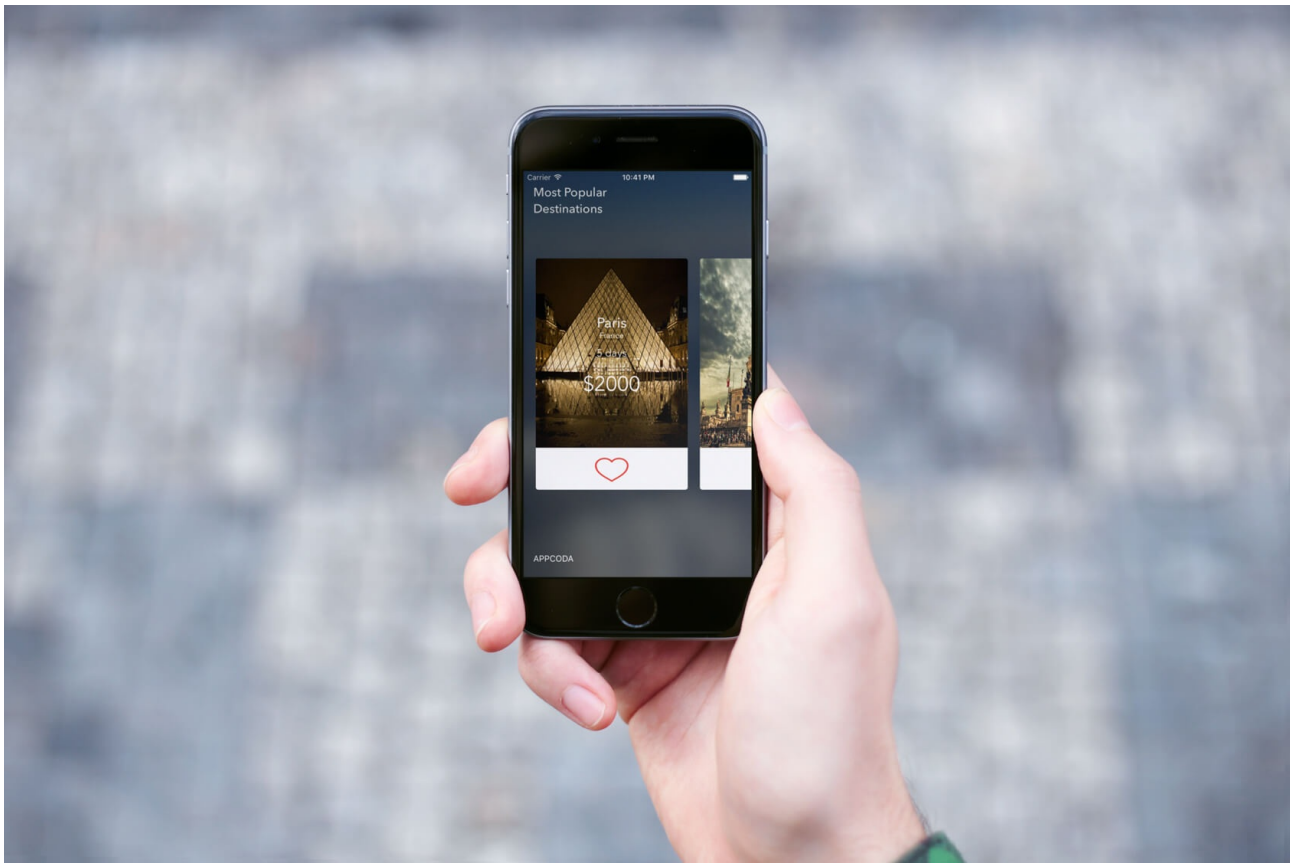
Carousel Like UI App

Carousel is a popular way to showcase a variety of featured content. Not only can you find carousel design in mobile apps, but it has also been applied to web applications for many years. A carousel arranges a set of items horizontally, where each item usually includes a thumbnail.

Users can scroll through the list of items by flicking left or right.

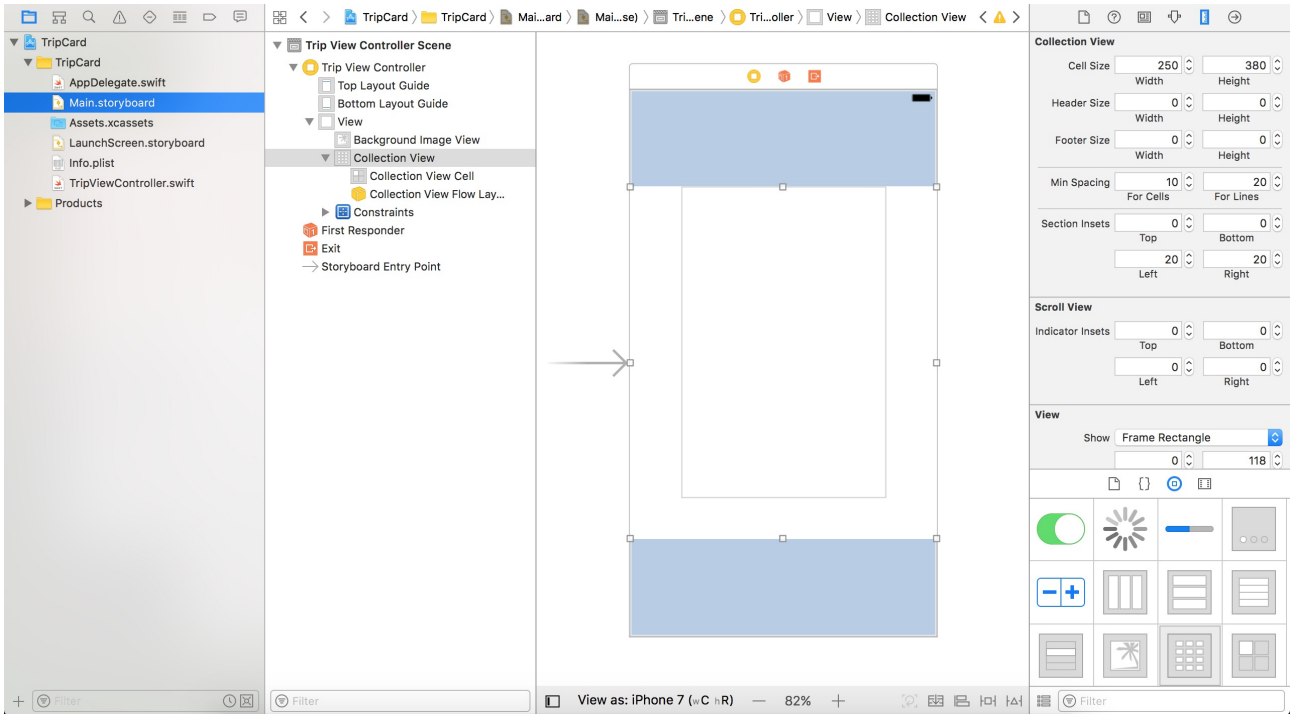


In this practice, we will build a carousel in iOS apps. All you need to do is to implement a UICollectionView . We walk through the building a demo app with a simple carousel that displays a list of trips.

Designing the Storyboard

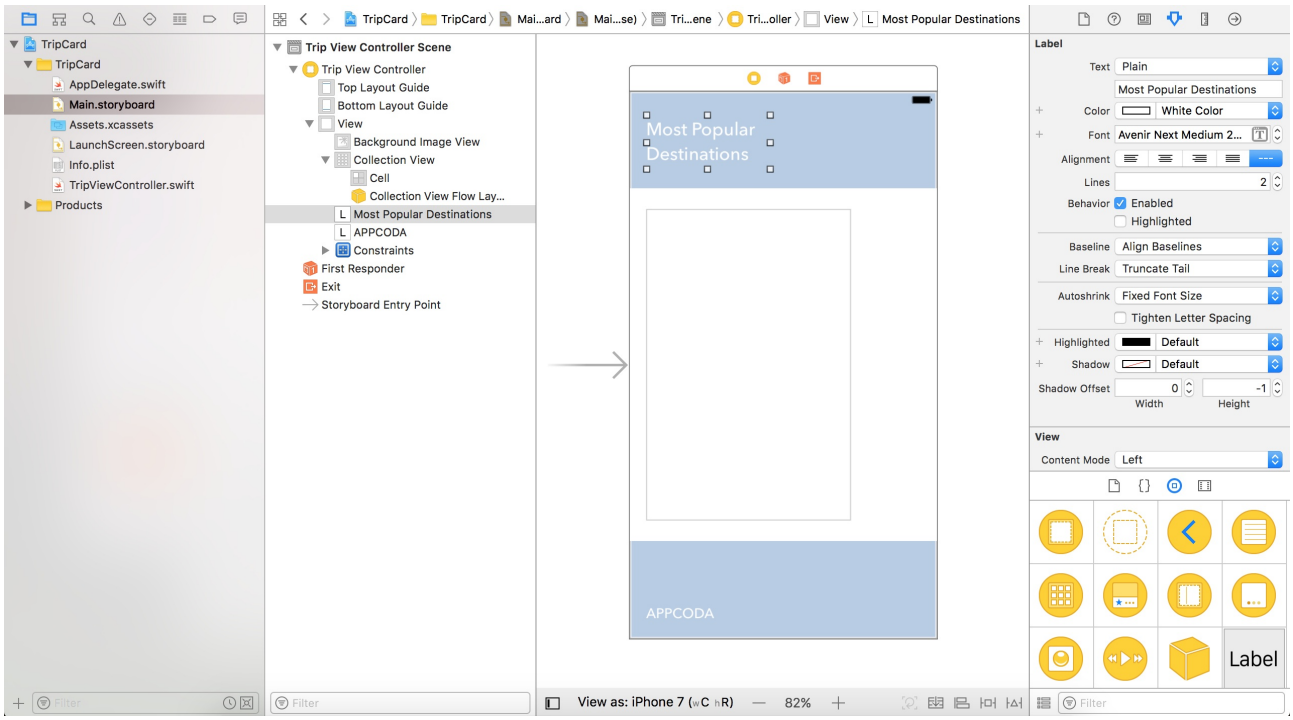
To begin with, you can create a new project named TripCard.

Okay, go to Main.storyboard . Drag a collection view from the Object library to the view controller. Resize its width to 375 points and height to 430 points. Place it at the center of the view controller. Next, go to the Size inspector. In the cell size option, set the width to 250 points and height to 380 points. Also change minimum spacing for lines to 20 points to add some spacing between cell items. Lastly, set the left and right values of section insets to 20 points.

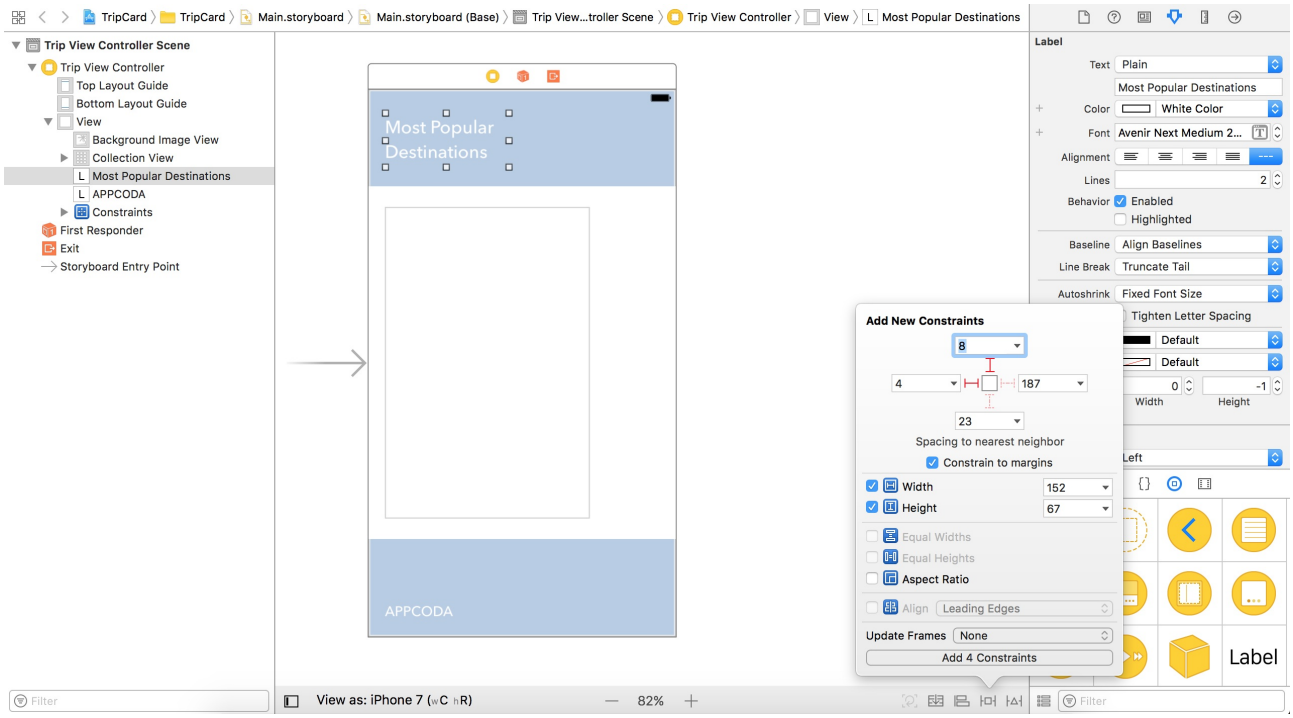


Your storyboard should now look similar to the screenshot above. Now select the collection view and go to the Attributes inspector. Change the scroll direction from vertical to horizontal . Once you have made this change, users will be able to scroll through the collection view horizontally instead of vertically. This is the real trick to building a carousel. Don't forget to set the identifier of the collection view cell to Cell .

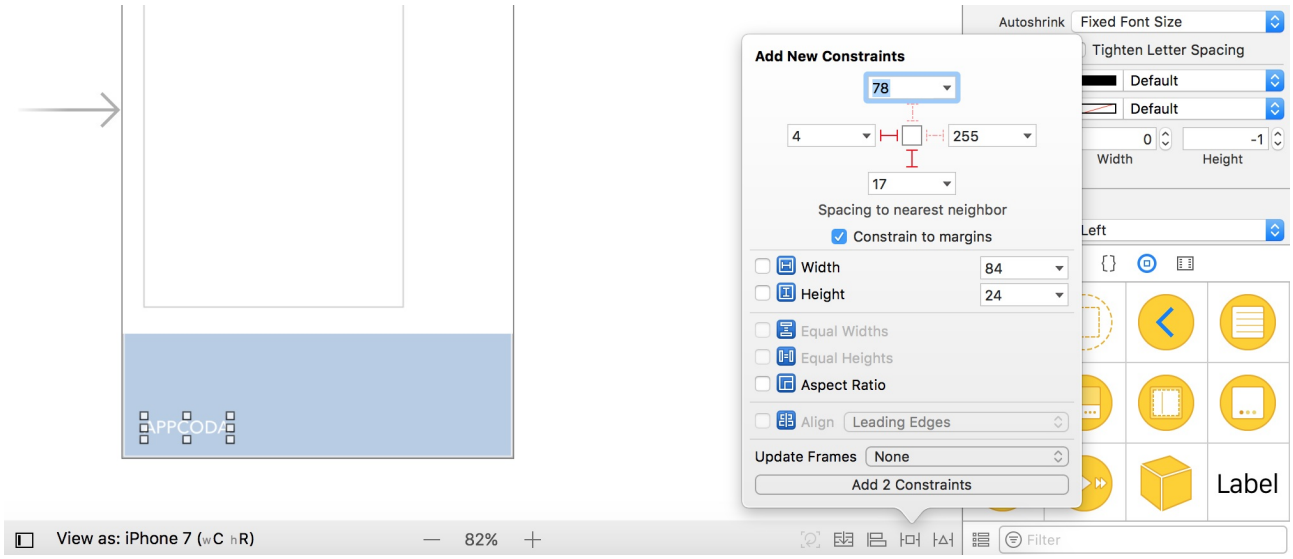
Next, drag a label to the view controller and place it at the top-left corner of the view. Set the text to Most Popular Destinations and the color to white . Change to your preferred font and size. Then, add another label to the view controller but put it below the view controller. Change its text to TRIPCARD or whatever you prefer. Your view controller will look similar to this:



So far we haven't configured any auto layout constraint. First, select the Most Popular Destinations label. Click the Add New Constraint (or Pin) button to add a couple of spacing and size constraints. Select the left and top bar, and check both width and height checkboxes. Click Add 4 Constraints to add the constraints.

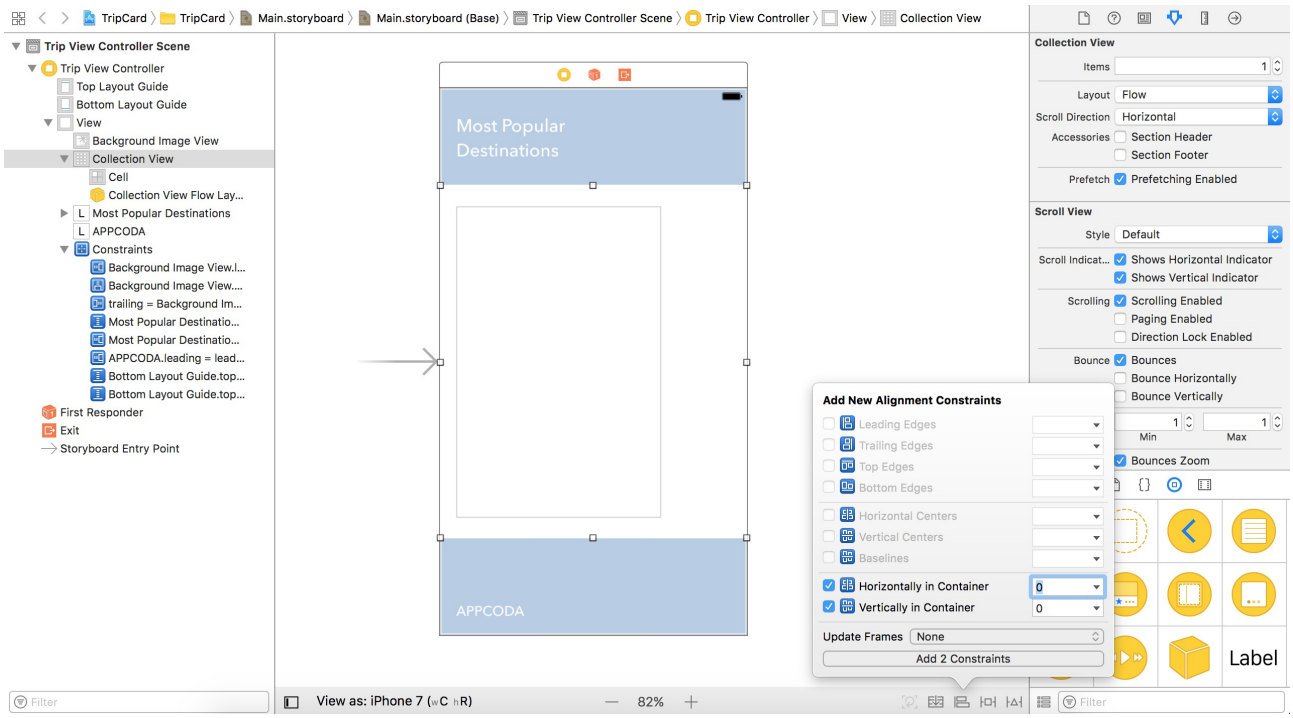


For the bottom label, click Add New Constraints button to add two spacing constraints. Click the bar of both left and bottom sides, and then click Add 2 Constraints .

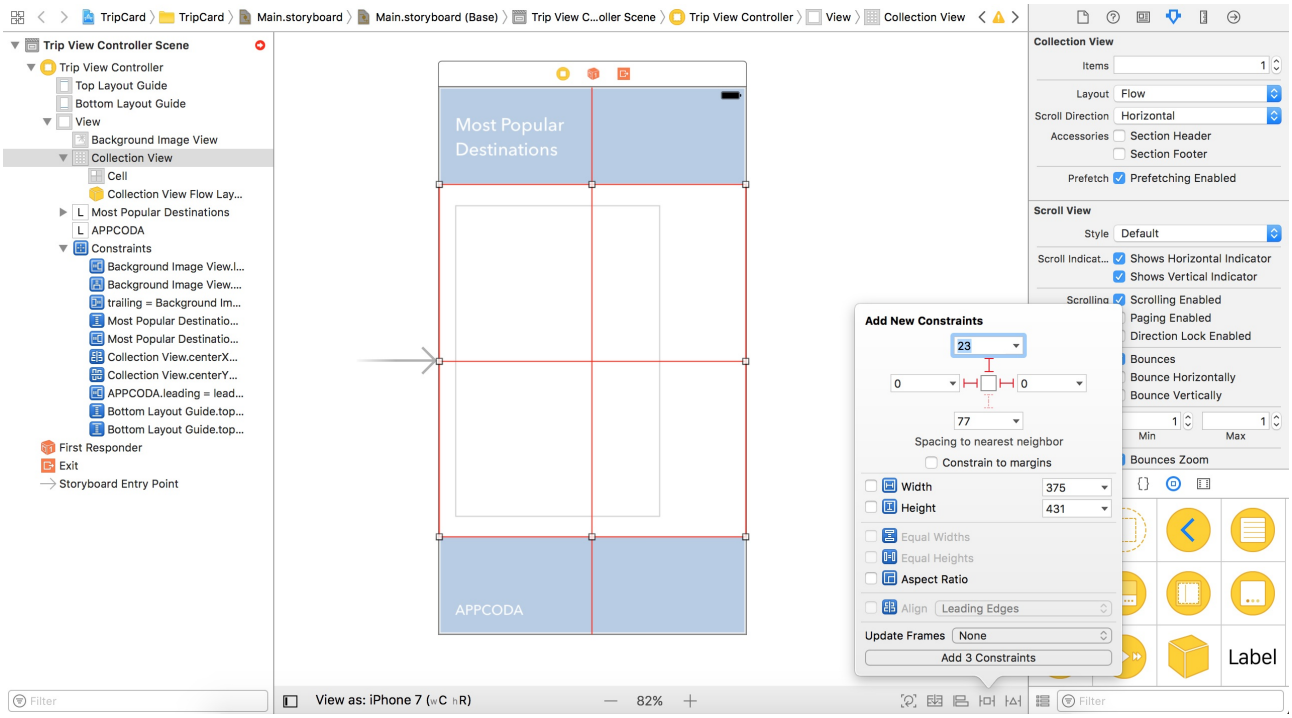


Now let's add a few layout constraints for the collection view. Select the collection view and click the Align button of the auto layout bar. Check both the Horizontal Center in Container and Vertical Center in

Container options, and click Add 2 Constraints. This will align the collection view to the center of the view.



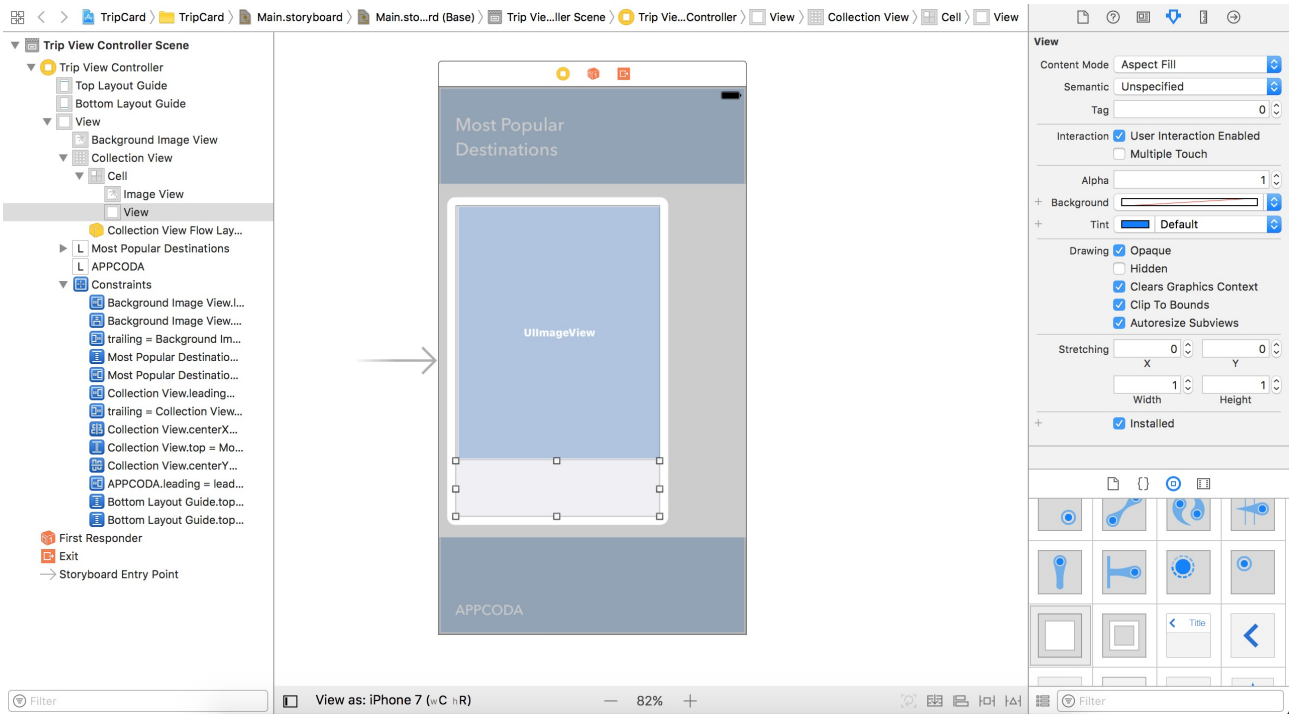
Xcode should indicate some missing constraints. Click the Add New Constraints button and select the dashed red line corresponding to the top, left and right sides. Uncheck the Constrain to margins option and click Add 3 Constraints. This ensures that the left and right sides of the collection view align perfectly with the background image view. Also, the collection view is several points away from the title label.



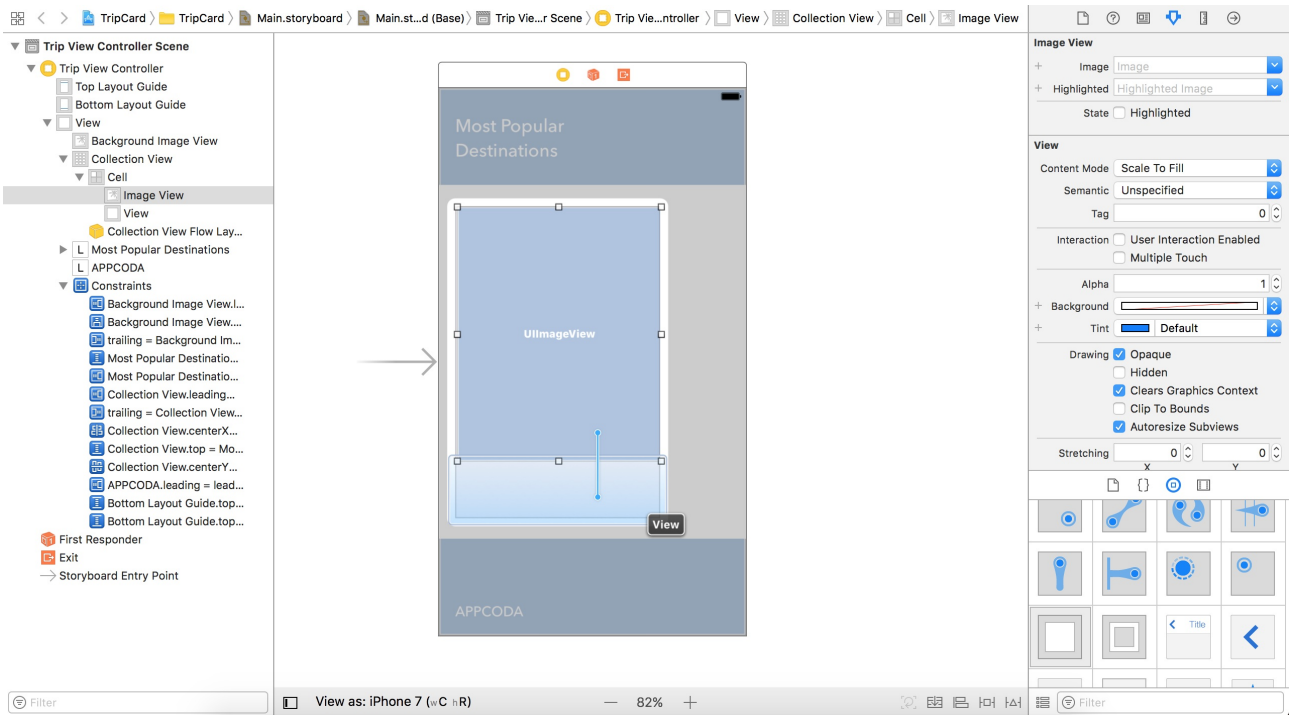
Now that you have created the skeleton of the collection view, let's configure the cell content, which will be used to display trip information. First, select the cell and change its background to light gray . Then drag an image view to the cell and change its size to 250x311 points.

Next, drag a view from the Object Library and place it right below the image view. In the Attributes inspector, change its background color to Default, set the mode to Aspect Fill and enable the Clip to Bounds option. This view serves as a container to hold other UI elements. Sometimes it is good to use a view to group multiple UI elements together so that it is easier for you to define the layout constraints later.

If you follow the procedures correctly, your storyboard should look similar to this:

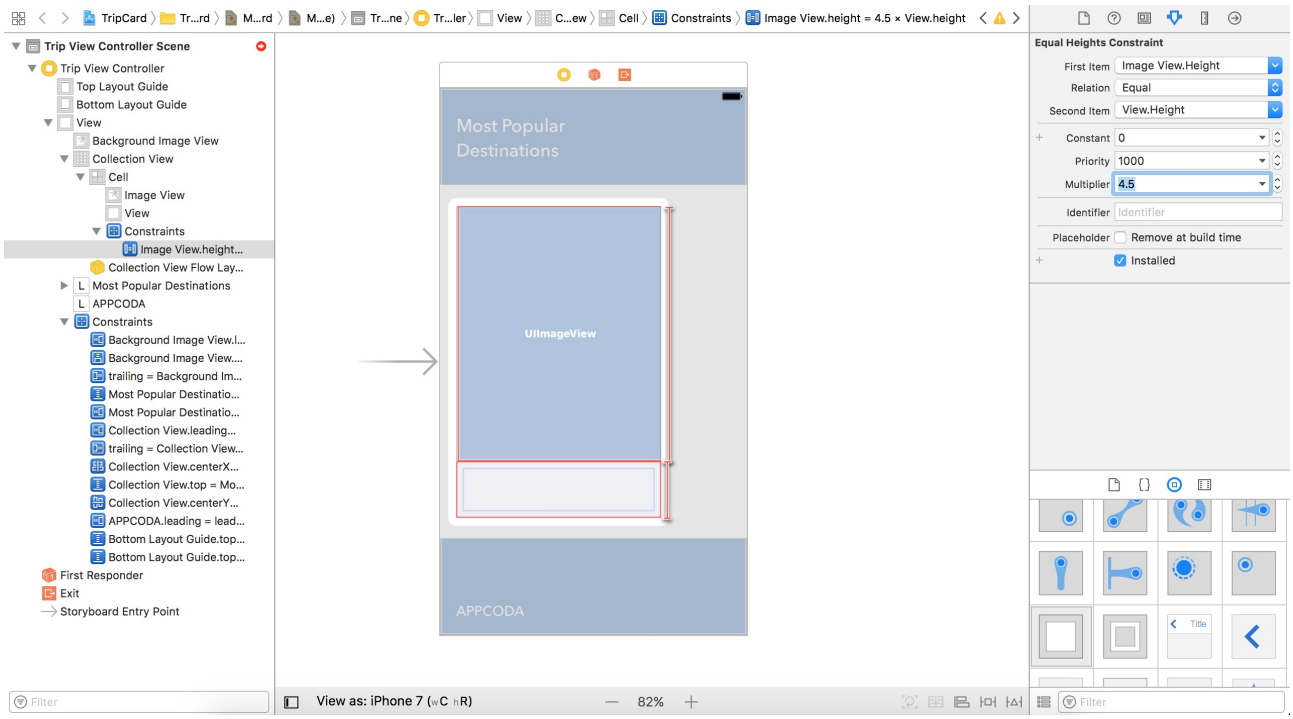


Later we will change the size of the collection view with reference to the screen height. But we still want to keep the height of the image view and the view inside the cell proportional. To do that, control-drag from the image view to the view and select Equal Heights.



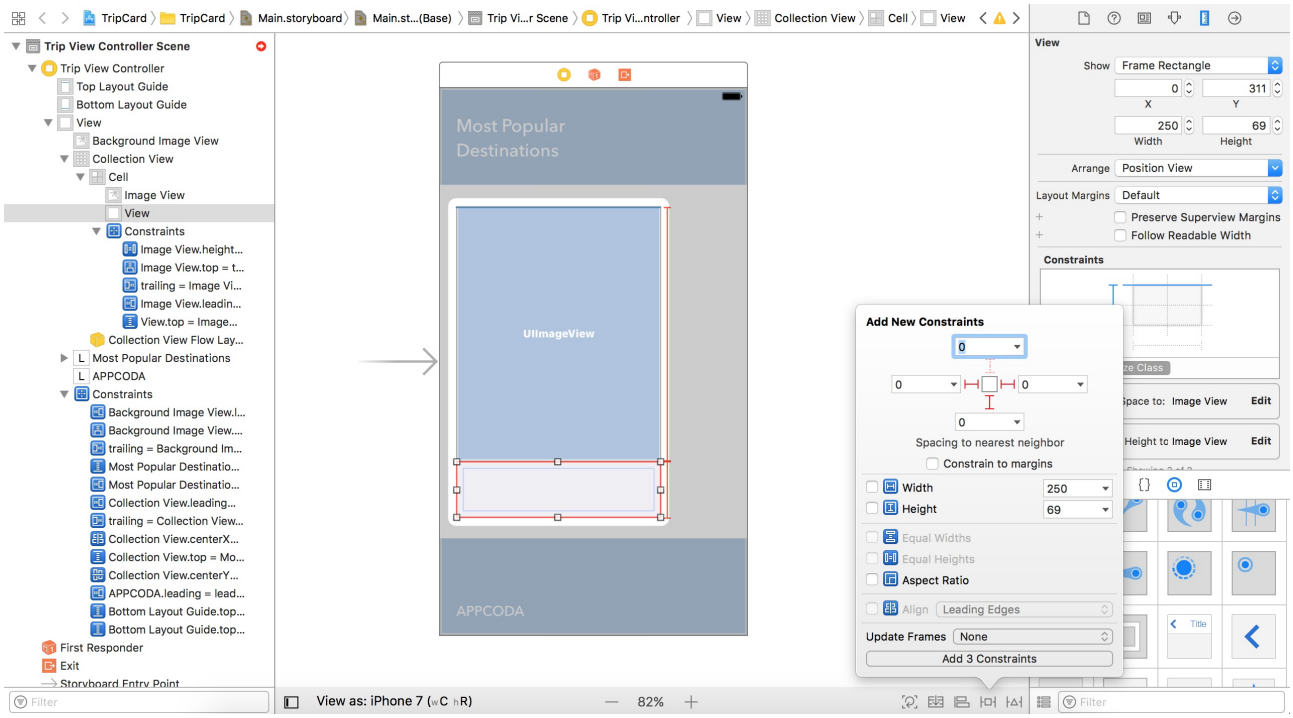
Next, select the constraint just created and go to the Size inspector. Change the multiplier from 1 to 4.5 . Make sure the first and second items are set to Image View.height and View.height respectively. This defines a constraint so that the height of the image view is

always 4.5 times taller than the view.



Now select the image view and define the spacing constraints. Click the Add New Constraints button and select the dashed red lines of all sides. Click the Add 4 Constraints button to define the layout constraints.

Select the view inside the collection view cell and click the Add New Constraints button. Click the dashed red lines that correspond to the left, right and bottom sides.

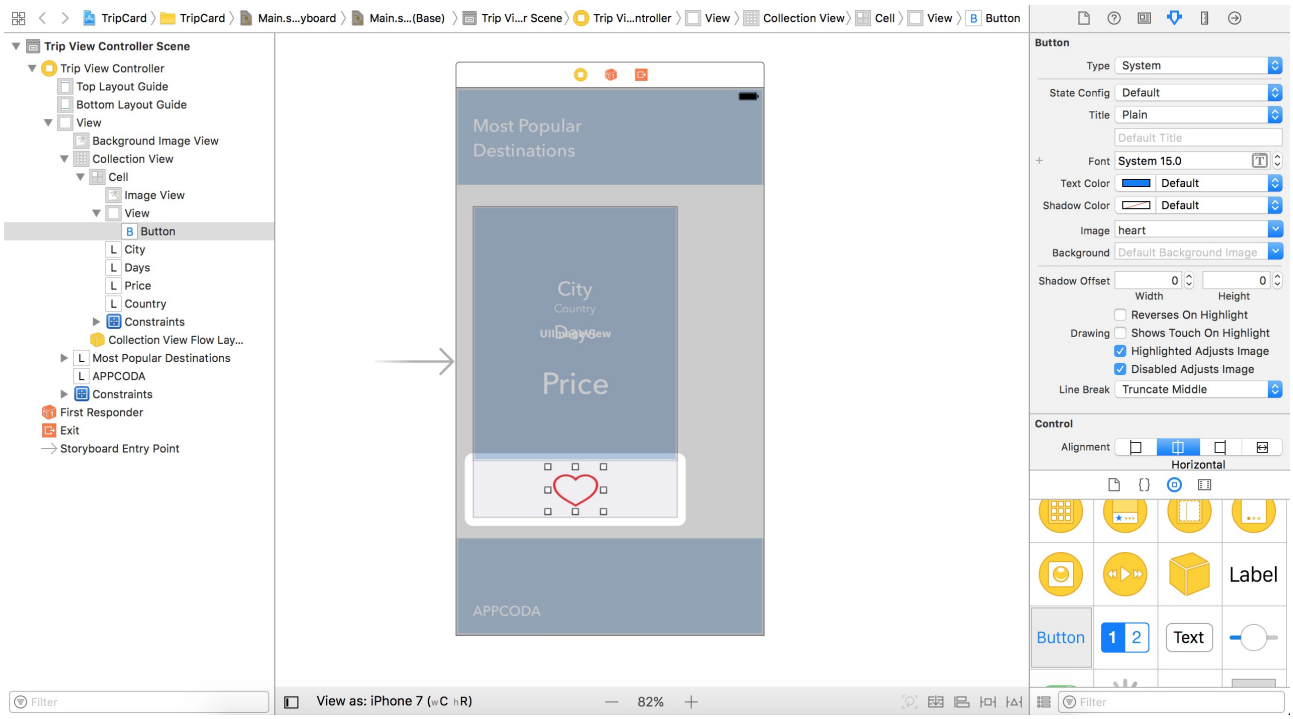


If you follow every step correctly, you've defined all the required constraints for the image view and the internal view. It's now time to add some UI elements to the image view for displaying the trip information.

- First, add a label to the image view of the cell. Name it City and change its color to white . You may change its font and size.
- Second, drag another label to the image view. Name it Country and set the color to white . Again, change its font to whatever you like
- Next, add another label to the image view. Name it Days and set the color to white. Change the font to whatever you like (e.g. Avenir Next), but make it larger than the other two labels.
- Drag another label to the image view. Name it Price and set the color to white . Change its size such that it is larger than the rest of the labels.
- Finally, add a button object to the view (below the image view) and place it at the center of the view. In the Attributes inspector,

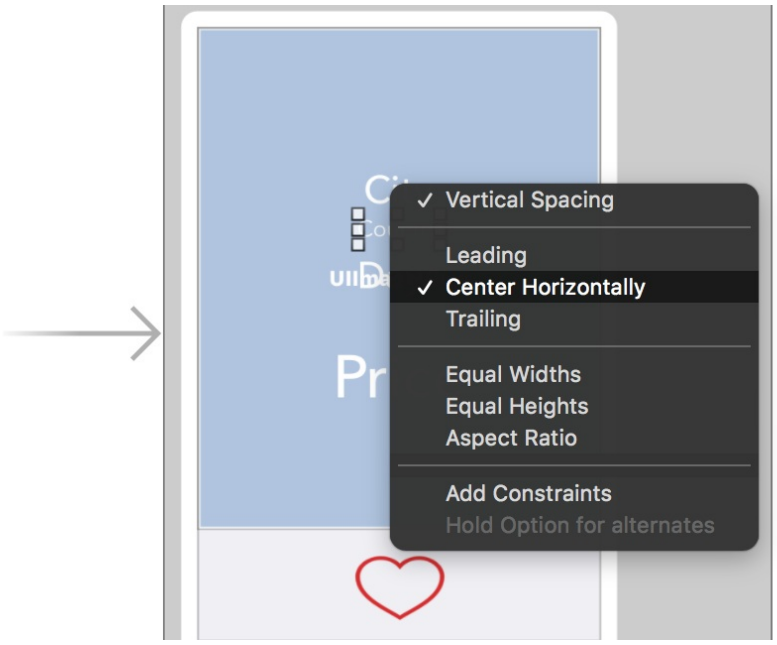
change its title to blank and set the image to heart. Also change its type to System and tint color to red . In the Size inspector, set its width to

- 69 points and height to 56 points.



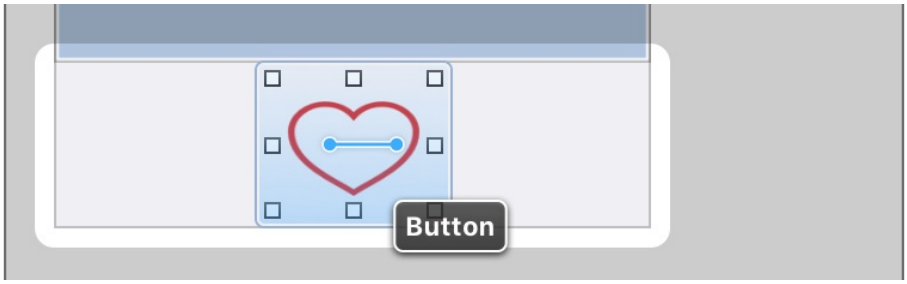
The UI design is almost complete. We simply need to add a few layout constraints for the elements we just added. First, control-drag from the City label to the image view of the cell.

In the popover menu, select both Vertical Spacing and Center Horizontally (hold the shift key to select multiple options). Next, control-drag from the Country label to the City label. Release the buttons and select both the Vertical Spacing and Center Horizontally options.

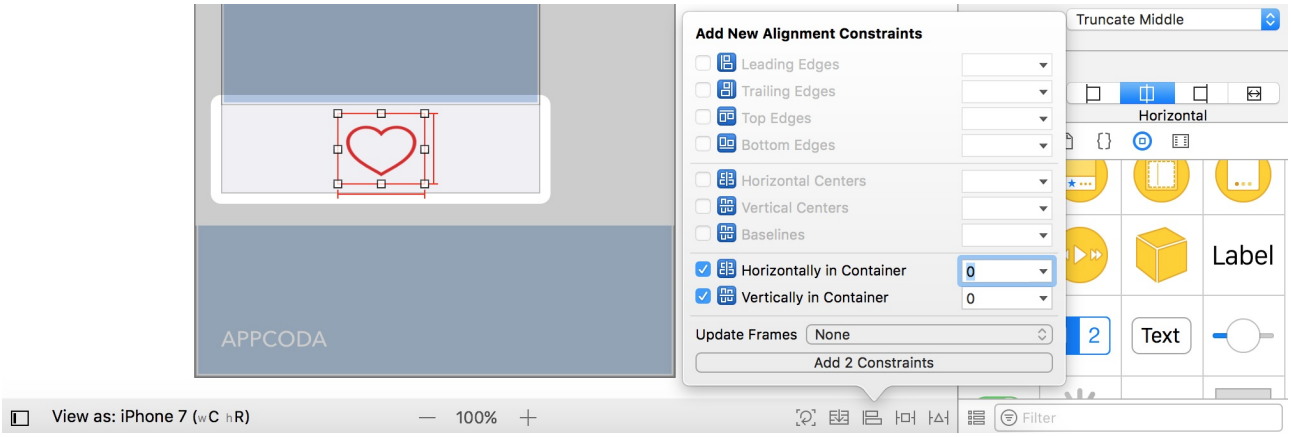


Then, control-drag from the Days label to the Country label. Repeat the procedure and set the same set of constraints. Lastly, control-drag from the Price label to the Days label and define the same layout constraints.

For the heart button, we want it to be a fixed size. Control-drag to the right (see below) and set the Width constraint. Next, control-drag vertically to set the Height constraint for the button.



To ensure the heart button is always displayed at the center of the view, click the Align button and select Horizontal Center in Container and Vertical Center in Container.



We have completed the UI design. Now we will move onto the coding part.

Creating a Custom Class for the Collection View Cell

As the collection view cell is customized, we will first create a custom class for it. In the Project Navigator, right-click the TripCard

folder and select New File... . Choose the Cocoa Touch Class template and proceed.

Name the class TripCollectionViewCell and set it as a subclass of UICollectionViewCell . Once the class is created, open up TripCollectionViewCell.swift and update the code to the following:

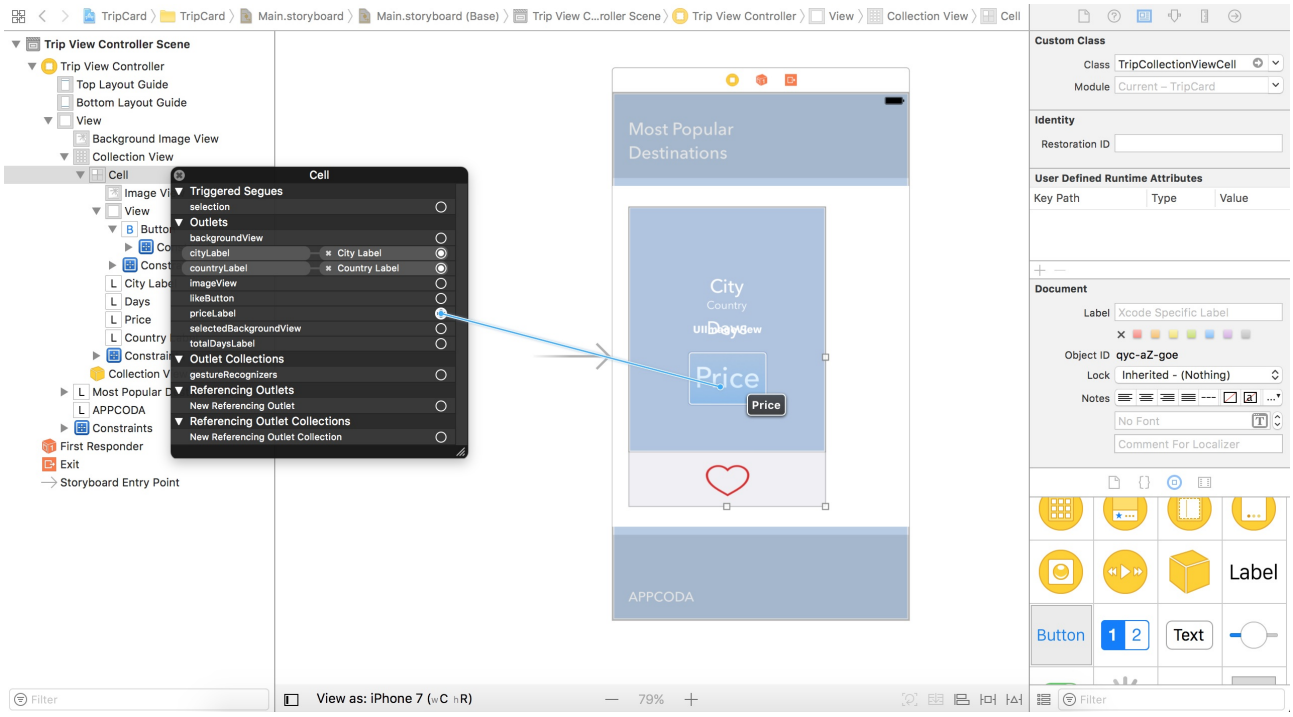
```
class TripCollectionViewCell: UICollectionViewCell {
    @IBOutlet var imageView: UIImageView!
    @IBOutlet var cityLabel: UILabel!
    @IBOutlet var countryLabel: UILabel!
    @IBOutlet var totalDaysLabel: UILabel!
    @IBOutlet var priceLabel: UILabel!
    @IBOutlet var likeButton: UIButton!
    var isLiked:Bool = false  {
        didSet {
            if isLiked {
                likeButton.setImage(UIImage(named: "heartfull"), for: .normal)
            } else {
                likeButton.setImage(UIImage(named: "heart"), for: .normal)
            }

        }

    }

}
```

The above lines of code should be very familiar to you. We simply define the outlet variables to associate with the labels, image view and button of the collection view cell in storyboard. The

isLiked variable is a boolean to indicate whether a user favors a trip or not. In the above code, we declare a didSet observer for the isLiked property. If this is the first time you have heard of property observer, it is a great feature of Swift. When the isLiked property is stored, the

didSet observer will be called immediately. Here we simply set the image of the like button according to the value of isLiked .

Now go back to the storyboard and select the collection view cell. In the Identity inspector, set the custom class to TripCollectionViewCell . Right click the Cell in Document Outline. Connect each of the outlet variables to the corresponding visual element.



Creating the Model Class

Before we implement the TripViewController class to populate the data, we will create a model class named Trip to represent a trip. Create a new file using the Swift File template and name the class Trip . Proceed to create and save the Trip.swift file. Open Trip.swift and update the code to the following:

```
import UIKit

struct Trip {
    var tripId = ""
    var city = ""
    var country = ""
```

```
    var featuredImage: UIImage?
    var price:Int = 0
    var totalDays:Int = 0
    var isLiked = false
    init(tripId: String, city: String, country: String, featuredImage:
UIImage!, price: Int, totalDays: Int, isLiked: Bool) {
        self.tripId = tripId
        self.city = city
        self.country = country
        self.featuredImage = featuredImage
        self.price = price
        self.totalDays = totalDays
        self.isLiked = isLiked
    }
}
```

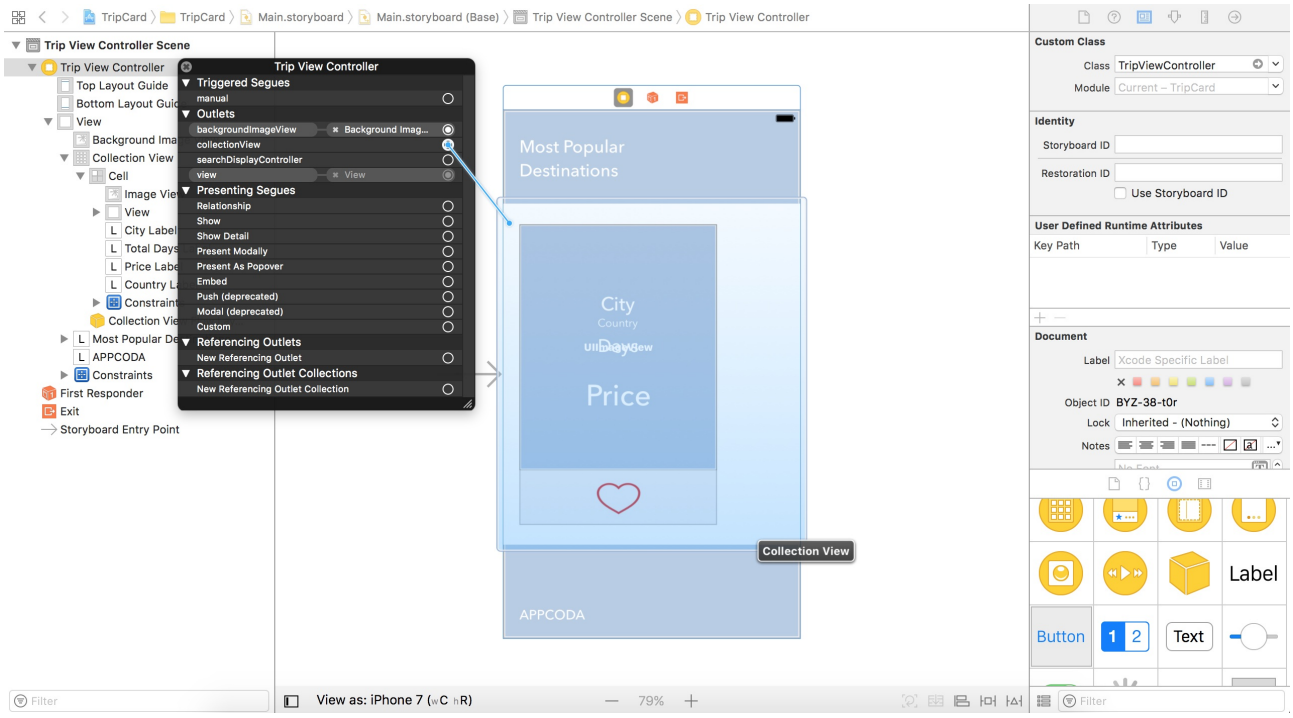
The Trip structure contains a few properties for holding the trip data including ID, city, country, featured image, price, total number of days and isLiked. Other than the ID and isLiked properties, the rest of the properties are self-explanatory. Regarding the trip ID property, it is used for holding a unique ID of a trip. isLiked is a boolean variable that indicates whether a user favors the trip.

Populating the Collection View

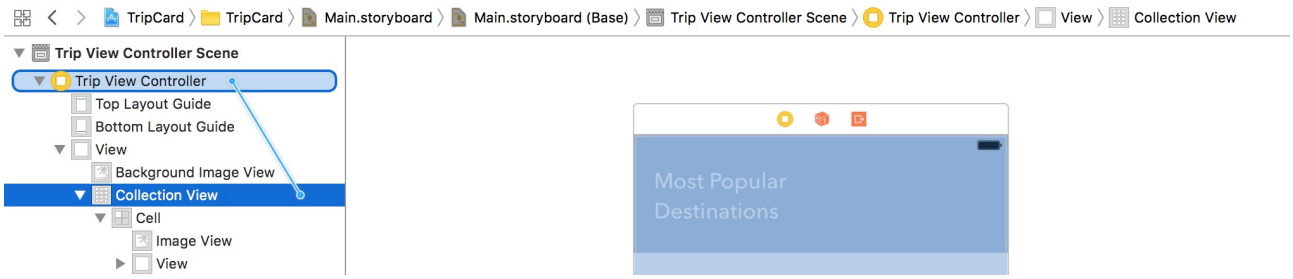
Now we are ready to populate the collection view with some trip data. First, declare an outlet

variable for the collection view in TripViewController.swift : `@IBOutlet var collectionView: UICollectionView!`

Go to the storyboard. In the Document Outline, right click Trip View Controller. Connect collectionView outlet variable with the collection view.



Furthermore, control-drag from collection view to trip view controller to connect the data source and delegate.



To keep things simple, we will just put the trip data into an array. Declare the following variable in TripViewController.swift :

```
private var trips = [Trip(tripId: "Paris001", city: "Paris", country: "France",
featuredImage: UIImage(named: "paris"), price: 2000, totalDays: 5, isLiked:
false),
    Trip(tripId: "Rome001", city: "Rome", country: "Italy", featuredImage:
UIImage(named: "rome"), price: 800, totalDays: 3, isLiked: false),
    Trip(tripId: "Istanbul001", city: "Istanbul", country: "Turkey",
featuredImage: UIImage(named: "istanbul"), price: 2200, totalDays: 10, isLiked:
```

```
false),
]
```

To manage the data in the collection view, the TripViewController class has to adopt the UICollectionViewDelegate and UICollectionViewDataSource protocols. So change the class

declaration like this:

```
class TripViewController: UIViewController, UICollectionViewDelegate,
UICollectionViewDataSource
```

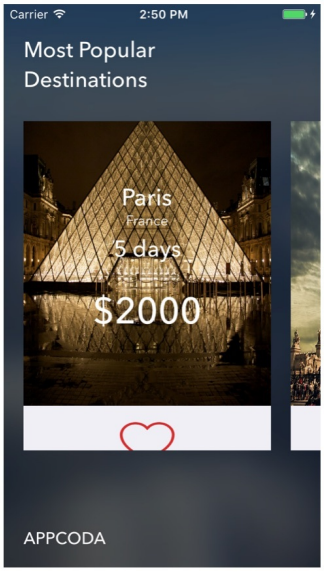
Next, implement the required methods of the protocols:

```
func numberOfSections(in collectionView: UICollectionView) -> Int {
    return 1
}

func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int {
    return trips.count
}
func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "Cell",
for: indexPath) as! TripCollectionViewCell
    // Configure the cell
    cell.cityLabel.text = trips[indexPath.row].city
    cell.countryLabel.text = trips[indexPath.row].country
    cell.imageView.image = trips[indexPath.row].featuredImage
    cell.priceLabel.text = "$\(String(trips[indexPath.row].price))"
    cell.totalDaysLabel.text = "\(trips[indexPath.row].totalDays) days"
    cell.isLiked = trips[indexPath.row].isLiked
    // Apply round corner
    cell.layer.cornerRadius = 4.0
    return cell
}
```

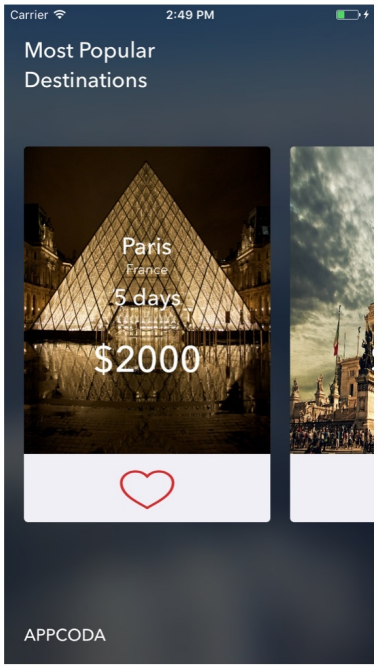
We will not go into the details of the implementation as you should be very familiar with the methods. Finally, insert this line of code in the viewDidLoad method to make the collection view transparent:

```
collectionView.backgroundColor = UIColor.clear
```

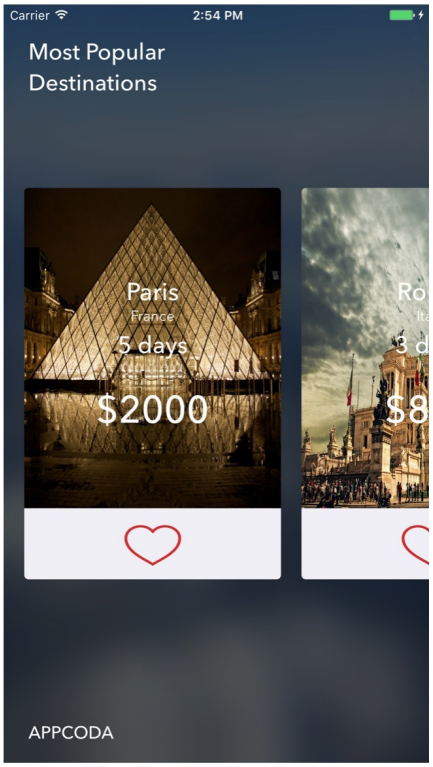
Now it's time to test the app. Hit the Run button, and you should have a carousel showing a list of trips. The app works properly on devices with at least 4.7-inch display. If you run the app on iPhone 5s, however, parts of the collection view are blocked.



4-inch



4.7-inch



5.5-inch

We have to reduce the height of the collection view for 4-inch devices. Insert the following block of code in the viewDidLoad method of TripViewController.swift :

```
if UIScreen.main.bounds.size.height == 568.0 {
    let flowLayout = self.collectionView.collectionViewLayout as!
UICollectionViewFlowLayout
    flowLayout.itemSize = CGSize(width: 250.0, height: 330.0)
```

```
}
```

Base on the screen height, we can deduce if the device has a 4-inch screen. If it meets the criteria, we adjust the height of the collection view from 380 points to 330 points. Once you have made the change, try to test the app on iPhone 5s again. This should work now.

Handling the Like Button

We want to toggle the heart button when a user taps on it. We don't want to toggle it when a user taps on the featured image or the price label.

To fit the requirement, we are going to define a new protocol named TripCollectionCellDelegate in the TripCollectionViewCell class:

```
protocol TripCollectionCellDelegate {
func didLikeButtonPressed(cell: TripCollectionViewCell)
}
```

And declare a variable in the class to hold the delegate object:

```
var delegate:TripCollectionCellDelegate?
```

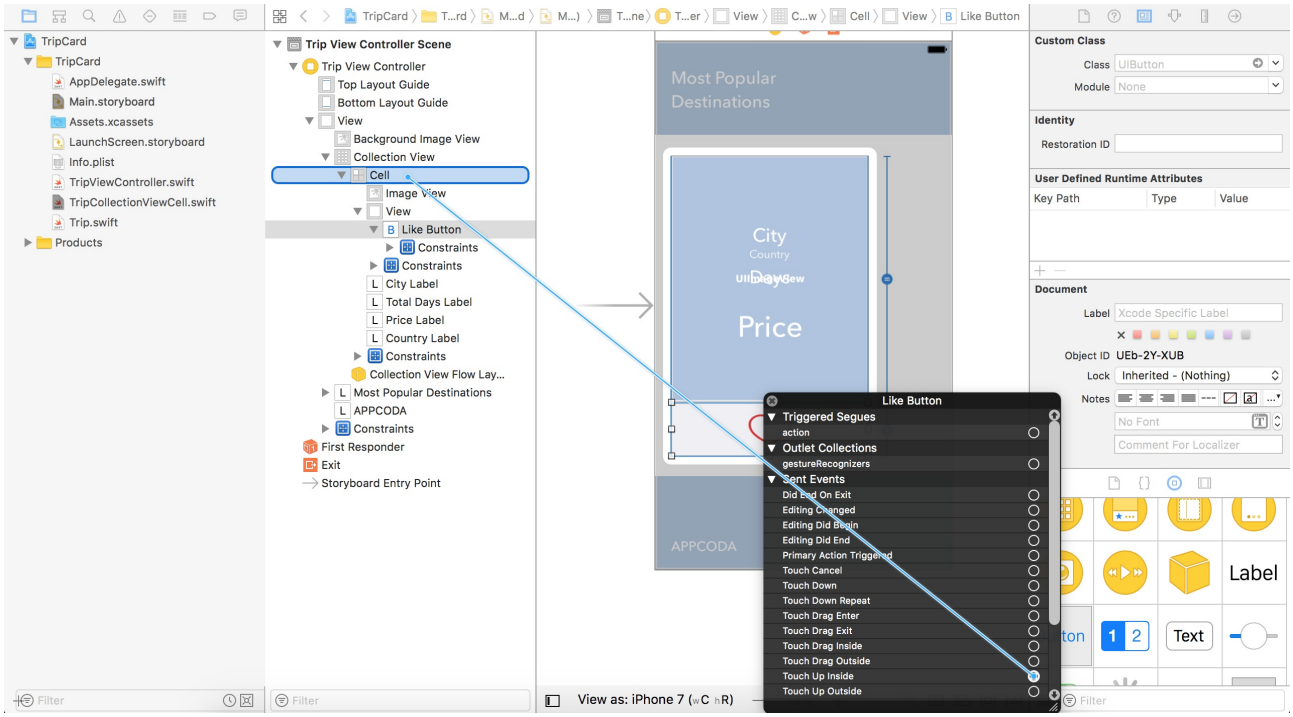
In the protocol, we define a method called didLikeButtonPressed , which will be invoked when the heart button is tapped. The object that implements the delegate protocol is responsible for handling the button press.

Add the following action method, which is triggered when a user taps the heart button:

```
@IBAction func likeButtonTapped(sender: AnyObject) {
    delegate?.didLikeButtonPressed(cell: self)
```

}

Now go back to the storyboard to associate the heart button with this method. Right-click the heart button and drag from Touch Up Inside to the Cell in the Document Outline. Select likeButtonTappedWithSender: when the popover appears.



Now open TripViewController.swift . It is the object that adopts the TripCollectionCellDelegate protocol:

class TripViewController: UIViewController, UICollectionViewDelegate, UICollectionViewDataSource, TripCollectionCellDelegate

In the class, implement the required method of the protocol like this:

```
func didLikeButtonPressed(cell: TripCollectionViewCell) {
    if let indexPath = collectionView.indexPath(for: cell) {
      trips[indexPath.row].isLiked = trips[indexPath.row].isLiked ? false :
        cell.isLiked = trips[indexPath.row].isLiked
    }

}
```

When the heart button is tapped, the didLikeButtonPressed method is called, along with the selected cell. Based on selected cell, we can determine the index path using the

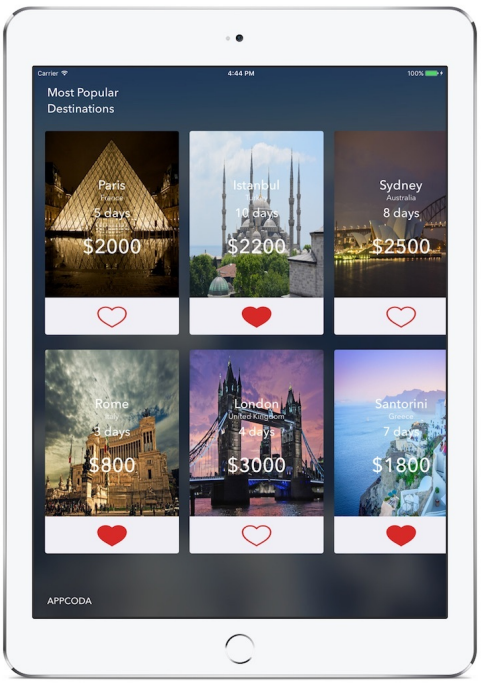
indexPath(for:) method and toggle the status of isLiked accordingly.

Recall that we have defined a didSet observer for the isLiked property of TripCollectionViewCell . The heart button will change its images according to the value of isLiked . For instance, the app displays an empty heart if isLiked is set to false .

Lastly, insert a line of code in the collectionView(_:cellForItemAt:) method to set the cell's delegate:

```
cell.delegate = self
```

Okay, let's test the app again. When it launches, tapping the heart button of a trip can now favor the trip.



***