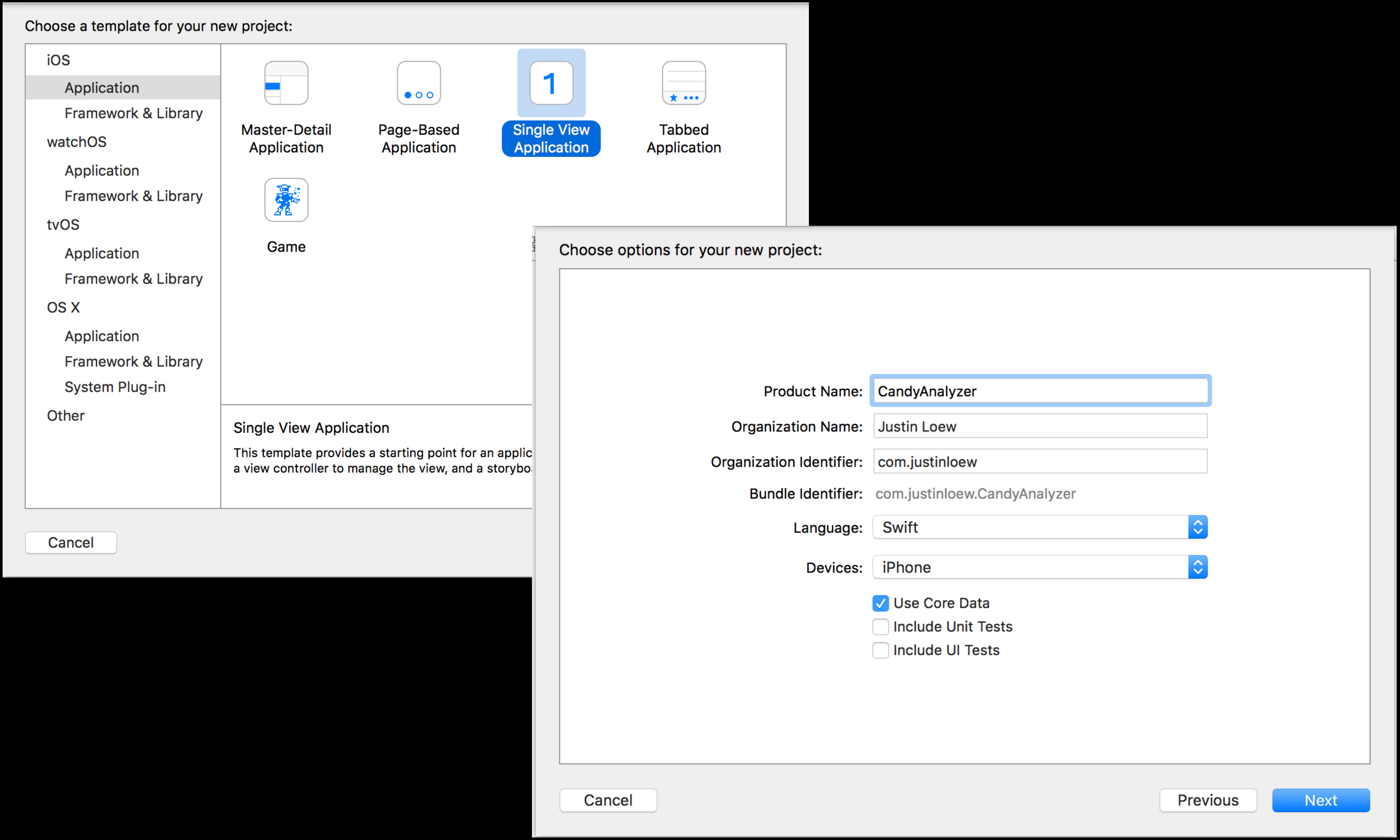
Candy Analyzer

Core Data
Made with Xcode 7.1 and iOS 9
By Justin Loew


Choose a template for your new project:

Single View Application

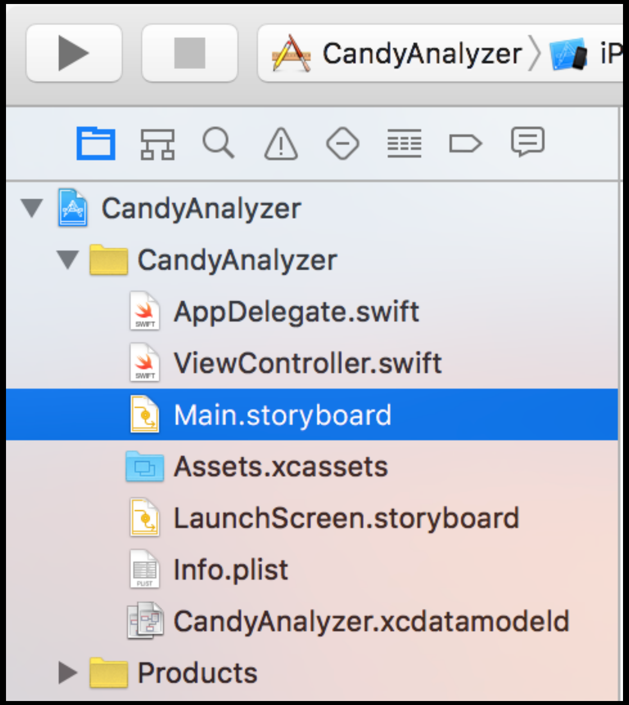
This template provides a starting point for an applic
a view controller to manage the view, and a storybo

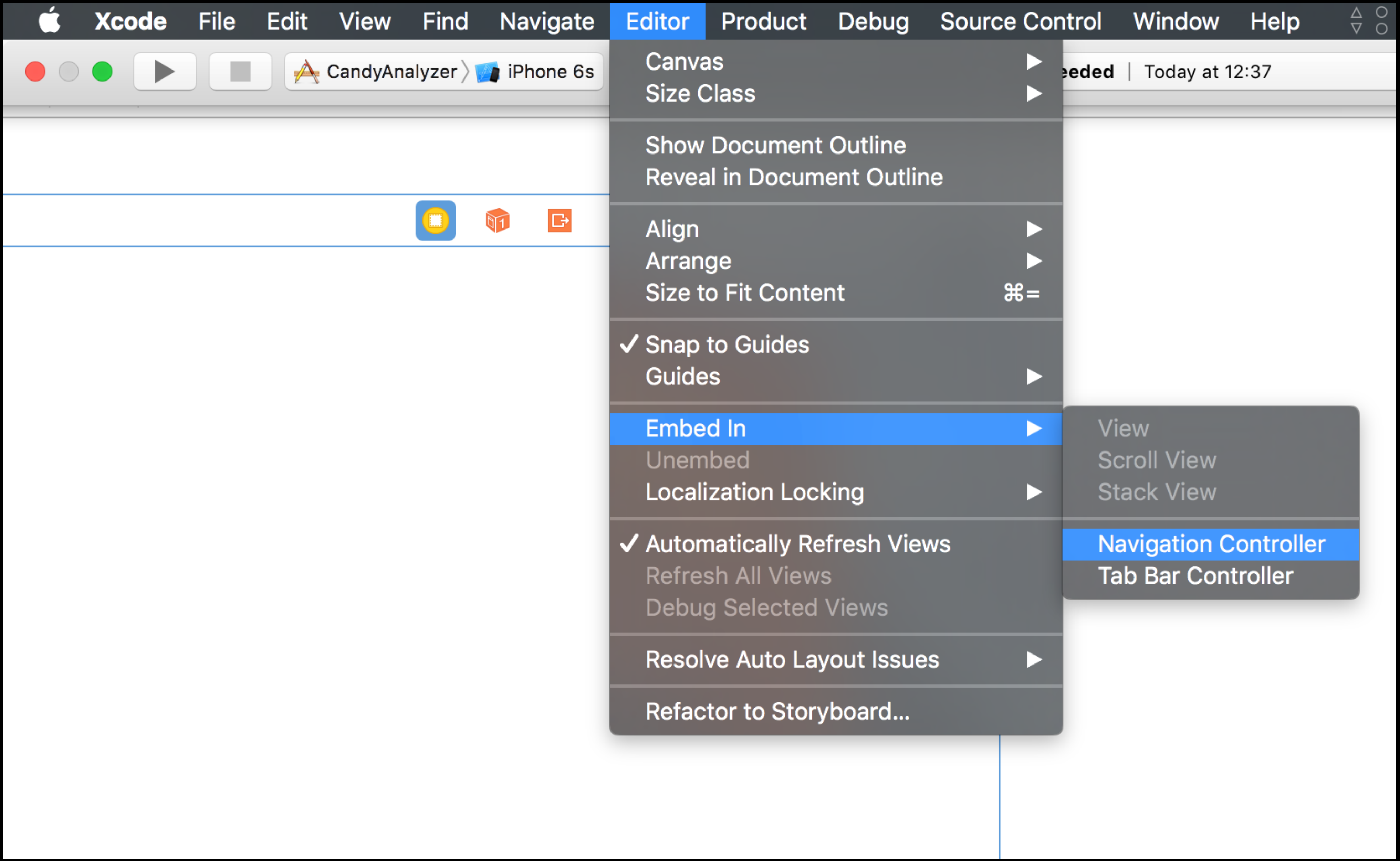
Choose options for your new project:

Product Name:	CandyAnalyzer
Organization Name:	Justin Loew
Organization Identifier:	com.justinloew
Bundle Identifier:	com.justinloew.CandyAnalyzer
Language:	Swift
Devices:	iPhone

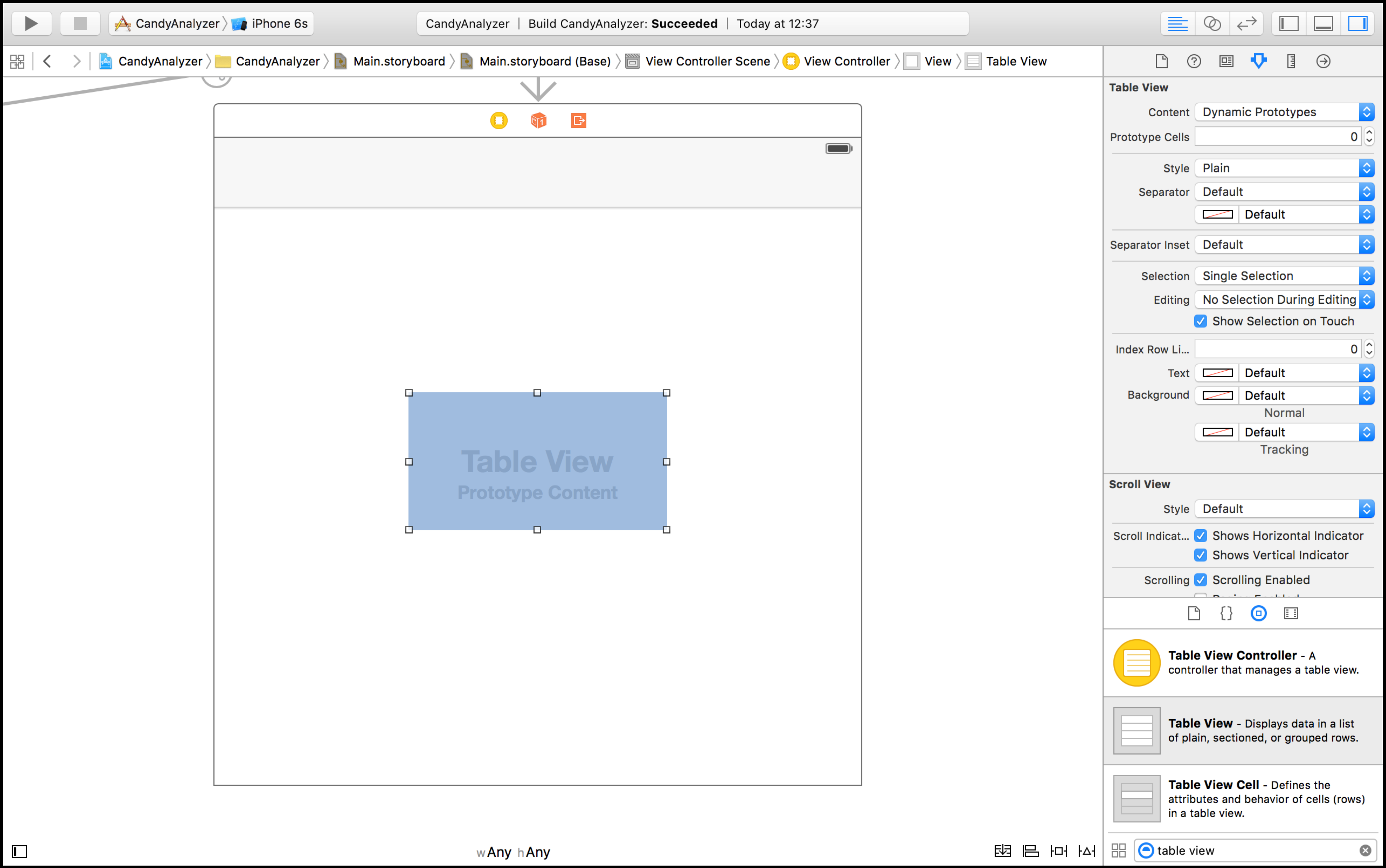
- [x] Use Core Data
- [ ] Include Unit Tests
- [ ] Include UI Tests

Create a new Xcode project. Single View Application, called `CandyAnalyzer`. Make sure `Use Core Data` is checked.





Open up `Main.storyboard`. Select the view controller by its yellow dot, and click `Editor > Embed In > Navigation Controller`.



CandyAnalyzer 〉 iPhone 6s | CandyAnalyzer | Build CandyAnalyzer: **Succeeded** | Today at 12:37

CandyAnalyzer 〉 CandyAnalyzer 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View 〉 Table View

Table View

Content: Dynamic Prototypes
Prototype Cells: 0
Style: Plain
Separator: Default
Default
Separator Inset: Default
Selection: Single Selection
Editing: No Selection During Editing
✓ Show Selection on Touch
Index Row Li...: 0
Text: Default
Background: Default
Normal
Default
Tracking

Scroll View

Style: Default
Scroll Indicat...: ✓ Shows Horizontal Indicator
✓ Shows Vertical Indicator
Scrolling: ✓ Scrolling Enabled

Table View Controller - A controller that manages a table view.

Table View - Displays data in a list of plain, sectioned, or grouped rows.

Table View Cell - Defines the attributes and behavior of cells (rows) in a table view.

table view

From the Objects pane in the bottom right, drag a table view into our view controller.

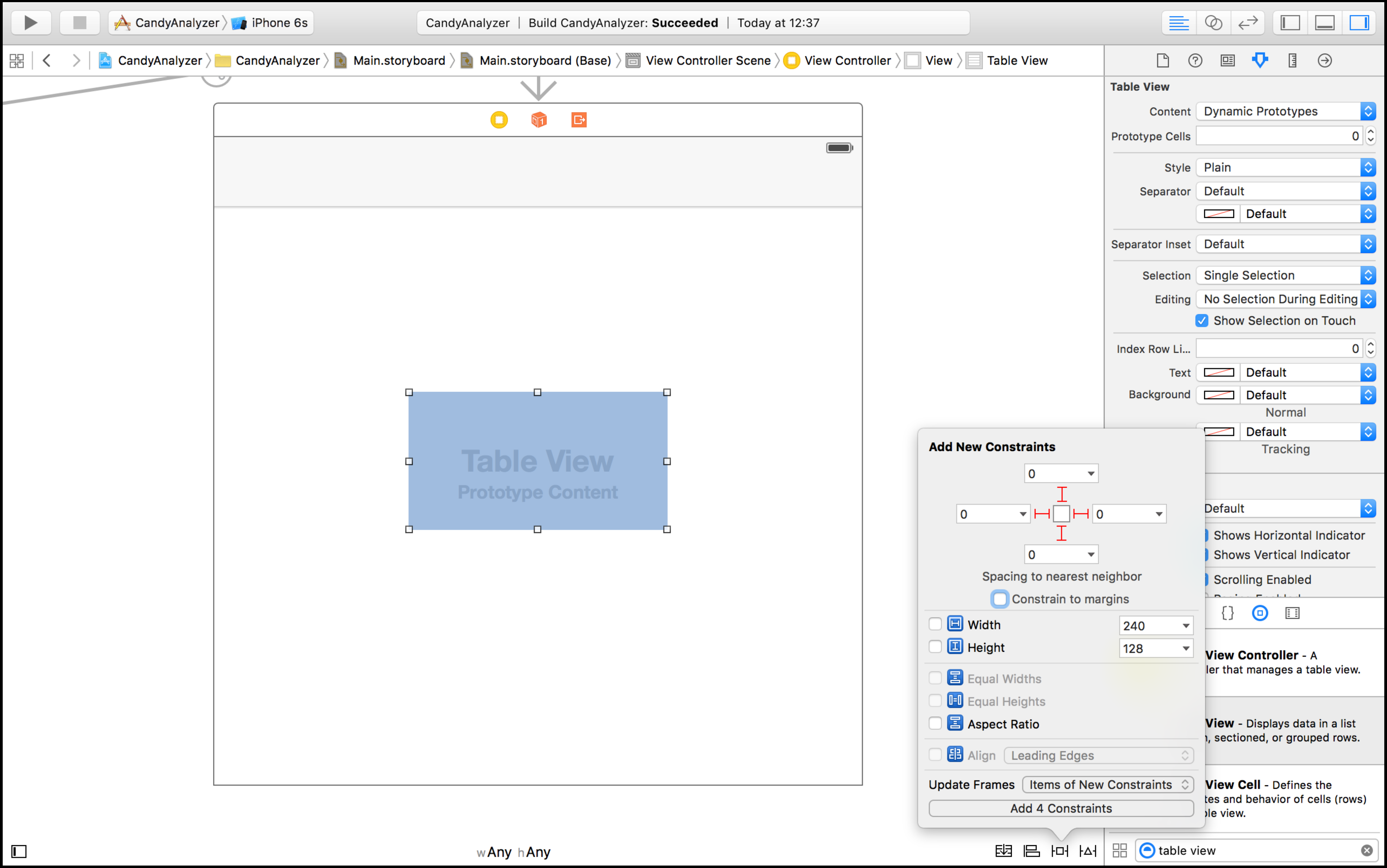

CandyAnalyzer 〉 iPhone 6s
CandyAnalyzer | Build CandyAnalyzer: Succeeded | Today at 12:37
CandyAnalyzer 〉 CandyAnalyzer 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View 〉 Table View
Table View
Prototype Content
wAny hAny
table view

Table View

Content	Dynamic Prototypes
Prototype Cells	0
Style	Plain
Separator	Default
	Default
Separator Inset	Default
Selection	Single Selection
Editing	No Selection During Editing
	✓ Show Selection on Touch
Index Row Li...	0
Text	Default
Background	Default
	Normal
	Default
	Tracking

Default
Shows Horizontal Indicator
Shows Vertical Indicator
Scrolling Enabled

Add New Constraints

Top	0
Left	0
Right	0
Bottom	0

Spacing to nearest neighbor
☐ Constrain to margins

☐	Width	240
☐	Height	128
☐	Equal Widths	
☐	Equal Heights	
☐	Aspect Ratio	
☐	Align	Leading Edges

Update Frames: Items of New Constraints

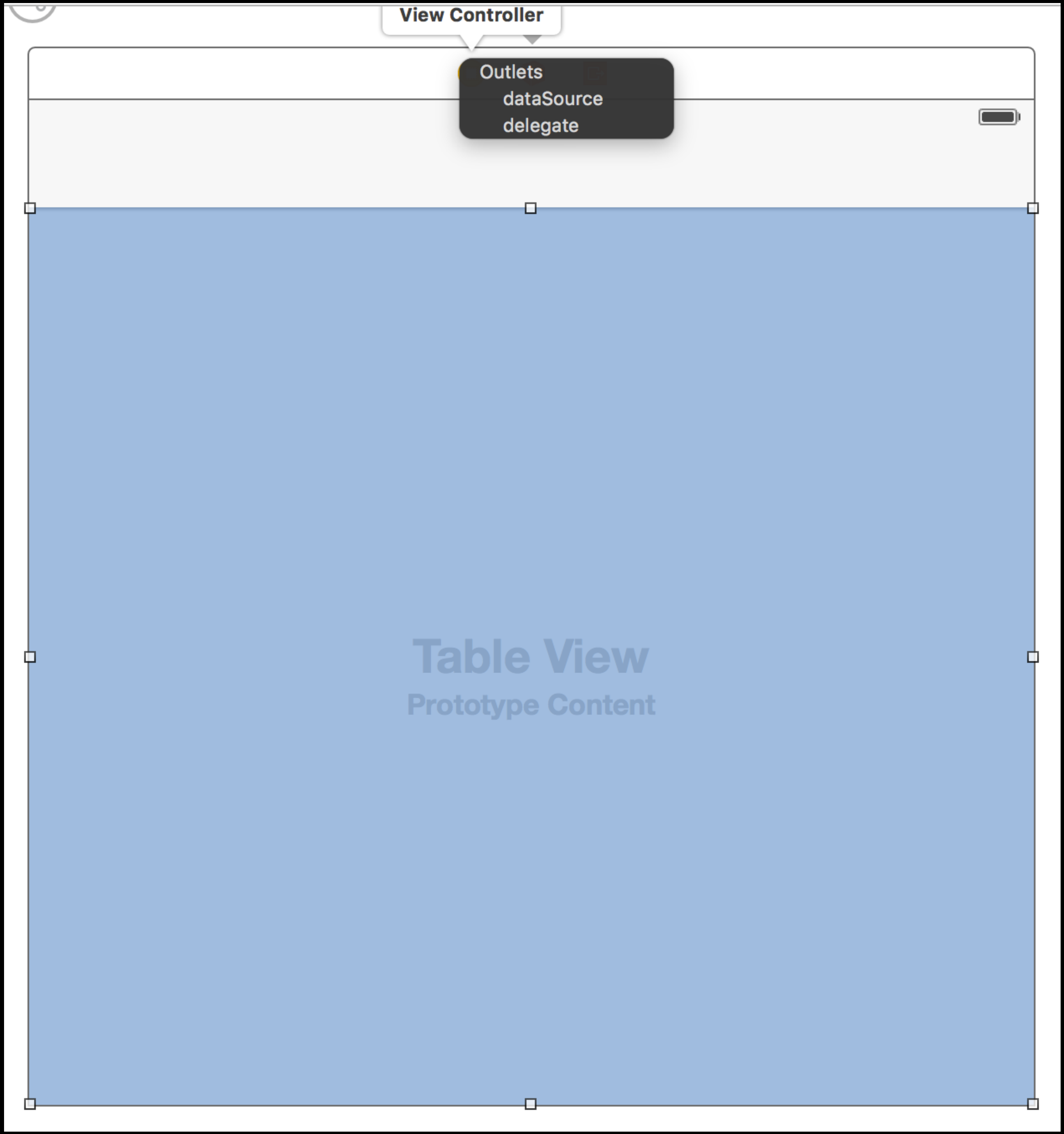
Add 4 Constraints

View Controller - A ler that manages a table view.

View - Displays data in a list , sectioned, or grouped rows.

View Cell - Defines the tes and behavior of cells (rows) le view.

Let’s make the table view fill the screen. With the table view selected, click the square button in the bottom right and add the constraints shown here.



It should now fill the screen. Control-drag from the table view to the yellow view controller circle, and select both `dataSource` and `delegate`.

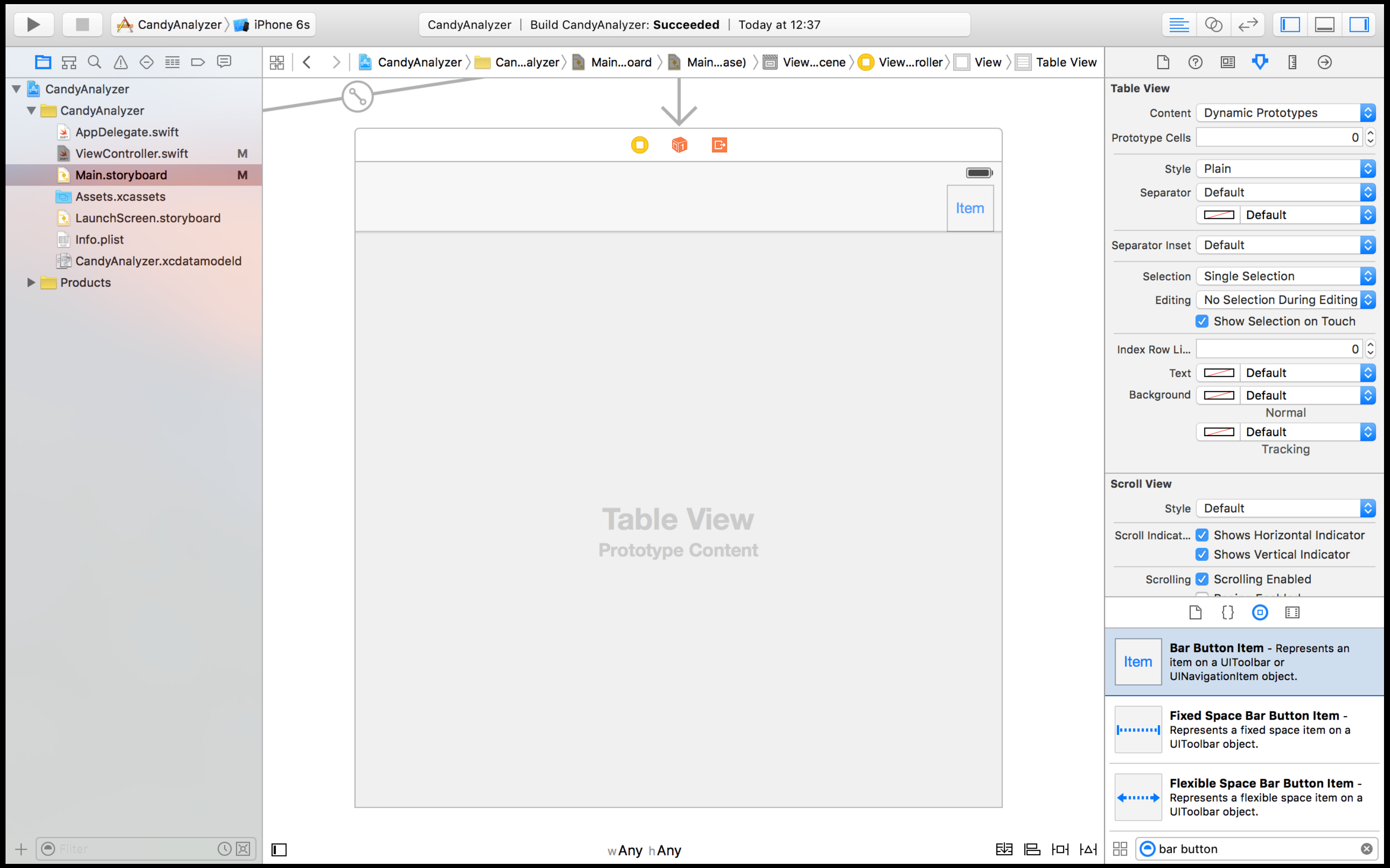


Table View

Content	Dynamic Prototypes
Prototype Cells	0
Style	Plain
Separator	Default
	Default
Separator Inset	Default
Selection	Single Selection
Editing	No Selection During Editing
	✓ Show Selection on Touch
Index Row Li...	0
Text	Default
Background	Default
	Normal
	Default
	Tracking

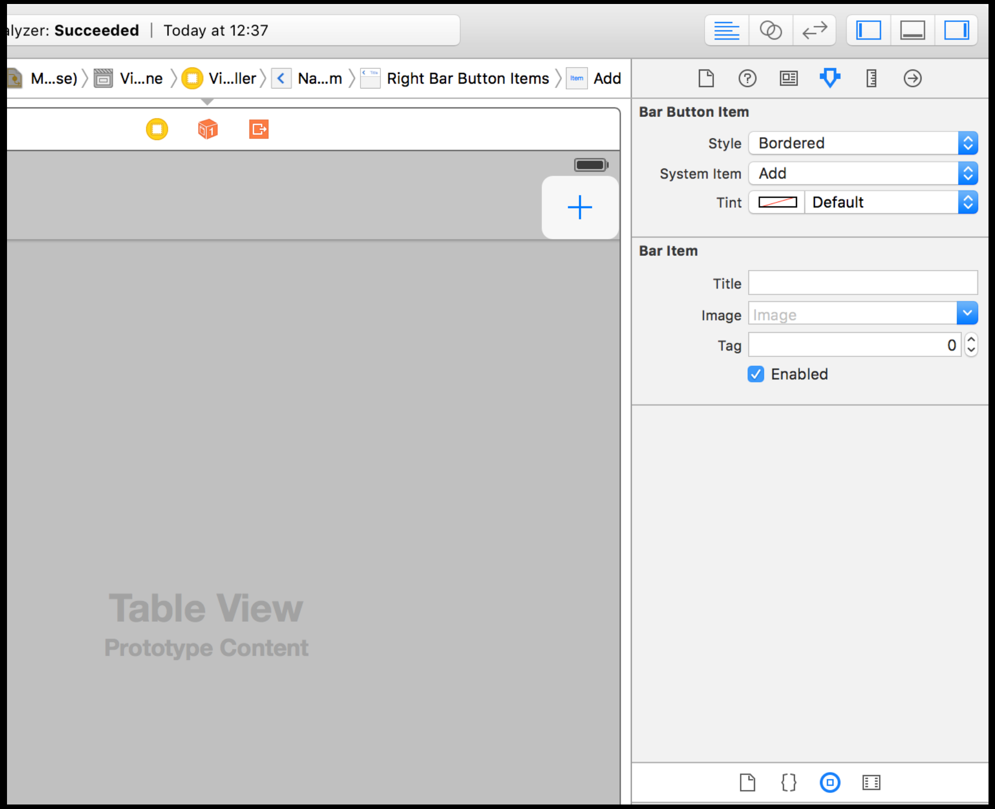
Scroll View

Style	Default
Scroll Indicat...	✓ Shows Horizontal Indicator
	✓ Shows Vertical Indicator
Scrolling	✓ Scrolling Enabled

Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.

Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.

Flexible Space Bar Button Item - Represents a flexible space item on a UIToolbar object.


lyzer: Succeeded | Today at 12:37
M...se) 〉 Vi...ne 〉 Vi...ller 〉 Na...m 〉 Right Bar Button Items 〉 Add
+
Table View
Prototype Content

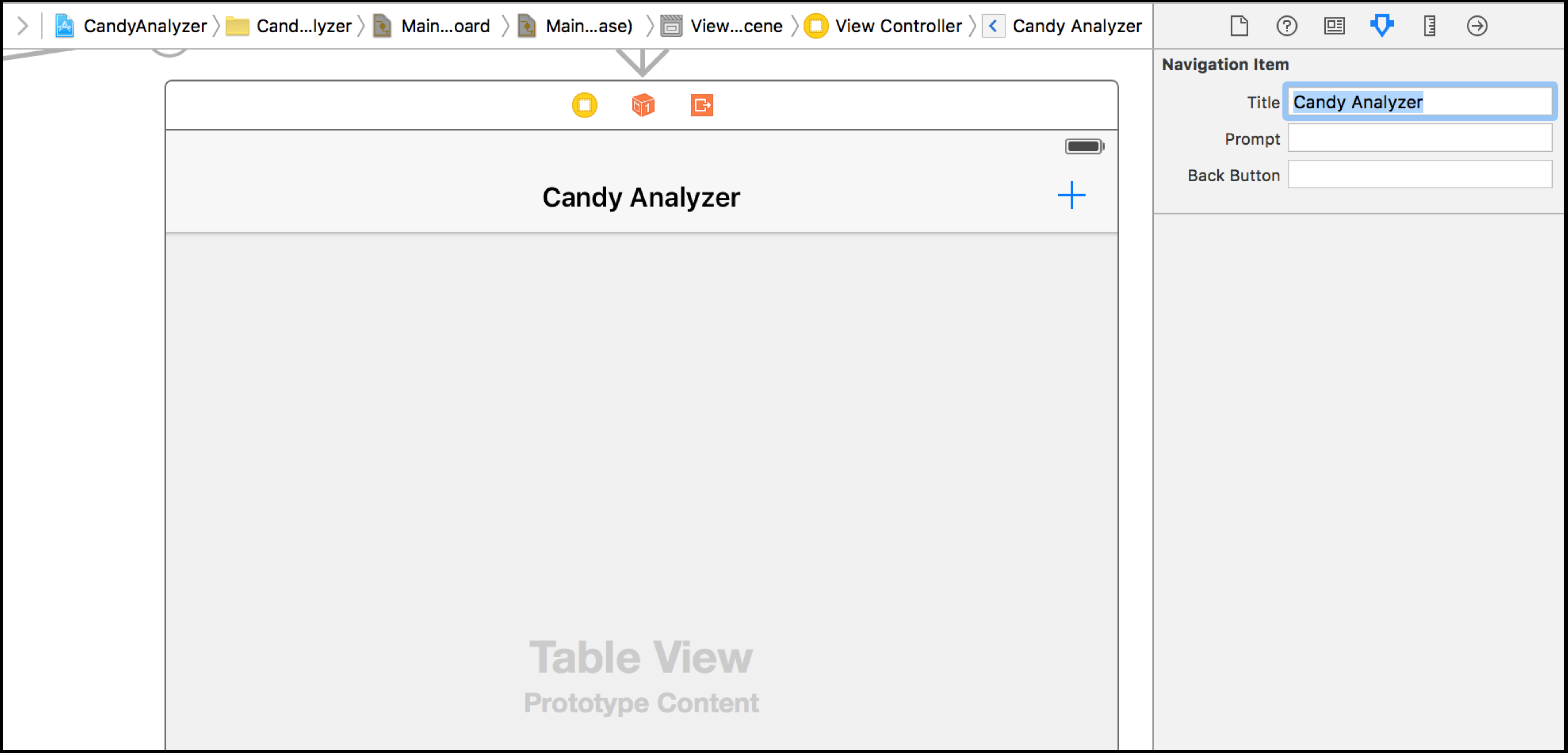
Bar Button Item

Style	Bordered
System Item	Add
Tint	Default

Bar Item

Title	
Image	Image
Tag	0
	✓ Enabled

Now, drag a `Bar Button Item` out onto the navigation bar. In the attributes inspector (the 4th tab in the right panel), change its `System Item` to **Add**.



Navigation Item

Title: Candy Analyzer

Prompt:

Back Button:

Select the top navigation bar itself, and change its `Title` to `Candy Analyzer`.

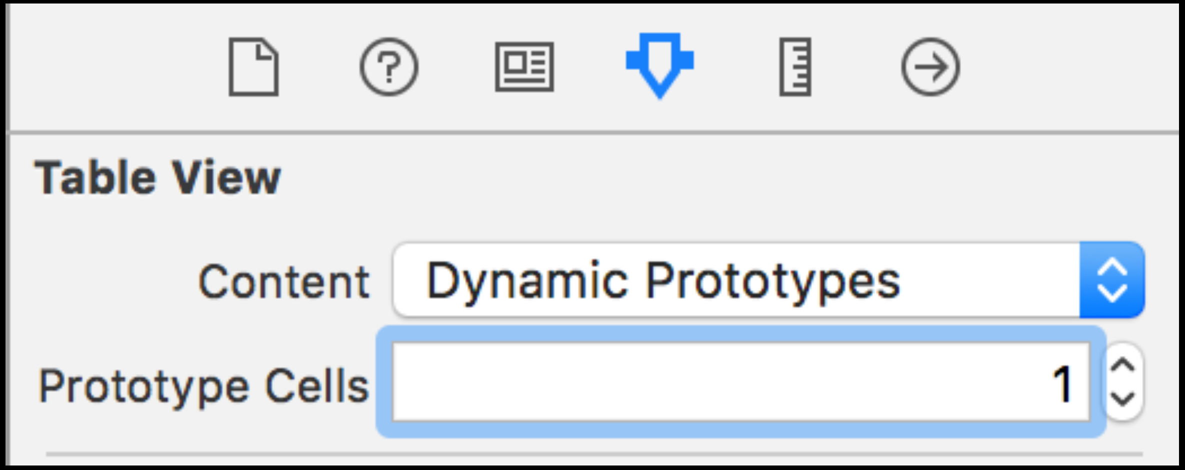


Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Select the table view again. In the attributes inspector, change its `Prototype Cells` from `0` to `1`.

Select the cell that just appeared, and change its `Style` to `Right Detail`. Change its `Identifier` to `Cell`.

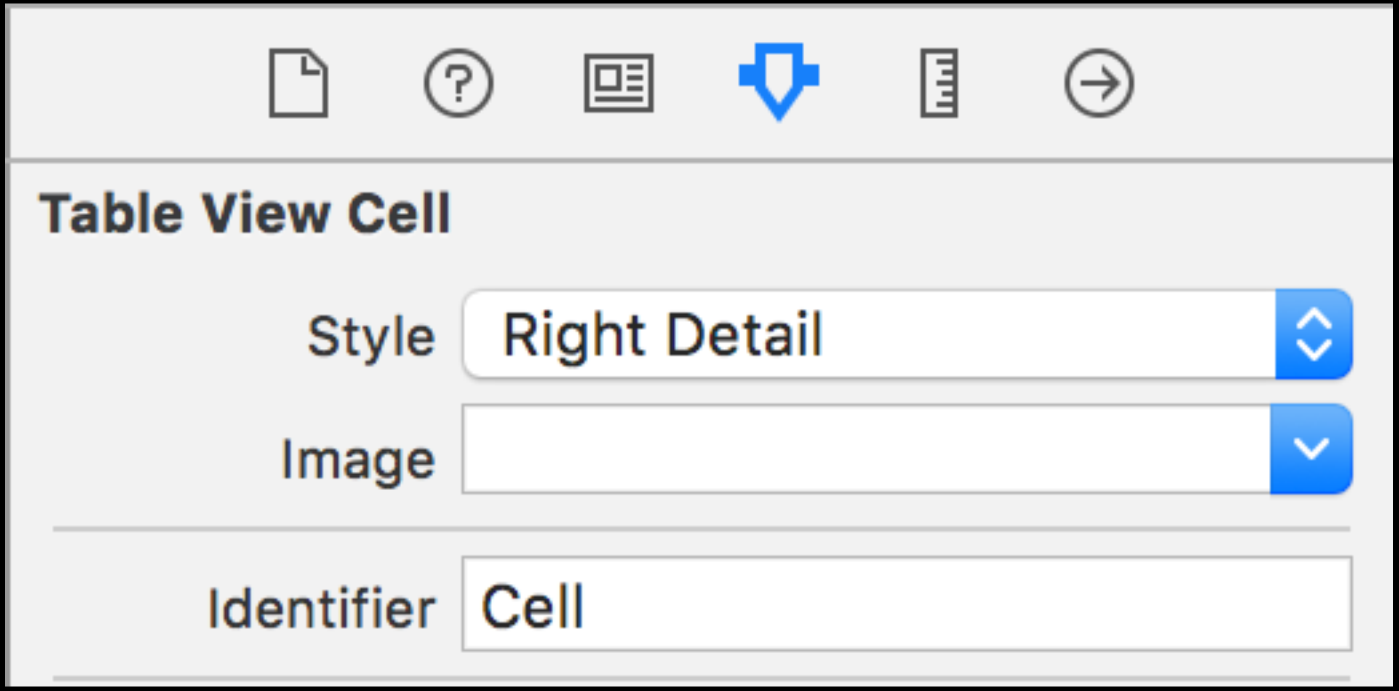
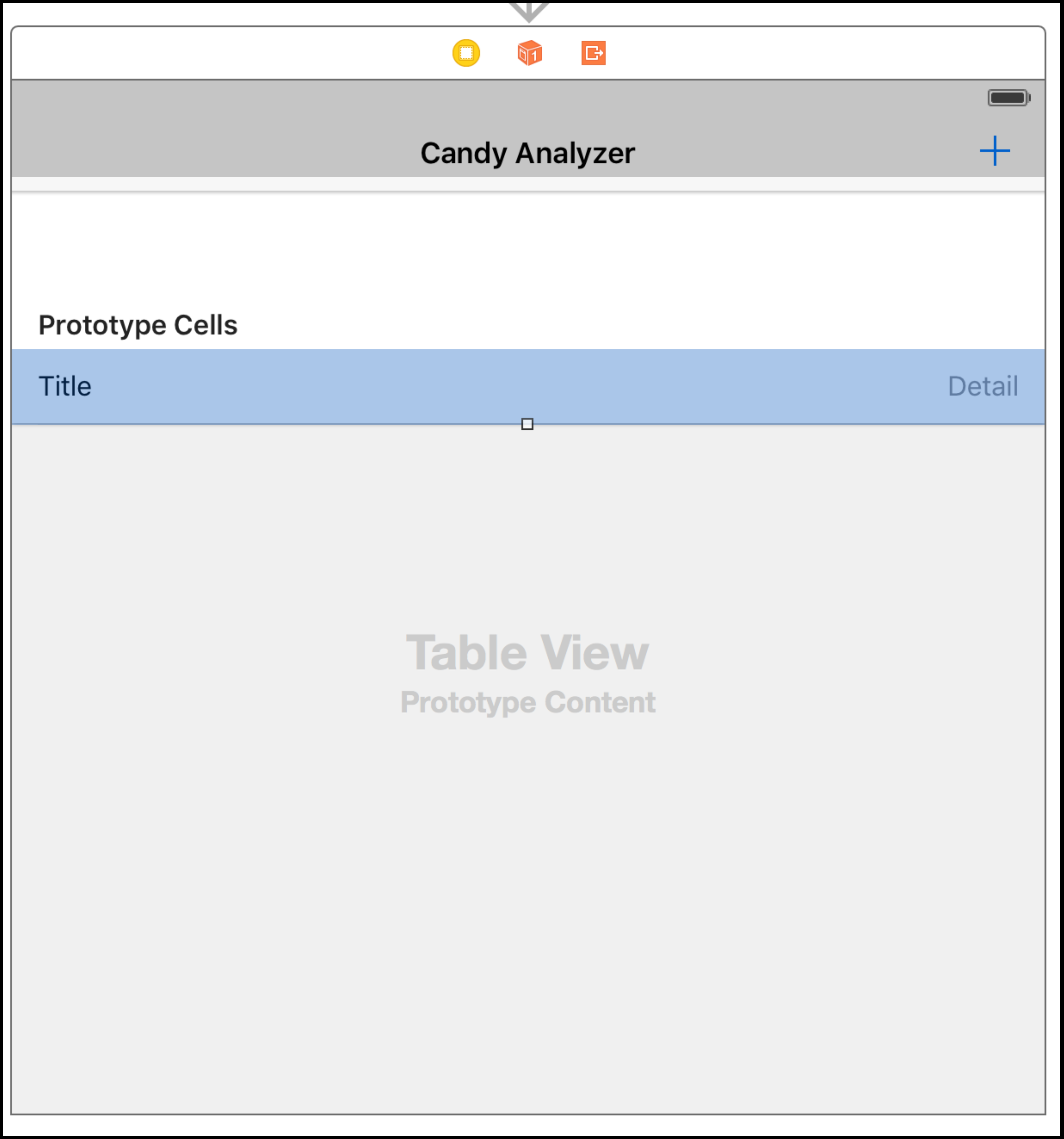


Table View Cell

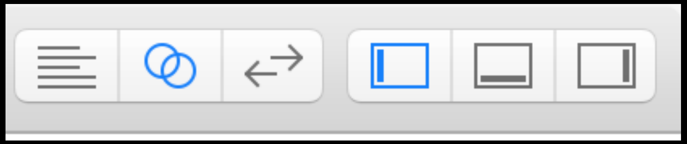
Style: Right Detail

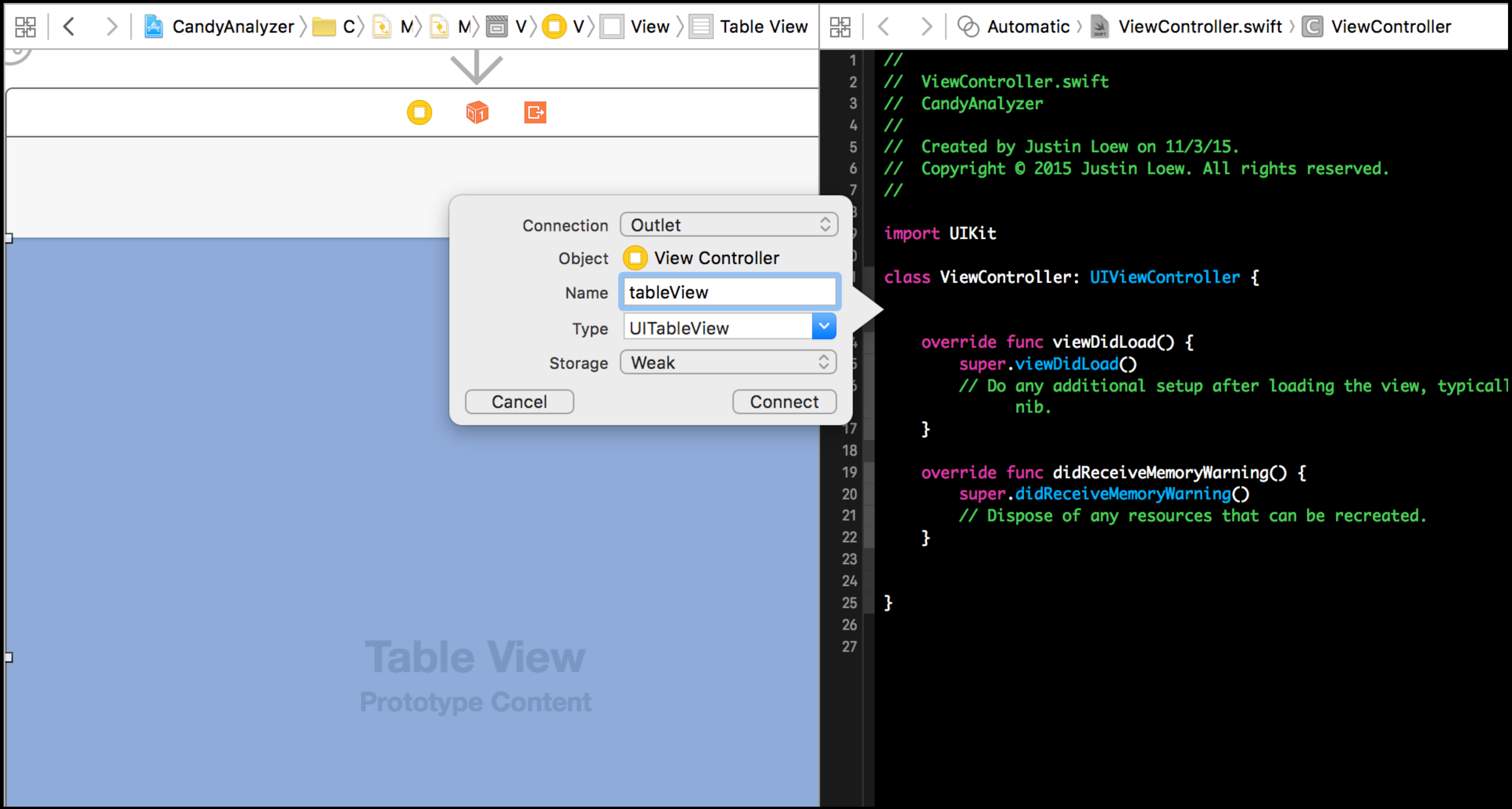
Image:

Identifier: Cell



Your storyboard should now look like this.

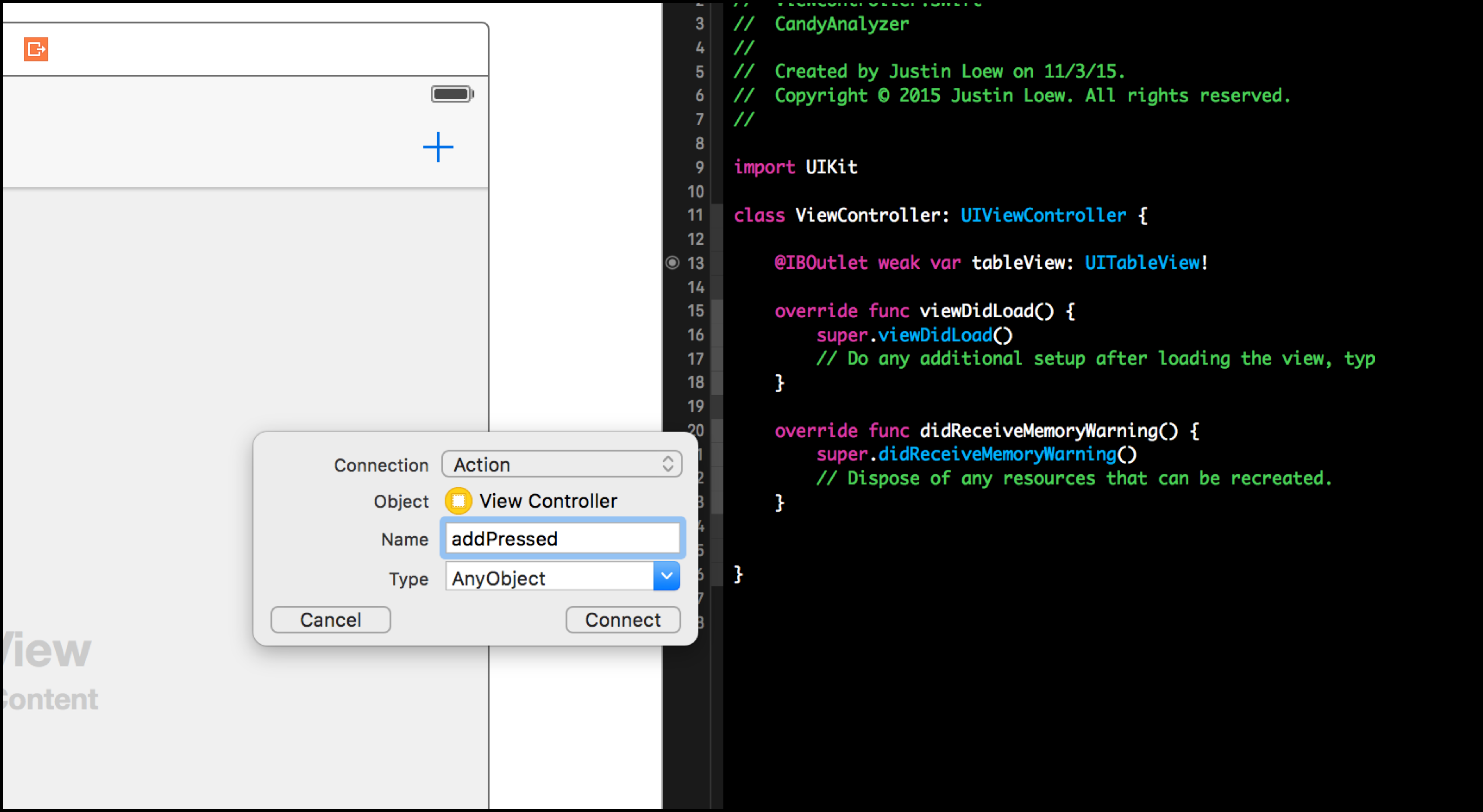



CandyAnalyzer 〉 C 〉 M 〉 M 〉 V 〉 V 〉 View 〉 Table View
Automatic 〉 ViewController.swift 〉 ViewController
Table View
Prototype Content
Cancel
Connect

Connection: Outlet
Object: View Controller
Name: tableView
Type: UITableView
Storage: Weak

```
//
//  ViewController.swift
//  CandyAnalyzer
//
//  Created by Justin Loew on 11/3/15.
//  Copyright © 2015 Justin Loew. All rights reserved.
//

import UIKit

class ViewController: UIViewController {


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

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


}
```

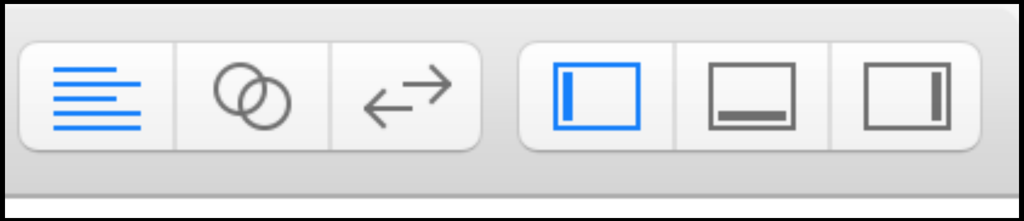
Open the assistant editor (the linked rings in the top right). Control-drag from the table view into the **`ViewController`** class, just above **`viewDidLoad`**. Create an **`Outlet`** of type **`UITableView`** called **`tableView`**.

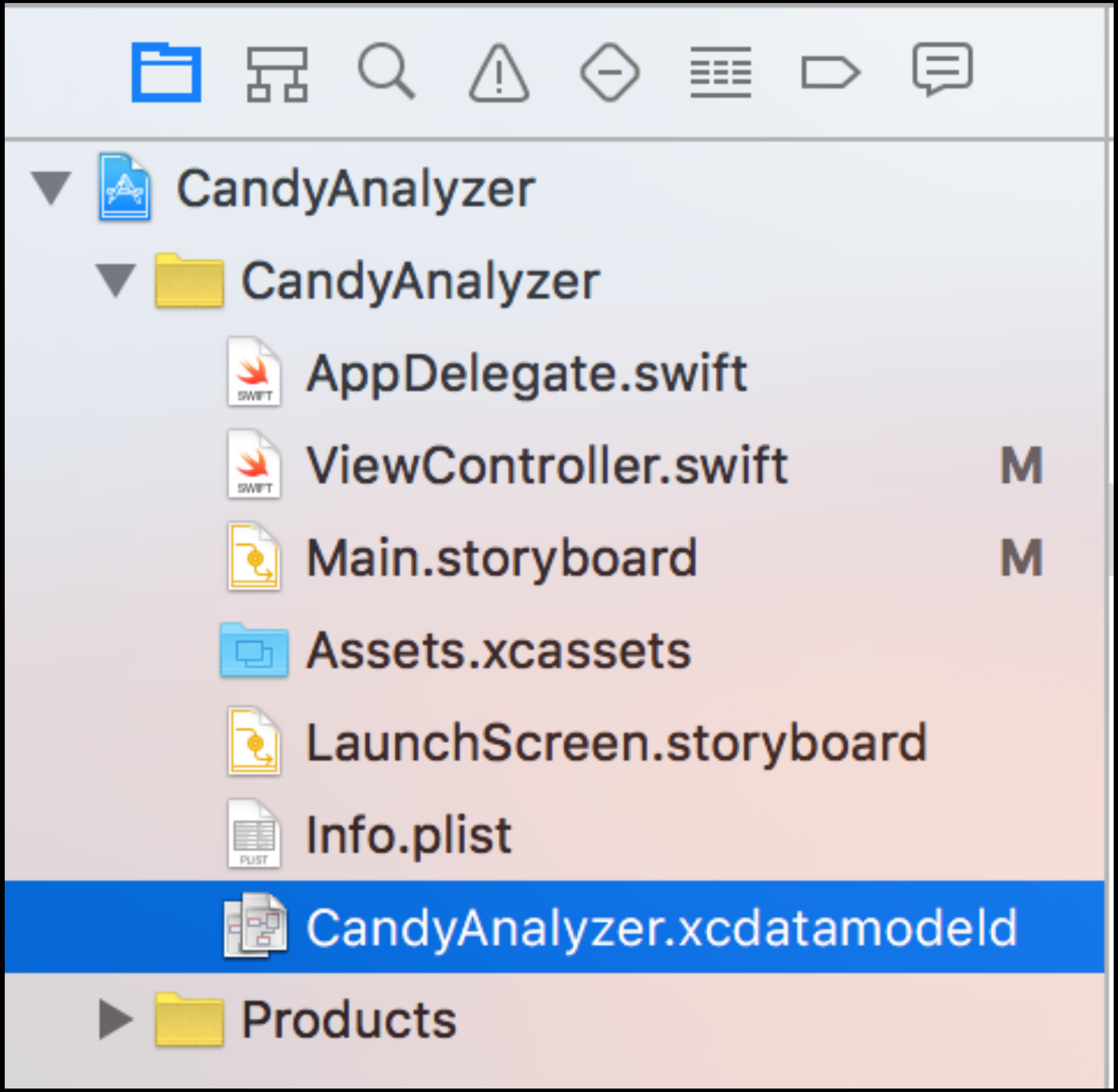


Connection: Action
Object: View Controller
Name: addPressed
Type: AnyObject

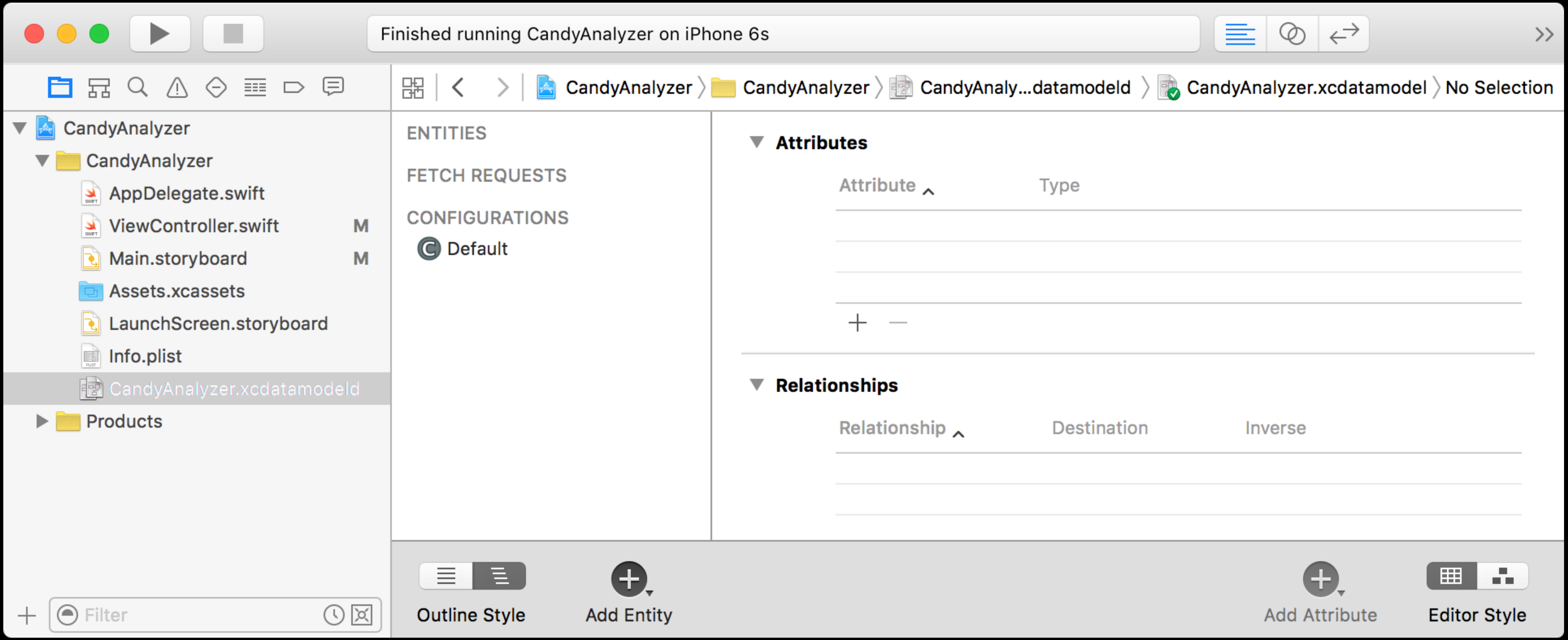
```swift
//  CandyAnalyzer
//
//  Created by Justin Loew on 11/3/15.
//  Copyright © 2015 Justin Loew. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typ
    }

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


}
```

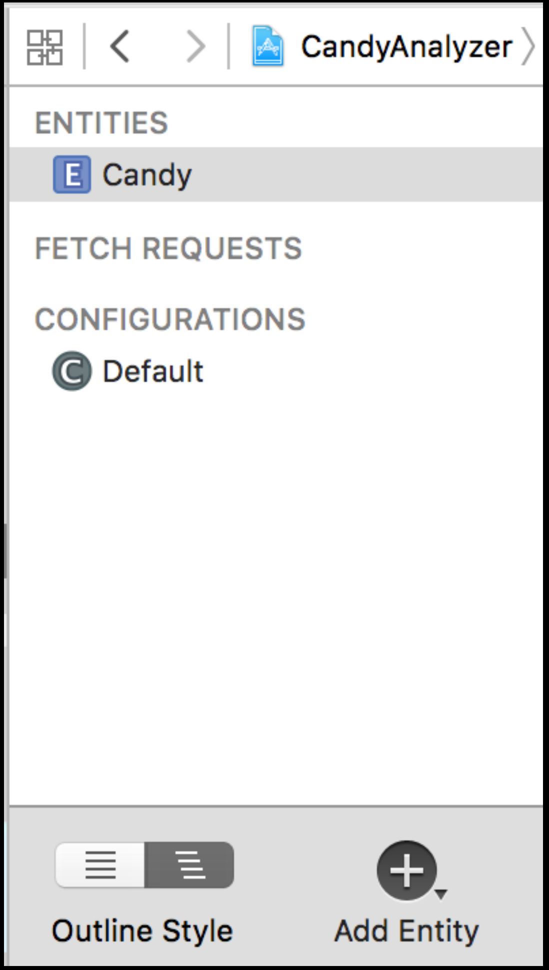
Control-drag from the + button to below `didReceiveMemoryWarning` to create an `Action` whose **Name** is `addPressed`.





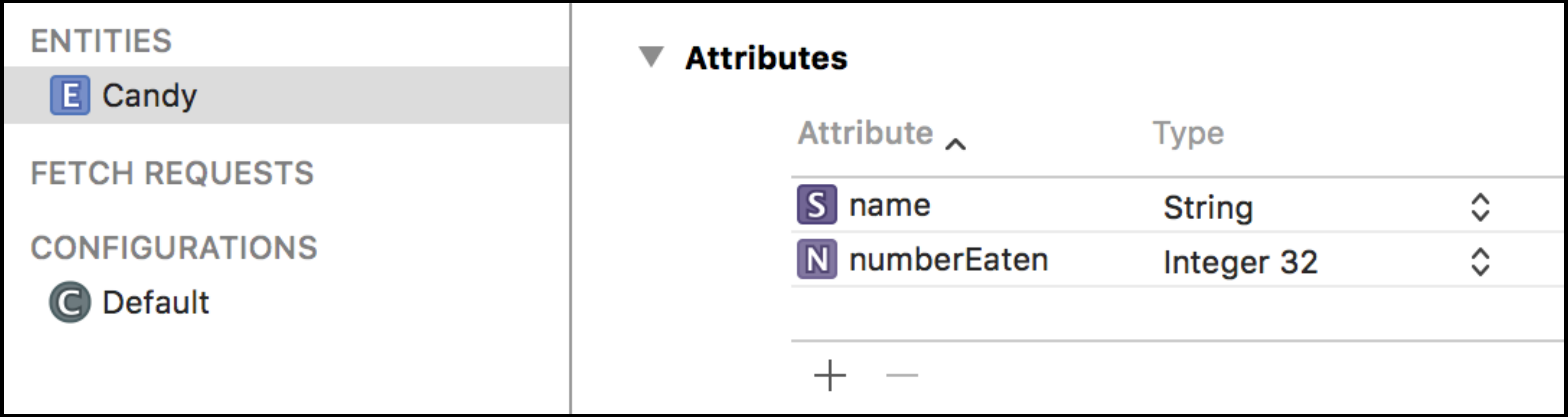
Switch back to the normal editor, and open `CandyAnalyzer.xcdatamodeld`. We're going to set up our Core Data model.





Choose the nested outline style in the bottom left and the grid editor style in the bottom right. Click `Add Entity` and rename it to `Candy`.

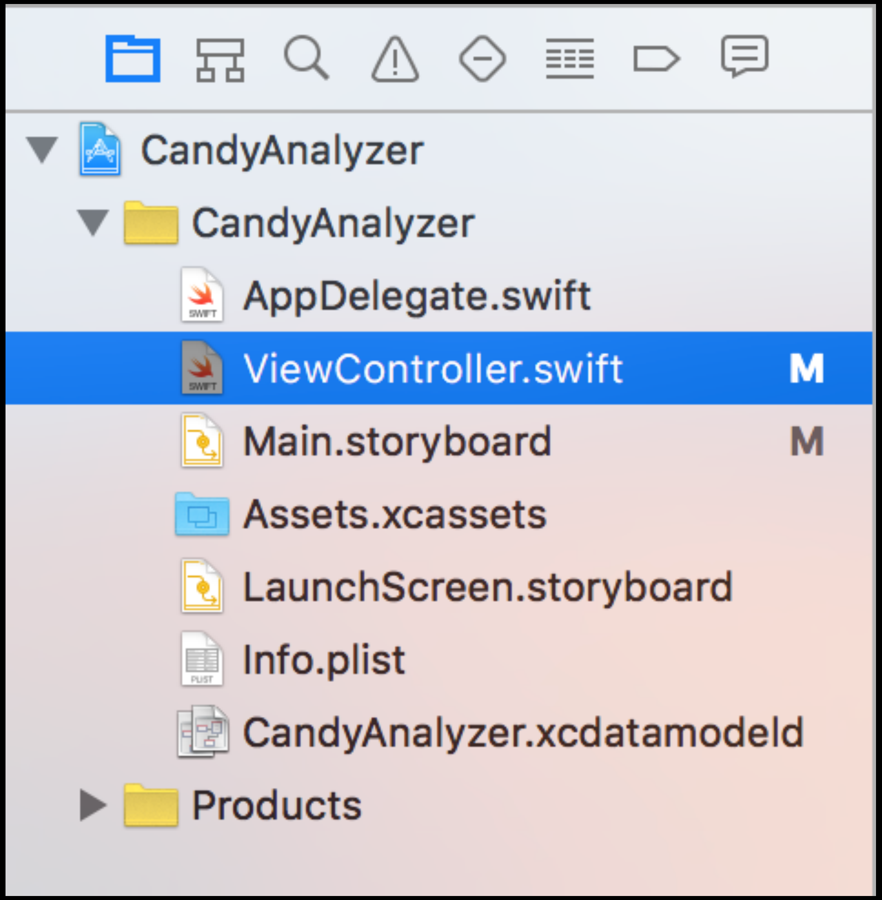
An entity is kind of like a class. It’s got different attributes (properties) that describe it.
We just have to tell Core Data what we want our entity to have in it, and Core Data will take care of figuring out how to save our entities into its database.

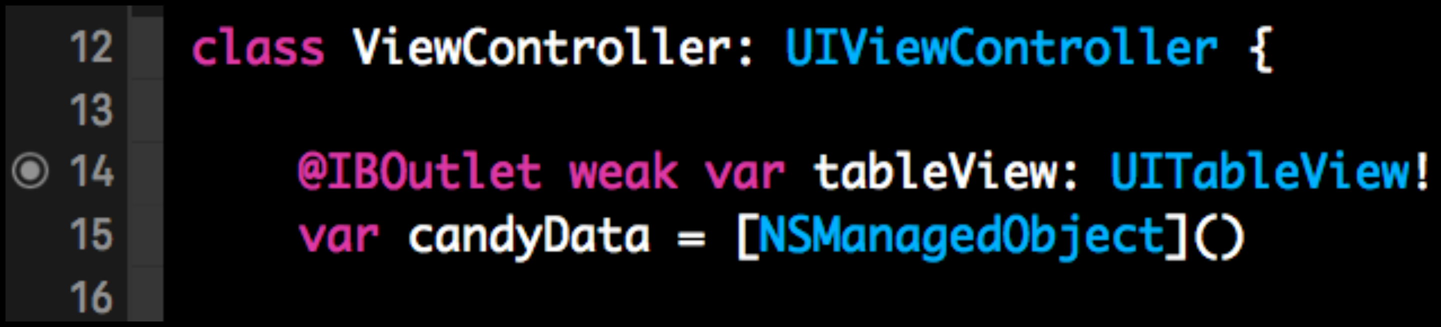


Attribute	Type
name	String
numberEaten	Integer 32

Our Candy entity will have two attributes:

- A name, to describing what kind of candy it is, and
- A number, to describe how many of this type of candy we've eaten

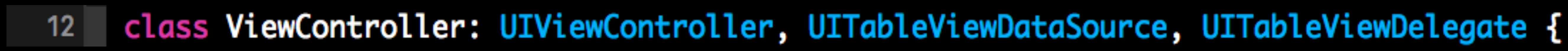




```
class ViewController: UIViewController {

    @IBOutlet weak var tableView: UITableView!
    var candyData = [NSManagedObject]()

```

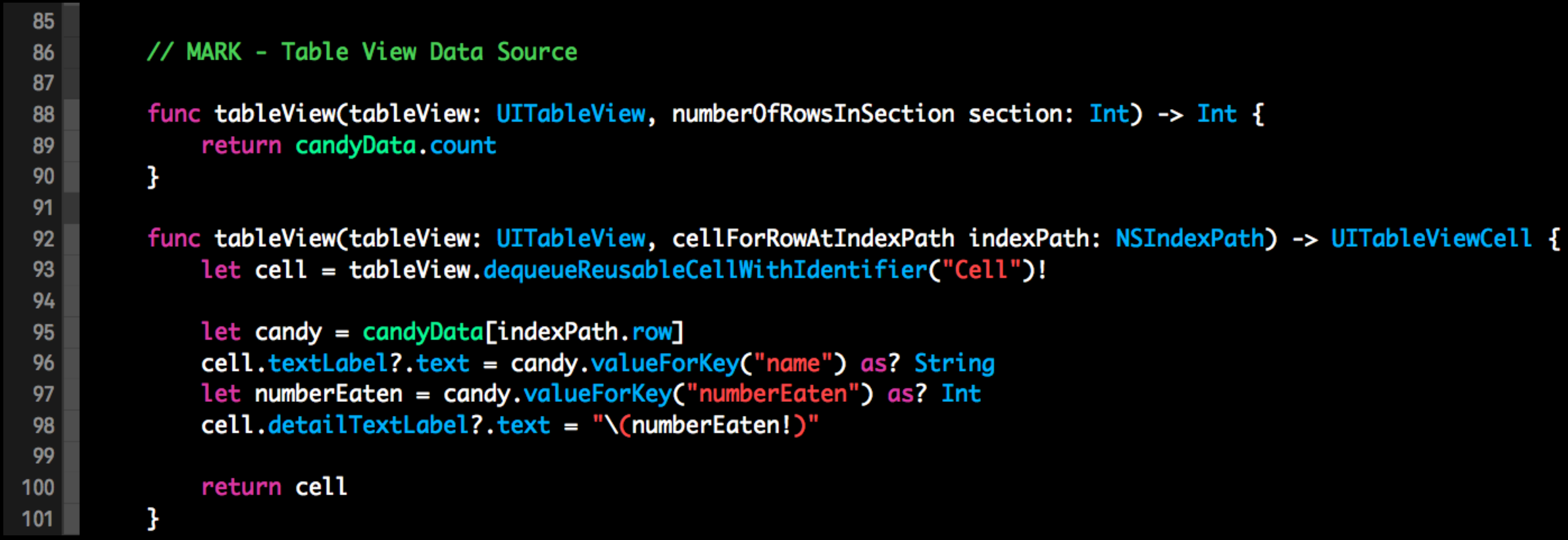
Now that we've defined our model, we can start work on putting it on-screen. Switch to your **`ViewController.swift`**. Below the **`tableView`**, add an array of **`NSManagedObjects`** called **`candyData`**. This will hold all the types of candy Core Data saves.

`NSManagedObject` is a class that Core Data uses to save things. Pretty much any time you see **`managed`**, it's something to do with Core Data.



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

At the top of your ViewController, make it into a table view data source, so the table view can ask us what to put for each row, and make it a table view delegate, so the table view can ask us what to do when the user taps a row. Don't worry about the error, we'll take care of that right now.

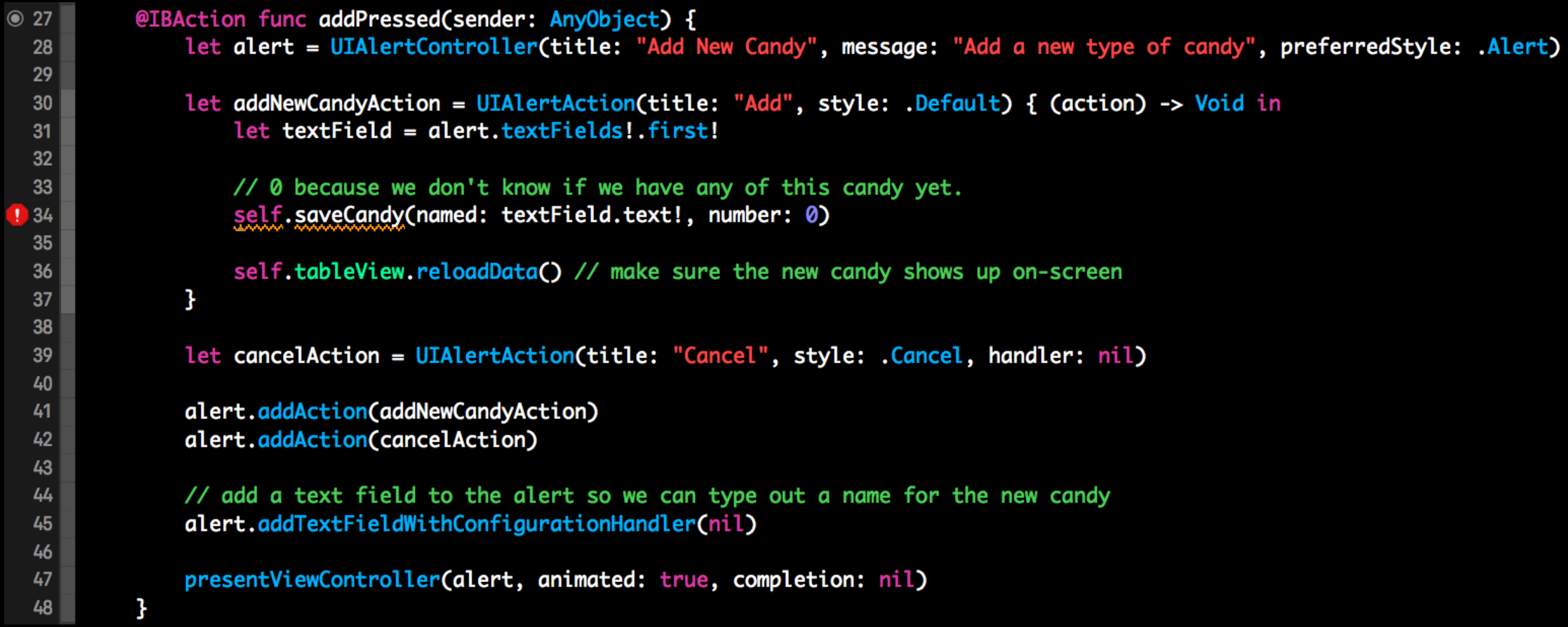


```
    // MARK - Table View Data Source

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

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("Cell")!

        let candy = candyData[indexPath.row]
        cell.textLabel?.text = candy.valueForKey("name") as? String
        let numberEaten = candy.valueForKey("numberEaten") as? Int
        cell.detailTextLabel?.text = "\(numberEaten!)"

        return cell
    }
```

Below the empty **`addPressed`** function we created earlier, add this code to give the table view data to display. Notice how we have to call **`valueForKey`** and cast to get the name of the candy from an **`NSManagedObject`**. This is because **`NSManagedObject`** has to work for everybody's Core Data apps, not just ours.

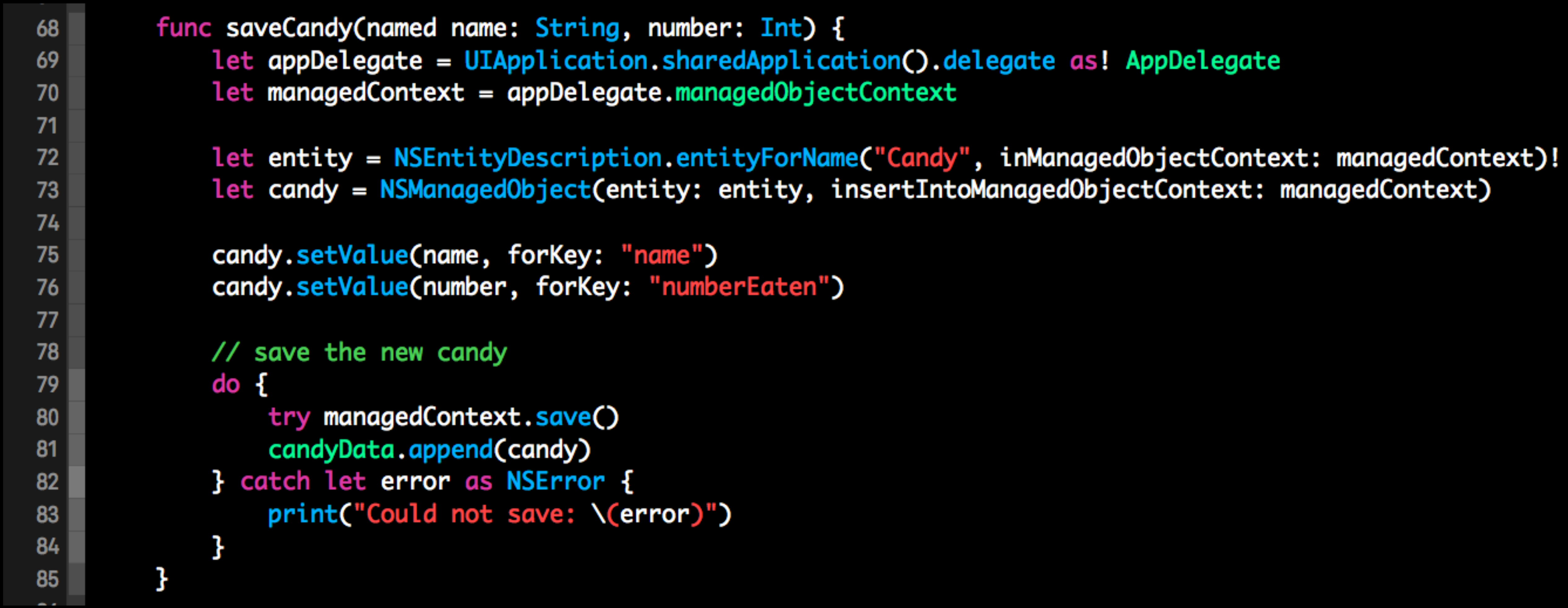
```
    // MARK - Table View Delegate

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        let candy = candyData[indexPath.row]
        // add one piece of candy
        let numberEaten = candy.valueForKey("numberEaten") as? Int
        candy.setValue(numberEaten! + 1, forKey: "numberEaten")
        // without this, the candy the user tapped would stay gray
        tableView.deselectRowAtIndexPath(indexPath, animated: true)
        // show the new number on-screen
        tableView.reloadData()
    }

}
```

Below that, add this function, which is called whenever the user taps on a row of the table view. We'll increment the number of pieces of candy the user tapped.



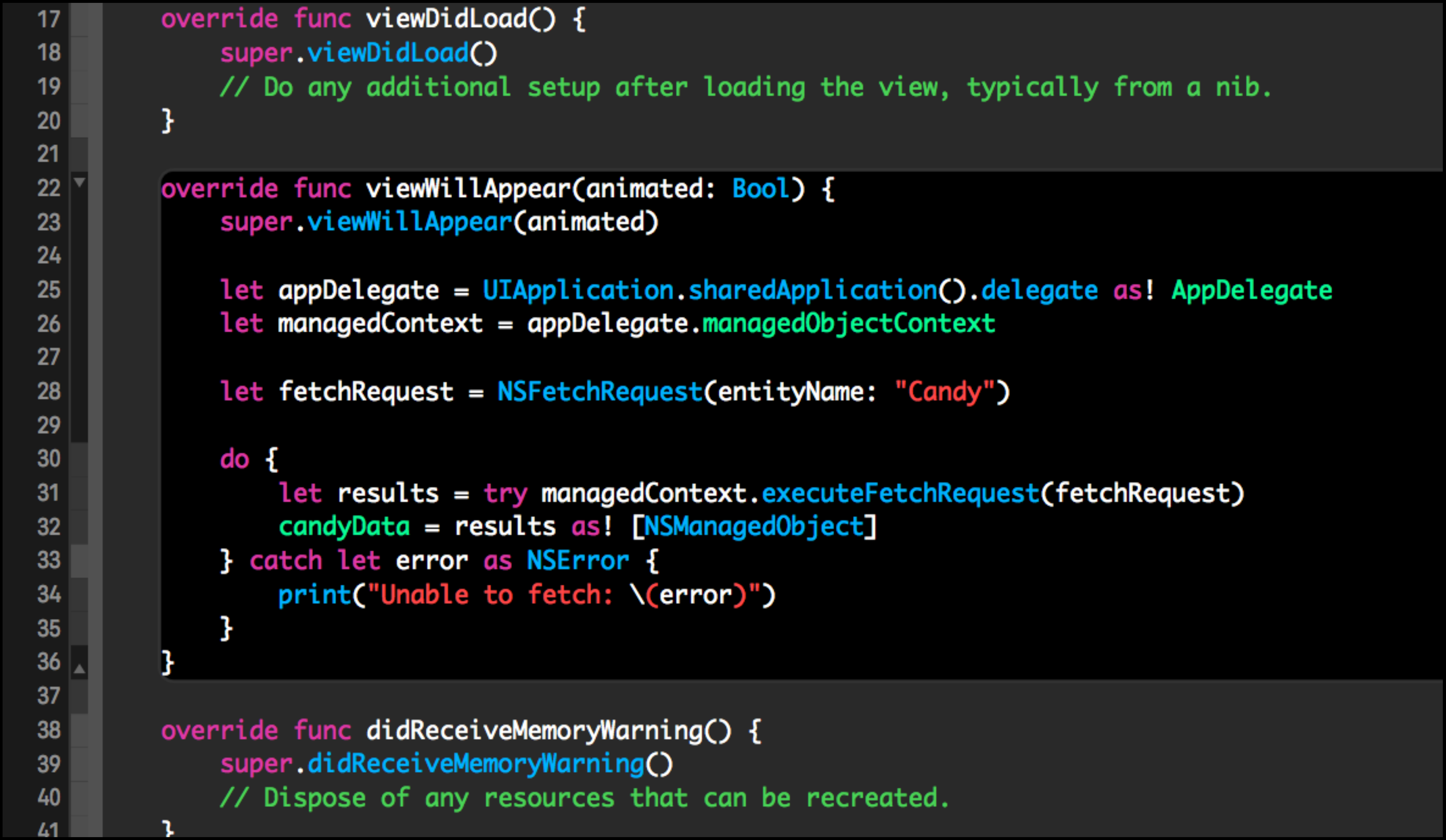
```
    @IBAction func addPressed(sender: AnyObject) {
        let alert = UIAlertController(title: "Add New Candy", message: "Add a new type of candy", preferredStyle: .Alert)

        let addNewCandyAction = UIAlertAction(title: "Add", style: .Default) { (action) -> Void in
            let textField = alert.textFields!.first!

            // 0 because we don't know if we have any of this candy yet.
            self.saveCandy(named: textField.text!, number: 0)

            self.tableView.reloadData() // make sure the new candy shows up on-screen
        }

        let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)

        alert.addAction(addNewCandyAction)
        alert.addAction(cancelAction)

        // add a text field to the alert so we can type out a name for the new candy
        alert.addTextFieldWithConfigurationHandler(nil)

        presentViewController(alert, animated: true, completion: nil)
    }
```

Let’s fill out the `addPressed` function we created earlier. Whenever the user taps the add button, we’ll pop up an alert for the user to type in the name of the new candy.



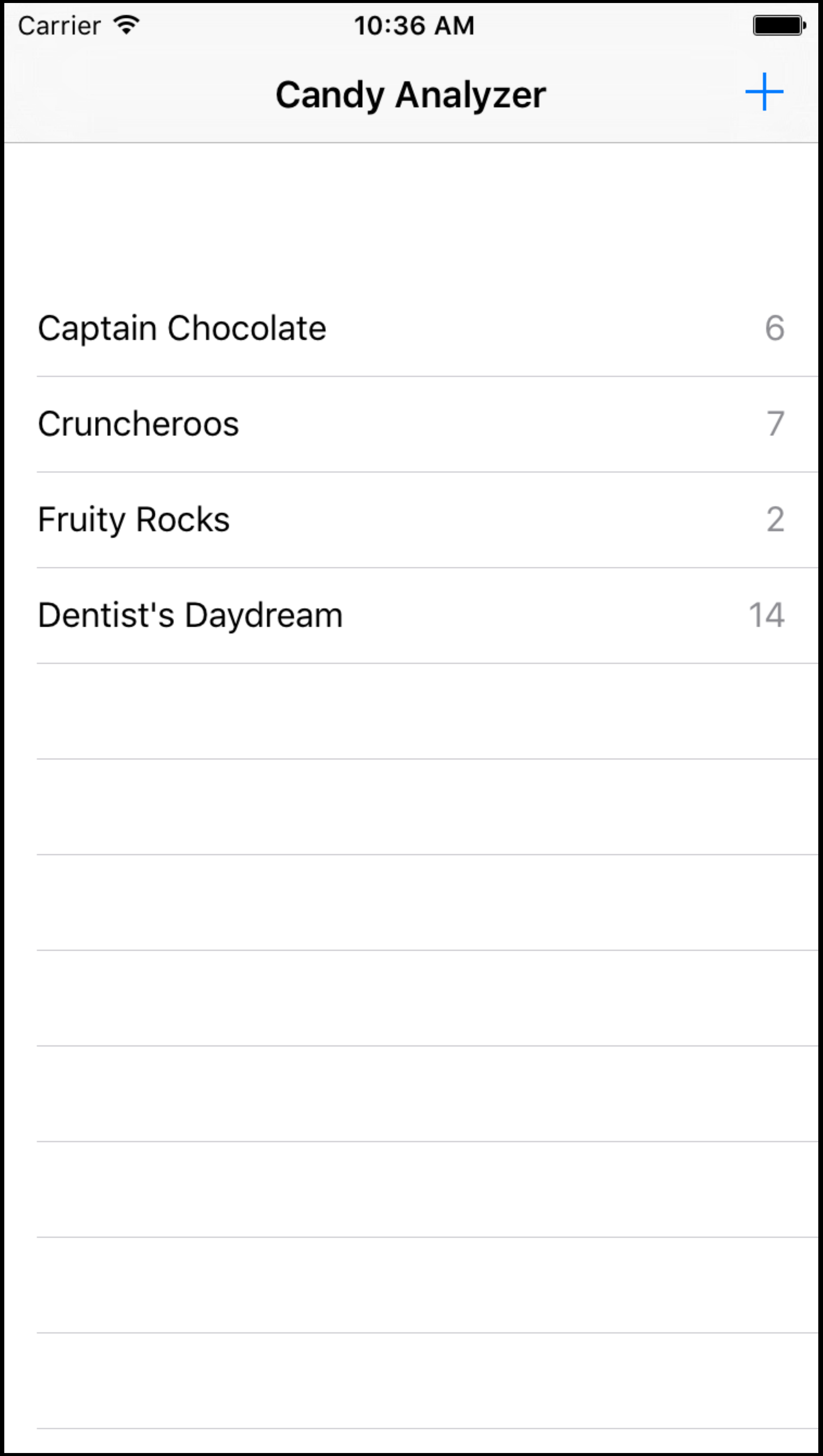
```swift
func saveCandy(named name: String, number: Int) {
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
    let managedContext = appDelegate.managedObjectContext

    let entity = NSEntityDescription.entityForName("Candy", inManagedObjectContext: managedContext)!
    let candy = NSManagedObject(entity: entity, insertIntoManagedObjectContext: managedContext)

    candy.setValue(name, forKey: "name")
    candy.setValue(number, forKey: "numberEaten")

    // save the new candy
    do {
        try managedContext.save()
        candyData.append(candy)
    } catch let error as NSError {
        print("Could not save: \(error)")
    }
}
```

Below `addPressed`, create a new function called `saveCandy`. This creates a new kind of candy and tells Core Data to start keeping track of it.
This is a bit more advanced, so if you don't really understand what it's doing here or how it works, that's fine. Just know that it works.



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

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)

        let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
        let managedContext = appDelegate.managedObjectContext

        let fetchRequest = NSFetchRequest(entityName: "Candy")

        do {
            let results = try managedContext.executeFetchRequest(fetchRequest)
            candyData = results as! [NSManagedObject]
        } catch let error as NSError {
            print("Unable to fetch: \(error)")
        }
    }

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

Almost done, but we still need to load up our saved data when we first run. Between `viewDidLoad` and `didReceiveMemoryWarning`, add this `viewWillAppear` function.


Carrier 10:36 AM

Candy Analyzer +

Captain Chocolate	6
Cruncheroos	7
Fruity Rocks	2
Dentist's Daydream	14

That's it! Give it a whirl. Tap the add button to create a new kind of candy, and tap the name of the candy to increment the number of pieces of that candy you have.

Troubleshooting tip: if it crashes as soon as you run it, try deleting the app from the phone and running it again. This deletes anything that Core Data may have saved.