

Introducing Parse

Copyright © 2014 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written per- mission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connec- tion with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this book are the property of their respective owners.

# Demo

Each step of this tutorial builds upon the completed version of the previous step. The starter project has Parse installed, a Storyboard designed, and most of the needed functions are stubbed out.

First we’re going to show the Storyboard and walk through each screen of the app.

When the app first loads, ViewController will be the first view controller to appear. In its **viewDidAppear** method, it will check to see if a Parse User is currently logged in. If there is a user present, it will perform a segue connected to the **GabsViewController**. If not, it will not transition. On it are two buttons, one to log in, one to sign up. We’re going to add the Sign Up functionality, using the very handy **PFSignUpViewController**, provided by the ParseUI framework.

Open up **ViewController.swift** and update the class definition to declare it as a delegate:

class ViewController: UIViewController, PFSignUpViewControllerDelegate {

Now update the **didTapSignUp** method:

@IBAction func didTapSignUp(sender: AnyObject) {

var signupController = PFSignUpViewController()

signupController.fields = PFSignUpFields.UsernameAndPassword

| PFSignUpFields.SignUpButton

| PFSignUpFields.DismissButton

signupController.delegate = self

presentViewController(signupController, animated: true, completion: nil)

}

Next we need to add a method, which as the delegate for the **PFSignUpViewController** will be called when a sign up succeeds:

// Successful signup:

func signUpViewController(signUpController: PFSignUpViewController!, didSignUpUser user: PFUser!) {

signUpController.dismissViewControllerAnimated(true, completion: nil)

performSegueWithIdentifier("goToMain", sender: self)

}

In this case, we’ll just perform a segue to the **GabsViewController** which is already defined in the storyboard.

We can run the app now, and verify that the sign up controller works. After testing the failure cases, we’ll actually complete a valid signup, and we’ll wind up at the **GabsViewController**. Just a few lines of code, and we can register and save users on Parse.

**GabsViewController** will display data from Parse, but before there’s anything to display, lets add some data. **GabsViewController** has a “New” button at the top right of the navigation bar. It segues to the **NewGabViewController**. This screen has a single text field. Open **NewGabViewController** and update the didTapSend method where the comment is:

var gab = PFObject(className: "GibGabs")

gab["gabText"] = gabText.text

gab["gabVotes"] = 0

gab["gabVoters"] = []

gab.saveInBackgroundWithBlock {

(success: Bool, error: NSError!) -> Void in

activity.stopAnimating()

activity.removeFromSuperview()

self.navigationController?.popViewControllerAnimated(true)

}

Just a few lines of code, and this will store the “Gab” on Parse. Now let’s get back to the **GabsViewController** to display what we’ve saved.

**GabsViewController** is using another ParseUI feature, the **PFQueryTableViewController**. It makes it really easy to manage a table of results from a Parse Query.

Open **GabsViewController.swift** and in the **queryForTable** method, and replace the “invalid” constraint with an ordering constraint and a limit:

query.orderByDescending("createdAt")

query.limit = 50

We’ll get at most 50 results, and the newest records will be first. Update the **tableView:cellForRowAtIndexPath** method to configure the cell with the values from the Gab object:

var object = objectAtIndexPath(indexPath)

cell.gabText.text = object["gabText"] as String!

cell.gabVote.text = object["gabVotes"].stringValue

cell.GabObject = object

We can run the application, create a Gab, and see it displayed in the table along with its vote count. Each **GabCell** has up and down arrows, for voting on a Gab. Open up **GabCell.swift** and update the **didTapVoteUp** method:

upButton.enabled = false

downButton.hidden = true

GabObject?.incrementKey("gabVotes")

GabObject?.addUniqueObject(

PFUser.currentUser().objectId, forKey: "gabVoters")

gabVote.text = GabObject?.objectForKey("gabVotes").stringValue

GabObject?.saveEventually()

This will perform an increment operation on the **gabVotes** column, and add the current users “id” to an array field named **gabVoters**.

We also need to go update **GabsViewController**, to check the **gabVoters** key, to prevent the user from voting again after refreshing the table or reloading the app.

var voters = object["gabVoters"] as Array<String>

for voter in voters {

if (voter == PFUser.currentUser().objectId) {

cell.upButton.hidden = true

cell.downButton.hidden = true

break

}

}

At this point, we have a social app with user sign-up, content submission, and upvoting. In the lab, we’ll add replies, and in the challenge, you’ll add user login and downvoting.

## Lab: Y U No Reply?

We want users to be able to reply to our Gabs. The **GabsViewController** is already configured to segue to the **GabRepliesViewController** when a Gab is selected, and it already passes the Parse Object in the **prepareForSegue** method.

Open **NewGabReplyViewController.swift** and update the **didTapSend** method where commented.

Create a Parse Object with a class name of **GabReplies**

var reply = PFObject(className: "GabReplies")

Set the text of the reply on the object, and assign the Gab itself to the reply

reply["replyText"] = replyText.text

reply["replyUsername"] = PFUser.currentUser().objectForKey("username")

reply["gabParent"] = GabObject

Save the reply object, replacing the 3 lines at the end which stop the activity indicator and pop the view controller

reply.saveInBackgroundWithBlock {

(success: Bool, error: NSError!) -> Void in

activity.stopAnimating()

activity.removeFromSuperview()

self.navigationController?.popViewControllerAnimated(true)

}

Open **GabRepliesViewController.swift** and lets update the **queryForTable** method.

We want to view replies for only the Gab we selected in the previous table view, so we will constrain the query based on a column **gabParent**.

query.whereKey("gabParent", equalTo: GabObject)

When we created a reply, we saved the Gab we’re replying to in this field.

Unlike the Gabs, lets display the replies in chronological order:

query.orderByAscending("createdAt")

The default limit for a Parse Query is 100. We can change this up to a maximum of 1000.

query.limit = 1000

Now, in the **tableView:cellForRowAtIndexPath** method, configure the **GabReplyCell** to display the reply and the username of the person who replied.

var object = objectAtIndexPath(indexPath)

cell.replyText.text = object["replyText"] as String!

cell.replyUsername.text = object["replyUsername"] as String!

Now you should be able to add replies to Gabs and see the replies for any Gab in the system.

## Challenge: Log out, log in, vote down

1. The **GabsViewController** has a Log Out button as its top left bar button item. Update the **didTapLogOut** method to use the **logOut** method of **PFUser** and dismiss the current view controller. Now that we’ve logged out, we need to be able to log back in to our account.
2. Much like the **PFSignUpViewController**, ParseUI offers a **PFLogInViewController**, it works just like the signup controller.
   1. Update the **ViewController** class to specify it will be a delegate for **PFLogInViewController**
   2. Add the delegate method for a **logInViewController:didLogInUser** which does the same that the signup delegate did.

We also want users to be able to down-vote a Gab. Update **GabCell.swift** to use the **didTapVoteDown** method to reduce the vote count of the Gab by one, just like the **didTapViewUp** method incremented it. Hint: You can increment the field by a negative number.

Now the app, while basic, is complete. Users can sign up, log in, share Gabs, vote on Gabs, and reply to Gabs.

This app barely scratches the surface of Parse and what it can do, and over-simplifies for the point of brevity and speed. It is recommended to, during development, secure objects further using Access Control Lists, and moving certain functionality to our powerful Cloud Code functionality. Adding Push Notifications would allow you to notify users when others vote on or reply to their Gabs. GeoLocation could be added to Gabs, and our Geo-spatial queries can retrieve Gabs that are nearby. Image uploads, user profiles, increased analytics… There are many great and time-saving features you can access with Parse.