iOS Apps with REST APIs

Building Web-Driven Apps in Swift

Christina Moulton

This book is for sale at http://leanpub.com/iosappswithrest

This version was published on 2018-06-26

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

2. Our App's Requirements

It's always tempting to jump right into coding but it usually goes a lot smoother if we plan it out in advance. At the least, we need some idea of what we're building. Let's lay that out for the gists app and you can modify it to suit your app.

The first thing to do is to figure out what our app needs to do. There are a few ways to do this task but I prefer to make a list of things that users will want to do with your app then design the screens to make those things easy.

So what do people do with gists? Gists are snippets of text, often bits of code that are easily shared. So people might:

1. Look at a list of public gists to see what's new
2. Search for interesting gists, maybe by programming language
3. Star a gist so they can find it later
4. Look at a list of gists they've starred
5. Look at a list of their own gists to grab code they commonly use but don't want to retype all the time
6. Look at details for a gist in a list (public, their own, or starred)
7. Create a new gist
8. Delete one of their gists

List the tasks or user stories for your app. Compare them to the list for the gists app, focusing on the number of different objects (like stars, users, and gists) and the types of action taken (like viewing a list, viewing an object's details, adding, deleting, etc.).

You might end up with a really long list. Consider each item and whether it's really necessary for the first version of your app. Maybe it can be part of the next release if the first one gets some traction?

Evaluate each task on your list. Decide which ones will form v1.0 of your app. You might even want to design v2.0 now so you're not tempted to put everything in the first version. A good shipped app is far better than a perfect app that's indefinitely delayed.

2.1 Match Tasks to Endpoints

Next look at each of those tasks and figure out how you can use the API to accomplish them or to get the data you'll need to display. We'll check the documentation for the GitHub gists API[1] to find the endpoint for each task. We'll make notes of anything special that we need to do, like authentication or pagination.

2.1.1 List Public Gists

```
GET /gists/public
```

No authentication required. Will be paginated so we'll have to load more results if they want to see more than 20 or so.

2.1.2 Search Gists

Hmm, there isn't an API for searching gists. Is our app still useful without search? I think so, so we don't need to abandon the project.

2.1.3 Star/Unstar a Gist

```
PUT /gists/:id/star
DELETE /gists/:id/star
```

Requires authentication.

2.1.4 List Starred Gists

```
GET /gists/starred
```

Requires authentication.

2.1.5 List my Gists

There are two ways to get a list of a user's gists:

```
GET /users/:username/gists
```

Or, if authenticated we can get the current user's gists:

[1] https://developer.github.com/v3/gists/

```
GET /gists
```

2.1.6 View Gist Details

We'll probably be able to pass the data from the list of gists to the detail view but if we can't then we can get a single gist's details:

```
GET /gists/:id
```

If we want to display whether a gist is starred then we can use:

```
GET /gists/:id/star
```

2.1.7 Create Gist

```
POST /gists
```

Requires authentication to create a gist owned by a user. Otherwise the gist is created anonymously.

The JSON to send to create a gist looks like:

```
{
  "description": "the description for this gist",
  "public": true,
  "files": {
    "file1.txt": {
      "content": "String file content"
    }
  }
}
```

2.1.8 Delete Gists

```
DELETE /gists/:id
```

Requires authentication.

Those are the endpoints for our tasks. Other than not being able to build our search feature, we shouldn't have any trouble building our demo app around this API.

Analyze each action and list the API endpoint or iOS feature that will be needed for it. Make sure that everything is possible using the API that's available. If not, and the API is being built by your team, then request what you need now so there's plenty of time to get it implemented. If you need features that aren't available and you don't have control over the API then you'll have to figure out how to work around those requirements.

2.2 User Interface

Now we have to figure out how we're going to make the app usable by the users. Let's look at each task and figure out how we'd like it to work. I've reordered the tasks below a bit to group together bits that will share parts of the interface.

2.2.1 Authentication Flow

Since there isn't much they can do in the app without being logged in, we'll check at launch if they're authenticated. If not, we'll start the login process immediately.

If your API provides some functionality without authentication then you might want to delay requiring the user to log in. If that's the case you can add authentication checks before making the API calls that require authentication.

2.2.2 List Public Gists

On launch the user sees a list (table view) with the public gists.

2.2.3 List Starred Gists

From the public gists the user can switch to a similar list of my starred gists.

2.2.4 List My Gists

From the public or starred gists the user can switch to a similar list of their own gists.

Do display these three lists of gists, we'll be able to use a single table view with have a selector so the user can pick which set of gists they want to view.

2.2.5 View Gist Details

When they tap on a gist in one of the lists we'll transition to a different view. That view will list details about the gist (description and filenames) and let them view the text of the files. It will also show whether we've starred the gist.

2.2.6 Star/Unstar a Gist

Within a gist's detail view we'll show the starred status. They will be able to tap to star or unstar a gist in that view.

2.2.7 Create Gist

On the list of My Gists we'll have a + button in the upper right corner. Tapping on that button will display a form where they can enter the info for the new gist:

- Description: text
- Whether it's a public or private gist: Boolean
- Filename: text
- File content: text

To keep it simple we'll only allow a single file in gists created in the app in this initial version.

2.2.8 Delete Gists

We'll allow swipe to delete on the list of My Gists.

Go through your tasks and figure out the user interface that people will use to accomplish those tasks.

2.3 API Requirements

We'll have some requirements to interact with the API that aren't obvious when we consider the user's tasks. Reading through the documentation carefully can help us make a list.

2.3.1 Authentication

> You can read public gists and create them for anonymous users without a token; however, to read or write gists on a user's behalf the gist OAuth scope is required. GitHub Gists API docs[2]

We will need to set up authentication, preferably OAuth 2.0, including the `gist` scope. The API will work with a username/password but then we'd have to worry about securing that data within the app. With OAuth 2.0 we never see the username & password, only the token for our app.

We will store the OAuth token securely.

Check your APIs authentication requirements. In the auth chapters we'll cover how to implement OAuth 2.0, token-based authentication, and basic auth with username/password.

[2] https://developer.github.com/v3/gists/#authentication

2.3.2 Handle App Transport Security

In iOS 9, Apple introduced Apple's App Transport Security[3]. ATS requires SSL to be used for transferring data and it's pretty picky about just how it's implemented. Sadly this means that a lot of servers out there don't meet ATS's requirements. GitHub's gist API complies with the ATS requirements so we won't have to add an exception.

If you find that you get SSL errors when calling your API from iOS 9 then you'll probably need to add an exception to ATS.

2.4 Make a Plan

Now that we know what we need to do we can figure out how we're going to do it. We'll build the app up incrementally, feature by feature:

- Set up the app with a table view displaying the public gists
- Add custom HTTP headers
- Load images in table view cells
- Load more gists when they scroll down
- Add pull to refresh
- Add authentication and let them switch to displaying My Gists and Starred Gists
- Create a detail view for the gists
- Add starring & unstarring gists in the detail view
- Add deleting and creating gists
- Handle not having an internet connection

Put your views and tasks in order to implement them. Try to match up roughly with the order for the gists app. If you don't have an API call to start with that doesn't require authentication you might need to jump ahead to the auth chapters before starting on the table view chapter. If your API requires custom headers to be sent with all requests then you'll want to start with the headers chapter then come back to the table view chapter.

Now that we've sorted out the basic requirements for our app we know where to start. First, we'll spend a little time looking at how to make web requests and parse JSON in Swift so we don't get bogged down with those details later.

[3] https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33