

***Attention :** la couleur des pions pour l'arbitrage diffère beaucoup d'une version de Mastermind à une autre, durant tout le rapport et dans l'application nous utiliserons cette manière d'arbitrer.*
(Voir 1)

Le joueur doit alors trouver la bonne combinaison en s'aidant des pions mis sur la grille d'arbitrage. S'il ne trouve pas en 10 manches, il a alors perdu.

1.2 Description de l'application

Nous avons présenté le principe général du Mastermind dans le but d'expliquer le fonctionnement de notre application.
Elle aura 5 écrans (une *Vue* sous iOS et une *Activité* sous Android) énumérés ci-dessous :

— **Le menu d'accueil** : C'est un menu constitué de trois boutons, un bouton *Jouer* qui amène à l'écran principal qui est le jeu Mastermind, un bouton **Règle** qui présente simplement les règles du jeu et un bouton **Scores** qui présente les différents scores de joueurs. (Voir figure 2)

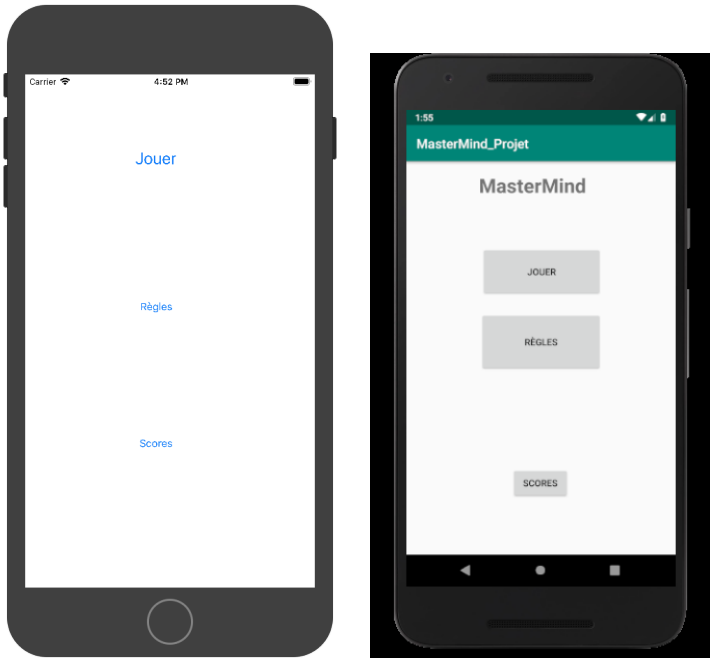
FIGURE 1 – Grille d'un mastermind avec les rangées de pions de couleurs au centre et les pions d'arbitrage à droite

— **Le jeu Mastermind** : Comme dit précédemment, c'est l'écran où le jeu se déroule le jeu. Contrairement au jeu classique, on ne joue qu'a un seul joueur et le but est de déterminer la combinaison qui est générée aléatoirement. On y retrouve une grille de 10 rangées de 4

  boutons qui sont notre grille de jeu, la même grille mais avec des boutons plus petits qui sont notre grille d'arbitrage. En bas de l'écran, nous avons 4 boutons de couleurs, lorsque l'on appui dessus la couleur des boutons change et on peut ensuite valider notre manche en cliquant sur le bouton **Valider** (*Ce sont les seuls boutons dans cet écran qui ont une interaction avec le joueur*). Chaque fois que l'on appuiera sur le bouton **Valider** la rangée correspondante à la manche prendra la couleur des 4 boutons et l'arbitrage se mettra en place. (Voir 1.1 pour l'arbitrage).Pour finir un Timer est situé en haut à droite de l'écran qui nous servira de score. Si le joueur arrive a deviner la combinaison il gagne et passe à

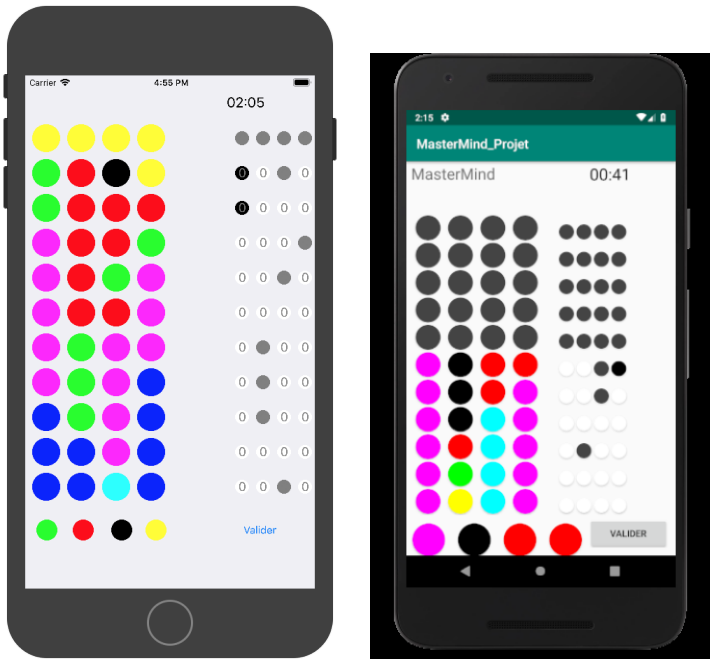
  l'écran suivant sinon il revient au menu principal. (Voir figure 3)

— L'écran **Gagner** : Lorsque que l'on a deviné la combinaison, on gagne le jeu ce qui nous amène à cet écran. Il est composé d'un texte disant que l'on a gagné avec le temps écoulé pour y parvenir. Nous devons mettre notre nom dans un champ prévu à cet effet, on appuiera alors sur un bouton pour rajouter notre nom et le temps à notre liste des scores. (Voir figure 4)
— La liste des scores : La liste des scores référence tous les joueurs ayant gagné à notre application Mastermind en marquant le *nom* du joueur et le **temps** qu'il a mis pour finir le jeu dans une liste. On a un bouton **Quitter** pour revenir au menu principal. (Voir figure 5)



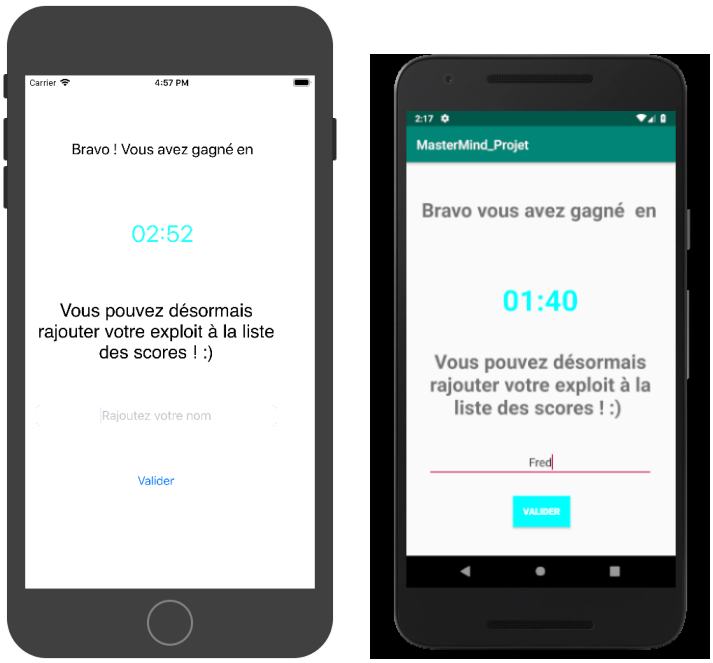
(Menu iOS) (Menu Android)

FIGURE 2 – Menu du jeu en iOS et Android



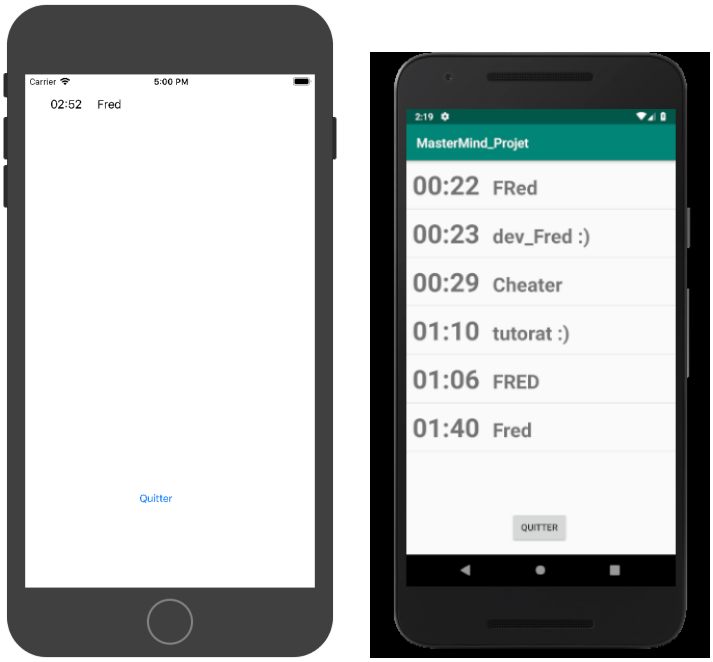
(iOS) (Android)

FIGURE 3 – Jeu Mastermind en iOS et Android



(iOS) (Android)

FIGURE 4 – Ecran **gagner** avec le nom à rentrer dans un champ.



(iOS) (Android)

FIGURE 5 – Liste des scores avec des scores rentrés par d'autres joueurs

2 Architecture de l'application

Nous avons vu le principe général de l'application nous allons pouvoir voir maintenant les détails de son architecture. Nous allons décrire chacune des **Vues**/**Activités**.

Le but ici n'est pas de décrire complètement dans les moindres recoins chaque composant de l'application, mais de comprendre son fonctionnement plus en détails. Nous y décrirons lorsque c'est nécessaire la mécanique sous XCode et sous Android Studio (Swift et Java).

2.1 Le menu

Comme dit dans la description de l'application (Voir 1.2) le menu sert de passerelle entres les différentes **Vues**/**Activités** (nous verrons lesquelles plus tard). Cette transition se fait par l'intermédiaire :

— D'un objet **Intent** sous Android Studio, on va créer une instance de cet objet qui prendra en paramètre le nom de la classe (L'activité) vers laquelle on veut se diriger. On utilisera ensuite la fonction **startActivity()** qui prend justement en paramètre cet **Intent**.

— D'un **segue** sous XCode, c'est lui qui "transporte" d'une Vue à une autre. Sous XCode nous avons uniquement besoin de faire un **CTRL + glisser** sur une autre vue via le **Storyboard** pour que la transition se fasse, mais nous verrons plus tard qu'on peut le faire de manière programmatique.

Nous allons donc transiter vers les autres Activités/Vues (Voir) depuis ce menu, c'est la classe MainActivity.java sous Android et ViewController.swift, Main.storyboard.

*Note : Le mot clé **Intent** signifie de l'anglais une intention et une **Segue** est un mot italien qui signifie une transition*

2.2 Le Jeu Mastermind

Le jeu en lui même est géré par **MastermindActivity.java** et **JouerController.swift**.

2.2.1 Grille et boutons

Comme dit dans en 2 nous représentons notre grille de jeu et la grille d'arbitrage avec des tableaux de boutons que ce soit en Java ou en Swift (Button[][] et [[UIButton]]). Le choix des boutons a été pris car il était plus facile de pouvoir changer sa forme et ses attributs (couleur, texte, etc). Ceux-ci sont stockés dans des tableaux pour faciliter leur accès. On dispose de deux tableaux 11 par 4, un pour les pions normaux et l'autre pour les pions d'arbitrage.
La forme de rond pour les boutons est réalisable de deux manières différentes :

— Avec XCode, par l'intermédiaire de notre fonction **setupButtonStyle()** qui prend un UIButton en paramètre et va changer le "cornerRadius" du bouton pour qu'il passe d'une forme rectangulaire à ronde.

— Avec Android Studio en créant un **Drawable** *pion.xml* pour rajouter un attribut shape comme "oval" ce qui lui donnera une force de rond plutôt que rectangulaire.

Remarque : Ces deux tableaux de boutons sont affichés chacun dans un TableLayout sous Android .

La mise en place de ces grilles est effectué par les fonctions **setGrilleMastermind()** et **setScoreMastermind()** en Java. Et par les fonctions **setTabButton()** et **setTabScoreButton()** en Swift.

2.2.2 Interaction avec le joueur et déroulement de la partie

Le plateau de jeu étant mis en place, il faut maintenant pouvoir jouer. Nous disposons de 4 boutons en bas de notre écran lorsque le joueur appui dessus ils changent indépendamment de

couleur avec la fonction changeColor() (Swift et Java). Lorsque notre choix de couleurs est fait, l'on appui sur le bouton valider appelant la fonction **jouerManche()** qui va :

— Remplir la n-ième ligne correspondant à la n-ième manche (valeur de la variable manche) avec les 4 couleurs choisi par le joueur.

— Arbitrer la manche avec la fonction **arbitrageManche()** qui va comparer les couleurs que le joueur a entré avec les 4 couleurs de la combinaison et changer la couleur des boutons d'arbitrage. Si le pion n a la même couleur que la combinaison n c'est qu'il est bien placé donc ça sera noir, si le pion est inclus dans la combinaison, mais n'est pas au bon indice ça sera rouge et enfin s'il n'est pas dans la combinaison ça sera blanc.

— Vérifier si la manche est perdue ou gagnée, si on perd on revient au menu principal.

— Passer à la manche suivante en incrémentant la variable *manche*.

2.2.3 Fin de la partie

Lorsque l'on gagne (réussir à déterminer la combinaison aléatoire) on passe à l'écran "gagné" (Voir 4). On le fait par l'intermédiaire d'un **Intent/Segue** mais cette fois-ci, on envoie une ou des données à l'activité/vue suivante . Ici, on va envoyer le temps que l'on a mis pour gagner : En **Swift** on fait le changement de vue avec la fonction **perfomSegue()** qui prend en paramètre l'identifiant du segue. On envoie la donnée avec la fonction **prepare()** qui va pouvoir modifier une variable dans la Vue de destination ici **ScoreController**.

```
if (isWin()){

     pause()

     performSegue(withIdentifier: "segue1", sender: nil) //utilise la segue "segue1"


    }


override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
//modifie la variable temps de ScoreController

   if segue.identifier == "segue1" {


   let dest  = segue.destination as! ScoreController
   dest.temps = counterTimer.text!

        }
```

En **Java** on le fait en utilisant un **Intent** qui prend en paramètre l'activité de destination, et utilise la méthode **putExtra(String,String)** pour envoyer une chaîne de caractères avec un certain identifiant.

```
if (isWin()){ //si c'est gagné on passe à l'activité ScoreActivity
```

```
    Intent scoreActivity = new Intent(this ,ScoreActivity.class);
    scoreActivity.putExtra("scoreChrono" , scoreChrono.getText());
    startActivity(scoreActivity);

       }
```

2.3 Écran Gagné

Nous avons passé la valeur de notre score (Notre temps) à notre Activité/Vue (ScoreController.swift et ScoreActivity.java) . On la récupère pour qu'elle soit affichée dans un **TextView/UILabel**. En Swift, on a directement modifié la valeur de cette variable dans notre fonction **prepare()** (Voir 2.2.3). En Java, on récupère la valeur passée dans l'activité précédente en créant un objet **Intent** qui prendra la valeur *getIntent()* puis en utilisant la méthode *getStringExtra()* en précisant l'identifiant.

On entre notre nom dans un EditText/UITextField pour ensuite l'envoyer dans notre prochaine Activité/Vue avec un Intent et une Segue lorsque l'on appui sur le bouton valider comme précédemment.

Swift :

```
@IBAction func valider(_ sender: Any) {

performSegue(withIdentifier: "segue2", sender: nil)

  }

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    if segue.identifier == "segue2"{

        let dest2 = segue.destination as! ListScoreController

        dest2.nom = nom.text!
        dest2.score = temps

        }
    }
```

Java :

```
public void validerScore(View v){

Intent listScoreActivity = new Intent(this ,ListScoreActivity.class);
    listScoreActivity.putExtra("chronoText" , chronoText);
    listScoreActivity.putExtra("nomScore" , nomScore.getText().toString());
    startActivity(listScoreActivity);
}
```

2.4 Liste des scores

La liste des scores est gérée par ListScoreActivity.java et ListScoreController.swift. On utilise une UITableView en swift et une ListView en java qui va manipuler une liste pour l'adapter.

On récupère encore une fois le nom et le score de l'écran précédent que l'on va ensuite rajouter à notre liste. On utilise en Swift les fonctions de "UITableViewDataSource" qui gèrent notre liste pour l'adapter à la UITableView et en Java nous avons une classe **ScoreArrayAdapter** qui hérite d'ArrayAdapter et c'est lui qui va prendre notre liste de score (Ici la liste prend un objet Score (nom + temps)) pour l'adapter à notre **ListView**.

2.4.1 Cas de la persistance de données

Normalement, lorsque l'on envoie des scores à notre liste des scores la fermeture de l'application entraîne la perte des données. Nous allons alors effectuer une **persistance des données** pour que les données (ici les éléments de notre liste) restent même après la fermeture de l'application.

Nous ne parlerons que de la persistance des données sous Java.

Pour sauvegarder notre liste des scores nous utilisons une **base de données**, en effet chaque fois que l'on envoie le nom et le score à notre activité, celle-ci va les ajouter dans la base et remplir notre liste de **Score** en allant récupérer chaque élément. On utilise la classe **DataBaseHelper** qui hérite de SQLiteOpenHelper, c'est elle qui va créer la base de données de nom *score.db* avec une table *scoreData*. Elle dispose de 3 colonnes un identifiant, le nom et le temps. Deux méthodes sont importantes, **addData(String nom,String score)** qui va rajouter le nom et le score dans la base et **getListContents()** qui va récupérer toutes les données via un **Cursor** en faisant un **SELECT * FROM scoreData**.
Dans notre activité, on instancie un **DataBaseHelper** où l'on rajoutera notre score et le nom en utilisant la méthode addData(). Puis on parcourt toute la base en instanciant un **Cursor** qui s'arrêtera quand il ne trouvera plus de données, à chaque fois, on rajoute ces éléments dans notre liste qui sera utilisée par notre **ScoreArrayAdapter**.

Ajout de la base dans la ListView :

```
scoreDB = new DatabaseHelper(this);
...

Cursor data = scoreDB.getListContents();
addData(chronoText,nomScore);
scores = new ArrayList<Score>();

if (data.getCount() != 0){

/*Parcours toute la base de donnée pour rajouter chaque élément dans notre liste
utilisable par l'ArrayAdapter*/

    while(data.moveToNext()){
     scores.add(new Score(data.getString(1) , data.getString(2)));
        adapter = new ScoreArrayAdapter(ListScoreActivity.this , scores);
        listViewScore.setAdapter(adapter);
}

}
```

3 Points délicats et améliorations possibles

3.1 Points délicats

Durant l'élaboration de ce projet il y a bien sûr eu des soucis qui ont pu ralentir ou compliquer son avancée :

— Le stockage des boutons dans un tableau à deux entrées était moins intuitif pour Swift car il faut déclarer celui-ci comme en Python en faisant une liste de listes.

— Le choix d'un tableau de boutons était plus abordable (alors que nos deux grilles de boutons n'ont pas d'interactions directes avec le joueur) car c'était plus adaptable dans les deux langages.
— Il était compliqué en Android de récupérer la couleur de fond d'un bouton (comparé à swift) car l'objet **Button** avec notre Drawable pion.xml ne dispose pas de méthode *getBackgroundColor()*. On est obligés de créer une fonction setColor() pour ça.
— De même pour changer la couleur de nos boutons on est obligé de récupérer le pion.xml en **GradiantDRawable** puis d'appliquer la méthode setBackgroundDrawable() sur notre bouton. Cette action est effectuée par la fonction **setColorPion()**.

```
private void setColorPion(Button b , int backgroundColor){

GradientDrawable pion=(GradientDrawable) getResources().getDrawable(R.drawable.pion).mutate();
pion.setColor(backgroundColor);
b.setBackgroundDrawable(pion);
b.setTextColor(backgroundColor);

    }
```

— Pour le score, on utilise un chronomètre que l'on peut démarrer et arrêter, sous **Android Studio** il existe un widget *Chronometer* que l'on peut intégrer dans le code, démarrer, arrêter et récupérer le temps en chaîne de caractère. Sous XCode il n'existe pas de widget chronometer il a fallu alors directement le programmer en changeant la valeur d'un **UILabel**.

3.2 Améliorations possibles du programme

— Le jeu se joue à un seul joueur pour le moment, on pourrait rajouter un mode deux joueurs où au lieu de définir la combinaison aléatoirement un autre joueur le ferai et on passerait notre combinaison de couleurs a notre activité/vue via un Intent/Segue.
— Pour choisir ses pions il faut appuyer sur un des boutons où va défiler l'ensemble des couleurs. C'est une petite perte de temps, on pourrait directement choisir la couleur que l'on veut en faisant un espace dans l'écran où toutes les couleurs seraient disponible, au lieu de toutes les parcourir à chaque fois .

4 Conclusion

Ce rapport explique tout ce qu'il y a à savoir sur le fonctionnement de l'application Mastermind nous avons d'abord rappelé le principe général du Mastermind puis ce que faisait notre application en montrant les différents écrans :

— Le menu où l'on choisit vers quel écran on veut se diriger (Jouer/Règles/Liste des scores)
— Le jeu Mastermind où se déroule l'ensemble du jeu.
— L'écran gagner qui indique en combien de temps l'on a gagné et où l'on va rentrer notre nom.
— Et enfin la liste des scores où sont affichés le nom du joueur et le temps qu'il a mis pour gagner.

Nous avons décrit ensuite plus en détails la conception même de l'application en regardant dans les deux langages comment elle était programmée. (Passage d'une vue/activité à une autre, fonctionnement du jeu, passage de donnée, liste des scores, persistance des données).

Références

[1] addsubview - uiview | apple developer documentation. https://developer.apple.com/documentation/uikit/uiview/1622616-addsubview.

[2] Android developers. https://developer.android.com/docs.

[3] Apple developer documentation. https://developer.apple.com/documentation.

[4] Chronometer | android developers. https://developer.android.com/reference/android/widget/Chronometer/.

[5] Save data using sqlite. https://developer.android.com/training/data-storage/sqlite.

[6] Start another activity | android developers. https://developer.android.com/training/basics/firstapp/starting-activity.

[7] How to make a round/circle button in android. https://android--code.blogspot.com/2015/01/android-round-button.html.

[8] How to set corner radius for uibuttons using swift. https://peterwitham.com/swift-archives/how-to-set-corner-radius-for-buttons-using-swift.

[9] Pass data between view controllers in swift - learnappmaking.com. https://learnappmaking.com/pass-data-between-view-controllers-swift-how-to/.

[10] How to change color of drawable shapes in android - stack overflow. https://stackoverflow.com/questions/15127351/how-to-change-color-of-drawable-shapes-in-android.

[11] Swift.org - documentation. https://swift.org/documentation.

[12] Tablelayout | android developers. https://developer.android.com/reference/android/widget/TableLayout.

[13] Create android tablelayout programatically. http://puneetaggarwal.in/android/android-tablelayout-programatically.

[14] Uibutton - uikit | apple developer documentation. https://developer.apple.com/documentation/uikit/uibutton.

[15] Android list view - tutorialspoint. https://www.tutorialspoint.com/android/android_list_view.html.

[16] Uitableview - uikit | apple developer documentation. https://developer.apple.com/documentation/uikit/uitableview.