Advanced Mobile Application Development
Week 3: Table Views

We're going to create a simple table view to see how they work.

scrabbleQ
File | New Project
Single View App
iPhone

Go into MainStoryboard. The initial scene is a view controller but we want a table view controller.
Click on the scene and delete it. Then drag onto the canvas a table view controller.
In the attributes inspector check Is Initial View Controller.

We want our class to be the controller so go into ViewController.swift and change its super class to `UITableViewController`.
Now go back into MainStoryboard and select the view controller and change its class to ViewController.
Select the table view and note that it has the class UITableView.
In the Connections inspector check that the dataSource and delegate for the table view are set to View Controller. If not, drag from the circles to the View Controller icon.
In the attributes inspector see that the table view's style is plain. Try changing it to grouped and see how that looks.
Select a table view cell and in the attributes inspector you can see that the Table View Cell style is custom. We'll look at the others in a minute.
Select the Table View Cell and in the attributes inspector make the identifier "CellIdentifier".

Build and run and you should see a blank table view.

Add qwordswithoutu1.plist into your project and make sure Copy Items if Needed is checked as well as your project target.

Go into ViewController.swift and add an array that will hold the letters and words. This makes it an implicitly unwrapped optional. That means it's an optional so it can be nil but whenever it's accessed it will always have a value.

```
    var words : [String]!
```

Now let's get the data loaded

```
    override func viewDidLoad() {
        super.viewDidLoad()
        //use a NSBundle object of the directory for our application to
retrieve the pathname of qwordswithoutu1.plist
        let path = NSBundle.mainBundle().pathForResource("qwordswithoutu1",
ofType: "plist")
        //load the words of the plist file into the array
        words = NSArray(contentsOfFile: path!) as! Array
        print(words)
    }
```

Now we implement the required methods for the UITableViewDataSource protocol

```
//Required methods for UITableViewDataSource
// Customize the number of rows in the section
    override func tableView(tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        return words.count
    }

// Displays table view cells
    override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
        //configure the cell
        let cell =
tableView.dequeueReusableCellWithIdentifier("CellIdentifier", forIndexPath:
indexPath)
        //set the text of the cell
        cell.textLabel?.text=words[indexPath.row]
        return cell
    }
```

You should now see your table view with the list of words.
To fix the fact that the table scrolls under the status bar
Add to viewDidLoad()

```
self.tableView.contentInset = UIEdgeInsetsMake(20, 0, 0, 0);
```

Looks good initially but it still scrolls under the status bar
Can't seem to set constraints on the table view.
But table views are usually in navigation controllers and that will fix the problem

Now let's add an image.
Drag scrabble_q_tile.png into Assets.xcassets
In MainStoryboard select the table view cell and change the style to basic and under image choose scrabble_q_tile.png.
You can also do the image programmatically

```
cell.imageView?.image=UIImage(named: "scrabbletile90.png")
```

Now when you run it you'll see the image.

Now change the Table View Cell style to subtitle.
Notice this adds a detail label.
Make its text say Q no U. (You can either add it in the label in the storyboard or do it programmatically)

```
cell.detailTextLabel?.text="Q no U"
```

Now when you run it you'll see a subtitle as well.
(Snapshot: Table cell subtitle)

Now what if you want to do something when the user selects a row. It's a UITableViewDelegate method that handles this.

```
    //UITableViewDelegate method that is called when a row is selected
    override func tableView(tableView: UITableView, didSelectRowAtIndexPath
indexPath: NSIndexPath) {
        let alert = UIAlertController(title: "Row selected", message: "You
selected \(words[indexPath.row])", preferredStyle:
UIAlertControllerStyle.Alert)
```

```
        let okAction = UIAlertAction(title: "OK", style:
UIAlertActionStyle.Default, handler:nil)
        alert.addAction(okAction)
        presentViewController(alert, animated: true, completion: nil)
        //deselects the row that had been choosen
        tableView.deselectRowAtIndexPath(indexPath, animated: false)
    }
```

scrabbleQgrouped
Now let's look at the table view grouped style.
You can create a new project(follow steps from first one) or change the one we've been working on.
In the attributes inspector change the table view's style to Grouped.
Add qwordswithoutu2.plist into your project and make sure Copy items if needed is checked

Go into ViewController.swift and create a Dictionary for the words and an Array for the letters

```
    var allwords : [String : [String]]!
    var letters : [String]!
```

(this will break some of the code you wrote, but we're about to fix it)

Change viewDidLoad

```
    override func viewDidLoad() {
       super.viewDidLoad()
        //use a NSBundle object of the directory for our application to
retrieve the pathname of qwordswithoutu1.plist
        let path = NSBundle.mainBundle().pathForResource("qwordswithoutu2",
ofType: "plist")
        //load the words of the plist file into the dictionary
        allwords = NSDictionary(contentsOfFile: path!) as! [String :
[String]]
        //puts all the letters in an array
        letters = Array(allwords.keys)
        // sorts the array
        letters.sortInPlace({$0 < $1})
    }
```

The Array class has a method called sortInPlace that sorts the array and assigns it back to itself. It takes a closure for how to do the sort. This is the shorthand way to sort Strings in ascending order. For more complex sorts you can write a function.

Change the following methods:

```
    override func tableView(tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        let letter = letters[section]
        let letterSection = allwords[letter]! as [String]
        return letterSection.count
    }

override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
        let section = indexPath.section
        let letter = letters[section]
        let wordsSection = allwords[letter]! as [String]
        //configure the cell
        let cell =
tableView.dequeueReusableCellWithIdentifier("CellIdentifier", forIndexPath:
indexPath) as UITableViewCell
        //set the text of the cell
        cell.textLabel?.text=wordsSection[indexPath.row]
        return cell
    }

    //UITableViewDelegate method that is called when a row is selected
    override func tableView(tableView: UITableView, didSelectRowAtIndexPath
indexPath: NSIndexPath) {
        let section = indexPath.section
        let letter = letters[section]
        let wordsSection = allwords[letter]! as [String]
        let alert = UIAlertController(title: "Row selected", message: "You
selected \(wordsSection[indexPath.row])", preferredStyle:
UIAlertControllerStyle.Alert)
        let okaction = UIAlertAction(title: "OK", style:
UIAlertActionStyle.Default, handler:nil)
        alert.addAction(okaction)
        presentViewController(alert, animated: true, completion: nil)
        //deselects the row that had been choosen
        tableView.deselectRowAtIndexPath(indexPath, animated: false)
    }
```

Add this method to handle the number of sections. The default is 1 so we didn't need to implement it earlier.

```
//UITableViewDatasource methods
    override func numberOfSectionsInTableView(tableView: UITableView) -> Int
{
        return letters.count
    }
```

And add this one to have headers.

```
//Sets the header value for each section
    override func tableView(tableView: UITableView, titleForHeaderInSection
section: Int) -> String? {
        return letters[section]
    }
```

Let's add an index down the right side of the table.

```
//adds a section index
      override func sectionIndexTitlesForTableView(tableView: UITableView)
-> [AnyObject]! {
        return letters
    }
```

We only have 4 different sections so it doesn't look great, but this is especially helpful for really long lists.

search
Let's add the ability to search our table view by creating a new view controller class to handle search and its results.

File | New File
iOS | Cocoa Touch class
SearchResultsController
Subclass UITableViewController

Go into your new file and adopt the UISearchResultsUpdating protocol
```
class SearchResultsController: UITableViewController,
UISearchResultsUpdating
```

SearchResultsController needs access to the list of words that the main view controller is displaying, so we'll need to give it properties that we can use to pass to it the allwords dictionary and the list of keys that we're using for display in the main view controller. We also need an array to store the results of a search.
```
    var allwords = [String : [String]]()
    var letters = [String]()
    var filteredWords = [String]()
```

The file already contains some template code that provides a partial implementation of the UITableViewDataSource protocol and some commented-out methods UITableViewController subclasses often need. We're not going to use most of them so you can delete them if you want.

Since we won't have a scene for this view controller in our storyboard we need to register our cell reuse identifier programmatically. Add to viewDidLoad()
```
//register our table cell identifier
tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier:
"CellIdentifier")
```

The `UISearchResultsUpdating` protocol only has 1 method and it's required

```
    //UISearchResultsUpdating protocol required method to implement the
search
    func updateSearchResultsForSearchController(searchController:
UISearchController) {
        let searchString = searchController.searchBar.text //search string
        filteredWords.removeAll(keepCapacity: true) //removes all elements
        if searchString?.isEmpty == false {
//closure that will be called for each word to see if it matches the search
string
            let filter: String -> Bool = { name in
                //look for the search string as a substring of the word
                let range = name.rangeOfString(searchString!, options:
NSStringCompareOptions.CaseInsensitiveSearch)
                return range != nil //returns true if the value matches and
false if there's no match
            }
            //iterate over all the letters
            for key in letters {
                let wordsForKeys = allwords[key]! //array of names for each
key
                let matches = wordsForKeys.filter(filter) //filter using the
closure
                filteredWords.appendContentsOf(matches) //add words that
match
            }
        }
        tableView.reloadData() //reload table data with search results
    }
```

Then we need to implement the UITableViewDataSource methods to display the table view cells. We want the table rows to show the search results. Since SearchResultsController is a subclass of UITableViewController it automatically acts as the table's data source.

```
    override func tableView(tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        return filteredWords.count
    }


    override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
        let cell =
tableView.dequeueReusableCellWithIdentifier("CellIdentifier", forIndexPath:
indexPath)
        cell.textLabel?.text = filteredWords[indexPath.row]
        return cell
    }
```

Now we have to set up our main ViewController to add the search bar.
In ViewController.swift add an instance of UISearchController

```
    var searchController : UISearchController!
```

Update viewDidLoad() to implement and configure the search bar.

```
        //search results
        let resultsController = SearchResultsController() //create an
instance of our SearchResultsController class
        resultsController.allwords = allwords
        resultsController.letters = letters
        searchController = UISearchController(searchResultsController:
resultsController) //create a search controller and initialize with our
SearchResultsController instance

        //search bar configuration
        searchController.searchBar.placeholder = "Enter a search term"
//place holder text
        searchController.searchBar.sizeToFit() //sets appropriate size for
the search bar
```

```
        tableView.tableHeaderView=searchController.searchBar //install the
search bar as the table header
        searchController.searchResultsUpdater = resultsController //sets the
instance to update search results
```

Each time the user types something into the search bar, UISearchController uses the object stored in its searchResultsUpdater property to update the search results.

Now you should be able to search through the data in your table view. Very useful for tables with a lot of data.

Don’t forget your app icons and launch screen.