



PROF. RENÊ XAVIER

**DESENVOLVIMENTO PARA IOS 11 COM
SWIFT 4**

ONDE ENCONTRAR O MATERIAL?

[HTTPS://GITHUB.COM/RENEFX/IOS-2018-01](https://github.com/renefx/ios-2018-01)

GITHUB
ALÉM DO TRADICIONAL BLACKBOARD DO IESB

O QUE VAMOS FAZER HOJE?

AGENDA

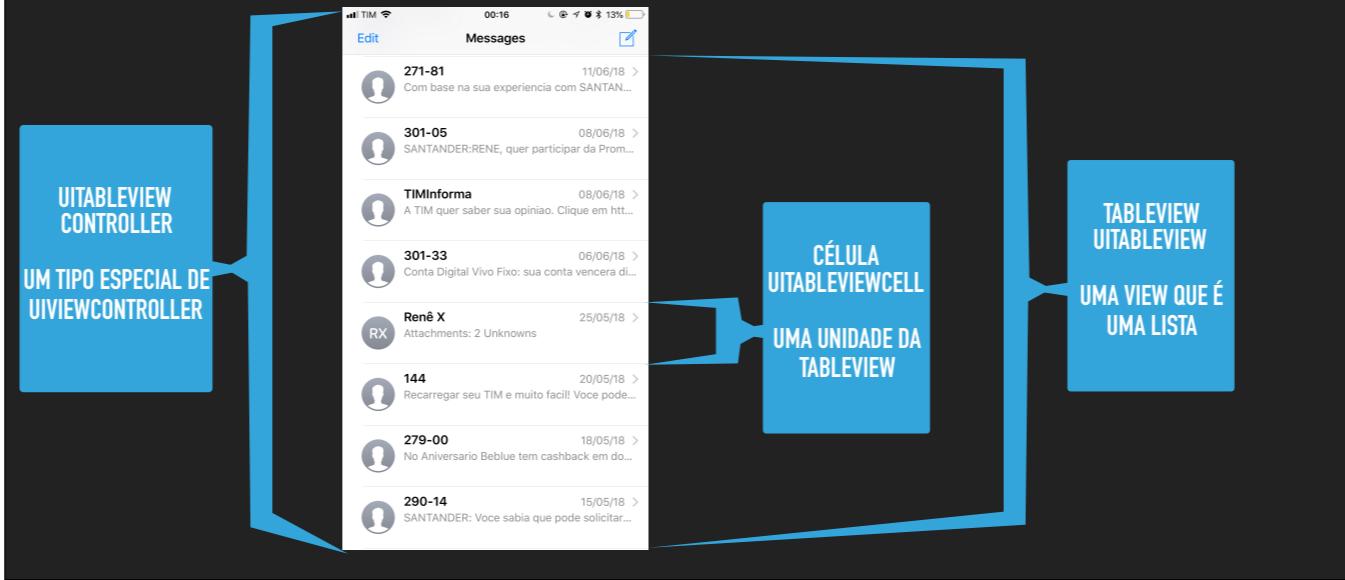
- ▶ UITableView e UITableViewController
- ▶ Células personalizadas
- ▶ Prática





REVIEW

TABLEVIEW CONTROLLER



TABLEVIEW CONTROLLER

► TableViewController

► TableView

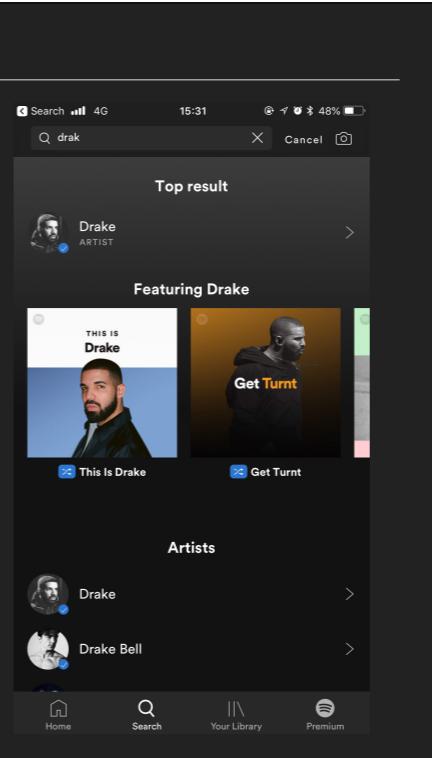
- Static Cells
- Dynamic Prototypes

► TableView Cell

- Basic
- Left / Right Detail
- Subtitle
- Custom

TABLEVIEW CONTROLLER

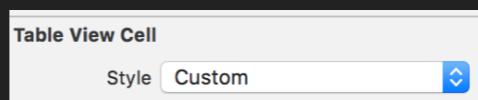
COMO CRIAR UMA
CÉLULA DA FORMA
QUE EU QUERO E
AINDA ASSIM SER UM
TEMPLATE?



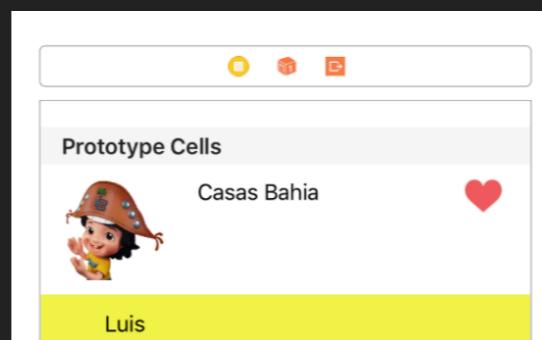


CUSTOM CELLS

TABLEVIEW CONTROLLER

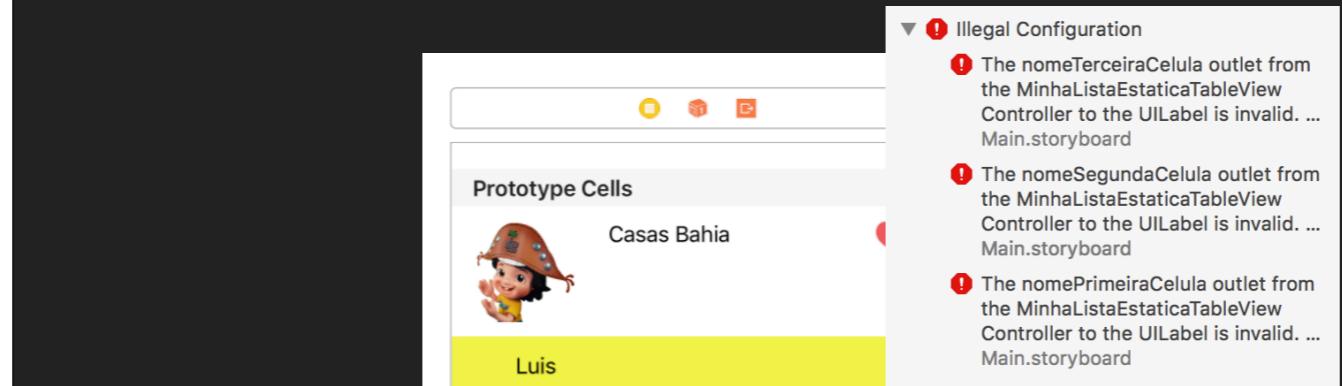


- ▶ TableViewCells
- ▶ Style Custom
- ▶ Então eu posso criar a Cell e ligar ao código?



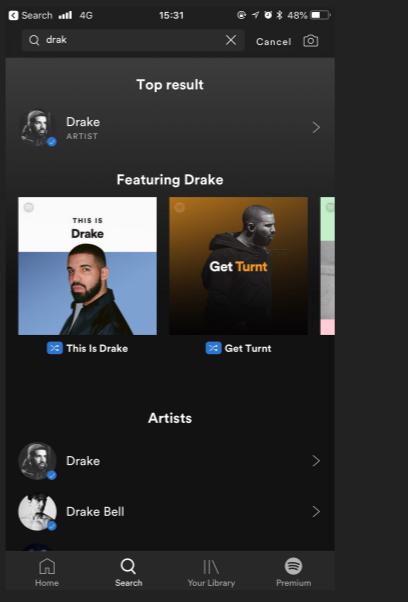
TABLEVIEW CONTROLLER

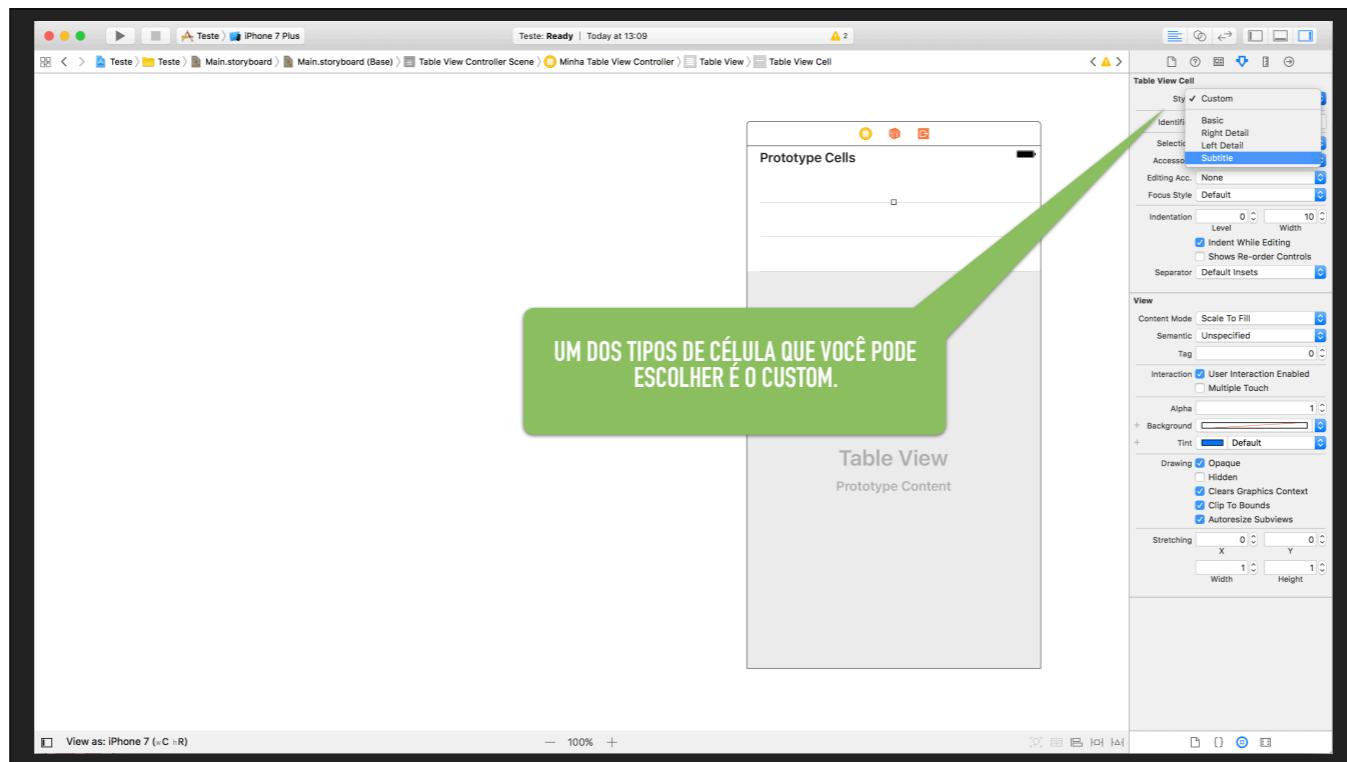
- ▶ Se você fizer isso, você irá receber um erro
- ▶ Todas as Outlets ligadas diretamente a TableView Controller irão indicar erro
- ▶ Inclusive as Outlets que colocamos quando era uma TableView Controller composta por Static Cells, pois mudamos para Dynamic Prototypes

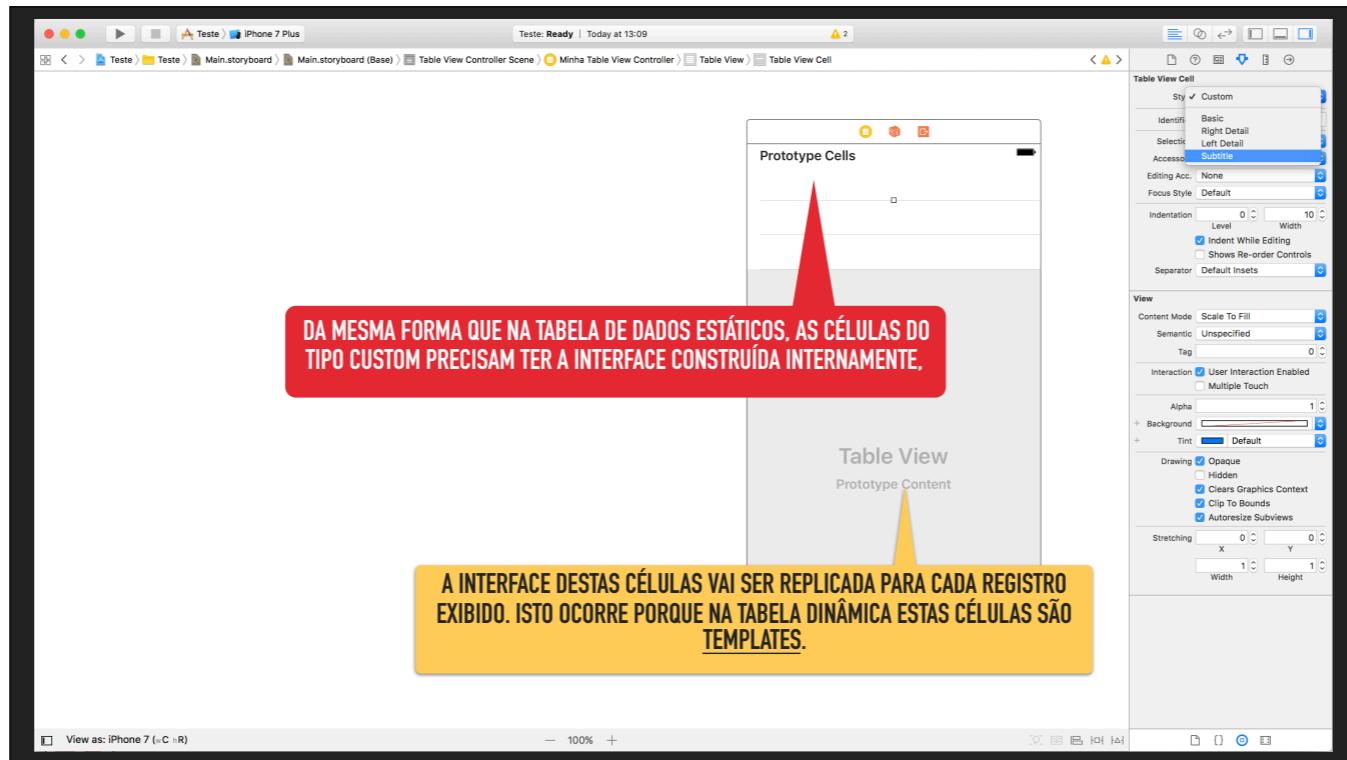


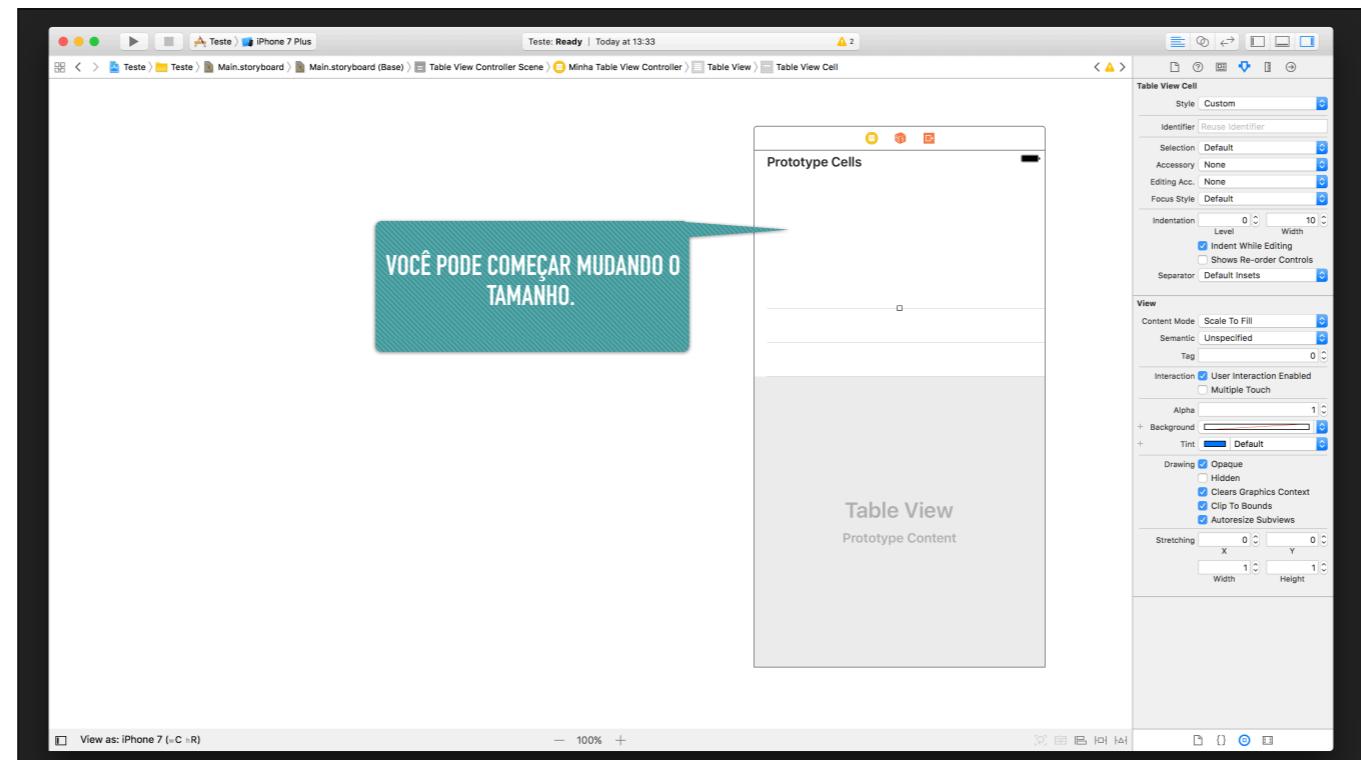
TABLEVIEW CONTROLLER

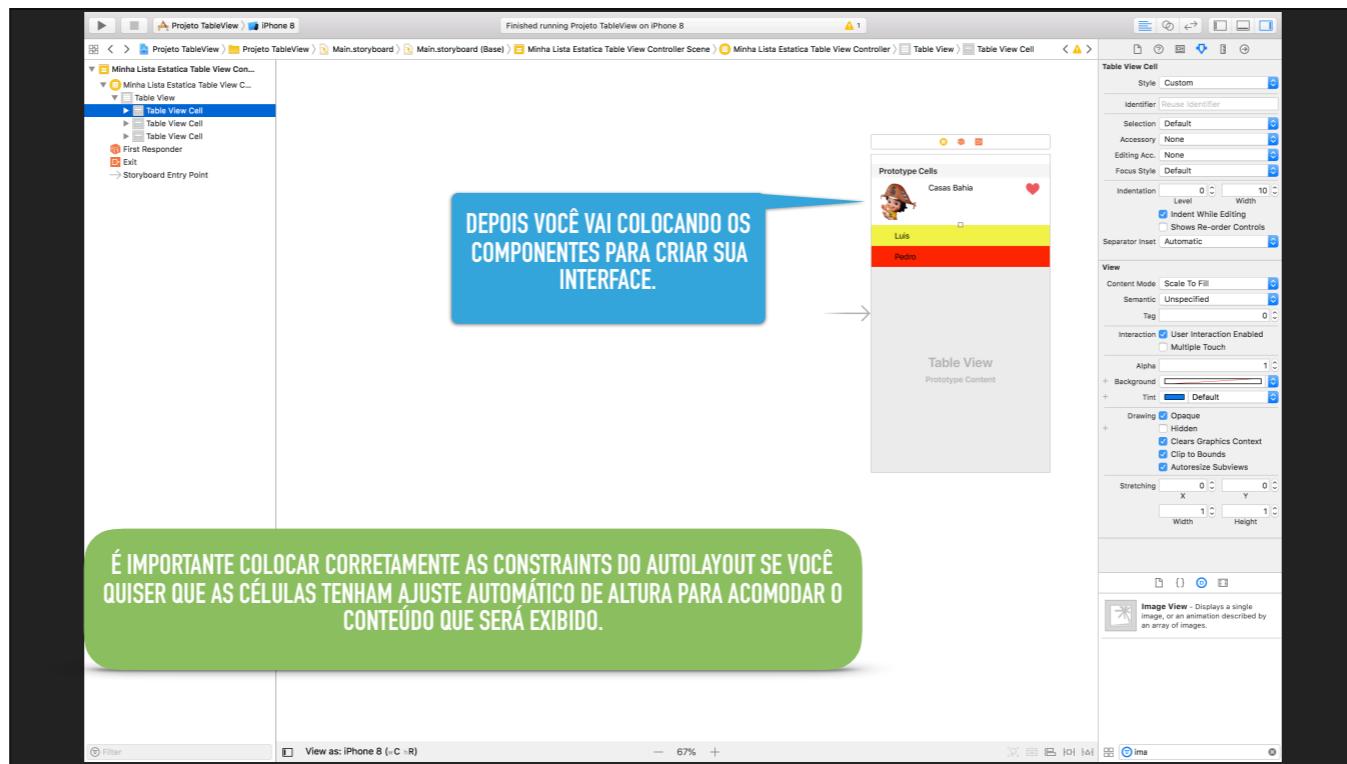
- ▶ Poderiam existir várias Outlets ligadas a mesma classe TableView Controller
- ▶ Com vários tipos de célula isso pode ficar confuso
- ▶ Essas Outlets são da célula
- ▶ Remover as Outlets da TableView Controller

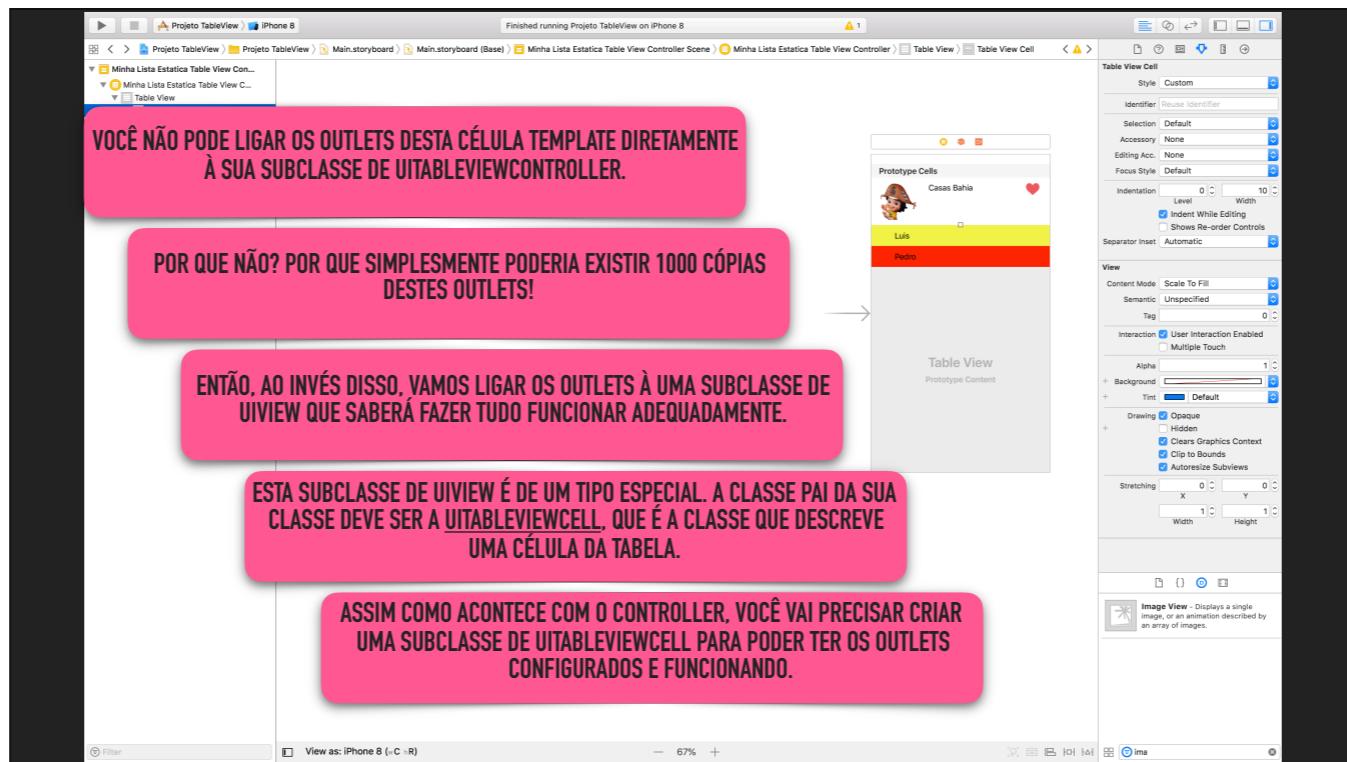


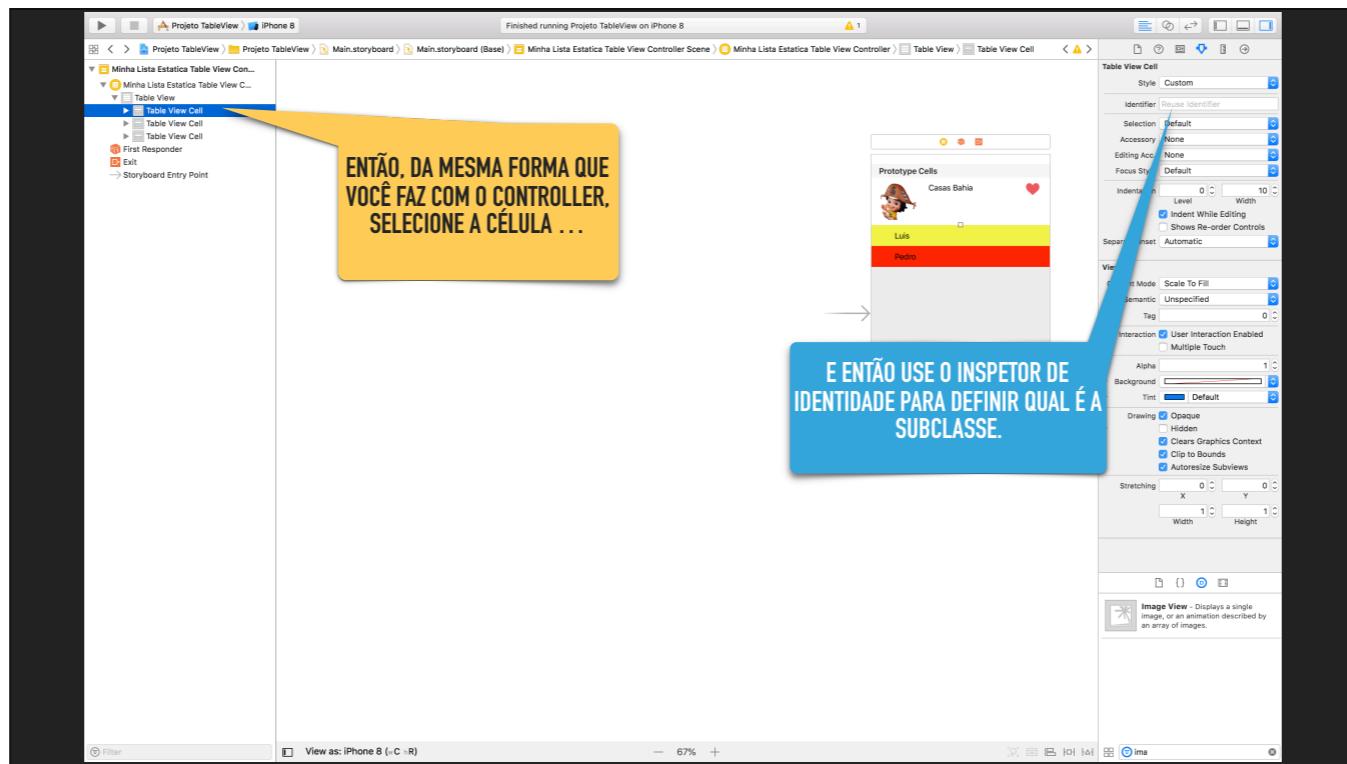




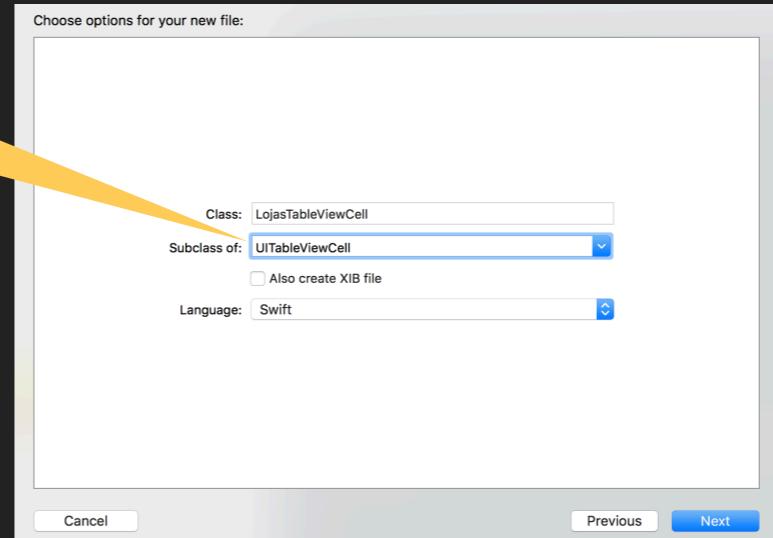


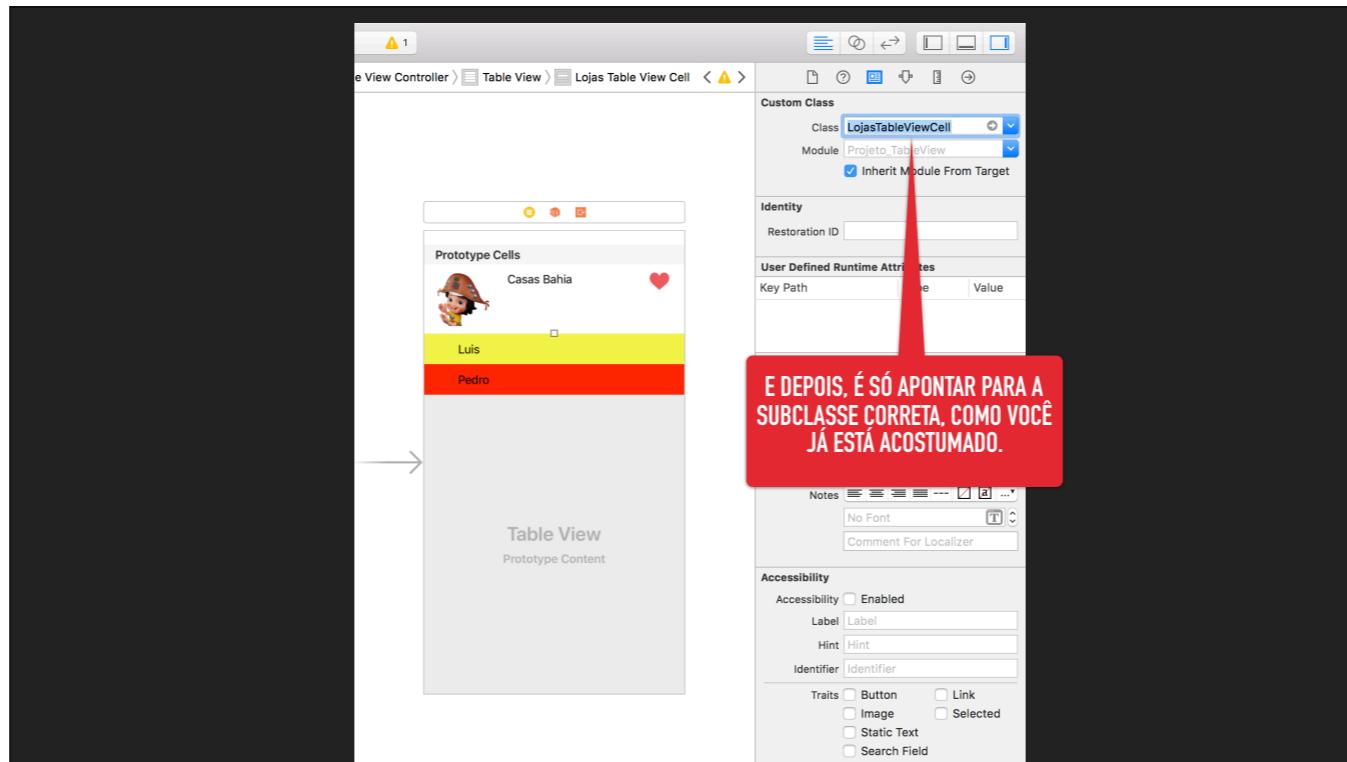


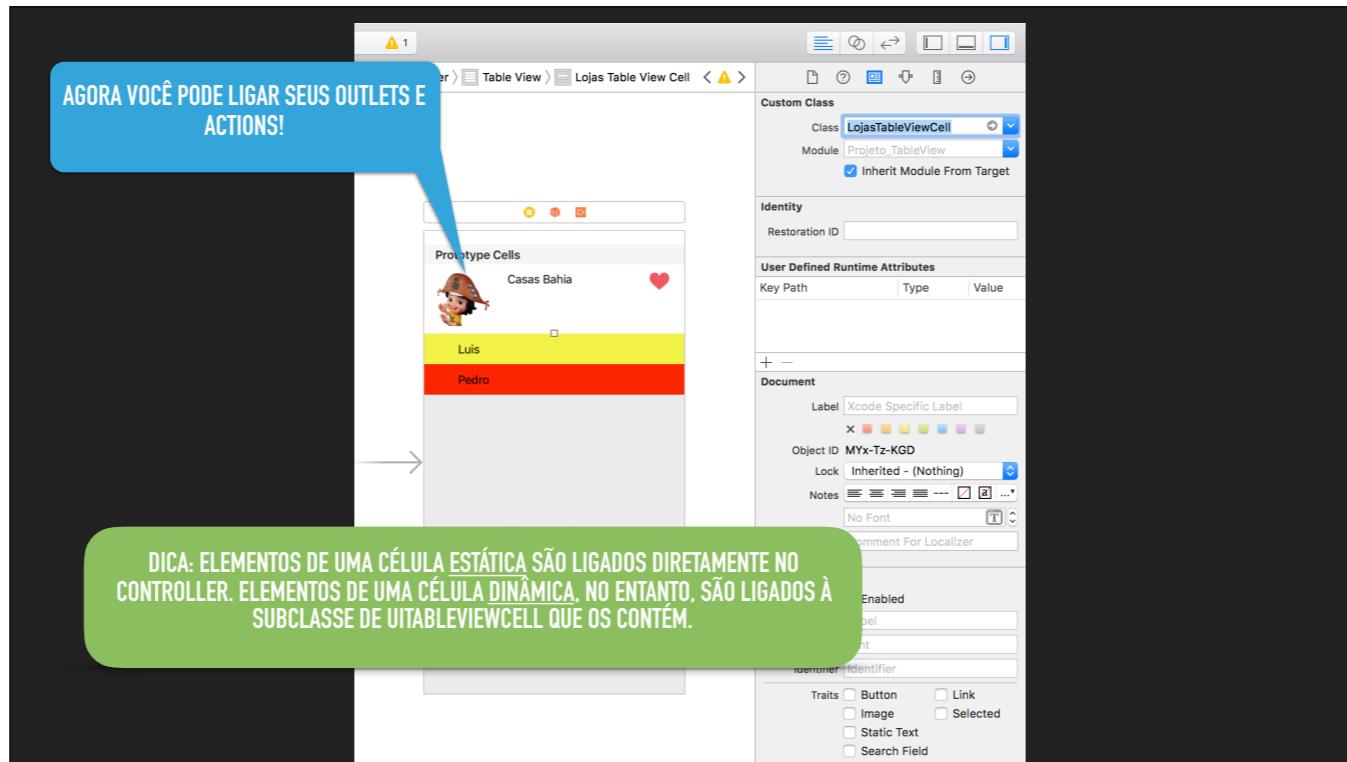


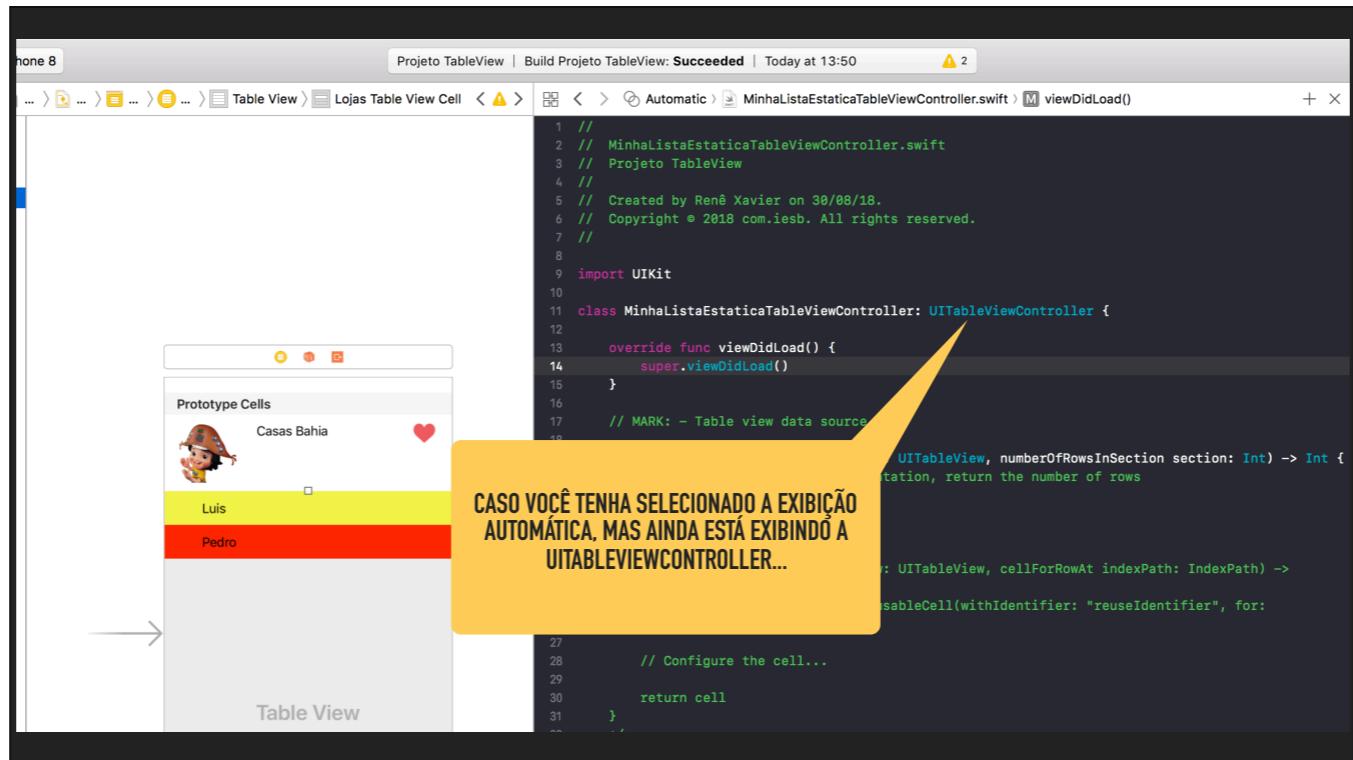


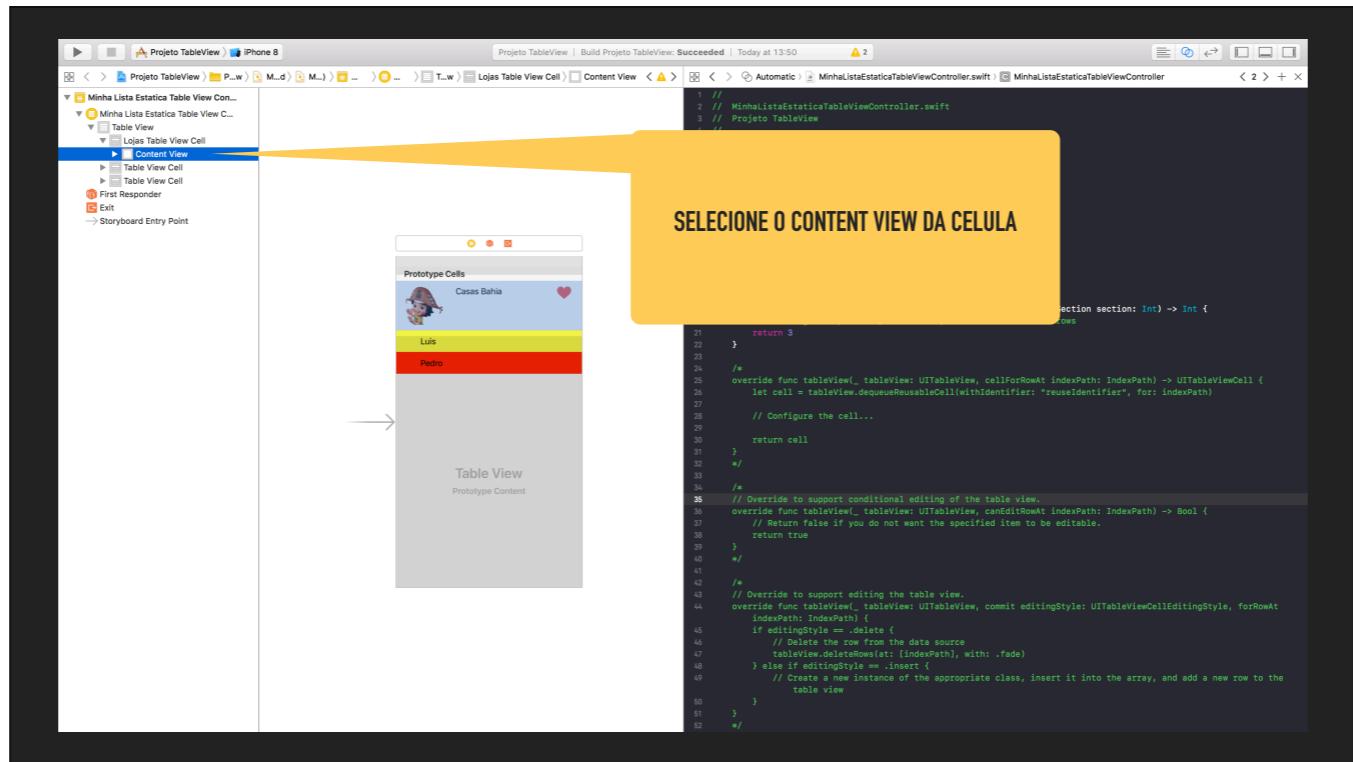
PARA ISSO, VOCÊ PRECISA CRIAR A SUA
SUBCLASSE, QUE HERA DE UITABLEVIEWCELL.

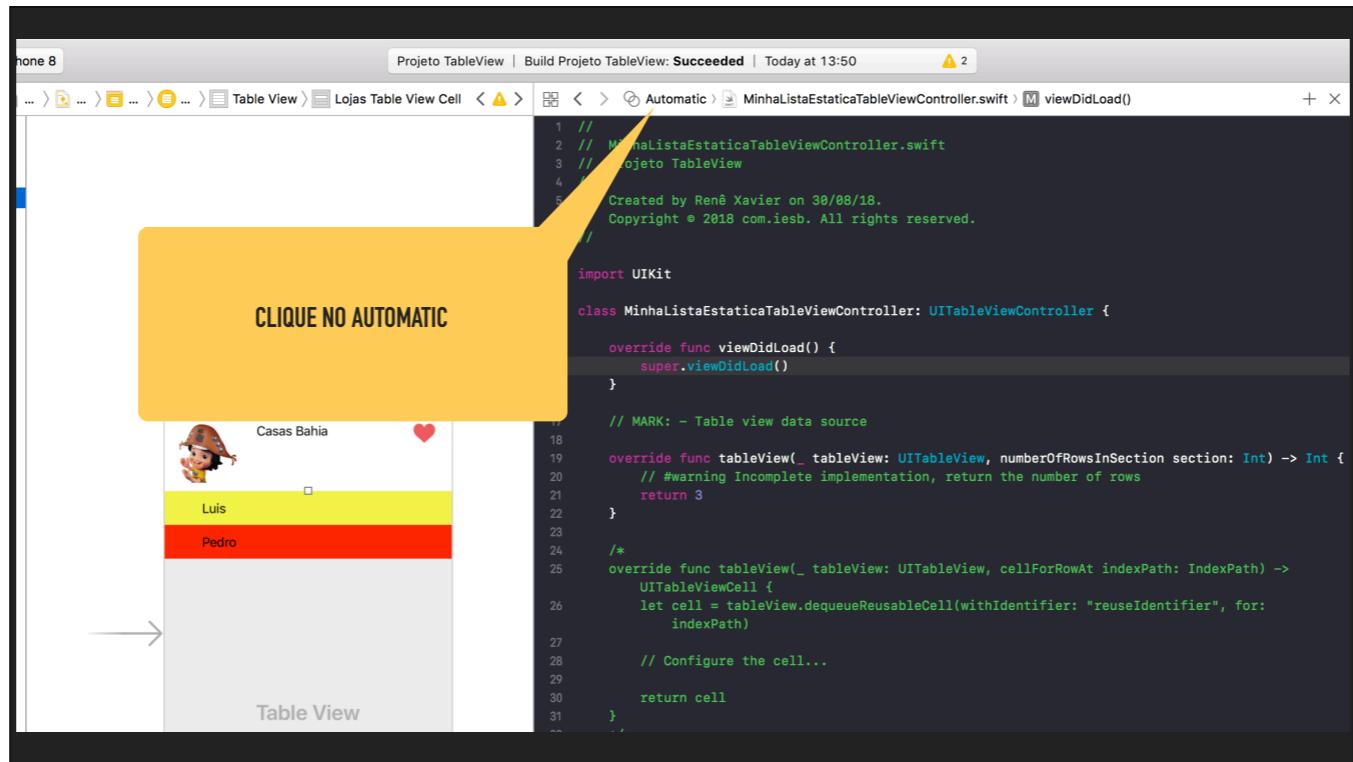


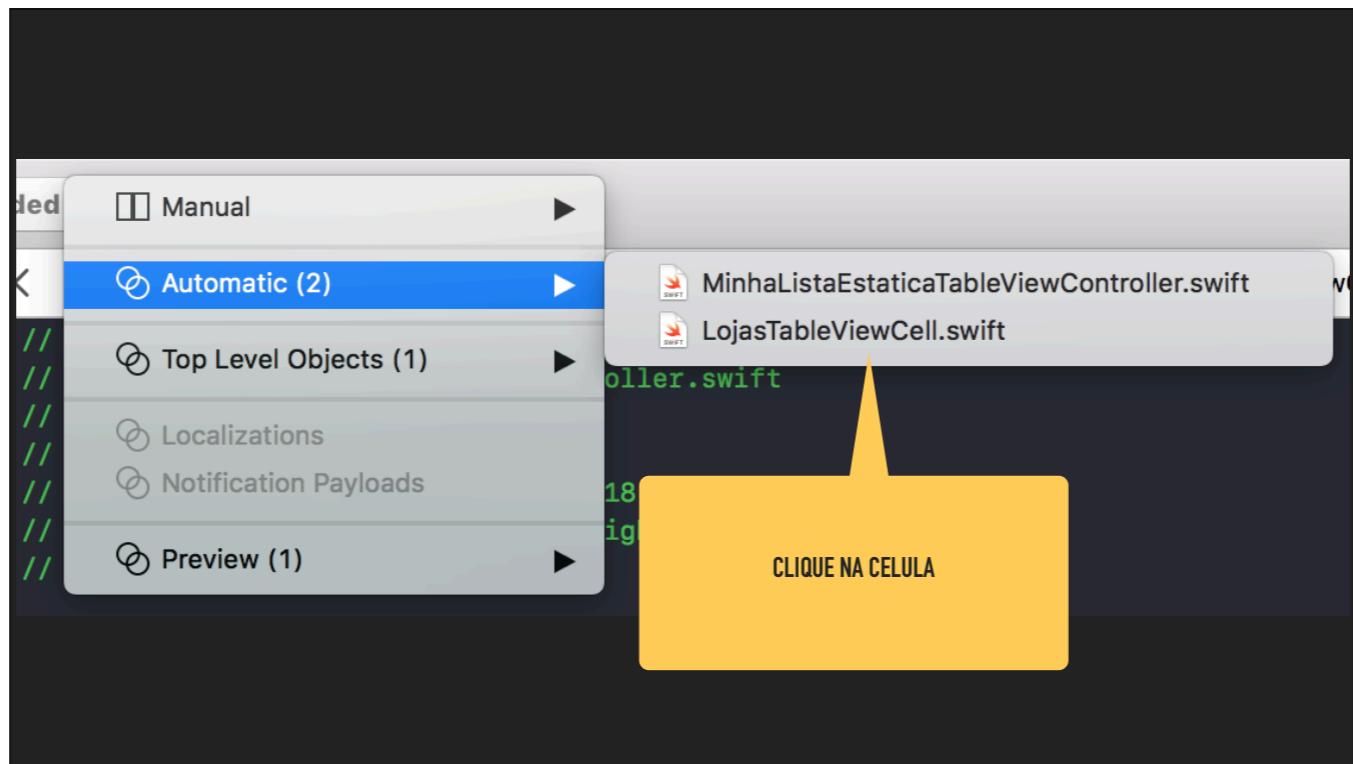


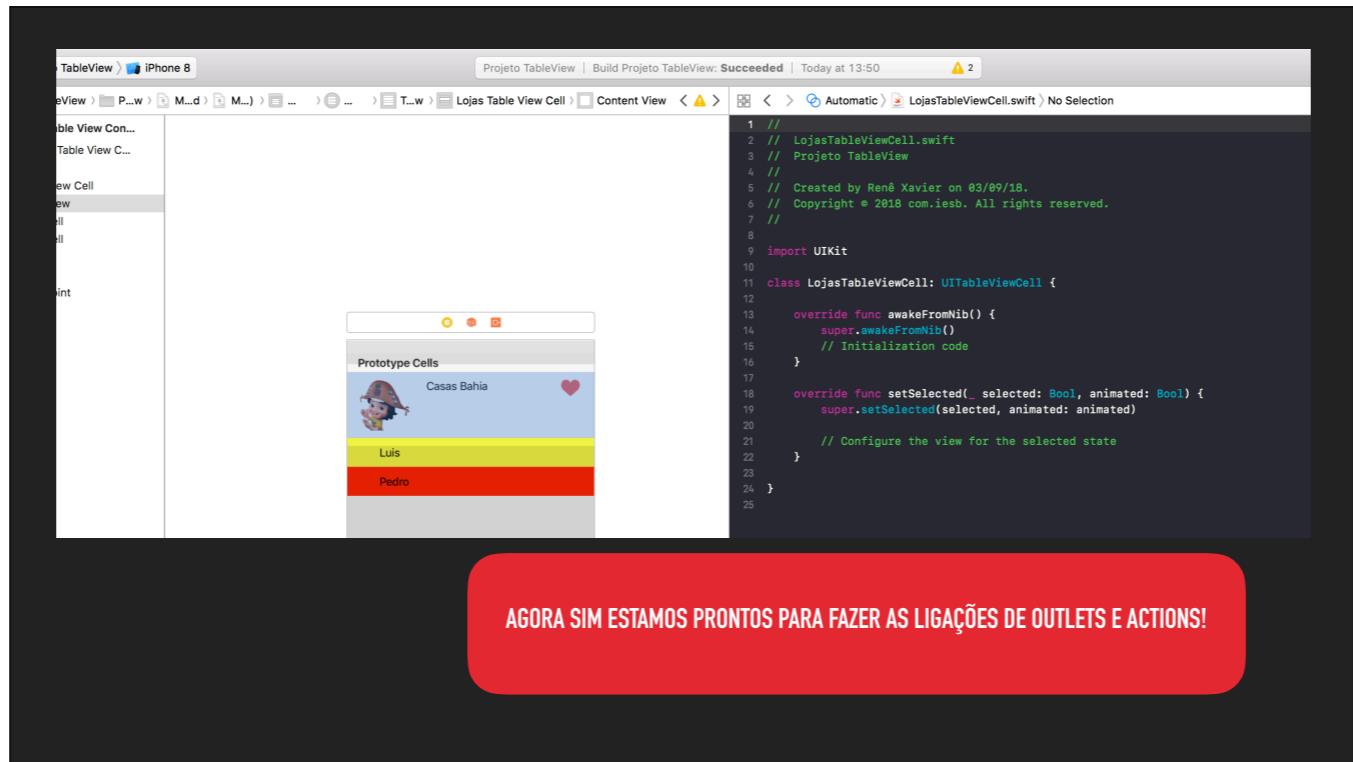




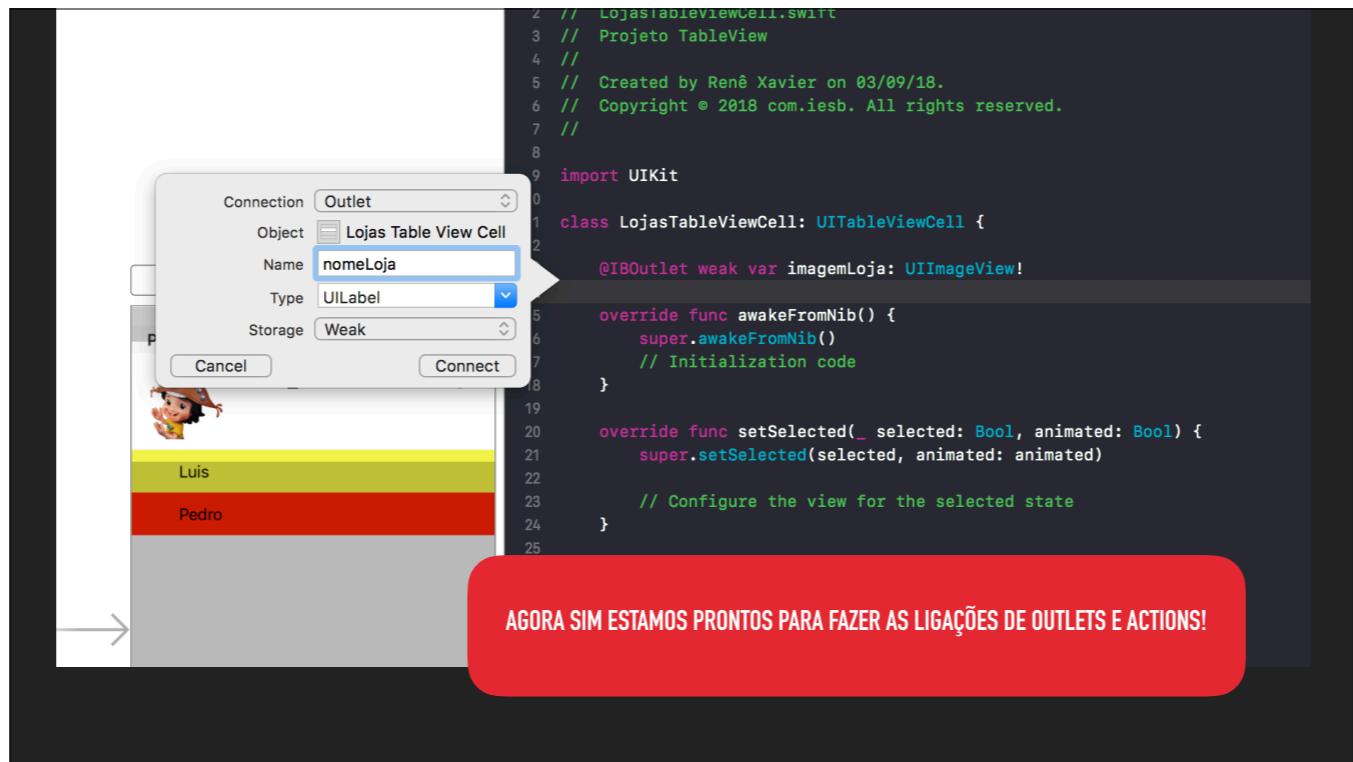


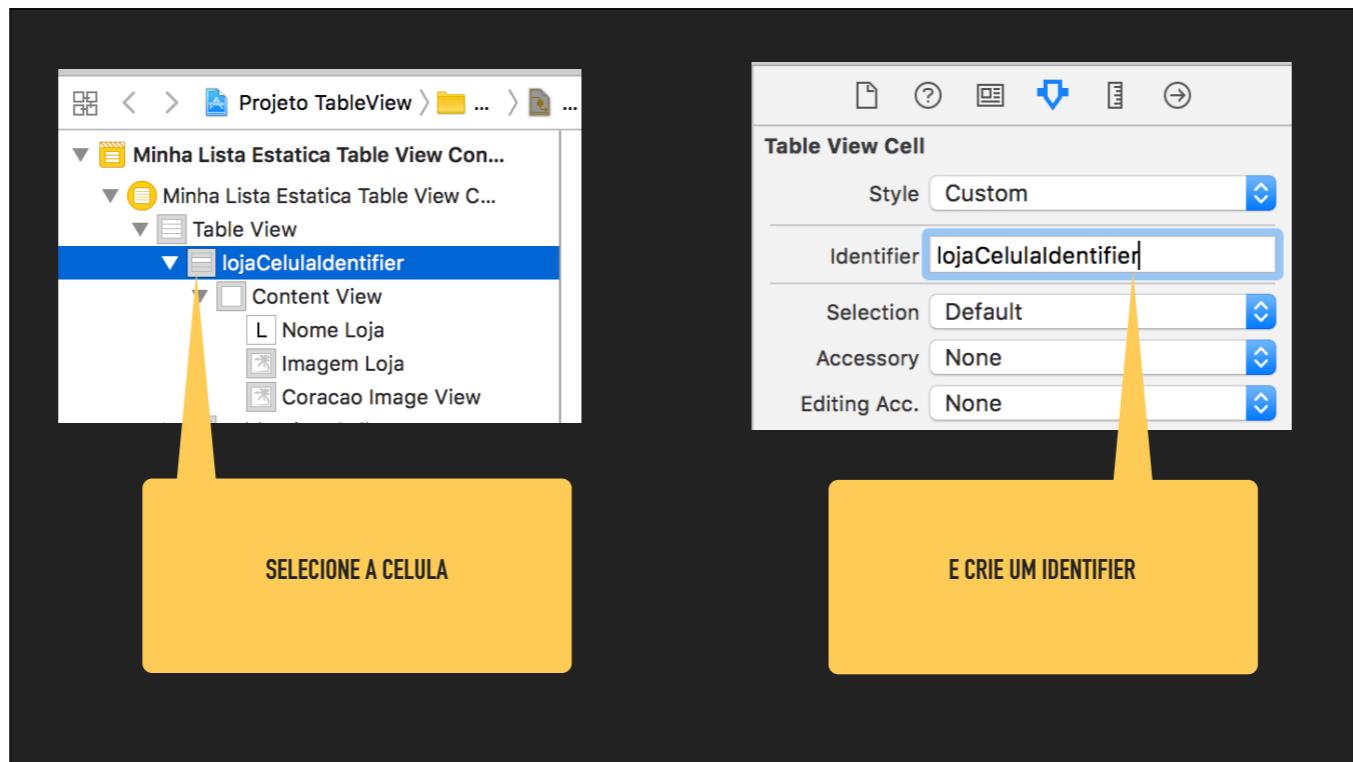


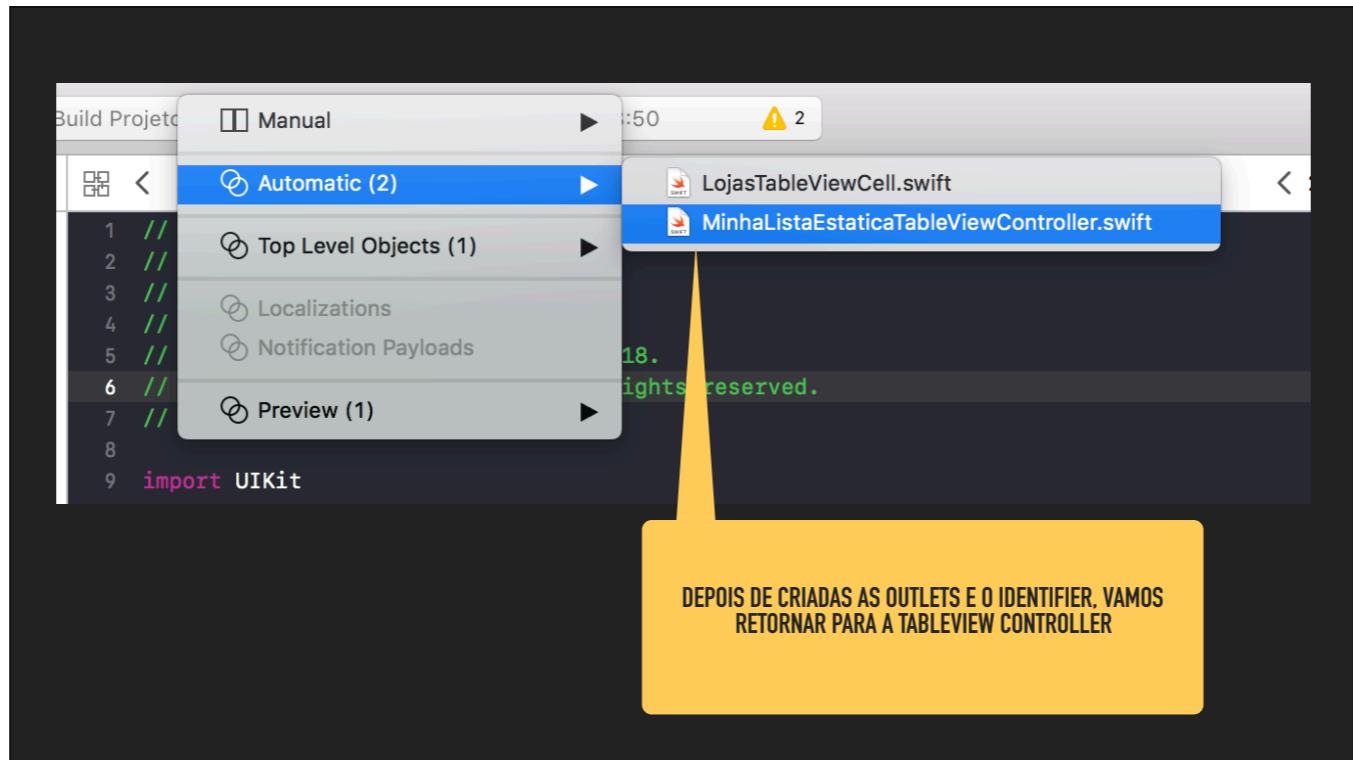




AGORA SIM ESTAMOS PRONTOS PARA FAZER AS LIGAÇÕES DE OUTLETS E ACTIONS!







- ▶ Na nossa ViewController, vamos usar o método numberOfRowsInSection e informar quantas células irão existir em cada section
- ▶ No nosso caso, só existe a Section 0, assim podemos colocar um return com o número de células que queremos

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    return 3  
}
```

PROVENDO UMA UIVIEW PARA DESENHAR CADA CÉLULA

- Antes de desenharmos a célula, vamos conversar um pouco sobre o **dequeReusableCell**
- A UIView que você vai fornecer para cada célula precisa ser subclasse de UITableViewCell (que, por sua vez, é subclasse de UIView);
- Se você tiver 10.000 registros, apenas os que estiverem visíveis irão possuir uma instância dessa classe;
- Isso significa, por outro lado, que as instâncias de UITableViewCell são **reutilizadas** à medida que células aparecem e desaparecem;
- Como é de se imaginar, isso tem complicadores para cenários **multithread**, então seja cuidadoso!

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)  
  
    return cell  
}
```

- ▶ Como vimos o método `cellForRowAt` é um método que editamos a célula
- ▶ Precisamos informar que não queremos usar a `UITableViewCell` (genérica), mas queremos usar na verdade a nossa `LojasTableViewCell` (específica)

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)  
  
    return cell  
}
```

- ▶ Como vimos o método `cellForRowAt` é um método que editamos a célula
- ▶ Precisamos informar que não queremos usar a `UITableViewCell` (genérica), mas queremos usar na verdade a nossa `LojasTableViewCell` (específica)
- ▶ Podemos fazer como já fizemos na segue, usando o `as? NomeDaClasse`
- ▶ Esse erro é lançado, pois o `as?` Indica que a cell pode ser nula e o método tem que retornar um `UITableViewCell` não nulo
- ▶ Vamos tratar

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath) as? LojasTableViewCell|  
  
    return cell  
}
```

! Value of optional type 'LojasTableViewCell?' not unwrapped; did you mean to use '!' or

► Usando o if let, o trecho de código só é executado se a célula for do tipo LojasTableViewCell

► Dentro do if let, a variável cellLojas é do tipo LojasTableViewCell e por ser desse tipo, temos acesso a todas as Outlets que criamos

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)  
    if let cellLojas = cell as? LojasTableViewCell {    ⚠ Value 'cellLojas' was defined but never used; consider replacing  
    }  
    return cell  
}
```

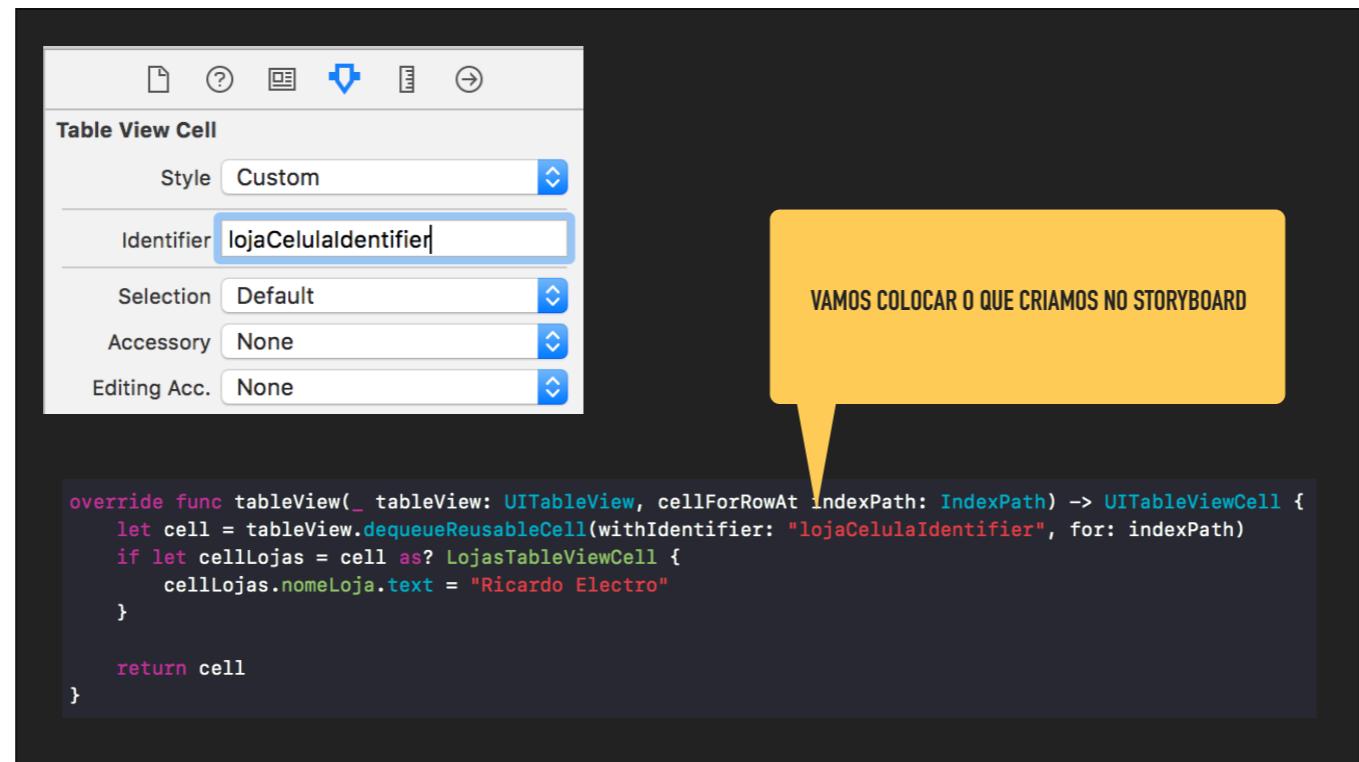
► Usando o if let, o trecho de código só é executado se a célula for do tipo `LojasTableViewCell`

► Dentro do if let, a variável `cellLojas` é do tipo `LojasTableViewCell` e por ser desse tipo, temos acesso a todas as Outlets que criamos

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
    if let cellLojas = cell as? LojasTableViewCell {
        cellLojas.nomeLoja.text = "Ricardo Electro"
    }
    return cell
}
```

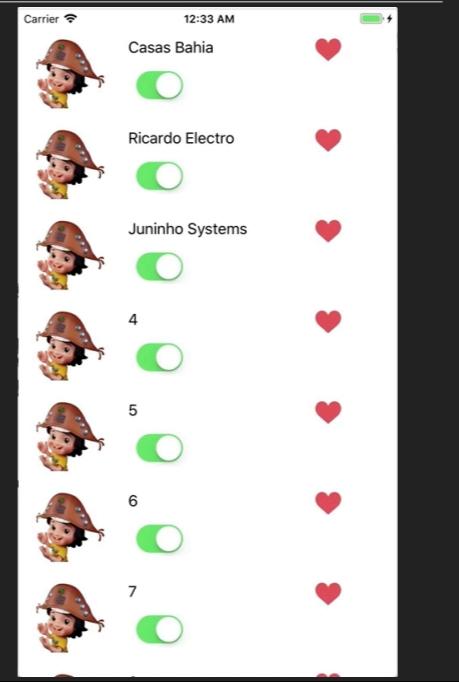
LÓGICO QUE TUDO ISSO NÃO VAI FUNCIONAR COM O
IDENTIFIER ERRADO

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
    if let cellLojas = cell as? LojasTableViewCell {
        cellLojas.nomeLoja.text = "Ricardo Electro"
    }
    return cell
}
```



TABLEVIEW CONTROLLER

UMA SITUAÇÃO CURIOSA



```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "lojaCelulaIdentifier", for: indexPath)
    if let cellLojas = cell as? LojasTableViewCell {
        cellLojas.nomeLoja.text = "Ricardo Electro"
    }
    return cell
}
```

E SE OS DADOS DAS NOSSAS LOJAS ESTIVESSEM EM UM ARRAY?

TABLEVIEW CONTROLLER

TORNANDO DINÂMICO USANDO UM ARRAY

► Primeiro vamos criar o array

```
class MinhaListaEstaticaTableViewController: UITableViewController {  
  
    let arrayLojas = ["Casas Bahia", "Ricardo Electro", "Juninho Systems"]
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "lojaCelulaIdentifier", for: indexPath)  
    if let cellLojas = cell as? LojasTableViewCell {  
        cellLojas.nomeLoja.text = arrayLojas[indexPath.row]  
    }  
  
    return cell  
}
```

O INDEXPATH CONTÉM A INDICAÇÃO DE QUAL SEÇÃO E QUAL CÉLULA ESTÁ SENDO DESENHADA

- ▶ No método numberOfRowsInSection, podemos então substituir pela contagem de elementos no array

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

TABLEVIEW CONTROLLER

**COMO ISSO FICARIA
COM A CAMADA
CONTROLLER?**

TABLEVIEW CONTROLLER

```
class LojasController {  
  
    let arrayLojas = ["Casas Bahia", "Ricardo Electro", "Juninho Systems"]  
  
    func getQuantidadeLojas() -> Int {  
        return arrayLojas.count  
    }  
  
    func nomeLoja(_ index: Int?) -> String {  
        guard let index = index, index < arrayLojas.count else {  
            return ""  
        }  
        return arrayLojas[index]  
    }  
}
```

VAMOS CRIAR DOIS MÉTODOS

TABLEVIEW CONTROLLER

```
class MinhaListaEstaticaTableViewController: UITableViewController {  
  
    let controller = LojasController() CRIAMOS A CONTROLLER  
  
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
        return controller.getQuantidadeLojas()  
    } VAMOS PEGAR A QUANTIDADE DA CONTROLLER  
  
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
        let cell = tableView.dequeueReusableCell(withIdentifier: "lojaCelulaIdentifier", for: indexPath)  
  
        if let cellLojas = cell as? LojasTableViewCell {  
            cellLojas.nomeLoja.text = controller.nomeLoja(indexPath.row) ...E O NOME TAMBÉM  
        }  
        return cell  
    }  
}
```



QUAL É A DIFERENÇA ENTRE A
TABLEVIEW E A TABLEVIEW
CONTROLLER?

UITABLEVIEW CONTROLLER

- ▶ A tela é completamente uma TableView, **NÃO** temos outros elementos;
- ▶ Pode ter **Dynamic Prototypes**;
- ▶ Pode ter **Static Cells**;
- ▶ A UITableViewController provê uma **Outlet** para a TableView
- ▶ O UITableViewController configura **automaticamente** as ligações de **delegate** e **dataSource**;
- ▶ **Não** temos que colocar **UITableViewDelegate** e **UITableViewDataSource** na declaração da classe

UITABLEVIEW

- ▶ A TableView faz parte da tela, mas não é o único elemento;
- ▶ Pode ter **Dynamic Prototypes**;
- ▶ **NÃO** é possível ter **Static Cells**;
- ▶ Temos que **criar** uma **Outlet** a partir do Storyboard para ter acesso a Table View
- ▶ Temos que fazer a conexão manual do **delegate** e do **dataSource**;
- ▶ Temos que colocar **UITableViewDelegate** e **UITableViewDataSource** na declaração da classe;

O QUE É O TABLEVIEW DELEGATE E O DATA SOURCE?

DELEGATE E DATA SOURCE

SE LEMBRAM DO PADRÃO DELEGATE?

- ▶ Para implementar um padrão Delegate temos que:
- ▶ Preencher a variável delegate
- ▶ Declarar que implementamos o protocolo
- ▶ Implementar as funções do protocolo na nossa classe
- ▶ Então o **Delegate** e o **Data Source** são nada mais que um padrão a se implementar para podermos ter controle sobre nossa UITableView

SE LEMBRAM QUE O PADRÃO DELEGATE TAMBÉM É USADO PARA CRIAR COMPONENTES?

OS PROTOCOLOS DA UITABLEVIEW...

- ▶ As conexões ao código são feitas usando os protocolos da UITableView chamados **dataSource** e **delegate**;
- ▶ O **delegate** é usado para controlar como a tabela vai ser exibida (o look and feel da tabela);
- ▶ Já o **dataSource** é responsável por prover os dados que são exibidos nas linhas da tabela - dentro das células;
- ▶ O **UITableViewController** vai configurar automaticamente as ligações de **delegate** e **dataSource**;
- ▶ No caso da **UITableView** temos que preencher com um objeto que implementa esses protocolos - self

UITABLEVIEW DATA SOURCE

- ▶ Quando os dados a exibir na tabela forem dinâmicos (as células não são estáticas);
- ▶ Existem três métodos muito importantes neste protocolo que fazem perguntas fundamentais para o funcionamento da tabela:
Quantas seções existem na tabela?
Quantas linhas existem em cada seção?
Qual é a célula que eu tenho que desenhar para uma dada posição (seção e linha)?
- ▶ Já conheciamos eles:

numberOfSections in tableView

numberOfRowsInSection

cellForRowAtIndexPath

UITABLEVIEW DELEGATE

- ▶ O delegate controla **como** a tabela é exibida. Não os dados que ela exibe;
- ▶ É normal que o dataSource e o delegate sejam o mesmo objeto. - SELF
- ▶ O delegate também de permite observar o que a tabela anda fazendo.

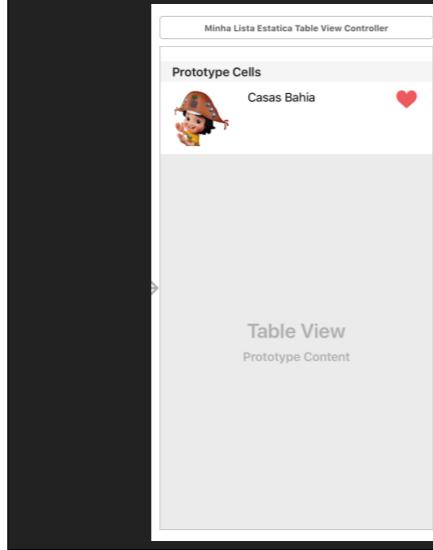
```
func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
```

```
func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {
```

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
```

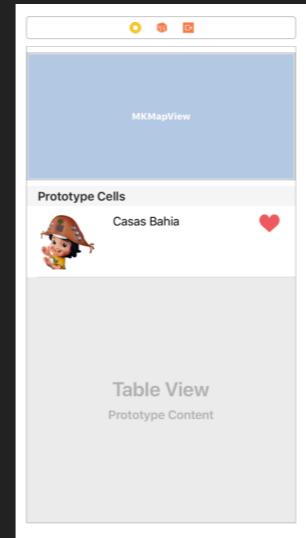
UITABLEVIEW CONTROLLER

- A tela é completamente uma TableView,
NÃO temos outros elementos;



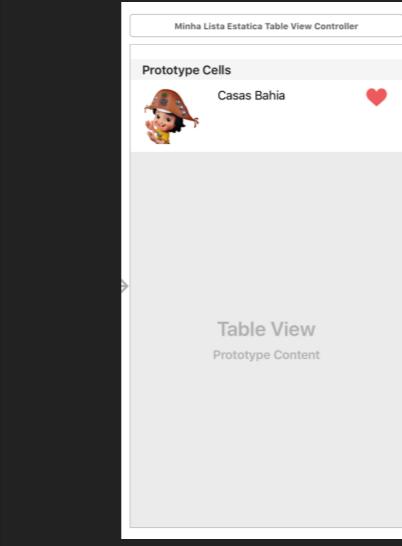
UITABLEVIEW

- A TableView faz parte da tela, mas
não é o único elemento;



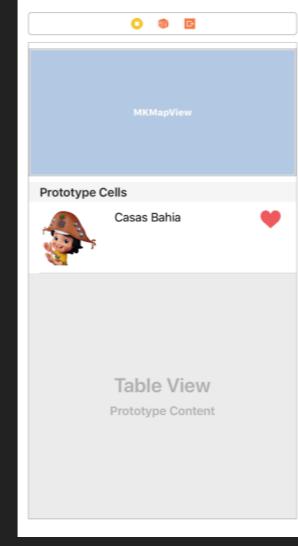
UITABLEVIEW CONTROLLER

- ▶ Pode ter Dynamic Prototypes;



UITABLEVIEW

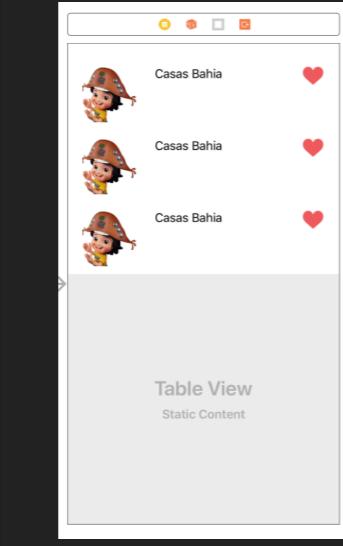
- ▶ Pode ter Dynamic Prototypes;



SEMELHANTE

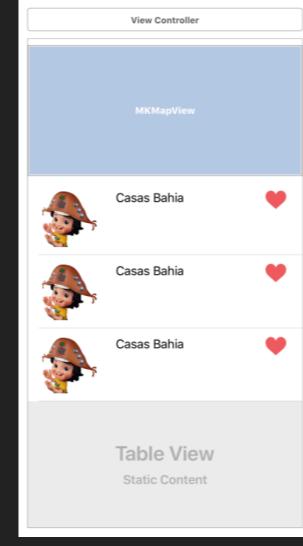
UITABLEVIEW CONTROLLER

- ▶ Pode ter Static Cells;



UITABLEVIEW

- ▶ NÃO é possível ter Static Cells;



UITABLEVIEW CONTROLLER

- ▶ A UITableViewController provê uma Outlet para a UITableView

```
class MinhaListaTableViewController: UITableViewController {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        tableView.backgroundColor = .red  
    }  
}
```

UITABLEVIEW

- ▶ Pode ter Dynamic Prototypes;

```
class ListaViewController: UIViewController {  
  
    @IBOutlet weak var tableView: UITableView!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        tableView.backgroundColor = .red  
    }  
}
```

UITABLEVIEW CONTROLLER

- ▶ O `UITableViewController` configura **automaticamente** as ligações de `delegate` e `dataSource`;

NÃO PRECISA, POIS A
UITABLEVIEW CONTROLLER
FAZ POR VOCÊ

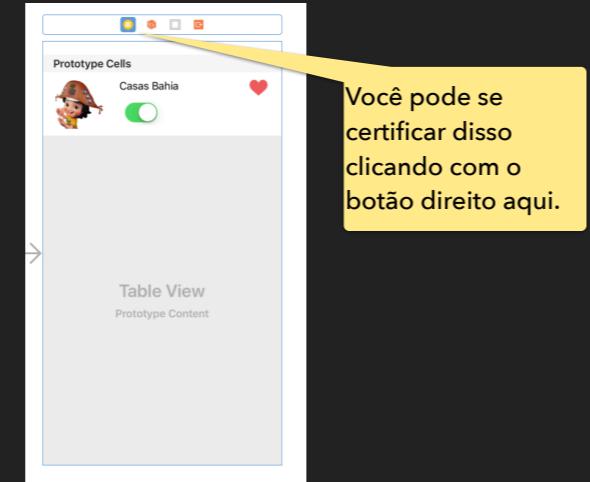
UITABLEVIEW

- ▶ Temos que fazer a conexão manual do `delegate` e do `dataSource`;

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    tableView.delegate = self  
    tableView.dataSource = self  
}
```

UITABLEVIEW CONTROLLER

- ▶ O UITableViewController configura automaticamente as ligações de delegate e dataSource;



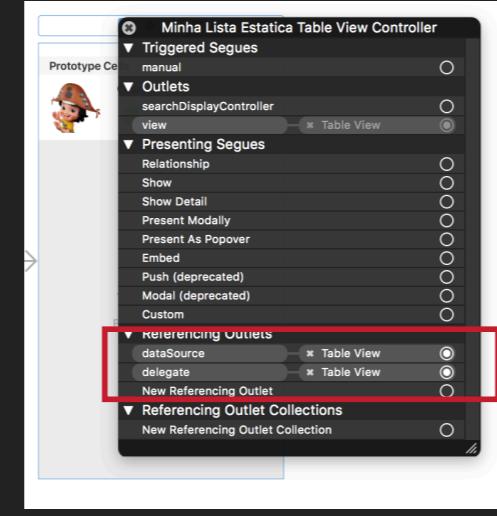
UITABLEVIEW

- ▶ Temos que fazer a conexão manual do delegate e do dataSource;

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    tableView.delegate = self  
    tableView.dataSource = self  
}
```

UITABLEVIEW CONTROLLER

- » O UITableViewController configura automaticamente as ligações de delegate e dataSource;



UITABLEVIEW

- » Temos que fazer a conexão manual do delegate e do dataSource;

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    tableView.delegate = self  
    tableView.dataSource = self  
}
```

UITABLEVIEW CONTROLLER

- ▶ **Não** temos que colocar UITableViewDelegate e UITableViewDataSource na declaração da classe

```
class MinhaListaEstaticaTableViewController:  
    UITableViewController {
```

UITABLEVIEW

- ▶ Temos que colocar UITableViewDelegate e UITableViewDataSource na declaração da classe;

```
class ListaViewController: UIViewController,  
    UITableViewDelegate, UITableViewDataSource {
```

UITABLEVIEW CONTROLLER

- ▶ Temos que implementar o método `numberOfRowsInSection`;
- ▶ Temos que implementar o método `cellForRowAt`;

```
func tableView(_ tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int {
    return 3
}

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
    IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
        "lojaCelulaIdentifier", for: indexPath)

    return cell
}
```

UITABLEVIEW

- ▶ Temos que implementar o método `numberOfRowsInSection`;
- ▶ Temos que implementar o método `cellForRowAt`;

```
func tableView(_ tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int {
    return 3
}

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
    IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
        "lojaCelulaIdentifier", for: indexPath)

    return cell
}
```

SEMELHANTE



**COMO RECARREGAR MINHA
TABLEVIEW CASO MEU ARRAY
MUDE?**

E SE A MINHA MODEL MUDAR?

- ▶ O método **reloadData()** apaga tudo que está na tabela e carrega tudo novamente, chamando todos os métodos envolvidos do **dataSource**;
- ▶ Como é de se imaginar, este método é relativamente pesado, mas se a sua estrutura de dados mudar completamente, é dele que você vai precisar;
- ▶ Mas, por outro lado, se apenas um pedaço do dado mudar, existe uma função mais leve para atualizar só o que interessa:
reloadRows(at indexPaths: [IndexPath], with animation: UITableViewRowAnimation)

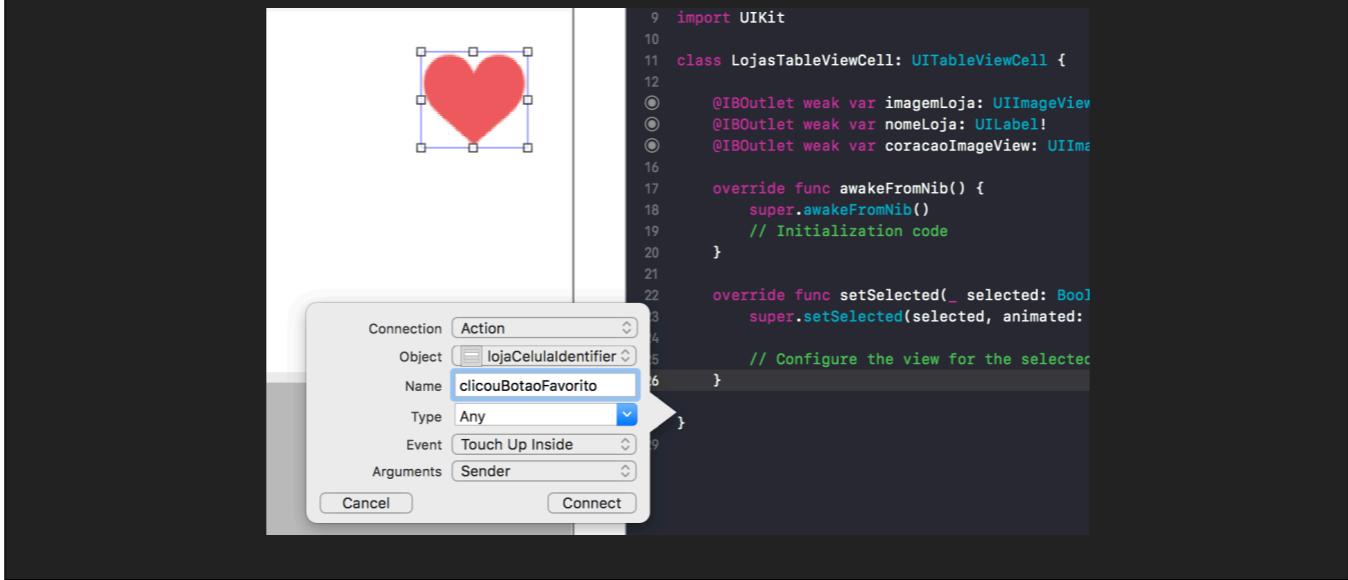


COMO TRATAR UMA
AÇÃO NA CÉLULA?

“ACTIONS” COM O UITABLEVIEW

- ▶ Como vimos, células podem ter segues para outras ViewControllers;
- ▶ E se existir um botão? Ele vai ser clicável mesmo com a segue?
- ▶ Sim. O botão é clicável, temos que configurar a IBAction na nossa célula para isso

"ACTIONS" COM O UITABLEVIEW



“ACTIONS” COM O UITABLEVIEW

```
@IBAction func clicouBotaoFavorito(_ sender: Any) {
    let image = coracaoButton.currentImage == UIImage(named: "like") ? UIImage(named: "liked") : UIImage(named: "like")
    coracaoButton.setImage(image, for: .normal)
}
```

- ▶ A partir daqui temos duas opções:
 - ▶ Criar uma Controller e atualizar a Model (Banco de dados)
 - ▶ Criar um delegate para a TableView Controller/ViewController fazer a atualização da Model



COMO AJUSTAR A ALTURA
DE UMA CÉLULA?

CONTROLANDO A ALTURA DAS CÉLULAS

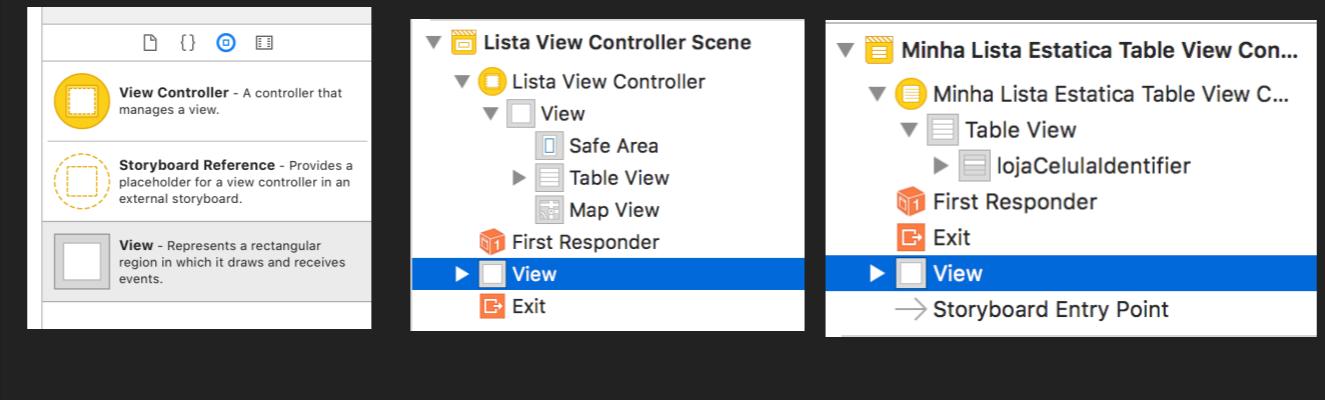
- ▶ A altura das células pode ser fixa (storyboard, via propriedade **rowHeight**);
- ▶ Ou, ela pode ser determinada usando o autolayout (setar a propriedade **rowHeight = UITableViewAutomaticDimension**);
- ▶ Caso você vá para o lado automático, você pode dar uma ajudinha para a tableView através da propriedade **estimatedRowHeight**. Quando esta propriedade está preenchida, a tabela só executa o autolayout das linhas que estão visíveis e aplica o valor estimado nas demais;



COMO ADICIONAR UM
CABEÇALHO PRA UMA SECTION?

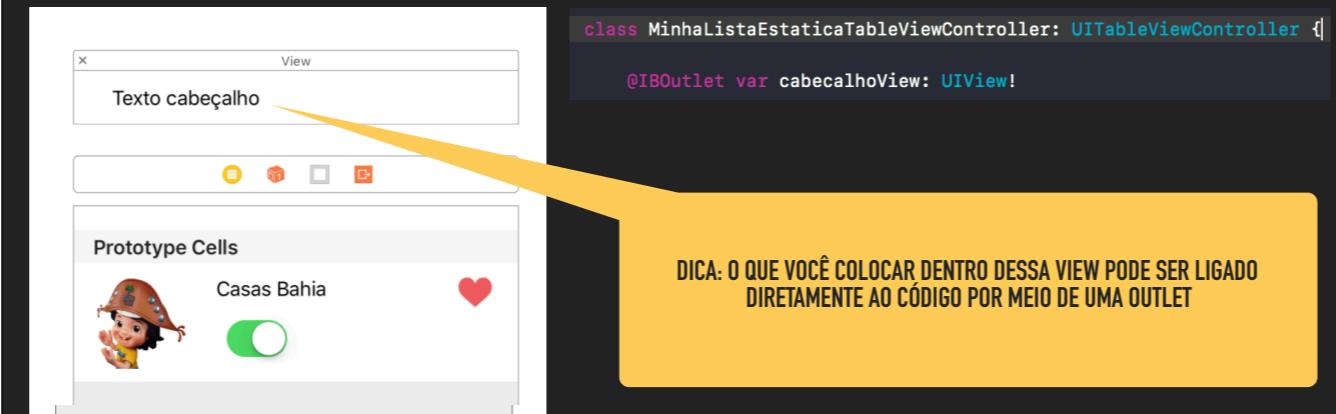
CABEÇALHO PARA SECTION

- ▶ Selecione uma UIView arraste para a sua ViewController, mas não dentro da TableView
- ▶ Arraste entre o **First Responder** e o **Exit**. Então solte.
- ▶ Podem existir mais de uma View ali, para diferentes tipos de Cabeçalhos.



CABEÇALHO PARA SECTION

- ▶ Essa view irá aparecer em cima da ViewController no Storyboard
- ▶ Crie uma Outlet da View criada no código
- ▶ Preencha essa view com o que você quiser no Cabeçalho.



CABEÇALHO PARA SECTION

- ▶ Implemente o método **viewForHeaderInSection** e retorne a view que você conectou pelo Outlet
- ▶ Você pode determinar qual Section irá exibir qual View avaliando a variável **section**

```
override func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {  
    return cabecalhoView  
}
```



CONCLUSÃO

CONCLUSÃO

- ▶ Existem várias dezenas de métodos e outras coisas interessantes sobre a tableView;
- ▶ Coisas para lidar com cabeçalhos, rodapés, controle refinado de aparência (estilo, cor e altura do separador, por exemplo), controle do scroll;
- ▶ Ainda temos meios para gerenciar as seções, lidar com seleção única e múltipla de registros, mover, inserir e excluir linhas;
- ▶ E, mais uma vez, não deixe de conferir da documentação para saber tudo mais que existe de incrível sobre este que é o componente mais usado do iOS, desde que surgiu.