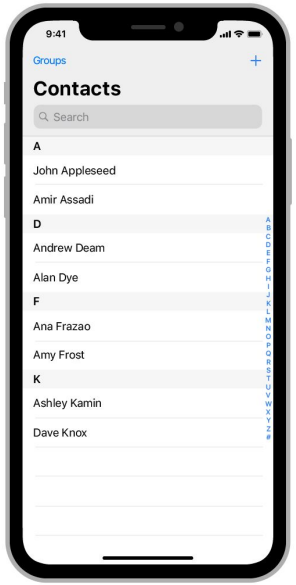
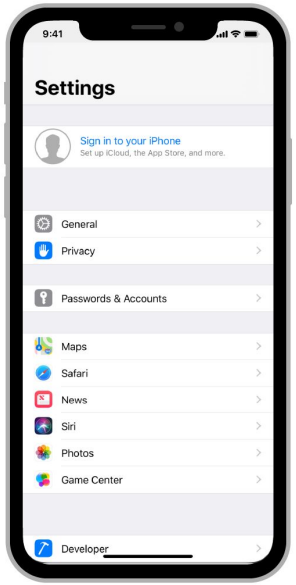
UITableView

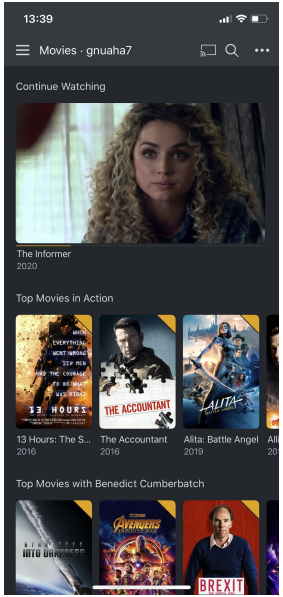
Sergio Padrino Recio

UITableView

Las tablas son uno de los elementos más versátiles en iOS







UITableView

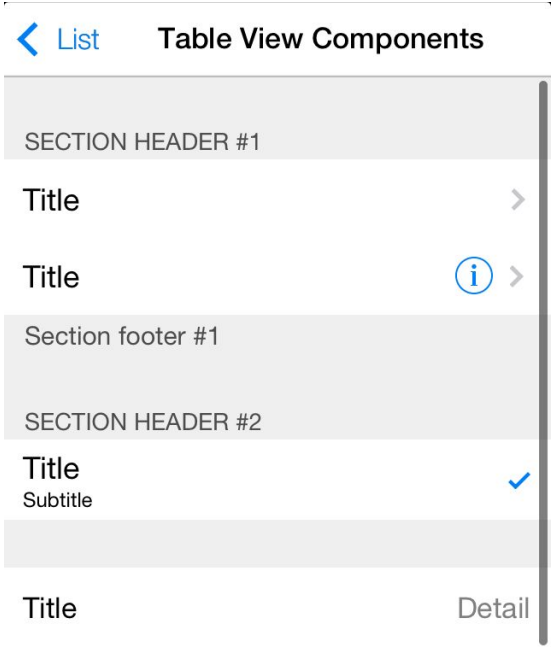
- Generar listas infinitas es muy costoso si generamos los elementos de las listas de antemano o a medida que hacemos scroll.

- En su lugar, `UITableView` reutiliza las los elementos que ya no son visibles, generando nuevos solo cuando es estrictamente necesario.

UITableView

Una tabla se compone de:

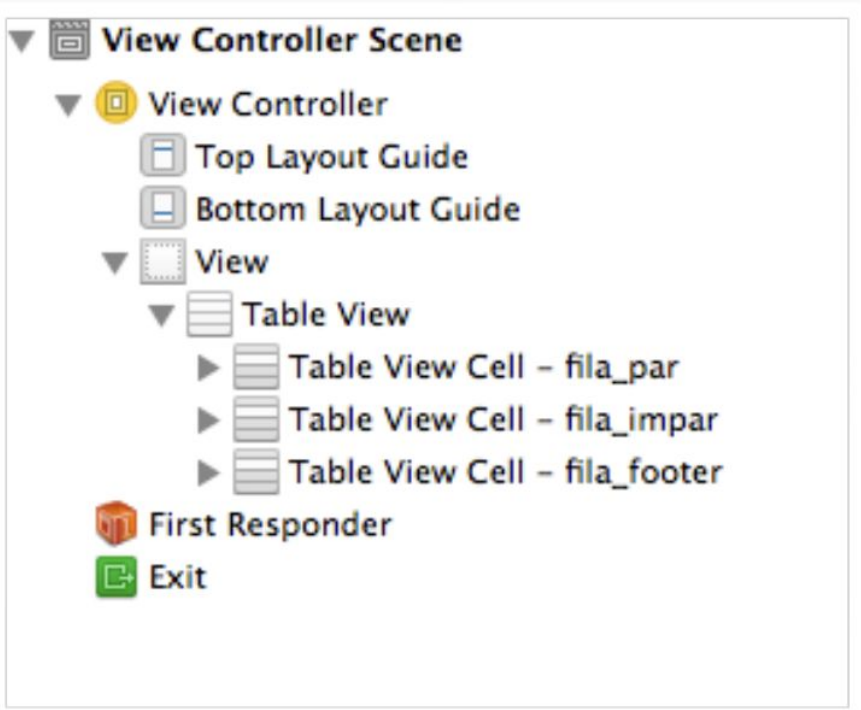
- Cabecera
- Pie
- Secciones (grupos)
  - Pueden tener su propia cabecera y pie
- Celdas (filas)



UITableView

- Añadir un UITableView en una vista.

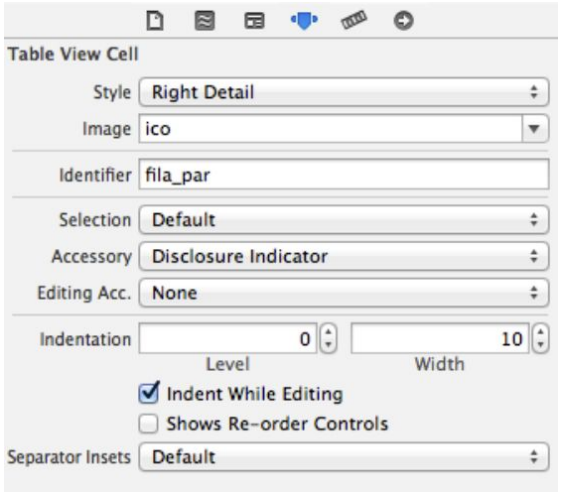
- Añadir los UITableViewCell dentro de la tabla. Uno por cada tipo de fila distinta a mostrar.



UITableView

En cada celda, configuramos:

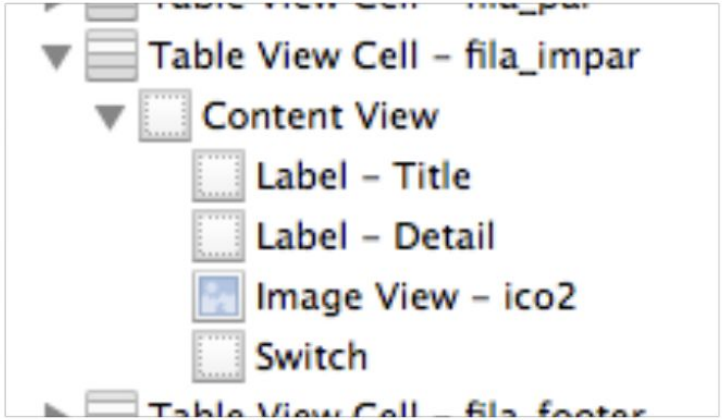
- Style
- Imagen (opcional)
- Identifier
- Accessory
- Row Height (opcional)



UITableView

- Dentro de cada celda, existe una vista llamada “Content View”

- Podemos introducir ahí cualquier otro componente gráfico (Celdas custom)



UITableView

Las tablas utilizan dos delegates diferentes:

- **UITableViewDataSource**: permite a la tabla obtener los datos que deben mostrar en cada fila.

- **UITableViewDelegate**: permite a la tabla notificar cuando ocurren eventos sobre la tabla.

UITableViewDataSource: Métodos

```
func tableView(_ tableView: UITableView,
               numberOfRowsInSection section: Int) -> Int
```

Hay que retornar el número de filas a mostrar para el grupo/sección pasado por parámetro.

UITableViewDataSource: Métodos

```
func tableView(_ tableView: UITableView,
               cellForRowAt indexPath: IndexPath)
              -> UITableViewCell
```

Hay que retornar un objeto celda que con el contenido que queramos para la fila (`indexPath`) recibida como parámetro.

UITableViewDataSource

La forma habitual de trabajar con el data source es:

- Tener un atributo de tipo `Array` en el view controller que contenga la información a mostrar en la tabla.

- En el método `numberOfRowsInSection` retornar el número de elementos del array.

- El el método `cellForRowAtIndexPath` pedir al sistema una celda para un identificador dado y rellenarla con la información que queramos.

UITableViewDataSource: Ejemplo

```
func tableView(_ tableView: UITableView,
               numberOfRowsInSection section: Int) -> Int
{
    return self.datos.count
}
```

UITableViewDataSource: Ejemplo

```
func tableView(_ tableView: UITableView,
               cellForRowAt indexPath: IndexPath)-> UITableViewCell
{
    let identificador = (indexPath.row % 2 == 0)
                        ? "fila_par"
                        : "fila_impar";

    let cell = tableView.dequeueReusableCellWithIdentifier(identificador)
    cell.textLabel?.text = self.datos[indexPath.row]

    return cell
}
```

UITableView: Métodos

Métodos para obtener indexPaths y celdas:

- `cellForRowAtIndexPath`: devuelve el objeto UITableViewCell para el indexPath pasado

- `indexPathForCell`: retorna el indexPath correspondiente al objeto UITableViewCell

- `visibleCells`: retorna el array de celdas que están visibles

- `indexPathsForVisibleRows`: array de indexPaths visibles

UITableView: Métodos

Métodos para cargar datos:

- `reloadData`: carga todas las filas visibles, llamando al delegado.
- `reloadRowsAtIndexPaths`: carga las filas especificadas.
- `reloadSections`: carga las filas de las secciones especificadas.

UITableView: Métodos

Métodos para insertar/borrar celdas:

- `insertRowsAtIndexPaths`: inserta nuevas filas en los indexPaths especificados. Llama al delegado para obtener los datos

- `deleteRowsAtIndexPaths`: borra las filas especificadas

- `insertSections`: inserta un grupo de secciones

- `deleteSections`: borra un grupo de secciones

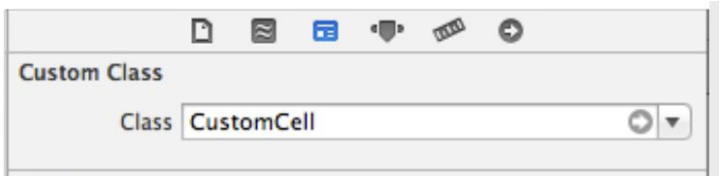
Celdas Custom

Cuando una celda tiene componentes personalizados necesitaremos código para gestionarlos:

- Hay que crear una clase hija de UITableViewCell

- Configurar el custom class del storyboard

- En esa clase pondremos todos los IBOutlet, IBAction y propiedades que necesitemos.



Celdas Custom

A la hora de hacer el dequeueReusableCellWithIdentifier podemos hacer casting a nuestra clase y llamar a las propiedades y métodos que queramos

```
func tableView(_ tableView: UITableView,
               cellForRowAt indexPath: IndexPath)
               -> UITableViewCell
{
    let cell =
        tableView.dequeueReusableCellWithIdentifier(identificador)
        as! MyCell
    cell.miOutlet = …
}
```

Apps Universales

Cuándo **NO** debemos hacer una app universal:

- Si queremos vender ambas versiones por separado: **más ingresos**.
- Si queremos distinto precio en cada versión.
- Si las version iPhone y iPad son completamente distintas
- Si quieres que las valoraciones de una y otra versión estén separadas: análisis de la audiencia.

Secciones

- Si queremos crear secciones, debemos implementar el método `numberOfSectionsInTableView` (para retornar el número de secciones) y `titleForHeaderInSection` (para retornar el título de la sección).

- Después, en los `numberOfRowsInSection` y `cellForRowAtIndexPath` debemos retornar los valores teniendo en cuenta tanto la sección como la fila del parámetro `indexPath`.

Cabeceras y Pies

- Podemos configurar cabeceras de los grupos con el método `viewForHeaderInSection` (para retornar la vista a utilizar) y `heightForHeaderInSection` (para retornar el alto de la cabecera).

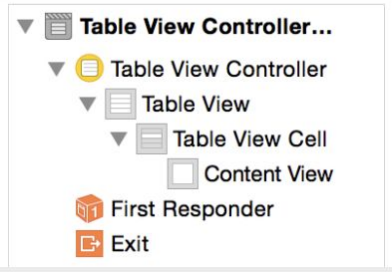
- Los footers se hacen de la misma forma con `viewForFooterInSection` y `heightForFooterInSection`

UITableViewController

- Es subclase directa de `UIViewController`

- Nos facilita algunas tarea típicas con tablas a pantalla completa.

- Debemos asegurarnos de que nuestra tabla es el único (o primer) componente de la vista.



Buenas prácticas

- Siempre reusar celdas. No crear instancias de `UITableCellView`. Usar `dequeueReusableCellWithIdentifier`.

- Cargar imágenes de forma asíncrona en un hilo en background para evitar bloquear el hilo principal.

Buenas prácticas

- Los métodos `cellForRowAtIndexPath` y `heightForRowAtIndexPath` deben retornar muy rápido y sin bloquear el hilo principal.

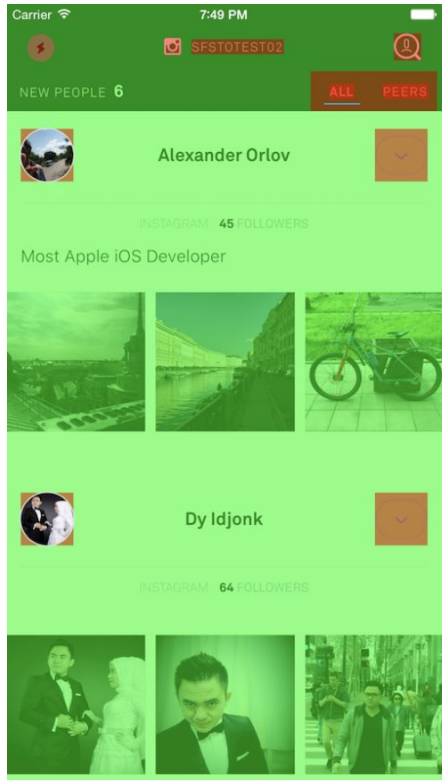
- Evitar tener muchas vistas y el uso de AutoLayout con muchas constraints en las celdas.

- Usar la propiedad `estimatedRowHeight`, y asignar el valor `UITableViewAutomaticDimension` la propiedad `rowHeight`. Es más sencillo pero puede afectar al rendimiento.

Buenas prácticas

Evitar Blending:

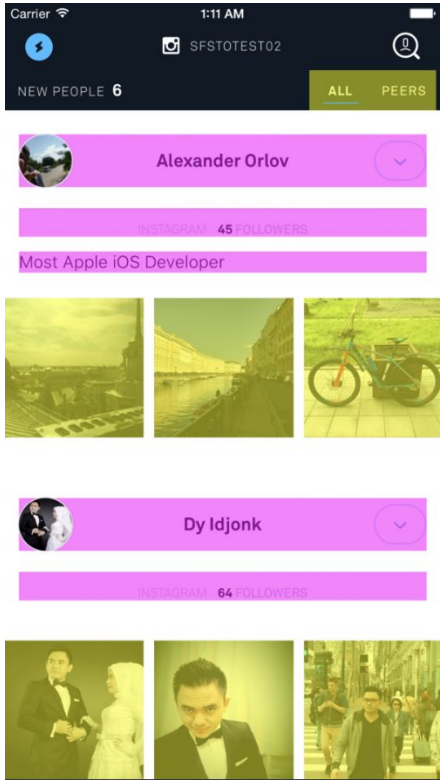
- Usar “Simulator > Debug > Color Blended Layers” para detectar blending
- Poner la propiedad `celda.opaque = true`
- No usar `backgroundColor` transparente o valores `alpha` distintos de 1



Buenas prácticas

Imágenes “Pixel-perfect”

- Usar “Simulador > Debug > Color Misaligned Images”
- Usar tamaños de imagen múltiplos de 2 o de 3.



Buenas prácticas

Offscreen rendering

- Usar “Simulador > Debug > Color Offscreen-Rendered”
- Para las que salen marcadas, evitar modificaciones como máscaras o bordes redondeados.
- Si nada funciona, activar `view.layer.drawAsynchronously = true`