Xcode

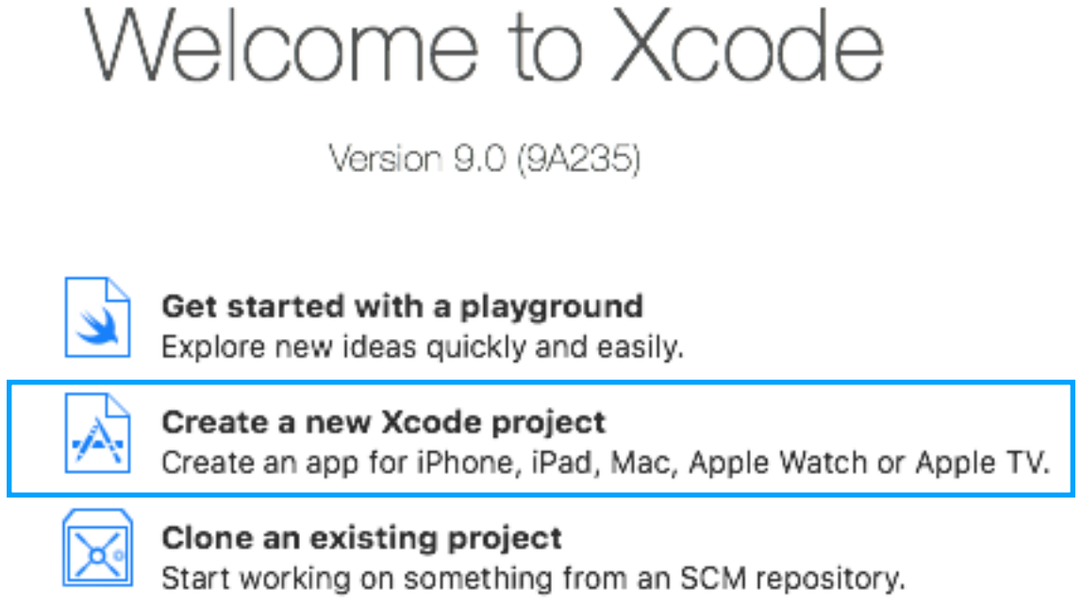
- Download Xcode from the Mac App Store
- Once downloaded, install the additional components when prompted

ACM iOS Tutorial

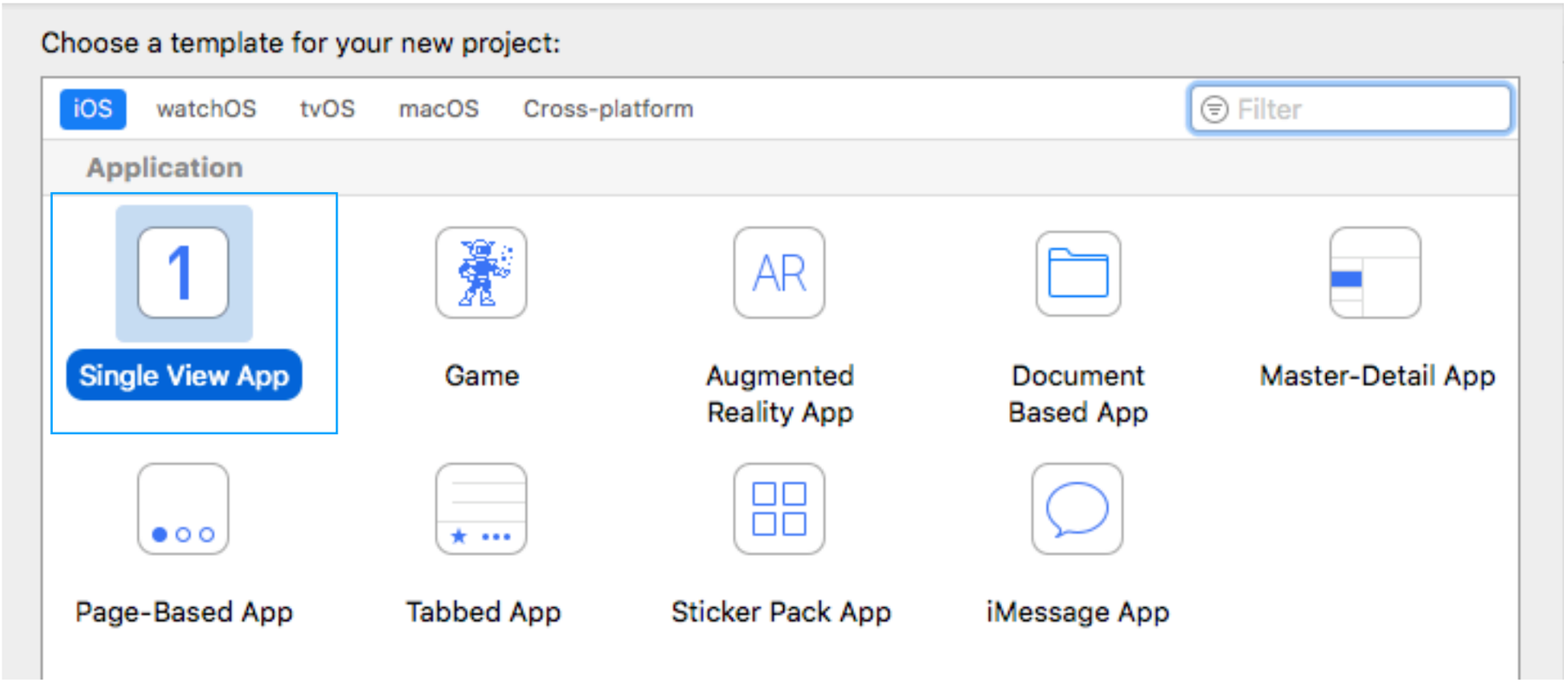
9/30/2017

Full code and steps:
https://github.com/masonBruce/ACM-Tutorial

Create a new project

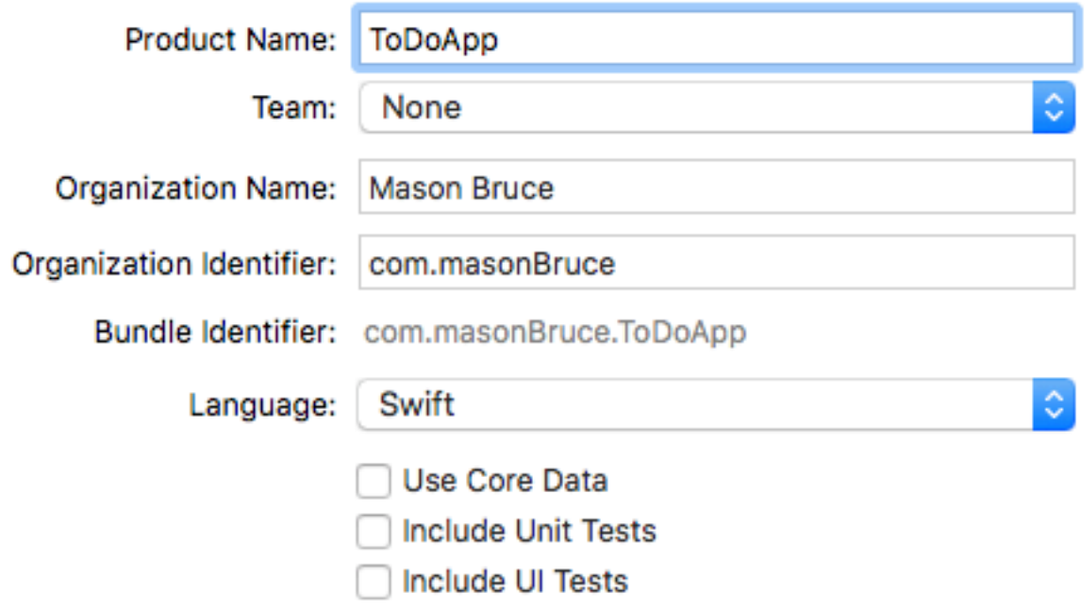


Create a new project



Choose iOS Single View App

Create a new project



Name your app
Choose Swift as the programming language

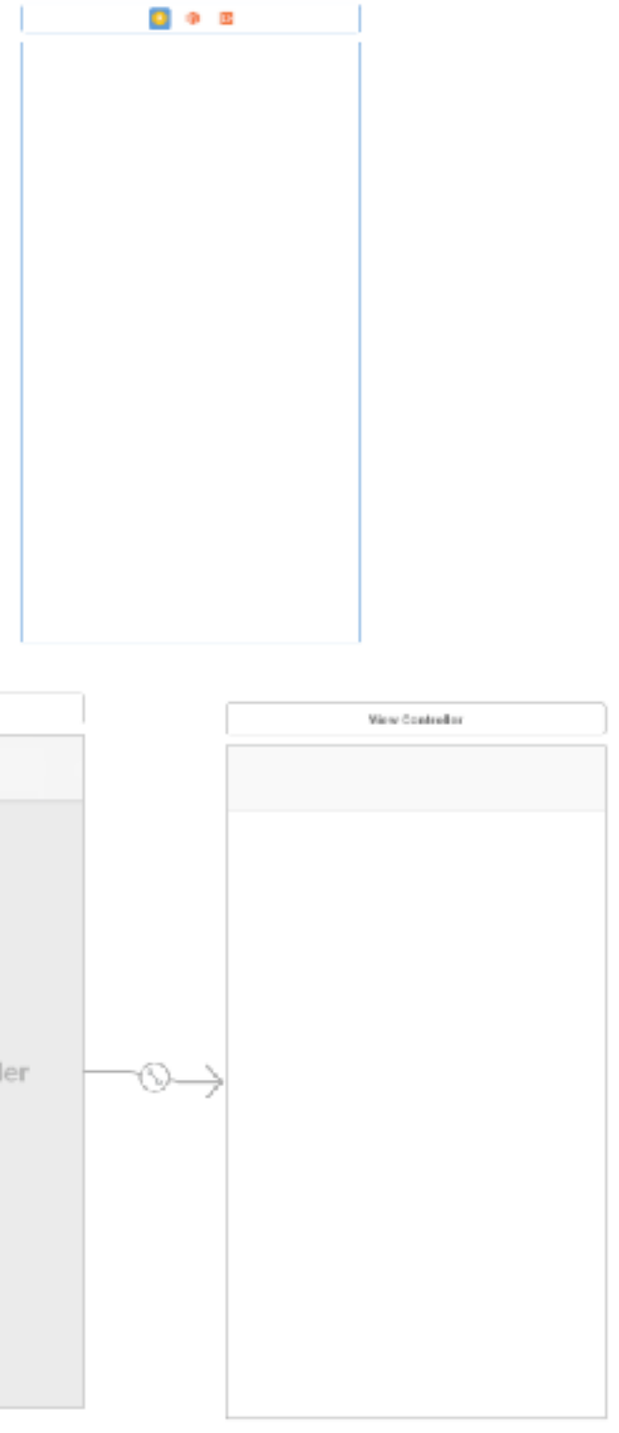
Choose a location and save your project

Storyboard

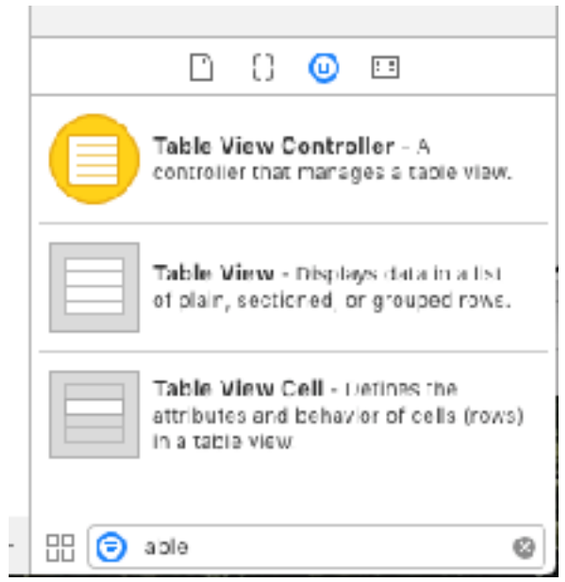
- Navigate to Main.storyboard

- Click on the bar above the UIViewController (The white rectangle) to select it

- Click on Editor>Embed In > Navigation Controller

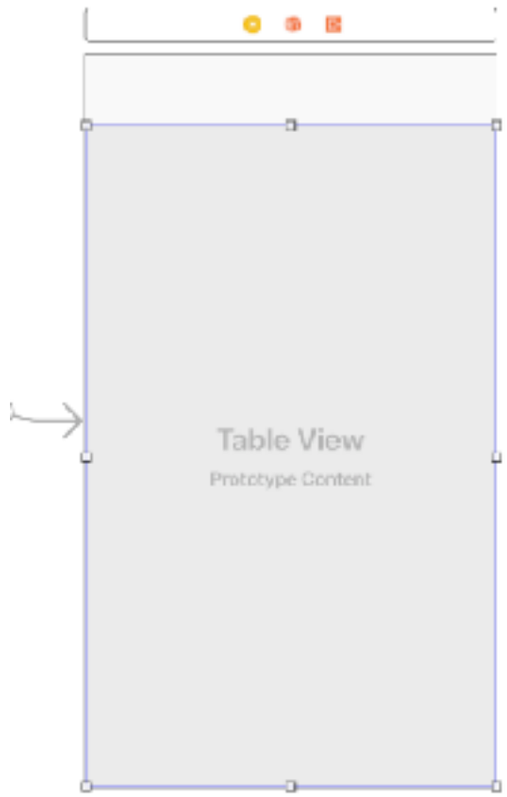


Storyboard



- Search for “Table View” in the menu in the bottom-right corner of Xcode

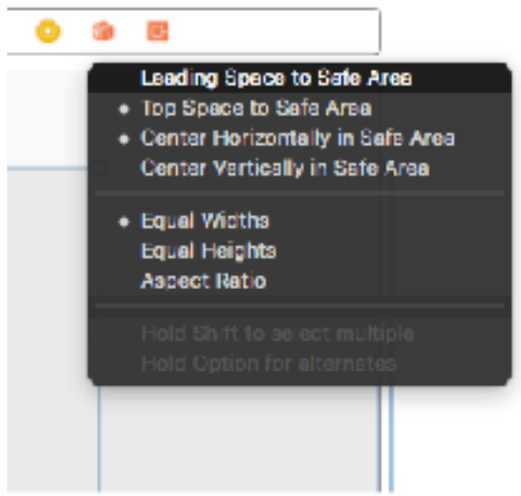
- Drag it into the ViewController and resize it to fit



Storyboard

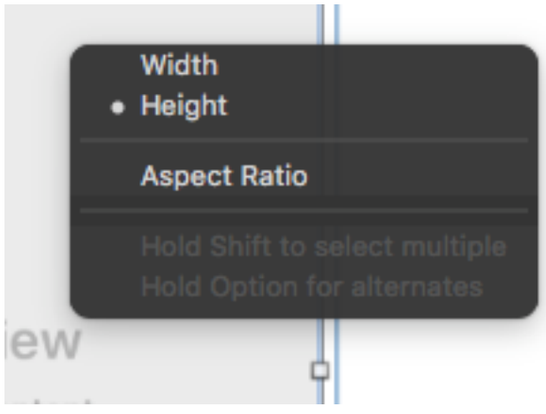
- Control + Click on the top of the ViewController

- Select “Top Space to Safe Area”, “Center Horizontally in Safe Area,” and “Equal Widths”



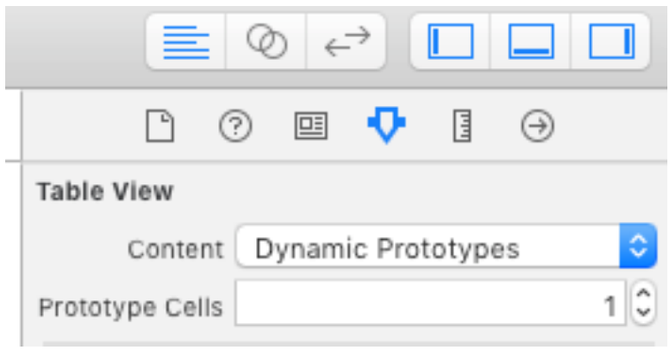
- Control + Click on the top of the TableView

- Select “Height”



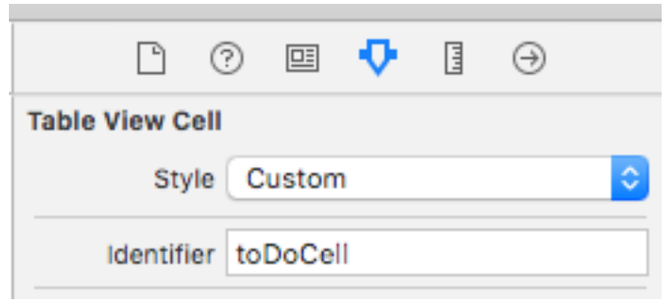
Storyboard

- With the TableView still selected, look at the menu on the right



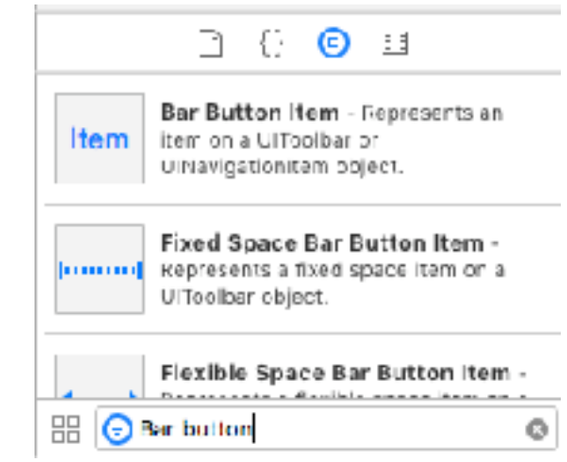
- Increment “Prototype Cells” to 1

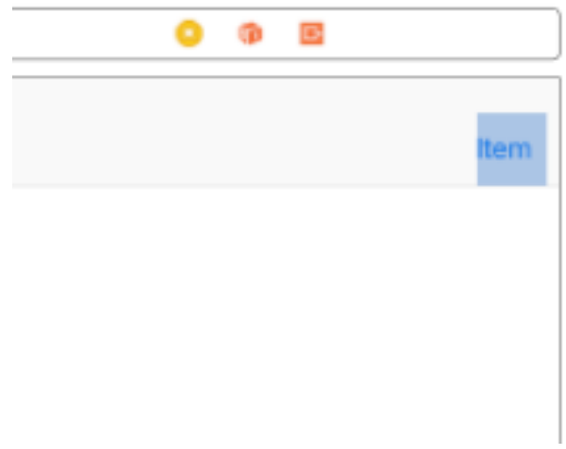
- Click on the cell displayed in the storyboard



- In the same menu on the right, change `Identifier` to "toDoCell"

Storyboard

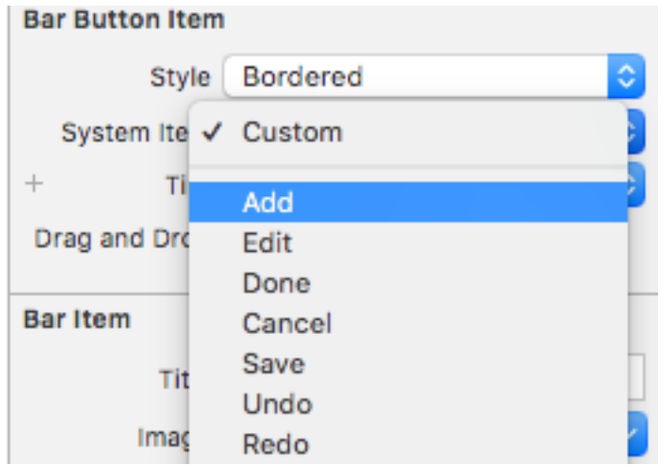




- Search for `Bar Button Item` in the menu in the bottom-right corner of Xcode

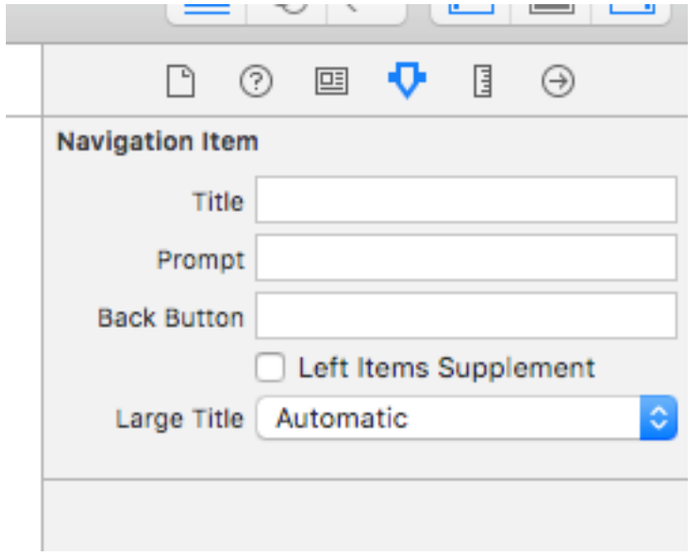
- Drag it to the right corner of the navigation bar

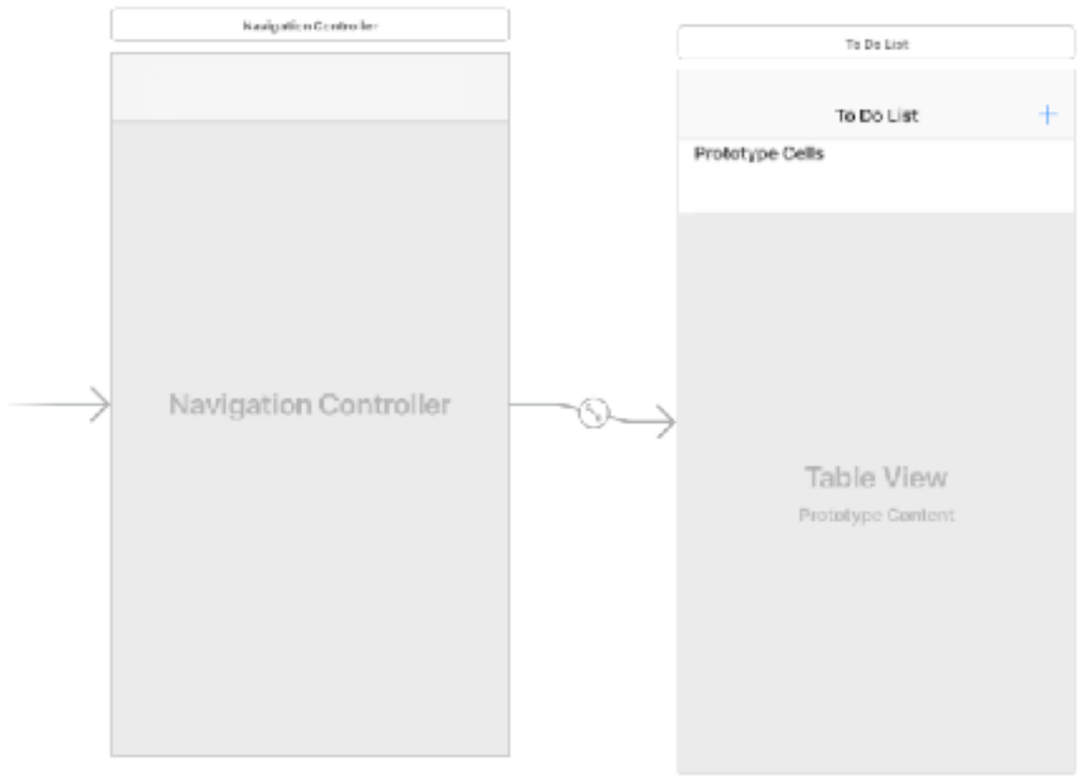
- Under `System Item` in the right-menu, choose `Add`



Storyboard

- Set a title for your ViewController by double-clicking on the navigation bar then setting a title in the upper-right corner of Xcode





Linking to code

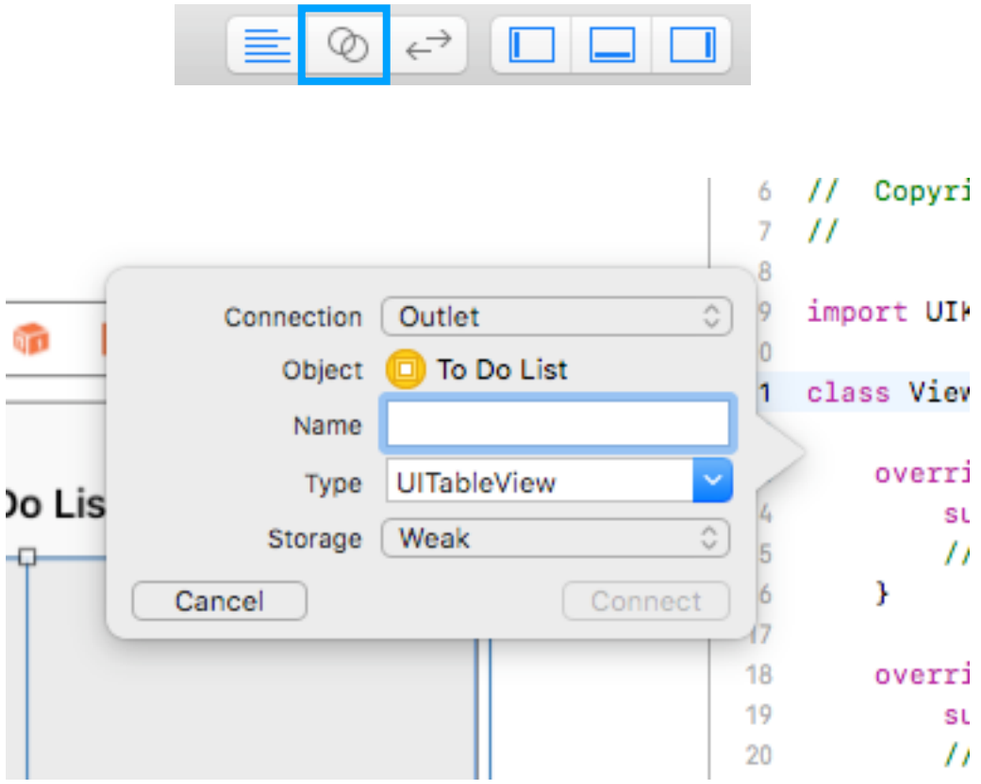
- Click the button with the intersecting circles in the upper-right corner of Xcode

- Control + drag from the TableView to a blank line under `class ViewController`

- Ensure `Connection` is set to `Outlet` and the type is `UITableView`

- Name the Outlet `tableView`

- This creates a variable in the class that is linked to the object in storyboard



Linking to code

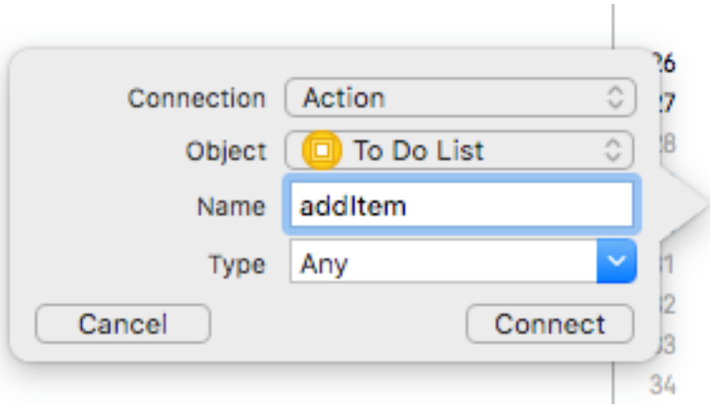
- control` + drag from the `+` button to a blank line at the end of the class but before `}`

- Set the connection type to `Action`

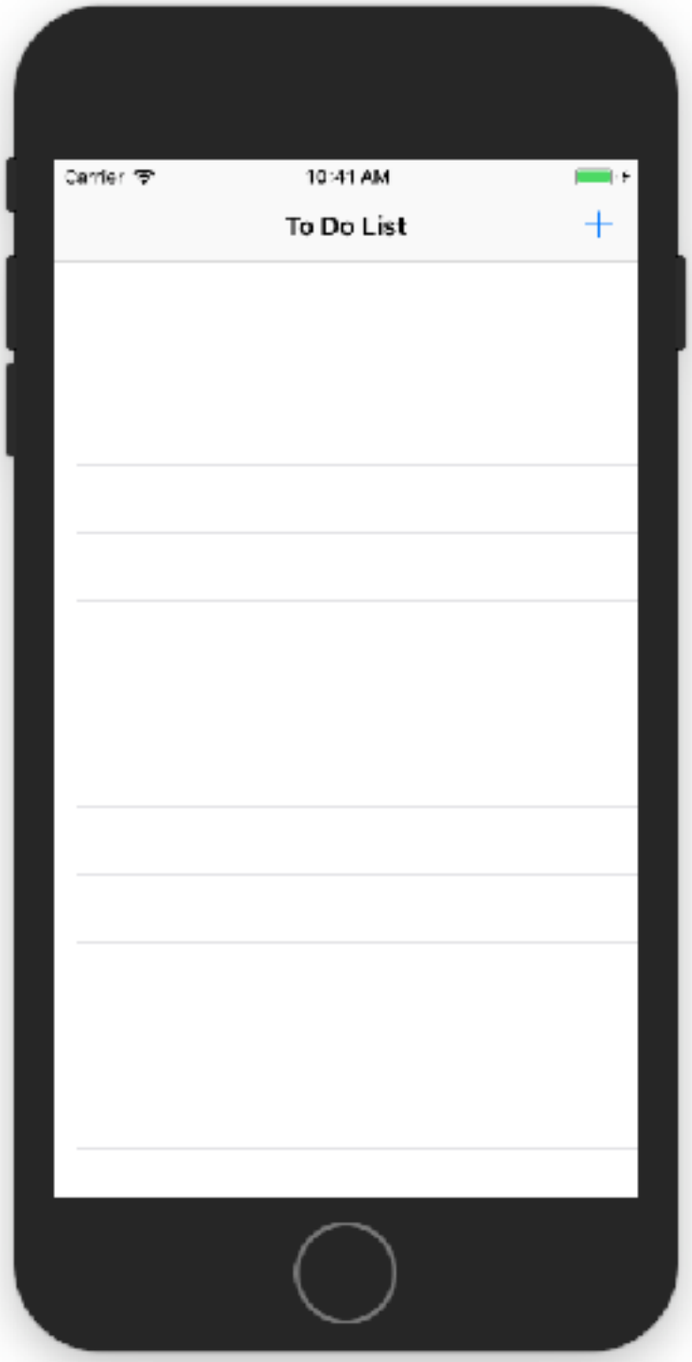
- Name it `addItem`

- This creates a function that is called when the `+` button is clicked

- Click the button to the left of the intersecting circles to return to storyboard



Try building it



Code!

- Navigate to `ViewController.swift`
- Add two parent classes to `class ViewController`; `UITableViewDataSource`, and `UITableViewDelegate`
- Note: You will get the error: `Type 'ViewController' does not conform to protocol 'UITableViewDataSource'` after this, ignore this error we will deal with it later.

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

    @IBOutlet weak var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func addItem(_ sender: Any) {
    }
}
```

Variables

- Each item in the list has two properties, a description, and whether or not that task has been completed or not.
- Structures in Swift are very simple, simply define it within the class

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

    struct toDoItem {
        var title: String
        var done: Bool
    }

    @IBOutlet weak var tableView: UITableView!

    /*
    ...
    */
}
```

Variables

- Each item in the list has two properties, a description, and whether or not that task has been completed or not. We will use a struct for this.

- We will use an array to store each of the items in our list

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

    struct toDoItem {
        var title: String
        var done: Bool
    }

    @IBOutlet weak var tableView: UITableView!
    var todoData: [toDoItem] = []

    /*
    ...
    */
}
```

Functions

- The UITableView classes ViewController now inherits from expect a number of functions. Add them:

```
class ViewController: UIViewController,  UITableViewDataSource, UITableViewDelegate {
    /*
    ...
    */
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func addItem(_ sender: Any) {
    }
    // tableView functions
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    }

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

    func numberOfSections(in tableView: UITableView) -> Int {
    }

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

    func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
    }

    func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
    }
}
```

viewDidLoad()

This is where all initial set up happens. Link the tableView to the class here.

```
// Do all your layout and setup in viewDidLoad()
override func viewDidLoad() {
    super.viewDidLoad() // handle all the setup for UIViewController

    // link the tableView to this class so that we can populate and control it
    tableView.delegate = self
    tableView.dataSource = self
}
```

didRecieveMemoryWarning()

This is used to handle memory issues, we shouldn't encounter any with this app. Ignore it for now

```
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}
```

addItem()

This is how a user will add an item to our list. When the `+` button is tapped, an alert with a text box will pop-up, and the user will enter their task in a `UITextField` within the pop-up

```
@IBAction func addItem(_ sender: Any) {
    let alert: UIAlertController = UIAlertController(title: "Add Item", message: "Add an item to your To-Do list",
preferredStyle: .alert) // creat a view for a pop-up

    // add a text field to read in the task
    alert.addTextField { (textField:UITextField) in
        textField.placeholder = "Item to add"
    }

    // create an empty action to handle cancelation
    let cancelAction = UIAlertAction(title: "Cancel", style: UIAlertActionStyle.destructive) { (action) in
        // Do nothing
    }

    alert.addAction(cancelAction) // add the action to the alert

    // create a confirmation action and handle adding something to the table
    let confirmAction = UIAlertAction(title: "Add", style: UIAlertActionStyle.default) { (action) in
        let itemField = alert.textFields![0] as UITextField // get the textfield we added to the alert
        if itemField.text != "" { // if the alert has text
            let itemToAdd = toDoItem(title: itemField.text!, done: false) // create a new toDoItem with the user-specified task
            self.tableView.beginUpdates() // start updating the tableView
            self.todoData.append(itemToAdd) // add the item to the list

            self.tableView.insertRows(at: [IndexPath(row: self.todoData.count-1, section: 0)], with: .top) // insert the new
item in the table
            self.tableView.endUpdates() // stop updating the tableView
        }
    }

    alert.addAction(confirmAction) // add the action to the alert

    self.present(alert, animated: true, completion: nil) // present the alert to the user
}
```

didSelectRowAt indexPath

This function is called whenever a user taps a row in your tableView

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    tableView.deselectRow(at: indexPath, animated: true) // deselect the row

    if indexPath.row < todoData.count { // if the row represents valid data
        todoData[indexPath.row].done = !todoData[indexPath.row].done // toggle the
'done' flag

        tableView.reloadRows(at: tableView.indexPathsForVisibleRows!,
with: .fade) // update the view
    }
}
```

cellForRowAt indexPath

Recall earlier we set an identifier for each cell in our tableView from storyboard. Here we will retrieve one of those cells, set it up, and return it to our tableView

We will access the variable in our `toDoData` array the given cell relates to, and populate it accordingly

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    // create a variable representing a cell in your table
    let cell = tableView.dequeueReusableCell(withIdentifier: "toDoCell", for: indexPath)

    if indexPath.row < todoData.count { // if it's valid
        let item = todoData[indexPath.row] // get the item in your array of tasks
        cell.textLabel?.text = item.title // set the label on the cell to represent the given
item

        // if the task is complete, set the accessory to a check mark, else do not
        if item.done {
            cell.accessoryType = .checkmark
        } else {
            cell.accessoryType = .none
        }
    }

    return cell // return the styled cell
}
```

numberOfSections

This returns the number of sections in the tableView. We only want 1 section but an example of an app that uses multiple sections is the Settings App on iPhone and iPad

```
func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}
```

numberOfRowsInSection

This simply returns the number of cells in our tableView, since we have only 1 section. This is going to be the number of elements in our array.

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

canEditRowAt

Since we want the user to be able to remove rows in the table, we want to set editing to true

```
func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
    return true
}
```

commit editingStyle

This function handles the editing of rows in the tableView, in our case only deletion

```
func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt
indexPath: IndexPath) {
    if editingStyle == .delete { // if the editing style is deletion
        tableView.beginUpdates() // begin updating the tableView
        todoData.remove(at: indexPath.row) // remove the item from our array
        tableView.deleteRows(at: [indexPath], with: .top) // delete that row in the table
        tableView.endUpdates() // stop updating the tableView
    }
}
```

You’re done!

Try building it!