App with Parse

Some of your apps may need to store data on a server. Let's create the TripCard app that stores the trip information locally using an array. If you were building a real-world app, you would not keep the data in that way. The reason is quite obvious: You want the data to be manageable and updatable without re- releasing your app on App Store. The best solution is put your data onto a backend server that allows your app to communicate with it in order to get or update the data. Here you have several options:

- You can come up with your own home-brewed backend server, plus server-side APIs for data transfer, user authentication, etc.
- You can use CloudKit (which was introduced in iOS 8) to store the data in iCloud.
  You can make use of a third-party Backend as a Service provider (BaaS) to manage your data.

The downside of the first option is that you have to develop the backend service on your own. This requires a different skill set and a huge amount of work. As an iOS developer, you may want to focus on app development rather than server side development. This is one of the reasons why Apple introduced CloudKit, which makes developers' lives easier by eliminating the need to develop their own server solutions. With minimal setup and coding, CloudKit empowers your app to store data (including structured data and assets) in its new public database, where the shared data would be accessible by all users of the app. CloudKit works pretty well and is very easy to integrate. However, CloudKit is only available for iOS. If you are going to port your app to Android that utilizes the shared data, CloudKit is not a viable option.

Parse is one of the BaaS that works across nearly all platforms including iOS, Android, Windows phone and web application. By providing an easy-to-use SDK, Parse allows iOS developers to easily manage the app data on the Parse cloud. This should save you development costs and time spent creating your own backend service. The service is free (with limits) and quick to set up.

If you haven't heard of Parse, it is better to understand its history.

Parse was acquired by Facebook in late April 2013. Since then, it has grown into one of the most popular mobile backend. Unfortunately, Facebook considered to shutdown the service and no longer provides the Parse cloud to developers. For now, you can still use Parse as your mobile backend. It comes down to these two solutions:

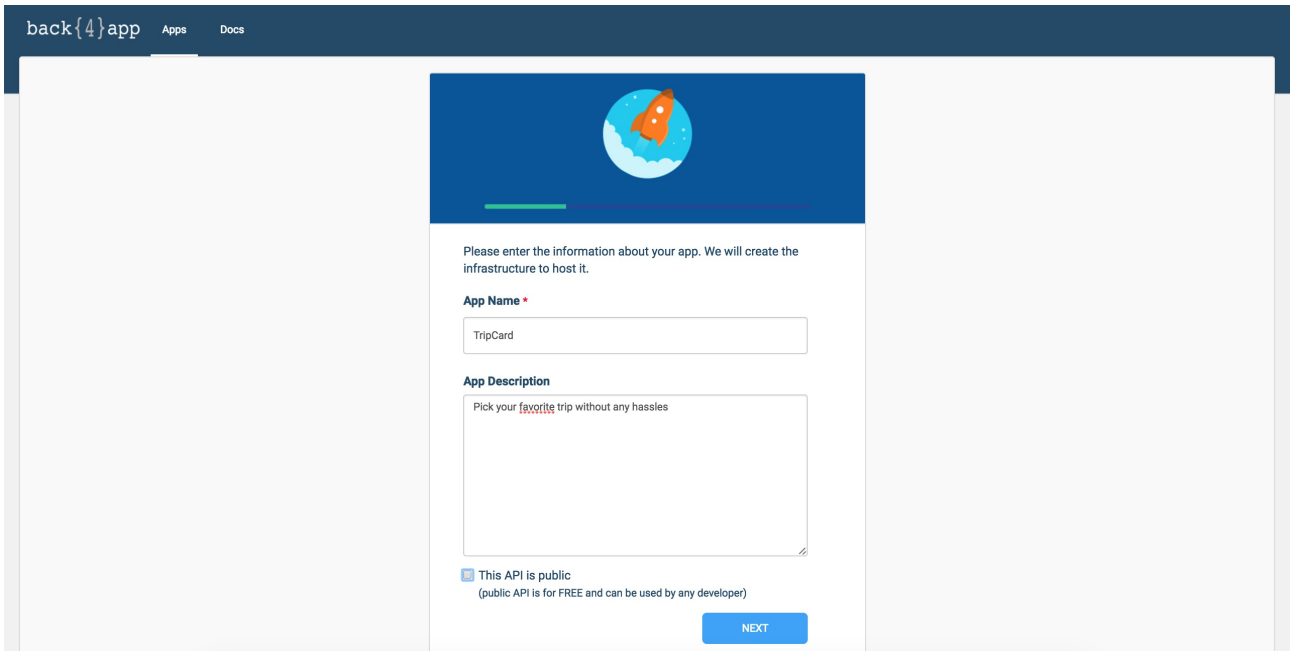
1. Install and host your own Parse servers -Although the Parse's hosted service has been retired on January 28, 2017, Facebook released an open source version of the Parse backend called Parse Server. Now everyone can install and host their own Parse servers on AWS and Heroku. The downside of this approach is that you will have to manage the servers yourself. For indie developers or those who do not have any backend management experience, this is not a perfect option.

2. Use Parse hosting service -Some companies such as SashiDo.io, Oursky and Back4App now offers managed Parse servers. In other words, they help you install the Parse servers, and host them for you. You do not need to learn AWS/Heroku or worry about the server infrastructure. These companies just manage the Parse cloud servers for you. It is very similar to the Parse hosted backend provided by Facebook but delivered by third-party companies. In this tutorial, We will use Back4App's Parse hosting service, simply because it is free to use.

In this practice, we will walk through the integration process of Parse using Back4app. We will create the TripCard app as a demo and see how to put its trip data onto the Parse cloud. To begin with, we have to create a new project.

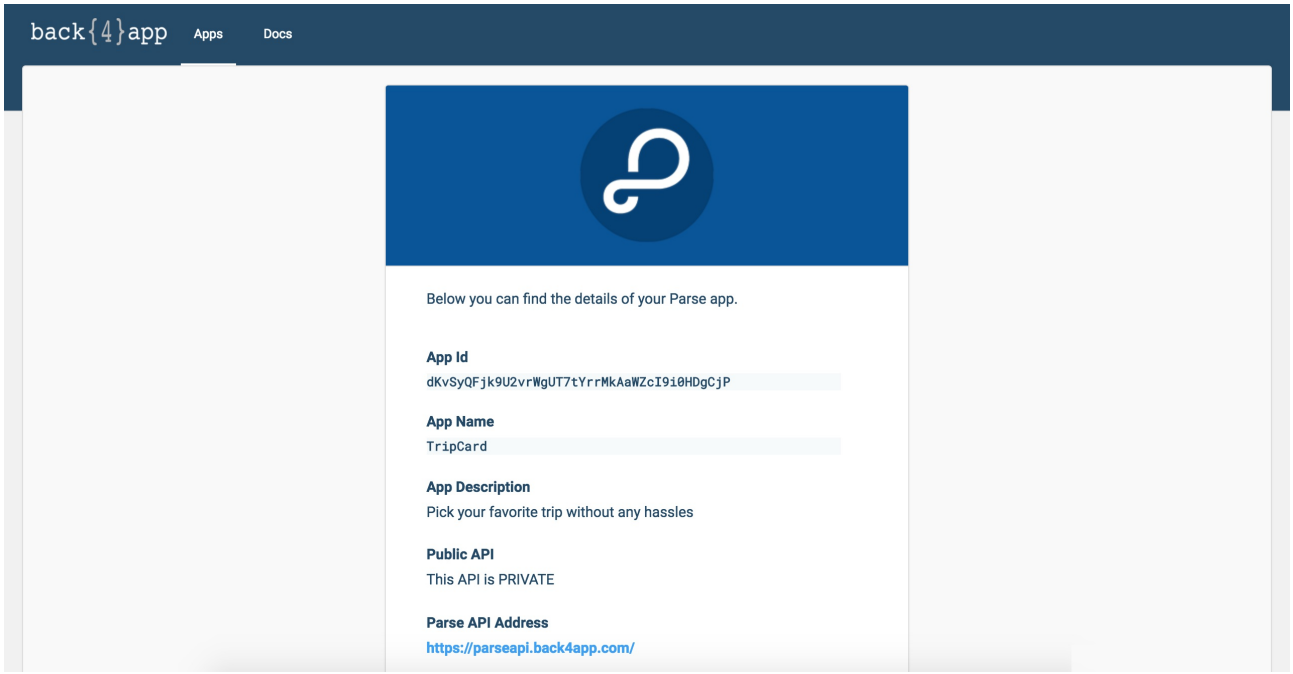
Creating Your App on Parse

First, you have to sign up for a free account on http://back4app.com. Once you sign up the account, you'll be brought to a dashboard. From there, click the Build new Parse app button to create a new application. Simply use TripCard as the app name, and leave the *This API is public* option uncheck.

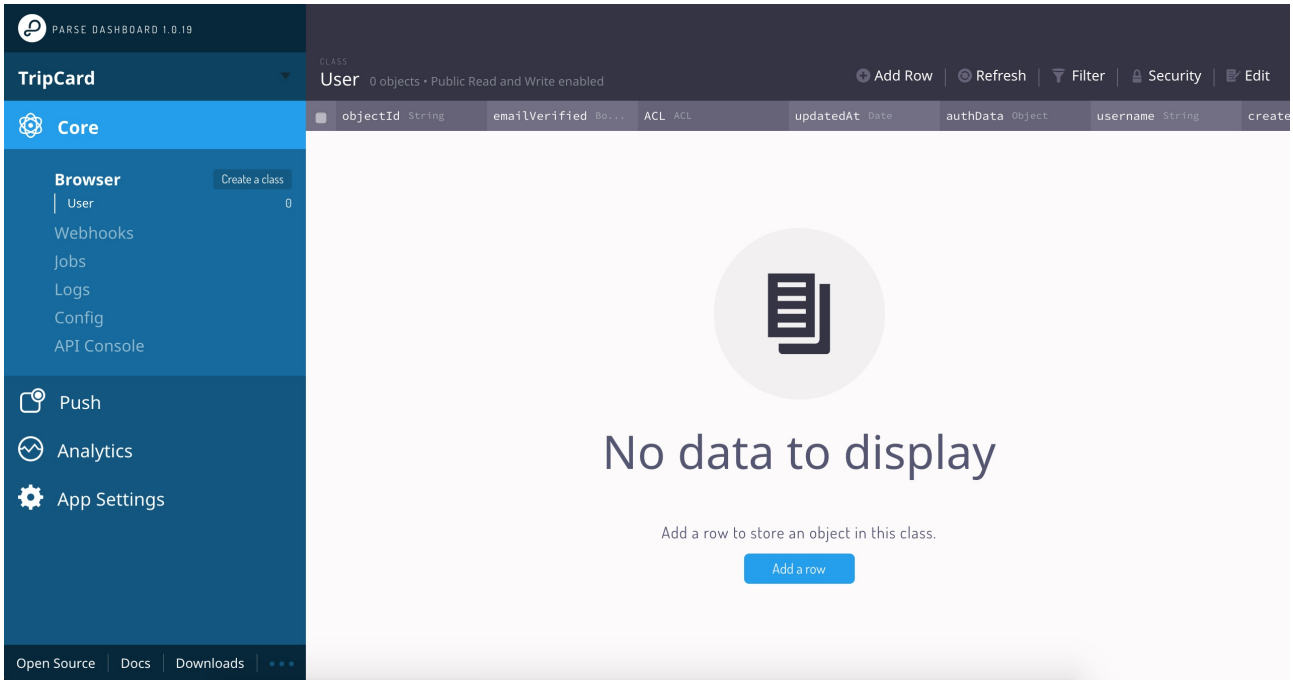


In the next screen, you will be asked to install Neumob. It is completely optional, so hit Finish to skip the step.

Once registered, you'll have the following details of your Parse app including App Id and client key. Scroll down and hit Close to access the main menu.



Like the Parse cloud, Back4app offers various backend services including data and push notification. What we will focus on in this practice is the data service. To manage the data of your Parse app, choose the Parse Dashboard option to access the Parse dashboard.

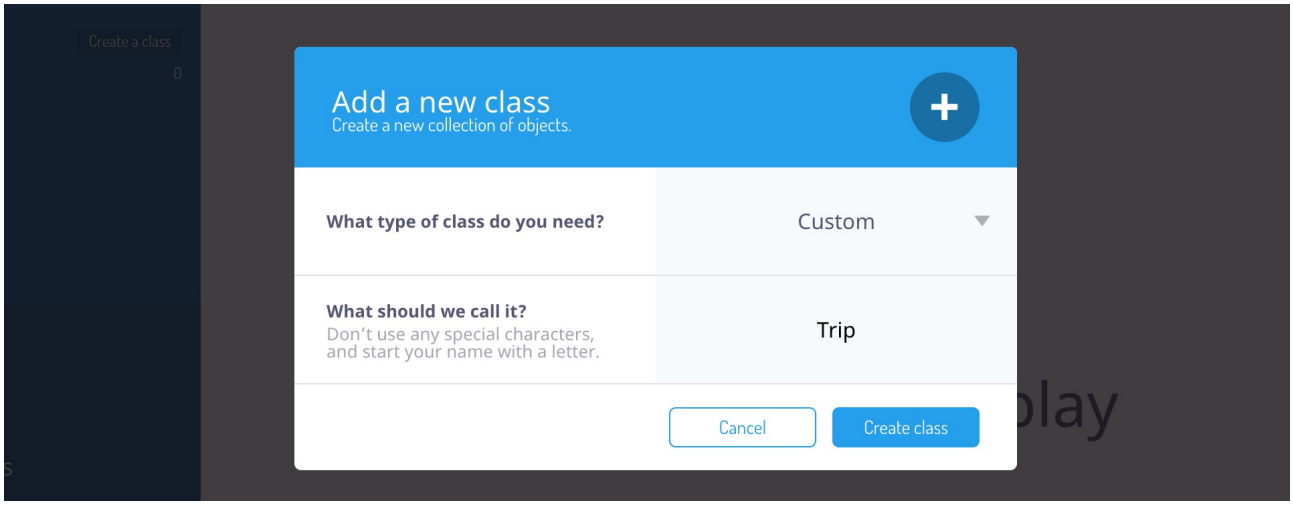


Setting up Your Data

The Parse dashboard lets developers manage their Parse app and data in a graphical UI. By default, the data browser shows no data. It is quite obvious because you do not have any data in the TripCard app.

You will need to create and upload the trip data manually. But before that, you will have to define a Trip class in the data browser. The Trip class defined in Parse is the cloud version of the counterpart class that we have declared in our code. Each property of the class (e.g. city) will be mapped to a table column of the Trip class defined in Parse.

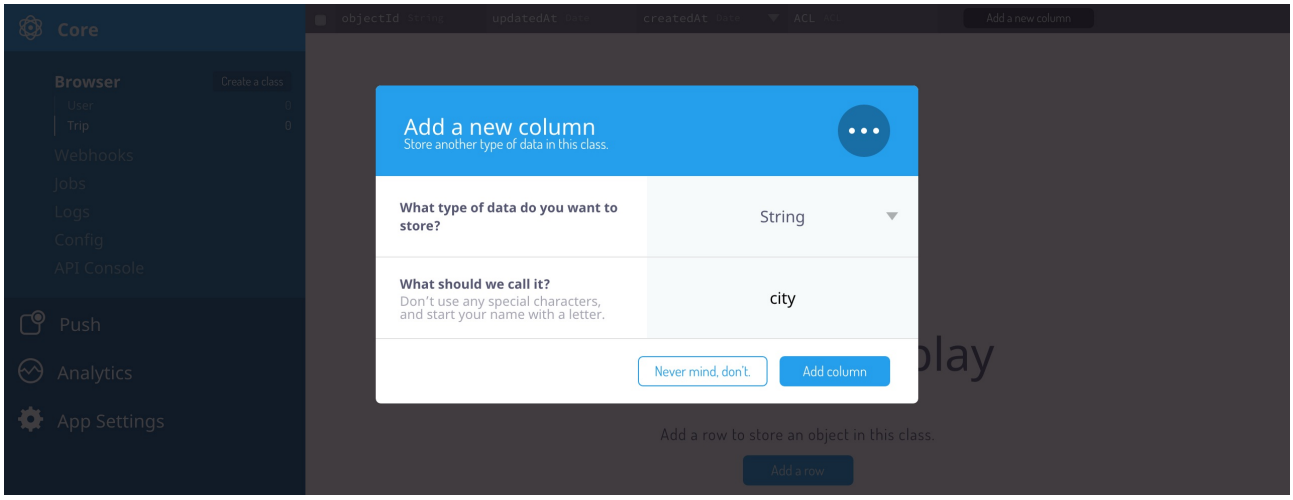
Now click the Create a class button to create a new class. Set the name to Trip and type to Custom , and then click Create class to proceed. Once created, you should see the class under the Browser section of the sidebar menu.



In the TripCard app, a trip consists of the following properties:

- Trip ID
- City
- Country
- Featured image
- Price
- Total number of days
- isLiked

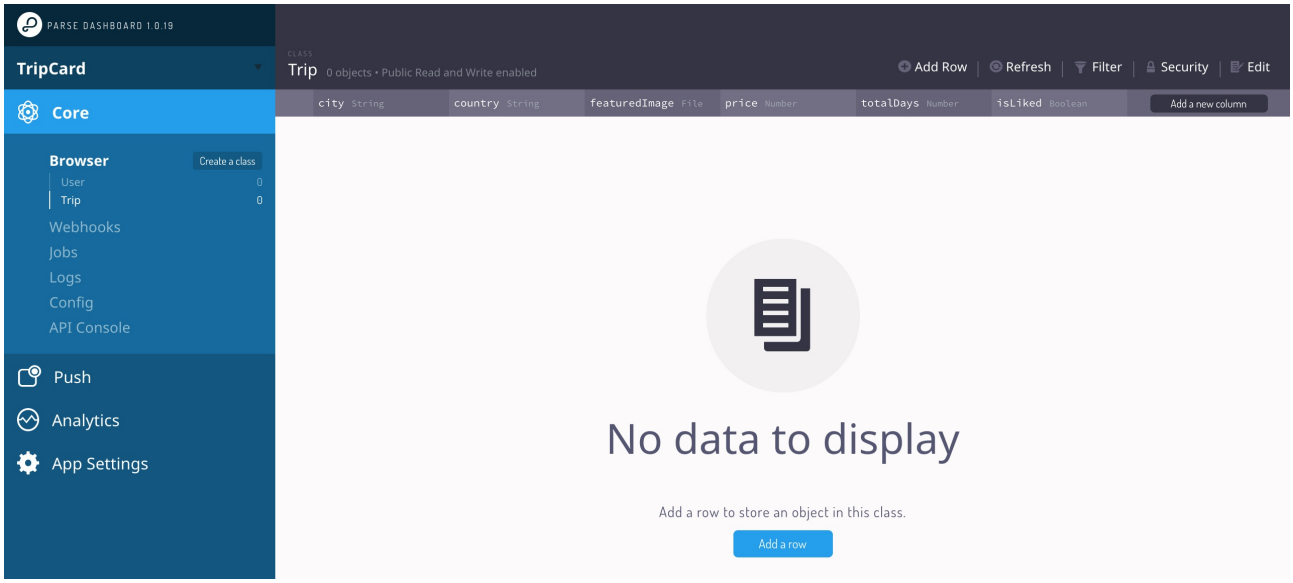
With the exception of the trip ID, each of the properties should be mapped to a corresponding column of the Trip class in the data browser. Select the Trip class and click the Add a new column button to add a new column.



When prompted, set the column name to city and type to String . Repeat the above procedures to add the rest of properties with the following column names and types:

- Country: Set the column name to country and type to String.
- Featured image: Set the column name to featuredImage and type to File . The File type is used for storing binary data such as image.
- Price: Set the column name to price and type to Number.
- Total number of days: Set the column name to totalDays and type to Number .
- isLiked: Set the column name to isLiked and type to Boolean .

Once you have added the columns, your table should look similar to the screenshot below.



You may wonder why we do not create a column for the trip ID. As you can see from the table, there is a default column named objectId . For each new row (or object), Parse automatically generates a unique ID. We will simply use this ID as the trip ID. You may also be wondering how we can convert the data stored in the Parse cloud to objects in our code? The Parse SDK is smart enough to handle the translation of native types. For instance, if you retrieve a String type from Parse, it will be translated into a String object in the app.

Now let's add some trip data into the data browser.

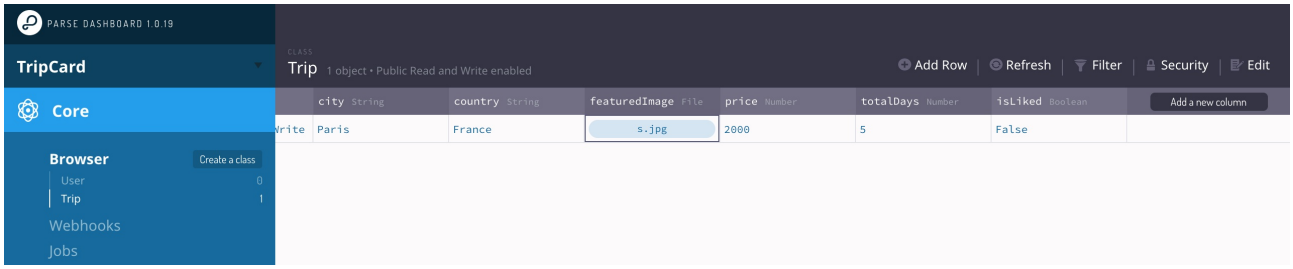
Click the Add Row button to create a new row. Each row represents a single Trip object. You only need to upload the image of a trip and fill in the city, country, price, totalDays and isLiked columns. For the objectId, createdAt and updatedAt columns, the values will be generated by Parse.

If you look into TripViewController.swift , the trips array is defined as follows:

```
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)
]
```

To put the first item of the array into Parse, fill in the values of the row like this:

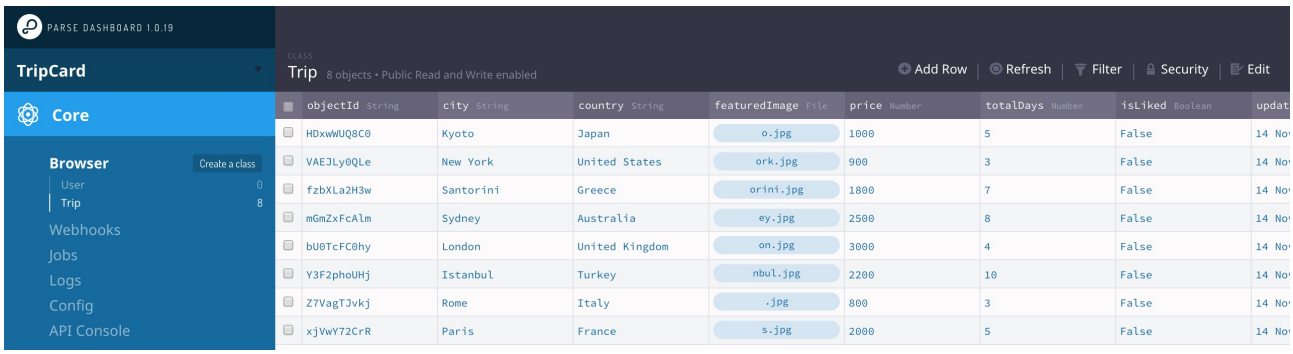


This is very straightforward. We just map the property of the Trip class to the column values of its Parse counterpart. Just note that Parse stores the actual image of the trip in the

featuredImage column. You should have to upload the paris.jpg file by clicking the Upload file button.

Note: You can find the images in the TripCard/Assets.xcassets folder of the ParseDemo project.

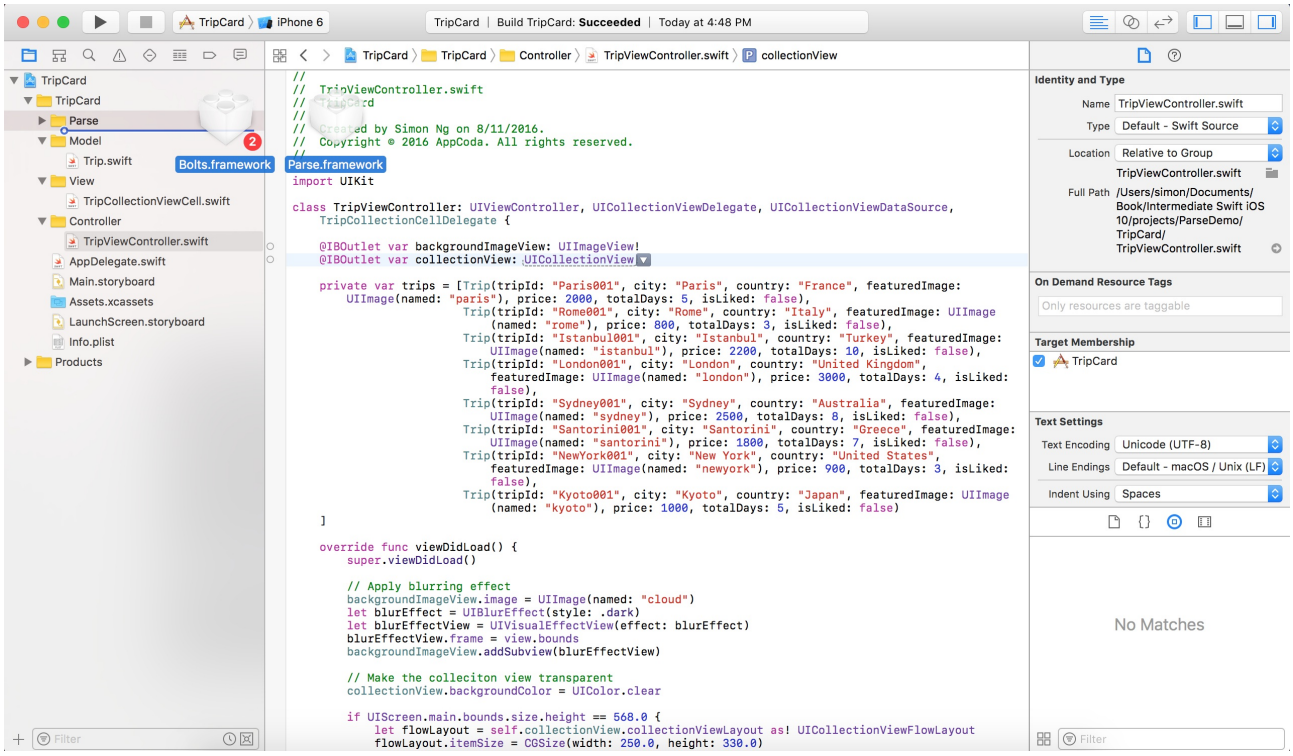
Repeat the above procedures and add the rest of the trip data. You will end up with a screen similar to this:



Configuring the Xcode Project for Parse

Now that you have configured the trip data on the Parse cloud, we will start to integrate the TripCard project with Parse.

To begin with, download the Parse SDK for iOS from https://www.parse.com/downloads/ios/parse-library/latest. Unzip the file and drag both Bolts.framework and Parse.framework into the TripCard project. Optionally, you can create a new group called Parse to better organize the files. When prompted, make sure you enable the Copy items if needed option and click Finish to proceed.

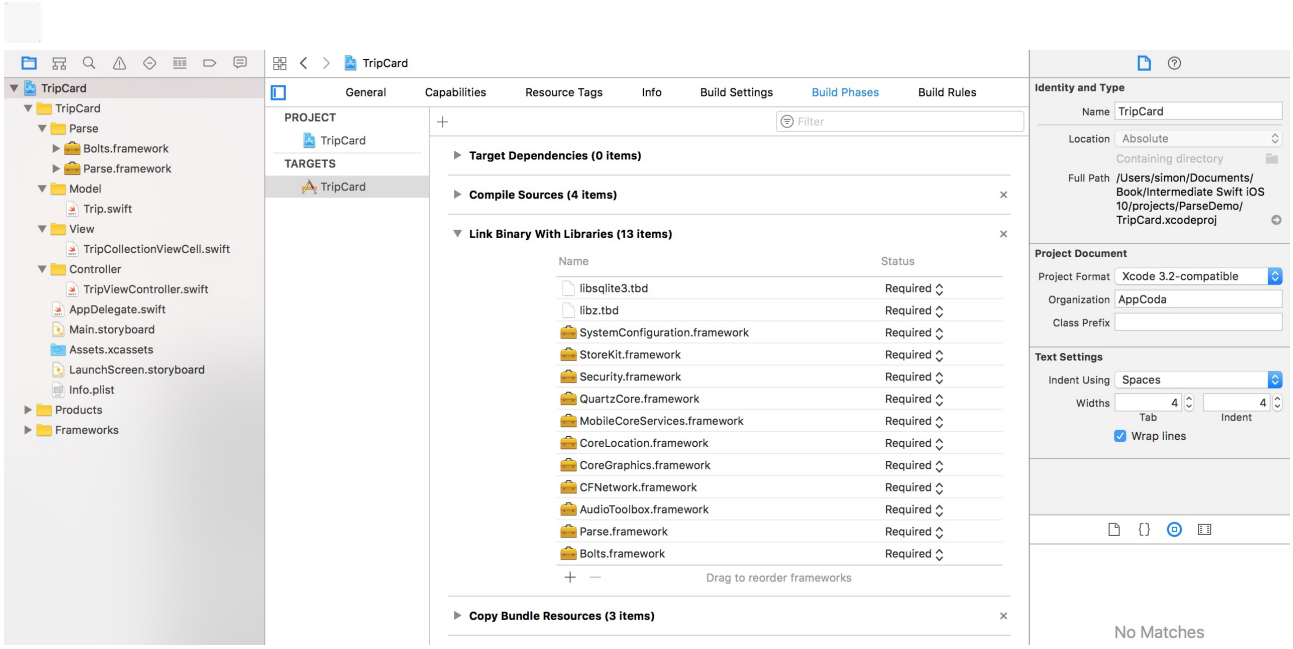


The Parse SDK depends on other frameworks in iOS SDK. You will need to add the following libraries into the project:

- AudioToolbox.framework
- CFNetwork.framework
- CoreGraphics.framework

- CoreLocation.framework
- MobileCoreServices.framework
- QuartzCore.framework
- Security.framework
- StoreKit.framework
- SystemConfiguration.framework
- libz.tbd
- libsqlite3.tbd

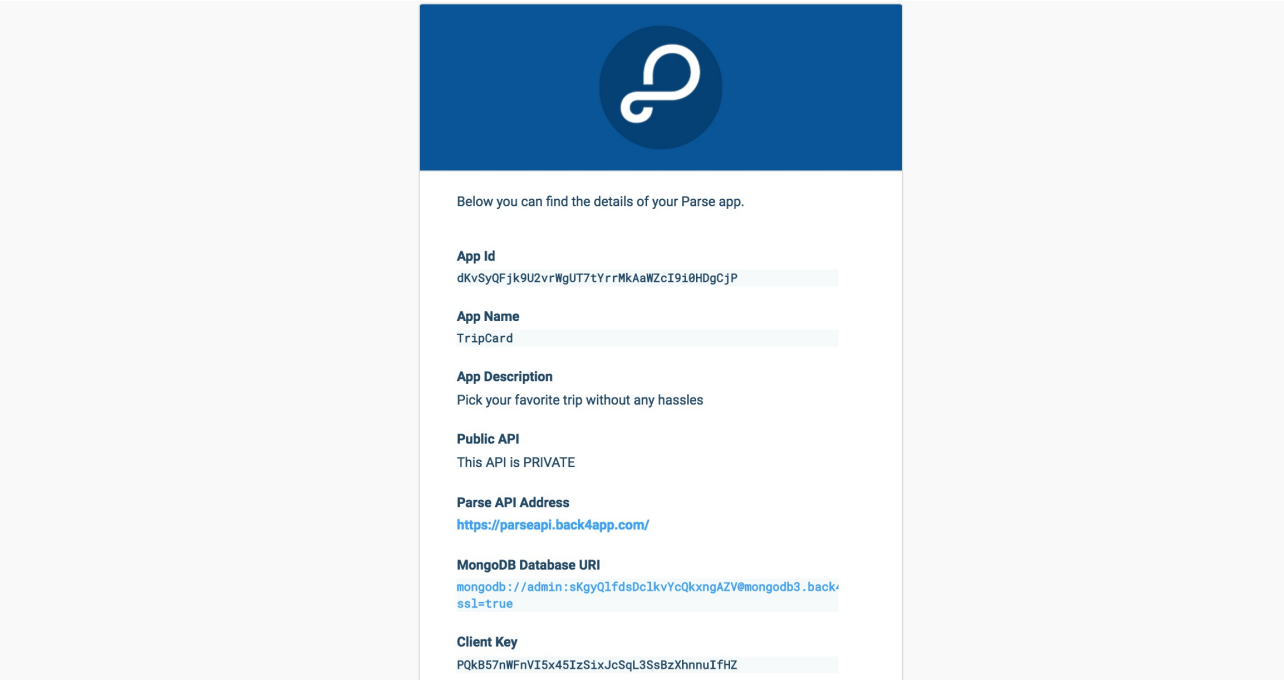
Select the TripCard project in the project navigator. Under the TripCard target, select Build Phases and expand the Link Binary with Libraries. Click the + button and add the above libraries one by one.



Connecting with Parse

To access your app data on Parse, you first need to find out the Application Key and the Client

Key. Go back to back4app.com APP Admin Panel and select Core Settings .



Here you can reveal the application ID and client key. Remember to keep these keys safe, as one can access your Parse data with them.

Open up AppDelegate.swift and add an import statement at the very beginning to import the Parse framework:

```
import Parse
```

Next, add the following code in the application(_:didFinishLaunchingWithOptions:) method to initialize Parse:

```
// Initialize Parse.
let configuration = ParseClientConfiguration {
    $0.applicationId = "dKvSyQFjk9U2vrWgUT7tYrrMkAaWZcI9i0HDgCjP"
    $0.clientKey = "PQkB57nWFnVI5x45IzSixJcSqL3SsBzXhnnuIfHZ"
    $0.server = "https://parseapi.back4app.com"
}
Parse.initialize(with: configuration)
```

Note that you should replace the Application ID and the Client Key with your own keys. With just a couple lines of code, your app is

ready to connect to Parse. Try to compile and run it. If you get everything correct, you should be able to run the app without any error.

Retrieving Data from Parse

Now we are ready to modify the TripCard app to pull data from the Parse cloud. We'll first replace the trips array with the cloud data. To do that, you will have to retrieve the Trip objects that were just created on Parse.

The Parse SDK provides a class called PFQuery for retrieving a list of objects (PFObjects) from Parse. The general usage is like this:

```
let query = PFQuery(className: "Trip")
query.findObjectsInBackground { (objects, error) in
    if let error = error {
        print("Error: \(error) \(error.localizedDescription)")
        return
}

    if let objects = objects {
        // Do something
} }
```

You create a PFQuery object with a specific class name that matches the one created on Parse. For example, for the TripCard app, the class name is Trip . By calling the

findObjectsInBackground method of the query object, the app will go up to Parse and retrieve the available Trip objects. The method works in an asynchronous manner. When it finishes, the block of code will be called and you can perform additional processing based on the returned results.

With a basic understanding of data retrieval, we will modify the TripCard app to get the data from the Parse cloud.

First, open the TripViewController.swift file and change the declaration of trips array to this: private var trips = [Trip]()

Instead of populating the array with static data, we initialize an empty array. Later we will get the trip data from Parse at runtime and save them into the array.

If you look into the Trip structure (i.e. Trip.swift), you may notice that the featuredImage property is of the type UIImage . As we have defined the featuredImage column as a File type on Parse, we have to change the type of the featuredImage property accordingly. This will allow us to convert a PFObject to a Trip object easily.

The corresponding class of a File type in Parse, that lets you store application files (e.g. images) in the cloud, is PFFile . Now open Trip.swift and update it to the following:

```
import UIKit
import Parse
struct Trip {
    var tripId = ""
    var city = ""
    var country = ""
    var featuredImage: PFFile?
    var price:Int = 0
    var totalDays:Int = 0
    var isLiked = false
    init(tripId: String, city: String, country: String, featuredImage: PFFile!,
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
    }
    init(pfObject: PFObject) {
```

```
        self.tripId = pfObject.objectId!
        self.city = pfObject["city"] as! String
        self.country = pfObject["country"] as! String
        self.price = pfObject["price"] as! Int
        self.totalDays = pfObject["totalDays"] as! Int
        self.featuredImage = pfObject["featuredImage"] as? PFFile
        self.isLiked = pfObject["isLiked"] as! Bool
}

    func toPFObject() -> PFObject {
        let tripObject = PFObject(className: "Trip")
        tripObject.objectId = tripId
        tripObject["city"] = city
        tripObject["country"] = country
        tripObject["featuredImage"] = featuredImage
        tripObject["price"] = price
        tripObject["totalDays”] = totalDays
        tripObject["isLiked"] = isLiked

        return tripObject
    }

}
```

Here we added another initialization method for PFObject and a helper method called toPFObject . In the method, we change the type of featuredImage from UIImage to PFFile .

For the purpose of convenience, we create a new initialization method for PFObject and another method for PFObject conversion.

Next, open the TripViewController.swift file and insert the following import statement:

```
import Parse
```

Then add the following method:

```
func loadTripsFromParse() {
    // Clear up the array
    trips.removeAll(keepingCapacity: true)
    collectionView.reloadData()
    // Pull data from Parse
    let query = PFQuery(className: "Trip")
    query.findObjectsInBackground { (objects, error) -> Void in
        if let error = error {
            print("Error: \(error) \(error.localizedDescription)")
            return
}

        if let objects = objects {
            for (index, object) in objects.enumerated() {
                // Convert PFObject into Trip object
                let trip = Trip(pfObject: object)
                self.trips.append(trip)
                let indexPath = IndexPath(row: index, section: 0)
                self.collectionView.insertItems(at: [indexPath])
            }
        }

    }

}
```

The loadTripsFromParse method is created for retrieving trip information from Parse. At the very beginning, we clear out the trips array so as to have a fresh start. We then pull the trip data from the Parse cloud using PFQuery . If the objects are successfully retrieved from the cloud, we convert each of the PFObjects into Trip objects and append them to the trips array. Lastly, we insert the trip to the collection view by calling the insertItems(at:) method.

For the collectionView(_:cellForItemAt:) method, you will need to change the following line of code from:

```
cell.imageView.image = trips[indexPath.row].featuredImage
```

to:

```
// Load image in background
```

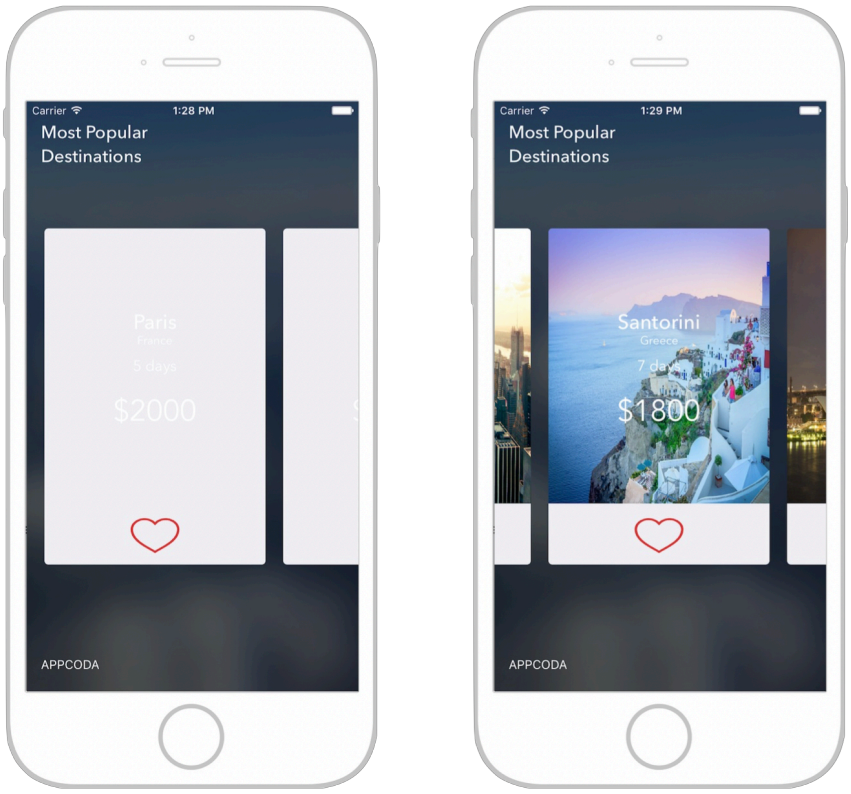
```
cell.imageView.image = UIImage()
if let featuredImage = trips[indexPath.row].featuredImage {
    featuredImage.getDataInBackground(block: { (imageData, error) in
        if let tripImageData = imageData {
            cell.imageView.image = UIImage(data: tripImageData)
        }
    })

}
```

The trip images are no longer bundled in the app. Instead, we will pull them from the Parse cloud. The time required to load the images varies depending on the network speed. This is why we handle the image download in background. Parse stores files (such as images, audio and documents) in the cloud in the form of PFFile . We use PFFile to reference the featured image. The class provides the getDataInBackground method to perform the file download in background. Once the download completes, we load it onto the screen.

Finally insert this line of code in the viewDidLoad method to start the data retrieval:

```
loadTripsFromParse()
```

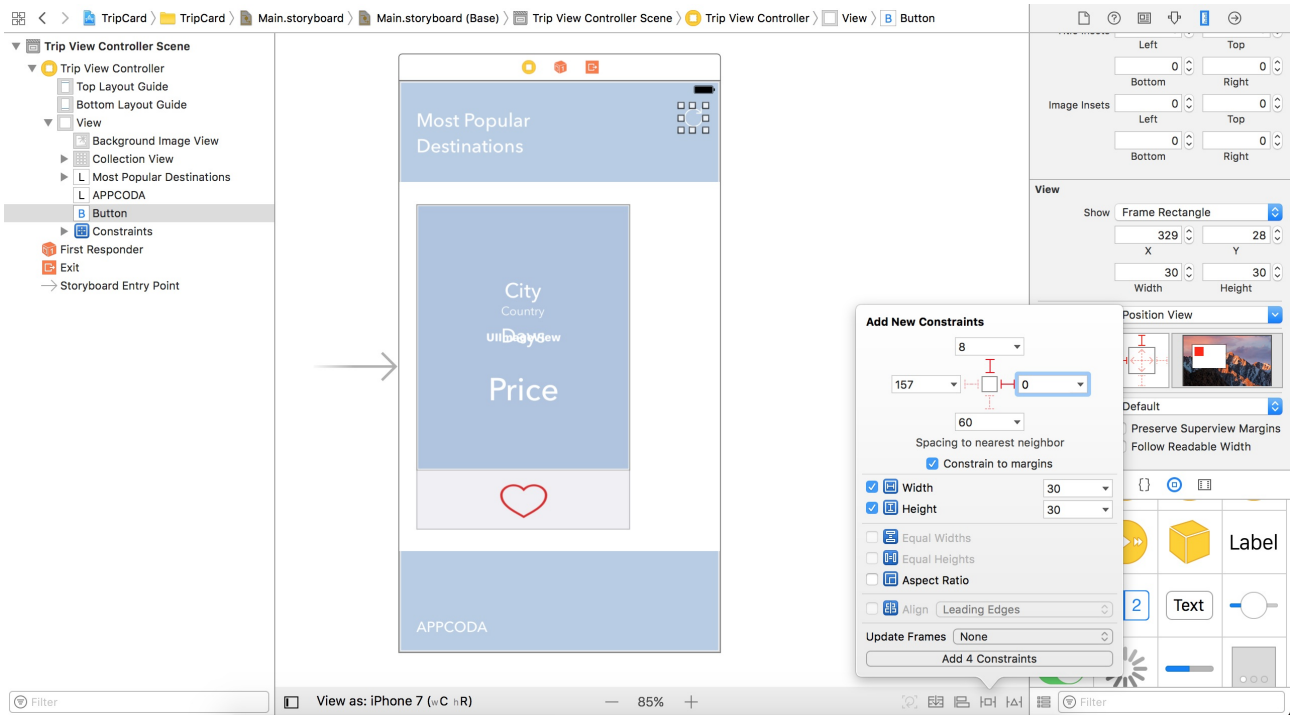
Now hit the Run button to test the app. Make sure your computer/ device is connected to the Internet. The TripCard app should now retrieve the trip information from Parse. Depending on your network speed, it will take a few seconds for the images to load.



Refreshing Data

Currently, there is no way to refresh the data. Let's add a button to the Trip View Controller in storyboard. When a user taps the button, the app will go up to Parse and refresh the trip information.

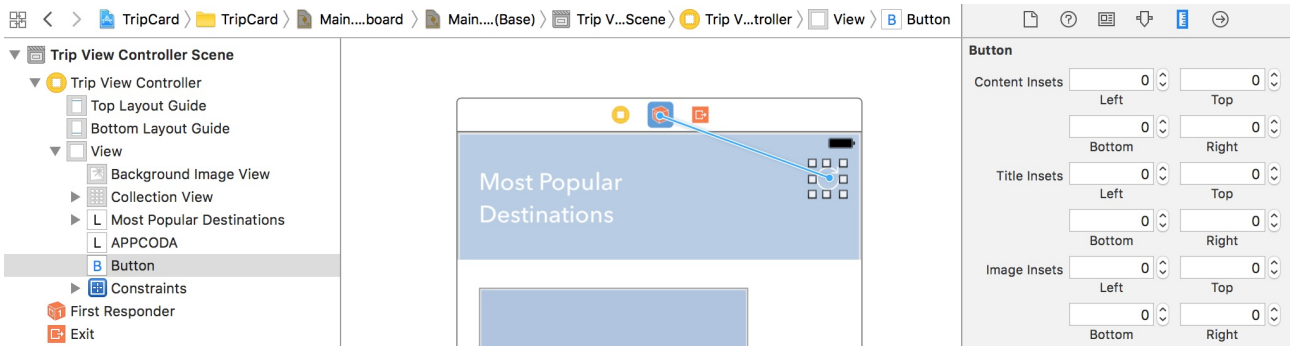
The project template already bundled a reload image for the button. Open Main.storyboard and drag a button object to the view controller. Set its width and height to 30 points. Also, change its image to reload and tint color to white . Finally, click the Add New Constrants button of the auto layout menu to add the layout constraints (see figure).



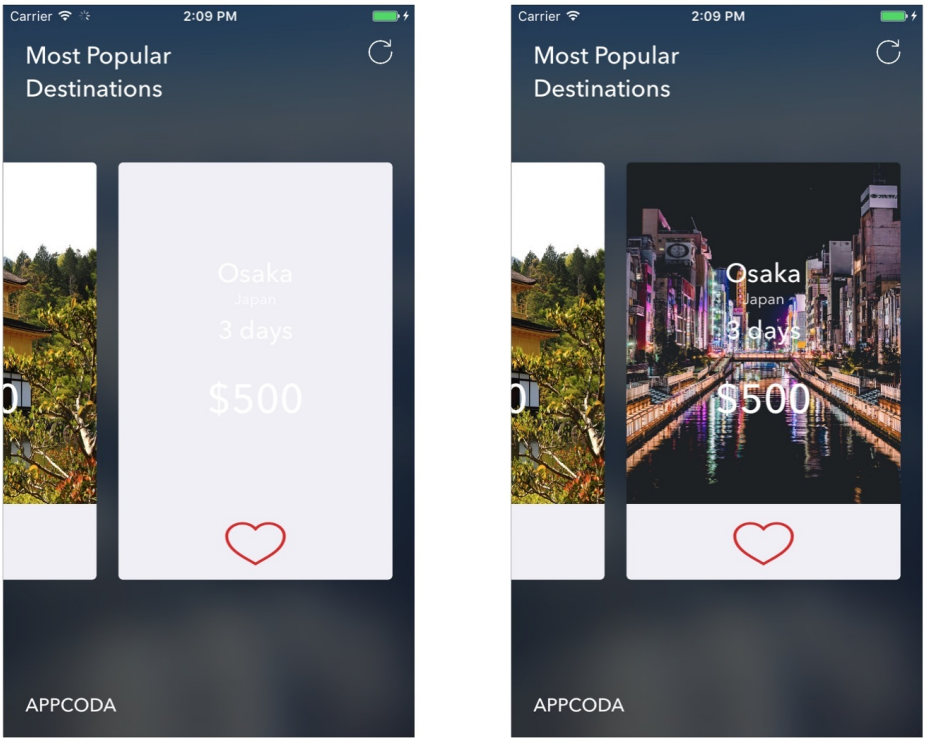
Next, insert an action method in TripViewController.swift :

```
@IBAction func reloadButtonTapped(sender: Any) {
    loadTripsFromParse()
}
```

Go back to the storyboard and associate the refresh button with this action method. Control- drag from the reload button to the first responder button in the dock. After releasing the buttons, select reloadButtonTappedWithSender: .



Now run the app again. Once it's launched, go to the Parse dashboard and add/remove a new trip. Your app should now retrieve the new trip when the refresh button is tapped.



Caching for Speed and Offline Access

Try to close the app and re-launch it. Every time when it is launched, the app starts to download the trips from the Parse backend. What if there is no network access? Let's give it a try. Disable your iPhone or the simulator's network connection and run the app again. The app will not be able to display any trips with the errors in the console that "The Internet connection appears to be offline".

There is a better way to handle this situation. Parse has a built-in support for caching that

makes it a lot easier to save query results on local disk. In case if the Internet access is not available, your app can load the result from local cache.

Caching also improves the app's performance. Instead of loading data from Parse every time when the app runs, it retrieves the data from cache upon startup.

In the default setting, caching is disabled. However, you can easily enable it by writing a single line of code. Add the following code in the loadTripsFromParse method after the initialization of PFQuery :

```
query.cachePolicy = PFCachePolicy.networkElseCache
```

The Parse query supports various types of cache policy. The networkElseCache policy is just one of them. It first loads data from the network, then if that fails, it loads results from the cache.

Now compile and run the app again. After you run it once (with WiFi enabled), disable the WiFi or other network connections and launch the app again. This time, your app should be able to show the trips even if the network is unavailable.

Updating Data on Parse

When you like a trip by tapping the heart button, the result is not saved to the Parse cloud because we haven't written any code for pushing the updates to the cloud.

With the Parse SDK, it is pretty simple to update a PFObject . Recalled that each PFObject comes with a unique object ID, all you need to do is to set some new data to an existing

PFObject and then call the saveInBackground method to upload the changes to the cloud. Based on the object ID, Parse updates the data of the specific object.

Open TripViewController.swift and update the didLikeButtonPressed method, like this:

```
func didLikeButtonPressed(cell: TripCollectionViewCell) {
    if let indexPath = collectionView.indexPath(for: cell) {

        trips[indexPath.row].isLiked = trips[indexPath.row].isLiked ? false : true
        cell.isLiked = trips[indexPath.row].isLiked

        // Update the trip on Parse

       trips[indexPath.row].toPFObject().saveInBackground(block: { (success,
error) -> Void in
            if (success) {
                print("Successfully updated the trip")
            } else {
                print("Error: \(error?.localizedDescription)")
            }

        })

    }

}
```

In the if let block, the first line of code is to set the isLiked property of the corresponding Trip object to true when a user taps the heart button.

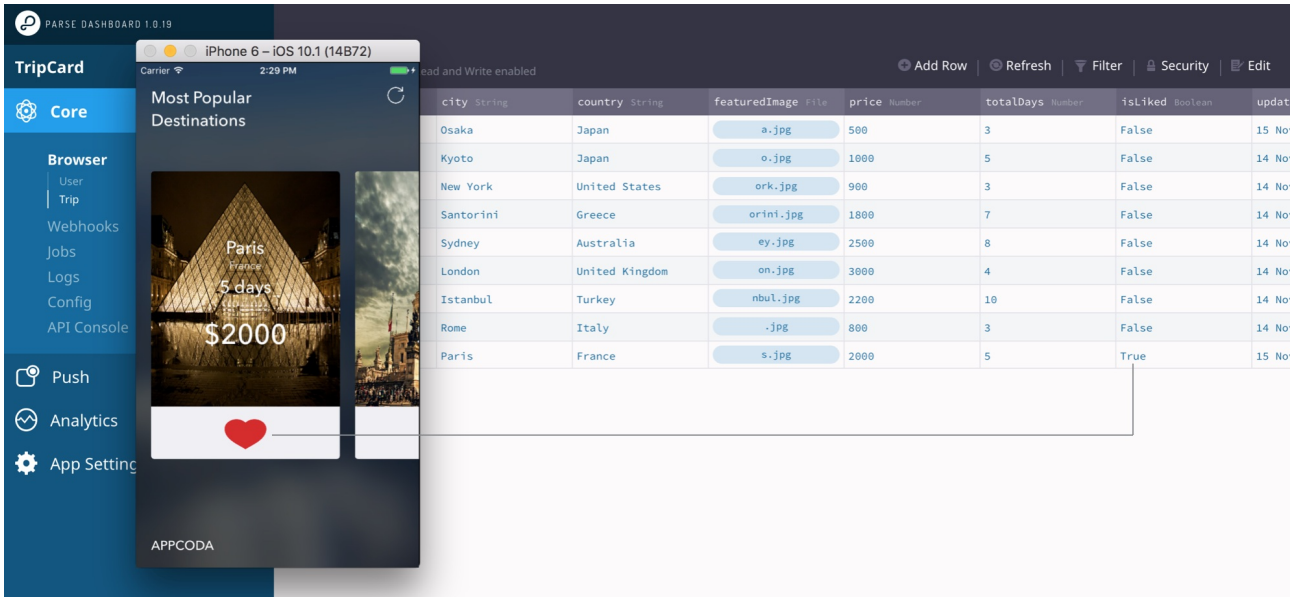
To upload the update to the Parse cloud, we first call the toPFObject method of the selected Trip object to convert itself to a PFObject . If you look into the toPFObject method of the Trip class, you will notice that the trip ID is set as the object ID of the PFObject . This is how

Parse identifies the object to update.

Once we have the PFObject , we simply call the saveInBackground method to upload the changes to Parse.

You can now run the app again. Tap the heart button of a trip and go up to the data browser of Parse. You should find that the isLiked value of the selected trip (say, Santorini) is changed to

true .



Deleting Data from Parse

Similarly, PFObject provides various methods for object deletion. In short, you call up the

deleteInBackground method of the PBObject class to delete the object from Parse.

Currently, the TripCard app does not allow users to remove a trip. We will modify the app to let users swipe up a trip item to delete it. iOS provides the UISwipeGestureRecognizer class to recognize swipe gestures. In the viewDidLoad method of the TripViewController class, insert the following lines of code to initialize a gesture recognizer:

```
// Setup swipe gesture
let swipeUpRecognizer = UISwipeGestureRecognizer(target: self, action:
"handleSwipe:")
swipeUpRecognizer.direction = .Up
swipeUpRecognizer.delegate = self
self.collectionView.addGestureRecognizer(swipeUpRecognizer)
```

When creating the UISwipeGestureRecognizer object, we specify the action method to call when the swipe gesture is recognized. Here we will invoke the handleSwipe method of the current object, which will be implemented later.

Because we only want to look for the swipe-up gesture, we specify the direction property of the recognizer as .up . When using a gesture recognizer, you must associate it with a certain view that the touches happen. In the above code, we invoke the addGestureRecognizer method to associate the collection view with the recognizer.

The delegate of the recognizer should adopt the UIGestureRecognizerDelegate protocol. Thus, add it to the class declaration:

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

Next, implement the handleSwipe method like this:

```
func handleSwipe(gesture: UISwipeGestureRecognizer) {
   let point = gesture.location(in: self.collectionView)
    if (gesture.state == UIGestureRecognizerState.ended) {
       if let indexPath = collectionView.indexPathForItem(at: point) {
          // Remove trip from Parse, array and collection view
          trips[indexPath.row].toPFObject().deleteInBackground(block: {
(success, error) -> Void in
             if (success) {
                print("Successfully removed the trip")
             } else {
                print("Error: \(error?.localizedDescription)")
                return

             }

             self.trips.remove(at: indexPath.row)
             self.collectionView.deleteItems(at: [indexPath])
          })
       }
```

```
    }
}
```

When a user swipes up a trip item (i.e. a collection view cell), we first need to determine which cell is going to be removed. The location(in:) method provides the location of the gesture in the form of CGPoint . From the point returned, we can compute the index path of the collection cell by using the indexPathForItem(at:) method. Once we have the index path of the cell to be removed, we call the deleteInBackground method to delete it from Parse, and remove the item from the collection view.

Great! You've implemented the delete feature. Hit the Run button to launch the app and try to delete a record from Parse.

This practice gave you an idea about how to connect your app to the cloud. In this practice, we use Back4app.com as the Parse backend, which frees you from configuring and managing your own Parse servers. It is not a must to use back4app.com. There are quite a number of Parse hosting service providers you can try it out such as SashiDo.io and Oursky.

The startup cost of using a cloud is nearly zero. And, with the Parse SDK, it is very simple to add a cloud backend for your apps. So begin to consider implementing your existing apps with some cloud features.

***