Tutorial: Build an iOS App in Swift that uses a REST API and Stormpath

by Edward Jiang
March 22, 2016

Mobile (https://stormpath.com/blog/category/mobile), REST API (https://stormpath.com/blog/category/rest-api)

When building mobile apps, authentication is usually the last thing you and your team want to think about. With so many problems to be working on, why spend hours or even days working with your team to implement your authentication system, of all things?

Unfortunately (and embarrassingly, as many of us mobile developers know), teams often ship apps with weak security in the name of speed. In a world where security and privacy matters heavily to the average user, what can we do?

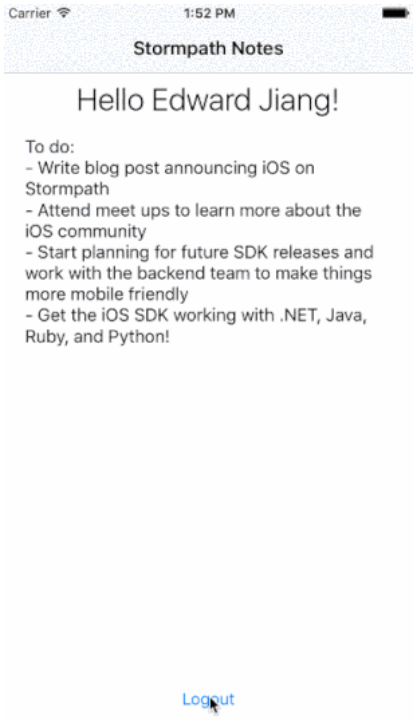
Fortunately, we built Stormpath (https://stormpath.com/) to solve this problem. Stormpath is a complete solution designed to plug into your backend and mobile apps, and securely manage your users and authentication (https://stormpath.com/product/authentication/). Integrating a secure user datastore into your app can now take less than 15 minutes.

Today, we announced support for iOS and Android (https://stormpath.com/blog/stormpath-mobile-support-ios-android/), and this post will walk you setting up a basic Swift app with user login, with some technical discussion along the way.

Overview

To show what you can do with Stormpath, we're building *Stormpath Notes*, a simple note-taking application that allows a person to log in, edit, and save their personal notes on a server. This tutorial is for an iOS app in Swift, but you can also try out our Android tutorial (https://stormpath.com/blog/build-user-authentication-for-android-app/)!

Here's a demo of the app in action:



I've already built and hosted the backend at https://stormpathnotes.herokuapp.com (https://stormpathnotes.herokuapp.com), so we will take the perspective of an iOS developer using the Stormpath SDK to build against this backend. If you want to learn how to build this backend, I've written a tutorial on building the Stormpath Notes Backend with Node.js (https://stormpath.com/blog/tutorial-build-rest-api-mobile-apps-using-node-js).

Stormpath's backend integrations expose a common API for mobile clients to connect to, which we'll use the iOS SDK (https://github.com/stormpath/stormpath-sdk-ios) for. This allows your backend developers to protect additional endpoints with Stormpath authentication.

For Stormpath Notes' backend, I've exposed and protected these endpoints:

`GET /notes` – returns the notes for the authenticated user in the form of a JSON object.

`POST /notes` – takes a JSON object with the notes and saves it for the authenticated user.

The JSON object takes the form of:

```
{"notes": "The notes the user saved"}
```

In case you're curious, we used the following tools to build the backend for Stormpath Notes:

- Express (http://expressjs.com) – A Node.js framework that allows us to write the `/notes` endpoints.
- Express-Stormpath (https://github.com/stormpath/express-stormpath) – Exposes a configurable REST API for our mobile clients within Express.
- Stormpath (https://stormpath.com) – Allows us to store and authenticate users without having to create our own backend for it.
- Heroku (https://www.heroku.com) – Hosts the code for Stormpath Notes online.

Setting up Our iOS Project

To get started, I've built a barebones project with views already created, so we can get to the fun stuff quickly. Start out by downloading from GitHub (https://github.com/stormpath/stormpath-ios-notes-example) or cloning it:

```
git clone https://github.com/stormpath/stormpath-ios-notes-
```

Try running the application and clicking around. I've put together the login, registration, and notes screens already, with the basic ability to navigate between them. If you'd like to, you can also try browsing through the finished version of the example (https://github.com/stormpath/stormpath-ios-notes-example/tree/finished).

Note: This tutorial uses Xcode 8 / Swift 3. An earlier version will not work.

Install Stormpath

Now that we've cloned the app and have it running, the first thing we need to do is add the Stormpath iOS SDK (https://github.com/stormpath/stormpath-sdk-ios) to our iPhone app! We'll use Cocoapods (https://cocoapods.org), a popular dependency manager for iOS projects, to install Stormpath.

To get started with Cocoapods, open up the Terminal and install it:

```
$ sudo gem install cocoapods
```

Then, navigate in the command line to the iOS project. We're going to initialize this project in Cocoapods:

```
$ pod init
```

This creates a `Podfile` in your current directory, which stores information about libraries that your project depends on. Open the `Podfile` in your iOS project's directory, and replace it with the following:

```
# Uncomment the next line to define a global platform
# platform :ios, '9.0'

target 'Stormpath Notes' do
  # Comment the next line if you're not using Swift an
  use_frameworks!

  # Pods for Stormpath Notes
  pod 'Stormpath', '~> 2.0'

end
```

From the `Podfile` Cocoapods created, we added one line:

`pod 'Stormpath', '~> 2.0'` – installs Stormpath 2.0.x. We don't need to specify the version, but we're doing this so that any future updates we make are intentional.

After saving the Podfile, go back to the terminal and run `pod install`. This will now grab Stormpath from GitHub, and install it into your project. Quit out of your Xcode project, and you'll see that Cocoapods generated a new file, `Stormpath Notes.xcworkspace`. You'll open this up instead of `Stormpath Notes.xcodeproj`, as the workspace contains all of your dependencies in addition to your code.

Now that we have Stormpath installed, open `AppDelegate.swift` and add the following lines to the file:

Under the `import UIKit` statement:

```
import Stormpath
```

In `application(application:didFinishLaunchingWithOptions:)`

```
Stormpath.sharedSession.configuration.APIURL = URL(str
```

This will configure your application to use our API server for Stormpath.

Note: Xcode may not recognize the `import Stormpath` command after installing it via Cocoapods. You can get rid of the error by pressing ⌘-B to rebuild your project.

Add User Registration

Now that we have the Stormpath SDK embedded in your Xcode project, let's add it to your `RegisterViewController` so that users can sign up for an account!

As earlier, we need to add `import Stormpath` to the top of the file so you can use the Stormpath SDK.

Looking through the file, you'll notice a function called `register:`, which will be run whenever someone presses the register button in the app. Let's start writing some code so that we can register a user.

In `register(_:)`, we need to create a `RegistrationModel`, which represents the registration form for the user. Depending on the configuration in your backend, you may require different fields for registration. In this case, we are using the default of `email`, `password`, `givenName`, and `surname`:

```
let newUser = RegistrationModel(email: emailTextField.
newUser.givenName = firstNameTextField.text!
newUser.surname = lastNameTextField.text!
```

After we create the `RegistrationModel`, we can send it to the API server by calling Stormpath's register method. Stormpath exposes itself as a singleton named `sharedSession`, so you can easily access it in any view controller without problems. When the registration completes, it calls back on the main thread (so you can do UI work) and sends an `account` and `error` parameter, both which are optionals.

In this case, we want to show an error alert if there's an error, otherwise close the registration screen. Add below your previous code:

```
// Register the new user
Stormpath.sharedSession.register(newUser) { (account,
    if let error = error {
        self.showAlert(withTitle: "Error", message: er
    } else {
        self.exit()
    }
}
```

Note: `showAlert(withTitle:message:)` is a helper extension to `UIViewController` I've added to make displaying an `UIAlert` easier

Try running your app and testing the registration page. You should be able to register a new user!

Add Login Functionality

Now that people can register for your app easily, let's configure your login screen in `LoginViewController`!

As always, don't forget to `import Stormpath` at the top of the file.

In this file, we have four stubs: `login(_:)`, `loginWithFacebook(_:)`, `loginWithGoogle(_:)`, and `resetPassword(_:)`. We're going to add login and reset password functionality for now, and implement Login with Facebook/Google at the end of the tutorial.

To add login functionality, in `login(_:)`, replace the existing view code with:

```
Stormpath.sharedSession.login(emailTextField.text!, pa
```

This will call Stormpath with the email and password provided, and attempt to log the user in. When done, Stormpath calls `openNotes(_:error:)`.

Speaking of which, in this case the callback is a `StormpathSuccessCallback`, which means it's looking for a method of type `(Bool, NSError?) -> Void`. The `Bool` represents if the login was successful, while `NSError?` will be set if there's an error.

Let's modify `openNotes` so it alerts the user on an error, and otherwise opens the login page:

```
func openNotes(success: Bool, error: NSError?) {
    if let error = error {
        showAlert(withTitle: "Error", message: error.l
    }
    else {
        performSegueWithIdentifier("login", sender: se
    }
}
```

Try this out; you can now log in with the account you just registered!

Note: the iOS Simulator currently has a bug which may prevent you from logging in on your phone. See this Stackoverflow thread for more details (http://stackoverflow.com/questions/20344255/secitemadd-and-secitemcopymatching-returns-error-code-34018-errsecmissingentit/22305193). In the meantime, run this example on your physical phone.

In addition, we need to add functionality to `resetPassword:` to send you a password reset email. Stormpath's reset password method calls back with a `StormpathSuccessCallback` just like earlier, so we'll have it alert the user with the result:

```
Stormpath.sharedSession.resetPassword(emailTextField.t
    if let error = error {
        self.showAlert(withTitle: "Error", message: er
    } else {
        self.showAlert(withTitle: "Success", message:
    }
}
```

Retrieve Your Notes from the API

Now that we can register and login to Stormpath Notes, let's build the core of the app: the note taking view!

When we first open the notes view, we see two things: your notes, and a "Hello [Name]!" message. Since we register our users with a name property, Stormpath exposes a `/me` endpoint in the backend which allows you to access a user's profile data. Let's put code to retrieve the user's account, and add the user's name to the `helloLabel`!

In `viewWillAppear(_:)`

```
Stormpath.sharedSession.me { (account, error) -> Void
    if let account = account {
        self.helloLabel.text = "Hello \(account.fullNa
    }
}
```

Right below this segment of code, we'll retrieve our notes by sending an authenticated GET request to `/notes`.

Stormpath uses token based authentication (https://stormpath.com/blog/the-ultimate-guide-to-mobile-api-security/), which means that after authenticating, your application receives an access token and refresh token. When making an API request, you send the access token along with the request in the form of an `Authorization: Bearer [AccessToken]` header. The access token expires after some time, and when it expires, you can request a new one with the refresh token.

While this sounds complex, this ultimately makes your backend more secure and scalable (https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication#the-benefits-of-tokens).

To retrieve our notes, add this code in `viewWillAppear(_:)`:

```
let notesEndpoint = URL(string: "https://stormpathnote
var request = URLRequest(url: notesEndpoint)
request.setValue("Bearer \(Stormpath.sharedSession.acc
let task = URLSession.shared.dataTask(with: request, c
    guard let data = data, let json = (try? JSONSerial
        return
    }
    DispatchQueue.main.async(execute: {
        self.notesTextView.text = notes
    })
})
task.resume()
```

In addition to constructing the HTTP request with the `Authorization` header, we convert the JSON returned by the server into text, and put it in the notes view.

Note: we wrapped our view code in `DispatchQueue.main.async(execute:)`, as we can only manipulate UIKit views on the main thread, and NSURLSession calls back on a secondary thread.

Save Your Notes to the API

Let's also make our app save when we stop editing the text field. This API endpoint requires that we send a `POST` request to `/notes`, with JSON of the notes. Since we're sending JSON to the server, we also need to additionally specify that we're sending `Content-Type: application/json`.

In `textViewDidEndEditing(_:)`, add:

```
let postBody = ["notes": notesTextView.text]

let notesEndpoint = URL(string: "https://stormpathnote
var request = URLRequest(url: notesEndpoint)
request.httpMethod = "POST"
request.httpBody = try? JSONSerialization.data(withJSO
request.setValue("application/json", forHTTPHeaderFiel
request.setValue("Bearer \(Stormpath.sharedSession.acc

let task = URLSession.shared.dataTask(with: request)
task.resume()
```

Finally, we'll make sure to have Stormpath log us out when we click the `logout` button.