*Note – for best reading experience, uncheck **View > Print layout***

> ***Hello Zac,***
>
> *This document outlines the info and guidelines you'll need to complete Square's mobile take home question: an employee directory app.*
>
> *You'll have a week to complete the question – the actual engineering work should take about 2-4 hours, and you can split this across the week however you see fit (all at once, over a few days, etc).*

Overview

Build an employee directory app that shows a list of employees parsed from the provided JSON endpoint.

Take care to properly handle errors returned from the endpoint (or other network errors like timeouts), and ensure you do not waste network bandwidth – load expensive resources such as large photos on-demand only. Photos at a given URL will never change. Once one is loaded, you do not need to reload the photo. If an employee's photo changes, they will be given a new photo URL (eg, you can treat the URL as a GUID).

All screens and views which load from the network should display proper loading, empty, and error states when content is not available. If images fail to load, displaying a placeholder is fine.

The employee list ***does not*** need to be persisted to disk – you can reload it from the network on each app launch. On the contrary, images ***should*** be cached on disk as to not waste device bandwidth.

You should architect the app in such a way that you can provide unit tests for critical pieces of functionality. There is no minimum or maximum amount of test coverage we're looking for – instead try to provide enough coverage such that regressions are caught by automated tests (though you are not expected to set up a CI system). Only worry about unit tests – you can skip view snapshot or integration tests.

Do not worry about building custom controls and UI elements – using system-provided, standard elements is fine.

Finally, you should also utilize modern engineering practices, language paradigms, and language features relevant to your mobile platform.

Details

JSON Endpoints

We have provided an endpoint, which when called, returns a dictionary containing a JSON array containing employee information for a fictitious list of employees. Each item in the array represents an employee.

`https://s3.amazonaws.com/sq-mobile-interview/employees.json`

There are also other endpoints you can call to simulate error states such as malformed employees and an empty employee list:

`https://s3.amazonaws.com/sq-mobile-interview/employees_malformed.json`

If a malformed employee is returned, **only that employee should be excluded from the list of employees** – valid employees should continue to load. If a subsequent request returns the employee correctly, it should be included in the list. Errors can include missing required fields, duplicate employee UUIDs, etc.

`https://s3.amazonaws.com/sq-mobile-interview/employees_empty.json`

In the case that no employees are returned, the application should present an empty state view.

JSON Structure

Key	Type	Required?	Notes
uuid	string	yes	The unique identifier for the employee. Represented as a UUID.
full_name	string	yes	The full name of the employee.
phone_number	string	no	The phone number of the employee, sent as an unformatted string (eg, 5556661234).
email_address	string	yes	The email address of the employee.
biography	string	no	A short, tweet-length (~300 chars) string that the employee provided to describe themselves.

photo_url_small	string	no	The URL of the employee’s small photo. Useful for list view.
photo_url_large	string	no	The URL of the employee’s full-size photo.
team	string	yes	The team they are on, represented as a human readable string.
employee_type	enum FULL_TIME PART_TIME CONTRACTOR	yes	How the employee is classified.

```
{
  “employees” : [
    {
      “uuid” : “some-uuid”,
      “full_name” : “Eric Rogers”,
      “phone_number” : “5556669870”,
      “email_address” : “erogers.demo@squareup.com”,
      “biography” : “A short biography describing the employee.”,

      “photo_url_small” : “https://some.url/path1.jpg”,
      “photo_url_large” : “https://some.url/path2.jpg”,

      “team” : “Seller”,
      “employee_type” : “FULL_TIME”,
    },

    ...

  ]
}
```

Primary Employee List View

Display a list / table view (or other collection view) which shows all employees returned from the JSON endpoint above. Each row / item in the list should contain a summary of the employee, such as their photo, name, and team. There is no defined way to sort employees – you can sort and group by name, team, etc.

Submitting The Final Project

Feel free to either…

1) Zip up the project and email it to your recruiter. This can be either flat files, or a git repo containing development history.

2) Share the Github repo with ***@sq-mobile-interview*** if you worked on Github in a private repo, and send the link to the repository to your recruiter.

Use What You Know

We're looking to see you at your best – as such, please use the language and frameworks you're most comfortable with on your chosen mobile platform. This means that if you're most proficient with Objective-C or Java, use that – don't feel pressured to use Swift or Kotlin! Square ships all four in production, so we're not going to prefer answers in one language over another.

FAQ

Q) How long should I spent on the solution / how long should the solution take?

We estimate the solution should take 3-6 hours. Many candidates take around 4 hours. Feel free to split up this time as best it works for you.

Q) Do I need to worry about separate mobile and tablet experiences?

No – feel free to focus on either mobile or tablet – just note which one you're focusing on.

Q) What quality should the code in the final app be?

Treat the code as if you're merging it to master in a real app. Take as few shortcuts / hacks as possible. If you do need to take shortcuts, please note as such in a code comment. This also applies to architecture and testability: we're looking for code as close to the real world as possible: utilize architecture and testability best practices. As mentioned: there is no minimum or maximum amount of test coverage we're looking for – instead try to provide enough coverage such that regressions are caught by automated tests. Only worry about unit tests – you can skip view snapshot or integration tests.

Q) Do I need to worry about future extensibility?

Nope! Focus on solving the problem at hand. If generalizing / abstracting a problem makes it easier to solve, or helps you test it, go for it! But generally, do not worry about this.

Q) Can I look at Google / Stack Overflow / my current app / etc while working on this?

Definitely! Please utilize as many Google searches and Stack Overflow searches as needed. We want your work on this project to mirror "real" development as close as possible. That being said, please do not bulk copy and paste large sections of code from 3rd party sources, as this would defeat the purpose of the evaluation.

Q) Can I utilize code I've written for my personal / side projects?

Do feel free to copy in (or pull in as a dependency) utility functions, classes, types (etc) you usually bring into your personal projects, as long as you wrote them.

Please **do not** use code from previous/current employers, for IP / legal reasons :-)

Q) What programming languages can I use?

Please utilize Java or Kotlin for Android, or Objective-C or Swift for iOS. For iOS, if you need to drop down to CoreFoundation, Accelerate, etc, C is fine too (but you shouldn't need to).

Do not feel pressured or required to use Swift or Kotlin if you've primarily worked in Objective-C or Java in the past – use whatever you're most comfortable with! For context, Square still has plenty of Objective-C and Java, and will for some time.

Please **do not** utilize cross-platform SDKs such as React Native, Flutter, Phonegap, Cordova, etc.

Q) Can I use 3rd party frameworks?

Sure – feel free to use any "no brainer" frameworks you'd usually pull into an app such as RxJava, RxSwift, etc. However, since this is a relatively simple application, you shouldn't need many external dependencies. In particular, we'd like to see no 3rd party image caching framework used for loading and storing employee's profile images (since this is an interesting part of the question).

...And of course, 1st-party frameworks that come with iOS or Android are fine.

Q) What build systems & build tooling should I use?

Please utilize platform-standard build systems and tooling. If you decide to use tooling such as CocoaPods or Bazel to assist in project configuration and set up, include a README explaining how to build and run the app.

Q) What version of iOS or Android SDKs and tooling should I use?

Please utilize modern tooling and development practices. For this case, consider anything released in the last 1 year as “modern”.

Q) What are you looking for? What is the right answer?

There isn’t one right answer! We’re looking to see how you approach the problem. The generic things we’re evaluating are architecture and structure of your code, how maintainable and testable it is, isolation of concerns, etc.

Q) Does UI design / UI appearance matter?

Make something with a sensible UX flow, but do not worry about building custom controls. Using standard system UI controls are totally fine.

Q) Does on-device performance matter?

Not really, but be reasonable: phones and tablets are pretty fast these days. Don’t prematurely optimize anything or do microbenchmarks. If you find something that is actually slow when using the app (eg, causing dropped frames, freezing), go ahead and optimize it.

That being said, ***do*** consider network usage: we will want to lazy load images in order to avoid wasting the user’s data plan.

Q) For iOS, can I use xibs and storyboards?

Use whatever you are most comfortable with – however do note that most development at Square does ***not*** use xibs or storyboards (preferring in-code layout), so that’s what you should expect once working within our codebases if you are hired.

Q) Something else not covered here?

Post a comment on the google doc (*highlight relevant text > right click > comment*) and we’ll get back to you ASAP.

🏢 Onsite Interview Additions

When you come in for your onsite interview, you will work with two engineers over the course of two pair programming sessions (1 hour each) to add more functionality to your application. In the first pair programming session, we'll focus on extending the data model. In the second pair programming, we'll focus on extending the UI.

During the interview, the interviewer will also ask you about general architecture choices you made while writing the app. Please be prepared to answer these questions as they come up!