Alfonso López Ruiz

alr00048@red.ujaen.es



**Realización de búsquedas y filtrados**

**Minitutorial 8**

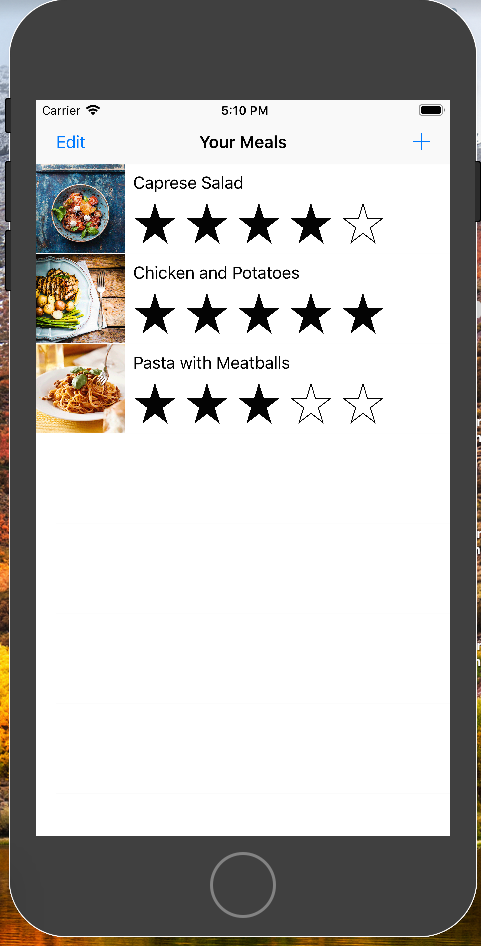
En este minitutorial se pretende englobar los pasos necesarios para añadir una barra de búsqueda en la aplicación base de FoodTracker. Dicha barra nos deberá permitir filtrar todas las comidas guardadas en función de su nombre, de tal forma que sólo se muestren aquellas que coinciden con el texto introducido. Adicionalmente, también se podrá elegir si filtrar por nombre o por puntuación.

# Fundamentos teóricos

Los fundamentos teóricos necesarios para la correcta comprensión de este minitutorial son los siguientes:

# Implementación de la barra de búsqueda

El punto de partida para nuestra aplicación será, como ya sabemos, la aplicación FoodTracker de las que nos provee Apple (<https://developer.apple.com/sample-code/swift/downloads/09_PersistData.zip>). En este momento la pantalla de nuestra aplicación debería quedar como sigue:

La pantalla que tendremos que modificar para incluir nuestra barra de búsqueda será la correspondiente a la tabla de comidas, la cual es gestionada por el archivo **MealTableViewController.swift**.

El elemento que vamos a introducir forma parte también de la navegación que ya está incluida, por lo que nuestro primero paso será modificar dicha parte de la vista en el método viewDidLoad de la clase ya mencionada.

Figura 1. Apariencia de la aplicación base.

Nuestro primer paso será incluir el controlador de la barra de búsqueda (**UISearchController**) como un atributo en nuestro archivo MealTableViewController:

let searchController = UISearchController(searchResultsController: nil)

Como argumento recibirá la vista que se encarga de mostrar el resultado de la búsqueda. En nuestro caso, la vista donde se busca y donde se muestran los resultado es la misma, por lo que utilizaremos *nil*.

También necesitaremos que nuestra clase implemente el protocolo UISearchResultsUpdating para poder responder a los eventos que suceden en la barra de búsqueda. El único método que tendremos que implementar es el siguiente:

class MealTableViewController: UITableViewController, UISearchResultsUpdating {

func updateSearchResults(for searchController: UISearchController) {

}

}

Será al iniciarse nuestra tabla de comidas (**viewDidLoad**) donde tengamos que incorporar el controlador de la barra de búsqueda, para lo cual tendremos que utilizar el siguiente bloque de código:

searchController.searchResultsUpdater = self

searchController.obscuresBackgroundDuringPresentation = false

searchController.hidesNavigationBarDuringPresentation = false

searchController.searchBar.placeholder = "Search meal"

navigationItem.searchController = searchController

navigationItem.hidesSearchBarWhenScrolling = false

definesPresentationContext = true

* **SearchResultsUpdater**: indica la clase que recibirá información cuando se modifique el contenido de la barra de búsqueda. Esta propiedad se deriva del protocolo **UISearchResultsUpdating**.
* **ObscuresBackgroundDuringPresentation**: es útil en el caso de que las vistas donde se busca y donde se muestran los resultados son diferentes. En nuestro ejemplo, esto no es así y por lo tanto usaremos *false*.
* **HidesNavigationBarDuringPresentation**: por defecto será *true*, y de ser así, al buscar (utilizar la barra de búsqueda) desaparecería la barra de navegación (título, botón de editar y de añadir). En una aplicación donde la barra de navegación sólo mostrara un título podría ser interesante ponerla a verdadera para tener más espacio. En nuestro caso, permitiremos que se editen comidas que aparecen como resultado de una búsqueda, y también añadir, por lo que no podemos ocultar esta barra.
* **Placeholder**: será el texto que se muestre en la barra de búsqueda cuando esté vacía. Hemos elegido la cadena ‘Search meal’ pero también podríamos elegir una cadena a modo de ejemplo de lo que podemos buscar: ‘Caprese Salad’.
* **Search controller**: como ya comentamos, la barra de búsqueda formará parte de la navegación, por lo que habrá que especificar en navigationItem cuál es nuestro controlador de búsqueda.
* **HidesSearchBarWhenScrolling**: lo pondremos a false ya que de no ser así la barra de búsqueda no saldría desde un comienzo. Es una forma de dejarla permanente en la vista.
* **DefinesPresentationContext**: evita que la barra de búsqueda aparezca incluso cuando cambiamos a otras vistas.

Como ya sabemos, guardamos un vector de comidas que representa la totalidad de las que disponemos. Sin embargo, el resultado de la búsqueda será un subconjunto de éstas, por lo que será necesario otro vector que guarde dicho subconjunto: filteredMeals.

Comenzamos a implementar métodos que nos servirán de ayuda en la búsqueda:

private func searchBarIsEmpty() -> Bool {

return searchController.searchBar.text?.isEmpty ?? true

}

private func filterContentForSearchText(\_ searchText: String, scope: String = "Name") {

filteredMeals = meals.filter({(meal: Meal) -> Bool in

return meal.name.lowercased().contains(searchText.lowercased()) })

tableView.reloadData()

}

func isFiltering() -> Bool {

return searchController.isActive && !searchBarIsEmpty()

}

Es de especial importancia el segundo método, el cual podemos personalizar como deseemos. Actualiza el vector de resultados de la búsqueda y determina mediante una función definida inline qué elementos y cuales no forman parte del vector de resultados.

Será una función que devuelve un booleano, y en nuestro caso transformamos tanto el nombre de la comida como el texto de la barra a minúsculas. Devolveremos true si el nombre de la comida contiene el texto introducido por el usuario.

Por ejemplo, ‘Cap’ o ‘Sal’ devolvería ‘Caprese Salad’.

El argumento scope determinará, por ejemplo, el campo por el que estamos buscando: Nombre, clasificación… Para la implementación básica no es necesario conocer para qué es esta variable, pero sí que aparecerá en el último apartado opcional.

Por último, necesitamos un método para determinar cuándo está activa la búsqueda, ya que nos ayudará a la hora de saber qué vector utilizar, meals o filteredMeals. Lo veremos más tarde.

**¿Dónde usaremos este filtro?** Se usará en la función updateSearchResults que antes dejamos vacía.

func updateSearchResults(for searchController: UISearchController) {

filterContentForSearchText(searchController.searchBar.text!)

}

Podemos hacer una pausa y ver como quedaría nuestra aplicación en este momento:

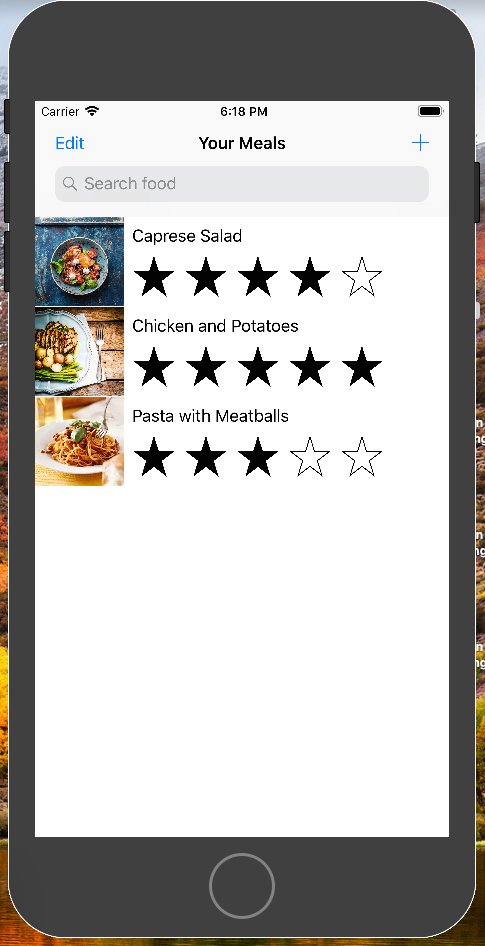


Figura 2. Apariencia de nuestra aplicación con la barra de búsqueda.

Llegados a este paso tenemos nuestra barra de búsqueda pero ésta no es funcional ya que sólo estamos utilizando los datos del vector meals, aunque sí se está actualizando el vector filteredMeals.

Primero gestionaremos cómo mostrar los resultados, y después necesitaremos modificar la comunicación con otras vistas.

Modificaremos los siguientes métodos, ya implementados en el proyecto base, como sigue:

override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

if isFiltering() {

return filteredMeals.count

}

return meals.count

}

Primero tenemos que controlar que número de filas se van a mostrar. Dependiendo de si la búsqueda está activa se extraerá este número de un vector u otro.

override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let cellIdentifier = "MealTableViewCell"

guard let cell = tableView.dequeueReusableCell(withIdentifier: cellIdentifier, for: indexPath) as? MealTableViewCell else {

fatalError("The dequeued cell is not an instance of MealTableViewCell.")

}

let meal : Meal

if isFiltering() {

meal = filteredMeals[indexPath.row]

} else {

meal = meals[indexPath.row]

}

cell.nameLabel.text = meal.name

cell.photoImageView.image = meal.photo

cell.ratingControl.rating = meal.rating

return cell

}

Cada una de las filas adoptará la información de una comida, por lo que tendremos que saber de dónde extraer (de qué vector) las comidas. De nuevo, es la misma situación: de dónde extraer la información.

Por último nos queda la eliminación de filas:

override func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

if editingStyle == .delete {

if isFiltering() {

filteredMeals.remove(at: indexPath.row)

if let index = meals.index(of: filteredMeals[indexPath.row]) {

meals.remove(at: index)

}

} else {

meals.remove(at: indexPath.row)

}

saveMeals()

tableView.deleteRows(at: [indexPath], with: .fade)

} else if editingStyle == .insert {

// Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view

}

}

Habrá que distinguir dos situaciones muy diferentes:

* **La búsqueda está activa**, luego el índice que nos devuelve el evento es del vector filteredMeals, por lo que para actualizar el vector meals habrá que hacer una operación un poco más compleja que en el siguiente caso.
* **La búsqueda no está activa**, se actualiza el vector principal meals.

Llegados a este punto deberíamos ser capaces de buscar y filtrar nuestras comidas, como en la siguiente pantalla:

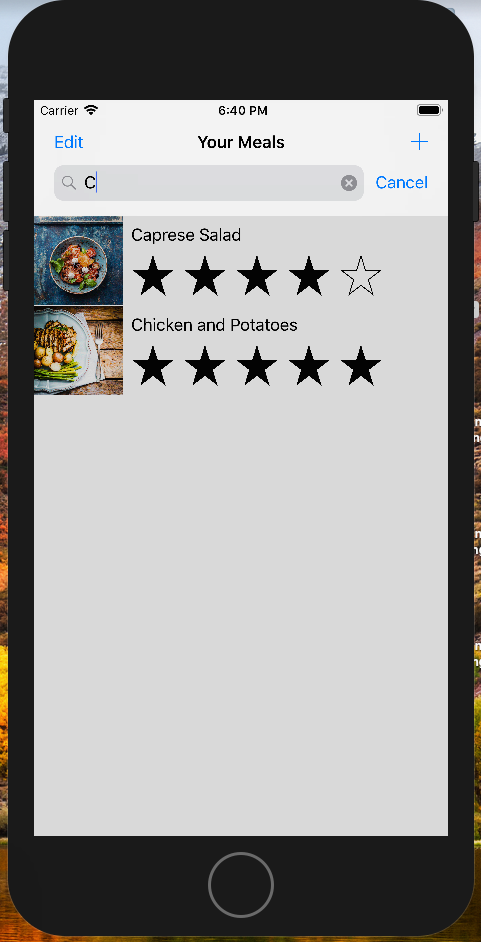


Figura 3. Filtrado de comidas en nuestra aplicación.

Nuestro último paso será adoptar un comportamiento correcto para comunicarnos con las vistas de añadir y actualizar.

Muy importante. Nuestro enfoque es el siguiente: permitiremos añadir y actualizar comidas una vez se han mostrado resultados de la búsqueda, teniendo en cuenta que la modificación o añadir nuevas comidas puede hacer que se modifique la lista de resultados obtenidos.

Otro enfoque: no permitir añadir pero sí editar (que puede ser más normal). No lo hemos implementado pero como vistazo rápido a este enfoque podemos decir que se pueden habilitar y deshabilitar los botones de la barra de navegación:

navigationItem.rightBarButtonItem.isEnabled = true / false

Además, podemos deshabilitar el botón cuando se llame al método updateSearchResults, al cual se acude siempre que haya cambios en la barra, incluyendo los eventos de limpiar y cancelar. Por lo que si el texto de la barra está vacía podemos habilitar de nuevo el botón.